OpenShift Container Platform 4.13

Virtualization

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

Last Updated: 2024-12-15

OpenShift Container Platform 4.13 Virtualization

OpenShift 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 で OpenShift 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 OPENSHIFT VIRTUALIZATION について

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

1.1. OPENSHIFT VIRTUALIZATION の機能

OpenShift 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。

OpenShift Virtualization は、Kubernetes カスタムリソースにより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 (VM) の作成と管理
- クラスター内で Pod と仮想マシンのワークロードの同時実行
- 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イ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。

OpenShift Virtualization は、Red Hat OpenShift Data Foundation の機能とうまく連携するように設計およびテストされています。

重要

OpenShift Data Foundation を使用して OpenShift Virtualization をデプロイする場合は、Windows 仮想マシンディスク用の専用ストレージクラスを作成する必要があります。詳細は 、Windows VM の ODF PersistentVolumes の最適化 を参照してください。

OpenShift Virtualization は、OVN-Kubernetes、OpenShift SDN、または 認定 OpenShift CNI プラグイン にリストされているその他の認定ネットワークプラグインのいずれかで使用できます。

Compliance Operator をインストールし、**ocp4-moderate** および **ocp4-moderate-node** プロファイルを使用してスキャンを実行することで、OpenShift Virtualization クラスターのコンプライアンス問題を確認できます。Compliance Operator は、NIST 認定ツール である OpenSCAP を使用して、セキュリティーポリシーをスキャンし、適用します。

1.1.1. OpenShift Virtualization サポートのクラスターバージョン

OpenShift Virtualization 4.13 は OpenShift Container Platform 4.13 クラスターで使用するためにサポートされます。Open Shift Virtualization の最新の z-stream リリースを使用するには、最初に Open Shift Container Platform の最新バージョンにアップグレードする必要があります。

1.2. 仮想マシンディスクのストレージボリュームについて

既知のストレージプロバイダーでストレージ API を使用する場合、ボリュームモードとアクセスモードは自動的に選択されます。ただし、ストレージプロファイルのないストレージクラスを使用する場合は、ボリュームとアクセスモードを選択する必要があります。

accessMode: ReadWriteMany と **volumeMode: Block** を使用すると、最適な結果を得ることができます。これは、以下の理由により重要です。

- ライブマイグレーションには ReadWriteMany (RWX) アクセスモードが必要です。
- **Block** ボリュームモードは、**Filesystem** ボリュームモードと比較してパフォーマンスが大幅に向上します。これは、**Filesystem** ボリュームモードでは、ファイルシステムレイヤーやディスクイメージファイルなどを含め、より多くのストレージレイヤーが使用されるためです。仮想マシンのディスクストレージに、これらのレイヤーは必要ありません。
  たとえば、Red Hat OpenShift Data Foundation を使用する場合は、CephFS ボリュームよりも Ceph RBD ボリュームの方が推奨されます。

重要

以下を使用する仮想マシンのライブマイグレーションを行うことはできません。

- ReadWriteOnce (RWO) アクセスモードのストレージボリューム
- GPU などのパススルー機能

それらの仮想マシンの **evictionStrategy** フィールドを **LiveMigrate** に設定しないでください。

1.3. 単一ノードの OPENSHIFT の違い

OpenShift Virtualization はシングルノード OpenShift にインストールできます。

ただし、シングルノード OpenShift は次の機能をサポートしていないことに注意してください。

- 高可用性
- Pod の中断
- ライブマイグレーション
- エビクションストラテジーが設定されている仮想マシンまたはテンプレート

1.4. 関連情報

- OpenShift Container Platform ストレージの共通用語集
- 単一ノード OpenShift について
- アシステッドインストーラー
- hostPath Provisioner (HPP)
- OpenShift Container Platform Data Foundation Logical Volume Manager Operator
- Pod の Disruption Budget (停止状態の予算)
- ライブマイグレーション

- エビクションストラテジー
- Tuning & Scaling Guide
- Supported limits for OpenShift Virtualization 4.x

第2章 OPENSHIFT VIRTUALIZATION アーキテクチャー

OpenShift Virtualization アーキテクチャーについて学習します。

2.1. OPENSHIFT VIRTUALIZATION アーキテクチャーの仕組み

OpenShift Virtualization をインストールすると、Operator Lifecycle Manager (OLM) は、OpenShift Virtualization の各コンポーネントのオペレーター Pod をデプロイします。

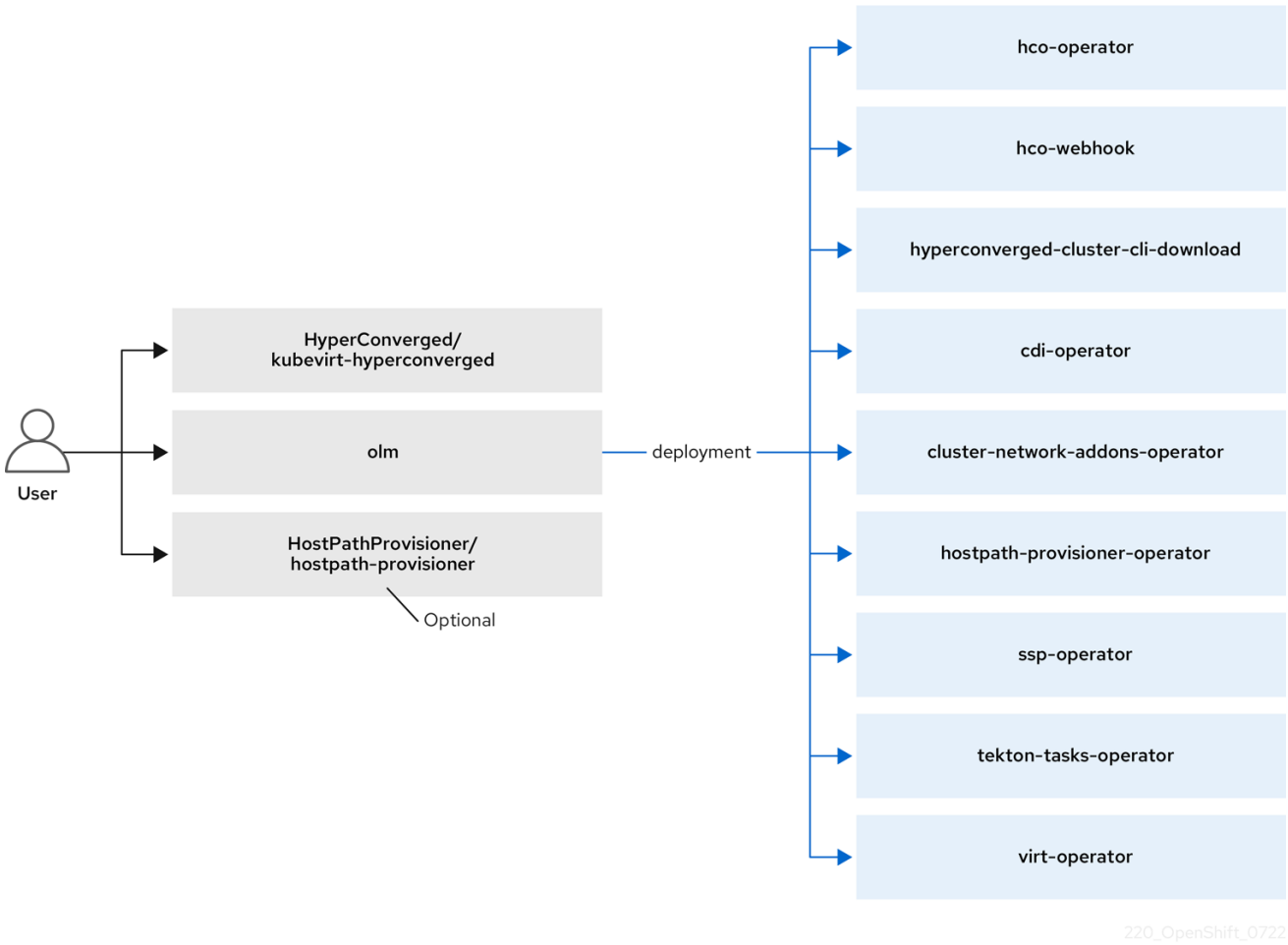
- コンピューティング: **virt-operator**
- ストレージ: **cdi-operator**
- ネットワーク: **cluster-network-addons-operator**
- スケーリング: **ssp-operator**
- テンプレート作成: **tekton-tasks-operator**

OLM は、他のコンポーネントのデプロイ、設定、およびライフサイクルを担当する **hyperconverged-cluster-operator** Pod と、いくつかのヘルパー Pod (**hco-webhook** および **hyperconverged-cluster-cli-download**) もデプロイします。

すべての Operator Pod が正常にデプロイされたら、**HyperConverged** カスタムリソース (CR) を作成する必要があります。**HyperConverged** CR で設定された設定は、信頼できる唯一の情報源および OpenShift Virtualization のエントリーポイントとして機能し、CR の動作をガイドします。

HyperConverged CR は、調整ループ内の他のすべてのコンポーネントの Operator に対応する CR を作成します。その後、各 Operator は、デーモンセット、config map、および OpenShift Virtualization コントロールプレーン用の追加コンポーネントなどのリソースを作成します。たとえば、**hco-operator** が **KubeVirt** CR を作成すると、**virt-operator** はそれを調整し、**virt-controller**、**virt-handler**、**virt-api** などの追加リソースを作成します。

OLM は **hostpath-provisioner-operator** をデプロイしますが、**hostpath provisioner** (HPP) CR を作成するまで機能しません。



関連情報

- HyperConverged CR 設定
- Virtctl クライアントコマンド

2.2. HCO-OPERATOR について

hco- operator (HCO) は、OpenShift Virtualization をデプロイおよび管理するための単一のエントリーポイントと、独自のデフォルトを持ついくつかのヘルパー Operator を提供します。また、これらの Operator のカスタム リソース (CR) も作成します。

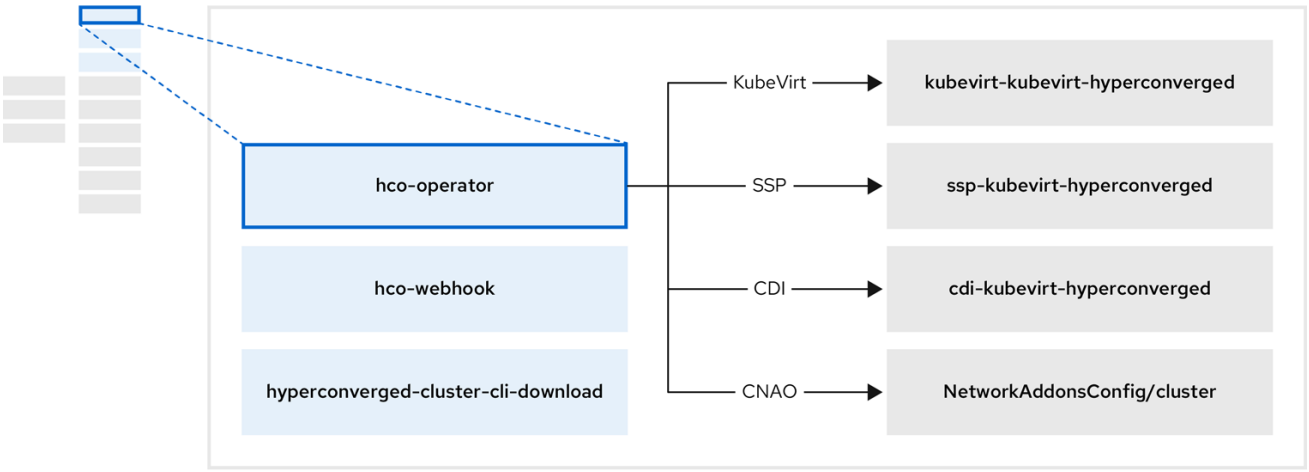


表2.1 hco-operator コンポーネント

コンポーネント	説明
deployment/hco-webhook	**HyperConverged** カスタムリソースの内容を検証します。
deployment/hyperconverged-cluster-cli-download	クラスターから直接ダウンロードできるように、**virtctl** ツールのバイナリーをクラスターに提供します。
KubeVirt/kubevirt-kubevirt-hyperconverged	OpenShift Virtualization に必要なすべての Operator、CR、およびオブジェクトが含まれています。
SSP/ssp-kubevirt-hyperconverged	SSP CR。これは、HCO によって自動的に作成されます。
CDI/cdi-kubevirt-hyperconverged	A CDI CR。これは、HCO によって自動的に作成されます。
NetworkAddonsConfig/cluster	**cluster-network-addons-operator** によって指示および管理される CR。

2.3. CDI-OPERATOR について

cdi-operator は、Containerized Data Importer (CDI) とその関連リソースを管理します。これは、データ ボリュームを使用して仮想マシン (VM) イメージを永続ボリューム要求 (PVC) にインポートします。

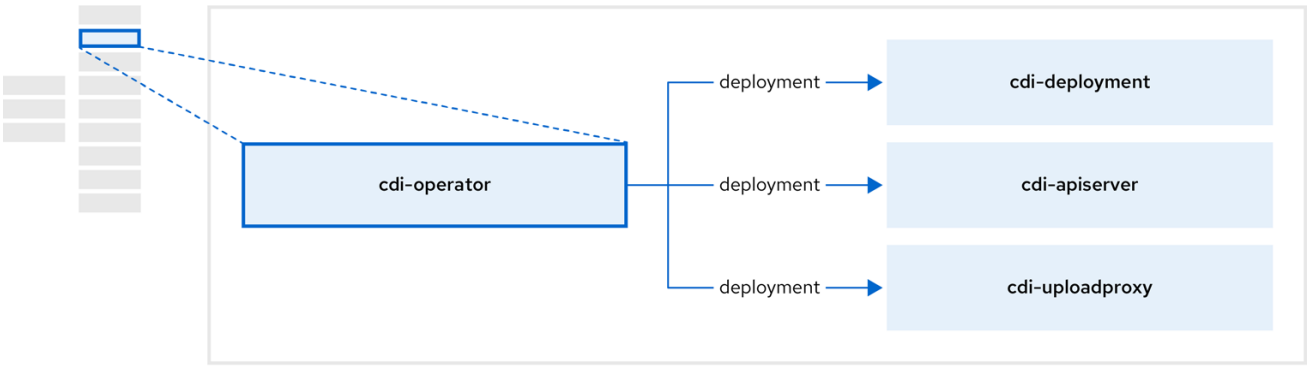


表2.2 cdi-operator コンポーネント

コンポーネント	説明
deployment/cdi-apiserver	安全なアップロードトークンを発行して、VM ディスクを PVC にアップロードするための承認を管理します。
deployment/cdi-uploadproxy	外部ディスクのアップロードトラフィックを適切なアップロードサーバー Pod に転送して、正しい PVC に書き込むことができるようにします。有効なアップロードトークンが必要です。
pod/cdi-importer	データ ボリュームの作成時に仮想マシンイメージを PVC にインポートするヘルパー Pod。

2.4. CLUSTER-NETWORK-ADDONS-OPERATOR について

cluster-network-addons-operator は、ネットワーク コンポーネントをクラスターにデプロイし、ネットワーク機能を拡張するための関連リソースを管理します。

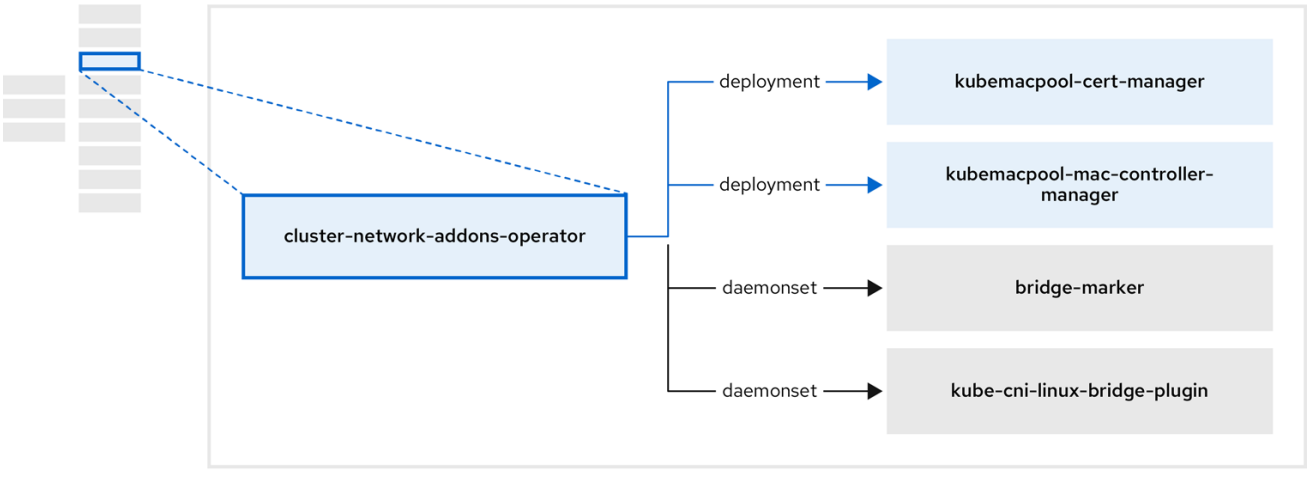


表2.3 cluster-network-addons-operator コンポーネント

コンポーネント	説明
deployment/kubemacpool-cert-manager	Kubemacpool の Webhook の TLS 証明書を管理します。
deployment/kubemacpool-mac-controller-manager	仮想マシン (VM) ネットワークインターフェイスカード (NIC) の MAC アドレスプールサービスを提供します。
daemonset/bridge-marker	ノードで使用可能なネットワークブリッジをノードリソースとしてマークします。
daemonset/kube-cni-linux-bridge-plugin	クラスターノードに CNI プラグインをインストールし、ネットワーク接続定義を介して Linux ブリッジに VM を接続できるようにします。

2.5. HOSTPATH-PROVISIONER-OPERATOR について

hostpath-provisioner-operator は、マルチノードホストパスプロビジョナー (HPP) および関連リソースをデプロイおよび管理します。

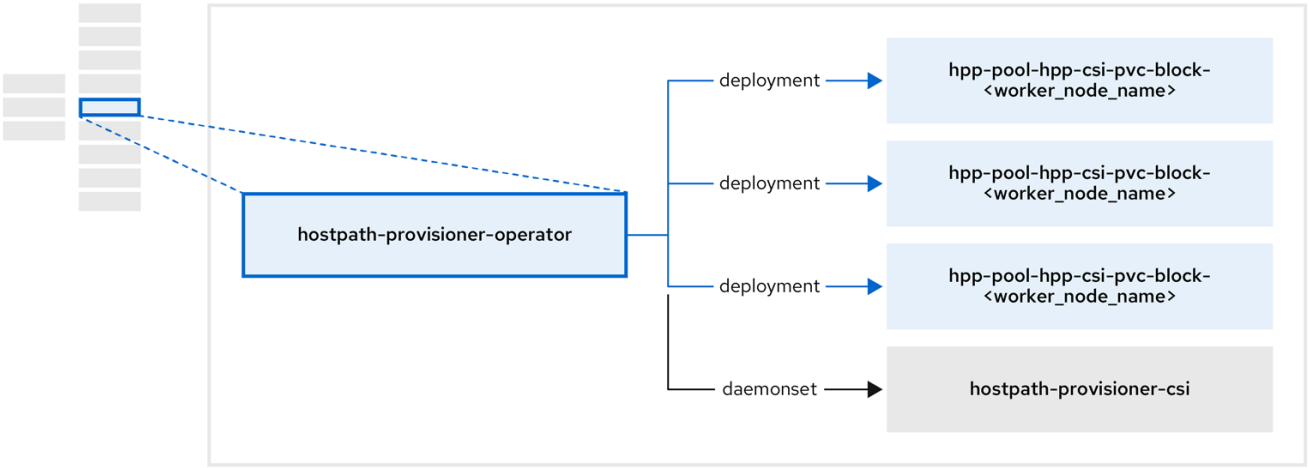


表2.4 hostpath-provisioner-operator コンポーネント

コンポーネント	説明
deployment/hpp-pool-hpp-csi-pvc-block-<worker_node_name>	ホストパスプロビジョナー (HPP) の実行が指定されている各ノードにワーカーを提供します。Pod は、指定されたバッキングストレージをノードにマウントします。
daemonset/hostpath-provisioner-csi	HPP の Container Storage Interface (CSI) ドライバーインターフェイスを実装します。

コンポーネント	説明
daemonset/hostpath-provisioner	HPP のレガシードライバーインターフェイスを実装します。

2.6. SSP-OPERATOR について

ssp-operator は、共通テンプレート、関連するデフォルトのブートソース、およびテンプレートバリデーターをデプロイします。

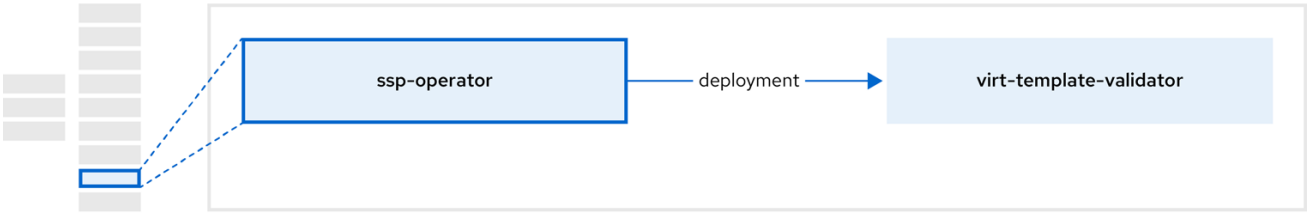


表2.5 ssp-operator components

コンポーネント	説明
deployment/virt-template-validator	テンプレートから作成された仮想マシンの **vm.kubevirt.io/validations** アノテーションをチェックし、無効な場合は拒否します。

2.7. TEKTON-TASKS-OPERATOR について

tekton-tasks-operator は、VM 用の OpenShift パイプラインの使用法を示すサンプルパイプラインをデプロイします。また、ユーザーがテンプレートから VM を作成し、テンプレートをコピーおよび変更し、データボリュームを作成できるようにする追加の OpenShift Pipeline タスクをデプロイします。

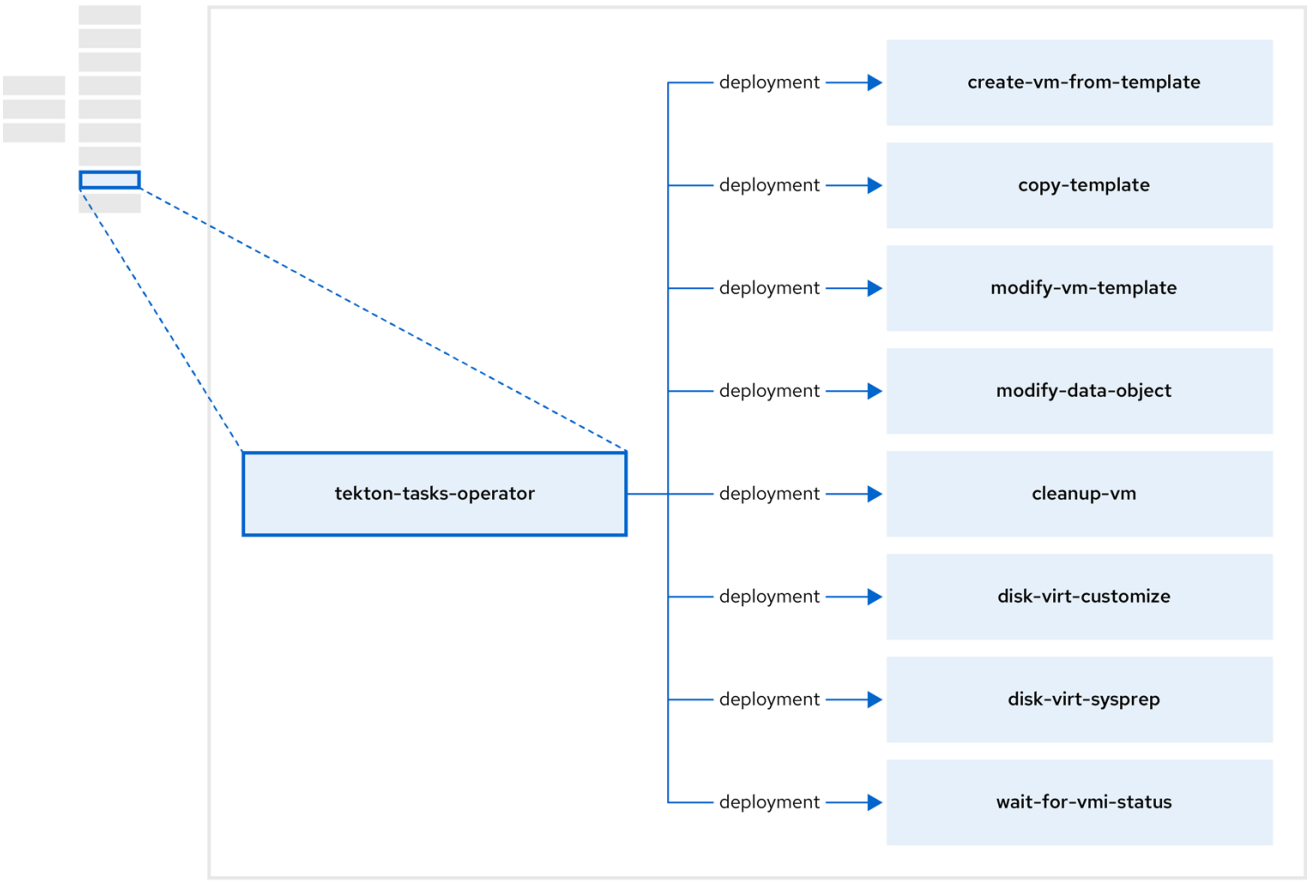


表2.6 tekton-tasks-operator components

コンポーネント	説明
deployment/create-vm-from-template	テンプレートから仮想マシンを作成します。
deployment/copy-template	仮想マシンテンプレートをコピーします。
deployment/modify-vm-template	仮想マシンテンプレートを作成または削除します。
deployment/modify-data-object	データボリュームまたはデータソースを作成または削除します。
deployment/cleanup-vm	仮想マシンでスクリプトまたはコマンドを実行し、後で仮想マシンを停止または削除します。
deployment/disk-virt-customize	**virt-customize** を使用して、ターゲット PVC で **customize** スクリプトを実行します。
deployment/disk-virt-sysprep	**virt-sysprep** を使用して、ターゲット PVC で **sysprep** スクリプトを実行します。
deployment/wait-for-vmi-status	特定の VMI ステータスを待機し、そのステータスに従って失敗または成功します。

2.8. 仮想 OPERATOR について

virt-operator は、現在の仮想マシン (VM) のワークロードを中断することなく、OpenShift Virtualization をデプロイ、アップグレード、および管理します。

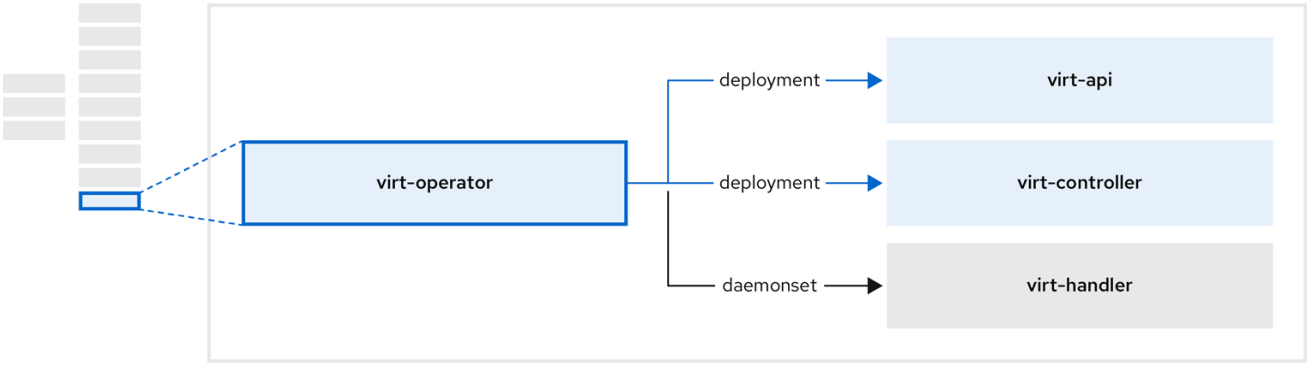


表2.7 virt-operator コンポーネント

コンポーネント	説明
deployment/virt-api	すべての仮想化関連フローのエントリーポイントとして機能する HTTP API サーバー。
deployment/virt-controller	新しい VM インスタンスオブジェクトの作成を監視し、対応する Pod を作成します。Pod がノードでスケジュールされると、**virt-controller** は VM をノード名で更新します。
daemonset/virt-handler	VM への変更を監視し、必要な操作を実行するように **virt-launcher** に指示します。このコンポーネントはノード固有です。
pod/virt-launcher	**libvirt** および **qemu** によって実装された、ユーザーによって作成された VM が含まれます。

第3章 OPENSHIFT VIRTUALIZATION の開始

基本的な環境をインストールして設定することにより、OpenShift Virtualization の特徴と機能を調べることができます。

注記

クラスター設定手順には、**cluster-admin** 権限が必要です。

3.1. OPENSHIFT VIRTUALIZATION の計画とインストール

OpenShift Container Platform クラスターで OpenShift Virtualization を計画およびインストールします。

- OpenShift Virtualization のベアメタルクラスターを計画 します。
- OpenShift Virtualization 用にクラスターを準備 します。
- OpenShift Virtualization Operator をインストール します。
- **virtctl** コマンドラインインターフェイス (CLI) ツールをインストール します。

計画およびインストールのリソース

- 仮想マシンディスクのストレージボリュームについて
- CSI 対応のストレージプロバイダーの使用
- 仮想マシンのローカルストレージの設定
- Kubernetes NMState Operator のインストール
- 仮想マシンのノードの指定
- **Virtctl** コマンド

3.2. 仮想マシンの作成と管理

Web コンソールを使用して仮想マシン (VM) を作成します。

- 仮想マシンをクイック作成します。
- テンプレートをカスタマイズして仮想マシンを作成します。

仮想マシンに接続します。

- Web コンソールを使用して、仮想マシンの Serial コンソール または VNC コンソール に接続します。
- SSH を使用して仮想マシンに接続します。
- RDP を使用して Windows 仮想マシンに接続します。

仮想マシンを管理します。

- Web コンソールを使用して、仮想マシンを停止、開始、一時停止、再起動します。

- **virtctl** CLI ツールを使用して、仮想マシンを管理したり、ポートを公開したり、シリアルコンソールに接続したりします。
- 仮想マシンのエクスポート。

3.3. 次のステップ

- VM をセカンダリーネットワークに接続します。
  - 仮想マシンを Linux ブリッジネットワークに接続します。
  - 仮想マシンを SR-IOV ネットワークに接続します。

注記

VM はデフォルトで Pod ネットワークに接続されます。Linux ブリッジや SR-IOV などのセカンダリーネットワークを設定してから、そのネットワークを仮想マシン設定に追加する必要があります。

- 仮想マシンをライブマイグレーションします。
- 仮想マシンをバックアップおよび復元します。
- クラスターのチューニングとスケーリング

第4章 WEB コンソールの概要

OpenShift Container Platform Web コンソールの **Virtualization** セクションには、OpenShift Virtualization 環境を管理および監視するための以下のページが含まれています。

表4.1 Virtualization ページ

Page	説明
Overview ページ	OpenShift Virtualization 環境を管理および監視します。
Catalog ページ	テンプレートのカタログから VirtualMachine を作成します。
VirtualMachines ページ	VirtualMachine を設定および監視します。
Templates ページ	テンプレートを作成および管理します。
DataSources ページ	VirtualMachine ブートソースの DataSource を作成および管理します。
MigrationPolicies ページ	ワークロードの MigrationPolicy を作成および管理します。

表4.2 キー

アイコン	説明
	Edit アイコン
	Link アイコン

4.1. OVERVIEW ページ

Overview ページには、リソース、メトリック、移行の進行状況、およびクラスターレベルの設定が表示されます。

例4.1 Overview ページ

要素	説明
virtctl のダウンロード 	リソースを管理するには、**virtctl** コマンドラインツールをダウンロードします。
Overview タブ	リソース、使用率、アラート、およびステータス。
Top consumers タブ	CPU、メモリー、およびストレージリソースのトップコンシューマー。
Migrations タブ	ライブマイグレーションのステータス。

要素	説明
Settings タブ	ライブマイグレーションの制限やパーミッションなど、クラスター全体の設定。

4.1.1. Overview タブ

Overview タブには、リソース、使用率、アラート、およびステータスが表示されます。

例4.2 Overview タブ

要素	説明
Getting started resources カード	• **Quick Starts** タイル: VirtualMachine の作成、インポート、および実行方法の段階的な手順とタスクを確認できます。 • **Feature highlights** タイル: 主要な仮想化機能に関する最新情報が表示されます。 • **Related operators** タイル: Kubernetes NMState Operator や OpenShift Data Foundation Operator などの Operator をインストールします。
VirtualMachines タイル	過去 7 日間の傾向を示すグラフを含む VirtualMachine の数。
vCPU usage タイル	過去 7 日間の傾向を示すグラフを含む vCPU 使用率。
Memory タイル	過去 7 日間の傾向を示すグラフを含むメモリー使用率。
Storage タイル	過去 7 日間の傾向を示すグラフを含むストレージ使用率。
Alerts タイル	重大度別にグループ化された OpenShift Virtualization アラート。
VirtualMachine statuses タイル	ステータス別にグループ化された VirtualMachine の数。
VirtualMachines per template グラフ	テンプレート名でグループ化された、テンプレートから作成された VirtualMachine の数。

4.1.2. Top consumers タブ

Top consumers タブには、CPU、メモリー、およびストレージのトップコンシューマーが表示されます。

例4.3 Top consumers タブ

要素	説明
仮想化ダッシュボードを表示	OpenShift Virtualization のトップコンシューマーを表示する **Observe → Dashboards** へのリンク。
Time period リスト	結果をフィルタリングする期間を選択します。
Top consumers リスト	結果をフィルタリングするトップコンシューマーの数を選択します。
CPU チャート	CPU 使用率が最も高い VirtualMachine。
Memory チャート	メモリー使用率が最も多い VirtualMachine。
Memory swap traffic チャート	メモリースワップトラフィックが最も多い VirtualMachine。
vCPU wait チャート	vCPU 待機時間が最も長い VirtualMachine。
Storage throughput チャート	ストレージスループットの使用率が最も高い VirtualMachines。
Storage IOPS チャート	1 秒あたりのストレージ入出力操作の使用率が最も高い VirtualMachines。

4.1.3. Migrations タブ

Migrations タブには、VirtualMachineInstance の移行のステータスが表示されます。

例4.4 Migrations タブ

要素	説明
Time period リスト	VirtualMachineInstanceMigration をフィルタリングする期間を選択します。
VirtualMachineInstanceMigrations テーブル	VirtualMachineInstance 移行のリスト。

4.1.4. Settings タブ

Settings タブには、次のタブにクラスター全体の設定が表示されます。

表4.3 Settings タブ上のタブ

タブ	説明
General タブ	OpenShift Virtualization のバージョンと更新ステータス。
Live migration タブ	ライブマイグレーションの制限とネットワーク設定。
Templates project タブ	Red Hat テンプレートのプロジェクト。
User permissions タブ	クラスター全体のパーミッション。

4.1.4.1. General タブ

General タブには、OpenShift Virtualization のバージョンと更新ステータスが表示されます。

例4.5 General タブ

Label	説明
サービス名	OpenShift Virtualization
Provider	Red Hat
インストール済みバージョン	4.13.11
更新ステータス	例: **Up to date**
チャネル	更新用に選択されたチャネル。

4.1.4.2. Live migration タブ

Live migration タブでライブマイグレーションを設定できます。

例4.6 Live migration タブ

要素	説明
Max. migrations per cluster フィールド	クラスターごとのライブマイグレーションの最大数を選択します。
Max. migrations per node フィールド	ノードごとのライブマイグレーションの最大数を選択します。
Live migration network リスト	ライブマイグレーション専用のセカンダリーネットワークを選択します。

4.1.4.3. Templates project タブ

Templates project タブで、テンプレートのプロジェクトを選択できます。

例4.7 Templates project タブ

要素	説明
Project リスト	Red Hat テンプレートを保存するプロジェクトを選択します。デフォルトのテンプレートプロジェクトは **openshift** です。 複数のテンプレートプロジェクトを定義する場合は、各プロジェクトの Templates ページ でテンプレートを複製する必要があります。

4.1.4.4. User permissions タブ

User permissions タブには、タスクに対するクラスター全体のパーミッションが表示されます。

例4.8 User permissions タブ

要素	説明
User Permissions テーブル	**テンプレートの共有** やパーミッションなどのタスクのリスト。

4.2. CATALOG ページ

Catalog ページでテンプレートを選択して、VirtualMachine を作成できます。

例4.9 Catalog ページ

要素	説明
Template catalog タブ	VirtualMachine の作成元となるテンプレートを選択します。

4.2.1. Template catalog タブ

要素	説明

要素	説明
Template project リスト	テンプレートが配置されているプロジェクトを選択します。 デフォルトでは、Red Hat テンプレートは **openshift** プロジェクトに保存されます。Overview → Settings → Template projectタブ でテンプレートプロジェクトを編集できます。
All items|Default templates	**Default templates** をクリックして、デフォルトのテンプレートのみを表示します。
Boot source available チェックボックス	チェックボックスをオンにして、使用可能なブートソースを含むテンプレートを表示します。
Operating system チェックボックス	チェックボックスをオンにして、選択したオペレーティングシステムのテンプレートを表示します。
Workload チェックボックス	チェックボックスをオンにして、選択したワークロードを含むテンプレートを表示します。
Search フィールド	テンプレートをキーワードで検索します。
テンプレートタイル	テンプレートタイルをクリックして、テンプレートの詳細を表示し、VirtualMachine を作成します。

4.3. VIRTUALMACHINES ページ

VirtualMachines ページで VirtualMachine を作成および管理できます。

例4.10 VirtualMachines ページ

要素	説明
Create → From template	**Catalog** ページ →**Template catalog** タブで VirtualMachine を作成します。
Create → From YAML	YAML 設定ファイルを編集して VirtualMachine を作成します。
Filter フィールド	ステータス、テンプレート、オペレーティングシステム、またはノードで VirtualMachine をフィルタリングします。
Search フィールド	名前またはラベルで VirtualMachine を検索します。

要素	説明
VirtualMachines テーブル	VirtualMachine のリスト。 VirtualMachine の横の **Options** メニュー ⋮ をクリックして、**Stop**、**Restart**、**Pause**、**Clone**、**Migrate**、**Copy SSH command**、**Edit labels**、**Edit annotations**、または **Delete** を選択します。 VirtualMachine をクリックして、VirtualMachine details ページに移動します。

4.3.1. VirtualMachine details ページ

VirtualMachine details ページで VirtualMachine を設定できます。

例4.11 VirtualMachine details ページ

要素	説明
Actions メニュー	**Actions** メニューをクリックして、**Stop**、**Restart**、**Pause**、**Clone**、**Migrate**、**Copy SSH command**、**Edit labels**、**Edit annotations**、または **Delete** を選択します。
Overview タブ	リソースの使用率、アラート、ディスク、およびデバイス。
Details タブ	VirtualMachine の詳細と設定
Metrics タブ	メモリー、CPU、ストレージ、ネットワーク、移行のメトリック。
YAML タブ	VirtualMachine YAML 設定ファイル。
Configuration タブ	**Scheduling**、**Environment**、**Network interfaces**、**Disks**、および **Scripts** タブが含まれています。
Configuration → Scheduling タブ	特定のノードで実行するように VirtualMachine をスケジュールする
Configuration → Environment タブ	設定マップ、シークレット、およびサービスアカウントの管理。
Configuration → Network interfaces タブ	ネットワークインターフェイス
Configuration → Disks タブ	ディスク

要素	説明
Configuration → Scripts タブ	Cloud-init 設定、Linux VirtualMachines の SSH キー、Windows VirtualMachines の Sysprep 応答ファイル
Events タブ	VirtualMachine イベントストリーム。
Console タブ	コンソールセッション管理。
Snapshots タブ	スナップショット管理
Diagnostics タブ	ステータス条件とボリュームスナップショットのステータス

4.3.1.1. Overview タブ

Overview タブには、リソースの使用率、アラート、および設定情報が表示されます。

例4.12 Overview タブ

要素	説明
Details タイル	VirtualMachine の一般情報。
Utilization タイル	CPU、Memory、Storage、および Network transfer グラフ。デフォルトでは、**ネットワーク転送** は、すべてのネットワークの合計を表示します。特定のネットワークの内訳を表示するには、Breakdown by network をクリックします。
Hardware devices タイル	GPU とホストデバイス。
Alerts タイル	重大度別にグループ化された OpenShift Virtualization アラート。
Snapshots タイル	Take snapshot および Snapshots テーブル。
Network interfaces タイル	Network interfaces テーブル。
Disks タイル	Disks テーブル

4.3.1.2. Details タブ

Details タブで、VirtualMachine に関する情報を表示し、ラベル、アノテーション、およびその他のメタデータを編集できます。

例4.13 Details タブ

要素	説明
YAML スイッチ	**ON** を設定して、YAML 設定ファイルでライブの変更を表示します。
Name	VirtualMachine 名。
Namespace	VirtualMachine namespace
Labels	編集アイコンをクリックして、ラベルを編集します。
Annotations	編集アイコンをクリックして、注釈を編集します。
説明	編集アイコンをクリックして、説明を入力します。
オペレーティングシステム	オペレーティングシステム名。
CPU|Memory	編集アイコンをクリックして、CPU|Memory 要求を編集します。 CPU の数は、**sockets * threads * cores** の式を使用して計算されます。
Machine type	VirtualMachine マシンタイプ。
Boot mode	編集アイコンをクリックして、起動モードを編集します。
一時停止モードで開始	編集アイコンをクリックして、この設定を有効にします。
テンプレート	VirtualMachine の作成に使用されるテンプレートの名前。
Created at	VirtualMachine の作成日。
Owner	VirtualMachine の所有者。
ステータス	VirtualMachine のステータス。
Pod	**virt-launcher** Pod 名。
VirtualMachineInstance	VirtualMachine インスタンス名。
Boot order	編集アイコンをクリックして、起動ソースを選択します。
IP アドレス	VirtualMachine の IP アドレス。

要素	説明
Hostname	VirtualMachine のホスト名。
タイムゾーン	VirtualMachine のタイムゾーン。
ノード	VirtualMachine が実行されているノード。
Workload profile	編集アイコンをクリックして、ワークロードプロファイルを編集します。
virtctl を使用している SSH	コピーアイコンをクリックして、**virtctl ssh** コマンドをクリップボードにコピーします。
SSH サービスタイプのオプション	**SSH over LoadBalancer** または **SSH over NodePort** を選択します。
GPU devices	編集アイコンをクリックして、GPU デバイスを追加します。
Host devices	編集アイコンをクリックして、ホストデバイスを追加します。
Headless モード	編集アイコンをクリックして、ヘッドレスモードを有効にします。
Services セクション	QEMU ゲストエージェントがインストールされている場合、サービスを表示します。
Active users セクション	QEMU ゲストエージェントがインストールされている場合、アクティブなユーザーを表示します。

4.3.1.3. Metrics タブ

Metrics タブには、メモリー、CPU、ストレージ、ネットワーク、および移行の使用率グラフが表示されます。

例4.14 Metrics タブ

要素	説明
Time range リスト	結果をフィルタリングする期間を選択します。
Virtualization ダッシュボード	現在のプロジェクトの **Workloads** タブにリンクします。
Utilization セクション	**Memory** および **CPU** グラフ
Storage セクション	**Storage total read/write** および **Storage IOPS total read/write** グラフ。

要素	説明
Network セクション	**Network in**、**Network out**、**Network bandwidth**、および **Network interface** グラフ。**Network interface** ドロップダウンから **All networks** または特定のネットワークを選択します。
Migration セクション	**Migration** および **KV data transfer rate** グラフ。

4.3.1.4. YAML タブ

YAML タブで YAML ファイルを編集して、VirtualMachine を設定できます。

例4.15 YAML タブ

要素	説明
Save ボタン	変更を YAML ファイルに保存します。
Reload ボタン	変更を破棄し、YAML ファイルをリロードします。
Cancel ボタン	**YAML** タブを終了します。
Download ボタン	YAML ファイルをローカルマシンにダウンロードします。

4.3.1.5. Configuration タブ

Configuration タブで、スケジュール、ネットワークインターフェイス、ディスク、およびその他のオプションを設定できます。

例4.16 Configuration タブ上のタブ

タブ	説明
Scheduling タブ	特定のノードで実行するように VirtualMachine をスケジュールする
Environment タブ	設定マップ、シークレット、およびサービスアカウント
Network interfaces タブ	ネットワークインターフェイス
Disks タブ	ディスク

タブ	説明
Scripts タブ	Cloud-init 設定、Linux VirtualMachines の SSH キー、Windows VirtualMachines の Sysprep 応答ファイル

4.3.1.5.1. Scheduling タブ

Scheduling タブで、特定のノードで実行するように VirtualMachine を設定できます。

例4.17 Scheduling タブ

設定	Description
YAML スイッチ	ON を設定して、YAML 設定ファイルでライブの変更を表示します。
Node selector	編集アイコンをクリックして、ラベルを追加し、適格なノードを指定します。
容認	編集アイコンをクリックして、許容範囲を追加し、適格なノードを指定します。
アフィニティールール	編集アイコンをクリックして、アフィニティールールを追加します。
Descheduler スイッチ	Descheduler を有効または無効にします。Descheduler は、実行中の Pod をエビクトして、Pod をより適切なノードに再スケジュールできるようにします。
専用リソース	編集アイコンをクリックして、**Schedule this workload with dedicated resources (guaranteed policy)** を選択します。
エビクションストラテジー	編集アイコンをクリックして、VirtualMachineInstance エビクション戦略として **LiveMigrate** を選択します。

4.3.1.5.2. Environment タブ

Environment タブで設定マップ、シークレット、およびサービスアカウントを管理できます。

例4.18 Environment タブ

要素	説明
YAML スイッチ	ON を設定して、YAML 設定ファイルでライブの変更を表示します。
Add Config Map, Secret or Service Account 	リンクをクリックして、リソースリストから設定マップ、シークレット、またはサービスアカウントを選択します。

4.3.1.5.3. Network interfaces タブ

Network interfaces タブでネットワークインターフェイスを管理できます。

例4.19 Network interfaces タブ

設定	Description
YAML スイッチ	ON を設定して、YAML 設定ファイルでライブの変更を表示します。
Add network interface ボタン	VirtualMachine にネットワークインターフェイスを追加します。
Filter フィールド	インターフェイスタイプでフィルタリングします。
Search フィールド	名前またはラベルでネットワークインターフェイスを検索します。
Network interface テーブル	ネットワークインターフェイスのリスト。 ネットワークインターフェイスの横にある Options メニュー  をクリックして、Edit または Delete を選択します。

4.3.1.5.4. Disks タブ

Disks タブでディスクを管理できます。

例4.20 Disks タブ

設定	Description
YAML スイッチ	ON を設定して、YAML 設定ファイルでライブの変更を表示します。
Add disk ボタン	VirtualMachine にディスクを追加します。
Filter フィールド	ディスクの種類でフィルタリングします。
Search フィールド	ディスクを名前で検索します。
Mount Windows drivers disk チェックボックス	一時コンテナーディスクを CD-ROM としてマウントする場合に選択します。

設定	Description
Disks テーブル	VirtualMachine ディスクのリスト。 ディスクの横にある **Options** メニュー  をクリックして、**Edit**、**Detach**、または **Make persistent** を選択します。
File systems テーブル	QEMU ゲストエージェントがインストールされている場合の VirtualMachine ファイルシステムのリスト

4.3.1.5.5. Scripts タブ

Scripts タブで、cloud-init を設定し、Linux VirtualMachine の SSH キーを追加し、Windows VirtualMachine の Sysprep 応答ファイルをアップロードできます。

例4.21 Scripts タブ

要素	説明
YAML スイッチ	**ON** を設定して、YAML 設定ファイルでライブの変更を表示します。
Cloud-init	編集アイコンをクリックして、cloud-init 設定を編集します。
認可された SSH キー	編集アイコンをクリックして、新しいシークレットを作成するか、既存のシークレットを添付します。
Sysprep	編集アイコンをクリックして **Autounattend.xml** または **Unattend.xml** 応答ファイルをアップロードし、Windows VirtualMachine のセットアップを自動化します。

4.3.1.6. Events タブ

Events タブには、VirtualMachine イベントのリストが表示されます。

4.3.1.7. Console タブ

Console タブで、VirtualMachine へのコンソールセッションを開くことができます。

例4.22 Console タブ

要素	説明

要素	説明
Guest login credentials セクション	**Guest login credentials** を展開して、**cloud-init** で作成された認証情報を表示します。コピーアイコンをクリックして、認証情報をクリップボードにコピーします。
Console リスト	**VNC コンソール** または **Serial コンソール** を選択します。 **デスクトップビューアー** を選択して、リモートデスクトッププロトコル (RDP) を使用して Windows Virtual Machines に接続できます。同じネットワーク上のマシンに RDP クライアントをインストールする必要があります。
Send key リスト	コンソールに送信するキーストロークの組み合わせを選択します。
Disconnect ボタン	コンソール接続を切断します。 新しいコンソールセッションを開く場合は、コンソール接続を手動で切断する必要があります。それ以外の場合、最初のコンソールセッションは引き続きバックグラウンドで実行されます。
Paste ボタン	VNC コンソールを使用すると、クライアントのクリップボードからゲストに文字列を貼り付けることができます。

4.3.1.8. Snapshots タブ

Snapshots タブで、スナップショットを作成し、スナップショットから VirtualMachine を復元できます。

例4.23 Snapshots タブ

要素	説明
Take snapshot ボタン	スナップショットを作成します。
Filter フィールド	スナップショットをステータスでフィルタリングします。
Search フィールド	名前またはラベルでスナップショットを検索します。
Snapshot テーブル	スナップショットのリスト。 スナップショット名をクリックして、ラベルまたはアノテーションを編集します。 スナップショットの横にある **Options** メニュー  をクリックして、**Restore** または **Delete** を選択します。

4.3.1.9. Diagnostics タブ

Diagnostics タブでステータス条件とボリュームスナップショットのステータスを表示できます。

例4.24 Diagnostics タブ

要素	説明
Status conditions テーブル	仮想マシンのあらゆる側面を報告する条件リストを表示します。
Filter フィールド	ステータス条件をカテゴリーと条件でフィルタリングします。
Search フィールド	ステータス条件を理由で検索します。
Manage columns アイコン	表示する列を選択します。
Volume snapshot テーブル	ボリューム、スナップショットの有効化ステータス、および理由のリスト

4.4. TEMPLATES ページ

Templates ページで VirtualMachine テンプレートを作成、編集、および複製できます。

注記

Red Hat テンプレートは編集できません。Red Hat テンプレートを複製して編集し、カスタムテンプレートを作成できます。

例4.25 Templates ページ

要素	説明
Create Template ボタン	YAML 設定ファイルを編集してテンプレートを作成します。
Filter フィールド	テンプレートをタイプ、ブートソース、テンプレートプロバイダー、またはオペレーティングシステムでフィルタリングします。
Search フィールド	名前またはラベルでテンプレートを検索します。

要素	説明
Templates テーブル	テンプレートのリスト。 テンプレートの横にある Options メニュー  をクリックして、Edit、Clone、Edit boot source、Edit boot source reference、Edit labels、Edit annotations、または Delete を選択します。

4.4.1. テンプレートの詳細ページ

テンプレートの詳細 ページで、テンプレートの設定を表示し、カスタムテンプレートを編集できます。

例4.26 テンプレートの詳細ページ

要素	説明
Actions メニュー	Actions メニューをクリックして、Edit、Clone、Edit boot source、Edit boot source reference の Edit labels の Edit annotations、または Delete を選択します。
Details タブ	テンプレートの設定
YAML タブ	YAML 設定ファイル。
Scheduling タブ	スケジューリング設定。
Network interfaces タブ	ネットワークインターフェイス管理。
Disks タブ	ディスク管理。
Scripts タブ	Cloud-init、SSH キー、および Sysprep 管理
Parameters タブ	パラメーター

4.4.1.1. Details タブ

Details タブでカスタムテンプレートを設定できます。

例4.27 Details タブ

要素	説明
YAML スイッチ	**ON** を設定して、YAML 設定ファイルでライブの変更を表示します。
Name	テンプレート名
Namespace	テンプレートの namespace。
Labels	編集アイコンをクリックして、ラベルを編集します。
Annotations	編集アイコンをクリックして、注釈を編集します。
Display name	編集アイコンをクリックして、表示名を編集します。
Description	編集アイコンをクリックして、説明を入力します。
オペレーティングシステム	オペレーティングシステム名。
CPU|Memory	編集アイコンをクリックして、CPU|Memory 要求を編集します。 CPU の数は、**sockets * threads * cores** の式を使用して計算されます。
Machine type	テンプレートマシンタイプ。
Boot mode	編集アイコンをクリックして、起動モードを編集します。
Base template	このテンプレートの作成に使用されたベーステンプレートの名前。
Created at	テンプレートの作成日。
Owner	テンプレートの所有者。
Boot order	テンプレートの起動順序。
Boot source	ブートソースの可用性。
Provider	テンプレートプロバイダー
Support	テンプレートのサポートレベル。
GPU devices	編集アイコンをクリックして、GPU デバイスを追加します。
Host devices	編集アイコンをクリックして、ホストデバイスを追加します。

4.4.1.2. YAML タブ

YAML タブで YAML ファイルを編集して、カスタムテンプレートを設定できます。

例4.28 YAML タブ

要素	説明
Save ボタン	変更を YAML ファイルに保存します。
Reload ボタン	変更を破棄し、YAML ファイルをリロードします。
Cancel ボタン	**YAML** タブを終了します。
Download ボタン	YAML ファイルをローカルマシンにダウンロードします。

4.4.1.3. Scheduling タブ

Scheduling タブでスケジュールを設定できます。

例4.29 Scheduling タブ

設定	Description
YAML スイッチ	**ON** を設定して、YAML 設定ファイルでライブの変更を表示します。
Node selector	編集アイコンをクリックして、ラベルを追加し、適格なノードを指定します。
容認	編集アイコンをクリックして、許容範囲を追加し、適格なノードを指定します。
アフィニティールール	編集アイコンをクリックして、アフィニティールールを追加します。
Descheduler スイッチ	Descheduler を有効または無効にします。Descheduler は、実行中の Pod をエビクトして、Pod をより適切なノードに再スケジュールできるようにします。
専用リソース	編集アイコンをクリックして、**Schedule this workload with dedicated resources (guaranteed policy)** を選択します。
エビクションストラテジー	編集アイコンをクリックして、VirtualMachineInstance エビクション戦略として **LiveMigrate** を選択します。

4.4.1.4. Network interfaces タブ

Network interfaces タブでネットワークインターフェイスを管理できます。

例4.30 Network interfaces タブ

設定	Description
YAML スイッチ	ON を設定して、YAML 設定ファイルでライブの変更を表示します。
Add network interface ボタン	テンプレートにネットワークインターフェイスを追加します。
Filter フィールド	インターフェイスタイプでフィルタリングします。
Search フィールド	名前またはラベルでネットワークインターフェイスを検索します。
Network interface テーブル	ネットワークインターフェイスのリスト。 ネットワークインターフェイスの横にある Options メニュー  をクリックして、Edit または Delete を選択します。

4.4.1.5. Disks タブ

Disks タブでディスクを管理できます。

例4.31 Disks タブ

設定	Description
YAML スイッチ	ON を設定して、YAML 設定ファイルでライブの変更を表示します。
Add disk ボタン	テンプレートにディスクを追加します。
Filter フィールド	ディスクの種類でフィルタリングします。
Search フィールド	ディスクを名前で検索します。
Disks テーブル	テンプレートディスクのリスト。 ディスクの横にある Options メニュー  をクリックして、Edit または Detach を選択します。

4.4.1.6. Scripts タブ

Scripts タブで、cloud-init 設定、SSH キー、および Sysprep 応答ファイルを管理できます。

例4.32 Scripts タブ

要素	説明
YAML スイッチ	**ON** を設定して、YAML 設定ファイルでライブの変更を表示します。
Cloud-init	編集アイコンをクリックして、cloud-init 設定を編集します。
認可された SSH キー	編集アイコンをクリックして、新しいシークレットを作成するか、既存のシークレットを添付します。
Sysprep	編集アイコンをクリックして **Autounattend.xml** または **Unattend.xml** 応答ファイルをアップロードし、Windows VirtualMachine のセットアップを自動化します。

4.4.1.7. パラメータータブ

パラメーター タブで、選択したテンプレート設定を編集できます。

例4.33 パラメータータブ

要素	説明
YAML スイッチ	**ON** を設定して、YAML 設定ファイルでライブの変更を表示します。
VM name	生成された値には **Generated (expression)**、デフォルト値を設定するには **Value**、または **Default value type** リストから **None** を選択します。
DataSource 名。	生成された値には **Generated (expression)**、デフォルト値を設定するには **Value**、または **Default value type** リストから **None** を選択します。
DataSource の namespace。	生成された値には **Generated (expression)**、デフォルト値を設定するには **Value**、または **Default value type** リストから **None** を選択します。
cloud-user パスワード	生成された値には **Generated (expression)**、デフォルト値を設定するには **Value**、または **Default value type** リストから **None** を選択します。

4.5. DATASOURCES ページ

DataSources ページで、VirtualMachine ブートソースの DataSource を作成および設定できます。

DataSource を作成すると、**DataImportCron** リソースは、ブートソースの自動更新を無効にしないかぎり、ディスクイメージをポーリングしてインポートする cron ジョブを定義します。

例4.34 DataSources ページ

要素	Description
Create DataSource → With form	レジストリー URL、ディスクサイズ、リビジョン数、および cron 式をフォームに入力して、DataSource を作成します。
Create DataSources → With YAML	YAML 設定ファイルを編集して DataSource を作成します。
Filter フィールド	利用可能な DataImportCron などの属性で DataSource をフィルタリングします。
Search フィールド	名前またはラベルで DataSource を検索します。
DataSources テーブル	DataSource のリスト。 DataSource の横にある **Options** メニュー  をクリックして、**Edit labels**、**Edit annotations**、または **Delete** を選択します。

DataSource をクリックして、DataSource details ページを表示します。

4.5.1. DataSource details ページ

DataSource details ページで DataSource を設定できます。

例4.35 DataSource details ページ

要素	説明
Details タブ	フォームを編集して DataSource を設定します。
YAML タブ	YAML 設定ファイルを編集して DataSource を設定します。
Actions メニュー	**Edit labels**、**Edit annotations**、**Delete**、または **Manage source** を選択します。
Name	DataSource 名。
Namespace	DataSource の namespace。
DataImportCron	DataSource DataImportCron
Labels	編集アイコンをクリックして、ラベルを編集します。
Annotations	編集アイコンをクリックして、注釈を編集します。

要素	説明
Conditions	DataSource のステータス条件を表示します。
Created at	DataSource の作成日
Owner	DataSource の所有者

4.6. MIGRATIONPOLICIES ページ

MigrationPolicies ページでワークロードの MigrationPolicy を管理できます。

例4.36 MigrationPolicies ページ

要素	Description
Create MigrationPolicy → With form	フォームに設定とラベルを入力して、MigrationPolicy を作成します。
Create MigrationPolicy → With YAML	YAML 設定ファイルを編集して MigrationPolicy を作成します。
Name | Label 検索フィールド	名前またはラベルで MigrationPolicy を検索します。
MigrationPolicies テーブル	MigrationPolicy のリスト。 MigrationPolicy の横にある **Options** メニュー  をクリックして、**Edit** または **Delete** を選択します。

MigrationPolicy をクリックして、MigrationPolicy details ページを表示します。

4.6.1. MigrationPolicy details ページ

MigrationPolicy details ページで MigrationPolicy を設定できます。

例4.37 MigrationPolicy details ページ

要素	説明
Details タブ	フォームを編集して MigrationPolicy を設定します。

要素	説明
YAML タブ	YAML 設定ファイルを編集して MigrationPolicy を設定します。
Actions メニュー	**Edit** または **Delete** を選択します。
Name	MigrationPolicy 名。
説明	MigrationPolicy の説明。
設定	編集アイコンをクリックして、MigrationPolicy 設定を更新します。
移行ごとの帯域幅	移行ごとの帯域幅要求。帯域幅を無制限にするには、値を **0** に設定します。
自動収束	自動収束ポリシー。
ポストコピー	ポストコピーポリシー。
完了タイムアウト	秒単位の完了タイムアウト値。
プロジェクトラベル	**Edit** をクリックして、プロジェクトラベルを編集します。
VirtualMachine のラベル	**Edit** をクリックして、VirtualMachine のラベルを編集します。

第5章 OPENSHIFT VIRTUALIZATION リリースノート

5.1. 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、用語の置き換えは、今後の複数のリリースにわたって段階的に実施されます。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

5.2. RED HAT OPENSHIFT VIRTUALIZATION について

Red Hat OpenShift Virtualization は、従来の仮想マシン (VM) をコンテナーと共に実行される OpenShift Container Platform に組み込み、それらをネイティブ Kubernetes オブジェクトとして管理することを可能にします。

OpenShift Virtualization は、  アイコンで表されます。

OVN-Kubernetes または OpenShiftSDN のデフォルトの Container Network Interface (CNI) ネットワークプロバイダーで、OpenShift Virtualization を使用できます。

OpenShift Virtualization の機能 を参照してください。

OpenShift Virtualization のアーキテクチャーとデプロイメント の詳細を参照してください。

OpenShift Virtualization 用に クラスターを準備します。

5.2.1. OpenShift Virtualization サポートのクラスターバージョン

OpenShift Virtualization 4.13 は OpenShift Container Platform 4.13 クラスターで使用するためにサポートされます。Open Shift Virtualization の最新の z-stream リリースを使用するには、最初に Open Shift Container Platform の最新バージョンにアップグレードする必要があります。

重要

OpenShift Virtualization 4.12.2 から OpenShift Virtualization 4.13 への更新はサポート対象外です。

5.2.2. サポート対象のゲストオペレーティングシステム

OpenShift Virtualization でサポートされているゲストオペレーティングシステムを確認するには、Red Hat OpenStack Platform、Red Hat Virtualization、OpenShift Virtualization、Red Hat Enterprise Linux with KVM の認定ゲストオペレーティングシステム を参照してください。

5.3. 新機能および変更された機能

- OpenShift Virtualization が FIPS に対応しました。ただし、OpenShift Container Platform 4.13 は Red Hat Enterprise Linux (RHEL) 9.2 をベースにしています。RHEL 9.2 はまだ FIPS 認定のために提出されていません。ただし、特定の期限は確約できませんが、Red Hat は RHEL 9.0

および RHEL 9.2 モジュール、その後は RHEL 9.x のマイナーリリースについても、FIPS 認定を取得することを想定しています。更新は Compliance Activities and Government Standards から入手できる予定です。

- OpenShift Virtualization は、Windows Server のワークロードを実行する Microsoft の Windows Server Virtualization Validation Program (SVVP) で認定されています。
  SVVP の認定は以下に適用されます。
  - Red Hat Enterprise Linux CoreOS ワーカー。Microsoft SVVP Catalog では、**Red Hat OpenShift Container Platform 4 on RHEL CoreOS 9** という名前が付けられます。
  - Intel および AMD CPU。
- OpenShift Virtualization は、**restricted** Kubernetes pod security standards プロファイルに準拠するようになりました。詳細は、OpenShift Virtualization の セキュリティーポリシー ドキュメントを参照してください。
- 現在、OpenShift Virtualization は Red Hat Enterprise Linux (RHEL) 9 をベースにしています。
  - 仮想マシン用の新しい RHEL 9 マシンタイプ (**machineType: pc-q35-rhel9.2.0**) があります。
    OpenShift Virtualization に含まれるすべての仮想マシンテンプレートは、デフォルトでこのマシンタイプを使用するようになりました。
  - 詳細は、RHEL 9 上の OpenShift Virtualization を参照してください。
- 仮想マシンまたはスナップショットをエクスポートした後、エクスポートサーバーから **VirtualMachine**、**ConfigMap**、**Secret** マニフェストを取得できるようになりました。詳細は、エクスポートされた仮想マシンマニフェストへのアクセス を参照してください。
- 「ロギング、イベント、モニタリング」ドキュメントの名前は、サポート に変更されました。モニタリングツールのドキュメントは モニタリング に移動されました。
- LokiStack を使用すると、Web コンソールに集約された OpenShift Virtualization ログを表示およびフィルタリングできます。

5.3.1. クイックスタート

- クイックスタートツアーは、複数の OpenShift Virtualization 機能で利用できます。ツアーを表示するには、OpenShift Virtualization コンソールのヘッダーのメニューバーにある **Help** アイコン ? をクリックし、**Quick Starts** を選択します。**Filter** フィールドに **virtualization** キーワードを入力して、利用可能なツアーをフィルターできます。

5.3.2. ネットワーク

- OVN-Kubernetes CNI プラグインを使用すると、デフォルトの Pod ネットワークに接続されている 2 つの仮想マシン (VM) 間で、フラグメント化されていない jumbo フレームパケット を送信できます。

5.3.3. ストレージ

- OpenShift 仮想化ストレージリソースは、ベータ API バージョンに自動的に移行されるようになりました。Alpha API バージョンはサポートされなくなりました。

5.3.4. Web コンソール

- **VirtualMachine details** ページでは、**Scheduling**、**Environment**、**Network interfaces**、**Disks**、および **Scripts** タブが新しい **Configuration** タブ に表示されます。
- VNC コンソールの使用時に、クライアントのクリップボードからゲストに 文字列を貼り付ける ことができるようになりました。
- **VirtualMachine details** → **Details** タブに、ロードバランサー経由で SSH サービスを公開するための新しい SSH サービスタイプ (**SSH over LoadBalancer**) が追加されました。
- **Disks** タブ に、ホットプラグボリュームを永続ボリュームにするオプションが 追加されました。
- **VirtualMachine details** → **Diagnostics** タブが追加され、仮想マシンのステータス条件とボリュームのスナップショットステータスを表示できるようになりました。
- Web コンソールで高パフォーマンス仮想マシンのヘッドレスモードを有効化できるようになりました。

5.4. 非推奨の機能と削除された機能

5.4.1. 非推奨の機能

非推奨の機能は現在のリリースに含まれており、サポートされています。ただし、これらは今後のリリースで削除されるため、新規デプロイメントでの使用は推奨されません。

- RPM による Red Hat Enterprise Linux (RHEL) 7 および RHEL 9 の **virtctl** コマンドラインツールのインストールのサポートは非推奨となり、将来のリリースで削除される予定です。

5.4.2. 削除された機能

削除された機能は、現在のリリースではサポートされません。

- Red Hat Enterprise Linux 6 は、OpenShift Virtualization でサポートされなくなりました。
- 従来の HPP カスタムリソースと関連するストレージクラスのサポートは、すべての新しいデプロイメントで削除されました。OpenShift Virtualization 4.13 では、HPP Operator は Kubernetes Container Storage Interface (CSI) ドライバーを使用してローカルストレージを設定します。レガシー HPP カスタムリソースは、以前のバージョンの OpenShift Virtualization にインストールされていた場合にのみサポートされます。

5.5. テクノロジープレビューの機能

現在、今回のリリースに含まれる機能にはテクノロジープレビューのものがあります。これらの実験的機能は、実稼働環境での使用を目的としていません。これらの機能に関しては、Red Hat カスタマーポータルの以下のサポート範囲を参照してください。

テクノロジープレビュー機能のサポート範囲

- Prometheus を使用して次のメトリクスを監視できるようになりました。
  - **kubevirt_vmi_cpu_system_usage_seconds** は、ハイパーバイザーが消費した物理システム CPU 時間を返します。
  - **kubevirt_vmi_cpu_user_usage_seconds** は、ハイパーバイザーが消費した物理ユーザー CPU 時間を返します。

- **kubevirt_vmi_cpu_usage_seconds** は、vCPU とハイパーバイザーの合計使用量を計算し、使用された合計 CPU 時間を秒単位で返します。

- checkup を実行して、OpenShift Container Platform クラスターノードがパケット損失ゼロで Data Plane Development Kit (DPDK) ワークロードがある仮想マシンを実行できるか検証できるようになりました。

- 仮想マシンを DPDK ワークロードを実行するように設定 すると、ユーザー空間でのパケット処理を高速化するために、レイテンシーの短縮とスループットの向上を実現できます。

- 完全修飾ドメイン名 (FQDN) を使用して、クラスター外部から セカンダリーネットワークインターフェイスにアタッチされている仮想マシンにアクセス できるようになりました。

- OpenShift Virtualization VM によってホストされるワーカーノードを使用して OpenShift Container Platform クラスターを作成できるようになりました。詳細は、Red Hat Advanced Cluster Management (RHACM) ドキュメントの OpenShift Virtualization でのホストされたコントロールプレーンクラスターの管理 を参照してください。

- Microsoft Windows 11 をゲスト OS として使用できるようになりました。ただし、OpenShift Virtualization 4.13 は、BitLocker リカバリーの重要な機能に必要な USB ディスクをサポートしていません。回復キーを保護するには、BitLocker recovery guide で説明されている他の方法を使用します。

5.6. バグ修正

- Containerized Data Importer (CDI) が作成した一部の Persistent Volume Claim (PVC) アノテーションが原因で、仮想マシンのスナップショットのリストア操作が無期限にハングすることがなくなりました。(**BZ#2070366**)

5.7. 既知の問題

- RHSA-2023:3722 アドバイザリーのリリースにより、FIPS 対応 RHEL 9 システム上の TLS 1.2 接続に、TLS **Extended Master Secret** (EMS) エクステンション (RFC 7627) エクステンションが必須になりました。これは FIPS-140-3 要件に準拠しています。TLS 1.3 は影響を受けません。
  EMS または TLS 1.3 をサポートしていないレガシー OpenSSL クライアントは、RHEL 9 で実行されている FIPS サーバーに接続できなくなりました。同様に、FIPS モードの RHEL 9 クライアントは、EMS なしでは TLS 1.2 のみをサポートするサーバーに接続できません。これは実際には、これらのクライアントが RHEL 6、RHEL 7、および RHEL 以外のレガシーオペレーティングシステム上のサーバーに接続できないことを意味します。これは、OpenSSL のレガシー 1.0.x バージョンが EMS または TLS 1.3 をサポートしていないためです。詳細は、TLS Extension "Extended Master Secret" enforced with Red Hat Enterprise Linux 9.2 を参照してください。

  回避策として、レガシー OpenSSL クライアントを TLS 1.3 をサポートするバージョンにアップグレードし、FIPS モードで **最新の** TLS セキュリティープロファイルタイプで TLS 1.3 を使用するように OpenShift Virtualization を設定します。

- OpenShift Virtualization 4.12.4 で **HyperConverged** カスタムリソースを編集して **DisableMDEVConfiguration** 機能ゲートを有効にした場合は、バージョン 4.13.0 または 4.13.1 にアップグレードした後、JSON Patch アノテーション (**BZ#2184439)** を作成して機能ゲートを再度有効にする必要があります。):

```
$ oc annotate --overwrite -n openshift-cnv hyperconverged kubevirt-hyperconverged \
    kubevirt.kubevirt.io/jsonpatch='[{"op": "add","path":
```

```
"/spec/configuration/developerConfiguration/featureGates/-", \
    "value": "DisableMDEVConfiguration"}]'
```

- OpenShift Virtualization バージョン 4.12.2 以前は、OpenShift Container Platform 4.13 との互換性がありません。OpenShift Virtualization 4.12.1 および 4.12.2 では、設計により OpenShift Container Platform 4.13 への更新がブロックされますが、この制限は OpenShift Virtualization 4.12.0 に追加できませんでした。OpenShift Virtualization 4.12.0 を使用している場合は、OpenShift Container Platform を 4.13 に更新しないでください。

重要

OpenShift Container Platform と OpenShift Virtualization の互換性のないバージョンを実行すると、クラスターはサポートされなくなります。

- 仮想マシン上での Descheduler エビクションの有効化はテクニカルプレビュー機能であり、移行の失敗や不安定なスケジューリングを引き起こす可能性があります。
- シングルスタックの IPv6 クラスターで OpenShift Virtualization は実行できません。(BZ#2193267)
- 異なる SELinux コンテキストで 2 つの Pod を使用すると、**ocs-storagecluster-cephfs** ストレージクラスの VM が移行に失敗し、VM のステータスが **Paused** に変わります。これは、両方の Pod が **ReadWriteMany** CephFS の共有ボリュームに同時にアクセスしようとするためです。(BZ#2092271)
    - 回避策として、**ocs-storagecluster-ceph-rbd** ストレージクラスを使用して、Red Hat Ceph Storage を使用するクラスターで VM をライブマイグレーションします。
- **csi-clone** クローンストラテジーを使用して 100 台以上の仮想マシンのクローンを作成する場合、Ceph CSI はクローンをパージしない可能性があります。クローンの手動削除も失敗する可能性があります。(BZ#2055595)
    - 回避策として、**ceph-mgr** を再起動して仮想マシンのクローンをパージすることができます。
- クラスター上のノードを停止し、Node Health Check Operator を使用してノードを再起動すると、Multus への接続が失われる可能性があります。(OCPBUGS-8398)
- OpenShift Virtualization 4.12 では、**TopoLVM** プロビジョナー名の文字列が変更されました。その結果、オペレーティングシステムイメージの自動インポートが失敗し、次のエラーメッセージが表示される場合があります (BZ#2158521)。

```
DataVolume.storage spec is missing accessMode and volumeMode, cannot get access mode from StorageProfile.
```

    - 回避策として、以下を実施します。
        1. ストレージプロファイルの **claimPropertySets** 配列を更新します。

```
$ oc patch storageprofile <storage_profile> --type=merge -p '{"spec":
{"claimPropertySets": [{"accessModes": ["ReadWriteOnce"], "volumeMode": "Block"},
\
    {"accessModes": ["ReadWriteOnce"], "volumeMode": "Filesystem"}]}}'
```

2. **openshift-virtualization-os-images** namespace で影響を受けるデータボリュームを削除します。これらは、更新されたストレージプロファイルのアクセスモードとボリュームモードで再作成されます。

- バインディングモードが **WaitForFirstConsumer** であるストレージの VM スナップショットを復元すると、復元された PVC が **Pending** 状態のままになり、復元操作が進行しません。
  - 回避策として、復元された仮想マシンを起動し、停止してから、再度起動します。仮想マシンがスケジュールされ、PVC が **Bound** 状態になり、復元操作が完了します。(BZ#2149654)

- テンプレートのデフォルトのエビクションストラテジーが **LiveMigrate** であるため、Single Node OpenShift (SNO) クラスター上の共通テンプレートから作成された仮想マシンは **VMCannotBeEvicted** アラートを表示します。仮想マシンのエビクションストラテジーを更新することで、このアラートを無視するか、アラートを削除できます。(BZ#2092412)

- OpenShift Virtualization をアンインストールしても、OpenShift Virtualization によって作成された **feature.node.kubevirt.io** ノードラベルは削除されません。ラベルは手動で削除する必要があります。(CNV-22036)

- Windows 11 仮想マシンは、FIPS モード で実行されているクラスターで起動しません。Windows 11 には、デフォルトで Trusted Platform Module (TPM) デバイスが必要です。ただし、**swtpm** (ソフトウェア TPM エミュレーター) パッケージは FIPS と互換性がありません。(BZ#2089301)

- OpenShift Container Platform クラスターが OVN-Kubernetes をデフォルトの Container Network Interface (CNI) プロバイダーとして使用する場合、OVN-Kubernetes のホストネットワークトポロジーの変更により、Linux ブリッジまたはボンディングデバイスをホストのデフォルトインターフェイスに割り当てることはできません。(BZ#1885605)
  - 回避策として、ホストに接続されたセカンダリーネットワークインターフェイスを使用するか、OpenShift SDN デフォルト CNI プロバイダーに切り替えることができます。

- 場合によっては、複数の仮想マシンが読み取り/書き込みモードで同じ PVC をマウントできるため、データが破損する可能性があります。(BZ#1992753)
  - 回避策として、複数の仮想マシンで読み取り/書き込みモードで単一の PVC を使用しないでください。

- Pod Disruption Budget (PDB) は、移行可能な仮想マシンイメージについての Pod の中断を防ぎます。PDB が Pod の中断を検出する場合、**openshift-monitoring** は **LiveMigrate** エビクションストラテジーを使用する仮想マシンイメージに対して 60 分ごとに **PodDisruptionBudgetAtLimit** アラートを送信します。(BZ#2026733)
  - 回避策として、アラートをサイレント にします。

- OpenShift Virtualization は、Pod によって使用されるサービスアカウントトークンをその特定の Pod にリンクします。OpenShift Virtualization は、トークンが含まれるディスクイメージを作成してサービスアカウントボリュームを実装します。仮想マシンを移行すると、サービスアカウントボリュームが無効になります。(BZ#2037611)
  - 回避策として、サービスアカウントではなくユーザーアカウントを使用してください。ユーザーアカウントトークンは特定の Pod にバインドされていないためです。

- コンピュートノードがさまざま含まれる、異種クラスターでは、HyperV Reenlightenment が有効になっている仮想マシンは、タイムスタンプカウンタースケーリング (TSC) をサポートしていないノード、TSC 頻度が非季節なノードでスケジュールできません。(BZ#2151169)

- Red Hat OpenShift Data Foundation を使用して OpenShift Virtualization をデプロイする場合は、Windows 仮想マシンディスク用の専用ストレージクラスを作成する必要があります。詳細は Windows VM の ODF PersistentVolumes の最適化 を参照してください。

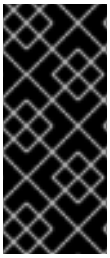
- ブロックストレージデバイスで論理ボリューム管理(LVM)を使用する VM では、Red Hat Enterprise Linux CoreOS (RHCOS)ホストと競合しないように追加の設定が必要になります。

  - 回避策として、仮想マシンの作成、LVM のプロビジョニング、および仮想マシンの再起動を行うことができます。これにより、空の **system.lvmdevices** ファイルが作成されます。(OCPBUGS-5223)

第6章 インストール

6.1. OPENSHIFT VIRTUALIZATION のクラスターの準備

OpenShift Virtualization をインストールする前にこのセクションを確認して、クラスターが要件を満たしていることを確認してください。



重要

ユーザープロビジョニング、インストーラープロビジョニング、またはアシステッドインストーラーなど、任意のインストール方法を使用して、OpenShift Container Platform をデプロイできます。ただし、インストール方法とクラスタートポロジーは、スナップショットやライブマイグレーションなどの OpenShift Virtualization 機能に影響を与える可能性があります。

IPv6

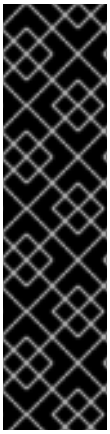
シングルスタックの IPv6 クラスターで OpenShift Virtualization は実行できません。(BZ#2193267)

6.1.1. ハードウェアとオペレーティングシステムの要件

OpenShift Virtualization の次のハードウェアおよびオペレーティングシステム要件を確認してください。

サポート対象プラットフォーム

- オンプレミスのベアメタルサーバー
- Amazon Web Services ベアメタルインスタンス。詳細は、AWS ベアメタルノードへの OpenShift Virtualization のデプロイ を参照してください。
- IBM Cloud Bare Metal Servers。詳細は IBM Cloud Bare Metal Nodes への OpenShift Virtualization のデプロイ を参照してください。



重要

AWS ベアメタルインスタンスまたは IBM Cloud Bare Metal Servers への OpenShift Virtualization のインストールは、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

- **他のクラウドプロバイダーが提供するベアメタルインスタンスまたはサーバーはサポートされていません。**

CPU の要件

- Red Hat Enterprise Linux (RHEL) 9 でサポート

- AMD および Intel 64 ビットアーキテクチャー (x86-64-v2) のサポート
- Intel 64 または AMD64 CPU 拡張機能のサポート
- Intel VT または AMD-V ハードウェア仮想化拡張機能が有効
- NX (実行なし) フラグが有効

ストレージ要件

- OpenShift Container Platform によるサポート

警告

Red Hat OpenShift Data Foundation を使用して OpenShift Virtualization をデプロイする場合は、Windows 仮想マシンディスク用の専用ストレージクラスを作成する必要があります。詳細は 、Windows VM の ODF PersistentVolumes の最適化 を参照してください。

オペレーティングシステム要件

- ワーカーノードにインストールされた Red Hat Enterprise Linux CoreOS (RHCOS)

注記

RHEL ワーカー ノードはサポートされていません。

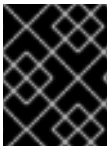
- クラスターが CPU の異なるワーカーノードを使用している場合、CPU ごとに能力が異なるため、ライブマイグレーションの失敗が発生する可能性があります。このような失敗を回避するには、各ノードに適切な能力の CPU を使用し、仮想マシンにノードアフィニティーを設定して、移行が成功するようにします。詳細は、必須のノードのアフィニティールールの設定 を参照してください。

関連情報

- RHCOS について
- サポートされている CPU の Red Hat Ecosystem Catalog
- 対応ストレージ

6.1.2. 物理リソースのオーバーヘッド要件

OpenShift Virtualization は OpenShift Container Platform のアドオンであり、クラスターの計画時に考慮する必要のある追加のオーバーヘッドを強要します。各クラスターマシンは、OpenShift Container Platform の要件に加えて、以下のオーバーヘッドの要件を満たす必要があります。クラスター内の物理リソースを過剰にサブスクライブすると、パフォーマンスに影響する可能性があります。



重要

このドキュメントに記載されている数は、Red Hat のテスト方法およびセットアップに基づいています。これらの数は、独自のセットアップおよび環境に応じて異なります。

6.1.2.1. メモリーのオーバーヘッド

以下の式を使用して、OpenShift Virtualization のメモリーオーバーヘッドの値を計算します。

クラスターメモリーのオーバーヘッド

```
Memory overhead per infrastructure node ≈ 150 MiB
```

```
Memory overhead per worker node ≈ 360 MiB
```

さらに、OpenShift Virtualization 環境リソースには、すべてのインフラストラクチャーノードに分散される合計 2179 MiB の RAM が必要です。

仮想マシンのメモリーオーバーヘッド

```
Memory overhead per virtual machine ≈ (1.002 × requested memory) \
            + 218 MiB \ 1
            + 8 MiB × (number of vCPUs) \ 2
            + 16 MiB × (number of graphics devices) \ 3
            + (additional memory overhead) 4
```

1. **virt-launcher** Pod で実行されるプロセスに必要です。
2. 仮想マシンが要求する仮想 CPU の数。
3. 仮想マシンが要求する仮想グラフィックスカードの数。
4. 追加のメモリーオーバーヘッド:
    - お使いの環境に Single Root I/O Virtualization (SR-IOV) ネットワークデバイスまたは Graphics Processing Unit (GPU) が含まれる場合、それぞれのデバイスに 1 GiB の追加のメモリーオーバーヘッドを割り当てます。

6.1.2.2. CPU オーバーヘッド

以下の式を使用して、OpenShift Virtualization のクラスタープロセッサーのオーバーヘッド要件を計算します。仮想マシンごとの CPU オーバーヘッドは、個々の設定によって異なります。

クラスターの CPU オーバーヘッド

```
CPU overhead for infrastructure nodes ≈ 4 cores
```

OpenShift Virtualization は、ロギング、ルーティング、およびモニタリングなどのクラスターレベルのサービスの全体的な使用率を増加させます。このワークロードに対応するには、インフラストラクチャーコンポーネントをホストするノードに、4 つの追加コア (4000 ミリコア) の容量があり、これがそれらのノード間に分散されていることを確認します。

```
CPU overhead for worker nodes ≈ 2 cores + CPU overhead per virtual machine
```

仮想マシンをホストする各ワーカーノードには、仮想マシンのワークロードに必要な CPU に加えて、OpenShift Virtualization 管理ワークロード用に 2 つの追加コア (2000 ミリコア) の容量が必要です。

仮想マシンの CPU オーバーヘッド

専用の CPU が要求される場合は、仮想マシン 1 台につき CPU 1 つとなり、クラスターの CPU オーバーヘッド要件に影響が出てきます。それ以外の場合は、仮想マシンに必要な CPU の数に関する特別なルールはありません。

6.1.2.3. ストレージのオーバーヘッド

以下のガイドラインを使用して、OpenShift Virtualization 環境のストレージオーバーヘッド要件を見積もります。

クラスターストレージオーバーヘッド

```
Aggregated storage overhead per node ≈ 10 GiB
```

10 GiB は、OpenShift Virtualization のインストール時にクラスター内の各ノードに関するディスク上のストレージの予想される影響に相当します。

仮想マシンのストレージオーバーヘッド

仮想マシンごとのストレージオーバーヘッドは、仮想マシン内のリソース割り当ての特定の要求により異なります。この要求は、クラスター内の別の場所でホストされるノードまたはストレージリソースの一時ストレージに対するものである可能性があります。OpenShift Virtualization は現在、実行中のコンテナー自体に追加の一時ストレージを割り当てていません。

6.1.2.4. 例

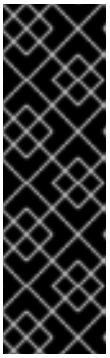
クラスター管理者が、クラスター内の 10 台の (それぞれ 1 GiB の RAM と 2 つの vCPU の) 仮想マシンをホストする予定の場合、クラスター全体で影響を受けるメモリーは 11.68 GiB になります。クラスターの各ノードについて予想されるディスク上のストレージの影響は 10 GiB で示され、仮想マシンのワークロードをホストするワーカーノードについての CPU の影響は最小 2 コアで示されます。

6.1.3. 仮想マシンディスクのストレージボリュームについて

既知のストレージプロバイダーでストレージ API を使用する場合、ボリュームモードとアクセスモードは自動的に選択されます。ただし、ストレージプロファイルのないストレージクラスを使用する場合は、ボリュームとアクセスモードを選択する必要があります。

accessMode: ReadWriteMany と **volumeMode: Block** を使用すると、最適な結果を得ることができます。これは、以下の理由により重要です。

- ライブマイグレーションには ReadWriteMany (RWX) アクセスモードが必要です。
- **Block** ボリュームモードは、**Filesystem** ボリュームモードと比較してパフォーマンスが大幅に向上します。これは、**Filesystem** ボリュームモードでは、ファイルシステムレイヤーやディスクイメージファイルなどを含め、より多くのストレージレイヤーが使用されるためです。仮想マシンのディスクストレージに、これらのレイヤーは必要ありません。
  たとえば、Red Hat OpenShift Data Foundation を使用する場合は、CephFS ボリュームよりも Ceph RBD ボリュームの方が推奨されます。



重要

以下を使用する仮想マシンのライブマイグレーションを行うことはできません。

- ReadWriteOnce (RWO) アクセスモードのストレージボリューム
- GPU などのパススルー機能

それらの仮想マシンの **evictionStrategy** フィールドを **LiveMigrate** に設定しないでください。

関連情報

- OpenShift Container Platform ストレージの共通用語集

6.1.4. オブジェクトの最大値

クラスターを計画するときは、次のテスト済みオブジェクトの最大数を考慮する必要があります。

- OpenShift Container Platform オブジェクトの最大値
- OpenShift Virtualization オブジェクトの最大数

6.1.5. 制限されたネットワーク環境

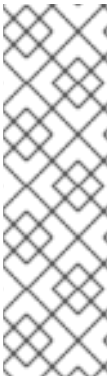
インターネット接続のない制限された環境に OpenShift Virtualization をインストールする場合は、制限されたネットワーク用に Operator Lifecycle Manager を設定 する必要があります。

インターネット接続が制限されている場合、Operator Lifecycle Manager でプロキシーサポートを設定して、Red Hat が提供する OperatorHub にアクセスすることができます。

6.1.6. ライブマイグレーション

ライブマイグレーションには次の要件があります。

- **ReadWriteMany** (RWX) アクセスモードの共有ストレージ
- 十分な RAM およびネットワーク帯域幅
- 仮想マシンがホストモデルの CPU を使用する場合、ノードは仮想マシンのホストモデルの CPU をサポートする必要があります。



注記

ライブマイグレーションを引き起こすノードドレインをサポートするために、クラスター内に十分なメモリーリクエスト容量があることを確認する必要があります。以下の計算を使用して、必要な予備のメモリーを把握できます。

> Product of (Maximum number of nodes that can drain in parallel) and (Highest total VM memory request allocations across nodes)

クラスターで 並行して実行できるデフォルトの移行数 は 5 です。

6.1.7. クラスターの高可用性オプション

クラスターには、次の高可用性 (HA) オプションのいずれかを設定できます。

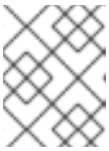
- installer-provisioned infrastructure (IPI) の自動高可用性は、 マシンヘルスチェック をデプロイすることで利用できます。

注記

インストーラーでプロビジョニングされるインフラストラクチャーを使用してインストールされ、MachineHealthCheck が適切に設定された OpenShift Container Platform クラスターで、ノードで MachineHealthCheck が失敗し、クラスターで利用できなくなると、そのノードは再利用されます。障害が発生したノードで実行された仮想マシンでは、一連の条件によって次に起こる動作が変わります。潜在的な結果と RunStrategy がそれらの結果にどのように影響するかについての詳細情報は、仮想マシンの RunStrategy について を参照してください。

- IPI と非 IPI の両方の自動高可用性は、OpenShift Container Platform クラスターで **Node Health Check Operator** を使用して **NodeHealthCheck** コントローラーをデプロイすることで利用できます。コントローラーは異常なノードを特定し、Self Node Remediation Operator や Fence Agents Remediation Operator などの修復プロバイダーを使用して異常なノードを修復します。ノードの修復、フェンシング、メンテナンスの詳細は、Red Hat OpenShift のワークロードの可用性 を参照してください。

- モニタリングシステムまたは有資格者を使用してノードの可用性をモニターすることにより、あらゆるプラットフォームの高可用性を利用できます。ノードが失われた場合は、これをシャットダウンして **oc delete node <lost_node>** を実行します。

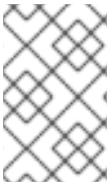


注記

外部モニタリングシステムまたは資格のある人材によるノードの正常性の監視が行われない場合、仮想マシンは高可用性を失います。

6.2. OPENSHIFT VIRTUALIZATION コンポーネントのノードの指定

ノードの配置ルールを設定して、OpenShift Virtualization Operator、ワークロード、およびコントローラーをデプロイするノードを指定します。



注記

OpenShift Virtualization のインストール後に一部のコンポーネントのノードの配置を設定できますが、ワークロード用にノードの配置を設定する場合には仮想マシンを含めることはできません。

6.2.1. 仮想化コンポーネントのノード配置について

OpenShift Virtualization がそのコンポーネントをデプロイする場所をカスタマイズして、以下を確認する必要がある場合があります。

- 仮想マシンは、仮想化ワークロード用のノードにのみデプロイされる。

- Operator はインフラストラクチャーノードにのみデプロイされる。

- 特定のノードは OpenShift Virtualization の影響を受けない。たとえば、クラスターで実行される仮想化に関連しないワークロードがあり、それらのワークロードを OpenShift Virtualization から分離する必要があるとします。

6.2.1.1. ノードの配置ルールを仮想化コンポーネントに適用する方法

対応するオブジェクトを直接編集するか、Web コンソールを使用して、コンポーネントのノードの配置ルールを指定できます。

- Operator Lifecycle Manager (OLM) がデプロイする OpenShift Virtualization Operator の場合は、OLM **Subscription** オブジェクトを直接編集します。現時点では、Web コンソールを使用して **Subscription** オブジェクトのノードの配置ルールを設定することはできません。
- OpenShift Virtualization Operator がデプロイするコンポーネントの場合は、**HyperConverged** オブジェクトを直接編集するか、OpenShift Virtualization のインストール時に Web コンソールを使用してこれを設定します。
- ホストパスプロビジョナーの場合、**HostPathProvisioner** オブジェクトを直接編集するか、Web コンソールを使用してこれを設定します。

警告

ホストパスプロビジョナーと仮想化コンポーネントを同じノードでスケジュールする必要があります。スケジュールしない場合は、ホストパスプロビジョナーを使用する仮想化 Pod を実行できません。

オブジェクトに応じて、以下のルールタイプを 1 つ以上使用できます。

nodeSelector
: Pod は、キーと値のペアまたはこのフィールドで指定したペアを使用してラベルが付けられたノードに Pod をスケジュールできます。ノードには、リスト表示されたすべてのペアに一致するラベルがなければなりません。

affinity
: より表現的な構文を使用して、ノードと Pod に一致するルールを設定できます。アフィニティーを使用すると、ルールの適用方法に追加のニュアンスを持たせることができます。たとえば、ルールがハード要件ではなく基本設定になるように指定し、ルールの条件が満たされない場合も Pod がスケジュールされるようにすることができます。

tolerations
: 一致するテイントを持つノードで Pod をスケジュールできます。テイントがノードに適用される場合、そのノードはテイントを容認する Pod のみを受け入れます。

6.2.1.2. OLM Subscription オブジェクトのノード配置

OLM が OpenShift Virtualization Operator をデプロイするノードを指定するには、OpenShift Virtualization のインストール時に **Subscription** オブジェクトを編集します。以下の例に示されるように、**spec.config** フィールドにノードの配置ルールを追加できます。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
```

```
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.13.11
  channel: "stable"
  config: 1
```

1 **config** フィールドは **nodeSelector** および **tolerations** をサポートしますが、**affinity** はサポートしません。

6.2.1.3. HyperConverged オブジェクトのノード配置

OpenShift Virtualization がそのコンポーネントをデプロイするノードを指定するには、OpenShift Virtualization のインストール時に作成する HyperConverged Cluster カスタムリソース (CR) ファイルに **nodePlacement** オブジェクトを含めることができます。以下の例のように、**spec.infra** および **spec.workloads** フィールドに **nodePlacement** を含めることができます。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  infra:
    nodePlacement: 1
    ...
  workloads:
    nodePlacement:
    ...
```

1 **nodePlacement** フィールドは、**nodeSelector**、**affinity**、および **tolerations** フィールドをサポートします。

6.2.1.4. HostPathProvisioner オブジェクトのノード配置

ノードの配置ルールは、ホストパスプロビジョナーのインストール時に作成する **HostPathProvisioner** オブジェクトの **spec.workload** フィールドで設定できます。

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  pathConfig:
    path: "</path/to/backing/directory>"
    useNamingPrefix: false
  workload: 1
```

1 **workload** フィールドは、**nodeSelector**、**affinity**、および **tolerations** フィールドをサポートします。

6.2.1.5. 関連情報

- 仮想マシンのノードの指定
- ノードセレクターの使用による特定ノードへの Pod の配置
- ノードのアフィニティールールを使用したノード上での Pod 配置の制御
- ノードテイントを使用した Pod 配置の制御
- CLI を使用した OpenShift Virtualization のインストール
- Web コンソールを使用した OpenShift Virtualization のインストール
- 仮想マシンのローカルストレージの設定

6.2.2. マニフェストの例

以下の YAML ファイルの例では、**nodePlacement**、**affinity**、および **tolerations** オブジェクトを使用して OpenShift Virtualization コンポーネントのノード配置をカスタマイズします。

6.2.2.1. Operator Lifecycle Manager サブスクリプションオブジェクト

6.2.2.1.1. 例: OLM Subscription オブジェクトの nodeSelector を使用したノード配置

この例では、OLM が **example.io/example-infra-key = example-infra-value** のラベルが付けられたノードに OpenShift Virtualization Operator を配置するように、**nodeSelector** を設定します。

```
apiVersion: operators.coreos.com/v1beta1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.13.11
  channel: "stable"
  config:
    nodeSelector:
      example.io/example-infra-key: example-infra-value
```

6.2.2.1.2. 例: OLM Subscription オブジェクトの容認を使用したノード配置

この例では、OLM が OpenShift Virtualization Operator をデプロイするために予約されるノードには **key=virtualization:NoSchedule** テイントのラベルが付けられます。一致する容認のある Pod のみがこれらのノードにスケジュールされます。

```
apiVersion: operators.coreos.com/v1beta1
```

```
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source:  redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.13.11
  channel: "stable"
  config:
    tolerations:
    - key: "key"
      operator: "Equal"
      value: "virtualization"
      effect: "NoSchedule"
```

6.2.2.2. HyperConverged オブジェクト

6.2.2.2.1. 例: HyperConverged Cluster CR の nodeSelector を使用したノード配置

この例では、**nodeSelector** は、インフラストラクチャーリソースが **example.io/example-infra-key = example-infra-value** のラベルが付けられたノードに配置されるように設定され、ワークロードは **example.io/example-workloads-key = example-workloads-value** のラベルが付けられたノードに配置されるように設定されます。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  infra:
    nodePlacement:
      nodeSelector:
        example.io/example-infra-key: example-infra-value
  workloads:
    nodePlacement:
      nodeSelector:
        example.io/example-workloads-key: example-workloads-value
```

6.2.2.2.2. 例: HyperConverged Cluster CR のアフィニティーを使用したノード配置

この例では、**affinity** は、インフラストラクチャーリソースが **example.io/example-infra-key = example-infra-value** のラベルが付けられたノードに配置されるように設定され、ワークロードが **example.io/example-workloads-key = example-workloads-value** のラベルが付けられたノードに配置されるように設定されます。ワークロード用には 9 つ以上の CPU を持つノードが優先されますが、それらが利用可能ではない場合も、Pod は依然としてスケジュールされます。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
```

```
spec:
  infra:
    nodePlacement:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-infra-key
                operator: In
                values:
                - example-infra-value
  workloads:
    nodePlacement:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-workloads-key
                operator: In
                values:
                - example-workloads-value
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              - key: example.io/num-cpus
                operator: Gt
                values:
                - 8
```

6.2.2.2.3. 例: HyperConverged Cluster CR の容認を使用したノード配置

この例では、OpenShift Virtualization コンポーネント用に予約されるノードには **key=virtualization:NoSchedule** テイントのラベルが付けられます。一致する容認のある Pod のみがこれらのノードにスケジュールされます。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  workloads:
    nodePlacement:
      tolerations:
      - key: "key"
        operator: "Equal"
        value: "virtualization"
        effect: "NoSchedule"
```

6.2.2.3. HostPathProvisioner オブジェクト

6.2.2.3.1. 例: HostPathProvisioner オブジェクトの nodeSelector を使用したノード配置

この例では、**example.io/example-workloads-key = example-workloads-value** のラベルが付けられたノードにワークロードが配置されるように **nodeSelector** を設定します。

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  pathConfig:
    path: "</path/to/backing/directory>"
    useNamingPrefix: false
  workload:
    nodeSelector:
      example.io/example-workloads-key: example-workloads-value
```

6.3. WEB コンソールを使用した OPENSHIFT VIRTUALIZATION のインストール

OpenShift 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 4.13 Web コンソール を使用して、OpenShift 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。

6.3.1. OpenShift Virtualization Operator のインストール

OpenShift Container Platform Web コンソールから OpenShift Virtualization Operator をインストールできます。

前提条件

- OpenShift Container Platform 4.13 をクラスターにインストールしていること。
- **cluster-admin** パーミッションを持つユーザーとして OpenShift Container Platform Web コンソールにログインすること。

手順

1. **Administrator** パースペクティブから、**Operators → OperatorHub** をクリックします。
2. **Filter by keyword** に **Virtualization** と入力します。
3. **Red Hat** ソースラベルが示されている **OpenShift Virtualization Operator** タイルを選択します。
4. Operator の情報を確認してから、**Install** をクリックします。
5. **Install Operator** ページで以下を行います。
    a. 選択可能な **Update Channel** オプションの一覧から **stable** を選択します。これにより、OpenShift Container Platform バージョンと互換性がある OpenShift Virtualization のバージョンをインストールすることができます。

b. **インストールされた namespace** の場合、**Operator recommended namespace** オプションが選択されていることを確認します。これにより、Operator が必須の **openshift-cnv** namespace にインストールされます。この namespace は存在しない場合は、自動的に作成されます。

警告

OpenShift Virtualization Operator を **openshift-cnv** 以外の namespace にインストールしようとすると、インストールが失敗します。

c. **Approval Strategy** の場合に、**stable** 更新チャネルで新しいバージョンが利用可能になったときに OpenShift Virtualization が自動更新されるように、デフォルト値である **Automatic**を選択することを強く推奨します。
Manual 承認ストラテジーを選択することは可能ですが、クラスターのサポート容易性および機能に対応するリスクが高いため、推奨できません。これらのリスクを完全に理解していて、**Automatic** を使用できない場合のみ、**Manual** を選択してください。

警告

OpenShift Virtualization は対応する OpenShift Container Platform バージョンで使用される場合にのみサポートされるため、OpenShift Virtualization が更新されないと、クラスターがサポートされなくなる可能性があります。

6. **Install** をクリックし、Operator を **openshift-cnv** namespace で利用可能にします。
7. Operator が正常にインストールされたら、**Create HyperConverged** をクリックします。
8. オプション: OpenShift Virtualization コンポーネントの **Infra** および **Workloads** ノード配置オプションを設定します。
9. **Create** をクリックして OpenShift Virtualization を起動します。

検証

- **Workloads → Pods** ページに移動して、OpenShift 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、OpenShift Virtualization を使用できます。

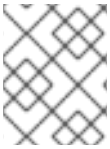
6.3.2. 次のステップ

以下のコンポーネントを追加で設定する必要がある場合があります。

- ホストパスプロビジョナー は、OpenShift Virtualization 用に設計されたローカルストレージプロビジョナーです。仮想マシンのローカルストレージを設定する必要がある場合、まずホストパスプロビジョナーを有効にする必要があります。

6.4. CLI を使用した OPENSHIFT VIRTUALIZATION のインストール

OpenShift 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。コマンドラインを使用してマニフェストをクラスターに適用し、OpenShift Virtualization Operator にサブスクライブし、デプロイできます。



注記

OpenShift Virtualization がそのコンポーネントをインストールするノードを指定するには、ノードの配置ルールを設定 します。

6.4.1. 前提条件

- OpenShift Container Platform 4.13 をクラスターにインストールしていること。
- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

6.4.2. CLI を使用した OpenShift Virtualization カタログのサブスクライブ

OpenShift Virtualization をインストールする前に、OpenShift Virtualization カタログにサブスクライブする必要があります。サブスクライブにより、**openshift-cnv** namespace に OpenShift Virtualization Operator へのアクセスが付与されます。

単一マニフェストをクラスターに適用して **Namespace**、**OperatorGroup**、および **Subscription** オブジェクトをサブスクライブし、設定します。

手順

1. 以下のマニフェストを含む YAML ファイルを作成します。

```
apiVersion: v1
kind: Namespace
metadata:
  name: openshift-cnv
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: kubevirt-hyperconverged-group
  namespace: openshift-cnv
spec:
  targetNamespaces:
    - openshift-cnv
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
```

```
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.13.11
  channel: "stable" (1)
```

(1) **stable** チャネルを使用することで、OpenShift Container Platform バージョンと互換性のある OpenShift Virtualization のバージョンをインストールすることができます。

2. 以下のコマンドを実行して、OpenShift Virtualization に必要な **Namespace**、**OperatorGroup**、および **Subscription** オブジェクトを作成します。

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

注記

YAML ファイルで、証明書のローテーションパラメーターを設定 できます。

6.4.3. CLI を使用した OpenShift Virtualization Operator のデプロイ

oc CLI を使用して OpenShift Virtualization Operator をデプロイすることができます。

前提条件

- **openshift-cnv** namespace の OpenShift Virtualization カタログへのアクティブなサブスクリプション。

手順

1. 以下のマニフェストを含む YAML ファイルを作成します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
```

2. 以下のコマンドを実行して OpenShift Virtualization Operator をデプロイします。

```
$ oc apply -f <file_name>.yaml
```

検証

- **openshift-cnv** namespace の Cluster Service Version (CSV) の **PHASE** を監視して、OpenShift Virtualization が正常にデプロイされたことを確認します。以下のコマンドを実行します。

```
$ watch oc get csv -n openshift-cnv
```

以下の出力は、デプロイメントに成功したかどうかを表示します。

出力例

```
NAME                                      DISPLAY                VERSION   REPLACES   PHASE
kubevirt-hyperconverged-operator.v4.13.11   OpenShift Virtualization   4.13.11
Succeeded
```

6.4.4. 次のステップ

以下のコンポーネントを追加で設定する必要がある場合があります。

- ホストパスプロビジョナー は、OpenShift Virtualization 用に設計されたローカルストレージプロビジョナーです。仮想マシンのローカルストレージを設定する必要がある場合、まずホストパスプロビジョナーを有効にする必要があります。

6.5. OPENSHIFT VIRTUALIZATION のアンインストール

Web コンソールまたはコマンドラインインターフェイス (CLI) を使用して OpenShift Virtualization をアンインストールし、OpenShift Virtualization ワークロード、Operator、およびそのリソースを削除します。

6.5.1. Web コンソールを使用した OpenShift Virtualization のアンインストール

OpenShift Virtualization をアンインストールするには、Web コンソール を使用して次のタスクを実行します。

1. **HyperConverged** CR を削除 します。
2. OpenShift Virtualization Operator を削除 します。
3. **openshift-cnv** namespace を削除 します。
4. OpenShift Virtualization カスタムリソース定義 (CRD) を削除 します。

重要

まず、すべての 仮想マシン と 仮想マシンインスタンス を削除する必要があります。

ワークロードがクラスターに残っている間は、OpenShift Virtualization をアンインストールできません。

6.5.1.1. HyperConverged カスタムリソースの削除

OpenShift Virtualization をアンインストールするには、最初に **HyperConverged** カスタムリソース (CR) を削除します。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。

手順

1. **Operators → Installed Operators** ページに移動します。

2. OpenShift Virtualization Operator を選択します。
3. **OpenShift Virtualization Deployment** タブをクリックします。
4. **kubevirt-hyperconverged** の横にある Options メニュー をクリックし、**Delete HyperConverged** を選択します。
5. 確認ウィンドウで **Delete** をクリックします。

6.5.1.2. Web コンソールの使用によるクラスターからの Operator の削除

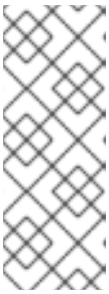
クラスター管理者は Web コンソールを使用して、選択した namespace からインストールされた Operator を削除できます。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスター Web コンソールにアクセスできる。

手順

1. **Operators → Installed Operators** ページに移動します。
2. スクロールするか、キーワードを **Filter by name** フィールドに入力して、削除する Operator を見つけます。次に、それをクリックします。
3. **Operator Details** ページの右側で、**Actions** 一覧から **Uninstall Operator** を選択します。**Uninstall Operator?** ダイアログボックスが表示されます。
4. **Uninstall** を選択し、Operator、Operator デプロイメント、および Pod を削除します。このアクションの後には、Operator は実行を停止し、更新を受信しなくなります。



注記

このアクションは、カスタムリソース定義 (CRD) およびカスタムリソース (CR) など、Operator が管理するリソースは削除されません。Web コンソールおよび継続して実行されるクラスター外のリソースによって有効にされるダッシュボードおよびナビゲーションアイテムには、手動でのクリーンアップが必要になる場合があります。Operator のアンインストール後にこれらを削除するには、Operator CRD を手動で削除する必要があります。

6.5.1.3. 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。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。

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。
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。

6.5.1.4. OpenShift Virtualization カスタムリソース定義の削除

Web コンソールを使用して、OpenShift Virtualization カスタムリソース定義 (CRD) を削除できます。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。

手順

1. **Administration** → **CustomResourceDefinitions** に移動します。
2. **Label** フィルターを選択し、**Search** フィールドに **operators.coreos.com/kubevirt-hyperconverged.openshift-cnv** と入力して OpenShift Virtualization CRD を表示します。
3. 各 CRD の横にある Options メニュー  をクリックし、**Delete CustomResourceDefinition** の削除を選択します。

6.5.2. CLI を使用した OpenShift Virtualization のアンインストール

OpenShift CLI (**oc**) を使用して OpenShift Virtualization をアンインストールできます。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- すべての仮想マシンと仮想マシンインスタンスを削除した。ワークロードがクラスターに残っている間は、OpenShift Virtualization をアンインストールできません。

手順

1. **HyperConverged** カスタムリソースを削除します。

```
$ oc delete HyperConverged kubevirt-hyperconverged -n openshift-cnv
```

2. OpenShift Virtualization Operator サブスクリプションを削除します。

```
$ oc delete subscription kubevirt-hyperconverged -n openshift-cnv
```

3. OpenShift Virtualization **ClusterServiceVersion** リソースを削除します。

```
$ oc delete csv -n openshift-cnv -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv
```

4. OpenShift Virtualization namespace を削除します。

```
$ oc delete namespace openshift-cnv
```

5. **dry-run** オプションを指定して **oc delete crd** コマンドを実行し、OpenShift Virtualization カスタムリソース定義 (CRD) を一覧表示します。

```
$ oc delete crd --dry-run=client -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv
```

出力例

```
customresourcedefinition.apiextensions.k8s.io "cdis.cdi.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "hostpathprovisioners.hostpathprovisioner.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "hyperconvergeds.hco.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "kubevirts.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "networkaddonsconfigs.networkaddonsoperator.network.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "ssps.ssp.kubevirt.io" deleted (dry run)
customresourcedefinition.apiextensions.k8s.io "tektontasks.tektontasks.kubevirt.io" deleted (dry run)
```

6. **dry-run** オプションを指定せずに **oc delete crd** コマンドを実行して、CRD を削除します。

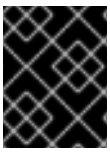
```
$ oc delete crd -l operators.coreos.com/kubevirt-hyperconverged.openshift-cnv
```

関連情報

- 仮想マシンの削除
- 仮想マシンインスタンスの削除

第7章 OPENSHIFT VIRTUALIZATION の更新

Operator Lifecycle Manager(OLM) が OpenShift Virtualization の z-stream およびマイナーバージョンの更新を提供する方法を確認します。



重要

OpenShift Virtualization 4.12.2 から OpenShift Virtualization 4.13 への更新はサポート対象外です。

7.1. RHEL 9 上の OPENSHIFT VIRTUALIZATION

OpenShift Virtualization 4.13 は、Red Hat Enterprise Linux (RHEL) 9 をベースにしています。標準の OpenShift Virtualization 更新手順に従って、RHEL 8 をベースとするバージョンから OpenShift Virtualization 4.13 に更新できます。追加の手順は必要ありません。

以前のバージョンと同様に、実行中のワークロードを中断することなく更新を実行できます。OpenShift Virtualization 4.13 では、RHEL 8 ノードから RHEL 9 ノードへのライブマイグレーションがサポートされています。

7.1.1. 新しい RHEL 9 マシンタイプ

この更新では、仮想マシンの新しい RHEL 9 マシンタイプである **machineType: pc-q35-rhel9.2.0** も導入されています。OpenShift Virtualization に含まれるすべての仮想マシンテンプレートは、デフォルトでこのマシンタイプを使用するようになりました。

OpenShift Virtualization を更新しても、既存仮想マシンの **machineType** 値は変更されません。これらの仮想マシンは、引き続き更新前と同様に機能します。

必須ではありませんが、仮想マシンのマシンタイプを **pc-q35-rhel9.2.0** に変更すると、RHEL 9 の改善点を活用できます。

重要

仮想マシンの **machineType** 値を変更する前に、仮想マシンをシャットダウンする必要があります。

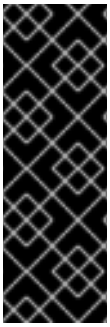
7.2. OPENSHIFT VIRTUALIZATION の更新について

- Operator Lifecycle Manager(OLM) は OpenShift Virtualization Operator のライフサイクルを管理します。OpenShift Container Platform のインストール時にデプロイされる Marketplace Operator により、クラスターで外部 Operator が利用できるようになります。
- OLM は、OpenShift Virtualization の z-stream およびマイナーバージョンの更新を提供します。OpenShift Container Platform を次のマイナーバージョンに更新すると、マイナーバージョンの更新が利用可能になります。OpenShift Container Platform を最初に更新しない限り、OpenShift Virtualization を次のマイナーバージョンに更新できません。
- OpenShift Virtualization サブスクリプションは、**stable** という名前の単一の更新チャネルを使用します。**stable** チャネルでは、OpenShift Virtualization および OpenShift Container Platform バージョンとの互換性が確保されます。
- サブスクリプションの承認ストラテジーが **Automatic** に設定されている場合に、更新プロセスは、Operator の新規バージョンが **stable** チャネルで利用可能になるとすぐに開始します。サ

ポート可能な環境を確保するために、**自動** 承認ストラテジーを使用することを強く推奨します。OpenShift Virtualization の各マイナーバージョンは、対応する OpenShift Container Platform バージョンを実行する場合にのみサポートされます。たとえば、OpenShift Virtualization 4.13 は OpenShift Container Platform 4.13 で実行する必要があります。

  - クラスターのサポート容易性および機能が損なわれるリスクがあるので、**Manual** 承認ストラテジーを選択することは可能ですが、推奨していません。**Manual** 承認ストラテジーでは、保留中のすべての更新を手動で承認する必要があります。OpenShift Container Platform および OpenShift Virtualization の更新の同期が取れていない場合には、クラスターはサポートされなくなります。

- 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。
- OpenShift Virtualization を更新しても、ネットワーク接続が中断されることはありません。
- データボリュームおよびその関連付けられた永続ボリューム要求は更新時に保持されます。



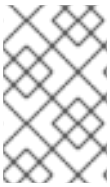
重要

ホストパスプロビジョナーストレージを使用する仮想マシンを実行している場合、それらをライブマイグレーションすることはできず、OpenShift Container Platform クラスターの更新をブロックする可能性があります。

回避策として、仮想マシンを再設定し、クラスターの更新時にそれらの電源を自動的にオフになるようにできます。**evictionStrategy: LiveMigrate** フィールドを削除し、**runStrategy** フィールドを **Always** に設定します。

7.2.1. ワークロードの更新について

OpenShift Virtualization を更新すると、ライブマイグレーションをサポートしている場合には **libvirt**、**virt-launcher**、および **qemu** などの仮想マシンのワークロードが自動的に更新されます。



注記

各仮想マシンには、仮想マシンインスタンス (VMI) を実行する **virt-launcher** Pod があります。**virt-launcher** Pod は、仮想マシン (VM) 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。

HyperConverged カスタムリソース (CR) の **spec.workloadUpdateStrategy** スタンザを編集して、ワークロードの更新方法を設定できます。ワークロードの更新方法として、**LiveMigrate** と **Evict** の 2 つが利用可能です。

Evictメソッドは VMI Pod をシャットダウンするため、デフォルトでは**LiveMigrate**更新ストラテジーのみが有効になっています。

LiveMigrateが有効な唯一の更新ストラテジーである場合:

- ライブマイグレーションをサポートする VMI は更新プロセス時に移行されます。VM ゲストは、更新されたコンポーネントが有効になっている新しい Pod に移動します。
- ライブマイグレーションをサポートしない VMI は中断または更新されません。
  - VMI に**LiveMigrate**エビクションストラテジーがあるが、ライブマイグレーションをサポートしていない場合、VMI は更新されません。

LiveMigrateと**Evict**の両方を有効にした場合:

- ライブマイグレーションをサポートする VMI は、**LiveMigrate** 更新ストラテジーを使用します。
- ライブマイグレーションをサポートしない VMI は、 **Evict**更新ストラテジーを使用します。VMI が、**runStrategy**の値が**always**である**VirtualMachine**オブジェクトによって制御されている場合、新しい VMI は、コンポーネントが更新された新しい Pod に作成されます。

移行の試行とタイムアウト
ワークロードを更新するときに、Pod が次の期間**Pending**状態の場合、ライブマイグレーションは失敗します。

5 分間

Pod が**Unschedulable**であるために保留中の場合。

15 分

何らかの理由で Pod が保留状態のままになっている場合。

VMI が移行に失敗すると、**virt-controller** は VMI の移行を再試行します。すべての移行可能な VMI が新しい**virt-launcher** Pod で実行されるまで、このプロセスが繰り返されます。ただし、VMI が不適切に設定されている場合、これらの試行は無限に繰り返される可能性があります。

注記

各試行は、移行オブジェクトに対応します。直近の 5 回の試行のみがバッファーに保持されます。これにより、デバッグ用の情報を保持しながら、移行オブジェクトがシステムに蓄積されるのを防ぎます。

7.2.2. EUS から EUS への更新について

4.10 および 4.12 を含む OpenShift Container Platform の偶数番号のマイナーバージョンはすべて、Extended Update Support (EUS) バージョンです。ただし、Kubernetes の設計ではシリアルマイナーバージョンの更新が義務付けられているため、ある EUS バージョンから次の EUS バージョンに直接更新することはできません。

ソース EUS バージョンから次の奇数番号のマイナーバージョンに更新した後、更新パス上にあるそのマイナーバージョンのすべての z-stream リリースに OpenShift Virtualization を順次更新する必要があります。適用可能な最新の z-stream バージョンにアップグレードしたら、OpenShift Container Platform をターゲットの EUS マイナーバージョンに更新できます。

OpenShift Container Platform の更新が成功すると、対応する OpenShift Virtualization の更新が利用可能になります。OpenShift Virtualization をターゲットの EUS バージョンに更新できるようになりました。

7.2.2.1. 更新の準備中

EUS から EUS への更新を開始する前に、次のことを行う必要があります。

- EUS から EUS への更新を開始する前に、ワーカーノードのマシン設定プールを一時停止して、ワーカーが 2 回再起動されないようにします。
- 更新プロセスを開始する前に、ワークロードの自動更新を無効にします。これは、ターゲットの EUS バージョンに更新するまで、OpenShift Virtualization が仮想マシン (VM) を移行または削除しないようにするためです。

注記

デフォルトでは、OpenShift Virtualization Operator を更新すると、OpenShift Virtualization は **virt-launcher** Pod などのワークロードを自動的に更新します。この動作は、**HyperConverged** カスタムリソースの **spec.workloadUpdateStrategy** スタンザで設定できます。

EUS から EUS への更新を実行するための準備 の詳細を参照してください。

7.3. EUS から EUS への更新中のワークロード更新の防止

ある Extended Update Support (EUS) バージョンから次のバージョンに更新する場合、自動ワークロード更新を手動で無効にして、更新プロセス中に OpenShift Virtualization がワークロードを移行または削除しないようにする必要があります。

前提条件

- OpenShift Container Platform の EUS バージョンを実行中で、次の EUS バージョンに更新する予定である。その間、奇数番号のバージョンにまだ更新していません。
- 「EUS から EUS への更新を実行するための準備」を読み、OpenShift Container Platform クラスターに関連する警告と要件を学習しました。
- OpenShift Container Platform ドキュメントの指示に従って、ワーカーノードのマシン設定プールを一時停止しました。
- デフォルトの **Automatic** 承認ストラテジーを使用することを推奨します。**Manual** 承認ストラテジーを使用する場合は、Web コンソールで保留中のすべての更新を承認する必要があります。詳細は、「保留中の Operator の更新を手動で承認する」セクションを参照してください。

手順

1. 次のコマンドを実行して、現在の **workloadUpdateMethods** 設定をバックアップします。

```
$ WORKLOAD_UPDATE_METHODS=$(oc get kv kubevirt-kubevirt-hyperconverged -n openshift-cnv -o jsonpath='{.spec.workloadUpdateStrategy.workloadUpdateMethods}')
```

2. 次のコマンドを実行して、すべてのワークロード更新方法をオフにします。

```
$ oc patch hco kubevirt-hyperconverged -n openshift-cnv --type json -p '[{"op":"replace","path":"/spec/workloadUpdateStrategy/workloadUpdateMethods", "value":[]}]'
```

出力例

```
hyperconverged.hco.kubevirt.io/kubevirt-hyperconverged patched
```

3. 続行する前に、**HyperConverged** Operator が **アップグレード可能** であることを確認してください。次のコマンドを入力して、出力をモニターします。

```
$ oc get hco kubevirt-hyperconverged -n openshift-cnv -o json | jq ".status.conditions"
```

例7.1 出力例

```
[
  {
    "lastTransitionTime": "2022-12-09T16:29:11Z",
    "message": "Reconcile completed successfully",
    "observedGeneration": 3,
    "reason": "ReconcileCompleted",
    "status": "True",
    "type": "ReconcileComplete"
  },
  {
    "lastTransitionTime": "2022-12-09T20:30:10Z",
    "message": "Reconcile completed successfully",
    "observedGeneration": 3,
    "reason": "ReconcileCompleted",
    "status": "True",
    "type": "Available"
  },
  {
    "lastTransitionTime": "2022-12-09T20:30:10Z",
    "message": "Reconcile completed successfully",
    "observedGeneration": 3,
    "reason": "ReconcileCompleted",
    "status": "False",
    "type": "Progressing"
  },
  {
    "lastTransitionTime": "2022-12-09T16:39:11Z",
    "message": "Reconcile completed successfully",
    "observedGeneration": 3,
    "reason": "ReconcileCompleted",
    "status": "False",
    "type": "Degraded"
  },
  {
    "lastTransitionTime": "2022-12-09T20:30:10Z",
    "message": "Reconcile completed successfully",
    "observedGeneration": 3,
    "reason": "ReconcileCompleted",
    "status": "True",
    "type": "Upgradeable" (1)
  }
]
```


(1) OpenShift Virtualization Operator のステータスは **Upgradeable** です。

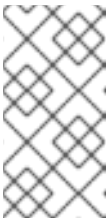
4. クラスターをソース EUS バージョンから OpenShift Container Platform の次のマイナーバージョンに手動で更新します。

```
$ oc adm upgrade
```

検証

- 次のコマンドを実行して、現在のバージョンを確認します。

```
$ oc get clusterversion
```



注記

OpenShift Container Platform を次のバージョンに更新することは、OpenShift Virtualization を更新するための前提条件です。詳細は、OpenShift Container Platform ドキュメントの「クラスターの更新」セクションを参照してください。

5. OpenShift Virtualization を更新します。
   - デフォルトの **自動** 承認ストラテジーでは、OpenShift Container Platform を更新した後、OpenShift Virtualization は対応するバージョンに自動的に更新します。
   - **手動** 承認ストラテジーを使用する場合は、Web コンソールを使用して保留中の更新を承認します。
6. 次のコマンドを実行して、OpenShift Virtualization の更新をモニターします。

```
$ oc get csv -n openshift-cnv
```

7. OpenShift Virtualization を非 EUS マイナーバージョンで使用可能なすべての z-stream バージョンに更新し、前の手順で示したコマンドを実行して各更新を監視します。
8. 以下のコマンドを実行して、OpenShift Virtualization が非 EUS バージョンの最新の z-stream リリースに正常に更新されたことを確認します。

```
$ oc get hco kubevirt-hyperconverged -n openshift-cnv -o json | jq ".status.versions"
```

出力例

```
[
  {
    "name": "operator",
    "version": "4.13.11"
  }
]
```

9. 次の更新を実行する前に、**HyperConverged** Operator が **Upgradeable** ステータスになるまで待ちます。次のコマンドを入力して、出力をモニターします。

```
$ oc get hco kubevirt-hyperconverged -n openshift-cnv -o json | jq ".status.conditions"
```

10. OpenShift Container Platform をターゲットの EUS バージョンに更新します。
11. クラスターのバージョンを確認して、更新が成功したことを確認します。

```
$ oc get clusterversion
```

12. OpenShift Virtualization をターゲットの EUS バージョンに更新します。
    - デフォルトの **自動** 承認ストラテジーでは、OpenShift Container Platform を更新した後、OpenShift Virtualization は対応するバージョンに自動的に更新します。

- **手動** 承認ストラテジーを使用する場合は、Web コンソールを使用して保留中の更新を承認します。

13. 次のコマンドを実行して、OpenShift Virtualization の更新をモニターします。

```
$ oc get csv -n openshift-cnv
```

VERSION フィールドがターゲットの EUS バージョンと一致し、**PHASE** フィールドが **Succeeded** になると、更新が完了します。

14. バックアップしたワークロードの更新方法の設定を復元します。

```
$ oc patch hco kubevirt-hyperconverged -n openshift-cnv --type json -p "
[{\"op\":\"add\",\"path\":\"/spec/workloadUpdateStrategy/workloadUpdateMethods\",
\"value\":$WORKLOAD_UPDATE_METHODS}]"
```

出力例

```
hyperconverged.hco.kubevirt.io/kubevirt-hyperconverged patched
```

検証

- 次のコマンドを実行して、VM 移行のステータスを確認します。

```
$ oc get vmim -A
```

次のステップ

- ワーカーノードのマシン設定プールの一時停止を解除できるようになりました。

7.4. ワークロードの更新方法の設定

HyperConverged カスタムリソース (CR) を編集することにより、ワークロードの更新方法を設定できます。

前提条件

- ライブマイグレーションを更新方法として使用するには、まずクラスターでライブマイグレーションを有効にする必要があります。

注記

VirtualMachineInstance CR に **evictionStrategy: LiveMigrate** が含まれており、仮想マシンインスタンス (VMI) がライブマイグレーションをサポートしない場合には、VMI は更新されません。

手順

1. デフォルトエディターで **HyperConverged** CR を作成するには、以下のコマンドを実行します。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

2. **HyperConverged** CR の **workloadUpdateStrategy** スタンザを編集します。以下に例を示します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  workloadUpdateStrategy:
    workloadUpdateMethods: 1
    - LiveMigrate 2
    - Evict 3
    batchEvictionSize: 10 4
    batchEvictionInterval: "1m0s" 5
# ...
```

1. ワークロードの自動更新を実行するのに使用できるメソッド。設定可能な値は **LiveMigrate** および **Evict** です。上記の例のように両方のオプションを有効にした場合に、ライブマイグレーションをサポートする VMI には **LiveMigrate** を、ライブマイグレーションをサポートしない VMI には **Evict** を、更新に使用します。ワークロードの自動更新を無効にするには、**workloadUpdateStrategy**スタンザを削除するか、**workloadUpdateMethods: []** を設定して配列を空のままにします。

2. 中断を最小限に抑えた更新メソッド。ライブマイグレーションをサポートする VMI は、仮想マシン (VM) ゲストを更新されたコンポーネントが有効なっている新規 Pod に移行することで更新されます。**LiveMigrate** がリストされている唯一のワークロード更新メソッドである場合には、ライブマイグレーションをサポートしない VMI は中断または更新されません。

3. アップグレード時に VMI Pod をシャットダウンする破壊的な方法。**Evict** は、ライブマイグレーションがクラスターで有効でない場合に利用可能な唯一の更新方法です。VMI が **runStrategy: always** に設定された **VirtualMachine** オブジェクトによって制御される場合には、新規の VMI は、更新されたコンポーネントを使用して新規 Pod に作成されます。

4. **Evict**メソッドを使用して一度に強制的に更新できる VMI の数。これは、**LiveMigrate** メソッドには適用されません。

5. 次のワークロードバッチをエビクトするまで待機する間隔。これは、**LiveMigrate** メソッドには適用されません。

注記

HyperConverged CR の**spec.liveMigrationConfig**スタンザを編集することにより、ライブマイグレーションの制限とタイムアウトを設定できます。

3. 変更を適用するには、エディターを保存し、終了します。

7.5. 保留中の OPERATOR 更新の承認

7.5.1. 保留中の Operator 更新の手動による承認

インストールされた Operator のサブスクリプションの承認ストラテジーが **Manual** に設定されている場合、新規の更新が現在の更新チャネルにリリースされると、インストールを開始する前に更新を手動で承認する必要があります。

前提条件

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

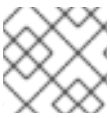
手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動します。
2. 更新が保留中の Operator は **Upgrade available** のステータスを表示します。更新する Operator の名前をクリックします。
3. **Subscription** タブをクリックします。承認が必要な更新は、**Upgrade status** の横に表示されます。たとえば、**1 requires approval** が表示される可能性があります。
4. **1 requires approval** をクリックしてから、**Preview Install Plan** をクリックします。
5. 更新に利用可能なリソースとして一覧表示されているリソースを確認します。問題がなければ、**Approve** をクリックします。
6. **Operators → Installed Operators** ページに戻り、更新の進捗をモニターします。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。

7.6. 更新ステータスの監視

7.6.1. OpenShift Virtualization アップグレードステータスのモニタリング

OpenShift Virtualization Operator のアップグレードのステータスをモニターするには、クラスターサービスバージョン (CSV) **PHASE** を監視します。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クラスターにログインしている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 以下のコマンドを実行します。

```
$ oc get csv -n openshift-cnv
```

2. 出力を確認し、**PHASE** フィールドをチェックします。以下に例を示します。

出力例

```
VERSION  REPLACES                                  PHASE
4.9.0    kubevirt-hyperconverged-operator.v4.8.2   Installing
4.9.0    kubevirt-hyperconverged-operator.v4.9.0   Replacing
```

3. オプション: 以下のコマンドを実行して、すべての OpenShift 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 kubevirt-hyperconverged \
-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。

出力例

```
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
```

7.6.2. 以前の OpenShift Virtualization ワークロードの表示

CLI を使用して、以前のワークロードのリストを表示できます。

注記

クラスターに以前の仮想化 Pod がある場合には、**OutdatedVirtualMachineInstanceWorkloads** アラートが実行されます。

手順

- 以前の仮想マシンインスタンス (VMI) の一覧を表示するには、以下のコマンドを実行します。

```
$ oc get vmi -l kubevirt.io/outdatedLauncherImage --all-namespaces
```

注記

ワークロードの更新を設定して、VMI が自動的に更新されるようにします。

7.7. 関連情報

- EUS から EUS への更新を実行するための準備
- Operator について
- Operator Lifecycle Manager の概念およびリソース
- Cluster Service Version (CSV)
- 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

- 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

第8章 セキュリティーポリシー

OpenShift Virtualization のセキュリティーと認可を説明します。

主なポイント

- OpenShift Virtualization は、Pod セキュリティーの現在のベストプラクティスを強制することを目的とした、**restricted** Kubernetes pod security standards プロファイルに準拠しています。
- 仮想マシン (VM) のワークロードは、特権のない Pod として実行されます。
- Security Context Constraints (SCC) は、**kubevirt-controller** サービスアカウントに対して定義されます。

8.1. ワークロードのセキュリティーについて

デフォルトでは、OpenShift Virtualization の仮想マシン (VM) ワークロードは root 権限では実行されず、root 権限を必要とするサポート対象の OpenShift Virtualization 機能はありません。

仮想マシンごとに、**virt-launcher** Pod が **libvirt** のインスタンスを **セッションモード** で実行し、仮想マシンプロセスを管理します。セッションモードでは、**libvirt** デーモンは root 以外のユーザーアカウントとして実行され、同じユーザー識別子 (UID) で実行されているクライアントからの接続のみを許可します。したがって、仮想マシンは権限のない Pod として実行し、最小権限のセキュリティー原則に従います。

8.2. KUBEVIRT-CONTROLLER サービスアカウントの追加の OPENSHIFT CONTAINER PLATFORM SCC (SECURITY CONTEXT CONSTRAINTS) および LINUX 機能

SCC (Security Context Constraints) は Pod のパーミッションを制御します。これらのパーミッションには、コンテナーのコレクションである Pod が実行できるアクションおよびそれがアクセスできるリソース情報が含まれます。SCC を使用して、Pod がシステムに受け入れられるために必要な Pod の実行に関する条件の一覧を定義できます。

virt-controller は、クラスター内の仮想マシンの **virt-launcher** Pod を作成するクラスターコントローラーです。これらの Pod には、**kubevirt-controller** サービスアカウントによってパーミッションが付与されます。

kubevirt-controller サービスアカウントには追加の SCC および Linux 機能が付与され、これにより適切なパーミッションを持つ **virt-launcher** Pod を作成できます。これらの拡張パーミッションにより、仮想マシンは通常の Pod の範囲外の OpenShift Virtualization 機能を利用できます。

kubevirt-controller サービスアカウントには以下の SCC が付与されます。

- **scc.AllowHostDirVolumePlugin = true**
  これは、仮想マシンが hostpath ボリュームプラグインを使用することを可能にします。
- **scc.AllowPrivilegedContainer = false**
  これは、virt-launcher Pod が権限付きコンテナーとして実行されないようにします。
- **scc.AllowedCapabilities = []corev1.Capability{"SYS_NICE", "NET_BIND_SERVICE"}**
  - **SYS_NICE** を使用すると、CPU アフィニティーを設定できます。
  - **NET_BIND_SERVICE** は、DHCP および Slirp 操作を許可します。

8.2.1. kubevirt-controller の SCC および RBAC 定義の表示

oc ツールを使用して **kubevirt-controller** の **SecurityContextConstraints** 定義を表示できます。

```
$ oc get scc kubevirt-controller -o yaml
```

oc ツールを使用して **kubevirt-controller** クラスターロールの RBAC 定義を表示できます。

```
$ oc get clusterrole kubevirt-controller -o yaml
```

8.3. AUTHORIZATION

OpenShift Virtualization では、ロールベースのアクセス制御 (RBAC) が認可に使用されます。たとえば管理者は、仮想マシンの起動に必要な権限を提供する RBAC ロールを作成できます。その後、管理者はロールを特定のユーザーにバインドすることで、その機能へのアクセスを制限できます。

8.3.1. OpenShift Virtualization のデフォルトのクラスターロール

クラスターロール集約を使用することで、OpenShift Virtualization はデフォルトの OpenShift Container Platform クラスターロールを拡張して、仮想化オブジェクトにアクセスするための権限を組み込みます。

表8.1 OpenShift Virtualization のクラスターロール

デフォルトのクラスターロール	OpenShift Virtualization のクラスターロール	OpenShift Virtualization クラスターロールの説明
view	**kubevirt.io:view**	クラスター内の OpenShift Virtualization リソースをすべて表示できるユーザー。ただし、リソースの作成、削除、変更、アクセスはできません。たとえば、ユーザーは仮想マシン (VM) が実行中であることを確認できますが、それをシャットダウンしたり、そのコンソールにアクセスしたりすることはできません。
edit	**kubevirt.io:edit**	クラスター内のすべての OpenShift Virtualization リソースを変更できるユーザー。たとえば、ユーザーは仮想マシンの作成、VM コンソールへのアクセス、仮想マシンの削除を行えます。
admin	**kubevirt.io:admin**	リソースコレクションの削除を含め、すべての OpenShift Virtualization リソースに対する完全な権限を持つユーザー。このユーザーは、**openshift-cnv** namespace の **HyperConverged** カスタムリソースにある OpenShift Virtualization ランタイム設定も表示および変更できます。

8.4. 関連情報

- Security Context Constraints の管理
- RBAC の使用によるパーミッションの定義および適用
- クラスターロールの作成

- クラスターのロールバインディングコマンド
- 複数の namespace 間でデータボリュームをクローン作成するためのユーザーパーミッションの有効化

第9章 VIRTCTL および LIBGUESTFS CLI ツールの使用

virtctl コマンドラインツールを使用して、OpenShift Virtualization リソースを管理できます。

virtctl を使用して **libguestfs-tools** コンテナーをデプロイすることもできます。**Libguestfs** は、仮想マシン (VM) ディスクイメージにアクセスして変更するためのツールセットです。

9.1. VIRTCTL のインストール

Linux、Windows、および MacOS オペレーティングシステムに **virtctl** をインストールするには、**virtctl** バイナリーファイルをダウンロードしてインストールします。

Red Hat Enterprise Linux (RHEL) に **virtctl** をインストールするには、OpenShift Virtualization リポジトリーを有効にしてから、**kubevirt-virtctl** パッケージをインストールします。

9.1.1. RHEL 9、Linux、Windows、macOS への virtctl バイナリーのインストール

OpenShift Container Platform Web コンソールからオペレーティングシステムの **virtctl** バイナリーをダウンロードし、それをインストールできます。

手順

1. Web コンソールの **Virtualization → Overview** ページに移動します。
2. **Download virtctl** リンクをクリックして、オペレーティングシステム用の **virtctl** バイナリーをダウンロードします。
3. **virtctl** をインストールします。
   - RHEL 9 およびその他の Linux オペレーティングシステムの場合:
     a. アーカイブファイルを解凍します。

     ```
     $ tar -xvf <virtctl-version-distribution.arch>.tar.gz
     ```

     b. 次のコマンドを実行して、**virtctl** バイナリーを実行可能にします。

     ```
     $ chmod +x <path/virtctl-file-name>
     ```

     c. **virtctl** バイナリーを **PATH 環境変数** にあるディレクトリーに移動します。
     次のコマンドを実行して、パスを確認できます。

     ```
     $ echo $PATH
     ```

     d. **KUBECONFIG** 環境変数を設定します。

     ```
     $ export KUBECONFIG=/home/<user>/clusters/current/auth/kubeconfig
     ```

   - Windows の場合:
     a. アーカイブファイルを展開します。
     b. 展開したフォルダー階層に移動し、**virtctl** 実行可能ファイルをダブルクリックしてクライアントをインストールします。

c. **virtctl** バイナリーを **PATH 環境変数** にあるディレクトリーに移動します。
次のコマンドを実行して、パスを確認できます。

```
C:\> path
```

- macOS の場合:

a. アーカイブファイルを展開します。

b. **virtctl** バイナリーを **PATH 環境変数** にあるディレクトリーに移動します。
次のコマンドを実行して、パスを確認できます。

```
echo $PATH
```

9.1.2. RHEL 8 への virtctl RPM のインストール

OpenShift 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することで、Red Hat Enterprise Linux (RHEL) 8 に **virtctl** RPM をインストールできます。

前提条件

- クラスター内の各ホストは Red Hat Subscription Manager (RHSM) に登録されており、アクティブな OpenShift Container Platform サブスクリプションを持つ必要があります。

手順

1. **subscription-manager** CLI ツールを使用して次のコマンドを実行し、お使いのオペレーティングシステムの OpenShift Virtualization リポジトリーを有効にします。

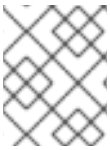
```
# subscription-manager repos --enable cnv-4.13-for-rhel-8-x86_64-rpms
```

2. 次のコマンドを実行して、**kubevirt-virtctl** パッケージをインストールします。

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

9.2. VIRTCTL コマンド

virtctl クライアントは、OpenShift 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。



注記

特に指定がない限り、仮想マシン (VM) コマンドは仮想マシンインスタンスにも適用されます。

9.2.1. virtctl 情報コマンド

virtctl information コマンドを使用して、**virtctl** クライアントに関する情報を表示します。

表9.1 情報コマンド

コマンド	説明
virtctl version	**virtctl** クライアントとサーバーのバージョンを表示します。
virtctl help	**virtctl** コマンドのリストを表示します。
virtctl <command> -h|--help	特定のコマンドのオプションのリストを表示します。
virtctl オプション	任意の **virtctl** コマンドのグローバルコマンドオプションのリストを表示します。

9.2.2. 仮想マシン情報コマンド

virtctl を使用して、仮想マシンおよび VMI に関する情報を表示できます。

表9.2 仮想マシン情報コマンド

コマンド	説明
virtctl fslist <vm_name>	ゲストマシンで使用可能なファイルシステムを表示します。
virtctl guestosinfo <vm_name>	ゲストマシンのオペレーティングシステムに関する情報を表示します。
virtctl userlist <vm_name>	ゲストマシンにログインしているユーザーを表示します。

9.2.3. 仮想マシン管理コマンド

virtctl 仮想マシン (VM) 管理コマンドを使用して、仮想マシンと VMI を管理および移行します。

表9.3 仮想マシン管理コマンド

コマンド	説明
virtctl create -name <vm_name>	**VirtualMachine** マニフェストを作成します。
virtctl start <vm_name>	仮想マシンを開始します。
virtctl start --paused <vm_name>	仮想マシンを一時停止状態で起動します。このオプションを使用すると、VNC コンソールからブートプロセスを中断できます。
virtctl stop <vm_name>	仮想マシンを停止します。
virtctl stop <vm_name> --grace-period 0 --force	仮想マシンを強制停止します。このオプションは、データの不整合またはデータ損失を引き起こす可能性があります。

コマンド	説明
virtctl pause vm <vm_name>	仮想マシンを一時停止します。マシンの状態がメモリーに保持されます。
virtctl unpause vm <vm_name>	仮想マシンの一時停止を解除します。
virtctl migrate <vm_name>	仮想マシンを移行します。
virtctl restart <vm_name>	仮想マシンを再起動します。

9.2.4. 仮想マシン接続コマンド

virtctl 接続コマンドを使用してポートを公開し、仮想マシンおよび VMI に接続します。

表9.4 仮想マシン接続コマンド

コマンド	説明
virtctl console <vm_name>	仮想マシンのシリアルコンソールに接続します。
virtctl expose <vm_name>	仮想マシンの指定されたポートを転送するサービスを作成し、ノードの指定されたポートでサービスを公開します。
virtctl scp -i <ssh_key> <file_name> <user_name>@<vm_name>	マシンから仮想マシンにファイルをコピーします。このコマンドは、SSH キーペアの秘密キーを使用します。仮想マシンは公開キーを使用して設定する必要があります。
virtctl scp -i <ssh_key> <user_name@<vm_name>: <file_name> .	仮想マシンからマシンにファイルをコピーします。このコマンドは、SSH キーペアの秘密キーを使用します。仮想マシンは公開キーを使用して設定する必要があります。
virtctl ssh -i <ssh_key> <user_name>@<vm_name>	仮想マシンとの SSH 接続を開きます。このコマンドは、SSH キーペアの秘密キーを使用します。仮想マシンは公開キーを使用して設定する必要があります。
virtctl vnc --kubeconfig=$KUBECONFIG <vm_name>	仮想マシンの VNC コンソールに接続します。 VNC を介して VM のグラフィカルコンソールにアクセスするには、ローカルマシンにリモートビューアーが必要です。
virtctl vnc --kubeconfig=$KUBECONFIG --proxy-only=true <vm_name>	ポート番号を表示し、VNC 接続を介してビューアーを使用して手動で VM に接続します。

コマンド	説明
virtctl vnc --kubeconfig=$KUBECONFIG --port=<port-number> <vm_name>	ポートが利用可能な場合、その指定されたポートでプロキシーを実行するためにポート番号を指定します。 ポート番号が指定されていない場合、プロキシーはランダムポートで実行されます。

9.2.5. 仮想マシンエクスポートコマンド

virtctl vmexport コマンドを使用して、仮想マシン、仮想マシンスナップショット、または永続ボリューム要求 (PVC) からエクスポートされたボリュームを作成、ダウンロード、または削除できます。

表9.5 仮想マシンエクスポートコマンド

コマンド	説明
virtctl vmexport create <vmexport_name> --vm|snapshot|pvc= <object_name>	仮想マシン、仮想マシンスナップショット、または PVC からボリュームをエクスポートするには、**VirtualMachineExport** カスタムリソース (CR) を作成します。 ● **--vm**: 仮想マシンの PVC をエクスポートします。 ● **--snapshot**: **VirtualMachineSnapshot** CR に含まれる PVC をエクスポートします。 ● **--pvc**: PVC をエクスポートします。 ● オプション: **--ttl=1h** は存続時間を指定します。デフォルトの期間は 2 時間です。
virtctl vmexport delete <vmexport_name>	**VirtualMachineExport** CR を手動で削除します。
virtctl vmexport download <vmexport_name> --output= <output_file> --volume= <volume_name>	**VirtualMachineExport** CR で定義されたボリュームをダウンロードします。 ● **--output** はファイル形式を指定します。例: **disk.img.gz**。 ● **--volume** は、ダウンロードするボリュームを指定します。使用可能なボリュームが 1 つだけの場合、このフラグはオプションです。 オプション: ● **--keep-vme** は、ダウンロード後に **VirtualMachineExport** CR を保持します。デフォルトの動作では、ダウンロード後に **VirtualMachineExport** CR を削除します。 ● **--insecure** は、安全でない HTTP 接続を有効にします。

コマンド	説明
virtctl vmexport download <vmexport_name> --<vm|snapshot|pvc>=<object_name> --output=<output_file> --volume=<volume_name>	**VirtualMachineExport** CR を作成し、CR で定義されたボリュームをダウンロードします。

9.2.6. 仮想マシンメモリーダンプコマンド

virtctl memory-dump コマンドを使用して、PVC に仮想マシンのメモリーダンプを出力できます。既存の PVC を指定するか、**--create-claim** フラグを使用して新しい PVC を作成できます。

前提条件

- PVC ボリュームモードは **FileSystem** である必要があります。
- PVC は、メモリーダンプを格納するのに十分な大きさである必要があります。
  PVC サイズを計算する式は **(VMMemorySize + 100Mi) * FileSystemOverhead** です。ここで、**100Mi** はメモリーダンプのオーバーヘッドです。
- 次のコマンドを実行して、**HyperConverged** カスタムリソースでホットプラグフィーチャーゲートを有効にする必要があります。

```
$ oc patch hco kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "add", "path": "/spec/featureGates", \
  "value": "HotplugVolumes"}]'
```

メモリーダンプのダウンロード

メモリーダンプをダウンロードするには、**virtctl vmexport download** コマンドを使用する必要があります。

```
$ virtctl vmexport download <vmexport_name> --vm\|pvc=<object_name> \
  --volume=<volume_name> --output=<output_file>
```

表9.6 仮想マシンメモリーダンプコマンド

コマンド	説明

コマンド	説明
virtctl memory-dump get <vm_name> --claim-name= <pvc_name>	仮想マシンのメモリーダンプを PVC に保存します。メモリーダンプのステータスは、**VirtualMachine** リソースの **status** セクションに表示されます。 オプション: ● **--create-claim** は、適切なサイズで新しい PVC を作成します。このフラグには次のオプションがあります。 ○ **--storage-class=<storage_class>**: PVC のストレージクラスを指定します。 ○ **--access-mode=<access_mode>**: **ReadWriteOnce** または **ReadWriteMany** を指定します。
virtctl memory-dump get <vm_name>	同じ PVC で **virtctl memory-dump** コマンドを再実行します。 このコマンドは、以前のメモリーダンプを上書きします。
virtctl memory-dump remove <vm_name>	メモリーダンプを削除します。 ターゲット PVC を変更する場合は、メモリーダンプを手動で削除する必要があります。 このコマンドは、**VirtualMachine** リソースの **status** セクションにメモリーダンプが表示されないように、仮想マシンと PVC の間の関連付けを削除します。PVC は影響を受けません。

9.2.7. ホットプラグおよびホットアンプラグコマンド

virtctl を使用して、実行中の VM および VMI にリソースを追加または削除します。

表9.7 ホットプラグおよびホットアンプラグコマンド

コマンド	説明
virtctl addvolume <vm_name> --volume-name= <datavolume_or_PVC> [--persist] [--serial=<label>]	データボリュームまたは永続ボリューム要求 (PVC) をホットプラグします。 オプション: ● **--persist** は仮想ディスクを VM に永続的にマウントします。**このフラグは VMI には適用されません。** ● **--serial=<label>** は仮想マシンにラベルを追加します。ラベルを指定しない場合、デフォルトのラベルはデータボリュームまたは PVC 名になります。
virtctl removevolume <vm_name> --volume-name=<virtual_disk>	仮想ディスクをホットアンプラグします。

9.2.8. イメージアップロードコマンド

virtctl image-upload コマンドを使用して、VM イメージをデータボリュームにアップロードできます。

表9.8 イメージアップロードコマンド

コマンド	説明
virtctl image-upload dv <datavolume_name> --image-path= </path/to/image> --no-create	VM イメージを既存のデータボリュームにアップロードします。
virtctl image-upload dv <datavolume_name> --size= <datavolume_size> --image-path=</path/to/image>	指定された要求されたサイズの新しいデータボリュームに VM イメージをアップロードします。

9.3. LIBGUESTFS の使用

9.3.1. virtctl を使用した libguestfs-tools コンテナーのデプロイ

virtctl guestfs コマンドを使用して、**libguestfs-tools** および永続ボリューム要求 (PVC) がアタッチされた対話型コンテナーをデプロイできます。

手順

- **libguestfs-tools** でコンテナーをデプロイして PVC をマウントし、シェルを割り当てるには、以下のコマンドを実行します。

```
$ virtctl guestfs -n <namespace> <pvc_name> 1
```

1 PVC 名は必須の引数です。この引数を追加しないと、エラーメッセージが表示されます。

9.3.2. Libguestfs および virtctl guestfs コマンド

Libguestfs ツールは、仮想マシン (VM) のディスクイメージにアクセスして変更するのに役立ちます。**libguestfs** ツールを使用して、ゲスト内のファイルの表示および編集、仮想マシンのクローンおよびビルド、およびディスクのフォーマットおよびサイズ変更を実行できます。

virtctl guestfs コマンドおよびそのサブコマンドを使用して、PVC で仮想マシンディスクを変更して検査し、デバッグすることもできます。使用可能なサブコマンドの完全なリストを表示するには、コマンドラインで **virt-** と入力して Tab を押します。以下に例を示します。

コマンド	説明
virt-edit -a /dev/vda /etc/motd	ターミナルでファイルを対話的に編集します。

コマンド	説明
virt-customize -a /dev/vda --ssh-inject root:string:<public key example>	ゲストに ssh キーを挿入し、ログインを作成します。
virt-df -a /dev/vda -h	仮想マシンによって使用されるディスク容量を確認します。
virt-customize -a /dev/vda --run-command 'rpm -qa > /rpm-list'	詳細のリストを含む出力ファイルを作成して、ゲストにインストールされたすべての RPM の詳細リストを参照してください。
virt-cat -a /dev/vda /rpm-list	ターミナルで **virt-customize -a /dev/vda --run-command 'rpm -qa > /rpm-list'** コマンドを使用して作成されたすべての RPM の出力ファイルのリストを表示します。
virt-sysprep -a /dev/vda	テンプレートとして使用する仮想マシンディスクイメージをシールします。

デフォルトでは、**virtctl guestfs** は、仮想ディスク管理に必要な項目を含めてセッションを作成します。ただし、動作をカスタマイズできるように、コマンドは複数のフラグオプションもサポートしています。

フラグオプション	説明
--h または **--help**	**guestfs** のヘルプを提供します。
-n <namespace> オプションと **<pvc_name>** 引数	特定の namespace から PVC を使用します。 **-n <namespace>** オプションを使用しない場合には、現在のプロジェクトが使用されます。プロジェクトを変更するには、**oc project <namespace>** を使用します。 **<pvc_name>** 引数を追加しないと、エラーメッセージが表示されます。
--image string	**libguestfs-tools** コンテナーイメージをリスト表示します。 **--image** オプションを使用して、コンテナーがカスタムイメージを使用するように設定できます。

フラグオプション	説明
--kvm	**kvm** が **libguestfs-tools** コンテナーによって使用されることを示します。 デフォルトでは、**virtctl guestfs** はインタラクティブなコンテナー向けに **kvm** を設定します。これは、QEMU を使用するため、**libguest-tools** の実行が大幅に加速されます。 クラスターに **kvm** をサポートするノードがない場合は、オプション **--kvm=false** を設定して **kvm** を無効にする必要があります。 設定されていない場合、**libguestfs-tools** Pod はいずれのノードにもスケジュールできないため保留状態のままになります。
--pull-policy string	**libguestfs** イメージのプルポリシーを表示します。 **pull-policy** オプションを設定してイメージのプルポリシーを上書きすることもできます。

このコマンドは、PVC が別の Pod によって使用されているかどうかを確認します。使用されている場合には、エラーメッセージが表示されます。ただし、**libguestfs-tools** プロセスが開始されると、設定では同じ PVC を使用する新規 Pod を回避できません。同じ PVC にアクセスする仮想マシンを起動する前に、アクティブな **virtctl guestfs** Pod がないことを確認する必要があります。

注記

virtctl guestfs コマンドは、インタラクティブな Pod に割り当てられている PVC 1 つだけを受け入れます。

第10章 仮想マシン

10.1. 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。

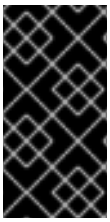
- クイックスタートのガイド付きツアー
- **カタログ**からクイック作成
- 仮想マシンウィザードによる事前に設定された YAML ファイルの貼り付け
- CLI の使用

警告

openshift-* namespace に仮想マシンを作成しないでください。代わりに、**openshift** 接頭辞なしの新規 namespace を作成するか、既存 namespace を使用します。

Web コンソールから仮想マシンを作成する場合、ブートソースで設定される仮想マシンテンプレートを選択します。ブートソースを含む仮想マシンテンプレートには **Available boot source** というラベルが付けられるか、それらはカスタマイズされたラベルテキストを表示します。選択可能なブートソースでテンプレートを使用すると、仮想マシンの作成プロセスをスピードアップできます。

ブートソースのないテンプレートには、**Boot source required** というラベルが付けられます。ブートソースを仮想マシンに追加する 手順を実行する場合、これらのテンプレートを使用できます。



重要

ストレージの動作の違いにより、一部の仮想マシンテンプレートは単一ノードの Openshift と互換性がありません。互換性を確保するためには、テンプレートまたはデータボリュームまたはストレージプロファイルを使用する仮想マシンに**evictionStrategy**フィールドを設定しないでください。

10.1.1. クイックスタートの使用による仮想マシンの作成

Web コンソールは、仮想マシンを作成するためのガイド付きツアーを含むクイックスタートを提供します。**Administrator** パースペクティブの Help メニューを選択して Quick Starts カタログにアクセスし、Quick Starts カタログを表示できます。Quick Starts タイルをクリックし、ツアーを開始すると、システムによるプロセスのガイドが開始します。

Quick Starts のタスクは、Red Hat テンプレートの選択から開始します。次に、ブートソースを追加して、オペレーティングシステムイメージをインポートできます。最後に、カスタムテンプレートを保存し、これを使用して仮想マシンを作成できます。

前提条件

- オペレーティングシステムイメージの URL リンクをダウンロードできる Web サイトにアクセスすること。

手順

1. Web コンソールで、Help メニューから **Quick Starts** を選択します。
2. Quick Starts カタログのタイルをクリックします。例: **Red Hat Linux Enterprise Linux 仮想マシンの作成**
3. ガイド付きツアーの手順に従い、オペレーティングシステムイメージのインポートと仮想マシンの作成タスクを実行します。**Virtualization → VirtualMachines** ページに仮想マシンが表示されます。

10.1.2. 仮想マシンのクイック作成

使用可能なブートソースを含むテンプレートを使用して、仮想マシン (VM) をすばやく作成できます。

手順

1. サイドメニューの **Virtualization → Catalog** をクリックします。
2. **利用可能なブートソース** をクリックして、テンプレートをブートソースでフィルタリングします。

注記

デフォルトでは、テンプレートリストには **Default Templates** のみが表示されます。選択したフィルターで使用可能なすべてのテンプレートを表示するには、フィルタリング時に **すべてのアイテム**をクリックします。

3. テンプレートをクリックして詳細を表示します。
4. 仮想マシンの**クイック作成**をクリックして、テンプレートから VM を作成します。
仮想マシンの**詳細**ページに、プロビジョニングステータスが表示されます。

検証

1. **Events** をクリックして、仮想マシンがプロビジョニングされたときにイベントのストリームを表示します。
2. **Console** をクリックして、仮想マシンが正常に起動したことを確認します。

10.1.3. カスタマイズされたテンプレートからの仮想マシンの作成

一部のテンプレートでは、追加のパラメーターが必要です。たとえば、ブート ソースを持つ PVC などです。テンプレートの選択パラメーターをカスタマイズして、仮想マシン (VM) を作成できます。

手順

1. Web コンソールで、テンプレートを選択します。
    a. サイドメニューの **Virtualization → Catalog** をクリックします。

b. オプション: プロジェクト、キーワード、オペレーティングシステム、またはワークロードプロファイルでテンプレートをフィルター処理します。

c. カスタマイズするテンプレートをクリックします。

2. **Customize VirtualMachine**を クリックします。

3. **Name** や **Disk source** など、仮想マシンのパラメーターを指定します。オプションで、複製するデータソースを指定できます。

検証

1. **Events** をクリックして、仮想マシンがプロビジョニングされたときにイベントのストリームを表示します。

2. **Console** をクリックして、仮想マシンが正常に起動したことを確認します。

Web コンソールから仮想マシンを作成する場合は、仮想マシンのフィールドセクションを参照してください。

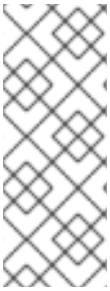
10.1.3.1. ネットワークフィールド

Name	説明
Name	ネットワークインターフェイスコントローラーの名前。
モデル	ネットワークインターフェイスコントローラーのモデルを示します。サポートされる値は **e1000e** および **virtio** です。
Network	利用可能なネットワーク接続定義のリスト。
型	利用可能なバインディングメソッドの一覧。ネットワークインターフェイスに適したバインド方法を選択します。 ● デフォルトの Pod ネットワーク: **masquerade** ● Linux ブリッジネットワーク: **bridge** ● SR-IOV ネットワーク: **SR-IOV**
MAC Address	ネットワークインターフェイスコントローラーの MAC アドレス。MAC アドレスが指定されていない場合、これは自動的に割り当てられます。

10.1.3.2. ストレージフィールド

Name	選択	Description
Source	空白 (PVC の作成)	空のディスクを作成します。
	URL を使用したインポート (PVC の作成)	URL (HTTP または HTTPS エンドポイント) を介してコンテンツをインポートします。
	既存 PVC の使用	クラスターですでに利用可能な PVC を使用します。
	既存の PVC のクローン作成 (PVC の作成)	クラスターで利用可能な既存の PVC を選択し、このクローンを作成します。
	レジストリーを使用したインポート (PVC の作成)	コンテナーレジストリーを使用してコンテンツをインポートします。
	コンテナー (一時的)	クラスターからアクセスできるレジストリーにあるコンテナーからコンテンツをアップロードします。コンテナーディスクは、CD-ROM や一時的な仮想マシンなどの読み取り専用ファイルシステムにのみ使用する必要があります。
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		ディスクのサイズ (GiB 単位)。
型		ディスクのタイプ。例: Disk または CD-ROM
Interface		ディスクデバイスのタイプ。サポートされるインターフェイスは、**virtIO**、**SATA**、および **SCSI** です。
Storage Class		ディスクの作成に使用されるストレージクラス。

ストレージの詳細設定
以下のストレージの詳細設定はオプションであり、**Blank**、**Import via URLURL**、および **Clone existing PVC** ディスクで利用できます。OpenShift Virtualization 4.11 より前では、これらのパラメーターを指定しない場合、システムは **kubevirt-storage-class-defaults** 設定マップのデフォルト値を使用します。OpenShift Virtualization 4.11 以降では、システムは ストレージプロファイル のデフォルト値を使用します。



注記

ストレージプロファイルを使用して、OpenShift Virtualization のストレージをプロビジョニングするときに一貫した高度なストレージ設定を確保します。

Volume Mode と **Access Mode** を手動で指定するには、デフォルトで選択されている **Apply optimized StorageProfile settings** チェックボックスをオフにする必要があります。

Name	モードの説明	パラメーター	パラメーターの説明
ボリュームモード	永続ボリュームがフォーマットされたファイルシステムまたは raw ブロック状態を使用するかどうかを定義します。デフォルトは **Filesystem** です。	Filesystem	ファイルシステムベースのボリュームで仮想ディスクを保存します。
		Block	ブロックボリュームで仮想ディスクを直接保存します。基礎となるストレージがサポートしている場合は、**Block** を使用します。
アクセスモード	永続ボリュームのアクセスモード。	ReadWriteOnce (RWO)	ボリュームはシングルノードで読み取り/書き込みとしてマウントできます。
		ReadWriteMany (RWX)	ボリュームは、一度に多くのノードで読み取り/書き込みとしてマウントできます。  **注記** これは、ノード間の仮想マシンのライブマイグレーションなどの、一部の機能で必要になります。
		ReadOnlyMany (ROX)	ボリュームは数多くのノードで読み取り専用としてマウントできます。

10.1.3.3. Cloud-init フィールド

Name	Description
認可された SSH キー	仮想マシンの **~/.ssh/authorized_keys** にコピーされるユーザーの公開鍵。
カスタムスクリプト	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。

ストレージクラスのデフォルトを設定するには、ストレージプロファイルを使用します。詳細については、ストレージプロファイルのカスタマイズを参照してください。

10.1.3.4. 仮想マシンウィザードの作成用の事前に設定された YAML ファイルの貼り付け

YAML 設定ファイルを作成し、解析し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 **Virtualization → VirtualMachines** をクリックします。
2. **Create** をクリックし、**With 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、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。

仮想マシンが **VirtualMachines** ページにリスト表示されます。

10.1.4. CLI の使用による仮想マシンの作成

virtualMachine マニフェストから仮想マシンを作成できます。

手順

1. 仮想マシンの **VirtualMachine** マニフェストを編集します。たとえば、次のマニフェストは Red Hat Enterprise Linux (RHEL) 仮想マシンを設定します。

例10.1 RHEL 仮想マシンのマニフェストの例

```
apiVersion: kubevirt.io/v1
```

```
kind: VirtualMachine
metadata:
  labels:
    app: <vm_name> 1
  name: <vm_name>
spec:
  dataVolumeTemplates:
  - apiVersion: cdi.kubevirt.io/v1beta1
    kind: DataVolume
    metadata:
      name: <vm_name>
    spec:
      sourceRef:
        kind: DataSource
        name: rhel9
        namespace: openshift-virtualization-os-images
      storage:
        resources:
          requests:
            storage: 30Gi
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/domain: <vm_name>
    spec:
      domain:
        cpu:
          cores: 1
          sockets: 2
          threads: 1
        devices:
          disks:
          - disk:
              bus: virtio
            name: rootdisk
          - disk:
              bus: virtio
            name: cloudinitdisk
          interfaces:
          - masquerade: {}
            name: default
          rng: {}
        features:
          smm:
            enabled: true
        firmware:
          bootloader:
            efi: {}
        resources:
          requests:
            memory: 8Gi
      evictionStrategy: LiveMigrate
      networks:
      - name: default
        pod: {}
```

```
    volumes:
    - dataVolume:
        name: <vm_name>
      name: rootdisk
    - cloudInitNoCloud:
        userData: |-
          #cloud-config
          user: cloud-user
          password: '<password>' 2
          chpasswd: { expire: False }
      name: cloudinitdisk
```

1 仮想マシンの名前を指定します。

2 cloud-user のパスワードを指定します。

2. マニフェストファイルを使用して仮想マシンを作成します。

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

3. オプション: 仮想マシンを開始します。

```
$ virtctl start <vm_name>
```

10.1.5. 仮想マシンのストレージボリュームタイプ

ストレージボリュームタイプ	Description
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。

ストレージボリュームタイプ	Description
dataVolume	データボリュームは、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。 **type: dataVolume** または **type: ""** を指定します。**persistentVolumeClaim** などの **type** に他の値を指定すると、警告が表示され、仮想マシンは起動しません。
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、仮想マシンの起動時にディスクとして仮想マシンに割り当てられます。 **containerDisk** ボリュームは、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。  **注記** **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、削除される際に破棄されます。**containerDisk** ボリュームは、CD-ROM などの読み取り専用ファイルシステムや破棄可能な仮想マシンに役立ちます。

ストレージボリュームタイプ	Description
emptyDisk	仮想マシンインターフェイ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。

10.1.6. 仮想マシンの RunStrategy について

仮想マシンの **RunStrategy** は、一連の条件に応じて仮想マシンインスタンス (VMI) の動作を判別します。**spec.runStrategy** 設定は、**spec.running** 設定の代わりに仮想マシン設定プロセスに存在します。**spec.runStrategy** 設定を使用すると、**true** または **false** の応答のみを伴う **spec.running** 設定とは対照的に、VMI の作成および管理をより柔軟に行えます。ただし、2 つの設定は相互排他的です。**spec.running** または **spec.runStrategy** のいずれかを使用できます。両方を使用する場合は、エラーが発生します。

4 つ RunStrategy が定義されています。

Always

VMI は仮想マシンの作成時に常に表示されます。元の VMI が何らかの理由で停止する場合に、新規の VMI が作成されます。これは **spec.running: true** と同じ動作です。

RerunOnFailure

前のインスタンスがエラーが原因で失敗する場合は、VMI が再作成されます。インスタンスは、仮想マシンが正常に停止する場合 (シャットダウン時など) には再作成されません。

Manual (手動)

start、**stop**、および **restart** virtctl クライアントコマンドは、 VMI の状態および存在を制御するために使用できます。

Halted

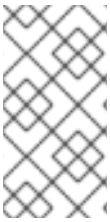
仮想マシンが作成される際に VMI は存在しません。これは **spec.running: false** と同じ動作です。

start、**stop**、および **restart** の virtctl コマンドの各種の組み合わせは、どの **RunStrategy** が使用されるかに影響を与えます。

以下の表は、仮想マシンの各種の状態からの移行について示しています。最初の列には、仮想マシンの初期の **RunStrategy** が表示されます。それぞれの追加の列には、virtctl コマンドと、このコマンド実行後の新規 **RunStrategy** が表示されます。

初期 RunStrategy	start	stop	restart
Always	-	Halted	Always
RerunOnFailure	-	Halted	RerunOnFailure

初期 RunStrategy	start	stop	restart
Manual	Manual	Manual	Manual
Halted	Always	-	-



注記

インストーラーでプロビジョニングされるインフラストラクチャーを使用してインストールされた OpenShift Virtualization クラスターでは、ノードで MachineHealthCheck に失敗し、クラスターで利用できなくなると、RunStrategy が **Always** または **RerunOnFailure** の仮想マシンが新規ノードで再スケジュールされます。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  RunStrategy: Always (1)
  template:
# ...
```

(1) VMI の現在の **RunStrategy** 設定。

10.1.7. 関連情報

- KubeVirt v0.59.0 API リファレンス の **VirtualMachineSpec** 定義は、仮想マシン仕様のパラメーターおよび階層に関するより広範なコンテキストを提供します。

注記

KubeVirt API リファレンスはアップストリームのプロジェクトリファレンスであり、OpenShift Virtualization でサポートされていないパラメーターが含まれる場合があります。

- CPU マネージャー が高パフォーマンスのワークロードプロファイルを使用できるようにします。
- **containerDisk** ボリュームとして仮想マシンに追加する前に、コンテナーディスクの準備 を参照してください。
- マシンヘルスチェックのデプロイと有効化の詳細は、マシンヘルスチェックのデプロイ を参照してください。
- インストーラーでプロビジョニングされるインフラストラクチャーについての詳細は、インストーラーでプロビジョニングされるインフラストラクチャーの概要 を参照してください。
- ストレージプロファイルのカスタマイズ

10.2. 仮想マシンの編集

Web コンソールの YAML エディターまたはコマンドラインの OpenShift CLI のいずれかを使用して、仮想マシン設定を更新できます。**Virtual Machine Details**画面でパラメーターのサブセットを更新することもできます。

10.2.1. Web コンソールでの仮想マシンの編集

OpenShift Container Platform Web コンソールまたはコマンドラインインターフェイスを使用して、仮想マシンを編集できます。

手順

1. Web コンソールで **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. フィールドが編集可能であることを示す鉛筆アイコンが付いているフィールドをクリックします。たとえば、BIOS や UEFI などの現在の **ブートモード** 設定をクリックして、**Boot mode** ウィンドウを開き、リストからオプションを選択します。
4. **Save** をクリックします。

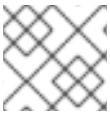
注記

仮想マシンが実行されている場合、**Boot Order** または **Flavor** への変更は仮想マシンを再起動するまで反映されません。

関連するフィールドの右側にある **View Pending Changes** をクリックして、保留中の変更を表示できます。ページ上部の **Pending Changes** バナーには、仮想マシンの再起動時に適用されるすべての変更のリストが表示されます。

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

Web コンソールで、仮想マシンの YAML 設定を編集できます。一部のパラメーターは変更できません。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. サイドメニューから **Virtualization → VirtualMachines** 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む、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。

10.2.3. CLI を使用した仮想マシン YAML 設定の編集

以下の手順を使用し、CLI を使用して仮想マシン YAML 設定を編集します。

前提条件

- YAML オブジェクト設定ファイルを使用し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。
   - 仮想マシンを再起動します。
   - 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。

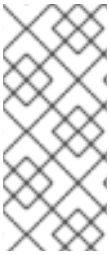
```
$ oc apply <object_type> <object_ID>
```

10.2.4. 仮想マシンへの仮想ディスクの追加

以下の手順を使用して仮想ディスクを仮想マシンに追加します。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration → Disks** タブで、**Add disk** をクリックします。
4. **Source**、**Name**、**Size**、**Type**、**Interface**、および **Storage Class** を指定します。
   a. オプション: 空のディスクソースを使用し、データボリュームの作成時に最大の書き込みパフォーマンスが必要な場合に、事前割り当てを有効にできます。そのためには、**Enable preallocation** チェックボックスをオンにします。
   b. オプション: **Apply optimized StorageProfile settings** をクリアして、仮想ディスクの **Volume Mode** と **Access Mode** を変更できます。これらのパラメーターを指定しない場合、システムは **kubevirt-storage-class-defaults** config map のデフォルト値を使用します。
5. **Add** をクリックします。



注記

仮想マシンが実行中の場合、新規ディスクは **pending restart** 状態にあり、仮想マシンを再起動するまで割り当てられません。

ページ上部の **Pending Changes** バナーには、仮想マシンの再起動時に適用されるすべての変更の一覧が表示されます。

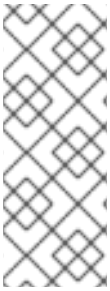
ストレージクラスのデフォルトを設定するには、ストレージプロファイルを使用します。詳細については、ストレージプロファイルのカスタマイズを参照してください。

10.2.4.1. ストレージフィールド

Name	選択	Description
Source	空白 (PVC の作成)	空のディスクを作成します。
	URL を使用したインポート (PVC の作成)	URL (HTTP または HTTPS エンドポイント) を介してコンテンツをインポートします。
	既存 PVC の使用	クラスターですでに利用可能な PVC を使用します。
	既存の PVC のクローン作成 (PVC の作成)	クラスターで利用可能な既存の PVC を選択し、このクローンを作成します。
	レジストリーを使用したインポート (PVC の作成)	コンテナーレジストリーを使用してコンテンツをインポートします。
	コンテナー (一時的)	クラスターからアクセスできるレジストリーにあるコンテナーからコンテンツをアップロードします。コンテナーディスクは、CD-ROM や一時的な仮想マシンなどの読み取り専用ファイルシステムにのみ使用する必要があります。
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		ディスクのサイズ (GiB 単位)。

Name	選択	Description
型		ディスクのタイプ。例: Disk または CD-ROM
Interface		ディスクデバイスのタイプ。サポートされるインターフェイスは、**virtIO**、**SATA**、および **SCSI** です。
Storage Class		ディスクの作成に使用されるストレージクラス。

ストレージの詳細設定
以下のストレージの詳細設定はオプションであり、**Blank**、**Import via URLURL**、および **Clone existing PVC** ディスクで利用できます。OpenShift Virtualization 4.11 より前では、これらのパラメーターを指定しない場合、システムは **kubevirt-storage-class-defaults** 設定マップのデフォルト値を使用します。OpenShift Virtualization 4.11 以降では、システムは ストレージプロファイル のデフォルト値を使用します。



注記

ストレージプロファイルを使用して、OpenShift Virtualization のストレージをプロビジョニングするときに一貫した高度なストレージ設定を確保します。

Volume Mode と **Access Mode** を手動で指定するには、デフォルトで選択されている **Apply optimized StorageProfile settings** チェックボックスをオフにする必要があります。

Name	モードの説明	パラメーター	パラメーターの説明
ボリュームモード	永続ボリュームがフォーマットされたファイルシステムまたは raw ブロック状態を使用するかどうかを定義します。デフォルトは **Filesystem** です。	Filesystem	ファイルシステムベースのボリュームで仮想ディスクを保存します。
		Block	ブロックボリュームで仮想ディスクを直接保存します。基礎となるストレージがサポートしている場合は、**Block** を使用します。
アクセスモード	永続ボリュームのアクセスモード。	ReadWriteOnce (RWO)	ボリュームはシングルノードで読み取り/書き込みとしてマウントできます。

Name	モードの説明	パラメーター	パラメーターの説明
		ReadWriteMany (RWX)	ボリュームは、一度に多くのノードで読み取り/書き込みとしてマウントできます。 **注記** これは、ノード間の仮想マシンのライブマイグレーションなどの、一部の機能で必要になります。
		ReadOnlyMany (ROX)	ボリュームは数多くのノードで読み取り専用としてマウントできます。

10.2.5. シークレット、設定マップ、またはサービスアカウントの仮想マシンへの追加

OpenShift Container Platform Web コンソールを使用して、シークレット、設定マップ、またはサービスアカウントを仮想マシンに追加します。

これらのリソースは、ディスクとして仮想マシンに追加されます。他のディスクをマウントするように、シークレット、設定マップ、またはサービスアカウントをマウントします。

仮想マシンが実行中の場合、仮想マシンを再起動するまで、変更は有効になりません。新しく追加されたリソースは、ページの上部で保留中の変更としてマークされます。

前提条件

- 追加するシークレット、設定マップ、またはサービスアカウントは、ターゲット仮想マシンと同じ namespace に存在する必要がある。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration → Environment** をクリックします。
4. **Add Config Map, Secret or Service Account**をクリックします。
5. **Select a resource** をクリックし、リストから resource を選択します。6 文字のシリアル番号が、選択したリソースについて自動的に生成されます。

6. オプション: **Reload** をクリックして、環境を最後に保存した状態に戻します。
7. **Save** をクリックします。

検証

1. **VirtualMachine details** ページで、**Configuration → Disks** をクリックし、リソースがディスクのリストに表示されていることを確認します。
2. **Actions → Restart** をクリックして、仮想マシンを再起動します。

他のディスクをマウントするように、シークレット、設定マップ、またはサービスアカウントをマウントできるようになりました。

config map、シークレット、サービスアカウントの追加リソース

- 設定マップについて
- Pod への機密性の高いデータの提供
- サービスアカウントの概要および作成

10.2.6. 仮想マシンへのネットワークインターフェイスの追加

以下の手順を使用してネットワークインターフェイスを仮想マシンに追加します。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration → Network interfaces** タブで、**Add Network Interface** をクリックします。
4. **Add Network Interface** ウィンドウで、ネットワークインターフェイスの **Name**、**Model**、**Network**、**Type**、および **MAC Address** を指定します。
5. **Add** をクリックします。

注記

仮想マシンが実行中の場合、新規ネットワークインターフェイスは **pending restart** 状態にあり、仮想マシンを再起動するまで変更は反映されません。

ページ上部の **Pending Changes** バナーには、仮想マシンの再起動時に適用されるすべての変更のリストが表示されます。

10.2.6.1. ネットワークフィールド

Name	説明
Name	ネットワークインターフェイスコントローラーの名前。

Name	説明
モデル	ネットワークインターフェイスコントローラーのモデルを示します。サポートされる値は **e1000e** および **virtio** です。
Network	利用可能なネットワーク接続定義のリスト。
型	利用可能なバインディングメソッドの一覧。ネットワークインターフェイスに適したバインド方法を選択します。 ● デフォルトの Pod ネットワーク: **masquerade** ● Linux ブリッジネットワーク: **bridge** ● SR-IOV ネットワーク: **SR-IOV**
MAC Address	ネットワークインターフェイスコントローラーの MAC アドレス。MAC アドレスが指定されていない場合、これは自動的に割り当てられます。

10.3. ブート順序の編集

Web コンソールまたは CLI を使用して、ブート順序リストの値を更新できます。

Virtual Machine Overview ページの **Boot Order** で、以下を実行できます。

- ディスクまたはネットワークインターフェイスコントローラー (NIC) を選択し、これをブート順序のリストに追加します。
- ブート順序の一覧でディスクまたは NIC の順序を編集します。
- ブート順序のリストからディスクまたは NIC を削除して、起動可能なソースのインベントリーに戻します。

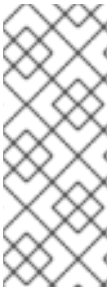
10.3.1. Web コンソールでのブート順序リストへの項目の追加

Web コンソールを使用して、ブート順序リストに項目を追加します。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Details** タブをクリックします。
4. **Boot Order** の右側にある鉛筆アイコンをクリックします。YAML 設定が存在しない場合や、これがブート順序リストの初回作成時の場合、以下のメッセージが表示されます。**No resource selected.仮想マシンは、YAML ファイルでの出現順にディスクからの起動を試行します。**

5. **Add Source** をクリックして、仮想マシンのブート可能なディスクまたはネットワークインターフェイスコントローラー (NIC) を選択します。
6. 追加のディスクまたは NIC をブート順序一覧に追加します。
7. **Save** をクリックします。



注記

仮想マシンが実行されている場合、**Boot Order** への変更は仮想マシンを再起動するまで反映されません。

Boot Order フィールドの右側にある **View Pending Changes** をクリックして、保留中の変更を表示できます。ページ上部の **Pending Changes** バナーには、仮想マシンの再起動時に適用されるすべての変更のリストが表示されます。

10.3.2. Web コンソールでのブート順序リストの編集

Web コンソールで起動順序リストを編集します。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Details** タブをクリックします。
4. **Boot Order** の右側にある鉛筆アイコンをクリックします。
5. ブート順序リストで項目を移動するのに適した方法を選択します。
   - スクリーンリーダーを使用しない場合、移動する項目の横にある矢印アイコンにカーソルを合わせ、項目を上下にドラッグし、選択した場所にドロップします。
   - スクリーンリーダーを使用する場合は、上矢印キーまたは下矢印を押して、ブート順序リストで項目を移動します。次に **Tab** キーを押して、選択した場所に項目をドロップします。
6. **Save** をクリックします。

注記

仮想マシンが実行されている場合、ブート順序の変更は仮想マシンが再起動されるまで反映されません。

Boot Order フィールドの右側にある **View Pending Changes** をクリックして、保留中の変更を表示できます。ページ上部の **Pending Changes** バナーには、仮想マシンの再起動時に適用されるすべての変更のリストが表示されます。

10.3.3. YAML 設定ファイルでのブート順序リストの編集

CLI を使用して、YAML 設定ファイルのブート順序のリストを編集します。

手順

1. 以下のコマンドを実行して、仮想マシンの YAML 設定ファイルを開きます。

```
$ oc edit vm example
```

2. YAML ファイルを編集し、ディスクまたはネットワークインターフェイスコントローラー (NIC) に関連付けられたブート順序の値を変更します。以下に例を示します。

```
disks:
  - bootOrder: 1 (1)
    disk:
      bus: virtio
    name: containerdisk
  - disk:
      bus: virtio
    name: cloudinitdisk
  - cdrom:
      bus: virtio
    name: cd-drive-1
interfaces:
  - boot Order: 2 (2)
    macAddress: '02:96:c4:00:00'
    masquerade: {}
    name: default
```

(1) ディスクに指定されたブート順序の値。

(2) ネットワークインターフェイスコントローラーに指定されたブート順序の値。

3. YAML ファイルを保存します。

4. **reload the content** をクリックして、Web コンソールで YAML ファイルの更新されたブート順序の値をブート順序リストに適用します。

10.3.4. Web コンソールでのブート順序リストからの項目の削除

Web コンソールを使用して、ブート順序のリストから項目を削除します。

手順

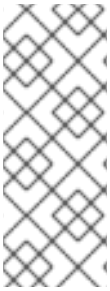
1. サイドメニューから **Virtualization** → **VirtualMachines** をクリックします。

2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。

3. **Details** タブをクリックします。

4. **Boot Order** の右側にある鉛筆アイコンをクリックします。

5. 項目の横にある **Remove** アイコン ⛔ をクリックします。この項目はブート順序のリストから削除され、利用可能なブートソースのリストに保存されます。ブート順序リストからすべての項目を削除する場合、以下のメッセージが表示されます。**No resource selected.仮想マシンは、YAML ファイルでの出現順にディスクからの起動を試行します。**



注記

仮想マシンが実行されている場合、**Boot Order** への変更は仮想マシンを再起動するまで反映されません。

Boot Order フィールドの右側にある **View Pending Changes** をクリックして、保留中の変更を表示できます。ページ上部の **Pending Changes** バナーには、仮想マシンの再起動時に適用されるすべての変更のリストが表示されます。

10.4. 仮想マシンの削除

Web コンソールまたは **oc** コマンドラインインターフェイスを使用して、仮想マシンを削除できます。

10.4.1. 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。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization** → **VirtualMachines** をクリックします。
2. 仮想マシンの横にある Options メニュー をクリックし、**Delete** を選択します。または、仮想マシン名をクリックして **VirtualMachine details** ページを開き、**Actions** → **Delete** をクリックします。
3. オプション: **With grace period** を選択するか、**Delete disks** をクリアします。
4. **Delete** をクリックして、仮想マシンを完全に削除します。

10.4.2. CLI の使用による仮想マシンの削除

oc コマンドラインインターフェイス (CLI) を使用して仮想マシンを削除できます。**oc** クライアントを使用すると、複数の仮想マシンでアクションを実行できます。

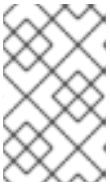
前提条件

- 削除する仮想マシンの名前を特定している。

手順

- 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 <vm_name>
```



注記

このコマンドは、現在のプロジェクト内の VM のみを削除します。削除する仮想マシンが別のプロジェクトまたは namespace にある場合は、**-n <project_name>** オプションを指定します。

10.5. 仮想マシンのエクスポート

仮想マシンを別のクラスターにインポートしたり、フォレンジック目的でボリュームを分析したりするために、仮想マシン (VM) とそれに関連付けられたディスクをエクスポートできます。

コマンドラインインターフェイスを使用して、**VirtualMachineExport** カスタムリソース (CR) を作成します。

または、**virtctl vmexport コマンド** を使用して **VirtualMachineExport** CR を作成し、エクスポートされたボリュームをダウンロードすることもできます。

10.5.1. VirtualMachineExport カスタムリソースの作成

VirtualMachineExport カスタムリソース (CR) を作成して、次のオブジェクトをエクスポートできます。

- 仮想マシン (VM): 指定された仮想マシンの永続ボリューム要求 (PVC) をエクスポートします。
- VM スナップショット: **VirtualMachineSnapshot** CR に含まれる PVC をエクスポートします。
- PVC: PVC をエクスポートします。PVC が **virt-launcher** Pod などの別の Pod で使用されている場合、エクスポートは PVC が使用されなくなるまで **Pending** 状態のままになります。

VirtualMachineExport CR は、エクスポートされたボリュームの内部および外部リンクを作成します。内部リンクはクラスター内で有効です。外部リンクには、**Ingress** または **Route** を使用してアクセスできます。

エクスポートサーバーは、次のファイル形式をサポートしています。

- **raw**: raw ディスクイメージファイル。
- **gzip**: 圧縮されたディスクイメージファイル。
- **dir**: PVC ディレクトリーとファイル。
- **tar.gz**: 圧縮された PVC ファイル。

前提条件

- 仮想マシンをエクスポートするために、仮想マシンがシャットダウンされている。

手順

1. 次の例に従って **VirtualMachineExport** マニフェストを作成し、**VirtualMachine**、**VirtualMachineSnapshot**、または **PersistentVolumeClaim** CR からボリュームをエクスポートし、**example-export.yaml** として保存します。

   VirtualMachineExport の例

```
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: example-export
spec:
  source:
    apiGroup: "kubevirt.io" 1
```

```
  kind: VirtualMachine (2)
  name: example-vm
ttlDuration: 1h (3)
```

(1) 適切な API グループを指定します。

- **VirtualMachine** の **"kubevirt.io"**。
- **VirtualMachineSnapshot** の **"snapshot.kubevirt.io"**。
- **PersistentVolumeClaim** の **""**。

(2) **VirtualMachine**、**VirtualMachineSnapshot**、または **PersistentVolumeClaim** を指定します。

(3) オプション: デフォルトの期間は 2 時間です。

2. **VirtualMachineExport** CR を作成します。

```
$ oc create -f example-export.yaml
```

3. **VirtualMachineExport** CR を取得します。

```
$ oc get vmexport example-export -o yaml
```

エクスポートされたボリュームの内部および外部リンクは、**status** スタンザに表示されます。

出力例

```
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: example-export
  namespace: example
spec:
  source:
    apiGroup: ""
    kind: PersistentVolumeClaim
    name: example-pvc
  tokenSecretRef: example-token
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2022-06-21T14:10:09Z"
    reason: podReady
    status: "True"
    type: Ready
  - lastProbeTime: null
    lastTransitionTime: "2022-06-21T14:09:02Z"
    reason: pvcBound
    status: "True"
    type: PVCReady
  links:
    external: (1)
```

```
      cert: |-
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
      volumes:
      - formats:
        - format: raw
          url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/volumes/example-disk/disk.img
        - format: gzip
          url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/volumes/example-disk/disk.img.gz
        name: example-disk
    internal:  2
      cert: |-
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
      volumes:
      - formats:
        - format: raw
          url: https://virt-export-example-export.example.svc/volumes/example-disk/disk.img
        - format: gzip
          url: https://virt-export-example-export.example.svc/volumes/example-disk/disk.img.gz
        name: example-disk
  phase: Ready
  serviceName: virt-export-example-export
```

1 外部リンクは、**Ingress** または **Route** を使用してクラスターの外部からアクセスできます。

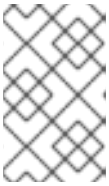
2 内部リンクは、クラスター内でのみ有効です。

10.5.2. エクスポートされた仮想マシンマニフェストへのアクセス

仮想マシン (VM) またはスナップショットをエクスポートすると、エクスポートサーバーから **VirtualMachine** マニフェストと関連情報を取得できます。

前提条件

- **VirtualMachineExport** カスタムリソース (CR) を作成して、仮想マシンまたは VM スナップショットをエクスポートしている。



注記

spec.source.kind: PersistentVolumeClaim パラメーターを持つ **VirtualMachineExport** オブジェクトは、仮想マシンマニフェストを生成しません。

手順

1. マニフェストにアクセスするには、まず証明書をソースクラスターからターゲットクラスターにコピーする必要があります。

   a. ソースクラスターにログインします。

   b. 次のコマンドを実行して、証明書を **cacert.crt** ファイルに保存します。

   ```
   $ oc get vmexport <export_name> -o jsonpath={.status.links.external.cert} > cacert.crt
   ```
   (1)

   (1) **<export_name>** を、**VirtualMachineExport** オブジェクトの **metadata.name** 値に置き換えます。

   c. **cacert.crt** ファイルをターゲットクラスターにコピーします。

2. 次のコマンドを実行して、ソースクラスター内のトークンをデコードし、**token_decode** ファイルに保存します。

   ```
   $ oc get secret export-token-<export_name> -o jsonpath={.data.token} | base64 --decode > token_decode
   ```
   (1)

   (1) **<export_name>** を、**VirtualMachineExport** オブジェクトの **metadata.name** 値に置き換えます。

3. **token_decode** ファイルをターゲットクラスターにコピーします。

4. 次のコマンドを実行して、**VirtualMachineExport** カスタムリソースを取得します。

   ```
   $ oc get vmexport <export_name> -o yaml
   ```

5. **status.links** スタンザを確認します。このスタンザは **external** セクションと **internal** セクションに分かれています。各セクション内の **manifests.url** フィールドに注意してください。

   出力例

   ```
   apiVersion: export.kubevirt.io/v1alpha1
   kind: VirtualMachineExport
   metadata:
     name: example-export
   spec:
     source:
       apiGroup: "kubevirt.io"
       kind: VirtualMachine
       name: example-vm
     tokenSecretRef: example-token
   status:
   #...
     links:
       external:
   #...
         manifests:
         - type: all
           url: https://vmexport-
   ```

```
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/external/manifests/all (1)
      - type: auth-header-secret
        url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/external/manifests/secret (2)
    internal:
#...
      manifests:
      - type: all
        url: https://virt-export-export-pvc.default.svc/internal/manifests/all (3)
      - type: auth-header-secret
        url: https://virt-export-export-pvc.default.svc/internal/manifests/secret
  phase: Ready
  serviceName: virt-export-example-export
```

(1) **VirtualMachine** マニフェスト、存在する場合は **DataVolume** マニフェスト、外部 URL の Ingress またはルートの公開証明書を含む **ConfigMap** マニフェストが含まれます。

(2) Containerized Data Importer (CDI) と互換性のあるヘッダーを含むシークレットが含まれます。ヘッダーには、エクスポートトークンのテキストバージョンが含まれています。

(3) **VirtualMachine** マニフェスト、存在する場合は **DataVolume** マニフェスト、および内部 URL のエクスポートサーバーの証明書を含む **ConfigMap** マニフェストが含まれます。

6. ターゲットクラスターにログインします。

7. 次のコマンドを実行して **Secret** マニフェストを取得します。

```
$ curl --cacert cacert.crt <secret_manifest_url> -H \ (1)
"x-kubevirt-export-token:token_decode" -H \ (2)
"Accept:application/yaml"
```

(1) **<secret_manifest_url>** を、**VirtualMachineExport** YAML 出力の **auth-header-secret** URL に置き換えます。

(2) 前に作成した **token_decode** ファイルを参照します。

以下に例を示します。

```
$ curl --cacert cacert.crt https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/external/manifests/secret -H "x-kubevirt-export-token:token_decode" -H
"Accept:application/yaml"
```

8. 次のコマンドを実行して、**ConfigMap** マニフェストや **VirtualMachine** マニフェストなどの **type: all** マニフェストを取得します。

```
$ curl --cacert cacert.crt <all_manifest_url> -H \ (1)
"x-kubevirt-export-token:token_decode" -H \ (2)
"Accept:application/yaml"
```

1. **<all_manifest_url>** を、**VirtualMachineExport** YAML 出力の URL に置き換えます。
2. 前に作成した **token_decode** ファイルを参照します。

以下に例を示します。

```
$ curl --cacert cacert.crt https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/external/manifests/all -H "x-kubevirt-export-token:token_decode" -H
"Accept:application/yaml"
```

次のステップ

- エクスポートしたマニフェストを使用して、ターゲットクラスター上に **ConfigMap** オブジェクトと **VirtualMachine** オブジェクトを作成できます。

10.5.3. 関連情報

- データボリュームの使用による仮想マシンイメージのインポート

10.6. 仮想マシンインスタンスの管理

OpenShift Virtualization 環境の外部で独立して作成されたスタンドアロン仮想マシンインスタンス (VMI) がある場合、Web コンソールを使用するか、コマンドラインインターフェイス (CLI) から **oc** または **virtctl** コマンドを使用してそれらを管理できます。

virtctl コマンドは、**oc** コマンドよりも多くの仮想化オプションを提供します。たとえば、**virtctl** を使用して仮想マシンを一時停止したり、ポートを公開したりできます。

10.6.1. 仮想マシンインスタンスについて

仮想マシンインスタンス (VMI) は、実行中の仮想マシンを表します。VMI が仮想マシンまたは別のオブジェクトによって所有されている場合、Web コンソールで、または **oc** コマンドラインインターフェイス (CLI) を使用し、所有者を通してこれを管理します。

スタンドアロンの VMI は、自動化または CLI で他の方法により、スクリプトを使用して独立して作成され、起動します。お使いの環境では、OpenShift Virtualization 環境外で開発され、起動されたスタンドアロンの VMI が存在する可能性があります。CLI を使用すると、引き続きそれらのスタンドアロン VMI を管理できます。スタンドアロン VMI に関連付けられた特定のタスクに Web コンソールを使用することもできます。

- スタンドアロン VMI とそれらの詳細をリスト表示します。
- スタンドアロン VMI のラベルとアノテーションを編集します。
- スタンドアロン VMI を削除します。

仮想マシンを削除する際に、関連付けられた VMI は自動的に削除されます。仮想マシンまたは他のオブジェクトによって所有されていないため、スタンドアロン VMI を直接削除します。

注記

OpenShift Virtualization をアンインストールする前に、CLI または Web コンソールを使用してスタンドアロンの VMI のリストを表示します。次に、未処理の VMI を削除します。

10.6.2. CLI を使用した仮想マシンインスタンスのリスト表示

oc コマンドラインインターフェイス (CLI) を使用して、スタンドアロンおよび仮想マシンによって所有されている VMI を含むすべての仮想マシンのリストを表示できます。

手順

- 以下のコマンドを実行して、すべての VMI のリストを表示します。

```
$ oc get vmis -A
```

10.6.3. Web コンソールを使用したスタンドアロン仮想マシンインスタンスのリスト表示

Web コンソールを使用して、仮想マシンによって所有されていないクラスター内のスタンドアロンの仮想マシンインスタンス (VMI) のリストを表示できます。

注記

仮想マシンまたは他のオブジェクトが所有する VMI は、Web コンソールには表示されません。Web コンソールは、スタンドアロンの VMI のみを表示します。クラスター内のすべての VMI をリスト表示するには、CLI を使用する必要があります。

手順

- サイドメニューから **Virtualization → VirtualMachines** をクリックします。
スタンドアロン VMI は、名前の横にある濃い色のバッジで識別できます。

10.6.4. Web コンソールを使用したスタンドアロン仮想マシンインスタンスの編集

Web コンソールを使用して、スタンドアロン仮想マシンインスタンスのアノテーションおよびラベルを編集できます。他のフィールドは編集できません。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. スタンドアロン VMI を選択して、**VirtualMachineInstance details** ページを開きます。
3. **Details** タブで、**Annotations** または **Labels** の横にある鉛筆アイコンをクリックします。
4. 関連する変更を加え、**Save** をクリックします。

10.6.5. CLI を使用したスタンドアロン仮想マシンインスタンスの削除

oc コマンドラインインターフェイス (CLI) を使用してスタンドアロン仮想マシンインスタンス (VMI) を削除できます。

前提条件

- 削除する必要のある VMI の名前を特定している。

手順

- 以下のコマンドを実行して VMI を削除します。

```
$ oc delete vmi <vmi_name>
```

10.6.6. Web コンソールを使用したスタンドアロン仮想マシンインスタンスの削除

Web コンソールからスタンドアロン仮想マシンインスタンス (VMI) を削除します。

手順

1. OpenShift Container Platform Web コンソールで、サイドメニューから**Virtualization → VirtualMachines**をクリックします。
2. **Actions → Delete VirtualMachineInstance** 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、スタンドアロン VMI を永続的に削除します。

10.7. 仮想マシンの状態の制御

Web コンソールから仮想マシンを停止し、起動し、再起動し、一時停止を解除することができます。

virtctl を使用して仮想マシンの状態を管理し、CLI から他のアクションを実行できます。たとえば、**virtctl** を使用して仮想マシンを強制停止したり、ポートを公開したりできます。

10.7.1. 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. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 起動する仮想マシンが含まれる行を見つけます。
3. ユースケースに応じて適切なメニューに移動します。
   - 複数の仮想マシンでのアクションの実行が可能なこのページに留まるには、以下を実行します。
     a. 行の右端にある Options メニュー  をクリックします。
   - 選択した仮想マシンを起動する前に、その仮想マシンの総合的な情報を表示するには、以下を実行します。

a. 仮想マシンの名前をクリックして、**VirtualMachine details** ページにアクセスします。

b. **Actions** をクリックします。

4. **再起動**を選択します。

5. 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、OpenShift 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。

10.7.2. 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

エラーを回避するには、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。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。

2. 再起動する仮想マシンが含まれる行を見つけます。

3. ユースケースに応じて適切なメニューに移動します。

- 複数の仮想マシンでのアクションの実行が可能なこのページに留まるには、以下を実行します。

a. 行の右端にある Options メニュー  をクリックします。

- 選択した仮想マシンを再起動する前に、その仮想マシンの総合的な情報を表示するには、以下を実行します。

a. 仮想マシンの名前をクリックして、**VirtualMachine details** ページにアクセスします。

b. **Actions → Restart** をクリックします。

4. 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。

10.7.3. 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. サイドメニューから **Virtualization → VirtualMachines** をクリックします。

2. 停止する仮想マシンが含まれる行を見つけます。
3. ユースケースに応じて適切なメニューに移動します。
   - 複数の仮想マシンでのアクションの実行が可能なこのページに留まるには、以下を実行します。
     a. 行の右端にある Options メニュー  をクリックします。
   - 選択した仮想マシンを停止する前に、その仮想マシンの総合的な情報を表示するには、以下を実行します。
     a. 仮想マシンの名前をクリックして、**VirtualMachine details** ページにアクセスします。
     b. **Actions → Stop** をクリックします。
4. 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。

10.7.4. 仮想マシンの一時停止の解除

Web コンソールから仮想マシンの一時停止を解除できます。

前提条件

- 1 つ以上の仮想マシンのステータスが **Paused** である必要がある。

注記

virtctl クライアントを使用して仮想マシンを一時停止することができます。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 一時停止を解除する仮想マシンが含まれる行を見つけます。
3. ユースケースに応じて適切なメニューに移動します。
   - 複数の仮想マシンでのアクションの実行が可能なこのページに留まるには、以下を実行します。
     a. **Status** 列で、**Paused** をクリックします。
   - 選択した仮想マシンの一時停止を解除する前に、その仮想マシンの総合的な情報を表示するには、以下を実行します。
     a. 仮想マシンの名前をクリックして、**VirtualMachine details** ページにアクセスします。
     b. **Status** の右側にある鉛筆アイコンをクリックします。
4. 確認ウィンドウで **Stop** をクリックし、仮想マシンの一時停止を解除します。

10.8. 仮想マシンコンソールへのアクセス

OpenShift 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。これらのコンソールには、OpenShift Container Platform Web コンソールから、または CLI コマンドを使用してアクセスできます。

注記

単一の仮想マシンに対する同時 VNC 接続の実行は、現在サポートされていません。

10.8.1. OpenShift Container Platform Web コンソールでの仮想マシンコンソールへのアクセス

OpenShift Container Platform Web コンソールでシリアルコンソールまたは VNC コンソールを使用して、仮想マシンに接続できます。

OpenShift Container Platform Web コンソールで、RDP (リモートデスクトッププロトコル) を使用するデスクトップビューアーコンソールを使用して、Windows 仮想マシンに接続できます。

10.8.1.1. シリアルコンソールへの接続

Web コンソールの **VirtualMachine details** ページにある **Console** 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. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization** → **VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Console** タブをクリックします。VNC コンソールがデフォルトで開きます。
4. 一度に 1 つのコンソール セッションのみが開かれるようにするには、**Disconnect** をクリックします。それ以外の場合、VNC コンソール セッションはバックグラウンドでアクティブなままになります。
5. **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。
6. **Disconnect** をクリックして、コンソールセッションを終了します。
7. オプション: **Open Console in New Window** をクリックして、別のウィンドウでシリアルコンソールを開きます。

10.8.1.2. VNC コンソールへの接続

Web コンソールの **VirtualMachine details** ページにある **Console** 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. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization** → **VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Console** タブをクリックします。VNC コンソールがデフォルトで開きます。

4. オプション: **Open Console in New Window**をクリックして、別のウィンドウで VNC コンソールを開きます。
5. オプション: **Send Key** をクリックして、キーの組み合わせを仮想マシンに送信します。
6. コンソールウィンドウの外側をクリックし、**Disconnect** をクリックしてセッションを終了します。

10.8.1.3. RDP を使用した Windows 仮想マシンへの接続

Remote Desktop Protocol (RDP) を使用する **Desktop viewer** 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 **VirtualMachine 詳細** ページの **Console** タブから仮想マシンの **console.rdp** ファイルをダウンロードし、優先する RDP クライアントに提供します。

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。
- 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. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. Windows 仮想マシンをクリックして、**VirtualMachine details** ページを開きます。
3. **Console** タブをクリックします。
4. コンソールのリストから、**Desktop viewer** を選択します。
5. **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。
6. 優先する RDP クライアントで **console.rdp** ファイルを参照して、Windows 仮想マシンに接続します。

10.8.1.4. 仮想マシンの表示の切り替え

Windows 仮想マシン (VM) に vGPU が接続されている場合、Web コンソールを使用してデフォルトのディスプレイと vGPU ディスプレイを切り替えることができます。

前提条件

- 仲介されたデバイスは、**HyperConverged** カスタムリソースで設定され、仮想マシンに割り当てられます。
- 仮想マシンは実行中です。

手順

1. OpenShift Container Platform コンソールで、**Virtualization → VirtualMachines** をクリックします。
2. Windows 仮想マシンを選択して **Overview** 画面を開きます。
3. **Console** タブをクリックします。
4. コンソールのリストから、**VNC console** を選択します。
5. **Send Key** リストから適切なキーの組み合わせを選択します。
   a. デフォルトの仮想マシン表示にアクセスするには、**Ctl + Alt+ 1** を選択します。
   b. vGPU ディスプレイにアクセスするには、**Ctl + Alt + 2** を選択します。

関連情報

- 仲介デバイスの設定

10.8.1.5. Web コンソールを使用した SSH コマンドのコピー

コマンドをコピーして、SSH 経由で仮想マシン (VM) ターミナルに接続します。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. **オプション** メニューをクリック  仮想マシンの **SSH コマンドのコピー** を選択します。
3. ターミナルに貼り付け、仮想マシンにアクセスします。

10.8.2. 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

10.8.2.1. virtctl を使用して SSH 経由で仮想マシンにアクセスする

virtctl ssh コマンドを使用して、ローカル SSH クライアントを使用して SSH トラフィックを仮想マシン (VM) に転送できます。仮想マシンで SSH キー認証を設定している場合は、手順 1 は必要ないため、手順 2 に進んでください。

注記

コントロールプレーンの SSH トラフィックが多いと、API サーバーの速度が低下する可能性があります。定期的に多数の接続が必要な場合は、専用の Kubernetes **Service** オブジェクトを使用して仮想マシンにアクセスします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **virtctl** クライアントをインストールしました。
- アクセスする仮想マシンが実行されています。

- 仮想マシンと同じプロジェクトにいます。

手順

1. SSH キー認証を設定します。
   a. **ssh-keygen** コマンドを使用して、SSH 公開鍵ペアを生成します。

```
$ ssh-keygen -f <key_file> 1
```

   1 キーを格納するファイルを指定します。

   b. 仮想マシンにアクセスするための SSH 公開鍵を含む SSH 認証シークレットを作成します。

```
$ oc create secret generic my-pub-key --from-file=key1=<key_file>.pub
```

   c. **VirtualMachine** マニフェストにシークレットへの参照を追加します。以下に例を示します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: testvm
spec:
  running: true
  template:
    spec:
      accessCredentials:
      - sshPublicKey:
          source:
            secret:
              secretName: my-pub-key 1
          propagationMethod:
            configDrive: {} 2
# ...
```

   1 SSH 認証 **Secret** オブジェクトへの参照。

   2 SSH 公開鍵は、**configDrive** プロバイダーを使用して cloud-init メタデータとして仮想マシンに挿入されます。

   d. 仮想マシンを再起動して変更を適用します。

2. SSH 経由で仮想マシンに接続します。
   a. 次のコマンドを実行して、SSH 経由で仮想マシンにアクセスします。

```
$ virtctl ssh -i <key_file> <vm_username>@<vm_name>
```

   b. オプション: 仮想マシンとの間でファイルを安全に転送するには、次のコマンドを使用します。

マシンから仮想マシンにファイルをコピーする

```
$ virtctl scp -i <key_file> <filename> <vm_username>@<vm_name>:
```

仮想マシンからマシンにファイルをコピーする

```
$ virtctl scp -i <key_file> <vm_username@<vm_name>:<filename> .
```

関連情報

- 仮想マシンを公開するサービスの作成
- シークレットについて

10.8.2.2. OpenSSH および virtctl port-forward の使用

ローカルの OpenSSH クライアントと **virtctl port-forward** コマンドを使用して、実行中の仮想マシン (VM) に接続できます。Ansible でこの方法を使用すると、VM の設定を自動化できます。

ポート転送トラフィックはコントロールプレーン経由で送信されるため、この方法はトラフィックの少ないアプリケーションに推奨されます。ただし、API サーバーに負荷が大きいため、Rsync や Remote Desktop Protocol などのトラフィックの高いアプリケーションには推奨されません。

前提条件

- **virtctl** クライアントをインストールしている。
- アクセスする仮想マシンが実行されている。
- **virtctl** ツールがインストールされている環境には、仮想マシンへのアクセスに必要なクラスターパーミッションがある。たとえば、**oc login** を実行するか、**KUBECONFIG** 環境変数を設定します。

手順

1. 以下のテキストをクライアントマシンの ~/**.ssh/config** ファイルに追加します。

```
Host vm/*
  ProxyCommand virtctl port-forward --stdio=true %h %p
```

2. 次のコマンドを実行して、仮想マシンに接続します。

```
$ ssh <user>@vm/<vm_name>.<namespace>
```

10.8.2.3. 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>
```

10.8.2.4. 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イスに接続します。

```
$ 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
```

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

ローカルのリモートデスクトッププロトコル (RDP) クライアントを使用して、Windows 仮想マシン (VM) に接続するための Kubernetes **Service** オブジェクトを作成します。

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** オブジェクトは VirtIO ドライバーに含まれています。
- ローカルマシンにインストールされた RDP クライアント。

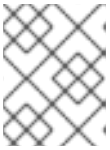
手順

1. **VirtualMachine** マニフェストを編集して、サービス作成のラベルを追加します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
```

```
  name: vm-ephemeral
  namespace: example-namespace
spec:
  running: false
  template:
    metadata:
      labels:
        special: key (1)
# ...
```

(1) ラベル **special: key** を **spec.template.metadata.labels** セクションに追加します。



注記

仮想マシンのラベルは Pod に渡されます。**special: キー** ラベルは、**Service** マニフェストの **spec.selector** 属性のラベルと一致する必要があります。

2. **VirtualMachine** マニフェストファイルを保存して変更を適用します。
3. 仮想マシンを公開するための **Service** マニフェストを作成します。

```
apiVersion: v1
kind: Service
metadata:
  name: rdpservice (1)
  namespace: example-namespace (2)
spec:
  ports:
  - targetPort: 3389 (3)
    protocol: TCP
  selector:
    special: key (4)
  type: NodePort (5)
# ...
```

(1) **Service** オブジェクトの名前。

(2) **Service** オブジェクトが存在する namespace。これは **VirtualMachine** マニフェストの **metadata.namespace** フィールドと同じである必要があります。

(3) サービスによって公開される仮想マシンポート。ポートリストが仮想マシンマニフェストに定義されている場合は、オープンポートを参照する必要があります。

(4) **VirtualMachine** マニフェストの **spec.template.metadata.labels** スタンザに追加したラベルへの参照。

(5) サービスのタイプ。

4. **サービス** マニフェストファイルを保存します。
5. 以下のコマンドを実行してサービスを作成します。

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

6. 仮想マシンを起動します。仮想マシンがすでに実行中の場合は、再起動します。
7. **Service** オブジェクトをクエリーし、これが利用可能であることを確認します。

```
$ oc get service -n example-namespace
```

NodePort サービスの出力例

```
NAME        TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
rdpservice  NodePort   172.30.232.73  <none>        3389:30000/TCP   5m
```

8. 以下のコマンドを実行して、ノードの IP アドレスを取得します。

```
$ oc get node <node_name> -o wide
```

出力例

```
NAME    STATUS   ROLES   AGE    VERSION  INTERNAL-IP     EXTERNAL-IP
node01  Ready    worker  6d22h  v1.24.0  192.168.55.101  <none>
```

9. 優先する RDP クライアントでノード IP アドレスと割り当てられたポートを指定します。
10. Windows 仮想マシンに接続するためのユーザー名とパスワードを入力します。

10.9. SYSPREP を使用した WINDOWS のインストールの自動化

Microsoft DVD イメージと**sysprep**を使用して、Windows 仮想マシンのインストール、セットアップ、およびソフトウェアプロビジョニングを自動化できます。

10.9.1. Windows DVD を使用した VM ディスクイメージの作成

Microsoft はダウンロード用のディスクイメージを提供していませんが、Windows DVD を使用してディスクイメージを作成できます。このディスクイメージを使用して、仮想マシンを作成できます。

手順

1. Open Shift Virtualization Web コンソールで、**Storage → PersistentVolumeClaims → Create PersistentVolumeClaim With Data upload form**をクリックします。
2. 目的のプロジェクトを選択します。
3. **永続ボリューム要求の名前**を設定します。
4. Windows DVD から仮想マシンディスクイメージをアップロードします。これで、イメージをブートソースとして使用して、新しい Windows 仮想マシンを作成できます。

10.9.2. ディスクイメージを使用した Windows のインストール

ディスクイメージを使用して、仮想マシンに Windows をインストールできます。

前提条件

- Windows DVD を使用してディスクイメージを作成する必要があります。
- **autounattend.xml** 応答ファイルを作成する必要があります。詳細は、Microsoft のドキュメント を参照してください。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → Catalog** をクリックします。
2. Windows テンプレートを選択し、**Customize VirtualMachine** をクリックします。
3. **Disk source** リストから **Upload (Upload a new file to a PVC)**を選択し、DVD イメージを参照します。
4. **Review and create VirtualMachine** をクリックします。
5. **Clone available operating system source to this Virtual Machine**のチェックを外します。
6. **Start this VirtualMachine after creation** のチェックを外します。
7. **Scripts** タブの **Sysprep** セクションで、**Edit** をクリックします。
8. **autounattend.xml** 応答ファイルを参照し、**Save** をクリックします。
9. **Create VirtualMachine** をクリックします。
10. **YAML** タブで、**running:false** を **runStrategy: RerunOnFailure** に置き換え、**Save** をクリックします。

VM は、**autounattend.xml** 応答ファイルを含む **sysprep** ディスクで開始されます。

10.9.3. sysprep を使用した Windows 仮想マシンの一般化

イメージを一般化すると、イメージが仮想マシン (VM) にデプロイされる際に、システム固有の設定データがすべて削除されます。

仮想マシンを一般化する前に、Windows の無人インストール後に**sysprep**ツールが応答ファイルを検出できないことを確認する必要があります。

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。

手順

1. OpenShift Container Platform コンソールで、**Virtualization → VirtualMachines** をクリックします。
2. Windows 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration → Disks** をクリックします。
4. **sysprep** ディスクの横にある Options メニュー  をクリックし、**Detach** を選択します。

5. **Detach** をクリックします。
6. **sysprep**ツールによる検出を回避するために、**C:\Windows\Panther\unattend.xml**の名前を変更します。
7. 次のコマンドを実行して、**sysprep** プログラムを開始します。

```
%WINDIR%\System32\Sysprep\sysprep.exe /generalize /shutdown /oobe /mode:vm
```

8. **sysprep**ツールが完了すると、Windows 仮想マシンがシャットダウンします。これで、仮想マシンのディスクイメージを Windows 仮想マシンのインストールイメージとして使用できるようになりました。

これで、仮想マシンを特殊化できます。

10.9.4. Windows 仮想マシンの特殊化

仮想マシン (VM) を特殊化すると、一般化された Windows イメージから VM にコンピューター固有の情報が設定されます。

前提条件

- 一般化された Windows ディスクイメージが必要です。
- **unattend.xml** 応答ファイルを作成する必要があります。詳細は、Microsoft のドキュメント を参照してください。

手順

1. OpenShift Container Platform コンソールで、**Virtualization → Catalog** をクリックします。
2. Windows テンプレートを選択し、**Customize VirtualMachine** をクリックします。
3. **Disk source** リストから **PVC (clone PVC)** を選択します。
4. 一般化された Windows イメージの **Persistent Volume Claim project**および **Persistent Volume Claim name** を指定します。
5. **Review and create VirtualMachine**をクリックします。
6. **Scripts** タブをクリックします。
7. **Sysprep** セクションで、**Edit** をクリックし、**unattend.xml** 応答ファイルを参照して、**Save** をクリックします。
8. **Create VirtualMachine** をクリックします。

Windows は初回起動時に、**unattend.xml** 応答ファイルを使用して VM を特殊化します。これで、仮想マシンを使用する準備が整いました。

10.9.5. 関連情報

- 仮想マシンの作成
- QEMU ゲストエージェントと VirtIO ドライバーのインストール

- Microsoft、Sysprep (Generalize) a Windows installation
- Microsoft、generalize
- Microsoft、specialize

10.10. 障害が発生したノードの解決による仮想マシンのフェイルオーバーのトリガー

ノードに障害が発生し、マシンヘルスチェック がクラスターにデプロイされていない場合、**runStrategy: Always** が設定された仮想マシンは正常なノードに自動的に再配置されません。仮想マシンのフェイルオーバーをトリガーするには、**Node** オブジェクトを手動で削除する必要があります。

注記

インストーラーでプロビジョニングされるインフラストラクチャー を使用してクラスターをインストールし、マシンヘルスチェックを適切に設定している場合は、以下のようになります。

- 障害が発生したノードは自動的に再利用されます。
- **runStrategy** が **Always** または **RerunOnFailure** に設定された仮想マシンは正常なノードで自動的にスケジュールされます。

10.10.1. 前提条件

- 仮想マシンが実行されていたノードに **NotReady** 状態 が設定されている。
- 障害のあるノードで実行されていた仮想マシンでは、 **RunStrategy** が **Always** に設定されている。
- OpenShift CLI (**oc**) がインストールされている。

10.10.2. ベアメタルクラスターからのノードの削除

CLI を使用してノードを削除する場合、ノードオブジェクトは Kubernetes で削除されますが、ノード自体にある Pod は削除されません。レプリケーションコントローラーで管理されないベア Pod は、OpenShift Container Platform からはアクセスできなくなります。レプリケーションコントローラーで管理されるベア Pod は、他の利用可能なノードに再スケジュールされます。ローカルのマニフェスト Pod は削除する必要があります。

手順

以下の手順を実行して、ベアメタルで実行されている OpenShift Container Platform クラスターからノードを削除します。

1. ノードにスケジュール対象外 (unschedulable) のマークを付けます。

```
$ oc adm cordon <node_name>
```

2. ノード上のすべての Pod をドレイン (解放) します。

```
$ oc adm drain <node_name> --force=true
```

このステップは、ノードがオフラインまたは応答しない場合に失敗する可能性があります。ノードが応答しない場合でも、共有ストレージに書き込むワークロードを実行している可能性があります。データの破損を防ぐには、続行する前に物理ハードウェアの電源を切ります。

3. クラスターからノードを削除します。

```
$ oc delete node <node_name>
```

ノードオブジェクトはクラスターから削除されていますが、これは再起動後や kubelet サービスが再起動される場合にクラスターに再び参加することができます。ノードとそのすべてのデータを永続的に削除するには、ノードの使用を停止 する必要があります。

4. 物理ハードウェアを電源を切っている場合は、ノードがクラスターに再度加わるように、そのハードウェアを再びオンに切り替えます。

10.10.3. 仮想マシンのフェイルオーバーの確認

すべてのリソースが正常でないノードで終了すると、移行した仮想マシンのそれぞれについて、新しい仮想マシンインスタンス (VMI) が正常なノードに自動的に作成されます。VMI が作成されていることを確認するには、**oc** CLI を使用してすべての VMI を表示します。

10.10.3.1. CLI を使用した仮想マシンインスタンスのリスト表示

oc コマンドラインインターフェイス (CLI) を使用して、スタンドアロンおよび仮想マシンによって所有されている VMI を含むすべての仮想マシンのリストを表示できます。

手順

- 以下のコマンドを実行して、すべての VMI の一覧を表示します。

```
$ oc get vmis -A
```

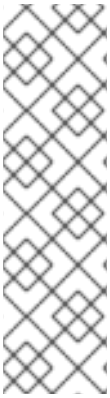
10.11. QEMU ゲストエージェントと VIRTIO ドライバーのインストール

QEMU ゲストエージェント は、仮想マシンで実行され、仮想マシン、ユーザー、ファイルシステム、およびセカンダリーネットワークに関する情報をホストに渡すデーモンです。

10.11.1. QEMU ゲストエージェントのインストール

10.11.1.1. 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 は広範な使用が可能で、Red Hat Enterprise Linux (RHEL) 仮想マシン (VM) においてデフォルトで使用できます。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。



注記

整合性が最も高いオンライン (実行状態) の仮想マシンのスナップショットを作成するには、QEMU ゲストエージェントをインストールします。

QEMU ゲストエージェントは、システムのワークロードに応じて、可能な限り仮想マシンのファイルシステムの休止しようとすることで一貫性のあるスナップショットを取得します。これにより、スナップショットの作成前にインフライトの I/O がディスクに書き込まれるようになります。ゲストエージェントが存在しない場合は、休止はできず、ベストエフォートスナップショットが作成されます。スナップショットの作成条件は、Web コンソールまたは CLI に表示されるスナップショットの指示に反映されます。

手順

1. コンソールのいずれか、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. サービスに永続性があることを確認し、これを起動します。

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

検証

1. 次のコマンドを実行して、**AgentConnected** が VM 仕様にリストされていることを確認します。

```
$ oc get vm <vm_name>
```

10.11.1.2. 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 仮想マシンの場合には、QEMU ゲストエージェントは VirtIO ドライバーに含まれます。既存または新規の Windows インストールにドライバーをインストールします。

注記

整合性が最も高いオンライン (実行状態) の仮想マシンのスナップショットを作成するには、QEMU ゲストエージェントをインストールします。

QEMU ゲストエージェントは、システムのワークロードに応じて、可能な限り仮想マシンのファイルシステムの休止しようとすることで一貫性のあるスナップショットを取得します。これにより、スナップショットの作成前にインフライトの I/O がディスクに書き込まれるようになります。ゲストエージェントが存在しない場合は、休止はできず、ベストエフォートスナップショットが作成されます。スナップショットの作成条件は、Web コンソールまたは CLI に表示されるスナップショットの指示に反映されます。

手順

1. Windows Guest Operating System (OS) で、**File Explorer** を使用して、**virtio-win** CD ドライブの **guest-agent** ディレクトリーに移動します。
2. **qemu-ga-x86_64.msi** インストーラーを実行します。

検証

1. 次のコマンドを実行して、出力に **QEMU Guest Agent** が含まれていることを確認します。

```
$ net start
```

10.11.2. VirtIO ドライバーのインストール

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

表10.1 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。

10.11.2.2. 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 OpenShift 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 Ecosystem 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。

10.11.2.3. 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** をリスト表示します。
   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。

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

Windows のインストール中またはインストール後に、**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 のバージョンに関するドキュメントを参照してください。

前提条件

- **virtio** ドライバーを含むストレージデバイスを仮想マシンに接続している。

手順

1. Windows Guest OS では、**File Explorer** を使用して **virtio-win** CD ドライブに移動します。
2. ダブルクリックして、使用している仮想マシンに適切なインストーラーを実行します。
   a. 64 ビット vCPU の場合は、**virtio-win-gt-x64** インストーラーを使用します。32 ビット

vCPU はサポート対象外になりました。

3. オプション: インストーラーの **Custom Setup** 手順で、インストールするデバイスドライバーを選択します。デフォルトでは、推奨ドライバーセットが選択されています。

4. インストールが完了したら、**Finish** を選択します。

5. 仮想マシンを再起動します。

検証

1. PC でシステムディスクを開きます。通常は **(C:)** です。

2. **Program Files** → **Virtio-Win** に移動します。

Virtio-Win ディレクトリーが存在し、各ドライバーのサブディレクトリーが含まれていればインストールは成功です。

10.11.2.5. 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

OpenShift Virtualization は、Red Hat Ecosystem 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 Ecosystem Catalog からダウンロードすること。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) OpenShift 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、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、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。

10.12. 仮想マシンの QEMU ゲストエージェント情報の表示

QEMU ゲストエージェントが仮想マシンで実行されている場合は、Web コンソールを使用して、仮想マシン、ユーザー、ファイルシステム、およびセカンダリーネットワークに関する情報を表示できます。

QEMU ゲストエージェントがインストールされていないと、**Overview** タブおよび **Details** タブには、仮想マシンの作成時に指定したオペレーティングシステムについての情報が表示されます。

10.12.1. Web コンソールでの QEMU ゲストエージェント情報の表示

Web コンソールを使用して、QEMU ゲストエージェントによってホストに渡される仮想マシンの情報を表示できます。

前提条件

- 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いる。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシン名を選択して、**VirtualMachine details** ページを開きます。
3. **Details** タブをクリックして、アクティブなユーザーを表示します。
4. **Configuration → Disks** タブをクリックして、ファイルシステムに関する情報を表示します。

10.13. 仮想 TRUSTED PLATFORM MODULE デバイスの使用

VirtualMachine (VM) または **VirtualMachineInstance** (VMI) マニフェストを編集して、仮想 Trusted Platform Module (vTPM) デバイスを新規または既存の仮想マシンに追加します。

10.13.1. vTPM デバイスについて

仮想トラステッドプラットフォームモジュール (vTPM) デバイスは、物理トラステッドプラットフォームモジュール (TPM) ハードウェアチップのように機能します。

vTPM デバイスはどのオペレーティングシステムでも使用できますが、Windows 11 をインストールまたは起動するには TPM チップが必要です。vTPM デバイスを使用すると、Windows 11 イメージから作成された VM を物理 TPM チップなしで機能させることができます。

vTPM を有効にしないと、ノードに TPM デバイスがある場合でも、VM は TPM デバイスを認識しません。

また、vTPM デバイスは、物理ハードウェアなしでシークレットを一時的に保存することで、仮想マシンを保護します。ただし、永続的なシークレット ストレージに vTPM を使用することは現在サポートされていません。vTPM は、VM のシャットダウン後に保存されたシークレットを破棄します。

10.13.2. 仮想マシンへの vTPM デバイスの追加

仮想トラステッドプラットフォームモジュール (vTPM) デバイスを仮想マシン (VM) に追加すると、物理 TPM デバイスなしで Windows 11 イメージから作成された仮想マシンを実行できます。vTPM デバイスは、その仮想マシンのシークレットも一時的に保存します。

手順

1. 次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 vm <vm_name>
```

2. **tpm: {}** 行が含まれるように仮想マシン **spec** を編集します。以下に例を示します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          tpm: {} (1)
# ...
```

(1) TPM デバイスを仮想マシンに追加します。

3. 変更を適用するには、エディターを保存し、終了します。

4. オプション: 実行中の仮想マシンを編集している場合は、変更を有効にするためにこれを再起動する必要があります。

10.14. OPENSHIFT PIPELINES を使用した仮想マシンの管理

Red Hat OpenShift Pipelines は、開発者が独自のコンテナーで CI/CD パイプラインの各ステップを設計および実行できるようにする、Kubernetes ネイティブの CI/CD フレームワークです。

Tekton Tasks Operator (TTO) は、OpenShift Virtualization と OpenShift Pipelines を統合します。TTO には、次のことを可能にするクラスタータスクとサンプルパイプラインが含まれています。

- 仮想マシン (VM)、永続ボリューム要求 (PVC)、およびデータボリュームの作成と管理
- 仮想マシンでコマンドを実行する
- **libguestfs** ツールを使用してディスクイメージを操作する

重要

Red Hat OpenShift Pipelines を使用した仮想マシンの管理は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

10.14.1. 前提条件

- **cluster-admin** 権限を使用して OpenShift Container Platform クラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Pipelines CLI がインストールされている。

10.14.2. Tekton Tasks Operator リソースのデプロイ

Tekton Tasks Operator (TTO) クラスタータスクとサンプルパイプラインは、OpenShift Virtualization のインストール時にデフォルトではデプロイされません。TTO リソースをデプロイするには、**HyperConverged** カスタムリソース (CR) で **deployTektonTaskResources** フィーチャーゲートを有効にします。

手順

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

2. **spec.featureGates.deployTektonTaskResources** フィールドを **true** に設定します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: kubevirt-hyperconverged
spec:
  tektonPipelinesNamespace: <user_namespace> 1
  featureGates:
    deployTektonTaskResources: true 2
# ...
```

1. パイプラインが実行される namespace。
2. TTO リソースをデプロイするために有効にするフィーチャーゲート。

注記

後でフィーチャーゲートを無効にしても、クラスタータスクとサンプルパイプラインは引き続き使用できます。

3. 変更を保存し、エディターを終了します。

10.14.3. Tekton Tasks Operator によってサポートされる仮想マシンタスク

次の表は、Tekton Tasks Operator の一部として含まれているクラスタータスクを示しています。

表10.2 Tekton Tasks Operator によってサポートされる仮想マシンタスク

タスク	Description
create-vm-from-template	テンプレートからの仮想マシンの作成
copy-template	仮想マシンテンプレートをコピーします。
modify-vm-template	仮想マシンテンプレートを変更します。
modify-data-object	データボリュームまたはデータソースを作成または削除します。
cleanup-vm	仮想マシンでスクリプトまたはコマンドを実行し、後で仮想マシンを停止または削除します。
disk-virt-customize	**virt-customize** ツールを使用して、ターゲット PVC でカスタマイズスクリプトを実行します。
disk-virt-sysprep	**virt-sysprep** ツールを使用して、ターゲット PVC で sysprep スクリプトを実行します。
wait-for-vmi-status	仮想マシンインスタンスの特定のステータスを待機し、ステータスに基づいて失敗または成功します。

10.14.4. パイプラインの例

Tekton Tasks Operator には、次の **Pipeline** マニフェストの例が含まれています。サンプルパイプラインは、Web コンソールまたは CLI を使用して実行できます。

複数バージョンの Windows が必要な場合は、複数のインストーラーパイプラインを実行する必要がある可能性があります。複数のインストーラーパイプラインを実行する場合、それぞれに **autounattend** config map やベースイメージ名などの一意のパラメーターが必要です。たとえば、Windows 10 および Windows 11 または Windows Server 2022 イメージが必要な場合は、Windows efi インストーラーパイプラインと Windows bios インストーラーパイプラインの両方を実行する必要があります。一方で、Windows 11 および Windows Server 2022 イメージが必要な場合、実行する必要があるのは Windows efi インストーラーパイプラインのみです。

Windows EFI インストーラーパイプライン

このパイプラインは、Windows 11 または Windows Server 2022 を Windows インストールイメージ (ISO ファイル) から新しいデータボリュームにインストールします。インストールプロセスの実行には、カスタムアンサーファイルが使用されます。

Windows BIOS インストーラーパイプライン

このパイプラインは、Windows 10 を Windows インストールイメージ (ISO ファイル) から新しいデータボリュームにインストールします。インストールプロセスの実行には、カスタムアンサーファイルが使用されます。

Windows カスタマイズパイプライン

このパイプラインは、基本的な Windows 10、11、または Windows Server 2022 インストールのデータボリュームを複製し、Microsoft SQL Server Express または Microsoft Visual Studio Code をインストールすることでカスタマイズして、新しいイメージとテンプレートを作成します。

10.14.4.1. Web コンソールを使用してサンプルパイプラインを実行する

サンプルパイプラインは、Web コンソールの **Pipelines** メニューから実行できます。

手順

1. サイドメニューの **Pipelines** → **Pipelines** をクリックします。
2. パイプラインを選択して、**Pipeline details** ページを開きます。
3. **Actions** リストから、**Start** を選択します。**Start Pipeline** ダイアログが表示されます。
4. パラメーターのデフォルト値を保持し、**Start** をクリックしてパイプラインを実行します。**Details** タブでは、各タスクの進行状況が追跡され、パイプラインのステータスが表示されます。

10.14.4.2. CLI を使用してサンプルパイプラインを実行する

PipelineRun リソースを使用して、サンプルパイプラインを実行します。**PipelineRun** オブジェクトは、パイプラインの実行中のインスタンスです。これは、クラスター上の特定の入力、出力、および実行パラメーターで実行されるパイプラインをインスタンス化します。また、パイプライン内のタスクごとに **TaskRun** オブジェクトを作成します。

手順

1. Windows 10 インストーラーパイプラインを実行するには、次の **PipelineRun** マニフェストを作成します。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: windows10-installer-run-
  labels:
    pipelinerun: windows10-installer-run
spec:
  params:
  - name: winImageDownloadURL
    value: <link_to_windows_10_iso> (1)
  pipelineRef:
    name: windows10-installer
  taskRunSpecs:
    - pipelineTaskName: copy-template
```

```
    taskServiceAccountName: copy-template-task
  - pipelineTaskName: modify-vm-template
    taskServiceAccountName: modify-vm-template-task
  - pipelineTaskName: create-vm-from-template
    taskServiceAccountName: create-vm-from-template-task
  - pipelineTaskName: wait-for-vmi-status
    taskServiceAccountName: wait-for-vmi-status-task
  - pipelineTaskName: create-base-dv
    taskServiceAccountName: modify-data-object-task
  - pipelineTaskName: cleanup-vm
    taskServiceAccountName: cleanup-vm-task
status: {}
```

1 Windows 10 64 ビット ISO ファイルの URL を指定します。製品の言語は英語 (米国) でなければなりません。

2. **PipelineRun** マニフェストを適用します。

```
$ oc apply -f windows10-installer-run.yaml
```

3. Windows 10 カスタマイズパイプラインを実行するには、次の **PipelineRun** マニフェストを作成します。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: windows10-customize-run-
  labels:
    pipelinerun: windows10-customize-run
spec:
  params:
    - name: allowReplaceGoldenTemplate
      value: true
    - name: allowReplaceCustomizationTemplate
      value: true
  pipelineRef:
    name: windows10-customize
  taskRunSpecs:
    - pipelineTaskName: copy-template-customize
      taskServiceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template-customize
      taskServiceAccountName: modify-vm-template-task
    - pipelineTaskName: create-vm-from-template
      taskServiceAccountName: create-vm-from-template-task
    - pipelineTaskName: wait-for-vmi-status
      taskServiceAccountName: wait-for-vmi-status-task
    - pipelineTaskName: create-base-dv
      taskServiceAccountName: modify-data-object-task
    - pipelineTaskName: cleanup-vm
      taskServiceAccountName: cleanup-vm-task
    - pipelineTaskName: copy-template-golden
      taskServiceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template-golden
      taskServiceAccountName: modify-vm-template-task
status: {}
```

4. **PipelineRun** マニフェストを適用します。

```
$ oc apply -f windows10-customize-run.yaml
```

10.14.5. 関連情報

- Red Hat OpenShift Pipelines を使用してアプリケーションの CI/CD ソリューションを作成する

10.15. 高度な仮想マシン管理

10.15.1. 仮想マシンのリソースクォータの使用

仮想マシンのリソースクォータの作成および管理

10.15.1.1. 仮想マシンのリソースクォータ制限の設定

リクエストのみを使用するリソースクォータは、仮想マシン (VM) で自動的に機能します。リソースクォータで制限を使用する場合は、VM に手動でリソース制限を設定する必要があります。リソース制限は、リソース要求より少なくとも 100 MiB 大きくする必要があります。

手順

1. **VirtualMachine** マニフェストを編集して、VM の制限を設定します。以下に例を示します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: with-limits
spec:
  running: false
  template:
    spec:
      domain:
# ...
        resources:
          requests:
            memory: 128Mi
          limits:
            memory: 256Mi (1)
```

(1) この設定がサポートされるのは、**limits.memory** 値が **requests.memory** 値より少なくとも **100Mi** 大きいためです。

2. **VirtualMachine** マニフェストを保存します。

10.15.1.2. 関連情報

- プロジェクトごとのリソースクォータ
- 複数のプロジェクト間のリソースクォータ

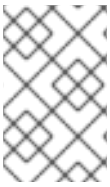
10.15.2. 仮想マシンのノードの指定

ノードの配置ルールを使用して、仮想マシン (VM) を特定のノードに配置することができます。

10.15.2.1. 仮想マシンのノード配置について

仮想マシン (VM) が適切なノードで実行されるようにするには、ノードの配置ルールを設定できます。以下の場合にこれを行うことができます。

- 仮想マシンが複数ある。フォールトトレランスを確保するために、これらを異なるノードで実行する必要がある。
- 2 つの相互間のネットワークトラフィックの多い chatty VM がある。冗長なノード間のルーティングを回避するには、仮想マシンを同じノードで実行します。
- 仮想マシンには、利用可能なすべてのノードにない特定のハードウェア機能が必要です。
- 機能をノードに追加する Pod があり、それらの機能を使用できるように仮想マシンをそのノードに配置する必要があります。



注記

仮想マシンの配置は、ワークロードの既存のノードの配置ルールに基づきます。ワークロードがコンポーネントレベルの特定のノードから除外される場合、仮想マシンはそれらのノードに配置できません。

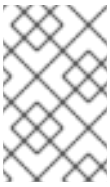
以下のルールタイプは、**VirtualMachine** マニフェストの **spec** フィールドで使用できます。

nodeSelector

仮想マシンは、キーと値のペアまたはこのフィールドで指定したペアを使用してラベルが付けられたノードに Pod をスケジュールできます。ノードには、リスト表示されたすべてのペアに一致するラベルがなければなりません。

affinity

より表現的な構文を使用して、ノードと仮想マシンに一致するルールを設定できます。たとえば、ルールがハード要件ではなく基本設定になるように指定し、ルールの条件が満たされない場合も仮想マシンがスケジュールされるようにすることができます。Pod のアフィニティー、Pod の非アフィニティー、およびノードのアフィニティーは仮想マシンの配置でサポートされます。Pod のアフィニティーは仮想マシンに対して動作します。**VirtualMachine** ワークロードタイプは **Pod** オブジェクトに基づくためです。



注記

アフィニティールールは、スケジューリング時にのみ適用されます。OpenShift Container Platform は、制約を満たさなくなった場合に実行中のワークロードを再スケジューリングしません。

tolerations

一致するテイントを持つノードで仮想マシンをスケジュールできます。テイントがノードに適用される場合、そのノードはテイントを容認する仮想マシンのみを受け入れます。

10.15.2.2. ノード配置の例

以下の YAML スニペットの例では、**nodePlacement**、**affinity**、および **tolerations** フィールドを使用して仮想マシンのノード配置をカスタマイズします。

10.15.2.2.1. 例: nodeSelector を使用した仮想マシンノードの配置

この例では、仮想マシンに **example-key-1 = example-value-1** および **example-key-2 = example-value-2** ラベルの両方が含まれるメタデータのあるノードが必要です。

警告

この説明に該当するノードがない場合、仮想マシンはスケジュールされません。

仮想マシンマニフェストの例

```
metadata:
  name: example-vm-node-selector
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      nodeSelector:
        example-key-1: example-value-1
        example-key-2: example-value-2
# ...
```

10.15.2.2.2. 例: Pod のアフィニティーおよび Pod の非アフィニティーによる仮想マシンノードの配置

この例では、仮想マシンはラベル **example-key-1 = example-value-1** を持つ実行中の Pod のあるノードでスケジュールされる必要があります。このようなノードで実行中の Pod がない場合、仮想マシンはスケジュールされません。

可能な場合に限り、仮想マシンはラベル **example-key-2 = example-value-2** を持つ Pod のあるノードではスケジュールされません。ただし、すべての候補となるノードにこのラベルを持つ Pod がある場合、スケジューラーはこの制約を無視します。

仮想マシンマニフェストの例

```
metadata:
  name: example-vm-pod-affinity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution: (1)
          - labelSelector:
              matchExpressions:
```

```
            - key: example-key-1
              operator: In
              values:
              - example-value-1
            topologyKey: kubernetes.io/hostname
        podAntiAffinity:
          preferredDuringSchedulingIgnoredDuringExecution: 2
          - weight: 100
            podAffinityTerm:
              labelSelector:
                matchExpressions:
                - key: example-key-2
                  operator: In
                  values:
                  - example-value-2
              topologyKey: kubernetes.io/hostname
# ...
```

1 **requiredDuringSchedulingIgnoredDuringExecution** ルールタイプを使用する場合、制約を満たさない場合には仮想マシンはスケジュールされません。

2 **preferredDuringSchedulingIgnoredDuringExecution** ルールタイプを使用する場合、この制約を満たさない場合でも、必要なすべての制約を満たす場合に仮想マシンは依然としてスケジュールされます。

10.15.2.2.3. 例: ノードのアフィニティーによる仮想マシンノードの配置

この例では、仮想マシンはラベル **example.io/example-key = example-value-1** またはラベル **example.io/example-key = example-value-2** を持つノードでスケジュールされる必要があります。この制約は、ラベルのいずれかがノードに存在する場合に満たされます。いずれのラベルも存在しない場合、仮想マシンはスケジュールされません。

可能な場合、スケジューラーはラベル **example-node-label-key = example-node-label-value** を持つノードを回避します。ただし、すべての候補となるノードにこのラベルがある場合、スケジューラーはこの制約を無視します。

仮想マシンマニフェストの例

```
metadata:
  name: example-vm-node-affinity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution: 1
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-key
                operator: In
                values:
                - example-value-1
```

```
          - example-value-2
      preferredDuringSchedulingIgnoredDuringExecution: 2
      - weight: 1
        preference:
          matchExpressions:
          - key: example-node-label-key
            operator: In
            values:
            - example-node-label-value
# ...
```

1 **requiredDuringSchedulingIgnoredDuringExecution** ルールタイプを使用する場合、制約を満たさない場合には仮想マシンはスケジュールされません。

2 **preferredDuringSchedulingIgnoredDuringExecution** ルールタイプを使用する場合、この制約を満たさない場合でも、必要なすべての制約を満たす場合に仮想マシンは依然としてスケジュールされます。

10.15.2.2.4. 例: 容認 (toleration) を使用した仮想マシンノードの配置

この例では、仮想マシン用に予約されるノードには、すでに **key=virtualization:NoSchedule** テイントのラベルが付けられています。この仮想マシンには一致する **tolerations** があるため、これをテイントが付けられたノードにスケジュールできます。

注記

テイントを容認する仮想マシンは、そのテイントを持つノードにスケジュールする必要はありません。

仮想マシンマニフェストの例

```
metadata:
  name: example-vm-tolerations
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  tolerations:
  - key: "key"
    operator: "Equal"
    value: "virtualization"
    effect: "NoSchedule"
# ...
```

10.15.2.3. 関連情報

- Virtualization コンポーネントのノードの指定
- ノードセレクターの使用による特定ノードへの Pod の配置
- ノードのアフィニティールールを使用したノード上での Pod 配置の制御
- ノードテイントを使用した Pod 配置の制御

10.15.3. 証明書ローテーションの設定

証明書ローテーションパラメーターを設定して、既存の証明書を置き換えます。

10.15.3.1. 証明書ローテーションの設定

これは、Web コンソールでの OpenShift Virtualization のインストール時に、または **HyperConverged** カスタムリソース (CR) でインストール後に実行することができます。

手順

1. 以下のコマンドを実行して **HyperConverged** CR を開きます。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

2. 以下の例のように **spec.certConfig** フィールドを編集します。システムのオーバーロードを避けるには、すべての値が 10 分以上であることを確認します。golang **ParseDuration** 形式 に準拠する文字列として、すべての値を表現します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
 name: kubevirt-hyperconverged
 namespace: openshift-cnv
spec:
  certConfig:
    ca:
      duration: 48h0m0s
      renewBefore: 24h0m0s (1)
    server:
      duration: 24h0m0s (2)
      renewBefore: 12h0m0s (3)
```

(1) **ca.renewBefore** の値は **ca.duration** の値以下である必要があります。

(2) **server.duration** の値は **ca.duration** の値以下である必要があります。

(3) **server.renewBefore** の値は **server.duration** の値以下である必要があります。

3. YAML ファイルをクラスターに適用します。

10.15.3.2. 証明書ローテーションパラメーターのトラブルシューティング

1 つ以上の **certConfig** 値を削除すると、デフォルト値が以下のいずれかの条件と競合する場合を除き、デフォルト値に戻ります。

- **ca.renewBefore** の値は **ca.duration** の値以下である必要があります。
- **server.duration** の値は **ca.duration** の値以下である必要があります。
- **server.renewBefore** の値は **server.duration** の値以下である必要があります。

デフォルト値がこれらの条件と競合すると、エラーが発生します。

以下の例で **server.duration** 値を削除すると、デフォルト値の **24h0m0s** は **ca.duration** の値よりも大きくなり、指定された条件と競合します。

例

```
certConfig:
  ca:
    duration: 4h0m0s
    renewBefore: 1h0m0s
  server:
    duration: 4h0m0s
    renewBefore: 4h0m0s
```

これにより、以下のエラーメッセージが表示されます。

```
error: hyperconvergeds.hco.kubevirt.io "kubevirt-hyperconverged" could not be patched: admission webhook "validate-hco.kubevirt.io" denied the request: spec.certConfig: ca.duration is smaller than server.duration
```

エラーメッセージには、最初の競合のみが記載されます。続行する前に、すべての certConfig の値を確認します。

10.15.4. デフォルトの CPU モデルの設定

HyperConverged カスタムリソース (CR) の **defaultCPUModel** 設定を使用して、クラスター全体のデフォルト CPU モデルを定義します。

仮想マシン (VM) の CPU モデルは、仮想マシンおよびクラスター内の CPU モデルの可用性によって異なります。

- 仮想マシンに定義された CPU モデルがない場合:
  - **defaultCPUModel** は、クラスター全体のレベルで定義された CPU モデルを使用して自動的に設定されます。
- 仮想マシンとクラスターの両方に CPU モデルが定義されている場合:
  - 仮想マシンの CPU モデルが優先されます。
- 仮想マシンにもクラスターにも CPU モデルが定義されていない場合:
  - ホストモデルは、ホストレベルで定義された CPU モデルを使用して自動的に設定されます。

10.15.4.1. デフォルトの CPU モデルの設定

HyperConverged カスタムリソース (CR) を更新して、**defaultCPUModel** を設定します。OpenShift Virtualization の実行中に、**defaultCPUModel** を変更できます。

注記

defaultCPUModel では、大文字と小文字が区別されます。

前提条件

- OpenShift CLI (oc) のインストール。

手順

1. 以下のコマンドを実行して **HyperConverged** CR を開きます。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

2. CR に **defaultCPUModel** フィールドを追加し、値をクラスター内に存在する CPU モデルの名前に設定します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
 name: kubevirt-hyperconverged
 namespace: openshift-cnv
spec:
  defaultCPUModel: "EPYC"
```

3. YAML ファイルをクラスターに適用します。

10.15.5. 仮想マシンに UEFI モードを使用する

Unified Extensible Firmware Interface (UEFI) モードで仮想マシン (VM) を起動できます。

10.15.5.1. 仮想マシンの UEFI モードについて

レガシー BIOS などの Unified Extensible Firmware Interface (UEFI) は、コンピューターの起動時にハードウェアコンポーネントやオペレーティングシステムのイメージファイルを初期化します。UEFI は BIOS よりも最新の機能とカスタマイズオプションをサポートするため、起動時間を短縮できます。

これは、**.efi** 拡張子を持つファイルに初期化と起動に関する情報をすべて保存します。このファイルは、EFI System Partition (ESP) と呼ばれる特別なパーティションに保管されます。ESP には、コンピューターにインストールされるオペレーティングシステムのブートローダープログラムも含まれます。

10.15.5.2. UEFI モードでの仮想マシンの起動

VirtualMachine マニフェストを編集して、UEFI モードで起動するように仮想マシンを設定できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. **VirtualMachine** マニフェストファイルを編集または作成します。**spec.firmware.bootloader** スタンザを使用して、UEFI モードを設定します。

セキュアブートがアクティブな状態の UEFI モードでのブート

```
apiversion: kubevirt.io/v1
kind: VirtualMachine
```

```
metadata:
  labels:
    special: vm-secureboot
  name: vm-secureboot
spec:
  template:
    metadata:
      labels:
        special: vm-secureboot
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: containerdisk
        features:
          acpi: {}
          smm:
            enabled: true (1)
        firmware:
          bootloader:
            efi:
              secureBoot: true (2)
# ...
```

1. OpenShift Virtualization では、UEFI モードでセキュアブートを実行するために **SMM** (System Management Mode) を有効にする必要があります。

2. OpenShift Virtualization は、UEFI モードを使用する場合に、セキュアブートの有無に関わらず、仮想マシンをサポートします。セキュアブートが有効な場合には、UEFI モードが必要です。ただし、セキュアブートを使用せずに UEFI モードを有効にできます。

2. 以下のコマンドを実行して、マニフェストをクラスターに適用します。

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

10.15.6. 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 OpenShift 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。

10.15.6.1. 前提条件

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

10.15.6.2. 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ネットワーク接続定義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。
- PXE サーバーがブリッジとして同じ VLAN に接続されている。

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** のネットワーク接続定義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",
        "vlan": 1 (1)
      },
      {
        "type": "cnv-tuning" (2)
      }
    ]
  }'
```

(1) オプション: VLAN タグ。

(2) **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ネットワーク接続定義を作成します。

```
$ 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、インターフェイ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。
bootOrder が **1** に設定されており、インターフェイスが最初に起動することを確認します。この例では、インターフェイスは **<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イ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. 直前に作成されたネットワーク接続定義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>** というネットワーク接続定義に接続されます。

```
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
```

出力例

```
virtualmachineinstance.kubevirt.io "vmi-pxe-boot" created
```

1. 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
```

2. 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
```

3. 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。

4. 仮想マシンインスタンスにログインします。

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

5. 仮想マシンのインターフェイスおよび 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イスに 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。他のインターフェイス **eth0** は OpenShift Container Platform から IP アドレスを取得しています。

```
$ ip addr
```

出力例

```
...
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
```

10.15.6.3. OpenShift Virtualization ネットワークの用語集

OpenShift 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、OpenShift 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。OpenShift 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イ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、Custom 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。

ネットワーク接続定義 (NAD)

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。

ノードネットワーク設定ポリシー (NNCP)

ノードで要求されるネットワーク設定の説明。**NodeNetworkConfigurationPolicy** マニフェストをクラスターに適用して、インターフェイスの追加および削除など、ノードネットワーク設定を更新します。

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イス。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。

10.15.7. 仮想マシンでの Huge Page の使用

Huge Page は、クラスター内の仮想マシンのバッキングメモリーとして使用できます。

10.15.7.1. 前提条件

- ノードに 事前に割り当てられた huge page が設定されている 。

10.15.7.2. Huge Page の機能

メモリーは Page と呼ばれるブロックで管理されます。多くのシステムでは、1 ページは 4Ki です。メモリー 1Mi は 256 ページに、メモリー 1Gi は 256,000 ページに相当します。CPU には、内蔵のメモリー管理ユニットがあり、ハードウェアでこのようなページリストを管理します。トランスレーションルックアサイドバッファー (TLB: Translation Lookaside Buffer) は、仮想から物理へのページマッピングの小規模なハードウェアキャッシュのことです。ハードウェアの指示で渡された仮想アドレスが TLB にあれば、マッピングをすばやく決定できます。そうでない場合には、TLB ミスが発生し、システムは速度が遅く、ソフトウェアベースのアドレス変換にフォールバックされ、パフォーマンスの問題が発生します。TLB のサイズは固定されているので、TLB ミスの発生率を減らすには Page サイズを大きくする必要があります。

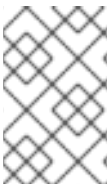
Huge Page とは、4Ki より大きいメモリーページのことです。x86_64 アーキテクチャーでは、2Mi と 1Gi の 2 つが一般的な Huge Page サイズです。別のアーキテクチャーではサイズは異なります。Huge Page を使用するには、アプリケーションが認識できるようにコードを書き込む必要があります。Transparent Huge Page (THP) は、アプリケーションによる認識なしに、Huge Page の管理を自動化しようとしますが、制約があります。特に、ページサイズは 2Mi に制限されます。THP では、THP のデフラグが原因で、メモリー使用率が高くなり、断片化が起こり、パフォーマンスの低下につながり、メモリーページがロックされてしまう可能性があります。このような理由から、アプリケーションは THP ではなく、事前割り当て済みの Huge Page を使用するように設計 (また推奨) される場合があります。

OpenShift Virtualization では、事前に割り当てられた Huge Page を使用できるように仮想マシンを設定できます。

10.15.7.3. 仮想マシンの Huge Page の設定

memory.hugepages.pageSize および **resources.requests.memory** パラメーターを仮想マシン設定に組み込み、仮想マシンを事前に割り当てられた Huge Page を使用するように設定できます。

メモリー要求はページサイズ別に分ける必要があります。たとえば、ページサイズ **1Gi** の場合に **500Mi** メモリーを要求することはできません。



注記

ホストおよびゲスト OS のメモリーレイアウトには関連性がありません。仮想マシンマニフェストで要求される Huge Page が QEMU に適用されます。ゲスト内の Huge Page は、仮想マシンインスタンスの利用可能なメモリー量に基づいてのみ設定できます。

実行中の仮想マシンを編集する場合は、変更を有効にするために仮想マシンを再起動する必要があります。

前提条件

- ノードには、事前に割り当てられた Huge Page が設定されている必要がある。

手順

1. 仮想マシン設定で、**resources.requests.memory** および **memory.hugepages.pageSize** パラメーターを **spec.domain** に追加します。以下の設定スニペットは、ページサイズが **1Gi** の合計 **4Gi** メモリーを要求する仮想マシンに関するものです。

```
kind: VirtualMachine
# ...
spec:
  domain:
    resources:
      requests:
        memory: "4Gi" (1)
    memory:
      hugepages:
        pageSize: "1Gi" (2)
# ...
```

(1) 仮想マシンに要求されるメモリーの合計量。この値はページサイズで分ける必要があります。

(2) 各 Huge Page のサイズ。x86_64 アーキテクチャーの有効な値は **1Gi** および **2Mi** です。ページサイズは要求されたメモリーよりも小さくなければなりません。

2. 仮想マシン設定を適用します。

```
$ oc apply -f <virtual_machine>.yaml
```

10.15.8. 仮想マシン用の専用リソースの有効化

パフォーマンスを向上させるために、CPU などのノードリソースを仮想マシン専用に確保できます。

10.15.8.1. 専用リソースについて

仮想マシンの専用リソースを有効にする場合、仮想マシンのワークロードは他のプロセスで使用されない CPU でスケジュールされます。専用リソースを使用することで、仮想マシンのパフォーマンスとレイテンシーの予測の精度を向上させることができます。

10.15.8.2. 前提条件

- CPU マネージャー がノードに設定されている。仮想マシンのワークロードをスケジュールする前に、ノードに **cpumanager = true** ラベルが設定されていることを確認する。
- 仮想マシンの電源がオフになっている。

10.15.8.3. 仮想マシンの専用リソースの有効化

Details タブで、仮想マシンの専用リソースを有効にすることができます。Red Hat テンプレートから作成された仮想マシンは、専用のリソースで設定できます。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration → Scheduling** タブで、**Dedicated Resources** の横にある編集アイコンをクリックします。
4. **Schedule this workload with dedicated resources (guaranteed policy)**を選択します。
5. **Save** をクリックします。

10.15.9. 仮想マシンのスケジュール

仮想マシンの CPU モデルとポリシー属性が、ノードがサポートする CPU モデルおよびポリシー属性との互換性について一致することを確認して、ノードで仮想マシン (VM) をスケジュールできます。

10.15.9.1. ポリシー属性

仮想マシン (VM) をスケジュールするには、ポリシー属性と、仮想マシンがノードでスケジュールされる際の互換性について一致する CPU 機能を指定します。仮想マシンに指定されるポリシー属性は、その仮想マシンをノードにスケジュールする方法を決定します。

ポリシー属性	説明
force	仮想マシンは強制的にノードでスケジュールされます。これは、ホストの CPU が仮想マシンの CPU に対応していない場合でも該当します。
require	仮想マシンが特定の CPU モデルおよび機能仕様で設定されていない場合に仮想マシンに適用されるデフォルトのポリシー。このデフォルトポリシー属性または他のポリシー属性のいずれかを持つ CPU ノードの検出をサポートするようにノードが設定されていない場合、仮想マシンはそのノードでスケジュールされません。ホストの CPU が仮想マシンの CPU をサポートしているか、ハイパーバイザーが対応している CPU モデルをエミュレートできる必要があります。
optional	仮想マシンがホストの物理マシンの CPU でサポートされている場合は、仮想マシンがノードに追加されます。
disable	仮想マシンは CPU ノードの検出機能と共にスケジュールすることはできません。
forbid	この機能がホストの CPU でサポートされ、CPU ノード検出が有効になっている場合でも、仮想マシンはスケジュールされません。

10.15.9.2. ポリシー属性および CPU 機能の設定

それぞれの仮想マシン (VM) にポリシー属性および CPU 機能を設定して、これがポリシーおよび機能に従ってノードでスケジュールされるようにすることができます。設定する CPU 機能は、ホストの CPU によってサポートされ、またはハイパーバイザーがエミュレートされることを確認するために検証されます。

手順

- 仮想マシン設定ファイルの **domain** 仕様を編集します。以下の例では、仮想マシン (VM) の CPU 機能および **require** ポリシーを設定します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          features:
            - name: apic 1
              policy: require 2
```

1 仮想マシンの名前。

2 仮想マシンのポリシー属性。

10.15.9.3. サポートされている CPU モデルでの仮想マシンのスケジューリング

仮想マシン (VM) の CPU モデルを設定して、CPU モデルがサポートされるノードにこれをスケジュールできます。

手順

- 仮想マシン設定ファイルの **domain** 仕様を編集します。以下の例は、VM 向けに定義された特定の CPU モデルを示しています。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: Conroe 1
```

1 VM の CPU モデル。

10.15.9.4. ホストモデルでの仮想マシンのスケジューリング

仮想マシン (VM) の CPU モデルが **host-model** に設定されている場合、仮想マシンはスケジュールされているノードの CPU モデルを継承します。

手順

- 仮想マシン設定ファイルの **domain** 仕様を編集します。以下の例は、仮想マシンに指定される **host-model** を示しています。

```
apiVersion: kubevirt/v1alpha3
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: host-model 1
```

1 スケジュールされるノードの CPU モデルを継承する仮想マシン。

10.15.10. PCI パススルーの設定

PCI (Peripheral Component Interconnect) パススルー機能を使用すると、仮想マシンからハードウェアデバイスにアクセスし、管理できます。PCI パススルーが設定されると、PCI デバイスはゲストオペレーティングシステムに物理的に接続されているかのように機能します。

クラスター管理者は、**oc** コマンドラインインターフェイス (CLI) を使用して、クラスターでの使用が許可されているホストデバイスを公開および管理できます。

10.15.10.1. PCI パススルー用ホストデバイスの準備について

CLI を使用して PCI パススルー用にホストデバイスを準備するには、**MachineConfig** オブジェクトを作成し、カーネル引数を追加して、Input-Output Memory Management Unit (IOMMU) を有効にします。PCI デバイスを Virtual Function I/O (VFIO) ドライバーにバインドしてから、**HyperConverged** カスタムリソース (CR) の **permittedHostDevices** フィールドを編集してクラスター内で公開します。OpenShift Virtualization Operator を最初にインストールする場合、**permittedHostDevices** のリストは空になります。

CLI を使用してクラスターから PCI ホストデバイスを削除するには、**HyperConverged** CR から PCI デバイス情報を削除します。

10.15.10.1.1. IOMMU ドライバーを有効にするためのカーネル引数の追加

カーネルの IOMMU (Input-Output Memory Management Unit) ドライバーを有効にするには、**MachineConfig** オブジェクトを作成し、カーネル引数を追加します。

前提条件

- 作業用の OpenShift Container Platform クラスターに対する管理者権限が必要です。
- Intel または AMD CPU ハードウェア。
- Intel Virtualization Technology for Directed I/O 拡張または BIOS (Basic Input/Output System) の AMD IOMMU が有効にされている。

手順

1. カーネル引数を識別する **MachineConfig** オブジェクトを作成します。以下の例は、Intel CPU のカーネル引数を示しています。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker (1)
  name: 100-worker-iommu (2)
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
      - intel_iommu=on (3)
# ...
```

(1) 新しいカーネル引数をワーカーノードのみに適用します。

(2) **name** は、マシン設定とその目的におけるこのカーネル引数 (100) のランクを示します。AMD CPU がある場合は、カーネル引数を **amd_iommu=on** として指定します。

(3) Intel CPU の **intel_iommu** としてカーネル引数を特定します。

2. 新規 **MachineConfig** オブジェクトを作成します。

```
$ oc create -f 100-worker-kernel-arg-iommu.yaml
```

検証

- 新規 **MachineConfig** オブジェクトが追加されていることを確認します。

```
$ oc get MachineConfig
```

10.15.10.1.2. PCI デバイスの VFIO ドライバーへのバインディング

PCI デバイスを VFIO (Virtual Function I/O) ドライバーにバインドするには、各デバイスから **vendor-ID** および **device-ID** の値を取得し、これらの値でリストを作成します。リストを **MachineConfig** オブジェクトに追加します。**MachineConfig** Operator は、PCI デバイスを持つノードで **/etc/modprobe.d/vfio.conf** を生成し、PCI デバイスを VFIO ドライバーにバインドします。

前提条件

- カーネル引数を CPU の IOMMU を有効にするために追加している。

手順

1. **lspci** コマンドを実行して、PCI デバイスの **vendor-ID** および **device-ID** を取得します。

```
$ lspci -nnv | grep -i nvidia
```

出力例

-

```
02:01.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB]
[10de:1eb8] (rev a1)
```

2. Butane 設定ファイル **100-worker-vfiopci.bu** を作成し、PCI デバイスを VFIO ドライバーにバインドします。

注記

Butane の詳細は、「Butane を使用したマシン設定の作成」を参照してください。

例

```
variant: openshift
version: 4.13.0
metadata:
  name: 100-worker-vfiopci
  labels:
    machineconfiguration.openshift.io/role: worker (1)
storage:
  files:
  - path: /etc/modprobe.d/vfio.conf
    mode: 0644
    overwrite: true
    contents:
      inline: |
        options vfio-pci ids=10de:1eb8 (2)
  - path: /etc/modules-load.d/vfio-pci.conf (3)
    mode: 0644
    overwrite: true
    contents:
      inline: vfio-pci
```

(1) 新しいカーネル引数をワーカーノードのみに適用します。

(2) 以前に決定された **vendor-ID** 値 (**10de**) と **device-ID** 値 (**1eb8**) を指定して、単一のデバイスを VFIO ドライバーにバインドします。複数のデバイスのリストをベンダーおよびデバイス情報とともに追加できます。

(3) ワーカーノードで vfio-pci カーネルモジュールを読み込むファイル。

3. Butane を使用して、ワーカーノードに配信される設定を含む **MachineConfig** オブジェクトファイル (**100-worker-vfiopci.yaml**) を生成します。

```
$ butane 100-worker-vfiopci.bu -o 100-worker-vfiopci.yaml
```

4. **MachineConfig** オブジェクトをワーカーノードに適用します。

```
$ oc apply -f 100-worker-vfiopci.yaml
```

5. **MachineConfig** オブジェクトが追加されていることを確認します。

```
$ oc get MachineConfig
```

出力例

```
NAME                          GENERATEDBYCONTROLLER                      IGNITIONVERSION
AGE
00-master                     d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0             25h
00-worker                     d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0             25h
01-master-container-runtime   d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-master-kubelet             d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-worker-container-runtime   d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-worker-kubelet             d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
100-worker-iommu                                                         3.2.0             30s
100-worker-vfiopci-configuration                                         3.2.0             30s
```

検証

- VFIO ドライバーがロードされていることを確認します。

```
$ lspci -nnk -d 10de:
```

この出力では、VFIO ドライバーが使用されていることを確認します。

出力例

```
04:00.0 3D controller [0302]: NVIDIA Corporation GP102GL [Tesla P40] [10de:1eb8] (rev a1)
        Subsystem: NVIDIA Corporation Device [10de:1eb8]
        Kernel driver in use: vfio-pci
        Kernel modules: nouveau
```

10.15.10.1.3. CLI を使用したクラスターでの PCI ホストデバイスの公開

クラスターで PCI ホストデバイスを公開するには、PCI デバイスの詳細を **HyperConverged** カスタムリソース (CR) の **spec.permittedHostDevices.pciHostDevices** 配列に追加します。

手順

1. 以下のコマンドを実行して、デフォルトエディターで **HyperConverged** CR を編集します。

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. PCI デバイス情報を **spec.permittedHostDevices.pciHostDevices** 配列に追加します。以下に例を示します。

設定ファイルのサンプル

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
```

```
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  permittedHostDevices: 1
    pciHostDevices: 2
    - pciDeviceSelector: "10DE:1DB6" 3
      resourceName: "nvidia.com/GV100GL_Tesla_V100" 4
    - pciDeviceSelector: "10DE:1EB8"
      resourceName: "nvidia.com/TU104GL_Tesla_T4"
    - pciDeviceSelector: "8086:6F54"
      resourceName: "intel.com/qat"
      externalResourceProvider: true 5
# ...
```

1. クラスターでの使用が許可されているホストデバイス。
2. ノードで利用可能な PCI デバイスのリスト。
3. PCI デバイスを識別するために必要な **vendor-ID** および **device-ID**。
4. PCI ホストデバイスの名前。
5. オプション: このフィールドを **true** に設定すると、リソースが外部デバイスプラグインにより提供されることを示します。OpenShift Virtualization はクラスターでこのデバイスの使用を許可しますが、割り当ておよびモニタリングを外部デバイスプラグインに残します。

注記

上記のスニペットの例は、**nvidia.com/GV100GL_Tesla_V100** および **nvidia.com/TU104GL_Tesla_T4** という名前の 2 つの PCI ホストデバイスが、**HyperConverged** CR の許可されたホストデバイスの一覧に追加されたことを示しています。これらのデバイスは、OpenShift Virtualization と動作することがテストおよび検証されています。

3. 変更を保存し、エディターを終了します。

検証

- 以下のコマンドを実行して、PCI ホストデバイスがノードに追加されたことを確認します。この出力例は、各デバイスが **nvidia.com/GV100GL_Tesla_V100**、**nvidia.com/TU104GL_Tesla_T4**、および **intel.com/qat** のリソース名にそれぞれ関連付けられたデバイスが 1 つあることを示しています。

```
$ oc describe node <node_name>
```

出力例

```
Capacity:
  cpu:                64
  devices.kubevirt.io/kvm:        110
  devices.kubevirt.io/tun:        110
  devices.kubevirt.io/vhost-net:  110
```

```
  ephemeral-storage:             915128Mi
  hugepages-1Gi:                 0
  hugepages-2Mi:                 0
  memory:                        131395264Ki
  nvidia.com/GV100GL_Tesla_V100  1
  nvidia.com/TU104GL_Tesla_T4    1
  intel.com/qat:                 1
  pods:                          250
Allocatable:
  cpu:                           63500m
  devices.kubevirt.io/kvm:       110
  devices.kubevirt.io/tun:       110
  devices.kubevirt.io/vhost-net: 110
  ephemeral-storage:             863623130526
  hugepages-1Gi:                 0
  hugepages-2Mi:                 0
  memory:                        130244288Ki
  nvidia.com/GV100GL_Tesla_V100  1
  nvidia.com/TU104GL_Tesla_T4    1
  intel.com/qat:                 1
  pods:                          250
```

10.15.10.1.4. CLI を使用したクラスターからの PCI ホストデバイスの削除

クラスターから PCI ホストデバイスを削除するには、**HyperConverged** カスタムリソース (CR) からそのデバイスの情報を削除します。

手順

1. 以下のコマンドを実行して、デフォルトエディターで **HyperConverged** CR を編集します。

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 適切なデバイスの **pciDeviceSelector**、**resourceName**、および **externalResourceProvider** (該当する場合) のフィールドを削除して、**spec.permittedHostDevices.pciHostDevices** 配列から PCI デバイス情報を削除します。この例では、**intel.com/qat** リソースが削除されました。

設定ファイルのサンプル

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  permittedHostDevices:
    pciHostDevices:
    - pciDeviceSelector: "10DE:1DB6"
      resourceName: "nvidia.com/GV100GL_Tesla_V100"
    - pciDeviceSelector: "10DE:1EB8"
      resourceName: "nvidia.com/TU104GL_Tesla_T4"
  # ...
```

3. 変更を保存し、エディターを終了します。

検証

- 以下のコマンドを実行して、PCI ホストデバイスがノードから削除されたことを確認します。この出力例は、**intel.com/qat** リソース名に関連付けられているデバイスがゼロであることを示しています。

```
$ oc describe node <node_name>
```

出力例

```
Capacity:
  cpu:                          64
  devices.kubevirt.io/kvm:      110
  devices.kubevirt.io/tun:      110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:            915128Mi
  hugepages-1Gi:                0
  hugepages-2Mi:                0
  memory:                       131395264Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                0
  pods:                         250
Allocatable:
  cpu:                          63500m
  devices.kubevirt.io/kvm:      110
  devices.kubevirt.io/tun:      110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:            863623130526
  hugepages-1Gi:                0
  hugepages-2Mi:                0
  memory:                       130244288Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                0
  pods:                         250
```

10.15.10.2. PCI パススルー用の仮想マシンの設定

PCI デバイスがクラスターに追加された後に、それらを仮想マシンに割り当てることができます。PCI デバイスが仮想マシンに物理的に接続されているかのような状態で利用できるようになりました。

10.15.10.2.1. PCI デバイスの仮想マシンへの割り当て

PCI デバイスがクラスターで利用可能な場合、これを仮想マシンに割り当て、PCI パススルーを有効にすることができます。

手順

- PCI デバイスをホストデバイスとして仮想マシンに割り当てます。

例

```
apiVersion: kubevirt.io/v1
```

```
kind: VirtualMachine
spec:
  domain:
    devices:
      hostDevices:
      - deviceName: nvidia.com/TU104GL_Tesla_T4 1
        name: hostdevices1
```

1 クラスターでホストデバイスとして許可される PCI デバイスの名前。仮想マシンがこのホストデバイスにアクセスできます。

検証

- 以下のコマンドを使用して、ホストデバイスが仮想マシンから利用可能であることを確認します。

```
$ lspci -nnk | grep NVIDIA
```

出力例

```
$ 02:01.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB]
[10de:1eb8] (rev a1)
```

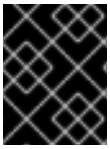
10.15.10.3. 関連情報

- BIOS での Intel VT-X および AMD-V Virtualization ハードウェア拡張の有効化
- ファイル権限の管理
- インストール後のマシン設定タスク

10.15.11. 仮想 GPU パススルーの設定

仮想マシンは仮想 GPU (vGPU) ハードウェアにアクセスできます。仮想マシンに仮想 GPU を割り当てると、次のことが可能になります。

- 基盤となるハードウェアの GPU の一部にアクセスして、仮想マシンで高いパフォーマンスのメリットを実現する。
- リソースを大量に消費する I/O 操作を合理化する。



重要

仮想 GPU パススルーは、ベアメタル環境で実行されているクラスターに接続されているデバイスにのみ割り当てることができます。

10.15.11.1. 仮想マシンへの vGPU パススルーデバイスの割り当て

Open Shift Container Platform Web コンソールを使用して、vGPU パススルーデバイスを仮想マシンに割り当てます。

前提条件

- 仮想マシンを停止する必要があります。

手順

1. Open Shift Container Platform Web コンソールで、サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. デバイスを割り当てる仮想マシンを選択します。
3. **Details** タブで、**GPU devices** をクリックします。
   vGPU デバイスをホストデバイスとして追加すると、VNC コンソールでデバイスにアクセスすることはできません。
4. **Add GPU device** をクリックし、**Name** を入力して、**Device name** リストからデバイスを選択します。
5. **Save** をクリックします。
6. **YAML**タブをクリックして、クラスター設定の **hostDevices**セクションに新しいデバイスが追加されていることを確認します。

注記

カスタマイズされたテンプレートまたは YAML ファイルから作成された仮想マシンに、ハードウェアデバイスを追加できます。Windows 10 や RHEL 7 などの特定のオペレーティングシステム用に事前に提供されているブートソーステンプレートにデバイスを追加することはできません。

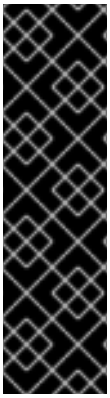
クラスターに接続されているリソースを表示するには、サイドメニューから **Compute → Hardware Devices** をクリックします。

10.15.11.2. 関連情報

- 仮想マシンの作成
- 仮想マシンテンプレートの作成

10.15.12. 仲介デバイスの設定

HyperConvergedカスタムリソース (CR) でデバイスのリストを提供すると、Open Shift Virtualization は仮想 GPU (vGPU) などの仲介デバイスを自動的に作成します。



重要

仲介デバイスの宣言型設定は、テクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

10.15.12.1. NVIDIA GPU Operator の使用について

NVIDIA GPU Operator は、OpenShift Container Platform クラスターで NVIDIA GPU リソースを管理し、GPU ノードのブートストラップに関連するタスクを自動化します。GPU はクラスター内の特別なリソースであるため、アプリケーションワークロードを GPU にデプロイするには、事前にいくつかのコンポーネントをインストールする必要があります。これらのコンポーネントには、コンピューティングユニファイドデバイスアーキテクチャー (CUDA)、Kubernetes デバイスプラグイン、コンテナーランタイムに加え、自動ノードラベル付け、監視などを可能にする NVIDIA ドライバーなどの機能も含まれます。

注記

NVIDIA GPU Operator は、NVIDIA によってのみサポートされています。NVIDIA からサポートを受ける方法は、Obtaining Support from NVIDIA を参照してください。

OpenShift Container Platform OpenShift Virtualization で GPU を有効にする方法は、OpenShift Container Platform ネイティブの方法と、NVIDIA GPU Operator を使用する方法の 2 つあります。ここでは、OpenShift Container Platform ネイティブの方法を説明します。

NVIDIA GPU Operator は、OpenShift Container Platform OpenShift Virtualization を使用して、OpenShift Container Platform 上で実行される仮想化ワークロードに GPU をプロビジョニングする Kubernetes Operator です。Operator を使用すると、GPU 対応の仮想マシンを簡単にプロビジョニングおよび管理し、複雑な人工知能/機械学習 (AI/ML) ワークロードを他のワークロードと同じプラットフォーム上で実行できます。また、Operator を使用することでインフラストラクチャーの GPU 容量を簡単に拡張できGPU ベースのワークロードの急速な成長が可能になります。

NVIDIA GPU Operator を使用して、GPU で高速化された VM を実行するためのワーカーノードをプロビジョニングする方法の詳細は、NVIDIA GPU Operator with OpenShift Virtualization を参照してください。

10.15.12.2. OpenShift Virtualization での仮想 GPU の使用について

一部のグラフィックス処理ユニット (GPU) カードは、仮想 GPU (vGPU) の作成をサポートしています。管理者が **HyperConverged** カスタムリソース (CR) で設定の詳細を提供すると、OpenShift Virtualization は仮想 GPU およびその他の仲介デバイスを自動的に作成できます。この自動化は、大規模なクラスターで特に役立ちます。

注記

機能とサポートの詳細は、ハードウェアベンダーのドキュメントを参照してください。

仲介デバイス

1 つまたは複数の仮想デバイスに分割された物理デバイス。仮想 GPU は、仲介デバイス (mdev) の一種です。物理 GPU のパフォーマンスが、仮想デバイス間で分割されます。仲介デバイスを 1 つまたは複数の仮想マシン (VM) に割り当てることができますが、ゲストの数は GPU と互換性がある必要があります。一部の GPU は複数のゲストをサポートしていません。

10.15.12.2.1. 前提条件

- ハードウェアベンダーがドライバーを提供している場合は、仲介デバイスを作成するノードにドライバーをインストールしている。
  - NVIDIA カードを使用する場合は、NVIDIAGRID ドライバーをインストールしている。

10.15.12.2.2. 設定の概要

仲介デバイスを設定する場合、管理者は次のタスクを完了する必要があります。

- 仲介デバイスを作成する。
- 仲介デバイスをクラスターに公開する。

HyperConverged CR には、両方のタスクを実行する API が含まれています。

仲介デバイスの作成

```
# ...
spec:
  mediatedDevicesConfiguration:
    mediatedDevicesTypes: 1
    - <device_type>
    nodeMediatedDeviceTypes: 2
    - mediatedDevicesTypes: 3
      - <device_type>
      nodeSelector: 4
        <node_selector_key>: <node_selector_value>
# ...
```

1. 必須: クラスターのグローバル設定を定義します。
2. オプション: 特定のノードまたはノードのグループのグローバル設定をオーバーライドします。グローバルの **mediatedDevicesTypes** 設定と併用する必要があります。
3. **nodeMediatedDeviceTypes** を使用する場合に必須です。指定されたノードのグローバル **MediedDevicesTypes** 設定をオーバーライドします。
4. **nodeMediatedDeviceTypes** を使用する場合に必須です。**key:value** ペアを含める必要があります。

仲介デバイスのクラスターへの公開

```
# ...
  permittedHostDevices:
    mediatedDevices:
    - mdevNameSelector: GRID T4-2Q 1
      resourceName: nvidia.com/GRID_T4-2Q 2
# ...
```

1. この値にマッピングする仲介デバイスをホスト上に公開します。

注記

実際のシステムの正しい値に置き換えて、**/sys/bus/pci/devices/<slot>:<bus>:<domain>.<function>/mdev_supported_types/<type>/name** の内容を表示し、デバイスがサポートする仲介デバイスのタイプを確認できます。

たとえば、**nvidia-231** タイプの name ファイルには、セレクター文字列 **GRID T4-2Q** が含まれます。**GRID T4-2Q** を **mdevNameSelector** 値として使用することで、ノードは **nvidia-231** タイプを使用できます。

2 **resourceName** は、ノードに割り当てられたものと一致する必要があります。次のコマンドを使用して、**resourceName** を検索します。

```
$ oc get $NODE -o json \
  | jq '.status.allocatable \
    | with_entries(select(.key | startswith("nvidia.com/"))) \
    | with_entries(select(.value != "0"))'
```

10.15.12.2.3. 仮想 GPU がノードに割り当てられる方法

物理デバイスごとに、OpenShift Virtualization は以下の値を設定します。

- 1 つの mdev タイプ。
- 選択した **mdev** タイプのインスタンスの最大数。

クラスターのアーキテクチャーは、デバイスの作成およびノードへの割り当て方法に影響します。

ノードごとに複数のカードを持つ大規模なクラスター

同様の仮想 GPU タイプに対応する複数のカードを持つノードでは、関連するデバイス種別がラウンドロビン方式で作成されます。以下に例を示します。

```
# ...
mediatedDevicesConfiguration:
  mediatedDevicesTypes:
  - nvidia-222
  - nvidia-228
  - nvidia-105
  - nvidia-108
# ...
```

このシナリオでは、各ノードに以下の仮想 GPU 種別に対応するカードが 2 つあります。

```
nvidia-105
# ...
nvidia-108
nvidia-217
nvidia-299
# ...
```

各ノードで、OpenShift Virtualization は以下の vGPU を作成します。

- 最初のカード上に nvidia-105 タイプの 16 の仮想 GPU
- 2 番目のカード上に nvidia-108 タイプの 2 つの仮想 GPU

1 つのノードに、要求された複数の仮想 GPU タイプをサポートするカードが 1 つある

OpenShift Virtualization は、**mediatedDevicesTypes** 一覧の最初のサポートされるタイプを使用します。
たとえば、ノードカードのカードは **nvidia-223** と **nvidia-224** をサポートします。以下の **mediatedDevicesTypes** 一覧が設定されます。

```
# ...
```

```
mediatedDevicesConfiguration:
  mediatedDevicesTypes:
  - nvidia-22
  - nvidia-223
  - nvidia-224
# ...
```

この例では、OpenShift Virtualization は **nvidia-223** タイプを使用します。

10.15.12.2.4. 仲介デバイスの変更および削除について

クラスターの仲介デバイス設定は、次の方法を使用して OpenShift Virtualization で更新できます。

- **HyperConverged** CR を編集し、**mediadDevicesTypes** スタンザの内容を変更します。
- **nodeMediatedDeviceTypes** ノードセレクターに一致するノードラベルを変更します。
- **HyperConverged** CR の **spec.mediaDevicesConfiguration** および **spec.permittedHostDevices** スタンザからデバイス情報を削除します。

注記

spec.permittedHostDevices スタンザからデバイス情報を削除したが、**spec.mediatedDevicesConfiguration** スタンザからは削除しなかった場合、同じノードで新規の仲介デバイスタイプを作成することはできません。仲介デバイスを適切に削除するには、両方のスタンザからデバイス情報を削除します。

具体的な変更に応じて、これらのアクションにより、OpenShift Virtualization は仲介デバイスを再設定するか、クラスターノードからそれらを削除します。

10.15.12.2.5. 仲介デバイス用のホストの準備

仲介デバイスを設定する前に、入出力メモリー管理ユニット (IOMMU) ドライバーを有効にする必要があります。

10.15.12.2.5.1. IOMMU ドライバーを有効にするためのカーネル引数の追加

カーネルの IOMMU (Input-Output Memory Management Unit) ドライバーを有効にするには、**MachineConfig** オブジェクトを作成し、カーネル引数を追加します。

前提条件

- 作業用の OpenShift Container Platform クラスターに対する管理者権限が必要です。
- Intel または AMD CPU ハードウェア。
- Intel Virtualization Technology for Directed I/O 拡張または BIOS (Basic Input/Output System) の AMD IOMMU が有効にされている。

手順

1. カーネル引数を識別する **MachineConfig** オブジェクトを作成します。以下の例は、Intel CPU のカーネル引数を示しています。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker 1
  name: 100-worker-iommu 2
spec:
  config:
    ignition:
      version: 3.2.0
  kernelArguments:
      - intel_iommu=on 3
# ...
```

1. 新しいカーネル引数をワーカーノードのみに適用します。
2. **name** は、マシン設定とその目的におけるこのカーネル引数 (100) のランクを示します。AMD CPU がある場合は、カーネル引数を **amd_iommu=on** として指定します。
3. Intel CPU の **intel_iommu** としてカーネル引数を特定します。

2. 新規 **MachineConfig** オブジェクトを作成します。

```
$ oc create -f 100-worker-kernel-arg-iommu.yaml
```

検証

- 新規 **MachineConfig** オブジェクトが追加されていることを確認します。

```
$ oc get MachineConfig
```

10.15.12.2.6. 仲介デバイスの追加および削除

仲介デバイスを追加または削除できます。

10.15.12.2.6.1. 仲介デバイスの作成および公開

HyperConverged カスタムリソース (CR) を編集して、仮想 GPU (vGPU) などの仲介デバイスを公開し、作成できます。

前提条件

- IOMMU (Input-Output Memory Management Unit) ドライバーを有効にしている。

手順

1. 以下のコマンドを実行して、デフォルトエディターで **HyperConverged** CR を編集します。

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 仲介デバイス情報を **HyperConverged** CR の**spec** に追加し、**mediatedDevicesConfiguration** および **permittedHostDevices** スタンザが含まれるようにします。以下に例を示します。

設定ファイルのサンプル

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  mediatedDevicesConfiguration: <.>
    mediatedDevicesTypes: <.>
    - nvidia-231
    nodeMediatedDeviceTypes: <.>
    - mediatedDevicesTypes: <.>
      - nvidia-233
      nodeSelector:
        kubernetes.io/hostname: node-11.redhat.com
  permittedHostDevices: <.>
    mediatedDevices:
    - mdevNameSelector: GRID T4-2Q
      resourceName: nvidia.com/GRID_T4-2Q
    - mdevNameSelector: GRID T4-8Q
      resourceName: nvidia.com/GRID_T4-8Q
# ...
```

<.> 仲介デバイスを作成します。<.> 必須: グローバル **MediedDevicesTypes** 設定。<.> 任意: 特定のノードのグローバル設定をオーバーライドします。<.> **nodeMediatedDeviceTypes** を使用する場合は必須。<.> 仲介デバイスをクラスターに公開します。

3. 変更を保存し、エディターを終了します。

検証

- 以下のコマンドを実行して、デバイスが特定のノードに追加されたことを確認できます。

```
$ oc describe node <node_name>
```

10.15.12.2.6.2. CLI を使用したクラスターからの仲介デバイスの削除

クラスターから仲介デバイスを削除するには、**HyperConverged** カスタムリソース (CR) からそのデバイスの情報を削除します。

手順

1. 以下のコマンドを実行して、デフォルトエディターで **HyperConverged** CR を編集します。

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **HyperConverged** CR の **spec.mediatedDevicesConfiguration** および **spec.permittedHostDevices** スタンザからデバイス情報を削除します。両方のエントリーを削除すると、後で同じノードで新しい仲介デバイスタイプを作成できます。以下に例を示します。

設定ファイルのサンプル

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  mediatedDevicesConfiguration:
    mediatedDevicesTypes: 1
      - nvidia-231
  permittedHostDevices:
    mediatedDevices: 2
    - mdevNameSelector: GRID T4-2Q
      resourceName: nvidia.com/GRID_T4-2Q
```

1. **nvidia-231** デバイスタイプを削除するには、これを **mediatedDevicesTypes** 配列から削除します。
2. **GRID T4-2Q** デバイスを削除するには、**mdevNameSelector** フィールドおよび対応する **resourceName** フィールドを削除します。

3. 変更を保存し、エディターを終了します。

10.15.12.3. 仲介デバイスの使用

vGPU は仲介デバイスの一種です。物理 GPU のパフォーマンスは仮想デバイス間で分割されます。仲介デバイスを 1 つ以上の仮想マシンに割り当てることができます。

10.15.12.3.1. 仮想マシンへの仲介デバイスの割り当て

仮想 GPU (vGPU) などの仲介デバイスを仮想マシンに割り当てます。

前提条件

- 仲介デバイスが **HyperConverged** カスタムリソースで設定されている。

手順

- **VirtualMachine** マニフェストの **spec.domain.devices.gpus** スタンザを編集して、仲介デバイスを仮想マシン (VM) に割り当てます。

  仮想マシンマニフェストの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  domain:
    devices:
      gpus:
      - deviceName: nvidia.com/TU104GL_Tesla_T4 1
        name: gpu1 2
      - deviceName: nvidia.com/GRID_T4-1Q
        name: gpu2
```

1. 仲介デバイスに関連付けられたリソース名。
2. 仮想マシン上のデバイスを識別する名前。

検証

- デバイスが仮想マシンで利用できることを確認するには、**<device_name>** を **VirtualMachine** マニフェストの **deviceName** の値に置き換えて以下のコマンドを実行します。

```
$ lspci -nnk | grep <device_name>
```

10.15.12.4. 関連情報

- BIOS での Intel VT-X および AMD-V Virtualization ハードウェア拡張の有効化

10.15.13. 仮想マシンでの Descheduler エビクションの有効化

Descheduler を使用して Pod を削除し、Pod をより適切なノードに再スケジュールできます。Pod が仮想マシンの場合、Pod の削除により、仮想マシンが別のノードにライブマイグレーションされます。

重要

仮想マシンの Descheduler エビクションはテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

10.15.13.1. Descheduler プロファイル

テクノロジープレビューの **DevPreviewLongLifecycle** プロファイルを使用して、仮想マシンで Descheduler を有効にします。これは、現在 OpenShift Virtualization で利用可能な唯一の Descheduler プロファイルです。適切なスケジューリングを確保するには、予想される負荷に応じた CPU およびメモリー要求で仮想マシンを作成します。

DevPreviewLongLifecycle

このプロファイルは、ノード間のリソース使用率のバランスを取り、以下のストラテジーを有効にします。

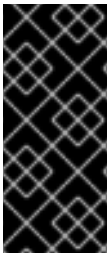
- **RemovePodsHavingTooManyRestarts**: コンテナーが何度も再起動された Pod、およびすべてのコンテナー (Init コンテナーを含む) の再起動の合計が 100 を超える Pod を削除します。仮想マシンのゲストオペレーティングシステムを再起動しても、この数は増えません。
- **LowNodeUtilization**: 使用率の低いノードがある場合に、使用率の高いノードから Pod をエビクトします。エビクトされた Pod の宛先ノードはスケジューラーによって決定されます。
  - ノードは、使用率がすべてのしきい値 (CPU、メモリー、Pod の数) について 20% 未満の場合に使用率が低いと見なされます。

- ノードは、使用率がすべてのしきい値 (CPU、メモリー、Pod の数) について 50% を超える場合に過剰に使用されていると見なされます。

10.15.13.2. Descheduler のインストール

Descheduler はデフォルトで利用できません。Descheduler を有効にするには、Kube Descheduler Operator を OperatorHub からインストールし、1 つ以上の Descheduler プロファイルを有効にする必要があります。

デフォルトで、Descheduler は予測モードで実行されます。つまり、これは Pod エビクションのみをシミュレートします。Pod エビクションを実行するには、Descheduler のモードを automatic に変更する必要があります。



重要

クラスターでホストされたコントロールプレーンを有効にしている場合は、カスタム優先度のしきい値を設定して、ホストされたコントロールプレーンの namespace の Pod が削除される可能性を下げます。ホストされたコントロールプレーンの優先度クラスの中で優先度値が最も低い (**100000000**) ため、優先度しきい値クラス名を **hypershift-control-plane** に設定します。

前提条件

- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform にログインしている。
- OpenShift Container Platform Web コンソールにアクセスできる。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. Kube Descheduler Operator に必要な namespace を作成します。
   a. **Administration → Namespaces** に移動し、**Create Namespace** をクリックします。
   b. **Name** フィールドに **openshift-kube-descheduler-operator** を入力し、**Labels** フィールドに **openshift.io/cluster-monitoring=true** を入力して Descheduler メトリックを有効にし、**Create** をクリックします。
3. Kube Descheduler Operator をインストールします。
   a. **Operators → OperatorHub** に移動します。
   b. **Kube Descheduler Operator** をフィルターボックスに入力します。
   c. **Kube Descheduler Operator** を選択し、**Install** をクリックします。
   d. **Install Operator** ページで、**A specific namespace on the cluster**を選択します。ドロップダウンメニューから **openshift-kube-descheduler-operator** を選択します。
   e. **Update Channel** および **Approval Strategy** の値を必要な値に調整します。
   f. **Install** をクリックします。

4. Descheduler インスタンスを作成します。

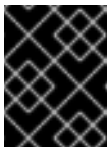
   a. **Operators → Installed Operators** ページから、**Kube Descheduler Operator** をクリックします。

   b. **Kube Descheduler** タブを選択し、**Create KubeDescheduler** をクリックします。

   c. 必要に応じて設定を編集します。

      i. エビクションをシミュレーションせずに Pod をエビクトするには、**Mode** フィールドを **Automatic** に変更します。

      ii. **Profiles** セクションを展開し、**DevPreviewLongLifecycle** を選択します。**AffinityAndTaints** プロファイルがデフォルトで有効になっています。



重要

OpenShift Virtualization で現在利用できるプロファイルは **DevPreviewLongLifecycle** のみです。

また、後で OpenShift CLI (**oc**) を使用して、Descheduler のプロファイルおよび設定を設定することもできます。

10.15.13.3. 仮想マシン (VM) での Descheduler エビクションの有効化

Descheduler のインストール後に、アノテーションを **VirtualMachine** カスタムリソース (CR) に追加して Descheduler エビクションを仮想マシンで有効にできます。

前提条件

- Descheduler を OpenShift Container Platform Web コンソールまたは OpenShift CLI (**oc**) にインストールしている。
- 仮想マシンが実行されていないことを確認します。

手順

1. 仮想マシンを起動する前に、**descheduler.alpha.kubernetes.io/evict** アノテーションを **VirtualMachine** CR に追加します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    metadata:
      annotations:
        descheduler.alpha.kubernetes.io/evict: "true"
```

2. インストール時に Web コンソールで **DevPreviewLongLifecycle** プロファイルをまだ設定していない場合は、**KubeDescheduler** オブジェクトの **spec.profile** セクションに **DevPreviewLongLifecycle** を指定します。

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
```

```
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600
  profiles:
  - DevPreviewLongLifecycle
  mode: Predictive 1
```

1 デフォルトでは、Descheduler は Pod をエビクトしません。Pod をエビクトするには、**mode** を **Automatic** に設定します。

Descheduler が仮想マシンで有効になりました。

10.15.13.4. 関連情報

- Descheduler を使用した Pod のエビクト

10.16. 仮想マシンのインポート

10.16.1. データボリュームインポートの TLS 証明書

10.16.1.1. データボリューム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 証明書を設定マップに追加する必要があります。この設定マップは、宛先データボリュームの 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設定マップ を作成します。

手順

1. 正しい namespace にあることを確認します。設定マップは、同じ namespace にある場合にデータボリュームによってのみ参照されます。

```
$ oc get ns
```

2. 設定マップを作成します。

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

10.16.1.2. 例: TLS 証明書から作成される設定マップ

以下は、**ca.pem** TLS 証明書で作成される設定マップの例です。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: tls-certs
data:
  ca.pem: |
```

```
-----BEGIN CERTIFICATE-----
... <base64 encoded cert> ...
-----END CERTIFICATE-----
```

10.16.2. データボリューム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 クラスターストレージにインポートできます。Containerized Data Importer (CDI) を使用すると、データボリュームを使用してイメージを永続ボリューム要求 (PVC) にインポートできます。OpenShift Virtualization は 1 つ以上のデータボリュームを使用して、データのインポートと基盤となる PVC の作成を自動化します。次に、データボリュームを永続ストレージの仮想マシンに割り当てることができます。

仮想マシンイメージは、HTTP または HTTPS エンドポイントでホストするか、コンテナーディスクに組み込み、コンテナーレジストリーで保存できます。

重要

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。

10.16.2.1. 前提条件

- エンドポイントに TLS 証明書が必要な場合、TLS 証明書がデータボリュームと同じ namespace の config map に組み込まれていて、データボリューム設定で参照されている。
- コンテナーディスクをインポートするには、以下を実行すること。
  - 仮想マシンイメージからコンテナーディスクを準備 し、これをコンテナーレジストリーに保存してからインポートする必要がある場合があります。
  - コンテナーレジストリーに TLS がない場合、レジストリーを **HyperConverged** カスタムリソースの **insecureRegistries** フィールドに追加 し、ここからコンテナーディスクをインポートできます。
- この操作を正常に完了するためには、ストレージクラスを定義するか、CDI スクラッチ領域を用意 しなければならない場合があります。

データボリュームを含むブロックストレージに仮想マシンイメージをインポートする場合は、使用可能なローカルブロック永続ボリュームが必要です。

10.16.2.2. 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*

✓ 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

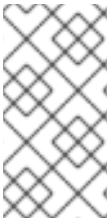
**カスタム認証局が必要な場合にスクラッチ領域が必要

注記

CDI は OpenShift Container Platform の クラスター全体のプロキシー設定 を使用するようになりました。

10.16.2.3. データボリュームについて

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。データボリュームは、基礎となる永続ボリューム要求 (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。データボリュームは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。



注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.16.2.4. ローカルブロック永続ボリューム

データボリュームを含むブロックストレージに仮想マシンイメージをインポートする場合は、使用可能なローカルブロック永続ボリュームが必要です。

10.16.2.4.1. ブロック永続ボリュームについて

ブロック永続ボリューム (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、オーバーヘッドを削減することで、仮想マシンのパフォーマンス上の利点をもたらすことができます。

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

10.16.2.4.2. ローカルブロック永続ボリュームの作成

データボリュームを含むブロックストレージに仮想マシンイメージをインポートする場合は、使用可能なローカルブロック永続ボリュームが必要です。

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永続ボリューム (PV) を作成します。次に、このループデバイスを PV マニフェストで **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** マニフェストを作成します。

```
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 にストレージクラスを設定します。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永続ボリュームのファイル名。

10.16.2.5. データボリュームを使用して仮想マシンイメージをストレージにインポートする

データボリュームを使用して、仮想マシンイメージをストレージにインポートできます。

仮想マシンイメージは、HTTP または HTTPS エンドポイントでホストするか、イメージをコンテナーディスクに組み込み、コンテナーレジストリーで保存できます。

イメージのデータソースは、**VirtualMachine** 設定ファイルで指定します。仮想マシンが作成されると、仮想マシンイメージを含むデータボリュームがストレージにインポートされます。

前提条件

- 仮想マシンイメージをインポートするには、以下が必要である。
  - 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 HTTPS エンドポイント
- コンテナーディスクをインポートするには、仮想マシンイメージをコンテナーディスクに組み込み、データソースにアクセスするために必要な認証クレデンシャルとともにコンテナーレジストリーに保存する必要があります。
- 仮想マシンが自己署名証明書またはシステム CA バンドルによって署名されていない証明書を使用するサーバーと通信する必要がある場合は、データボリュームと同じ namespace に config map を作成する必要があります。

手順

1. データソースに認証が必要な場合は、データソースのクレデンシャルを指定して **Secret** マニフェストを作成し、**endpoint-secret.yaml** として保存します。

```
apiVersion: v1
kind: Secret
```

```
metadata:
  name: endpoint-secret (1)
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" (2)
  secretKey:   "" (3)
```

(1) **Secret** の名前を指定します。

(2) Base64 でエンコードされたキー ID またはユーザー名を指定します。

(3) Base64 でエンコードされた秘密鍵またはパスワードを指定します。

2. **Secret** マニフェストを適用します。

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

3. **VirtualMachine** マニフェストを編集し、インポートする仮想マシンイメージのデータソースを指定して、**vm-fedora-datavolume.yaml** として保存します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  creationTimestamp: null
  labels:
    kubevirt.io/vm: vm-fedora-datavolume
  name: vm-fedora-datavolume (1)
spec:
  dataVolumeTemplates:
  - metadata:
      creationTimestamp: null
      name: fedora-dv (2)
    spec:
      storage:
        volumeMode: Block (3)
        resources:
          requests:
            storage: 10Gi
        storageClassName: local
      source: (4)
        http:
          url: "https://mirror.arizona.edu/fedora/linux/releases/35/Cloud/x86_64/images/Fedora-Cloud-Base-35-1.2.x86_64.qcow2" (5)
          secretRef: endpoint-secret (6)
          certConfigMap: "" (7)
        # To use a registry source, uncomment the following lines and delete the preceding HTTP source block
        # registry:
          # url: "docker://kubevirt/fedora-cloud-container-disk-demo:latest"
          # secretRef: registry-secret (8)
          # certConfigMap: "" (9)
```

```
    status: {}
  running: true
  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: 1.5Gi
      terminationGracePeriodSeconds: 180
      volumes:
      - dataVolume:
          name: fedora-dv
        name: datavolumedisk1
status: {}
```

1 仮想マシンの名前を指定します。

2 データボリュームの名前を指定します。

3 ボリュームおよびアクセスモードは、既知のストレージプロビジョナーに対して自動的に検出されます。または、**Block** を指定できます。

4 5 コメントブロックを使用して、インポートする仮想マシンイメージの URL またはレジストリーエンドポイントを指定します。たとえば、レジストリーソースを使用する場合は、HTTP または HTTPS ソースブロックをコメントアウトするか削除できます。ここに示されている例の値は、必ず実際の値に置き換えてください。

6 8 データソースの **Secret** を作成した場合は、**Secret** 名を指定します。

7 9 オプション: CA 証明書 config map を指定します。

4. 仮想マシンを作成します。

```
$ oc create -f vm-fedora-datavolume.yaml
```

注記

oc create コマンドは、データボリューム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用して基礎となる 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、データボリュームのステータスが **Succeeded** に変わります。仮想マシンを起動できます。

データボリューム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、これをプロセスをモニターする必要はありません。

検証

1. インポーター Pod は指定された URL から仮想マシンイメージまたはコンテナーディスクをダウンロードし、これをプロビジョニングされた PV に保存します。以下のコマンドを実行してインポーター Pod のステータスを確認します。

```
$ oc get pods
```

2. 次のコマンドを実行して、ステータスが **Succeeded** になるまでデータボリュームを監視します。

```
$ oc describe dv fedora-dv 1
```

1 **VirtualMachine** マニフェストで定義したデータボリューム名を指定します。

3. シリアルコンソールにアクセスして、プロビジョニングが完了し、仮想マシンが起動したことを確認します。

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

10.16.2.6. 関連情報

- 事前割り当てモードを設定 して、データボリューム操作の書き込みパフォーマンスを向上させます。

10.17. 仮想マシンのクローン作成

10.17.1. 複数の namespace 間でデータボリュームをクローン作成するためのユーザーパーミッションの有効化

namespace には相互に分離する性質があるため、ユーザーはデフォルトでは namespace をまたがってリソースのクローンを作成することができません。

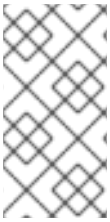
ユーザーが仮想マシンのクローンを別の namespace に作成できるようにするには、**cluster-admin** ロールを持つユーザーが新規のクラスターロールを作成する必要があります。このクラスターロールをユーザーにバインドし、それらのユーザーが仮想マシンのクローンを宛先 namespace に対して作成できるようにします。

10.17.1.1. 前提条件

- **cluster-admin** ロールを持つユーザーのみがクラスターロールを作成できること。

10.17.1.2. データボリュームについて

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。データボリュームは、基礎となる永続ボリューム要求 (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。データボリュームは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。



注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.17.1.3. データボリュームのクローン作成のための RBAC リソースの作成

datavolumes リソースのすべてのアクションのパーミッションを有効にする新規のくスターロールを作成します。

手順

1. **ClusterRole** マニフェストを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: <datavolume-cloner> 1
rules:
- apiGroups: ["cdi.kubevirt.io"]
  resources: ["datavolumes/source"]
  verbs: ["*"]
```

1 クラスターロールの一意の名前。

2. クラスターにクラスターロールを作成します。

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

1 直前の手順で作成された **ClusterRole** マニフェストのファイル名です。

3. 移行元および宛先 namespace の両方に適用される **RoleBinding** マニフェストを作成し、直前の手順で作成したクラスターロールを参照します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: <allow-clone-to-user> 1
  namespace: <Source namespace> 2
subjects:
- kind: ServiceAccount
  name: default
  namespace: <Destination namespace> 3
roleRef:
  kind: ClusterRole
  name: datavolume-cloner 4
  apiGroup: rbac.authorization.k8s.io
```

1 ロールバインディングの一意の名前。

2 ソースデータボリュームの namespace。

3 データボリュームのクローンが作成される namespace。

4 直前の手順で作成したクラスターロールの名前。

4. クラスターにロールバインディングを作成します。

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

1 直前の手順で作成された **RoleBinding** マニフェストのファイル名です。

10.17.2. 新規データボリュームへの仮想マシンディスクのクローン作成

データボリューム設定ファイルでソース PVC を参照し、新規データボリュームに仮想マシンディスクの永続ボリューム要求 (PVC) のクローンを作成できます。

警告

volumeMode: Block が指定された永続ボリューム (PV) から **volumeMode: Filesystem** が指定された PV へのクローンなど、異なるボリュームモード間でのクローン操作がサポートされます。

ただし、**contentType: kubevirt** を使用している場合にのみ異なるボリュームモード間のクローンが可能です。

ヒント

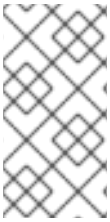
事前割り当てをグローバルに有効にする場合や、単一データボリュームについて、Containerized Data Importer (CDI) はクローン時にディスク領域を事前に割り当てます。事前割り当てにより、書き込みパフォーマンスが向上します。詳細は、データボリュームについての事前割り当ての使用 を参照してください。

10.17.2.1. 前提条件

- ユーザーは、仮想マシンディスクの PVC のクローンを別の namespace に作成するために 追加のパーミッション が必要です。

10.17.2.2. データボリュームについて

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。データボリュームは、基礎となる永続ボリューム要求 (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。データボリュームは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。

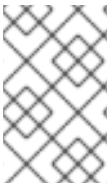


注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.17.2.3. 新規データボリュームへの仮想マシンディスクの永続ボリューム要求 (PVC) のクローン作成

既存の仮想マシンディスクの永続ボリューム要求 (PVC) のクローンを新規データボリュームに作成できます。その後、新規データボリュームは新規の仮想マシンに使用できます。



注記

データボリュームが仮想マシンとは別に作成される場合、データボリューム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、データボリューム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 を判別すること。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。
- OpenShift CLI (**oc**) がインストールされている。

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. 新規データボリュームの名前、ソース PVC の名前および namespace、および新規データボリュームのサイズを指定するデータボリュームの YAML ファイルを作成します。
以下に例を示します。

```
apiVersion: cdi.kubevirt.io/v1beta1
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. 新規データボリュームの名前。
2. ソース PVC が存在する namespace。

3 ソース PVC の名前。

4 新規データボリュームのサイズ。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、それよりも大きくなければなりません。

3. データボリュームを作成して PVC のクローン作成を開始します。

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

注記

データボリュームは仮想マシンが 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データボリュームを参照する仮想マシンを作成できます。

10.17.2.4. 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*

✓ 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

10.17.3. データボリュームテンプレート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永続ボリューム要求 (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 から新規のデータボリュームを作成します。

警告

volumeMode: Block が指定された永続ボリューム (PV) から **volumeMode: Filesystem** が指定された PV へのクローンなど、異なるボリュームモード間でのクローン操作がサポートされます。

ただし、**contentType: kubevirt** を使用している場合にのみ異なるボリュームモード間のクローンが可能です。

ヒント

事前割り当てをグローバルに有効にする場合や、単一データボリュームについて、Containerized Data Importer (CDI) はクローン時にディスク領域を事前に割り当てます。事前割り当てにより、書き込みパフォーマンスが向上します。詳細は、データボリュームについての事前割り当ての使用 を参照してください。

10.17.3.1. 前提条件

- ユーザーは、仮想マシンディスクの PVC のクローンを別の namespace に作成するために 追加のパーミッション が必要です。

10.17.3.2. データボリュームについて

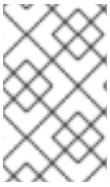
DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。データボリュームは、基礎となる永続ボリューム要求 (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。データボリュームは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。

注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.17.3.3. データボリュームテンプレートの使用による、クローン作成された永続ボリューム要求 (PVC) からの仮想マシンの新規作成

既存の仮想マシンの永続ボリューム要求 (PVC) のクローンをデータボリュームに作成する仮想マシンを作成できます。仮想マシンマニフェストの **dataVolumeTemplate** を参照することにより、**source** PVC のクローンがデータボリュームに作成され、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

データボリュームが仮想マシンのデータボリュームテンプレートの一部として作成されると、データボリューム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、データボリューム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 を判別すること。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。
- OpenShift CLI (**oc**) がインストールされている。

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** データは **my-favorite-vm-disk** から作成されます。
以下に例を示します。

```
apiVersion: kubevirt.io/v1
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:
      storage:
        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データボリュームで仮想マシンを作成します。

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

10.17.3.4. 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*

✓ 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

10.17.4. 仮想マシンディスクの新しいブロックストレージ永続ボリューム要求への複製

複製ターゲットのデータボリューム設定ファイル内にあるソース PVC を参照することで、仮想マシンディスクの永続ボリューム要求 (PVC) を新しいブロック PVC に複製できます。

警告

volumeMode: Block が指定された永続ボリューム (PV) から **volumeMode: Filesystem** が指定された PV へのクローンなど、異なるボリュームモード間でのクローン操作がサポートされます。

ただし、**contentType: kubevirt** を使用している場合にのみ異なるボリュームモード間のクローンが可能です。

ヒント

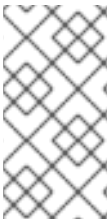
事前割り当てをグローバルに有効にする場合や、単一データボリュームについて、Containerized Data Importer (CDI) はクローン時にディスク領域を事前に割り当てます。事前割り当てにより、書き込みパフォーマンスが向上します。詳細は、データボリュームについての事前割り当ての使用 を参照してください。

10.17.4.1. 前提条件

- ユーザーは、仮想マシンディスクの PVC のクローンを別の namespace に作成するために 追加のパーミッション が必要です。

10.17.4.2. データボリュームについて

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。データボリュームは、基礎となる永続ボリューム要求 (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。データボリュームは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。



注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.17.4.3. ブロック永続ボリュームについて

ブロック永続ボリューム (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、オーバーヘッドを削減することで、仮想マシンのパフォーマンス上の利点をもたらすことができます。

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

10.17.4.4. ローカルブロック永続ボリュームの作成

データボリュームを含むブロックストレージに仮想マシンイメージをインポートする場合は、使用可能なローカルブロック永続ボリュームが必要です。

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永続ボリューム (PV) を作成します。次に、このループデバイスを PV マニフェストで **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** マニフェストを作成します。

```
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 にストレージクラスを設定します。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。

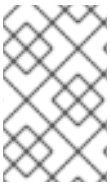
5. ブロック PV を作成します。

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

1. 直前の手順で作成された永続ボリュームのファイル名。

10.17.4.5. 新規データボリュームへの仮想マシンディスクの永続ボリューム要求 (PVC) のクローン作成

既存の仮想マシンディスクの永続ボリューム要求 (PVC) のクローンを新規データボリュームに作成できます。その後、新規データボリュームは新規の仮想マシンに使用できます。



注記

データボリュームが仮想マシンとは別に作成される場合、データボリューム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、データボリューム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 を判別すること。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。
- OpenShift CLI (**oc**) がインストールされている。
- ソース PVC と同じか、これよりも大きい 1 つ以上の利用可能なブロック永続ボリューム (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. 新規データボリュームの名前、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**、および新規データボリュームのサイズを指定するデータボリュームの YAML ファイルを作成します。
以下に例を示します。

```
apiVersion: cdi.kubevirt.io/v1beta1
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. 新規データボリュームの名前。
2. ソース PVC が存在する namespace。
3. ソース PVC の名前。
4. 新規データボリュームのサイズ。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、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. データボリュームを作成して PVC のクローン作成を開始します。

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

注記

データボリュームは仮想マシンが 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データボリュームを参照する仮想マシンを作成できます。

10.17.4.6. 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*

✓ サポートされる操作

□ サポートされない操作

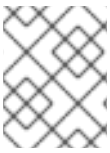
* スクラッチ領域が必要

**カスタム認証局が必要な場合にスクラッチ領域が必要

10.18. 仮想マシンのネットワーク

10.18.1. デフォルトの Pod ネットワーク用の仮想マシンの設定

masquerade バインディングモードを使用するようにネットワークインターフェイスを設定することで、仮想マシンをデフォルトの内部 Pod ネットワークに接続できます。



注記

デフォルトの Pod ネットワークに接続している仮想ネットワークインターフェイスカード (vNIC) 上のトラフィックは、ライブマイグレーション時に中断します。

10.18.1.1. 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る。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: default
          masquerade: {} 1
          ports: 2
            - port: 80
  networks:
  - name: default
    pod: {}
```

1. マスカレードモードを使用した接続
2. オプション: 仮想マシンから公開するポートを、**port** フィールドで指定して一覧表示します。**port** の値は 0 から 65536 の間の数字である必要があります。 **ports** 配列を使用しない場合、有効な範囲内の全ポートが受信トラフィックに対して開きます。この例では、着信トラフィックはポート **80** で許可されます。

注記

ポート 49152 および 49153 は libvirt プラットフォームで使用するために予約され、これらのポートへの他のすべての受信トラフィックは破棄されます。

2. 仮想マシンを作成します。

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

10.18.1.2. デュアルスタック (IPv4 および IPv6) でのマスカレードモードの設定

cloud-init を使用して、新規仮想マシン (VM) を、デフォルトの Pod ネットワークで IPv6 と IPv4 の両方を使用するように設定できます。

仮想マシンインスタンス設定の **Network.pod.vmIPv6NetworkCIDR** フィールドは、VM の静的 IPv6 アドレスとゲートウェイ IP アドレスを決定します。これらは IPv6 トラフィックを仮想マシンにルーティングするために virt-launcher Pod で使用され、外部では使用されませ

ん。**Network.pod.vmIPv6NetworkCIDR** フィールドは、Classless Inter-Domain Routing (CIDR) 表記で IPv6 アドレスブロックを指定します。デフォルト値は **fd10:0:2::2/120** です。この値は、ネットワーク要件に基づいて編集できます。

仮想マシンが実行されている場合、仮想マシンの送受信トラフィックは、virt-launcher Pod の IPv4 アドレスと固有の IPv6 アドレスの両方にルーティングされます。次に、virt-launcher Pod は IPv4 トラフィックを仮想マシンの DHCP アドレスにルーティングし、IPv6 トラフィックを仮想マシンの静的に設定された IPv6 アドレスにルーティングします。

前提条件

- OpenShift Container Platform クラスターは、デュアルスタック用に設定された OVN-Kubernetes Container Network Interface (CNI) ネットワークプラグインを使用する必要があります。

手順

1. 新規の仮想マシン設定では、**masquerade** を指定したインターフェイスを追加し、cloud-init を使用して IPv6 アドレスとデフォルトゲートウェイを設定します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm-ipv6
# ...
        interfaces:
          - name: default
            masquerade: {} (1)
            ports:
              - port: 80 (2)
    networks:
    - name: default
      pod: {}
    volumes:
    - cloudInitNoCloud:
        networkData: |
          version: 2
          ethernets:
            eth0:
              dhcp4: true
              addresses: [ fd10:0:2::2/120 ] (3)
              gateway6: fd10:0:2::1 (4)
```

(1) マスカレードモードを使用した接続

(2) ポート 80 の受信トラフィックを仮想マシンに対して許可します。

(3) 仮想マシンインスタンス設定の **Network.pod.vmIPv6NetworkCIDR** フィールドによって決定される静的 IPv6 アドレス。デフォルト値は **fd10:0:2::2/120** です。

(4) 仮想マシンインスタンス設定の **Network.pod.vmIPv6NetworkCIDR** フィールドによって決定されるゲートウェイ IP アドレス。デフォルト値は **fd10:0:2::1** です。

2. namespace で仮想マシンインスタンスを作成します。

```
$ oc create -f example-vm-ipv6.yaml
```

検証

- IPv6 が設定されていることを確認するには、仮想マシンを起動し、仮想マシンインスタンスのインターフェイスステータスを表示して、これに IPv6 アドレスがあることを確認します。

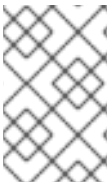
```
$ oc get vmi <vmi-name> -o jsonpath="{.status.interfaces[*].ipAddresses}"
```

10.18.1.3. ジャンボフレームのサポートについて

OVN-Kubernetes CNI プラグインを使用すると、デフォルトの Pod ネットワークに接続されている 2 つの仮想マシン (VM) 間でフラグメント化されていないジャンボフレームパケットを送信できます。ジャンボフレームの最大伝送単位 (MTU) 値は 1500 バイトを超えています。

VM は、クラスター管理者が設定したクラスターネットワークの MTU 値を、次のいずれかの方法で自動的に取得します。

- **libvirt**: ゲスト OS に VirtIO ドライバーの最新バージョンがあり、エミュレーションされたデバイスの Peripheral Component Interconnect (PCI) 設定レジスターを介して着信データを解釈できる場合。
- DHCP: ゲスト DHCP クライアントが DHCP サーバーの応答から MTU 値を読み取ることができる場合。



注記

VirtIO ドライバーを持たない Windows VM の場合、**netsh** または同様のツールを使用して MTU を手動で設定する必要があります。これは、Windows DHCP クライアントが MTU 値を読み取らないためです。

関連情報

- クラスターネットワークの MTU 変更
- ネットワークでの MTU の最適化

10.18.2. 仮想マシンを公開するサービスの作成

Service オブジェクトを使用して、クラスター内またはクラスターの外部に仮想マシンを公開することができます。

10.18.2.1. サービスについて

Kubernetes **サービス** は一連の Pod で実行されているアプリケーションへのクライアントのネットワークアクセスを公開します。サービスは抽象化、負荷分散を提供し、NodePort と LoadBalancer の場合は外部世界への露出を提供します。

サービスは、Web コンソールの **VirtualMachine details → Details** タブで、または **Service** オブジェクトで **spec.type** を指定することによって公開できます。

ClusterIP

内部 IP アドレスでサービスを公開し、クラスター内の他のアプリケーションに DNS 名として公開

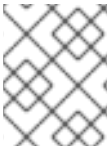
します。1つのサービスを複数の仮想マシンにマッピングできます。クライアントがサービスに接続しようとすると、クライアントのリクエストは使用可能なバックエンド間で負荷分散されます。**ClusterIP** はデフォルトのサービス **タイプ** です。

NodePort

クラスター内の選択した各ノードの同じポートでサービスを公開します。**NodePort** は、クラスター外からサービスにアクセスできるようにします。

LoadBalancer

現在のクラウド（サポートされている場合）に外部ロードバランサーを作成し、固定の外部 IP アドレスをサービスに割り当てます。



注記

オンプレミスクラスターの場合、MetalLB Operator をデプロイすることで負荷分散サービスを設定できます。

関連情報

- MetalLB Operator のインストール
- MetalLB を使用するためのサービスの設定

10.18.2.1.1. デュアルスタックサポート

IPv4 および IPv6 のデュアルスタックネットワークがクラスターに対して有効にされている場合、**Service** オブジェクトに **spec.ipFamilyPolicy** および **spec.ipFamilies** フィールドを定義して、IPv4、IPv6、またはそれら両方を使用するサービスを作成できます。

spec.ipFamilyPolicy フィールドは以下の値のいずれかに設定できます。

SingleStack

コントロールプレーンは、最初に設定されたサービスクラスターの IP 範囲に基づいて、サービスのクラスター IP アドレスを割り当てます。

PreferDualStack

コントロールプレーンは、デュアルスタックが設定されたクラスターのサービス用に IPv4 および IPv6 クラスター IP アドレスの両方を割り当てます。

RequireDualStack

このオプションは、デュアルスタックネットワークが有効にされていないクラスターの場合には失敗します。デュアルスタックが設定されたクラスターの場合、その値が **PreferDualStack** に設定されている場合と同じになります。コントロールプレーンは、IPv4 アドレスと IPv6 アドレス範囲の両方からクラスター IP アドレスを割り当てます。

単一スタックに使用する IP ファミリーや、デュアルスタック用の IP ファミリーの順序は、**spec.ipFamilies** を以下のアレイ値のいずれかに設定して定義できます。

- **[IPv4]**
- **[IPv6]**
- **[IPv4, IPv6]**
- **[IPv6, IPv4]**

10.18.2.2. 仮想マシンのサービスとしての公開

ClusterIP、**NodePort**、または **LoadBalancer** サービスを作成し、クラスター内外から実行中の仮想マシン (VM) に接続します。

手順

1. **VirtualMachine** マニフェストを編集して、サービス作成のラベルを追加します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-ephemeral
  namespace: example-namespace
spec:
  running: false
  template:
    metadata:
      labels:
        special: key (1)
# ...
```

(1) ラベル **special: key** を **spec.template.metadata.labels** セクションに追加します。

注記

仮想マシンのラベルは Pod に渡されます。**special: キー** ラベルは、**Service** マニフェストの **spec.selector** 属性のラベルと一致する必要があります。

2. **VirtualMachine** マニフェストファイルを保存して変更を適用します。

3. 仮想マシンを公開するための **Service** マニフェストを作成します。

```
apiVersion: v1
kind: Service
metadata:
  name: vmservice (1)
  namespace: example-namespace (2)
spec:
  externalTrafficPolicy: Cluster (3)
  ports:
  - nodePort: 30000 (4)
    port: 27017
    protocol: TCP
    targetPort: 22 (5)
  selector:
    special: key (6)
  type: NodePort (7)
```

(1) **Service** オブジェクトの名前。

(2) **Service** オブジェクトが存在する namespace。これは **VirtualMachine** マニフェストの **metadata.namespace** フィールドと同じである必要があります。

3 オプション: ノードが外部 IP アドレスで受信したサービストラフィックを分散する方法を指定します。これは **NodePort** および **LoadBalancer** サービスタイプにのみ適用されま

4 オプション: 設定する場合、**nodePort** 値はすべてのサービスで固有でなければなりません。指定しない場合、**30000** を超える範囲内の値は動的に割り当てられます。

5 オプション: サービスによって公開される VM ポート。ポートリストが仮想マシンマニフェストに定義されている場合は、オープンポートを参照する必要があります。**targetPort** が指定されていない場合は、**ポート** と同じ値を取ります。

6 **VirtualMachine** マニフェストの **spec.template.metadata.labels** スタンザに追加したラベルへの参照。

7 サービスのタイプ。使用できる値は **ClusterIP**、**NodePort**、および **LoadBalancer** です。

4. **サービス** マニフェストファイルを保存します。
5. 以下のコマンドを実行してサービスを作成します。

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

6. 仮想マシンを起動します。仮想マシンがすでに実行中の場合は、再起動します。

検証

1. **Service** オブジェクトをクエリーし、これが利用可能であることを確認します。

```
$ oc get service -n example-namespace
```

ClusterIP サービスの出力例

```
NAME        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)     AGE
vmservice   ClusterIP   172.30.3.149   <none>        27017/TCP   2m
```

NodePort サービスの出力例

```
NAME        TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)           AGE
vmservice   NodePort    172.30.232.73  <none>        27017:30000/TCP   5m
```

LoadBalancer サービスの出力例

```
NAME        TYPE           CLUSTER-IP     EXTERNAL-IP                   PORT(S)           AGE
vmservice   LoadBalancer   172.30.27.5    172.29.10.235,172.29.10.235   27017:31829/TCP   5s
```

2. 仮想マシンに接続するための適切な方法を選択します。

- **ClusterIP** サービスの場合は、サービス IP アドレスとサービスポートを使用して、クラスター内から仮想マシンに接続します。以下に例を示します。

```
$ ssh fedora@172.30.3.149 -p 27017
```

- **NodePort** サービスの場合、ノード IP アドレスとクラスターネットワーク外のノードポートを指定して仮想マシンに接続します。以下に例を示します。

```
$ ssh fedora@$NODE_IP -p 30000
```

- **LoadBalancer** サービスの場合は、**vinagre** クライアントを使用し、パブリック IP アドレスおよびポートで仮想マシンに接続します。外部ポートは動的に割り当てられます。

10.18.2.3. 関連情報

- NodePort を使用した ingress クラストラフィックの設定
- ロードバランサーを使用した ingress クラスターの設定
- Web コンソールの概要

10.18.3. Linux ブリッジ ネットワークへの仮想マシンの割り当て

デフォルトでは、OpenShift Virtualization は単一の内部 Pod ネットワークとともにインストールされます。

追加のネットワークに接続するには、Linux ブリッジ ネットワーク接続定義 (NAD) を作成する必要があります。

仮想マシンを追加のネットワークに割り当てるには、以下を実行します。

1. Linux ブリッジ ノード ネットワーク設定ポリシーを作成します。
2. Linux ブリッジ ネットワーク接続定義を作成します。
3. 仮想マシンを設定して、仮想マシンがネットワーク接続定義を認識できるようにします。

スケジューリング、インターフェイスタイプ、およびその他のノードのネットワークアクティビティーについての詳細は、node networking セクションを参照してください。

10.18.3.1. ネットワーク接続定義によるネットワークへの接続

10.18.3.1.1. Linux ブリッジ ノード ネットワーク設定ポリシーの作成

NodeNetworkConfigurationPolicy マニフェスト YAML ファイルを使用して、Linux ブリッジを作成します。

前提条件

- Kubernetes NMState Operator がインストールされている。

手順

- **NodeNetworkConfigurationPolicy** マニフェストを作成します。この例には、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
```

```
  name: br1-eth1-policy (1)
spec:
  desiredState:
    interfaces:
      - name: br1 (2)
        description: Linux bridge with eth1 as a port (3)
        type: linux-bridge (4)
        state: up (5)
        ipv4:
          enabled: false (6)
        bridge:
          options:
            stp:
              enabled: false (7)
          port:
            - name: eth1 (8)
```

1. ポリシーの名前。
2. インターフェイスの名前。
3. オプション: 人間が判読できるインターフェイスの説明。
4. インターフェイスのタイプ。この例では、ブリッジを作成します。
5. 作成後のインターフェイスの要求された状態。
6. この例では IPv4 を無効にします。
7. この例では STP を無効にします。
8. ブリッジが接続されているノード NIC。

10.18.3.2. Linux ブリッジネットワーク接続定義の作成

> **警告**
>
> 仮想マシンのネットワークアタッチメント定義での IP アドレス管理 (IPAM) の設定はサポートされていません。

10.18.3.2.1. Web コンソールでの Linux ブリッジネットワーク接続定義の作成

ネットワーク接続定義を作成して、Pod と仮想マシンにレイヤー 2 ネットワークを提供できます。

Linux ブリッジ ネットワーク接続定義は、仮想マシンを VLAN に接続するための最も効率的な方法です。

手順

1. Web コンソールで、**Networking → NetworkAttachmentDefinitions** をクリックします。
2. **Create Network Attachment Definition**をクリックします。

注記

ネットワーク接続定義は Pod または仮想マシンと同じ namespace にある必要があります。

3. 一意の **Name** およびオプションの **Description** を入力します。
4. **Network Type** リストから **CNV Linux bridge**を選択します。
5. **Bridge Name**フィールドにブリッジの名前を入力します。
6. オプション: リソースに VLAN ID が設定されている場合、**VLAN Tag Number** フィールドに ID 番号を入力します。
7. オプション: **MAC Spoof Check**を選択して、MAC スプーフフィルタリングを有効にします。この機能により、Pod を終了するための MAC アドレスを 1 つだけ許可することで、MAC スプーフィング攻撃に対してセキュリティーを確保します。
8. **Create** をクリックします。

10.18.3.2.2. CLI での Linux ブリッジネットワーク接続定義の作成

ネットワーク管理者は、タイプ **cnv-bridge** のネットワーク接続定義を、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

- MAC スプーフィングチェックを有効にするには、ノードが nftables をサポートして、**nft** バイナリーがデプロイされている必要があります。

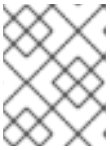
手順

1. 仮想マシンと同じ namespace にネットワーク接続定義を作成します。
2. 次の例のように、仮想マシンをネットワーク接続定義に追加します。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: <bridge-network> (1)
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/<bridge-interface> (2)
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "<bridge-network>", (3)
    "type": "cnv-bridge", (4)
    "bridge": "<bridge-interface>", (5)
    "macspoofchk": true, (6)
```

```
  "vlan": 100, 7
  "preserveDefaultVlan": false 8
}'
```

1. **NetworkAttachmentDefinition** オブジェクトの名前。
2. オプション: ノード選択のアノテーションのキーと値のペア。**bridge-interface** は一部のノードに設定されるブリッジの名前に一致する必要があります。このアノテーションをネットワーク接続定義に追加する場合、仮想マシンインスタンスは **bridge-interface** ブリッジが接続されているノードでのみ実行されます。
3. 設定の名前。設定名をネットワーク接続定義の **name** 値に一致させることが推奨されます。
4. このネットワーク接続定義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。
5. ノードに設定される Linux ブリッジの名前。
6. オプション:MAC スプーフィングチェックを有効にする。**true** に設定すると、Pod またはゲストインターフェイスの MAC アドレスを変更できません。この属性は、Pod からの MAC アドレスを 1 つだけ許可することで、MAC スプーフィング攻撃に対してセキュリティーを確保します。
7. オプション: VLAN タグ。ノードのネットワーク設定ポリシーでは、追加の VLAN 設定は必要ありません。
8. オプション: 仮想マシンがデフォルト VLAN 経由でブリッジに接続するかどうかを示します。デフォルト値は **true** です。



注記

Linux ブリッジネットワーク接続定義は、仮想マシンを VLAN に接続するための最も効率的な方法です。

3. ネットワーク接続定義を作成します。

```
$ oc create -f <network-attachment-definition.yaml> 1
```

1. ここで、**<network-attachment-definition.yaml>** はネットワーク接続定義マニフェストのファイル名です。

検証

- 次のコマンドを実行して、ネットワーク接続定義が作成されたことを確認します。

```
$ oc get network-attachment-definition <bridge-network>
```

10.18.3.3. Linux ブリッジネットワーク用の仮想マシンの設定

10.18.3.3.1. Web コンソールでの仮想マシンの 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。

前提条件

- ネットワーク接続定義が使用可能である必要があります。

手順

1. OpenShift Container Platform コンソールの正しいプロジェクトで、サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration → Network interfaces** をクリックして、仮想マシンにすでに接続されている NIC を表示します。
4. **Add Network Interface** をクリックし、一覧に新規スロットを作成します。
5. 追加ネットワークの **Network** リストからネットワーク接続定義を選択します。
6. 新規 NIC の **Name**、**Model**、**Type**、および **MAC Address** に入力します。
7. **Save** 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。

10.18.3.3.2. ネットワークフィールド

Name	説明
Name	ネットワークインターフェイスコントローラーの名前。
モデル	ネットワークインターフェイスコントローラーのモデルを示します。サポートされる値は **e1000e** および **virtio** です。
Network	利用可能なネットワーク接続定義のリスト。
型	利用可能なバインディングメソッドの一覧。ネットワークインターフェイスに適したバインド方法を選択します。 ● デフォルトの Pod ネットワーク: **masquerade** ● Linux ブリッジネットワーク: **bridge** ● SR-IOV ネットワーク: **SR-IOV**
MAC Address	ネットワークインターフェイスコントローラーの MAC アドレス。MAC アドレスが指定されていない場合、これは自動的に割り当てられます。

10.18.3.3.3. CLI で仮想マシンを追加のネットワークに接続する

ブリッジインターフェイスを追加し、仮想マシン設定でネットワーク接続定義を指定して、仮想マシンを追加のネットワークに割り当てます。

以下の手順では、YAML ファイルを使用して設定を編集し、更新されたファイルをクラスターに適用します。**oc edit <object> <name>** コマンドを使用して、既存の仮想マシンを編集することもできます。

前提条件

- 設定を編集する前に仮想マシンをシャットダウンします。実行中の仮想マシンを編集する場合は、変更を有効にするために仮想マシンを再起動する必要があります。

手順

1. ブリッジネットワークに接続する仮想マシンの設定を作成または編集します。
2. ブリッジインターフェイスを **spec.template.spec.domain.devices.interfaces** 一覧に追加し、ネットワーク接続定義を **spec.template.spec.networks** 一覧に追加します。この例では、**a-bridge-network** ネットワーク接続定義に接続される **bridge-net** というブリッジインターフェイスを追加します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
    name: <example-vm>
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - masquerade: {}
              name: <default>
            - bridge: {}
              name: <bridge-net> 1
# ...
      networks:
        - name: <default>
          pod: {}
        - name: <bridge-net> 2
          multus:
            networkName: <network-namespace>/<a-bridge-network> 3
# ...
```

1. ブリッジインターフェイスの名前。
2. ネットワークの名前。この値は、対応する **spec.template.spec.domain.devices.interfaces** エントリーの **name** 値と一致する必要があります。
3. ネットワーク接続定義の名前。接頭辞は、存在する namespace になります。namespace は、**default** の namespace または仮想マシンが作成される namespace と同じでなければなりません。この場合、**multus** が使用されます。Multus は、Pod または仮想マシンが必要なインターフェイスを使用できるように、複数の CNI が存在できるようにするクラウド

ネットワークインターフェイス (CNI) プラグインです。

3. 設定を適用します。

```
$ oc apply -f <example-vm.yaml>
```

4. オプション: 実行中の仮想マシンを編集している場合は、変更を有効にするためにこれを再起動する必要があります。

10.18.3.4. 次のステップ

- クラスタードメイン名を使用したセカンダリーネットワーク上の仮想マシンへのアクセス

10.18.4. 仮想マシンの SR-IOV ネットワークへの接続

次の手順を実行して、仮想マシン (VM) を Single Root I/O Virtualization (SR-IOV) ネットワークに接続できます。

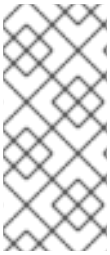
1. SR-IOV ネットワーク デバイスを設定します。
2. SR-IOV ネットワークを設定します。
3. 仮想マシンを SR-IOV ネットワークに接続します。

10.18.4.1. 前提条件

- ホストのファームウェアでグローバル SR-IOV および VT-d 設定を有効 にしておく必要があります。
- SR-IOV Network Operator をインストール しておく必要があります。

10.18.4.2. SR-IOV ネットワークデバイスの設定

SR-IOV Network Operator は **SriovNetworkNodePolicy.sriovnetwork.openshift.io** CustomResourceDefinition を OpenShift Container Platform に追加します。SR-IOV ネットワークデバイスは、SriovNetworkNodePolicy カスタムリソース (CR) を作成して設定できます。



注記

SriovNetworkNodePolicy オブジェクトで指定された設定を適用する際に、SR-IOV Operator はノードをドレイン (解放) する可能性があり、場合によってはノードの再起動を行う場合があります。

設定の変更が適用されるまでに数分かかる場合があります。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- SR-IOV Network Operator がインストールされている。

- ドレイン (解放) されたノードからエビクトされたワークロードを処理するために、クラスター内に利用可能な十分なノードがある。
- SR-IOV ネットワークデバイス設定についてコントロールプレーンノードを選択していない。

手順

1. **SriovNetworkNodePolicy** オブジェクトを作成してから、YAML を **<name>-sriov-node-network.yaml** ファイルに保存します。**<name>** をこの設定の名前に置き換えます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true" 4
  priority: <priority> 5
  mtu: <mtu> 6
  numVfs: <num> 7
  nicSelector: 8
    vendor: "<vendor_code>" 9
    deviceID: "<device_id>" 10
    pfNames: ["<pf_name>", ...] 11
    rootDevices: ["<pci_bus_id>", "..."] 12
  deviceType: vfio-pci 13
  isRdma: false 14
```

1. CR オブジェクトの名前を指定します。
2. SR-IOV Operator がインストールされている namespace を指定します。
3. SR-IOV デバイスプラグインのリソース名を指定します。1 つのリソース名に複数の **SriovNetworkNodePolicy** オブジェクトを作成できます。
4. 設定するノードを選択するノードセレクターを指定します。選択したノード上の SR-IOV ネットワークデバイスのみが設定されます。SR-IOV Container Network Interface (CNI) プラグインおよびデバイスプラグインは、選択したノードにのみデプロイされます。
5. オプション: **0** から **99** までの整数値を指定します。数値が小さいほど優先度が高くなります。したがって、**10** は **99** よりも優先度が高くなります。デフォルト値は **99** です。
6. オプション: Virtual Function の最大転送単位 (MTU) の値を指定します。MTU の最大値は NIC モデルによって異なります。
7. SR-IOV 物理ネットワークデバイス用に作成する仮想機能 (VF) の数を指定します。Intel ネットワークインターフェイスコントローラー (NIC) の場合、VF の数はデバイスがサポートする VF の合計よりも大きくすることはできません。Mellanox NIC の場合、VF の数は **127** よりも大きくすることはできません。
8. **nicSelector** マッピングは、Operator が設定するイーサネットデバイスを選択します。すべてのパラメーターの値を指定する必要はありません。意図せずにイーサネットデバイスを選択する可能性を最低限に抑えるために、イーサネットアダプターを正確に特定できる

ようにすることが推奨されます。**rootDevices** を指定する場合、**vendor**、**deviceID**、または **pfName** の値も指定する必要があります。**pfNames** と **rootDevices** の両方を同時に指定する場合、それらが同一のデバイスをポイントすることを確認します。

9 オプション: SR-IOV ネットワークデバイスのベンダー 16 進コードを指定します。許可される値は **8086** または **15b3** のいずれかのみになります。

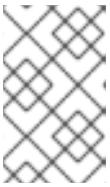
10 オプション: SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。許可される値は **158b**、**1015**、**1017** のみになります。

11 オプション: このパラメーターは、1 つ以上のイーサネットデバイスの物理機能 (PF) 名の配列を受け入れます。

12 このパラメーターは、イーサネットデバイスの物理機能に関する 1 つ以上の PCI バスアドレスの配列を受け入れます。以下の形式でアドレスを指定します: **0000:02:00.1**

13 OpenShift Virtualization の仮想機能には、**vfio-pci** ドライバータイプが必要です。

14 オプション: Remote Direct Memory Access (RDMA) モードを有効にするかどうかを指定します。Mellanox カードの場合、**isRdma** を **false** に設定します。デフォルト値は **false** です。



注記

isRDMA フラグが **true** に設定される場合、引き続き RDMA 対応の VF を通常のネットワークデバイスとして使用できます。デバイスはどちらのモードでも使用できます。

2. オプション: SR-IOV 対応のクラスターノードにまだラベルが付いていない場合は、**SriovNetworkNodePolicy.Spec.NodeSelector** でラベルを付けます。ノードのラベル付けの詳細は、「ノードのラベルを更新する方法について」を参照してください。

3. **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f <name>-sriov-node-network.yaml
```

ここで、**<name>** はこの設定の名前を指定します。

設定の更新が適用された後に、**sriov-network-operator** namespace のすべての Pod が **Running** ステータスに移行します。

4. SR-IOV ネットワークデバイスが設定されていることを確認するには、以下のコマンドを実行します。**<node_name>** を、設定したばかりの SR-IOV ネットワークデバイスを持つノードの名前に置き換えます。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

10.18.4.3. SR-IOV の追加ネットワークの設定

SriovNetwork オブジェクトを作成して、SR-IOV ハードウェアを使用する追加のネットワークを設定できます。

SriovNetwork オブジェクトの作成時に、SR-IOV Network Operator は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。

注記

SriovNetwork オブジェクトが **running** 状態の Pod または仮想マシンに割り当てられている場合、これを変更したり、削除したりしないでください。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. 以下の **SriovNetwork** オブジェクトを作成してから、YAML を **<name>-sriov-network.yaml** ファイルに保存します。**<name>** を、この追加ネットワークの名前に置き換えます。

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: <name> 1
  namespace: openshift-sriov-network-operator 2
spec:
  resourceName: <sriov_resource_name> 3
  networkNamespace: <target_namespace> 4
  vlan: <vlan> 5
  spoofChk: "<spoof_check>" 6
  linkState: <link_state> 7
  maxTxRate: <max_tx_rate> 8
  minTxRate: <min_rx_rate> 9
  vlanQoS: <vlan_qos> 10
  trust: "<trust_vf>" 11
  capabilities: <capabilities> 12
```

1. **<name>** をオブジェクトの名前に置き換えます。SR-IOV Network Operator は、同じ名前を持つ **NetworkAttachmentDefinition** オブジェクトを作成します。
2. SR-IOV ネットワーク Operator がインストールされている namespace を指定します。
3. **<sriov_resource_name>** を、この追加ネットワークの SR-IOV ハードウェアを定義する **SriovNetworkNodePolicy** オブジェクトの **.spec.resourceName** パラメーターの値に置き換えます。
4. **<target_namespace>** を SriovNetwork のターゲット namespace に置き換えます。ターゲット namespace の Pod または仮想マシンのみを SriovNetwork に割り当てることができます。
5. オプション: **<vlan>** を、追加ネットワークの仮想 LAN (VLAN) ID に置き換えます。整数値は **0** から **4095** である必要があります。デフォルト値は **0** です。
6. オプション: **<spoof_check>** を VF の spoof check モードに置き換えます。許可される値は、文字列の **"on"** および **"off"** です。

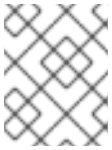
重要

指定する値を引用符で囲む必要があります。そうしないと、CR は SR-IOV ネットワーク Operator によって拒否されます。

7 オプション: **<link_state>** を仮想機能 (VF) のリンクの状態に置き換えます。許可される値は、**enable**、**disable**、および **auto** です。

8 オプション: **<max_tx_rate>** を VF の最大伝送レート (Mbps) に置き換えます。

9 オプション: **<min_tx_rate>** を VF の最小伝送レート (Mbps) に置き換えます。この値は、常に最大伝送レート以下である必要があります。



注記

Intel NIC は **minTxRate** パラメーターをサポートしません。詳細は、BZ#1772847 を参照してください。

10 オプション: **<vlan_qos>** を VF の IEEE 802.1p 優先レベルに置き換えます。デフォルト値は **0** です。

11 オプション: **<trust_vf>** を VF の信頼モードに置き換えます。許可される値は、文字列の **"on"** および **"off"** です。

重要

指定する値を引用符で囲む必要があります。そうしないと、CR は SR-IOV ネットワーク Operator によって拒否されます。

12 オプション: **<capabilities>** を、このネットワークに設定する機能に置き換えます。

2. オブジェクトを作成するには、以下のコマンドを入力します。**<name>** を、この追加ネットワークの名前に置き換えます。

```
$ oc create -f <name>-sriov-network.yaml
```

3. オプション: 以下のコマンドを実行して、直前の手順で作成した **SriovNetwork** オブジェクトに関連付けられた **NetworkAttachmentDefinition** オブジェクトが存在することを確認するには、以下のコマンドを入力します。**<namespace>** を、**SriovNetwork** オブジェクト で指定した namespace に置き換えます。

```
$ oc get net-attach-def -n <namespace>
```

10.18.4.4. 仮想マシンの SR-IOV ネットワークへの接続

仮想マシンの設定にネットワークの詳細を含めることで、仮想マシンを SR-IOV ネットワークに接続することができます。

手順

1. SR-IOV ネットワークの詳細を仮想マシン設定の **spec.domain.devices.interfaces** および **spec.networks** に追加します。

-

```
kind: VirtualMachine
# ...
spec:
  domain:
    devices:
      interfaces:
      - name: <default> 1
        masquerade: {} 2
      - name: <nic1> 3
        sriov: {}
  networks:
  - name: <default> 4
    pod: {}
  - name: <nic1> 5
    multus:
        networkName: <sriov-network> 6
# ...
```

1. Pod ネットワークに接続されているインターフェイスの一意の名前。
2. デフォルト Pod ネットワークへの **masquerade** バインディング。
3. SR-IOV インターフェイスの一意の名前。
4. Pod ネットワークインターフェイスの名前。これは、前のステップで定義した **interfaces.name** と同じである必要があります。
5. SR-IOV ネットワークの名前。これは、前のステップで定義した **interfaces.name** と同じである必要があります。
6. SR-IOV ネットワーク割り当て定義の名前。

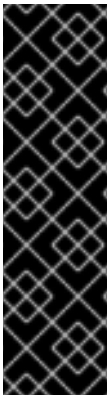
2. 仮想マシン設定を適用します。

```
$ oc apply -f <vm-sriov.yaml> 1
```

1. 仮想マシン YAML ファイルの名前。

10.18.4.5. DPDK ワークロード用のクラスター設定

以下の手順を使用して、Data Plane Development Kit (DPDK) ワークロードを実行する OpenShift Container Platform クラスターを設定できます。



重要

DPDK ワークロード用のクラスター設定は、テクノロジープレビュー機能としてのみ使用できます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator がインストールされている。
- Node Tuning Operator がインストールされている。

手順

1. コンピュートノードトポロジーのマッピングを実行し、DPDK アプリケーション用に分離する Non-Uniform Memory Access (NUMA) CPU と、オペレーティングシステム (OS) 用に予約する NUMA CPU を決定します。
2. コンピュートノードのサブセットにカスタムロール (例: **worker-dpdk**) のラベルを追加します。

```
$ oc label node <node_name> node-role.kubernetes.io/worker-dpdk=""
```

3. **spec.machineConfigSelector** オブジェクトに **worker-dpdk** ラベルを含む新しい **MachineConfigPool** マニフェストを作成します。

MachineConfigPool マニフェストの例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: worker-dpdk
  labels:
    machineconfiguration.openshift.io/role: worker-dpdk
spec:
  machineConfigSelector:
    matchExpressions:
      - key: machineconfiguration.openshift.io/role
        operator: In
        values:
          - worker
          - worker-dpdk
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-dpdk: ""
```

4. 前の手順で作成したラベル付きノードとマシン設定プールに適用する **PerformanceProfile** マニフェストを作成します。パフォーマンスプロファイルは、DPDK アプリケーション用に分離された CPU とハウスキーピング用に予約された CPU を指定します。

   PerformanceProfile マニフェストの例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: profile-1
spec:
  cpu:
    isolated: 4-39,44-79
    reserved: 0-3,40-43
  hugepages:
    defaultHugepagesSize: 1G
    pages:
    - count: 32
      node: 0
      size: 1G
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/worker-dpdk: ""
  numa:
    topologyPolicy: single-numa-node
```

注記

MachineConfigPool マニフェストと **PerformanceProfile** マニフェストを適用すると、コンピュートノードが自動的に再起動します。

5. **PerformanceProfile** オブジェクトの **status.runtimeClass** フィールドから、生成された **RuntimeClass** リソースの名前を取得します。

```
$ oc get performanceprofiles.performance.openshift.io profile-1 -
o=jsonpath='{.status.runtimeClass}{"\n"}'
```

6. 次のアノテーションを **HyperConverged** カスタムリソース (CR) に追加して、以前に取得した **RuntimeClass** 名を **virt-launcher** Pod のデフォルトコンテナーランタイムクラスとして設定します。

```
$ oc annotate --overwrite -n openshift-cnv hco kubevirt-hyperconverged \
   kubevirt.kubevirt.io/jsonpatch='[{"op": "add", "path":
"/spec/configuration/defaultRuntimeClass", "value": <runtimeclass_name>}]'
```

注記

HyperConverged CR にアノテーションを追加すると、アノテーションの適用後に作成されるすべての仮想マシンに影響するグローバル設定が変更されます。このアノテーションを設定すると、OpenShift Virtualization インスタンスのサポートに違反するため、必ずテストクラスター限定で使用する必要があります。最適なパフォーマンスを得るには、サポート例外を申請してください。

7. **spec.deviceType** フィールドを **vfio-pci** に設定して **SriovNetworkNodePolicy** オブジェクトを作成します。

   SriovNetworkNodePolicy マニフェストの例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intel_nics_dpdk
  deviceType: vfio-pci
  mtu: 9000
  numVfs: 4
  priority: 99
  nicSelector:
    vendor: "8086"
    deviceID: "1572"
    pfNames:
      - eno3
    rootDevices:
      - "0000:19:00.2"
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

関連情報

- CPU マネージャーおよび Topology Manager の使用
- Huge Page の設定
- カスタムマシン設定プールの作成

10.18.4.6. DPDK ワークロード用のプロジェクト設定

SR-IOV ハードウェアで DPDK ワークロードを実行するプロジェクトを設定できます。

前提条件

- DPDK ワークロードを実行するようにクラスターが設定されている。

手順

1. DPDK アプリケーションの namespace を作成します。

```
$ oc create ns dpdk-checkup-ns
```

2. **SriovNetworkNodePolicy** オブジェクトを参照する **SriovNetwork** オブジェクトを作成します。**SriovNetwork** オブジェクトの作成時に、SR-IOV Network Operator は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。

   SriovNetwork マニフェストの例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-sriovnetwork
  namespace: openshift-sriov-network-operator
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [{
        "dst": "0.0.0.0/0"
      }],
      "gateway": "10.56.217.1"
    }
  networkNamespace: dpdk-checkup-ns (1)
  resourceName: intel_nics_dpdk (2)
  spoofChk: "off"
  trust: "on"
  vlan: 1019
```

1. **NetworkAttachmentDefinition** オブジェクトがデプロイされる namespace。
2. DPDK ワークロード用クラスターの設定時に作成された **SriovNetworkNodePolicy** オブジェクトの **spec.resourceName** 属性値。

3. オプション: 仮想マシンレイテンシーチェックアップを実行して、ネットワークが適切に設定されていることを確認します。
4. オプション: DPDK チェックアップを実行して、namespace が DPDK ワークロード用に準備できているか確認します。

関連情報

- プロジェクトの使用
- 仮想マシンのレイテンシーチェックアップ
- DPDK チェックアップ

10.18.4.7. DPDK ワークロード用の仮想マシン設定

仮想マシン (VM) 上で Data Packet Development Kit (DPDK) ワークロードを実行すると、レイテンシーの短縮とスループットが向上し、ユーザー空間でのパケット処理を高速化できます。DPDK は、ハードウェアベースの I/O 共有に SR-IOV ネットワークを使用します。

前提条件

- DPDK ワークロードを実行するようにクラスターが設定されている。
- 仮想マシンを実行するプロジェクトを作成し、設定している。

手順

1. **VirtualMachine** マニフェストを編集して、SR-IOV ネットワークインターフェイス、CPU トポロジー、CRI-O アノテーション、Huge Page に関する情報を格納します。

VirtualMachine マニフェストの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: rhel-dpdk-vm
spec:
  running: true
  template:
    metadata:
      annotations:
        cpu-load-balancing.crio.io: disable (1)
        cpu-quota.crio.io: disable (2)
        irq-load-balancing.crio.io: disable (3)
    spec:
      domain:
        cpu:
          sockets: 1 (4)
          cores: 5 (5)
          threads: 2
          dedicatedCpuPlacement: true
          isolateEmulatorThread: true
        interfaces:
          - masquerade: {}
            name: default
          - model: virtio
            name: nic-east
            pciAddress: '0000:07:00.0'
            sriov: {}
          networkInterfaceMultiqueue: true
          rng: {}
      memory:
        hugepages:
          pageSize: 1Gi (6)
          guest: 8Gi
      networks:
        - name: default
          pod: {}
        - multus:
            networkName: dpdk-net (7)
          name: nic-east
# ...
```

(1) このアノテーションは、コンテナーが使用する CPU に対するロードバランシングが無効であることを示します。

(2) このアノテーションは、コンテナーが使用する CPU に対する CPU クォータが無効であることを示します。

(3)

このアノテーションは、コンテナーが使用する CPU に対する Interrupt Request (IRQ) のロードバランシングが無効であることを示します。

4 仮想マシン内のソケットの数。同じ Non-Uniform Memory Access (NUMA) ノードから CPU をスケジュールするには、このフィールドを **1** に設定する必要があります。

5 仮想マシン内のコアの数。値は **1** 以上とします。この例では、仮想マシンは 5 個のハイパースレッドか 10 個の CPU でスケジュールされています。

6 Huge Page のサイズ。x86-64 アーキテクチャーの有効な値は 1Gi と 2Mi です。この例は、サイズが 1Gi の 8 個の Huge Page に対する要求です。

7 SR-IOV **NetworkAttachmentDefinition** オブジェクトの名前。

2. エディターを保存し、終了します。
3. **VirtualMachine** マニフェストを適用します。

```
$ oc apply -f <file_name>.yaml
```

4. ゲストオペレーティングシステムを設定します。次の例は、RHEL 8 OS の設定手順を示しています。
   a. GRUB ブートローダーコマンドラインインターフェイスを使用して、Huge Page を設定します。次の例では、1G の Huge Page を 8 個指定しています。

```
$ grubby --update-kernel=ALL --args="default_hugepagesz=1GB hugepagesz=1G
hugepages=8"
```

   b. TuneD アプリケーションで **cpu-partitioning** プロファイルを使用して低レイテンシーチューニングを実現するには、次のコマンドを実行します。

```
$ dnf install -y tuned-profiles-cpu-partitioning
```

```
$ echo isolated_cores=2-9 > /etc/tuned/cpu-partitioning-variables.conf
```

   最初の 2 つの CPU (0 と 1) はハウスキーピングタスク用に確保され、残りは DPDK アプリケーション用に分離されます。

```
$ tuned-adm profile cpu-partitioning
```

   c. **driverctl** デバイスドライバー制御ユーティリティーを使用して、SR-IOV NIC ドライバーをオーバーライドします。

```
$ dnf install -y driverctl
```

```
$ driverctl set-override 0000:07:00.0 vfio-pci
```

5. 仮想マシンを再起動して変更を適用します。

10.18.4.8. 次のステップ

- クラスタードメイン名を使用したセカンダリーネットワーク上の仮想マシンへのアクセス

10.18.5. 仮想マシンのサービスメッシュへの接続

OpenShift Virtualization が OpenShift Service Mesh に統合されるようになりました。IPv4 を使用してデフォルトの Pod ネットワークで仮想マシンのワークロードを実行する Pod 間のトラフィックのモニター、可視化、制御が可能です。

10.18.5.1. 前提条件

- サービスメッシュ Operator を インストールし、サービスメッシュコントロールプレーン をデプロイしておく必要があります。
- 仮想マシンが作成される namespace を サービスメッシュメンバーロール に追加しておく必要があります。
- デフォルトの Pod ネットワークには **masquerade** バインディングメソッドを使用する必要があります。

10.18.5.2. サービスメッシュの仮想マシンの設定

仮想マシン (VM) ワークロードをサービスメッシュに追加するには、**sidecar.istio.io/inject** アノテーションを **true** に設定して、仮想マシン設定ファイルでサイドカーコンテナーの自動挿入を有効にします。次に、仮想マシンをサービスとして公開し、メッシュでアプリケーションを表示します。

前提条件

- ポートの競合を回避するには、Istio サイドカープロキシーが使用するポートを使用しないでください。これには、ポート 15000、15001、15006、15008、15020、15021、および 15090 が含まれます。

手順

1. 仮想マシン設定ファイルを編集し、**sidecar.istio.io/inject: "true"** アノテーションを追加します。

   設定ファイルのサンプル

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-istio
  name: vm-istio
spec:
  runStrategy: Always
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-istio
        app: vm-istio (1)
      annotations:
        sidecar.istio.io/inject: "true" (2)
    spec:
      domain:
        devices:
          interfaces:
```

```
      - name: default
        masquerade: {} 3
      disks:
      - disk:
          bus: virtio
        name: containerdisk
      - disk:
          bus: virtio
        name: cloudinitdisk
    resources:
      requests:
        memory: 1024M
  networks:
  - name: default
    pod: {}
  terminationGracePeriodSeconds: 180
  volumes:
  - containerDisk:
      image: registry:5000/kubevirt/fedora-cloud-container-disk-demo:devel
    name: containerdisk
```

1. サービスセレクターの属性と同じにする必要があるキー/値のペア (ラベル) です。
2. 自動のサイドカーコンテナー挿入を有効にするためのアノテーションです。
3. デフォルトの Pod ネットワークで使用するバインディングメソッド (マスカレードモード) です。

2. 仮想マシン設定を適用します。

```
$ oc apply -f <vm_name>.yaml 1
```

1. 仮想マシン YAML ファイルの名前。

3. **Service** オブジェクトを作成し、仮想マシンをサービスメッシュに公開します。

```
apiVersion: v1
kind: Service
metadata:
  name: vm-istio
spec:
  selector:
    app: vm-istio 1
  ports:
    - port: 8080
      name: http
      protocol: TCP
```

1. サービスの対象となる Pod セットを判別するサービスセレクターです。この属性は、仮想マシン設定ファイルの **spec.metadata.labels** フィールドに対応します。上記の例では、**vm-istio** という名前の **Service** オブジェクトは、ラベルが **app=vm-istio** の Pod の TCP ポート 8080 をターゲットにします。

4. サービスを作成します。

```
$ oc create -f <service_name>.yaml 1
```

1 サービス YAML ファイルの名前。

10.18.6. 仮想マシンの IP アドレスの設定

仮想マシンの静的および動的 IP アドレスを設定できます。

10.18.6.1. cloud-init を使用した新規仮想マシンの IP アドレスの設定

仮想マシン (VM) を作成するときに、cloud-init を使用してセカンダリー NIC の IP アドレスを設定できます。IP アドレスは、動的または静的にプロビジョニングできます。

注記

VM が Pod ネットワークに接続されている場合、更新しない限り、Pod ネットワークインターフェイスがデフォルトルートになります。

前提条件

- 仮想マシンはセカンダリーネットワークに接続されている。
- 仮想マシンの動的 IP を設定するために、セカンダリーネットワーク上で使用できる DHCP サーバーがある。

手順

- 仮想マシン設定の **spec.template.spec.volumes.cloudInitNoCloud.networkData** スタンザを編集します。
  - 動的 IP アドレスを設定するには、インターフェイス名を指定し、DHCP を有効にします。

```
kind: VirtualMachine
spec:
# ...
  template:
  # ...
    spec:
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth1: 1
                dhcp4: true
```

1 インターフェイス名を指定します。

  - 静的 IP を設定するには、インターフェイス名と IP アドレスを指定します。

```
kind: VirtualMachine
spec:
# ...
  template:
  # ...
    spec:
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth1: 1
                addresses:
                - 10.10.10.14/24 2
```

1 インターフェイス名を指定します。

2 静的 IP アドレスを指定します。

10.18.7. NIC の IP アドレスの仮想マシンへの表示

Web コンソールまたは **oc** クライアントを使用して、ネットワークインターフェイスコントローラー (NIC) の IP アドレスを表示できます。QEMU ゲストエージェント は、仮想マシンのセカンダリーネットワークに関する追加情報を表示します。

10.18.7.1. 前提条件

- 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いる。

10.18.7.2. CLI での仮想マシンインターフェイスの IP アドレスの表示

ネットワークインターフェイス設定は **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、**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イス設定を表示します。

```
$ oc describe vmi <vmi_name>
```

出力例

```
# ...
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
```

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

IP 情報は、仮想マシンの **VirtualMachine details** ページに表示されます。

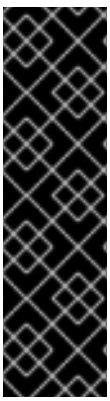
手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシン名を選択して、**VirtualMachine details** ページを開きます。

接続されている各 NIC の情報は、**Details** タブの **IP Address** の下に表示されます。

10.18.8. クラスタードメイン名を使用したセカンダリーネットワーク上の仮想マシンへのアクセス

クラスターの完全修飾ドメイン名 (FQDN) を使用して、クラスターの外部からセカンダリーネットワークインターフェイスに接続されている仮想マシン (VM) にアクセスできます。



重要

クラスター FQDN を使用した VM へのアクセスは、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

10.18.8.1. セカンダリーネットワーク用の DNS サーバーの設定

Cluster Network Addons Operator (CNAO) は、**HyperConverged** カスタムリソース (CR) で **KubeSecondaryDNS** 機能ゲートを有効にすると、ドメインネームサーバー (DNS) サーバーと監視コンポーネントをデプロイします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** パーミッションを持つ OpenShift Container Platform クラスターにアクセスできる。

手順

1. MetalLB またはその他のロードバランサーを使用して **LoadBalancer** サービスを作成し、DNS サーバーをクラスター外に公開します。サービスはポート 53 でリッスンし、ポート 5353 をターゲットにします。以下に例を示します。

```
$ oc expose -n openshift-cnv deployment/secondary-dns --name=dns-lb --
type=LoadBalancer --port=53 --target-port=5353 --protocol='UDP'
```

2. **Service** オブジェクトをクエリーして、サービスのパブリック IP アドレスを取得します。

```
$ oc get service -n openshift-cnv
```

出力例

```
NAME       TYPE           CLUSTER-IP     EXTERNAL-IP      PORT(S)         AGE
dns-lb   LoadBalancer      172.30.27.5    10.46.41.94      53:31829/TCP    5s
```

3. **HyperConverged** CR を編集して、DNS サーバーと監視コンポーネントをデプロイします。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
    featureGates:
      deployKubeSecondaryDNS: true (1)
    kubeSecondaryDNSNameServerIP: "10.46.41.94" (2)
# ...
```

(1) **KubeSecondaryDNS** 機能ゲートを **true** に設定します。

(2) サービスの IP アドレスを手順 2 で取得した値に設定します。

4. 以下のコマンドを使用して、OpenShift Container Platform クラスターの FQDN を取得します。

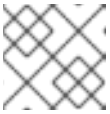
```
$ oc get dnses.config.openshift.io cluster -o json | jq .spec.baseDomain
```

出力例

```
openshift.example.com
```

5. 次のいずれかの方法を使用して、DNS サーバーを指定します。

- ローカルマシンの **resolv.conf** ファイルに **kubeSecondaryDNSNameServerIP** 値を追加します。



注記

resolv.conf ファイルを編集すると、既存の DNS 設定が上書きされます。

- **kubeSecondaryDNSNameServerIP** 値とクラスター FQDN をエンタープライズ DNS サーバーレコードに追加します。以下に例を示します。

```
vm.<FQDN>. IN NS ns.vm.<FQDN>.
```

```
ns.vm.<FQDN>. IN A 10.46.41.94
```

10.18.8.2. クラスター FQDN を使用したセカンダリーネットワーク上の仮想マシンへの接続

クラスターの完全修飾ドメイン名 (FQDN) を使用して、クラスターの外部からセカンダリーネットワークインターフェイスに接続されている仮想マシン (VM) にアクセスできます。

前提条件

- QEMU ゲストエージェントが仮想マシンで実行されている必要があります。
- DNS クライアントを使用して接続する VM の IP アドレスは、パブリックである必要があります。
- セカンダリーネットワーク用の DNS サーバーを設定しました。
- クラスターの完全修飾ドメイン名 (FQDN) を取得しました。

手順

1. 次のコマンドを使用して、VM 設定を取得します。

```
$ oc get vm -n <namespace> <vm_name> -o yaml
```

出力例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: example-vm
  name: example-vm
  namespace: example-namespace
spec:
  running: true
  template:
    metadata:
      labels:
        kubevirt.io/vm: example-vm
    spec:
      domain:
```

```
      devices:
# ...
        interfaces:
          - bridge: {}
            name: example-nic
# ...
    networks:
    - multus:
        networkName: bridge-conf
      name: example-nic (1)
# ...
```

(1) セカンダリーネットワークインターフェイスの名前を指定します。

2. **ssh** コマンドを使用して仮想マシンに接続します。

```
$ ssh <user_name>@<interface_name>.<vm_name>.<namespace>.vm.<FQDN> (1)
```

(1) ユーザー名、インターフェイス名、仮想マシン名、VM 名前空間、および FQDN を指定します。

例

```
$ ssh you@example-nic.example-vm.example-namespace.vm.openshift.example.com
```

10.18.8.3. 関連情報

- ロードバランサーを使用した Ingress クラスターの設定
- MetalLB を使用した負荷分散
- 仮想マシンの IP アドレスの設定

10.18.9. 仮想マシンの MAC アドレスプールの使用

KubeMacPool コンポーネントは、指定の namespace の仮想マシン NIC に MAC アドレスプールサービスを提供します。

10.18.9.1. KubeMacPool について

KubeMacPool は namespace ごとに MAC アドレスプールを提供し、プールから仮想マシン NIC の MAC アドレスを割り当てます。これにより、NIC には別の仮想マシンの MAC アドレスと競合しない一意の MAC アドレスが割り当てられます。

仮想マシンから作成される仮想マシンインスタンスは、再起動時に割り当てられる MAC アドレスを保持します。

注記

KubeMacPool は、仮想マシンから独立して作成される仮想マシンインスタンスを処理しません。

KubeMacPool は、OpenShift Virtualization のインストール時にデフォルトで有効化されます。namespace の MAC アドレスプールは、**mutatevirtualmachines.kubemacpool.io=ignore** ラベルを namespace に追加して無効にできます。ラベルを削除して、namespace の KubeMacPool を再度有効にします。

10.18.9.2. CLI での namespace の MAC アドレスプールの無効化

mutatevirtualmachines.kubemacpool.io=ignore ラベルを namespace に追加して、namespace の仮想マシンの MAC アドレスプールを無効にします。

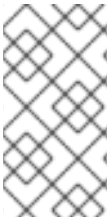
手順

- **mutatevirtualmachines.kubemacpool.io=ignore** ラベルを namespace に追加します。以下の例では、KubeMacPool を 2 つの namespace (**<namespace1>** および **<namespace2>**) について無効にします。

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io=ignore
```

10.18.9.3. CLI での namespace の MAC アドレスプールを再度有効にする

namespace の KubeMacPool を無効にしている場合で、これを再度有効にする必要がある場合は、namespace から **mutatevirtualmachines.kubemacpool.io=ignore** ラベルを削除します。



注記

以前のバージョンの OpenShift Virtualization では、**mutatevirtualmachines.kubemacpool.io=allocate** ラベルを使用して namespace の KubeMacPool を有効にしていました。これは引き続きサポートされますが、KubeMacPool がデフォルトで有効化されるようになったために不要になります。

手順

- KubeMacPool ラベルを namespace から削除します。以下の例では、KubeMacPool を 2 つの namespace (**<namespace1>** および **<namespace2>**) について再度有効にします。

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io-
```

10.19. 仮想マシンディスク

10.19.1. 仮想マシンのローカルストレージの設定

ホストパスプロビジョナー (HPP) を使用して、仮想マシンのローカルストレージを設定できます。

OpenShift Virtualization Operator のインストール時に、Hostpath Provisioner (HPP) Operator は自動的にインストールされます。HPP は、Hostpath Provisioner Operator によって作成される OpenShift Virtualization 用に設計されたローカルストレージプロビジョナーです。HPP を使用するには、HPP カスタムリソース (CR) を作成する必要があります。

10.19.1.1. 基本ストレージ プールを使用したホストパスプロビジョナーの作成

storagePools スタンザを使用して HPP カスタムリソース (CR) を作成することにより、基本ストレージプールを使用してホストパスプロビジョナー (HPP) を設定します。ストレージプールは、CSI ドライバーが使用する名前とパスを指定します。

前提条件

- **spec.storagePools.path** で指定されたディレクトリーには、読み取り/書き込みアクセス権が必要です。
- ストレージプールは、オペレーティングシステムと同じパーティションにあってはなりません。そうしないと、オペレーティングシステムのパーティションがいっぱいになり、パフォーマンスに影響を与えたり、ノードが不安定になったり、使用できなくなったりする可能性があります。

手順

1. 次の例のように、**storagePools** スタンザを含む **hpp_cr.yaml** ファイルを作成します。

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  storagePools: 1
  - name: any_name
    path: "/var/myvolumes" 2
workload:
  nodeSelector:
    kubernetes.io/os: linux
```

1. **storagePools** スタンザは、複数のエントリーを追加できる配列です。
2. このノードパスの下にストレージプールディレクトリーを指定します。

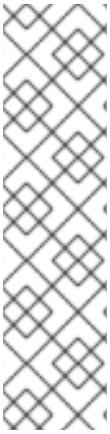
2. ファイルを保存して終了します。
3. 次のコマンドを実行して HPP を作成します。

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

10.19.1.1.1. ストレージクラスの作成について

ストレージクラスの作成時に、ストレージクラスに属する永続ボリューム (PV) の動的プロビジョニングに影響するパラメーターを設定します。**StorageClass** オブジェクトの作成後には、このオブジェクトのパラメーターを更新できません。

ホストパスプロビジョナー (HPP) を使用するには、**storagePools** スタンザで CSI ドライバーの関連付けられたストレージクラスを作成する必要があります。



注記

仮想マシンは、ローカル PV に基づくデータボリュームを使用します。ローカル PV は特定のノードにバインドされます。ディスクイメージは仮想マシンで使用するために準備されますが、ローカルストレージ PV がすでに固定されたノードに仮想マシンをスケジュールすることができない可能性があります。

この問題を解決するには、Kubernetes Pod スケジューラーを使用して、永続ボリューム要求 (PVC) を正しいノードの PV にバインドします。**volumeBindingMode** パラメーターが **WaitForFirstConsumer** に設定された **StorageClass** 値を使用することにより、PV のバインディングおよびプロビジョニングは、Pod が PVC を使用して作成されるまで遅延します。

10.19.1.1.2. storagePools スタンザを使用した CSI ドライバーのストレージクラスの作成

ホストパス プロビジョナー (HPP) CSI ドライバー用のストレージクラスカスタムリソース (CR) を作成します。

手順

1. **storageclass_csi.yaml** ファイルを作成して、ストレージクラスを定義します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hostpath-csi
provisioner: kubevirt.io.hostpath-provisioner
reclaimPolicy: Delete 1
volumeBindingMode: WaitForFirstConsumer 2
parameters:
  storagePool: my-storage-pool 3
```

1. **reclaimPolicy** には、**Delete** および **Retain** の 2 つの値があります。値を指定しない場合、デフォルト値は **Delete** です。
2. **volumeBindingMode** パラメーターは、動的プロビジョニングとボリュームのバインディングが実行されるタイミングを決定します。**WaitForFirstConsumer** を指定して、永続ボリューム要求 (PVC) を使用する Pod が作成されるまで PV のバインディングおよびプロビジョニングを遅延させます。これにより、PV が Pod のスケジュール要件を満たすようになります。
3. HPP CR で定義されているストレージプールの名前を指定します。

1. ファイルを保存して終了します。
2. 次のコマンドを実行して、**StorageClass** オブジェクトを作成します。

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

10.19.1.2. PVC テンプレートで作成されたストレージプールについて

単一の大きな永続ボリューム (PV) がある場合は、ホストパスプロビジョナー (HPP) カスタムリソース (CR) で PVC テンプレートを定義することにより、ストレージプールを作成できます。

PVC テンプレートで作成されたストレージプールには、複数の HPP ボリュームを含めることができます。PV を小さなボリュームに分割すると、データ割り当ての柔軟性が向上します。

PVC テンプレートは、**PersistentVolumeClaim** オブジェクトの **spec** スタンザに基づいています。

PersistentVolumeClaim オブジェクトの例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: iso-pvc
spec:
  volumeMode: Block (1)
  storageClassName: my-storage-class
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

(1) この値は、ブロックボリュームモードの PV にのみ必要です。

HPP CR の **pvcTemplate** 仕様を使用してストレージプールを定義します。Operator は、HPP CSI ドライバーを含む各ノードの **pvcTemplate** 仕様から PVC を作成します。PVC テンプレートから作成される PVC は単一の大きな PV を消費するため、HPP は小規模な動的ボリュームを作成できます。

基本的なストレージプールを、PVC テンプレートから作成されたストレージプールと組み合わせることができます。

10.19.1.2.1. PVC テンプレートを使用したストレージプールの作成

HPP カスタムリソース (CR) で PVC テンプレートを指定することにより、複数のホストパスプロビジョナー (HPP) ボリューム用のストレージプールを作成できます。

前提条件

- **spec.storagePools.path** で指定されたディレクトリーには、読み取り/書き込みアクセス権が必要です。
- ストレージプールは、オペレーティングシステムと同じパーティションにあってはなりません。そうしないと、オペレーティングシステムのパーティションがいっぱいになり、パフォーマンスに影響を与えたり、ノードが不安定になったり、使用できなくなったりする可能性があります。

手順

1. 次の例に従って、**storagePools** スタンザで永続ボリューム (PVC) テンプレートを指定する HPP CR の **hpp_pvc_template_pool.yaml** ファイルを作成します。

```
apiVersion: hostpathprovisioner.kubevirt.io/v1beta1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
```

```
  imagePullPolicy: IfNotPresent
  storagePools: (1)
  - name: my-storage-pool
    path: "/var/myvolumes" (2)
    pvcTemplate:
      volumeMode: Block (3)
      storageClassName: my-storage-class (4)
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 5Gi (5)
  workload:
    nodeSelector:
      kubernetes.io/os: linux
```

1. **storagePools** スタンザは、基本ストレージプールと PVC テンプレートストレージプールの両方を含むことができるアレイです。
2. このノードパスの下にストレージプールディレクトリーを指定します。
3. オプション: **volumeMode** パラメーターは、プロビジョニングされたボリューム形式と一致する限り、**Block** または **Filesystem** のいずれかにすることができます。値が指定されていない場合、デフォルトは **Filesystem** です。**volumeMode** が **Block** の場合、Pod をマウントする前にブロックボリュームに XFS ファイルシステムが作成されます。
4. **storageClassName** パラメーターを省略すると、デフォルトのストレージクラスを使用して PVC を作成します。**storageClassName** を省略する場合、HPP ストレージクラスがデフォルトのストレージクラスではないことを確認してください。
5. 静的または動的にプロビジョニングされるストレージを指定できます。いずれの場合も、要求されたストレージサイズが仮想的に分割する必要のあるボリュームに対して適切になるようにしてください。そうしないと、PVC を大規模な PV にバインドすることができません。使用しているストレージクラスが動的にプロビジョニングされるストレージを使用する場合、典型的な要求のサイズに一致する割り当てサイズを選択します。

2. ファイルを保存して終了します。
3. 次のコマンドを実行して、ストレージ プールを使用して HPP を作成します。

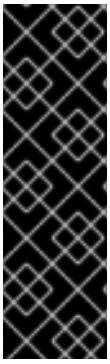
```
$ oc create -f hpp_pvc_template_pool.yaml
```

関連情報

- ストレージプロファイルのカスタマイズ

10.19.2. データボリュームの作成

PVC またはストレージ API のいずれかを使用して、データボリュームを作成できます。



重要

OpenShift Container Platform Container Storage と共に OpenShift Virtualization を使用する場合、仮想マシンディスクの作成時に RBD ブロックモードの永続ボリューム要求 (PVC) を指定します。仮想マシンディスクの場合、RBD ブロックモードのボリュームは効率的で、Ceph FS または RBD ファイルシステムモードの PVC よりも優れたパフォーマンスを提供します。

RBD ブロックモードの PVC を指定するには、'ocs-storagecluster-ceph-rbd' ストレージクラスおよび **VolumeMode: Block** を使用します。

ヒント

可能な限り、ストレージ API を使用して、スペースの割り当てを最適化し、パフォーマンスを最大化します。

ストレージプロファイル は、CDI が管理するカスタムリソースです。関連付けられたストレージクラスに基づく推奨ストレージ設定を提供します。ストレージクラスごとにストレージクラスが割り当てられます。

ストレージプロファイルを使用すると、コーディングを減らし、潜在的なエラーを最小限に抑えながら、データボリュームをすばやく作成できます。

認識されたストレージタイプの場合、CDI は PVC の作成を最適化する値を提供します。ただし、ストレージプロファイルをカスタマイズする場合は、ストレージクラスの自動設定を行うことができます。

10.19.2.1. データボリュームについて

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。データボリュームは、基礎となる永続ボリューム要求 (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。データボリュームは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。

注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.19.2.2. ストレージ API を使用したデータボリュームの作成

ストレージ API を使用してデータボリュームを作成する場合、Containerized Data Interface (CDI) は、選択したストレージクラスでサポートされるストレージのタイプに基づいて、永続ボリューム要求 (PVC) の割り当てを最適化します。データボリューム名、namespace、および割り当てるストレージの量のみを指定する必要があります。

以下に例を示します。

- Ceph RBD を使用する場合、**accessModes** は **ReadWriteMany** に自動設定され、ライブマイグレーションが可能になります。**volumeMode** は、パフォーマンスを最大化するために **Block** に設定されています。

- **volumeMode: Filesystem** を使用する場合、ファイルシステムのオーバーヘッドに対応する必要がある場合は、CDI が追加の領域を自動的に要求します。

以下の YAML では、ストレージ API を使用して、2 ギガバイトの使用可能な領域を持つデータボリュームを要求します。ユーザーは、必要な永続ボリューム要求 (PVC) のサイズを適切に予測するために **volumeMode** を把握する必要はありません。CDI は **accessModes** 属性と **volumeMode** 属性の最適な組み合わせを自動的に選択します。これらの最適値は、ストレージのタイプまたはストレージプロファイルで定義するデフォルトに基づいています。カスタム値を指定する場合は、システムで計算された値を上書きします。

DataVolume 定義の例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <datavolume> 1
spec:
  source:
    pvc: 2
      namespace: "<source_namespace>" 3
      name: "<my_vm_disk>" 4
  storage: 5
    resources:
      requests:
        storage: 2Gi 6
    storageClassName: <storage_class> 7
```

1. 新規データボリュームの名前。
2. インポートのソースが既存の永続ボリューム要求 (PVC) であることを示しています。
3. ソース PVC が存在する namespace。
4. ソース PVC の名前。
5. ストレージ API を使用した割り当てを示します。
6. PVC に要求する利用可能な領域のサイズを指定します。
7. オプション: ストレージクラスの名前。ストレージクラスが指定されていない場合、システムデフォルトのストレージクラスが使用されます。

10.19.2.3. PVC API を使用したデータボリュームの作成

PVC API を使用してデータボリュームを作成する場合、Containerized Data Interface (CDI) は、以下のフィールドに指定する内容に基づいてデータボリュームを作成します。

- **accessModes** (**ReadWriteOnce**、**ReadWrtieMany**、または **ReadOnlyMany**)
- **volumeMode** (**Filesystem** または **Block**)
- **storage** の **capacity** (例: **5Gi**)

以下の YAML では、PVC API を使用して、2 ギガバイトのストレージ容量を持つデータボリュームを割

り当てます。**ReadWriteMany** のアクセスモードを指定して、ライブマイグレーションを有効にします。システムがサポートできる値がわかっているので、デフォルトの **Filesystem** の代わりに **Block** ストレージを指定します。

DataVolume 定義の例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <datavolume> 1
spec:
  source:
    pvc: 2
      namespace: "<source_namespace>" 3
      name: "<my_vm_disk>" 4
  pvc: 5
    accessModes: 6
      - ReadWriteMany
    resources:
      requests:
        storage: 2Gi 7
    volumeMode: Block 8
    storageClassName: <storage_class> 9
```

1. 新規データボリュームの名前。
2. **source** セクションでは、**pvc** はインポートのソースが既存の永続ボリューム要求 (PVC) であることを示しています。
3. ソース PVC が存在する namespace。
4. ソース PVC の名前。
5. PVC API を使用した割り当てを示します。
6. PVC API を使用する場合は **accessModes** が必要です。
7. データボリュームに要求する領域のサイズを指定します。
8. 宛先がブロック PVC であることを指定します。
9. オプションで、ストレージクラスを指定します。ストレージクラスが指定されていない場合、システムデフォルトのストレージクラスが使用されます。

重要

PVC API を使用してデータボリュームを明示的に割り当て、**volumeMode: Block** を使用していない場合は、ファイルシステムのオーバーヘッドを考慮してください。

ファイルシステムのオーバーヘッドは、ファイルシステムのメタデータを維持するために必要な領域のサイズです。ファイルシステムメタデータに必要な領域のサイズは、ファイルシステムに依存します。ストレージ容量要求でファイルシステムのオーバーヘッドに対応できない場合は、基礎となる永続ボリューム要求 (PVC) が仮想マシンディスクに十分に対応できない大きさとなる可能性があります。

ストレージ API を使用する場合、CDI はファイルシステムのオーバーヘッドを考慮し、割り当て要求が正常に実行されるように大きな永続ボリューム要求 (PVC) を要求します。

10.19.2.4. ストレージプロファイルのカスタマイズ

プロビジョナーのストレージクラスの **StorageProfile** オブジェクトを編集してデフォルトパラメーターを指定できます。これらのデフォルトパラメーターは、**DataVolume** オブジェクトで設定されていない場合にのみ永続ボリューム要求 (PVC) に適用されます。

ストレージプロファイルの空の **status** セクションは、ストレージプロビジョナーが Containerized Data Interface (CDI) によって認識されないことを示します。CDI で認識されないストレージプロビジョナーがある場合、ストレージプロファイルをカスタマイズする必要があります。この場合、管理者はストレージプロファイルに適切な値を設定し、割り当てが正常に実行されるようにします。

警告

データボリュームを作成し、YAML 属性を省略し、これらの属性がストレージプロファイルで定義されていない場合は、要求されたストレージは割り当てられず、基礎となる永続ボリューム要求 (PVC) は作成されません。

前提条件

- 計画した設定がストレージクラスとそのプロバイダーでサポートされていることを確認してください。ストレージプロファイルに互換性のない設定を指定すると、ボリュームのプロビジョニングに失敗します。

手順

1. ストレージプロファイルを編集します。この例では、プロビジョナーは CDI によって認識されません。

```
$ oc edit -n openshift-cnv storageprofile <storage_class>
```

ストレージプロファイルの例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
```

```
  name: <unknown_provisioner_class>
# ...
spec: {}
status:
  provisioner: <unknown_provisioner>
  storageClass: <unknown_provisioner_class>
```

2. ストレージプロファイルに必要な属性値を指定します。

ストレージプロファイルの例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <unknown_provisioner_class>
# ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce (1)
    volumeMode:
      Filesystem (2)
status:
  provisioner: <unknown_provisioner>
  storageClass: <unknown_provisioner_class>
```

(1) 選択する **accessModes**

(2) 選択する **volumeMode**。

変更を保存した後、選択した値がストレージプロファイルの **status** 要素に表示されます。

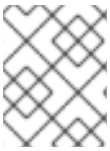
10.19.2.4.1. ストレージプロファイルを使用したデフォルトのクローンストラテジーの設定

ストレージプロファイルを使用してストレージクラスのデフォルトクローンメソッドを設定し、**クローンストラテジー** を作成できます。ストレージベンダーが特定のクローン作成方法のみをサポートする場合などに、クローンストラテジーを設定すると便利です。また、リソースの使用の制限やパフォーマンスの最大化を実現する手法を選択することもできます。

クローン作成ストラテジーは、ストレージプロファイルの **cloneStrategy** 属性を以下の値のいずれかに設定して指定できます。

- **snapshot** が設定されている場合、デフォルトでスナップショットが使用されます。このクローン作成ストラテジーは、一時的なボリュームスナップショットを使用してボリュームのクローンを作成します。ストレージプロビジョナーは、Container Storage Interface (CSI) スナップショットをサポートする必要があります。
- **copy** は、ソース Pod とターゲット Pod を使用して、ソースボリュームからターゲットボリュームにデータをコピーします。ホスト支援型でのクローン作成は、最も効率的な方法です。
- **csi-clone** は、CSI クローン API を使用して、中間ボリュームスナップショットを使用せずに、既存のボリュームのクローンを効率的に作成します。ストレージプロファイルが定義されていない場合にデフォルトで使用される **snapshot** または **copy** とは異なり、CSI ボリュームのク

ローンは、プロビジョナーのストレージクラスの **StorageProfile** オブジェクトに指定した場合にだけ使用されます。



注記

YAML **spec** セクションのデフォルトの **claimPropertySets** を変更せずに、CLI でクローンストラテジーを設定することもできます。

ストレージプロファイルの例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <provisioner_class>
# ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce (1)
    volumeMode:
      Filesystem (2)
  cloneStrategy: csi-clone (3)
status:
  provisioner: <provisioner>
  storageClass: <provisioner_class>
```

(1) アクセスモードを指定します。

(2) ボリュームモードを指定します。

(3) デフォルトのクローン戦略を指定します。

10.19.2.5. 関連情報

- ストレージクラスの作成について
- デフォルトのファイルシステムオーバーヘッド値の上書き
- smart-cloning を使用したデータボリュームのクローン作成

10.19.3. ファイルシステムオーバーヘッドの PVC 領域の確保

デフォルトでは、OpenShift Virtualization は、**Filesystem** ボリュームモードを使用する永続ボリューム要求 (PVC) のファイルシステムオーバーヘッドデータ用に領域を確保します。この目的のためにグローバルに、および特定のストレージクラス用に領域を予約する割合を設定できます。

10.19.3.1. ファイルシステムのオーバーヘッドが仮想マシンディスクの領域に影響を与える仕組み

仮想マシンディスクを **Filesystem** ボリュームモードを使用する永続ボリューム要求 (PVC) に追加する場合、PVC で十分な容量があることを確認する必要があります。

- 仮想マシンディスク。

- メタデータなどのファイルシステムオーバーヘッド用に予約された領域

デフォルトでは、OpenShift Virtualization は PVC 領域の 5.5% をオーバーヘッド用に予約し、その分、仮想マシンディスクに使用できる領域を縮小します。

HCO オブジェクトを編集して、別のオーバーヘッド値を設定できます。値はグローバルに変更でき、特定のストレージクラスの値を指定できます。

10.19.3.2. デフォルトのファイルシステムオーバーヘッド値の上書き

HCO オブジェクトの **spec.filesystemOverhead** 属性を編集することで、OpenShift Virtualization がファイルシステムオーバーヘッド用に予約する永続ボリューム要求 (PVC) 領域の量を変更します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. 次のコマンドを実行して、**HCO** オブジェクトを編集用に開きます。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

2. **spec.filesystemOverhead** フィールドを編集して、選択した値でデータを設定します。

```
# ...
spec:
  filesystemOverhead:
    global: "<new_global_value>" 1
    storageClass:
      <storage_class_name>: "<new_value_for_this_storage_class>" 2
```

1. まだ値が設定されていないストレージクラスに使用されるデフォルトのファイルシステムオーバーヘッドの割合。たとえば、**global: "0.07"** は、ファイルシステムのオーバーヘッド用に PVC の 7% を確保します。

2. 指定されたストレージクラスのファイルシステムのオーバーヘッドの割合 (パーセンテージ)。たとえば、**mystorageclass: "0.04"** は、**mystorageclass** ストレージクラスの PVC のデフォルトオーバーヘッド値を 4% に変更します。

3. エディターを保存して終了し、**HCO** オブジェクトを更新します。

検証

- 次のいずれかのコマンドを実行して、**CDIConfig** ステータスを表示し、変更を確認します。一般的に **CDIConfig** への変更を確認するには以下を実行します。

```
$ oc get cdiconfig -o yaml
```

CDIConfig に対する 特定の変更を表示するには以下を実行します。

```
$ oc get cdiconfig -o jsonpath='{.items..status.filesystemOverhead}'
```

10.19.4. コンピュートリソースクォータを持つ namespace で機能する CDI の設定

Containerized Data Importer (CDI) を使用して、CPU およびメモリーリソースの制限が適用される namespace に仮想マシンディスクをインポートし、アップロードし、そのクローンを作成できるようになりました。

10.19.4.1. namespace の CPU およびメモリークォータについて

ResourceQuota オブジェクトで定義される **リソースクォータ** は、その namespace 内のリソースが消費できるコンピュートリソースの全体量を制限する制限を namespace に課します。

HyperConverged カスタムリソース (CR) は、Containerized Data Importer (CDI) のユーザー設定を定義します。CPU とメモリーの要求値と制限値は、デフォルト値の **0** に設定されています。これにより、コンピュートリソース要件を指定しない CDI によって作成される Pod にデフォルト値が付与され、クォータで制限される namespace での実行が許可されます。

10.19.4.2. CPU およびメモリーのデフォルトの上書き

HyperConverged カスタムリソース (CR) に **spec.resourceRequirements.storageWorkloads** スタンザを追加して、CPU およびメモリー要求のデフォルト設定とユースケースの制限を変更します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. 以下のコマンドを実行して、**HyperConverged** CR を編集します。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

2. **spec.resourceRequirements.storageWorkloads** スタンザを CR に追加し、ユースケースに基づいて値を設定します。以下に例を示します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  resourceRequirements:
    storageWorkloads:
      limits:
        cpu: "500m"
        memory: "2Gi"
      requests:
        cpu: "250m"
        memory: "1Gi"
```

3. エディターを保存して終了し、**HyperConverged** CR を更新します。

10.19.4.3. 関連情報

- プロジェクトごとのリソースクォータ

10.19.5. データボリュームアノテーションの管理

データボリューム (DV) アノテーションを使用して Pod の動作を管理できます。1 つ以上のアノテーションをデータボリュームに追加してから、作成されたインポーター Pod に伝播できます。

10.19.5.1. 例: データボリュームアノテーション

以下の例は、インポーター Pod が使用するネットワークを制御するためにデータボリューム (DV) アノテーションを設定する方法を示しています。**v1.multus-cni.io/default-network: bridge-network** アノテーションにより、Pod は **bridge-network** という名前の multus ネットワークをデフォルトネットワークとして使用します。インポーター Pod にクラスターからのデフォルトネットワークとセカンダリー multus ネットワークの両方を使用させる必要がある場合は、**k8s.v1.cni.cncf.io/networks: <network_name>** アノテーションを使用します。

Multus ネットワークアノテーションの例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: dv-ann
  annotations:
    v1.multus-cni.io/default-network: bridge-network (1)
spec:
  source:
    http:
      url: "example.exampleurl.com"
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 1Gi
```

(1) Multus ネットワークアノテーション

10.19.6. データボリュームの事前割り当ての使用

Containerized Data Importer は、データボリュームの作成時に書き込みパフォーマンスを向上させるために、ディスク領域を事前に割り当てることができます。

特定のデータボリュームの事前割り当てを有効にできます。

10.19.6.1. 事前割り当てについて

Containerized Data Importer (CDI) は、データボリュームに QEMU 事前割り当てモードを使用し、書き込みパフォーマンスを向上できます。操作のインポートおよびアップロードには、事前割り当てモードを使用できます。また、空のデータボリュームを作成する際にも使用できます。

事前割り当てが有効化されている場合、CDI は基礎となるファイルシステムおよびデバイスタイプに応じて、より適切な事前割り当て方法を使用します。

fallocate

ファイルシステムがこれをサポートする場合、CDI は **posix_fallocate** 関数を使用して領域を事前に割り当てるためにオペレーティングシステムの **fallocate** 呼び出しを使用します。これは、ブロックを割り当て、それらを未初期化としてマークします。

full

fallocate モードを使用できない場合は、基礎となるストレージにデータを書き込むことで、**full** モードがイメージの領域を割り当てます。ストレージの場所によっては、空の割り当て領域がすべてゼロになる場合があります。

10.19.6.2. データボリュームの事前割り当ての有効化

データボリュームマニフェストに **spec.preallocation** フィールドを含めることにより、特定のデータボリュームの事前割り当てを有効にできます。Web コンソールで、または OpenShift CLI (**oc**) を使用して、事前割り当てモードを有効化することができます。

事前割り当てモードは、すべての CDI ソースタイプでサポートされます。

手順

- データボリュームマニフェストの **spec.preallocation** フィールドを指定します。

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: preallocated-datavolume
spec:
  source: 1
    ...
  pvc:
    ...
  preallocation: true 2
```

1. すべての CDI ソースタイプは事前割り当てをサポートしますが、クローンの操作では事前割り当ては無視されます。
2. **preallocation** フィールドは、デフォルトで false に設定されるブール値です。

10.19.7. Web コンソ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

Web コンソールを使用して、ローカルに保存されたディスクイメージファイルをアップロードできます。

10.19.7.1. 前提条件

- 仮想マシンのイメージファイルには、IMG、ISO、または QCOW2 形式のファイルを使用する必要がある。
- 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は、この操作が正常に完了するように、まずは ストレージクラスを定義するフィルタリングCDI スクラッチ領域を用意 すること。

10.19.7.2. 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*

✓ 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

10.19.7.3. Web コンソールを使用したイメージファイルのアップロード

Web コンソールを使用して、イメージファイルを新規の永続ボリューム要求 (PVC) にアップロードします。この PVC を後で使用して、イメージを新規の仮想マシンに割り当てることができます。

前提条件

- 以下のいずれかが必要である。
  - ISO または IMG 形式のいずれかの raw 仮想マシンイメージファイル。
  - QCOW2 形式の仮想マシンのイメージファイル。
- 最善の結果を得るには、アップロードする前にイメージファイルを以下のガイドラインに従って圧縮すること。
  - **xz** または **gzip** を使用して raw イメージファイルを圧縮します。

注記

圧縮された raw イメージファイルを使用すると、最も効率的にアップロードできます。

  - クライアントについて推奨される方法を使用して、QCOW2 イメージファイルを圧縮します。
    - Linux クライアントを使用する場合は、virt-sparsify ツールを使用して、QCOW2 ファイルをスパース化 (**sparsify**) します。
    - Windows クライアントを使用する場合は、**xz** または **gzip** を使用して QCOW2 ファイルを圧縮します。

手順

1. Web コンソールのサイドメニューから、**Storage → Persistent Volume Claims**をクリックします。
2. **Create Persistent Volume Claim**ドロップダウンリストをクリックし、これを拡張します。
3. **With Data Upload Form**をクリックし、**Upload Data to Persistent Volume Claim**ページを開きます。
4. **Browse** をクリックし、ファイルマネージャーを開き、アップロードするイメージを選択するか、ファイルを **Drag a file here or browse to upload**フィールドにドラッグします。
5. オプション: 特定のオペレーティングシステムのデフォルトイメージとしてこのイメージを設定します。
   a. **Attach this data to a virtual machine operating system**チェックボックスを選択します。
   b. リストからオペレーティングシステムを選択します。
6. **Persistent Volume Claim Name**フィールドには、一意の名前が自動的に入力され、これを編集することはできません。PVC に割り当てられた名前をメモし、必要に応じてこれを後で特定できるようにします。
7. **Storage Class** リストからストレージクラスを選択します。
8. **Size** フィールドに PVC のサイズ値を入力します。ドロップダウンリストから、対応する測定単位を選択します。

> **警告**
>
> PVC サイズは圧縮解除された仮想ディスクのサイズよりも大きくなければなりません。

9. 選択したストレージクラスに一致する **Access Mode** を選択します。
10. **Upload** をクリックします。

10.19.7.4. 関連情報

- 事前割り当てモードを設定 して、データボリューム操作の書き込みパフォーマンスを向上させます。

10.19.8. 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たディスクイメージを新規または既存の永続ボリューム要求 (PVC) にアップロードできます。

10.19.8.1. 前提条件

- virtctl をインストールします。

- この操作を正常に完了するためには、ストレージクラスを定義するか、CDI スクラッチ領域を用意 しなければならない場合があります。

10.19.8.2. データボリュームについて

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。データボリュームは、基礎となる永続ボリューム要求 (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。データボリュームは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。

注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.19.8.3. アップロードデータボリュームの作成

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** データソースでデータボリュームを手動で作成できます。

手順

1. **spec: source: upload{}** を指定するデータボリューム設定を作成します。

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <upload-datavolume> 1
spec:
  source:
    upload: {}
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> 2
```

1. データボリュームの名前。
2. データボリュームのサイズ。この値がアップロードするディスクのサイズ以上であることを確認します。

2. 以下のコマンドを実行してデータボリュームを作成します。

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

10.19.8.4. ローカルディスクイメージのデータボリュームへのアップロード

virtctl CLI ユーティリティーを使用して、ローカル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データボリューム (DV) にアップロードできます。この手順の実行時に、すでにクラスターに存在する DV を使用するか、新規の DV を作成することができます。

注記

ローカルディスクイメージのアップロード後に、これを仮想マシンに追加できます。

前提条件

- 以下のいずれかが必要である。
  - ISO または IMG 形式のいずれかの raw 仮想マシンイメージファイル。
  - QCOW2 形式の仮想マシンのイメージファイル。
- 最善の結果を得るには、アップロードする前にイメージファイルを以下のガイドラインに従って圧縮すること。
  - **xz** または **gzip** を使用して raw イメージファイルを圧縮します。

注記

圧縮された raw イメージファイルを使用すると、最も効率的にアップロードできます。

  - クライアントについて推奨される方法を使用して、QCOW2 イメージファイルを圧縮します。
    - Linux クライアントを使用する場合は、virt-sparsify ツールを使用して、QCOW2 ファイルをスパース化 (**sparsify**) します。
    - Windows クライアントを使用する場合は、**xz** または **gzip** を使用して QCOW2 ファイルを圧縮します。
- **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。
   - 使用するアップロードデータボリュームの名前。このデータボリュームが存在しない場合、これは自動的に作成されます。
   - データボリュームのサイズ (アップロード手順の実行時に作成する必要がある場合)。サイズはディスクイメージのサイズ以上である必要があります。
   - アップロードする必要のある仮想マシンディスクイメージのファイルの場所。
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。直前の手順で特定したパラメーターを指定します。以下に例を示します。

```
$ virtctl image-upload dv <datavolume_name> \ 1
--size=<datavolume_size> \ 2
--image-path=</path/to/image> \ 3
```

1. データボリュームの名前。
2. データボリュームのサイズ。例: **--size=500Mi**、**--size=1G**
3. 仮想マシンディスクイメージのファイルパス。

注記

- 新規データボリュームを作成する必要がない場合は、**--size** パラメーターを省略し、**--no-create** フラグを含めます。
- ディスクイメージを PVC にアップロードする場合、PVC サイズは圧縮されていない仮想ディスクのサイズよりも大きくなければなりません。
- 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. オプション:データボリュームが作成されたことを確認するには、以下のコマンドを実行してすべてのデータボリュームを表示します。

```
$ oc get dvs
```

10.19.8.5. 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*

✓ 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

10.19.8.6. 関連情報

- 事前割り当てモードを設定 して、データボリューム操作の書き込みパフォーマンスを向上させます。

10.19.9. ブロックストレージ永続ボリューム要求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、ローカルディスクイメージをブロック永続ボリューム要求 (PVC) 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永続ボリュームを使用し、このブロックボリュームを **upload** データボリュームに関連付け、 **virtctl** を使用してローカルディスクイメージを PVC にアップロードできます。

10.19.9.1. 前提条件

- virtctl をインストールします。
- この操作を正常に完了するためには、ストレージクラスを定義するか、CDI スクラッチ領域を用意 しなければならない場合があります。

10.19.9.2. データボリュームについて

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。データボリュームは、基礎となる永続ボリューム要求 (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。データボリュームは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。

注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.19.9.3. ブロック永続ボリュームについて

ブロック永続ボリューム (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、オーバーヘッドを削減することで、仮想マシンのパフォーマンス上の利点をもたらすことができます。

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

10.19.9.4. ローカルブロック永続ボリュームの作成

データボリュームを含むブロックストレージに仮想マシンイメージをインポートする場合は、使用可能なローカルブロック永続ボリュームが必要です。

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永続ボリューム (PV) を作成します。次に、このループデバイスを PV マニフェストで **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** マニフェストを作成します。

```
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 にストレージクラスを設定します。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永続ボリュームのファイル名。

10.19.9.5. アップロードデータボリュームの作成

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** データソースでデータボリュームを手動で作成できます。

手順

1. **spec: source: upload{}** を指定するデータボリューム設定を作成します。

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <upload-datavolume> 1
spec:
  source:
     upload: {}
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> 2
```

1 データボリュームの名前。

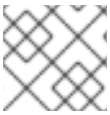
2 データボリュームのサイズ。この値がアップロードするディスクのサイズ以上であることを確認します。

2. 以下のコマンドを実行してデータボリュームを作成します。

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

10.19.9.6. ローカルディスクイメージのデータボリュームへのアップロード

virtctl CLI ユーティリティーを使用して、ローカル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データボリューム (DV) にアップロードできます。この手順の実行時に、すでにクラスターに存在する DV を使用するか、新規の DV を作成することができます。



注記

ローカルディスクイメージのアップロード後に、これを仮想マシンに追加できます。

前提条件

- 以下のいずれかが必要である。

- ISO または IMG 形式のいずれかの raw 仮想マシンイメージファイル。
- QCOW2 形式の仮想マシンのイメージファイル。

- 最善の結果を得るには、アップロードする前にイメージファイルを以下のガイドラインに従って圧縮すること。
  - **xz** または **gzip** を使用して raw イメージファイルを圧縮します。

注記

圧縮された raw イメージファイルを使用すると、最も効率的にアップロードできます。

  - クライアントについて推奨される方法を使用して、QCOW2 イメージファイルを圧縮します。
    - Linux クライアントを使用する場合は、virt-sparsify ツールを使用して、QCOW2 ファイルをスパース化 (**sparsify**) します。
    - Windows クライアントを使用する場合は、**xz** または **gzip** を使用して QCOW2 ファイルを圧縮します。

- **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。
   - 使用するアップロードデータボリュームの名前。このデータボリュームが存在しない場合、これは自動的に作成されます。
   - データボリュームのサイズ (アップロード手順の実行時に作成する必要がある場合)。サイズはディスクイメージのサイズ以上である必要があります。
   - アップロードする必要のある仮想マシンディスクイメージのファイルの場所。
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。直前の手順で特定したパラメーターを指定します。以下に例を示します。

```
$ virtctl image-upload dv <datavolume_name> \ 1
--size=<datavolume_size> \ 2
--image-path=</path/to/image> \ 3
```

1. データボリュームの名前。
2. データボリュームのサイズ。例: **--size=500Mi**、**--size=1G**
3. 仮想マシンディスクイメージのファイルパス。

注記

- 新規データボリュームを作成する必要がない場合は、**--size** パラメーターを省略し、**--no-create** フラグを含めます。
- ディスクイメージを PVC にアップロードする場合、PVC サイズは圧縮されていない仮想ディスクのサイズよりも大きくなければなりません。
- 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. オプション:データボリュームが作成されたことを確認するには、以下のコマンドを実行してすべてのデータボリュームを表示します。

```
$ oc get dvs
```

10.19.9.7. 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*

✓ 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

10.19.9.8. 関連情報

- 事前割り当てモードを設定 して、データボリューム操作の書き込みパフォーマンスを向上させます。

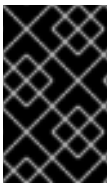
10.19.10. 仮想マシンスナップショットの管理

仮想マシンの電源がオフ (オフライン) またはオン (オンライン) であるかに拘らず、仮想マシンの仮想マシン (VM) スナップショットを作成および削除できます。電源オフ (オフライン) 状態の仮想マシンに対してのみ復元が可能です。OpenShift Virtualization は、以下にある仮想マシンのスナップショットを

サポートします。

- Red Hat OpenShift Data Foundation
- Kubernetes Volume Snapshot API をサポートする Container Storage Interface (CSI) ドライバーを使用するその他のクラウドストレージプロバイダー

オンラインスナップショットのデフォルト期限は 5 分 (**5m**) で、必要に応じて変更できます。



重要

オンラインスナップショットは、ホットプラグされた仮想ディスクを持つ仮想マシンでサポートされます。ただし、仮想マシンの仕様に含まれていないホットプラグされたディスクは、スナップショットに含まれません。

注記

整合性が最も高いオンライン (実行状態) の仮想マシンのスナップショットを作成するには、QEMU ゲストエージェントをインストールします。

QEMU ゲストエージェントは、システムのワークロードに応じて、可能な限り仮想マシンのファイルシステムの休止しようとすることで一貫性のあるスナップショットを取得します。これにより、スナップショットの作成前にインフライトの I/O がディスクに書き込まれるようになります。ゲストエージェントが存在しない場合は、休止はできず、ベストエフォートスナップショットが作成されます。スナップショットの作成条件は、Web コンソールまたは CLI に表示されるスナップショットの指示に反映されます。

10.19.10.1. 仮想マシンスナップショットについて

スナップショット は、特定の時点における仮想マシン (VM) の状態およびデータを表します。スナップショットを使用して、バックアップおよび障害復旧のために既存の仮想マシンを (スナップショットで表される) 以前の状態に復元したり、以前の開発バージョンに迅速にロールバックしたりできます。

仮想マシンのスナップショットは、電源がオフ (停止状態) またはオン (実行状態) の仮想マシンから作成されます。

実行中の仮想マシンのスナップショットを作成する場合には、コントローラーは QEMU ゲストエージェントがインストールされ、実行中であることを確認します。その場合、スナップショットを取得する前に仮想マシンファイルシステムをフリーズし、スナップショットを取得した後にファイルシステムをフリーズ解除します。

スナップショットは、仮想マシンに割り当てられた各 Container Storage Interface (CSI) ボリュームのコピーと、仮想マシンの仕様およびメタデータのコピーを保存します。スナップショットは作成後に変更できません。

仮想マシンスナップショット機能を使用すると、クラスター管理者、およびアプリケーション開発者は以下を実行できます。

- 新規 SCC の作成
- 特定の仮想マシンに割り当てられているすべてのスナップショットのリスト表示
- スナップショットからの仮想マシンの復元
- 既存の仮想マシンスナップショットの削除

10.19.10.1.1. 仮想マシンスナップショットコントローラーおよびカスタムリソース定義 (CRD)

仮想マシンスナップショット機能では、スナップショットを管理するための CRD として定義された 3 つの新規 API オブジェクトが導入されました。

- **VirtualMachineSnapshot**: スナップショットを作成するユーザー要求を表します。これには、仮想マシンの現在の状態に関する情報が含まれます。
- **VirtualMachineSnapshotContent**: クラスター上のプロビジョニングされたリソース (スナップショット) を表します。これは、仮想マシンのスナップショットコントローラーによって作成され、仮想マシンの復元に必要なすべてのリソースへの参照が含まれます。
- **VirtualMachineRestore**: スナップショットから仮想マシンを復元するユーザー要求を表します。

仮想マシンスナップショットコントローラーは、1 対 1 のマッピングで、**VirtualMachineSnapshotContent** オブジェクトを、この作成に使用した **VirtualMachineSnapshot** オブジェクトにバインドします。

10.19.10.2. 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 は広範な使用が可能で、Red Hat Enterprise Linux (RHEL) 仮想マシン (VM) においてデフォルトで使用できます。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。

注記

整合性が最も高いオンライン (実行状態) の仮想マシンのスナップショットを作成するには、QEMU ゲストエージェントをインストールします。

QEMU ゲストエージェントは、システムのワークロードに応じて、可能な限り仮想マシンのファイルシステムの休止しようとすることで一貫性のあるスナップショットを取得します。これにより、スナップショットの作成前にインフライトの I/O がディスクに書き込まれるようになります。ゲストエージェントが存在しない場合は、休止はできず、ベストエフォートスナップショットが作成されます。スナップショットの作成条件は、Web コンソールまたは CLI に表示されるスナップショットの指示に反映されます。

手順

1. コンソールのいずれか、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. サービスに永続性があることを確認し、これを起動します。

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

検証

1. 次のコマンドを実行して、**AgentConnected** が VM 仕様にリストされていることを確認します。

```
$ oc get vm <vm_name>
```

10.19.10.3. 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 仮想マシンの場合には、QEMU ゲストエージェントは VirtIO ドライバーに含まれます。既存または新規の Windows インストールにドライバーをインストールします。

注記

整合性が最も高いオンライン (実行状態) の仮想マシンのスナップショットを作成するには、QEMU ゲストエージェントをインストールします。

QEMU ゲストエージェントは、システムのワークロードに応じて、可能な限り仮想マシンのファイルシステムの休止しようとすることで一貫性のあるスナップショットを取得します。これにより、スナップショットの作成前にインフライトの I/O がディスクに書き込まれるようになります。ゲストエージェントが存在しない場合は、休止はできず、ベストエフォートスナップショットが作成されます。スナップショットの作成条件は、Web コンソールまたは CLI に表示されるスナップショットの指示に反映されます。

手順

1. Windows Guest Operating System (OS) で、**File Explorer** を使用して、**virtio-win** CD ドライブの **guest-agent** ディレクトリーに移動します。
2. **qemu-ga-x86_64.msi** インストーラーを実行します。

検証

1. 次のコマンドを実行して、出力に **QEMU Guest Agent** が含まれていることを確認します。

```
$ net start
```

10.19.10.3.1. 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

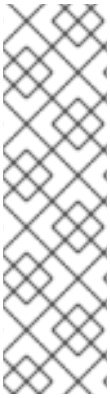
1. 仮想マシンを起動し、グラフィカルコンソールに接続します。
2. Windows ユーザーセッションにログインします。
3. **Device Manager** を開き、**Other devices** を拡張して、**Unknown device** をリスト表示します。
   a. **Device Properties** を開いて、不明なデバイスを特定します。デバイスを右クリックし、**Properties** を選択します。
   b. **Details** タブをクリックし、**Property** リストで **Hardware Ids** を選択します。

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。

10.19.10.4. Web コンソールでの仮想マシンのスナップショットの作成

Web コンソールを使用して仮想マシン (VM) を作成することができます。



注記

整合性が最も高いオンライン (実行状態) の仮想マシンのスナップショットを作成するには、QEMU ゲストエージェントをインストールします。

QEMU ゲストエージェントは、システムのワークロードに応じて、可能な限り仮想マシンのファイルシステムの休止しようとすることで一貫性のあるスナップショットを取得します。これにより、スナップショットの作成前にインフライトの I/O がディスクに書き込まれるようになります。ゲストエージェントが存在しない場合は、休止はできず、ベストエフォートスナップショットが作成されます。スナップショットの作成条件は、Web コンソールまたは CLI に表示されるスナップショットの指示に反映されます。

仮想マシンスナップショットには、以下の要件を満たすディスクのみが含まれます。

- データボリュームまたは永続ボリューム要求 (PVC) のいずれかでなければなりません。
- Container Storage Interface (CSI) ボリュームスナップショットをサポートするストレージクラスに属している必要があります。

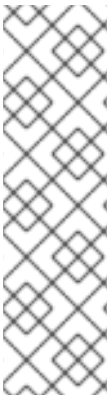
手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. 仮想マシンが実行中の場合は、**Actions → Stop** をクリックして電源を切ります。
4. **Snapshots** タブをクリックしてから **Take Snapshot** をクリックします。
5. **Snapshot Name** およびオプションの **Description** フィールドに入力します。
6. **Disks included in this Snapshot**を拡張し、スナップショットに組み込むストレージボリュームを表示します。
7. 仮想マシンにスナップショットに追加できないディスクがあり、それでも続行する場合は、**I am aware of this warning and wish to proceed** チェックボックスをオンにします。

8. **Save** をクリックします。

10.19.10.5. CLI での仮想マシンのスナップショットの作成

VirtualMachineSnapshot オブジェクトを作成し、オフラインまたはオンラインの仮想マシンの仮想マシン (VM) スナップショットを作成できます。KubeVirt は QEMU ゲストエージェントと連携し、オンライン仮想マシンのスナップショットを作成します。



注記

整合性が最も高いオンライン (実行状態) の仮想マシンのスナップショットを作成するには、QEMU ゲストエージェントをインストールします。

QEMU ゲストエージェントは、システムのワークロードに応じて、可能な限り仮想マシンのファイルシステムの休止しようとすることで一貫性のあるスナップショットを取得します。これにより、スナップショットの作成前にインフライトの I/O がディスクに書き込まれるようになります。ゲストエージェントが存在しない場合は、休止はできず、ベストエフォートスナップショットが作成されます。スナップショットの作成条件は、Web コンソールまたは CLI に表示されるスナップショットの指示に反映されます。

前提条件

- 永続ボリューム要求 (PVC) が Container Storage Interface (CSI) ボリュームスナップショットをサポートするストレージクラスにあることを確認している。
- OpenShift CLI (**oc**) がインストールされている。
- オプション: スナップショットを作成する仮想マシンの電源を切っている。

手順

1. YAML ファイルを作成し、新規の **VirtualMachineSnapshot** の名前およびソース仮想マシンの名前を指定する **VirtualMachineSnapshot** オブジェクトを定義します。
以下に例を示します。

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineSnapshot
metadata:
  name: my-vmsnapshot 1
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: my-vm 2
```

1. 新規 **VirtualMachineSnapshot** オブジェクトの名前。
2. ソース仮想マシンの名前。

2. **VirtualMachineSnapshot** リソースを作成します。スナップコントローラーは **VirtualMachineSnapshotContent** オブジェクトを作成し、これを **VirtualMachineSnapshot** にバインドし、 **VirtualMachineSnapshot** オブジェクトの **status** および **readyToUse** フィールドを更新します。

```
$ oc create -f <my-vmsnapshot>.yaml
```

3. オプション: オンラインスナップショットを作成している場合は、**wait** コマンドを使用して、スナップショットのステータスを監視できます。

   a. 以下のコマンドを入力します。

   ```
   $ oc wait my-vm my-vmsnapshot --for condition=Ready
   ```

   b. スナップショットのステータスを確認します。

   - **InProgress**: オンラインスナップショットの操作が進行中です。
   - **Succeeded**: オンラインスナップショット操作が正常に完了しました。
   - **Failed**: オンラインスナップショットの操作に失敗しました。

注記

オンラインスナップショットのデフォルト期限は 5 分 (**5m**) です。スナップショットが 5 分後に正常に完了しない場合には、ステータスが **failed** に設定されます。その後、ファイルシステムと仮想マシンのフリーズが解除され、失敗したスナップショットイメージが削除されるまで、ステータスは **failed** のままになります。

デフォルトの期限を変更するには、仮想マシンスナップショット仕様に **FailureDeadline** 属性を追加して、スナップショット操作がタイムアウトするまでの時間を分単位 (**m**) または秒単位 (**s**) で指定します。

期限を指定しない場合は、**0** を指定できますが、仮想マシンが応答しなくなる可能性があるため、通常は推奨していません。

m または **s**などの時間の単位を指定しない場合、デフォルトは秒 (**s**) です。

検証

1. **VirtualMachineSnapshot** オブジェクトが作成されており、**VirtualMachineSnapshotContent** にバインドされていることを確認します。**readyToUse** フラグを **true** に設定する必要があります。

```
$ oc describe vmsnapshot <my-vmsnapshot>
```

出力例

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineSnapshot
metadata:
  creationTimestamp: "2020-09-30T14:41:51Z"
  finalizers:
  - snapshot.kubevirt.io/vmsnapshot-protection
  generation: 5
  name: mysnap
  namespace: default
  resourceVersion: "3897"
```

```
  selfLink:
/apis/snapshot.kubevirt.io/v1beta1/namespaces/default/virtualmachinesnapshots/my-
vmsnapshot
  uid: 28eedf08-5d6a-42c1-969c-2eda58e2a78d
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: my-vm
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:42:03Z"
    reason: Operation complete
    status: "False" 1
    type: Progressing
  - lastProbeTime: null
    lastTransitionTime: "2020-09-30T14:42:03Z"
    reason: Operation complete
    status: "True" 2
    type: Ready
  creationTime: "2020-09-30T14:42:03Z"
  readyToUse: true 3
  sourceUID: 355897f3-73a0-4ec4-83d3-3c2df9486f4f
  virtualMachineSnapshotContentName: vmsnapshot-content-28eedf08-5d6a-42c1-969c-
2eda58e2a78d 4
```

1. **Progressing** 状態の **status** フィールドは、スナップショットが作成中であるかどうかを指定します。
2. **Ready** 状態の **status** フィールドは、スナップショットの作成プロセスが完了しているかどうかを指定します。
3. スナップショットを使用する準備ができているかどうかを指定します。
4. スナップショットが、スナップショットコントローラーで作成される **VirtualMachineSnapshotContent** オブジェクトにバインドされるように指定します。

2. **VirtualMachineSnapshotContent** リソースの **spec:volumeBackups** プロパティーをチェックし、予想される PVC がスナップショットに含まれることを確認します。

10.19.10.6. スナップショット指示を使用したオンラインスナップショット作成の確認

スナップショットの表示は、オンライン仮想マシン (VM) スナップショット操作に関するコンテキスト情報です。オフラインの仮想マシン (VM) スナップショット操作では、指示は利用できません。イベントは、オンラインスナップショット作成の詳説する際に役立ちます。

前提条件

- 指示を表示するには、CLI または Web コンソールを使用してオンライン仮想マシンスナップショットの作成を試行する必要があります。

手順

1. 以下のいずれかを実行して、スナップショット指示からの出力を表示します。
   - CLI で作成されたスナップショットの場合には、**status** フィールドの **VirtualMachineSnapshot** オブジェクト YAML にあるインジケーターの出力を確認します。
   - Web コンソールを使用して作成されたスナップショットの場合には、**Snapshot details** 画面で **VirtualMachineSnapshot > Status** をクリックします。
2. オンライン仮想マシンのスナップショットのステータスを確認します。
   - **Online** は、仮想マシンがオンラインスナップショットの作成時に実行されていることを示します。
   - **NoGuestAgent** は、QEMU ゲストエージェントがオンラインのスナップショットの作成時に実行されていないことを示します。QEMU ゲストエージェントがインストールされていないか、実行されていないか、別のエラーが原因で、QEMU ゲストエージェントを使用してファイルシステムをフリーズしてフリーズを解除できませんでした。

10.19.10.7. Web コンソールでのスナップショットからの仮想マシンの復元

仮想マシン (VM) は、Web コンソールのスナップショットで表される以前の設定に復元できます。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. 仮想マシンが実行中の場合は、**Actions → Stop** をクリックして電源を切ります。
4. **Snapshots** タブをクリックします。このページには、仮想マシンに関連付けられたスナップショットのリストが表示されます。
5. 仮想マシンのスナップショットを復元するには、以下のいずれかの方法を選択します。
   a. 仮想マシンを復元する際にソースとして使用するスナップショットの場合は、**Restore** をクリックします。
   b. スナップショットを選択して **Snapshot Details** 画面を開き、**Actions → Restore VirtualMachineSnapshot** をクリックします。
6. 確認のポップアップウィンドウで **Restore** をクリックし、仮想マシンをスナップショットで表される以前の設定に戻します。

10.19.10.8. CLI でのスナップショットからの仮想マシンの復元

仮想マシンスナップショットを使用して、既存の仮想マシン (VM) を以前の設定に復元できます。オフラインの仮想マシンスナップショットからしか復元できません。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- 以前の状態に復元する仮想マシンの電源を切ること。

手順

1. 復元する仮想マシンの名前およびソースとして使用されるスナップショットの名前を指定する **VirtualMachineRestore** オブジェクトを定義するために YAML ファイルを作成します。以下に例を示します。

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineRestore
metadata:
  name: my-vmrestore (1)
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: my-vm (2)
  virtualMachineSnapshotName: my-vmsnapshot (3)
```

(1) 新規 **VirtualMachineRestore** オブジェクトの名前。

(2) 復元するターゲット仮想マシンの名前。

(3) ソースとして使用する **VirtualMachineSnapshot** オブジェクトの名前。

2. **VirtualMachineRestore** リソースを作成します。スナップショットコントローラーは、**VirtualMachineRestore** オブジェクトのステータスフィールドを更新し、既存の仮想マシン設定をスナップショットのコンテンツに置き換えます。

```
$ oc create -f <my-vmrestore>.yaml
```

検証

- 仮想マシンがスナップショットで表される以前の状態に復元されていることを確認します。**complete** フラグは **true** に設定される必要があります。

```
$ oc get vmrestore <my-vmrestore>
```

出力例

```
apiVersion: snapshot.kubevirt.io/v1beta1
kind: VirtualMachineRestore
metadata:
creationTimestamp: "2020-09-30T14:46:27Z"
generation: 5
name: my-vmrestore
namespace: default
ownerReferences:
- apiVersion: kubevirt.io/v1
  blockOwnerDeletion: true
  controller: true
  kind: VirtualMachine
  name: my-vm
  uid: 355897f3-73a0-4ec4-83d3-3c2df9486f4f
  resourceVersion: "5512"
```

```
  selfLink: /apis/snapshot.kubevirt.io/v1beta1/namespaces/default/virtualmachinerestores/my-
vmrestore
  uid: 71c679a8-136e-46b0-b9b5-f57175a6a041
  spec:
    target:
      apiGroup: kubevirt.io
      kind: VirtualMachine
      name: my-vm
  virtualMachineSnapshotName: my-vmsnapshot
  status:
  complete: true (1)
  conditions:
  - lastProbeTime: null
  lastTransitionTime: "2020-09-30T14:46:28Z"
  reason: Operation complete
  status: "False" (2)
  type: Progressing
  - lastProbeTime: null
  lastTransitionTime: "2020-09-30T14:46:28Z"
  reason: Operation complete
  status: "True" (3)
  type: Ready
  deletedDataVolumes:
  - test-dv1
  restoreTime: "2020-09-30T14:46:28Z"
  restores:
  - dataVolumeName: restore-71c679a8-136e-46b0-b9b5-f57175a6a041-datavolumedisk1
  persistentVolumeClaim: restore-71c679a8-136e-46b0-b9b5-f57175a6a041-
datavolumedisk1
  volumeName: datavolumedisk1
  volumeSnapshotName: vmsnapshot-28eedf08-5d6a-42c1-969c-2eda58e2a78d-volume-
datavolumedisk1
```

(1) 仮想マシンをスナップショットで表される状態に復元するプロセスが完了しているかどうかを指定します。

(2) **Progressing** 状態の **status** フィールドは、仮想マシンが復元されているかどうかを指定します。

(3) **Ready** 状態の **status** フィールドは、仮想マシンの復元プロセスが完了しているかどうかを指定します。

10.19.10.9. Web コンソールでの仮想マシンのスナップショットの削除

Web コンソールを使用して既存の仮想マシンスナップショットを削除できます。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Snapshots** タブをクリックします。このページには、仮想マシンに関連付けられたスナップショットの一覧が表示されます。

4. 削除する仮想マシンスナップショットの Options メニュー  をクリックして、**Delete VirtualMachineSnapshot** を選択します。

5. 確認のポップアップウィンドウで、**Delete** をクリックしてスナップショットを削除します。

10.19.10.10. CLI での仮想マシンのスナップショットの削除

適切な **VirtualMachineSnapshot** オブジェクトを削除して、既存の仮想マシン (VM) スナップショットを削除できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

- **VirtualMachineSnapshot** オブジェクトを削除します。スナップショットコントローラーは、**VirtualMachineSnapshot** を、関連付けられた **VirtualMachineSnapshotContent** オブジェクトと共に削除します。

```
$ oc delete vmsnapshot <my-vmsnapshot>
```

検証

- スナップショットが削除され、この仮想マシンに割り当てられていないことを確認します。

```
$ oc get vmsnapshot
```

10.19.10.11. 関連情報

- CSI ボリュームスナップショット

10.19.11. ローカル仮想マシンディスクの別のノードへの移動

ローカルボリュームストレージを使用する仮想マシンは、特定のノードで実行されるように移動することができます。

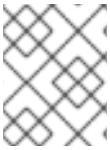
以下の理由により、仮想マシンを特定のノードに移動する場合があります。

- 現在のノードにローカルストレージ設定に関する制限がある。
- 新規ノードがその仮想マシンのワークロードに対して最適化されている。

ローカルストレージを使用する仮想マシンを移行するには、データボリュームを使用して基礎となるボリュームのクローンを作成する必要があります。クローン操作が完了したら、新規データボリュームを使用できるように 仮想マシン設定を編集 するフィルタリング新規データボリュームを別の仮想マシンに追加 することができます。

ヒント

事前割り当てをグローバルに有効にする場合や、単一データボリュームについて、Containerized Data Importer (CDI) はクローン時にディスク領域を事前に割り当てます。事前割り当てにより、書き込みパフォーマンスが向上します。詳細は、データボリュームについての事前割り当ての使用 を参照してください。



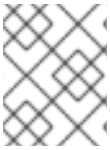
注記

cluster-admin ロールのないユーザーには、複数の namespace 間でボリュームのクローンを作成できるように 追加のユーザーパーミッション が必要になります。

10.19.11.1. ローカルボリュームの別のノードへのクローン作成

基礎となる永続ボリューム要求 (PVC) のクローンを作成して、仮想マシンディスクを特定のノードで実行するように移行することができます。

仮想マシンディスクのノードが適切なノードに作成されることを確認するには、新規の永続ボリューム (PV) を作成するか、該当するノードでそれを特定します。一意のラベルを PV に適用し、これがデータボリュームで参照できるようにします。



注記

宛先 PV のサイズはソース PVC と同じか、それよりも大きくなければなりません。宛先 PV がソース PVC よりも小さい場合、クローン作成操作は失敗します。

前提条件

- 仮想マシンが実行されていないこと。仮想マシンディスクのクローンを作成する前に、仮想マシンの電源を切ります。

手順

1. ノードに新規のローカル PV を作成するか、ノードにすでに存在しているローカル PV を特定します。
   - **nodeAffinity.nodeSelectorTerms** パラメーターを含むローカル PV を作成します。以下のマニフェストは、**node01** に **10Gi** のローカル PV を作成します。

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <destination-pv> 1
  annotations:
spec:
  accessModes:
  - ReadWriteOnce
  capacity:
    storage: 10Gi 2
  local:
    path: /mnt/local-storage/local/disk1 3
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
```

```
        - key: kubernetes.io/hostname
          operator: In
          values:
          - node01 4
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local
  volumeMode: Filesystem
```

1. PV の名前。
2. PV 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、それよりも大きくなければなりません。
3. ノードのマウントパス。
4. PV を作成するノードの名前。

- ターゲットノードに存在する PV を特定します。設定の **nodeAffinity** フィールドを確認して、PV がプロビジョニングされるノードを特定することができます。

```
$ oc get pv <destination-pv> -o yaml
```

以下のスニペットは、PV が **node01** にあることを示しています。

出力例

```
# ...
spec:
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname 1
          operator: In
          values:
          - node01 2
# ...
```

1. **kubernetes.io/hostname** キーでは、ノードを選択するためにノードホスト名を使用します。
2. ノードのホスト名。

2. PV に一意のラベルを追加します。

```
$ oc label pv <destination-pv> node=node01
```

3. 以下を参照するデータボリュームマニフェストを作成します。
   - 仮想マシンの PVC 名と namespace。
   - 直前の手順で PV に適用されたラベル。

- 宛先 PV のサイズ。

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <clone-datavolume> 1
spec:
  source:
    pvc:
      name: "<source-vm-disk>" 2
      namespace: "<source-namespace>" 3
  pvc:
    accessModes:
      - ReadWriteOnce
    selector:
      matchLabels:
        node: node01 4
    resources:
      requests:
        storage: <10Gi> 5
```

1. 新規データボリュームの名前。
2. ソース PVC の名前。PVC 名が分からない場合は、仮想マシン設定 **spec.volumes.persistentVolumeClaim.claimName** で確認できます。
3. ソース PVC が存在する namespace。
4. 直前の手順で PV に追加したラベル。
5. 宛先 PV のサイズ。

4. データボリュームマニフェストをクラスターに適用してクローン作成の操作を開始します。

```
$ oc apply -f <clone-datavolume.yaml>
```

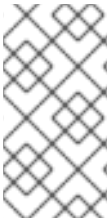
データボリュームは、仮想マシンの PVC のクローンを特定のノード上の PV に作成します。

10.19.12. 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 OpenShift 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。

10.19.12.1. データボリュームについて

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。データボリュームは、基礎となる永続ボリューム要求 (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。データボリュームは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。



注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.19.12.2. データボリュームを使用した空のディスクイメージの作成

データボリューム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永続ボリューム要求 (PVC) 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永続ボリュームがあること。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **DataVolume** マニフェストを編集します。

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  pvc:
    storageClassName: "hostpath" 1
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 500Mi
```

1 オプション: ストレージクラスを指定しない場合、デフォルトのストレージクラスが適用されます。

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
```

10.19.12.3. 関連情報

- 事前割り当てモードを設定 して、データボリューム操作の書き込みパフォーマンスを向上させます。

10.19.13. smart-cloning を使用したデータボリュームのクローン作成

スマートクローニングは、Red Hat OpenShift Data Foundation の組み込み機能です。スマートクローニングは、ホストアシストクローニングよりも高速で効率的です。

smart-cloning を有効にするためにアクションを実行する必要はありませんが、この機能を使用するには、ストレージ環境が smart-cloning と互換性があることを確認する必要があります。

永続ボリューム要求 (PVC) ソースでデータボリュームを作成すると、クローン作成プロセスが自動的に開始します。お使いの環境で smart-cloning をサポートするかどうかにかかわらず、データボリュームのクローンは常に受信できます。ただし、ストレージプロバイダーが smart-cloning に対応している場合、smart-cloning によるパフォーマンス上のメリットが得られます。

10.19.13.1. データボリュームについて

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。データボリュームは、基礎となる永続ボリューム要求 (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。データボリュームは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。

注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.19.13.2. smart-cloning について

データボリュームに smart-cloning が実行された場合、以下が発生します。

1. ソースの永続ボリューム要求 (PVC) のスナップショットが作成されます。
2. PVC はスナップショットから作成されます。
3. スナップショットは削除されます。

10.19.13.3. データボリュームのクローン作成

前提条件

smart-cloning が実行されるには、以下の条件が必要です。

- ストレージプロバイダーはスナップショットをサポートする必要がある。
- ソースおよびターゲット PVC は、同じストレージクラスに定義される必要がある。
- ソースおよびターゲット PVC は同じ **volumeMode** を共有します。
- **VolumeSnapshotClass** オブジェクトは、ソースおよびターゲット PVC の両方に定義されるストレージクラスを参照する必要がある。

手順

データボリュームのクローン作成を開始するには、以下を実行します。

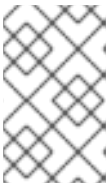
1. 新規データボリュームの名前およびソース PVC の名前と namespace 指定する **DataVolume** オブジェクトの YAML ファイルを作成します。この例では、**ストレージ** API を指定しているため、**accessModes** または **volumeMode** を指定する必要はありません。最適な値は、自動的に計算されます。

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <cloner-datavolume> 1
spec:
  source:
    pvc:
      namespace: "<source-namespace>" 2
      name: "<my-favorite-vm-disk>" 3
  storage: 4
    resources:
      requests:
        storage: <2Gi> 5
```

1. 新規データボリュームの名前。
2. ソース PVC が存在する namespace。
3. ソース PVC の名前。
4. **ストレージ** API を使用して割り当てを指定します。
5. 新規データボリュームのサイズ。

2. データボリュームを作成して PVC のクローン作成を開始します。

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



注記

データボリュームは仮想マシンが 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データボリュームを参照する仮想マシンを作成できます。

10.19.13.4. 関連情報

- 新規データボリュームへの仮想マシンディスクの永続ボリューム要求 (PVC) のクローン作成
- 事前割り当てモードを設定 して、データボリューム操作の書き込みパフォーマンスを向上させます。
- ストレージプロファイルのカスタマイズ

10.19.14. 仮想ディスクのホットプラグ

仮想マシン (VM) または仮想マシンインスタンス (VMI) を停止することなく、仮想ディスクを追加または削除できます。

10.19.14.1. 仮想ディスクのホットプラグについて

仮想ディスクを **ホットプラグ** すると、仮想マシンの実行中に仮想ディスク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。

仮想ディスクを **ホットアンプラグ** すると、仮想マシンの実行中に仮想ディスクの割り当てを仮想マシンインスタンスから解除できます。

データボリュームおよび永続ボリューム要求 (PVC) のみをホットプラグおよびホットアンプラグできます。コンテナーディスクのホットプラグおよびホットアンプラグはできません。

仮想ディスクをホットプラグした後は、仮想マシンを再起動しても、解除するまで接続されたままになります。

Web コンソールでは、ホットプラグされたボリュームは、ディスクリスト (**VirtualMachine Details → Configuration → Disks** タブ) のディスク上で "persistent hotplug" ラベルでマークされます。ホットプラグボリュームを永続ボリュームに変更するには、**Options** メニュー  をクリックし、**Make persistent** を選択します。"persistent hotplug" ラベルが削除され、次の再起動後に変更が適用されます。

10.19.14.2. virtio-scsi について

OpenShift Virtualization では、ホットプラグされたディスクが **SCSI** バスを使用できるように、各仮想マシン (VM) に **virtio-scsi** コントローラーがあります。**virtio-scsi** コントローラーは、パフォーマンス上の利点を維持しながら、**virtio** の制限を克服します。スケーラビリティが高く、400 万台を超えるディスクのホットプラグをサポートします。

通常の **virtio** は、スケーラブルではないため、ホットプラグされたディスクでは使用できません。各 **virtio** ディスクは、VM 内の制限された PCI Express (PCIe) スロットの 1 つを使用します。PCIe スロットは他のデバイスでも使用され、事前に予約する必要があるため、オンデマンドでスロットを利用できない場合があります。

10.19.14.3. CLI を使用した仮想ディスクのホットプラグ

仮想マシンの実行中に仮想マシンインスタンス (VMI) に割り当てる必要のある仮想ディスクをホットプラグします。

前提条件

- 仮想ディスクをホットプラグするために実行中の仮想マシンが必要です。
- ホットプラグ用に 1 つ以上のデータボリュームまたは永続ボリューム要求 (PVC) が利用可能である必要があります。

手順

- 以下のコマンドを実行して、仮想ディスクをホットプラグします。

```
$ virtctl addvolume <virtual-machine|virtual-machine-instance> --volume-name=
<datavolume|PVC> \
[--persist] [--serial=<label-name>]
```

  - オプションの **--persist** フラグを使用して、ホットプラグされたディスクを、永続的にマウントされた仮想ディスクとして仮想マシン仕様に追加します。仮想ディスクを永続的にマウントするために、仮想マシンを停止、再開または再起動します。**--persist** フラグを指定しても、仮想ディスクをホットプラグしたり、ホットアンプラグしたりできなくなります。**--persist** フラグは仮想マシンに適用され、仮想マシンインスタンスには適用されません。

- オプションの **--serial** フラグを使用すると、選択した英数字の文字列ラベルを追加できます。これは、ゲスト仮想マシンでホットプラグされたディスクを特定するのに役立ちます。このオプションを指定しない場合、ラベルはデフォルトでホットプラグされたデータボリュームまたは PVC の名前に設定されます。

10.19.14.4. CLI を使用した仮想ディスクのホットアンプラグ

仮想マシンの実行中に仮想マシンインスタンス (VMI) から割り当てを解除する必要のある仮想ディスクをホットアンプラグします。

前提条件

- 仮想マシンが実行中である必要があります。
- 1 つ以上のデータボリュームまたは永続ボリューム要求 (PVC) が利用可能であり、ホットプラグされている必要があります。

手順

- 以下のコマンドを実行して、仮想ディスクをホットアンプラグします。

```
$ virtctl removevolume <virtual-machine|virtual-machine-instance> --volume-name=
<datavolume|PVC>
```

10.19.14.5. Web コンソールを使用した仮想ディスクのホットプラグ

仮想マシンの実行中に仮想マシンインスタンス (VMI) に割り当てる必要のある仮想ディスクをホットプラグします。仮想ディスクをホットプラグすると、ホットアンプラグするまで VMI に接続されたままになります。

前提条件

- 仮想ディスクをホットプラグするために実行中の仮想マシンが必要です。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想ディスクをホットプラグする実行中の仮想マシンを選択します。
3. **VirtualMachine details** ページで、**Configuration → Disks** をクリックします。
4. **ディスクの追加**をクリックします。
5. **Add disk (hot plugged)** ウィンドウで、ホットプラグする仮想ディスクの情報を入力します。
6. **Save** をクリックします。

10.19.14.6. Web コンソールを使用した仮想ディスクのホットアンプラグ

仮想マシンの実行中に仮想マシンインスタンス (VMI) から割り当てを解除する必要のある仮想ディスクをホットアンプラグします。

前提条件

- 仮想マシンが実行中で、ホットプラグされたディスクが接続されている必要があります。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. ホットアンプラグするディスクを含む実行中の仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration → Disks** をクリックします。
4. ホットアンプラグする仮想ディスクの横にある Options メニュー  をクリックし、**Detach** を選択します。
5. **Detach** をクリックします。

10.19.15. 仮想マシンでのコンテナーディスクの使用

仮想マシンイメージをコンテナーディスクにビルドし、これをコンテナーレジストリーに保存することができます。次に、コンテナーディスクを仮想マシンの永続ストレージにインポートしたり、一時ストレージの仮想マシンに直接割り当てたりすることができます。

重要

大規模なコンテナーディスクを使用する場合、I/O トラフィックが増え、ワーカーノードに影響を与える可能性があります。これにより、ノードが利用できなくなる可能性があります。この問題を解決するには、以下を実行します。

- **DeploymentConfig** オブジェクトのプルーニング
- ガベージコレクションの設定

10.19.15.1. コンテナーディスクについて

コンテナーディスクは、コンテナーイメージレジストリーにコンテナーイメージとして保存される仮想マシンのイメージです。コンテナーディスクを使用して、同じディスクイメージを複数の仮想マシンに配信し、多数の仮想マシンのクローンを作成することができます。

コンテナーディスクは、仮想マシンに割り当てられるデータボリュームを使用して永続ボリューム要求 (PVC) にインポートするか、一時 **containerDisk** ボリュームとして仮想マシンに直接割り当てることができます。

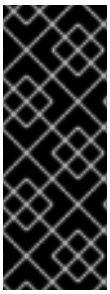
10.19.15.1.1. データボリュームの使用によるコンテナーディスクの PVC へのインポート

Containerized Data Importer (CDI) を使用し、データボリュームを使用してコンテナーディスクを PVC にインポートします。次に、データボリュームを永続ストレージの仮想マシンに割り当てることができます。

10.19.15.1.2. コンテナーディスクの containerDisk ボリュームとしての仮想マシンへの割り当て

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、削除される際に破棄されます。**containerDisk** ボリュームを持つ仮想マシンが起動すると、コンテナーイメージはレジストリーからプルされ、仮想マシンをホストするノードでホストされます。

containerDisk ボリュームは、CD-ROM などの読み取り専用ファイルシステム用に、または破棄可能な仮想マシン用に使用します。



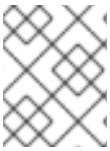
重要

データはホストノードのローカルストレージに一時的に書き込まれるため、読み取り/書き込みファイルシステムに **containerDisk** ボリュームを使用することは推奨されません。これにより、データを移行先ノードに移行する必要があるため、ノードのメンテナンス時など、仮想マシンのライブマイグレーションが遅くなります。さらに、ノードの電源が切れた場合や、予期せずにシャットダウンする場合にすべてのデータが失われます。

10.19.15.2. 仮想マシン用のコンテナーディスクの準備

仮想マシンイメージでコンテナーディスクをビルドし、これを仮想マシンで使用する前にこれをコンテナーレジストリーにプッシュする必要があります。次に、データボリュームを使用してコンテナーディスクを PVC にインポートし、これを仮想マシンに割り当てるか、コンテナーディスクを一時的な **containerDisk** ボリュームとして仮想マシンに直接割り当てることができます。

コンテナーディスク内のディスクイメージのサイズは、コンテナーディスクがホストされるレジストリーの最大レイヤーサイズによって制限されます。



注記

Red Hat Quay の場合、Red Hat Quay の初回デプロイ時に作成される YAML 設定ファイルを編集して、最大レイヤーサイズを変更できます。

前提条件

- **podman** がインストールされていない場合はインストールすること。
- 仮想マシンイメージは QCOW2 または RAW 形式のいずれかであること。

手順

1. Dockerfile を作成して、仮想マシンイメージをコンテナーイメージにビルドします。仮想マシンイメージは、UID が **107** の QEMU で所有され、コンテナー内の **/disk/** ディレクトリーに置かれる必要があります。次に、**/disk/** ディレクトリーのパーミッションは **0440** に設定する必要があります。
   以下の例では、Red Hat Universal Base Image (UBI) を使用して最初の段階でこれらの設定変更を処理し、2 番目の段階で最小の **scratch** イメージを使用して結果を保存します。

```
$ cat > Dockerfile << EOF
FROM registry.access.redhat.com/ubi8/ubi:latest AS builder
ADD --chown=107:107 <vm_image>.qcow2 /disk/ 1
RUN chmod 0440 /disk/*

FROM scratch
COPY --from=builder /disk/* /disk/
EOF
```

1 ここで、<vm_image> は QCOW2 または RAW 形式の仮想マシンイメージです。
リモートの仮想マシンイメージを使用するには、**<vm_image>.qcow2** をリモートイメージの完全な URL に置き換えます。

2. コンテナーをビルドし、これにタグ付けします。

```
$ podman build -t <registry>/<container_disk_name>:latest .
```

3. コンテナーイメージをレジストリーにプッシュします。

```
$ podman push <registry>/<container_disk_name>:latest
```

コンテナーレジストリーに TLS がない場合は、コンテナーディスクを永続ストレージにインポートする前に非セキュアなレジストリーとしてこれを追加する必要があります。

10.19.15.3. 非セキュアなレジストリーとして使用するコンテナーレジストリーの TLS の無効化

HyperConverged カスタムリソースの **insecureRegistries** フィールドを編集して、1 つ以上のコンテナーレジストリーの TLS(トランスポート層セキュリティー) を無効にできます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにログインすること。

手順

- **HyperConverged** カスタムリソースを編集し、非セキュアなレジストリーの一覧を **spec.storageImport.insecureRegistries** フィールドに追加します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  storageImport:
    insecureRegistries: 1
      - "private-registry-example-1:5000"
      - "private-registry-example-2:5000"
```

1 このリストのサンプルを、有効なレジストリーホスト名に置き換えます。

10.19.15.4. 次のステップ

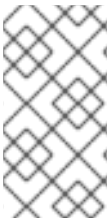
- コンテナーディスクを仮想マシンの永続ストレージにインポート します。
- 一時ストレージの **containerDisk** ボリュームを使用する 仮想マシンを作成 します。

10.19.16. CDI のスクラッチ領域の用意

10.19.16.1. データボリュームについて

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。データボリュームは、基礎となる永続ボリューム要求 (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。データボリュー

ムは、スタンドアロンリソースとして、または仮想マシン (VM) 仕様の **dataVolumeTemplate** フィールドを使用して作成できます。



注記

- スタンドアロンデータボリュームを使用して準備した仮想マシンディスクの PVC は、仮想マシンから独立したライフサイクルを維持します。仮想マシン仕様の **dataVolumeTemplate** フィールドを使用して PVC を準備すると、PVC は仮想マシンと同じライフサイクルを共有します。

10.19.16.2. 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、宛先データボリューム (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。

HyperConverged カスタムリソースの **spec.scratchSpaceStorageClass** フィールドで、スクラッチ領域 PVC をバインドするために使用されるストレージクラスを定義できます。

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

注記

CDI では、元のデータボリュームをサポートする 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ストレージクラス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。

10.19.16.3. スクラッチ領域を必要とする CDI 操作

型	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。

型	理由
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。

10.19.16.4. ストレージクラスの定義

spec.scratchSpaceStorageClass フィールドを **HyperConverged** カスタムリソース (CR) に追加することにより、スクラッチ領域を割り当てる際に、Containerized Data Importer (CDI) が使用するストレージクラスを定義できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. 以下のコマンドを実行して、**HyperConverged** CR を編集します。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

2. **spec.scratchSpaceStorageClass** フィールドを CR に追加し、値をクラスターに存在するストレージクラスの名前に設定します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  scratchSpaceStorageClass: "<storage_class>" 1
```

1 ストレージクラスを指定しない場合、CDI は設定されている永続ボリューム要求のストレージクラスを使用します。

3. デフォルトのエディターを保存して終了し、**HyperConverged** CR を更新します。

10.19.16.5. 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*

✓ 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

10.19.16.6. 関連情報

- 動的プロビジョニング

10.19.17. 永続ボリュームの再利用

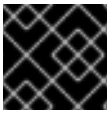
静的にプロビジョニングされた永続ボリューム (PV) を再利用するには、まずボリュームを回収する必要があります。これには、ストレージ設定を再利用できるように PV を削除する必要があります。

10.19.17.1. 静的にプロビジョニングされた永続ボリュームの回収について

永続ボリューム (PV) を回収する場合、PV のバインドを永続ボリューム要求 (PVC) から解除し、PV を削除します。基礎となるストレージによっては、共有ストレージを手動で削除する必要がある場合があります。

次に、PV 設定を再利用し、異なる名前の PV を作成できます。

静的にプロビジョニングされる PV には、回収に **Retain** の回収 (reclaim) ポリシーが必要です。これがない場合、PV は PVC が PV からのバインド解除時に failed 状態になります。



重要

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

10.19.17.2. 静的にプロビジョニングされた永続ボリュームの回収

永続ボリューム要求 (PVC) のバインドを解除し、PV を削除して静的にプロビジョニングされた永続ボリューム (PV) を回収します。共有ストレージを手動で削除する必要がある場合もあります。

静的にプロビジョニングされる PV の回収方法は、基礎となるストレージによって異なります。この手順では、ストレージに応じてカスタマイズする必要がある可能性のある一般的な方法を示します。

手順

手順

1. PV の回収ポリシーが **Retain** に設定されていることを確認します。
   a. PV の回収ポリシーを確認します。

```
$ oc get pv <pv_name> -o yaml | grep 'persistentVolumeReclaimPolicy'
```

   b. **persistentVolumeReclaimPolicy** が **Retain** に設定されていない場合、以下のコマンドで回収ポリシーを編集します。

```
$ oc patch pv <pv_name> -p '{"spec":{"persistentVolumeReclaimPolicy":"Retain"}}'
```

2. PV を使用しているリソースがないことを確認します。

```
$ oc describe pvc <pvc_name> | grep 'Mounted By:'
```

   PVC を使用するリソースをすべて削除してから継続します。

3. PVC を削除して PV を解放します。

```
$ oc delete pvc <pvc_name>
```

4. オプション: PV 設定を YAML ファイルにエクスポートします。この手順の後半で共有ストレージを手動で削除する場合は、この設定を参照してください。このファイルで **spec** パラメーターをベースとして使用し、PV の回収後に同じストレージ設定で新規 PV を作成することもできます。

```
$ oc get pv <pv_name> -o yaml > <file_name>.yaml
```

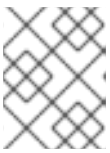
5. PV を削除します。

```
$ oc delete pv <pv_name>
```

6. オプション: ストレージタイプによっては、共有ストレージフォルダーの内容を削除する必要がある場合があります。

```
$ rm -rf <path_to_share_storage>
```

7. オプション: 削除された PV と同じストレージ設定を使用する PV を作成します。回収された PV 設定をすでにエクスポートしている場合、そのファイルの **spec** パラメーターを新規 PV マニフェストのベースとして使用することができます。



注記

競合の可能性を回避するには、新規 PV オブジェクトに削除されたものとは異なる名前を割り当てることが推奨されます。

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

関連情報

- 仮想マシンのローカルストレージの設定

- OpenShift Container Platform Storage ドキュメントには、永続ストレージ についての詳細情報が記載されています。

10.19.18. 仮想マシンディスクの拡張

仮想マシン (VM) のディスクのサイズを拡大して、ストレージ容量を増やすには、ディスクの永続ボリューム要求 (PVC) のサイズを変更します。

ただし、仮想マシンディスクのサイズを縮小することはできません。

10.19.18.1. 仮想マシンディスクの拡張

仮想マシン (VM) のディスクを拡大すると、仮想マシンで使用できる追加の領域が作成されます。ただし、仮想マシンの所有者は、ストレージの使用方法を決定する必要があります。

ディスクが **Filesystem** PVC の場合、一致するファイルは、ファイルシステムのオーバーヘッド用に一部の領域を確保しながら残りのサイズに拡張します。

手順

1. 拡張する必要のある仮想マシンディスクの **PersistentVolumeClaim** マニフェストを編集します。

```
$ oc edit pvc <pvc_name>
```

2. ディスクサイズを更新します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: vm-disk-expand
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 3Gi 1
# ...
```

1 新しいディスクサイズを指定します。

10.19.18.2. 関連情報

- Windows での基本ボリュームの拡張
- Red Hat Enterprise Linux でのデータ破損を伴わない既存ファイルシステムパーティションの拡張
- Red Hat Enterprise Linux での、論理ボリュームとそのファイルシステムのオンライン拡張

第11章 仮想マシンテンプレート

11.1. 仮想マシンテンプレートの作成

11.1.1. 仮想マシンテンプレートについて

事前設定された Red Hat 仮想マシンテンプレートは、**Virtualization → Templates** ページにリスト表示されます。このテンプレートは、Red Hat Enterprise Linux、Fedora、Microsoft Windows 10、および Microsoft Windows Server の異なるバージョンで利用できます。各 Red Hat 仮想マシンテンプレートは、オペレーティングシステムイメージ、オペレーティングシステム、フレーバー (CPU およびメモリー)、およびワークロードタイプ (サーバー) のデフォルト設定で事前に設定されます。

Templates ページには、4 種類の仮想マシンテンプレートが表示されます。

- **Red Hat でサポートされる** テンプレートは、Red Hat によって完全にサポートされています。
- **ユーザーがサポート** するテンプレートは、ユーザーがクローンして作成した **Red Hat でサポートされる** テンプレートです。
- **Red Hat が提供する** テンプレートには、Red Hat の制限されたサポートがあります。
- **User Provided** テンプレートは、ユーザーがクローンして作成した **Red Hat Provided** テンプレートです。

テンプレート**カタログ**のフィルターを使用して、ブート ソースの可用性、オペレーティングシステム、ワークロードなどの属性でテンプレートを並べ替えることができます。

Red Hat が **サポート**するテンプレートまたは **Red Hat が提供**するテンプレートを編集または削除することはできません。テンプレートのクローンを作成し、カスタム仮想マシンテンプレートとして保存してから編集できます。

YAML ファイルの例を編集して、カスタム仮想マシンテンプレートを作成することもできます。

11.1.2. 仮想マシンおよびブートソースについて

仮想マシンは、仮想マシン定義と、データボリュームでサポートされる 1 つ以上のディスクで設定されます。仮想マシンテンプレートを使用すると、事前定義された仮想マシン仕様を使用して仮想マシンを作成できます。

すべての仮想マシンテンプレートには、設定されたドライバーを含む完全に設定された仮想マシンディスクイメージであるブートソースが必要です。それぞれの仮想マシンテンプレートには、ブートソースへのポインターを含む仮想マシン定義が含まれます。各ブートソースには、事前に定義された名前および namespace があります。オペレーティングシステムによっては、ブートソースは自動的に提供されます。これが提供されない場合、管理者はカスタムブートソースを準備する必要があります。

提供されたブート ソースは、オペレーティングシステムの最新バージョンに自動的に更新されます。自動更新されたブートソースの場合、クラスターのデフォルトのストレージクラスを使用して、永続ボリューム要求 (PVC) が作成されます。設定後に別のデフォルトのストレージクラスを選択した場合は、以前のデフォルトのストレージクラスで設定されたクラスター namespace 内の既存のデータボリュームを削除する必要があります。

ブートソース機能を使用するには、OpenShift Virtualization の最新リリースをインストールします。namespace の **openshift-virtualization-os-images** はこの機能を有効にし、OpenShift Virtualization Operator でインストールされます。ブートソース機能をインストールしたら、ブートソースを作成してそれらをテンプレートに割り当て、テンプレートから仮想マシンを作成できます。

ローカルファイルのアップロード、既存 PVC のクローン作成、レジストリーからのインポート、または URL を使用して設定される永続ボリューム要求 (PVC) を使用してブートソースを定義します。Web コンソールを使用して、ブートソースを仮想マシンテンプレートに割り当てます。ブートソースが仮想マシンテンプレートに割り当てられた後に、テンプレートを使用して任意の数の完全に設定済みの準備状態の仮想マシンを作成できます。

11.1.3. Web コンソールでの仮想マシンテンプレートの作成

OpenShift Container Platform Web コンソールで YAML ファイルの例を編集して、仮想マシンテンプレートを作成します。

手順

1. Web コンソールのサイドメニューで、**Virtualization → Templates** をクリックします。
2. オプション: **Project** ドロップダウンメニューを使用して、新しいテンプレートに関連付けられたプロジェクトを変更します。すべてのテンプレートは、デフォルトで **openshift** プロジェクトに保存されます。
3. **Create Template** をクリックします。
4. YAML ファイルを編集して、テンプレートパラメーターを指定します。
5. **Create** をクリックします。
   テンプレートが **Templates** ページに表示されます。
6. オプション: **Download** をクリックして、YAML ファイルをダウンロードして保存します。

11.1.4. 仮想マシンテンプレートのブートソースの追加

ブートソースは、仮想マシンの作成に使用するすべての仮想マシンテンプレートまたはカスタムテンプレートに設定できます。仮想マシンテンプレートがブートソースを使用して設定されている場合、それらには **Templates** ページで **Source available** というラベルが付けられます。ブートソースをテンプレートに追加した後に、テンプレートから新規仮想マシンを作成できます。

Web コンソールでブートソースを選択および追加する方法は 4 つあります。

- **ローカルファイルのアップロード (PVC の作成)**
- **URL (PVC を作成)**
- **クローン (PVC を作成)**
- **レジストリー (PVC を作成)**

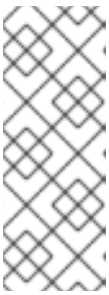
前提条件

- ブートソースを追加するには、**os-images.kubevirt.io:edit** RBAC ロールを持つユーザー、または管理者としてログインしていること。ブートソースが追加されたテンプレートから仮想マシンを作成するには、特別な権限は必要ありません。
- ローカルファイルをアップロードするには、オペレーティングシステムのイメージファイルがローカルマシンに存在している必要がある。

- URL を使用してインポートするには、オペレーティングシステムイメージのある Web サーバーへのアクセスが必要である。例: イメージが含まれる Red Hat Enterprise Linux Web ページ。
- 既存の PVC のクローンを作成するには、PVC を含むプロジェクトへのアクセスが必要である。
- レジストリーを使用してインポートするには、コンテナーレジストリーへのアクセスが必要である。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → Templates** をクリックします。
2. テンプレートの横にある Options メニューをクリックし、**Edit boot source** を選択します。
3. **ディスクの追加**をクリックします。
4. **Add disk** ウィンドウで、**Use this disk as a boot source**を選択します。
5. ディスク名を入力し、**Source** を選択します。たとえば、**Blank (creates PVC)** を選択します。
6. **Persistent Volume Claim size**の値を入力し、圧縮解除されたイメージおよび必要な追加の領域に十分な PVC サイズを指定します。
7. **Type** を選択します (**Disk** など)。
8. オプション: **Storage class** をクリックし、ディスクを作成するために使用されるストレージクラスを選択します。通常、このストレージクラスはすべての PVC で使用するために作成されるデフォルトのストレージクラスです。



注記

提供されたブート ソースは、オペレーティングシステムの最新バージョンに自動的に更新されます。自動更新されたブートソースの場合、クラスターのデフォルトのストレージクラスを使用して、永続ボリューム要求 (PVC) が作成されます。設定後に別のデフォルトのストレージクラスを選択した場合は、以前のデフォルトのストレージクラスで設定されたクラスター namespace 内の既存のデータボリュームを削除する必要があります。

9. オプション: **Apply optimized StorageProfile settings** をオフにして、アクセスモードまたはボリュームモードを編集します。
10. ブートソースを保存する適切な方法を選択します。
    a. ローカルファイルをアップロードしている場合に、**Save and upload**をクリックします。
    b. URL またはレジストリーからコンテンツをインポートしている場合は、**Save and import** をクリックします。
    c. 既存の PVC のクローンを作成している場合は、**Save and clone** をクリックします。

ブートソースを含むカスタム仮想マシンテンプレートが **Catalog** ページにリスト表示されています。このテンプレートを使用して仮想マシンを作成できます。

11.1.4.1. ブートソースを追加するための仮想マシンテンプレートフィールド

以下の表は、**Add boot source to template** ウインドウのフィールドについて説明しています。このウィンドウは、**Virtualization → Templates** ページで仮想マシンテンプレートの **Add source** をクリックしたときに表示されます。

Name	パラメーター	Description
ブートソースタイプ	ローカルファイルのアップロード (PVC の作成)	ローカルデバイスからファイルをアップロードします。サポートされるファイルタイプには、gz、xz、tar、および qcow2 が含まれます。
	URL (PVC を作成)	HTTP または HTTPS エンドポイントで利用できるイメージからコンテンツをインポートします。イメージのダウンロードが可能な Web ページからダウンロードリンクの URL を取得し、その URL リンクを **Import URL** フィールドに入力します。例: Red Hat Enterprise Linux イメージについては、Red Hat カスタマーポータルにログインしてイメージのダウンロードページにアクセスし、KVM ゲストイメージのダウンロードリンク URL をコピーします。
	PVC (PVC を作成)	クラスターですでに利用可能な PVC を使用し、このクローンを作成します。
	レジストリー (PVC を作成)	クラスターからアクセスでき、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を指定します。例: kubevirt/cirros-registry-dis-demo
ソースプロバイダー		オプションフィールド。テンプレートのソース、またはテンプレートを作成したユーザーの名前についての説明テキストを追加します。例: Red Hat
ストレージの詳細設定	StorageClass	ディスクの作成に使用されるストレージクラス。

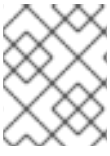
Name	パラメーター	Description
	アクセスモード	永続ボリュームのアクセスモード。サポートされるアクセスモードは、**Single User(RWO)**、**Shared Access(RWX)**、**Read Only (ROX)** です。**Single User (RWO)** が選択される場合、ディスクは単一ノードによって読み取り/書き込み (read/write) としてマウントできます。**Shared Access (RWX)** が選択される場合、ディスクは多数のノードによって読み取り/書き込み (read-write) としてマウントできます。**kubevirt-storage-class-defaults** 設定マップはデータボリュームのアクセスモードを提供します。デフォルト値は、クラスターの各ストレージクラスについての最適なオプションに従って設定されます。  **注記** Shared Access (RWX) は、ノード間の仮想マシンのライブマイグレーションなどの、一部の機能で必要になります。
	ボリュームモード	永続ボリュームがフォーマットされたファイルシステムまたは raw ブロック状態を使用するかどうかを定義します。サポートされるモードは **Block** および **Filesystem** です。**kubevirt-storage-class-defaults** 設定マップはデータボリュームのボリュームモードのデフォルト設定を提供します。デフォルト値は、クラスターの各ストレージクラスについての最適なオプションに従って設定されます。

11.1.5. 関連情報

- ブートソースの作成および使用
- ストレージプロファイルのカスタマイズ

11.2. 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。



注記

Red Hat Virtualization Operator が提供するテンプレートは編集できません。テンプレートのクローンを作成すると、編集できます。

11.2.1. Web コンソールでの仮想マシンテンプレートの編集

OpenShift Container Platform Web コンソールまたはコマンドラインインターフェイスを使用して、仮想マシンテンプレートを編集できます。

仮想マシンテンプレートの編集は、そのテンプレートですでに作成された仮想マシンには影響を与えません。

手順

1. Web コンソールで **Virtualization → Templates** に移動します。
2. 仮想マシンテンプレートの横にある オプションメニューをクリックし、編集するオブジェクトを選択します。
3. Red Hat テンプレートを編集するには、 オプションメニューで、**Clone** を選択してカスタムテンプレートを作成し、カスタムテンプレートを編集します。

注記

テンプレートのデータソースが **DataImportCron** カスタムリソースによって管理されている場合、またはテンプレートにデータボリューム参照がない場合、**Edit boot source reference** は無効になります。

4. **Save** をクリックします。

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

以下の手順を使用して、ネットワークインターフェイスを仮想マシンテンプレートに追加します。

手順

1. サイドメニューから **Virtualization → Templates** をクリックします。
2. 仮想マシン テンプレートを選択して、**Template details** ページを開きます。
3. **Network interfaces** タブで、**Add Network Interface** をクリックします。
4. **Add Network Interface** ウィンドウで、ネットワークインターフェイスの **Name**、**Model**、**Network**、 **Type**、および **MAC Address** を指定します。
5. **Add** をクリックします。

11.2.1.2. 仮想マシンテンプレートへの仮想ディスクの追加

以下の手順を使用して、仮想ディスクを仮想マシンテンプレートに追加します。

手順

1. サイドメニューから **Virtualization → Templates** をクリックします。

2. 仮想マシン テンプレートを選択して、**Template details** ページを開きます。
3. **Disks** タブで、**Add disk** をクリックします。
4. **Source**、**Name**、**Size**、**Type**、**Interface**、および **Storage Class** を指定します。
    a. オプション: 空のディスクソースを使用し、データボリュームの作成時に最大の書き込みパフォーマンスが必要な場合に、事前割り当てを有効にできます。そのためには、**Enable preallocation** チェックボックスをオンにします。
    b. オプション: **Apply optimized StorageProfile settings** をクリアして、仮想ディスクの **Volume Mode** と **Access Mode** を変更できます。これらのパラメーターを指定しない場合、システムは **kubevirt-storage-class-defaults** config map のデフォルト値を使用します。
5. **Add** をクリックします。

11.3. 仮想マシンテンプレートの専用リソースの有効化

パフォーマンスを向上させるために、仮想マシンには CPU などのノードの専用リソースを持たせることができます。

11.3.1. 専用リソースについて

仮想マシンの専用リソースを有効にする場合、仮想マシンのワークロードは他のプロセスで使用されない CPU でスケジュールされます。専用リソースを使用することで、仮想マシンのパフォーマンスとレイテンシーの予測の精度を向上させることができます。

11.3.2. 前提条件

- CPU マネージャー がノードに設定されている。仮想マシンのワークロードをスケジュールする前に、ノードに **cpumanager** = **true** ラベルが設定されていることを確認します。

11.3.3. 仮想マシンテンプレートの専用リソースの有効化

Details タブで仮想マシンテンプレートの専用リソースを有効にします。Red Hat テンプレートから作成された仮想マシンは、専用のリソースで設定できます。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → Templates** をクリックします。
2. 仮想マシン テンプレートを選択して、**Template details** ページを開きます。
3. **Scheduling** タブで、**Dedicated Resources** の横にある編集アイコンをクリックします。
4. **Schedule this workload with dedicated resources (guaranteed policy)**を選択します。
5. **Save** をクリックします。

11.4. 仮想マシンテンプレートのカスタム NAMESPACE へのデプロイ

Red Hat は、**openshift** namespace にインストールされる、事前に設定された仮想マシンテンプレート

を提供します。**ssp-operator** は、デフォルトで仮想マシンテンプレートを **openshift** namespace にデプロイします。**openshift** namespace のテンプレートは、すべてのユーザーに広く公開されます。これらのテンプレートは、さまざまなオペレーティングシステムの **Virtualization → Templates** ページにリスト表示されています。

11.4.1. テンプレート用のカスタム namespace の作成

仮想マシンテンプレートをデプロイするために使用されるカスタム namespace を作成できます。このテンプレートは、アクセス権のある任意のユーザーが使用できます。テンプレートをカスタム namespace に追加するには、**HyperConverged** カスタムリソース (CR) を編集し、**commonTemplatesNamespace** を spec に追加し、仮想マシンテンプレートのカスタム namespace を指定します。**HyperConverged** CR の変更後に、**ssp-operator** はカスタム namespace のテンプレートに反映します。

前提条件

- OpenShift Container Platform CLI (**oc**) をインストールしている。
- cluster-admin 権限を持つユーザーとしてログインしている。

手順

- 以下のコマンドを使用してカスタム namespace を作成します。

```
$ oc create namespace <mycustomnamespace>
```

11.4.2. カスタム namespace へのテンプレートの追加

ssp-operator は、デフォルトで仮想マシンテンプレートを **openshift** namespace にデプロイします。**openshift** namespace のテンプレートは、すべてのユーザーに広く公開されます。カスタム namespace が作成され、テンプレートがその namespace に追加されると、**openshift** namespace の仮想マシンテンプレートを変更または削除することができます。テンプレートをカスタム namespace に追加するには、**ssp-operator** が含まれる **HyperConverged** カスタムリソース (CR) を編集します。

手順

1. **openshift** namespace で利用可能な仮想マシンテンプレートの一覧を表示します。

```
$ oc get templates -n openshift
```

2. 以下のコマンドを実行して、デフォルトエディターで **HyperConverged** CR を編集します。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

3. カスタム namespace で利用可能な仮想マシンテンプレートのリストを表示します。

```
$ oc get templates -n customnamespace
```

4. **commonTemplatesNamespace** 属性を追加し、カスタム namespace を指定します。以下に例を示します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
```

```
metadata:
  name: kubevirt-hyperconverged
spec:
  commonTemplatesNamespace: customnamespace (1)
```

(1) テンプレートをデプロイするためのカスタム namespace。

5. 変更を保存し、エディターを終了します。**ssp-operator** は、デフォルトの **openshift** namespace にある仮想マシンテンプレートをカスタム namespace に追加します。

11.4.2.1. カスタム namespace からのテンプレートの削除

カスタム namespace から仮想マシンテンプレートを削除するには、**HyperConverged** カスタムリソース (CR) から **commonTemplateNamespace** 属性を削除し、そのカスタム namespace から各テンプレートを削除します。

手順

1. 以下のコマンドを実行して、デフォルトエディターで **HyperConverged** CR を編集します。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

2. **commonTemplateNamespace** 属性を削除します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  commonTemplatesNamespace: customnamespace (1)
```

(1) 削除する **commonTemplatesNamespace** 属性。

3. 削除されたカスタム namespace から特定のテンプレートを削除します。

```
$ oc delete templates -n customnamespace <template_name>
```

検証

- テンプレートがカスタム namespace から削除されていることを確認します。

```
$ oc get templates -n customnamespace
```

11.4.2.2. 関連情報

- 仮想マシンテンプレートの作成

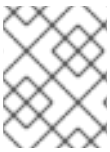
11.5. 仮想マシンテンプレートの削除

Web コンソールを使用して、Red Hat テンプレートに基づいてカスタマイズされた仮想マシンテンプレートを削除できます。

Red Hat テンプレートは削除できません。

11.5.1. 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。



注記

カスタマイズされた仮想マシンテンプレートを削除できます。Red Hat 提供のテンプレートは削除できません。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → Templates** をクリックします。
2. テンプレートの Options メニュー  をクリックし、**Delete template** を選択します。
3. **Delete** をクリックします。

11.6. ブートソースの作成および使用

ブートソースには、ブート可能なオペレーティングシステム (OS) とドライバーなどの OS のすべての設定が含まれます。

ブートソースを使用して、特定の設定で仮想マシンテンプレートを作成します。これらのテンプレートを使用して、利用可能な仮想マシンを多数作成することができます。

カスタムブートソースの作成、ブートソースのアップロード、およびその他のタスクを支援するために、OpenShift Container Platform Web コンソールでクイックスタートツアーを利用できます。**Help** メニューから **Quick Starts** を選択して、クイックスタートツアーを表示します。

11.6.1. 仮想マシンおよびブートソースについて

仮想マシンは、仮想マシン定義と、データボリュームでサポートされる 1 つ以上のディスクで設定されます。仮想マシンテンプレートを使用すると、事前定義された仮想マシン仕様を使用して仮想マシンを作成できます。

すべての仮想マシンテンプレートには、設定されたドライバーを含む完全に設定された仮想マシンディスクイメージであるブートソースが必要です。それぞれの仮想マシンテンプレートには、ブートソースへのポインターを含む仮想マシン定義が含まれます。各ブートソースには、事前に定義された名前および namespace があります。オペレーティングシステムによっては、ブートソースは自動的に提供されます。これが提供されない場合、管理者はカスタムブートソースを準備する必要があります。

提供されたブート ソースは、オペレーティングシステムの最新バージョンに自動的に更新されます。自動更新されたブートソースの場合、クラスターのデフォルトのストレージクラスを使用して、永続ボリューム要求 (PVC) が作成されます。設定後に別のデフォルトのストレージクラスを選択した場合は、以前のデフォルトのストレージクラスで設定されたクラスター namespace 内の既存のデータボリュームを削除する必要があります。

ブートソース機能を使用するには、OpenShift Virtualization の最新リリースをインストールします。namespace の **openshift-virtualization-os-images** はこの機能を有効にし、OpenShift Virtualization Operator でインストールされます。ブートソース機能をインストールしたら、ブートソースを作成し

てそれらをテンプレートに割り当て、テンプレートから仮想マシンを作成できます。

ローカルファイルのアップロード、既存 PVC のクローン作成、レジストリーからのインポート、または URL を使用して設定される永続ボリューム要求 (PVC) を使用してブートソースを定義します。Web コンソールを使用して、ブートソースを仮想マシンテンプレートに割り当てます。ブートソースが仮想マシンテンプレートに割り当てられた後に、テンプレートを使用して任意の数の完全に設定済みの準備状態の仮想マシンを作成できます。

11.6.2. RHEL イメージをブート ソースとしてインポートする

イメージの URL を指定して、Red Hat Enterprise Linux (RHEL) イメージをブートソースとしてインポートできます。

前提条件

- オペレーティングシステムイメージを含む Web ページにアクセスできる必要があります。例: イメージを含む Red Hat Enterprise Linux Web ページをダウンロードします。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → Templates** をクリックします。
2. ブートソースを設定する必要のある仮想マシンテンプレートを特定し、**Add source** をクリックします。
3. **Add boot source to template** ウィンドウで、**Boot source type** リストから **URL (creates PVC)** を選択します。
4. **RHEL download page** をクリックして Red Hat カスタマーポータルにアクセスします。利用可能なインストーラーおよびイメージのリストは、Red Hat Enterprise Linux のダウンロードページに表示されます。
5. ダウンロードする Red Hat Enterprise Linux KVM ゲストイメージを特定します。**Download Now** を右クリックして、イメージの URL をコピーします。
6. **Add boot source to template** ウィンドウで、URL を **Import URL** フィールドに貼り付け、**Save and import** をクリックします。

検証

1. テンプレートの **テンプレート** ページの **ブートソース** 列に緑色のチェックマークが表示されていることを確認します。

このテンプレートを使用して RHEL 仮想マシンを作成できます。

11.6.3. 仮想マシンテンプレートのブートソースの追加

ブートソースは、仮想マシンの作成に使用するすべての仮想マシンテンプレートまたはカスタムテンプレートに設定できます。仮想マシンテンプレートがブートソースを使用して設定されている場合、それらには **Templates** ページで **Source available** というラベルが付けられます。ブートソースをテンプレートに追加した後に、テンプレートから新規仮想マシンを作成できます。

Web コンソールでブートソースを選択および追加する方法は 4 つあります。

- **ローカルファイルのアップロード (PVC の作成)**
- **URL (PVC を作成)**
- **クローン (PVC を作成)**
- **レジストリー (PVC を作成)**

前提条件

- ブートソースを追加するには、**os-images.kubevirt.io:edit** RBAC ロールを持つユーザー、または管理者としてログインしていること。ブートソースが追加されたテンプレートから仮想マシンを作成するには、特別な権限は必要ありません。
- ローカルファイルをアップロードするには、オペレーティングシステムのイメージファイルがローカルマシンに存在している必要がある。
- URL を使用してインポートするには、オペレーティングシステムイメージのある Web サーバーへのアクセスが必要である。例: イメージが含まれる Red Hat Enterprise Linux Web ページ。
- 既存の PVC のクローンを作成するには、PVC を含むプロジェクトへのアクセスが必要である。
- レジストリーを使用してインポートするには、コンテナーレジストリーへのアクセスが必要である。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → Templates** をクリックします。
2. テンプレートの横にある Options メニューをクリックし、**Edit boot source** を選択します。
3. **ディスクの追加**をクリックします。
4. **Add disk** ウィンドウで、**Use this disk as a boot source**を選択します。
5. ディスク名を入力し、**Source** を選択します。たとえば、**Blank (creates PVC)** を選択します。
6. **Persistent Volume Claim size**の値を入力し、圧縮解除されたイメージおよび必要な追加の領域に十分な PVC サイズを指定します。
7. **Type** を選択します (**Disk** など)。
8. オプション: **Storage class** をクリックし、ディスクを作成するために使用されるストレージクラスを選択します。通常、このストレージクラスはすべての PVC で使用するために作成されるデフォルトのストレージクラスです。

注記

提供されたブート ソースは、オペレーティングシステムの最新バージョンに自動的に更新されます。自動更新されたブートソースの場合、クラスターのデフォルトのストレージクラスを使用して、永続ボリューム要求 (PVC) が作成されます。設定後に別のデフォルトのストレージクラスを選択した場合は、以前のデフォルトのストレージクラスで設定されたクラスター namespace 内の既存のデータボリュームを削除する必要があります。

9. オプション: **Apply optimized StorageProfile settings** をオフにして、アクセスモードまたはボリュームモードを編集します。
10. ブートソースを保存する適切な方法を選択します。
    a. ローカルファイルをアップロードしている場合に、**Save and upload** をクリックします。
    b. URL またはレジストリーからコンテンツをインポートしている場合は、**Save and import** をクリックします。
    c. 既存の PVC のクローンを作成している場合は、**Save and clone** をクリックします。

ブートソースを含むカスタム仮想マシンテンプレートが **Catalog** ページにリスト表示されています。このテンプレートを使用して仮想マシンを作成できます。

11.6.4. ブートソースが割り当てられたテンプレートからの仮想マシンの作成

ブートソースをテンプレートに追加した後に、テンプレートから仮想マシンを作成できます。

手順

1. Open Shift Container Platform Web コンソールのサイドメニューで、**Virtualization → Catalog** をクリックします。
2. 更新されたテンプレートを選択し、**Quick create VirtualMachine** をクリックします。

VirtualMachine details が、**Starting** のステータスで表示されます。

11.6.5. 関連情報

- 仮想マシンテンプレートの作成
- ブートソースの自動更新の管理

11.7. ブートソースの自動更新の管理

ブートソースの自動更新を管理します。

11.7.1. ブートソースの自動更新について

ブートソースにより、ユーザーは仮想マシン (VM) をよりアクセスしやすく効率的に作成できるようになります。ブートソースの自動更新が有効になっている場合、コンテナー化データインポーター (CDI) はイメージをインポート、ポーリング、更新して、新しい仮想マシン用にクローンを作成できるようにします。デフォルトでは、CDI は OpenShift Virtualization が提供する **システム定義** のブートソースを自動的に更新します。

EnableCommonBootImageImport 機能ゲートを無効にすることで、システム定義のすべてのブートソースの自動更新をオプトアウトできます。この機能ゲートを無効にすると、すべての **DataImportCron** オブジェクトが削除されます。これにより、オペレーティングシステムイメージを保存する以前にインポートされた **PersistentVolumeClaim** (PVC) オブジェクトは削除されませんが、管理者はこれらのオブジェクトを手動で削除できます。

EnableCommonBootImageImport 機能ゲートが無効になると、**DataSource** オブジェクトがリセットされ、元の PVC を指さなくなります。管理者は、**DataSource** オブジェクト用に新しい PVC を作成し、その PVC にオペレーティングシステムイメージを設定することにより、ブートソースを手動で提供できます。

OpenShift Virtualization によって提供されていない **カスタム** ブートソースは、機能ゲートによって制御されません。**HyperConverged** カスタムリソース (CR) を編集して、それらを個別に管理する必要があります。この方法を使用して、システム定義の個々のブートソースを管理することもできます。

11.7.2. すべてのシステムブートソースの自動更新を有効または無効にする

機能ゲートを使用して、システム定義のすべてのブートソースの自動更新を制御します。

11.7.2.1. すべてのシステム定義のブートソースの自動更新の管理

ブートソースの自動インポートと更新を無効にすると、リソースの使用量が削減される可能性があります。切断された環境では、ブートソースの自動更新を無効にすると、**CDIDataImportCronOutdated** アラートがログをいっぱいにするのを防ぎます。

すべてのシステム定義のブートソースの自動更新を無効にするには、値を **false** に設定して、**enableCommonBootImageImport** 機能ゲートをオフにします。この値を **true** に設定すると、機能ゲートが再度有効になり、自動更新が再びオンになります。

注記

カスタムブートソースは、この設定の影響を受けません。

手順

- **HyperConverged** カスタムリソース (CR) を編集して、ブートソースの自動更新の機能ゲートを切り替えます。
  - ブートソースの自動更新を無効にするには、**HyperConverged** CR の **spec.featureGates.enableCommonBootImageImport** フィールドを **false** に設定します。以下に例を示します。

```
$ oc patch hco kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/enableCommonBootImageImport", \
  "value": false}]'
```

  - ブートソースの自動更新を再び有効にするには、**HyperConverged** CR の **spec.featureGates.enableCommonBootImageImport** フィールドを **true** に設定します。以下に例を示します。

```
$ oc patch hco kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/enableCommonBootImageImport", \
  "value": true}]'
```

11.7.3. カスタムブートソースの自動更新を有効にする

クラスターにデフォルトのストレージクラスがあることを確認してください。次に、カスタムブートソースの自動更新を有効にします。

11.7.3.1. カスタムブートソース更新用のストレージクラスの設定

HyperConverged カスタムリソース (CR) で新しいデフォルトのストレージクラスを指定します。

重要

ブートソースは、デフォルトのストレージクラスを使用してストレージから作成されます。クラスターにデフォルトのストレージクラスがない場合は、カスタムブートソースの自動更新を設定する前に、デフォルトのストレージクラスを定義する必要があります。

手順

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

2. **storageClassName** フィールドに値を入力して、新しいストレージクラスを定義します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: rhel8-image-cron
    spec:
      template:
        spec:
          storageClassName: <new_storage_class> 1
#...
```

1 ストレージクラスを定義します。

3. 現在のデフォルトのストレージクラスから **storageclass.kubernetes.io/is-default-class** アノテーションを削除します。

   a. 次のコマンドを実行して、現在のデフォルトのストレージクラスの名前を取得します。

```
$ oc get sc
```

出力例

```
NAME PROVISIONER RECLAIMPOLICY VOLUMEBINDINGMODE
ALLOWVOLUMEEXPANSION AGE
csi-manila-ceph manila.csi.openstack.org Delete Immediate false 11d
```

```
hostpath-csi-basic (default) kubevirt.io.hostpath-provisioner Delete
WaitForFirstConsumer false 11d 1
...
```

1 この例では、現在のデフォルトのストレージクラスの名前は **hostpath-csi-basic** です。

b. 次のコマンドを実行して、現在のデフォルトのストレージクラスからアノテーションを削除します。

```
$ oc patch storageclass <current_default_storage_class> -p '{"metadata": {"annotations":
{"storageclass.kubernetes.io/is-default-class":"false"}}}' 1
```

1 **<current_default_storage_class>** をデフォルトのストレージクラスの **storageClassName** 値に置き換えます。

4. 次のコマンドを実行して、新しいストレージクラスをデフォルトとして設定します。

```
$ oc patch storageclass <new_storage_class> -p '{"metadata":{"annotations":
{"storageclass.kubernetes.io/is-default-class":"true"}}}' 1
```

1 **<new_storage_class>** を **HyperConverged** CR に追加した **storageClassName** 値に置き換えます。

11.7.3.2. カスタムブートソースの自動更新を有効にする

OpenShift Virtualization は、デフォルトでシステム定義のブートソースを自動的に更新しますが、カスタムブートソースは自動的に更新しません。**HyperConverged** カスタムリソース (CR) を編集して、自動更新を手動で有効にする必要があります。

前提条件

- クラスターにはデフォルトのストレージクラスがあります。

手順

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. 適切なテンプレートおよびブートソースを **dataImportCronTemplates** セクションで追加して、**HyperConverged** CR を編集します。以下に例を示します。

カスタムリソースの例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
```

```
- metadata:
    name: centos7-image-cron
    annotations:
      cdi.kubevirt.io/storage.bind.immediate.requested: "true" (1)
  spec:
    schedule: "0 */12 * * *" (2)
    template:
      spec:
        source:
          registry: (3)
            url: docker://quay.io/containerdisks/centos:7-2009
        storage:
          resources:
            requests:
              storage: 10Gi
    managedDataSource: centos7 (4)
    retentionPolicy: "None" (5)
```

(1) このアノテーションは、**volumeBindingMode** が **WaitForFirstConsumer** に設定されたストレージクラスに必要です。

(2) cron 形式で指定されるジョブのスケジュール。

(3) レジストリーソースからデータボリュームを作成するのに使用します。**node** docker キャッシュに基づくデフォルトの **node pullMethod** ではなく、デフォルトの **pod pullMethod** を使用します。**node** docker キャッシュはレジストリーイメージが**Container.Image** で利用可能な場合に便利ですが、CDI インポーターはこれにアクセスすることは許可されていません。

(4) 利用可能なブートソースとして検出するカスタムイメージの場合、イメージの **managedDataSource** の名前が、仮想マシンテンプレート YAML ファイルの **spec.dataVolumeTemplates.spec.sourceRef.name** にあるテンプレートの **DataSource** の名前に一致する必要があります。

(5) cron ジョブが削除されたときにデータボリュームおよびデータソースを保持するには、**All** を使用します。cron ジョブが削除されたときにデータボリュームおよびデータソースを削除するには、**None** を使用します。

3. ファイルを保存します。

11.7.4. 特定のブートソースの自動更新を無効にする

個々のブートソースの自動更新を無効にします。

11.7.4.1. 単一ブートソースの自動更新を無効にする

HyperConverged カスタムリソース (CR) を編集することで、カスタムブートソースかシステム定義ブートソースかに関係なく、個々のブートソースの自動更新を無効にできます。

手順

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **spec.dataImportCronTemplates** フィールドを編集して、個々のブートソースの自動更新を無効にします。

カスタムブートソース

- **spec.dataImportCronTemplates** フィールドからブートソースを削除します。カスタムブートソースの自動更新はデフォルトで無効になっています。

システム定義のブートソース

a. ブートソースを **spec.dataImportCronTemplates** に追加します。

注記

システム定義のブートソースの自動更新はデフォルトで有効になっていますが、これらのブートソースは追加しない限り CR にリストされません。

b. **dataimportcrontemplate.kubevirt.io/enable** アノテーションの値を **'false'** に設定します。
以下に例を示します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      annotations:
        dataimportcrontemplate.kubevirt.io/enable: 'false'
      name: rhel8-image-cron
# ...
```

3. ファイルを保存します。

11.7.5. ブートソースのステータスの確認

HyperConverged カスタムリソース (CR) を表示することで、ブートソースがシステム定義であるかカスタムであるかを判断できます。

手順

1. 次のコマンドを実行して、**HyperConverged** CR の内容を表示します。

```
$ oc get hco -n openshift-cnv kubevirt-hyperconverged -o yaml
```

出力例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
```

```
  name: kubevirt-hyperconverged
spec:
# ...
status:
# ...
  dataImportCronTemplates:
  - metadata:
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true"
      name: centos-7-image-cron
    spec:
      garbageCollect: Outdated
      managedDataSource: centos7
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              url: docker://quay.io/containerdisks/centos:7-2009
          storage:
            resources:
              requests:
                storage: 30Gi
        status: {}
    status:
      commonTemplate: true (1)
# ...
  - metadata:
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true"
      name: user-defined-dic
    spec:
      garbageCollect: Outdated
      managedDataSource: user-defined-centos-stream8
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              pullMethod: node
              url: docker://quay.io/containerdisks/centos-stream:8
          storage:
            resources:
              requests:
                storage: 30Gi
        status: {}
    status: {} (2)
# ...
```

(1) システム定義のブートソースを示します。

(2) カスタムブートソースを示します。

2. **status.dataImportCronTemplates.status** フィールドを確認して、ブートソースのステータスを確認します。

   - フィールドに **commonTemplate: true** が含まれている場合、それはシステム定義のブートソースです。

   - **status.dataImportCronTemplates.status** フィールドの値が **{}** の場合、それはカスタムブートソースです。

第12章 ライブマイグレーション

12.1. 仮想マシンのライブマイグレーション

12.1.1. 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 (VMI) をクラスター内の別のノードに移行するプロセスです。VMI が **LiveMigrate** エビクションストラテジーを使用する場合、VMI が実行されるノードがメンテナンスモードになると自動的に移行されます。移行する VMI を選択して、ライブマイグレーションを手動で開始することもできます。

以下の条件を満たす場合は、ライブマイグレーションを使用できます。

- **ReadWriteMany** (RWX) アクセスモードの共有ストレージ
- 十分な RAM およびネットワーク帯域幅
- 仮想マシンがホストモデルの CPU を使用する場合、ノードは仮想マシンのホストモデルの CPU をサポートする必要があります。

デフォルトでは、ライブマイグレーショントラフィックは Transport Layer Security (TLS) を使用して暗号化されます。

12.1.2. 関連情報

- 仮想マシンインスタンスの別のノードへの移行
- ライブマイグレーションのメトリック
- ライブマイグレーションの制限
- ストレージプロファイルのカスタマイズ
- 仮想マシン移行のチューニング

12.2. 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を適用し、移行プロセスによる負荷がクラスターにかからないようにします。**HyperConverged** カスタムリソース (CR) を編集してこれらの設定を行います。

12.2.1. 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

openshift-cnv namespace にある **HyperConverged** カスタムリソース (CR) を更新して、クラスターの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

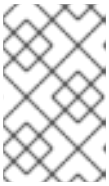
- **HyperConverged** CR を編集し、必要なライブマイグレーションパラメーターを追加します。

```
$ oc edit hco -n openshift-cnv kubevirt-hyperconverged
```

設定ファイルのサンプル

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  liveMigrationConfig: 1
    bandwidthPerMigration: 64Mi
    completionTimeoutPerGiB: 800
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    progressTimeout: 150
```

1 この例では、**spec.liveMigrationConfig** 配列には各フィールドのデフォルト値が含まれます。



注記

キー/値のペアを削除し、ファイルを保存して、**spec.liveMigrationConfig** フィールドのデフォルト値を復元できます。たとえば、**progressTimeout: <value>** を削除してデフォルトの **progressTimeout: 150** を復元します。

12.2.2. 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

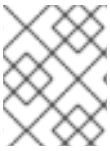
表12.1 移行パラメーター

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移行の数。	5
parallelOutboundMigrationsPerNode	ノードごとのアウトバウンドの移行の最大数。	2
bandwidthPerMigration	各マイグレーションの帯域幅制限。値は 1 秒あたりのバイト数です。たとえば、値 **2048Mi** は 2048 MiB/s を意味します。	0 [1]
completionTimeoutPerGiB	移行がこの時間内に終了しない場合 (単位はメモリーの GiB あたりの秒数)、移行は取り消されます。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 秒内に移行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	メモリーのコピーの進捗がこの時間内 (秒単位) に見られない場合に、移行は取り消されます。	150

1. デフォルト値の **0** は無制限です。

12.3. 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。



注記

仮想マシンがホストモデルの CPU を使用する場合は、そのホストモデル CPU をサポートするノード間でのみ仮想マシンのライブマイグレーションを行うことができます。

12.3.1. 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 アクションはすべてのユーザーに表示されますが、管理者ユーザーのみが仮想マシンの移行を開始できます。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. このページから移行を開始すると、同じページで複数の仮想マシンに対してアクションを簡単に実行できます。または、選択した仮想マシンの包括的な詳細を表示できる **VirtualMachine details** ページから移行を開始できます。
   - 仮想マシンの横にある Options メニュー  をクリックし、**Migrate** を選択します。
   - 仮想マシン名をクリックして **VirtualMachine details** ページを開き、**Actions → Migrate** 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。

12.3.1.1. Web コンソールを使用したライブマイグレーションの監視

Web コンソールの **Overview → Migrations タブ** で、すべてのライブマイグレーションの進行状況を監視できます。

仮想マシンの移行メトリクスは、Web コンソールの **VirtualMachine details → Metrics タブ** で表示できます。

12.3.2. 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/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: migration-job
spec:
  vmiName: vmi-fedora
```

2. 以下のコマンドを実行して、クラスター内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。

12.3.2.1. 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
```

出力例

```
# ...
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
```

12.3.3. 関連情報

- 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

12.4. 専用の追加ネットワークを介した仮想マシンの移行

ライブマイグレーション専用の Multus ネットワーク を設定できます。専用ネットワークは、ライブマイグレーション中のテナントワークロードに対するネットワークの飽和状態の影響を最小限に抑えます。

12.4.1. 仮想マシンのライブマイグレーション用の専用セカンダリーネットワークの設定

ライブマイグレーション専用のセカンダリーネットワークを設定するには、最初に CLI を使用して **openshift-cnv** namespace のブリッジネットワークアタッチメント定義を作成する必要があります。次に、**NetworkAttachmentDefinition** オブジェクトの名前を **HyperConverged** カスタムリソース (CR) に追加します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **cluster-admin** ロールを持つユーザーとしてクラスターにログインしている。
- Multus Container Network Interface (CNI) プラグインがクラスターにインストールされている。
- クラスター内のすべてのノードには少なくとも 2 つのネットワークインターフェイスカード (NIC) があり、ライブマイグレーションに使用される NIC が同じ VLAN に接続されている。
- 仮想マシン (VM) が **LiveMigrate** エビクションストラテジーで実行されている。

手順

1. **NetworkAttachmentDefinition** マニフェストを作成します。

設定ファイルのサンプル

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: my-secondary-network 1
  namespace: openshift-cnv 2
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", 3
    "mode": "bridge",
    "ipam": {
      "type": "whereabouts", 4
      "range": "10.200.5.0/24" 5
    }
  }'
```

1. **NetworkAttachmentDefinition** オブジェクトの名前。
2. **NetworkAttachmentDefinition** オブジェクトが存在する namespace。これは **openshift-cnv** である必要があります。

3 ライブマイグレーションに使用する NIC の名前。

4 このネットワーク接続定義のネットワークを提供する CNI プラグインの名前。

5 セカンダリーネットワークの IP アドレス範囲。この範囲は、メインのネットワークの IP アドレスと重複できません。

2. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

```
oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

3. **NetworkAttachmentDefinition** オブジェクトの名前を **HyperConverged** CR の **spec.liveMigrationConfig** スタンザに追加します。以下に例を示します。

設定ファイルのサンプル

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  liveMigrationConfig:
    completionTimeoutPerGiB: 800
    network: my-secondary-network 1
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    progressTimeout: 150
# ...
```

1 ライブマイグレーションに使用される Multus **NetworkAttachmentDefinition** オブジェクトの名前。

4. 変更を保存し、エディターを終了します。**virt-handler** Pod が再起動し、セカンダリーネットワークに接続されます。

検証

- 仮想マシンが実行されるノードがメンテナンスモードに切り替えられると、仮想マシンは自動的にクラスター内の別のノードに移行します。仮想マシンインスタンス (VMI) メタデータのターゲット IP アドレスを確認して、デフォルトの Pod ネットワークではなく、セカンダリーネットワーク上で移行が発生したことを確認できます。

```
oc get vmi <vmi_name> -o jsonpath='{.status.migrationState.targetNodeAddress}'
```

12.4.2. Web コンソールを使用して専用ネットワークを選択する

OpenShift Container Platform Web コンソールを使用して、ライブマイグレーション用の専用ネットワークを選択できます。

前提条件

- ライブマイグレーション用に Multus ネットワークが設定されている。

手順

1. OpenShift Container Platform Web コンソールで **Virtualization > Overview** に移動します。
2. **Settings** タブをクリックし、**Live migration** をクリックします。
3. **Live migration network** リストからネットワークを選択します。

12.4.3. 関連情報

- ライブマイグレーションの制限およびタイムアウト

12.5. 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。

12.5.1. 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

Web コンソール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キャンセルできます。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンの横にある Options メニュー  をクリックし、**Cancel Migration** を選択します。

12.5.2. 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

移行に関連付けられた **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
```

12.6. 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。

12.6.1. 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.template.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>
```

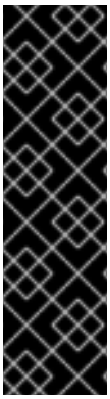
```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: custom-vm
spec:
  template:
    spec:
      evictionStrategy: LiveMigrate
# ...
```

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>
```

12.7. ライブマイグレーションポリシーの設定

ライブマイグレーションポリシーを使用して、指定した仮想マシンインスタンス (VMI) のグループに対して、さまざまな移行設定を定義できます。



重要

ライブマイグレーションポリシーは、テクノロジープレビュー機能のみとなります。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

Web コンソールを使用してライブマイグレーションポリシーを設定するには、MigrationPolicies ページのドキュメント を参照してください。

12.7.1. コマンドラインからのライブマイグレーションポリシーの設定

MigrationPolicy カスタムリソース定義 (CRD) を使用して、選択した仮想マシンインスタンス (VMI) の 1 つ以上のグループの移行ポリシーを定義します。

次の任意の組み合わせを使用して、VMI のグループを指定できます。

- **size**、**os**、**gpu**、およびその他の VMI ラベルなどの仮想マシンインスタンスラベル。

- **priority**、**bandwidth**、**hpc-workload**、およびその他の namespace ラベルなどの namespace ラベル。

ポリシーを特定の VMI グループに適用するには、VMI グループのすべてのラベルがポリシーのラベルと一致する必要があります。

注記

複数のライブマイグレーションポリシーが VMI に適用される場合は、一致するラベルの数が最も多いポリシーが優先されます。複数のポリシーがこの基準を満たす場合、ポリシーは一致するラベルキーの辞書式順序でソートされ、その順序の最初のポリシーが優先されます。

手順

1. 指定した VMI グループの **MigrationPolicy** CRD を作成します。次の YAML の例では、ラベル **hpc-workloads:true**、**xyz-workloads-type: ""**、**workload-type: db**、および **operating-system: ""** でグループを設定します。

```
apiVersion: migrations.kubevirt.io/v1beta1
kind: MigrationPolicy
metadata:
  name: my-awesome-policy
spec:
  # Migration Configuration
  allowAutoConverge: true
  bandwidthPerMigration: 217Ki
  completionTimeoutPerGiB: 23
  allowPostCopy: false

  # Matching to VMIs
  selectors:
    namespaceSelector: 1
      hpc-workloads: "True"
      xyz-workloads-type: ""
    virtualMachineInstanceSelector: 2
      workload-type: "db"
      operating-system: ""
```

1. namespace ラベルを使用して VMI のグループを定義するには、**namespaceSelector** を使用します。

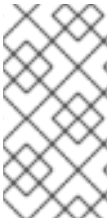
2. VMI ラベルを使用して VMI のグループを定義するには、**virtualMachineInstanceSelector** を使用します。

第13章 ノードのメンテナンス

13.1. ノードのメンテナンスについて

13.1.1. ノードのメンテナンスモードについて

oc adm ユーティリティーまたは **NodeMaintenance** カスタムリソース (CR) を使用してノードをメンテナンスモードにすることができます。



注記

node-maintenance-operator (NMO) は OpenShift Virtualization に同梱されなくなりました。OpenShift Container Platform Web コンソールの **OperatorHub** から、または OpenShift CLI (**oc**) を使用して、スタンドアロン Operator としてデプロイできるようになりました。

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シャットダウンします。**RunStrategy** が **Running** または **RerunOnFailure** の仮想マシンは別のノードで再作成されます。**RunStrategy** が **Manual** の仮想マシンは自動的に再起動されません。

重要

仮想マシンでは、共有 **ReadWriteMany** (RWX) アクセスモードを持つ永続ボリューム要求 (PVC) のライブマイグレーションが必要です。

Node Maintenance Operator は、新規または削除された **NodeMaintenance** CR をモニタリングします。新規の **NodeMaintenance** CR が検出されると、新規ワークロードはスケジュールされず、ノードは残りのクラスターから遮断されます。エビクトできるすべての Pod はノードからエビクトされます。**NodeMaintenance** CR が削除されると、CR で参照されるノードは新規ワークロードで利用可能になります。

注記

ノードのメンテナンスタスクに **NodeMaintenance** CR を使用すると、標準の OpenShift Container Platform カスタムリソース処理を使用して **oc adm cordon** および **oc adm drain** コマンドの場合と同じ結果が得られます。

13.1.2. ベアメタルノードのメンテナンス

OpenShift Container Platform をベアメタルインフラストラクチャーにデプロイする場合、クラウドインフラストラクチャーにデプロイする場合と比較すると、追加で考慮する必要のある点があります。クラスターノードが一時的とみなされるクラウド環境とは異なり、ベアメタルノードを再プロビジョニングするには、メンテナンスタスクにより多くの時間と作業が必要になります。

致命的なカーネルエラーが発生したり、NIC カードのハードウェア障害が発生する場合などにベアメタルノードに障害が発生した場合には、障害のあるノードが修復または置き換えられている間に、障害が

発生したノード上のワークロードをクラスターの別の場所で再起動する必要があります。ノードのメンテナンスモードにより、クラスター管理者はノードの電源を正常に停止し、ワークロードをクラスターの他の部分に移動させ、ワークロードが中断されないようにします。詳細な進捗とノードのステータスの詳細は、メンテナンス時に提供されます。

13.1.3. 関連情報

- CLI を使用した Node Maintenance Operator のインストール
- ノードのメンテナンスモードへの設定
- メンテナンスモードからのノードの再開
- 仮想マシンの RunStrategy について
- 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

13.2. TLS 証明書の自動更新

OpenShift Virtualization コンポーネントの TLS 証明書はすべて更新され、自動的にローテーションされます。手動で更新する必要はありません。

13.2.1. TLS 証明書の自動更新スケジュール

TLS 証明書は自動的に削除され、以下のスケジュールに従って置き換えられます。

- KubeVirt 証明書は毎日更新されます。
- Containerized Data Importer controller (CDI) 証明書は、15 日ごとに更新されます。
- MAC プール証明書は毎年更新されます。

TLS 証明書の自動ローテーションはいずれの操作も中断しません。たとえば、以下の操作は中断せずに引き続き機能します。

- 移行
- イメージのアップロード
- VNC およびコンソールの接続

13.3. 古い CPU モデルのノードラベルの管理

VM CPU モデルおよびポリシーがノードでサポートされている限り、ノードで仮想マシン (VM) をスケジュールできます。

13.3.1. 古い CPU モデルのノードラベリングについて

OpenShift Virtualization Operator は、古い CPU モデルの事前定義されたリストを使用して、ノードがスケジュールされた仮想マシンの有効な CPU モデルのみをサポートするようにします。

デフォルトでは、以下の CPU モデルはノード用に生成されたラベルのリストから削除されます。

例13.1 古い CPU モデル

```
"486"
Conroe
athlon
core2duo
coreduo
kvm32
kvm64
n270
pentium
pentium2
pentium3
pentiumpro
phenom
qemu32
qemu64
```

この事前定義された一覧は **HyperConverged** CR には表示されません。このリストから CPU モデルを **削除** できませんが、**HyperConverged** CR の **spec.obsoleteCPUs.cpuModels** フィールドを編集してリストに追加することはできます。

13.3.2. CPU 機能のノードのラベル付けについて

反復処理により、最小 CPU モデルのベース CPU 機能は、ノード用に生成されたラベルのリストから削除されます。

以下に例を示します。

- 環境には、**Penryn** と **Haswell** という 2 つのサポートされる CPU モデルが含まれる可能性があります。
- **Penryn** が **minCPU** の CPU モデルとして指定される場合、**Penryn** のベース CPU の各機能は **Haswell** によってサポートされる CPU 機能のリストと比較されます。

  例13.2 Penryn でサポートされる CPU 機能

  ```
  apic
  clflush
  cmov
  cx16
  cx8
  de
  fpu
  fxsr
  lahf_lm
  lm
  mca
  mce
  mmx
  msr
  mtrr
  nx
  pae
  ```

```
pat
pge
pni
pse
pse36
sep
sse
sse2
sse4.1
ssse3
syscall
tsc
```

例13.3 Haswell でサポートされる CPU 機能

```
aes
apic
avx
avx2
bmi1
bmi2
clflush
cmov
cx16
cx8
de
erms
fma
fpu
fsgsbase
fxsr
hle
invpcid
lahf_lm
lm
mca
mce
mmx
movbe
msr
mtrr
nx
pae
pat
pcid
pclmuldq
pge
pni
popcnt
pse
pse36
rdtscp
rtm
sep
```

```
smep
sse
sse2
sse4.1
sse4.2
ssse3
syscall
tsc
tsc-deadline
x2apic
xsave
```

- **Penryn** と **Haswell** の両方が特定の CPU 機能をサポートする場合、その機能にラベルは作成されません。ラベルは、**Haswell** でのみサポートされ、**Penryn** ではサポートされていない CPU 機能用に生成されます。

例13.4 反復後に CPU 機能用に作成されるノードラベル

```
aes
avx
avx2
bmi1
bmi2
erms
fma
fsgsbase
hle
invpcid
movbe
pcid
pclmuldq
popcnt
rdtscp
rtm
sse4.2
tsc-deadline
x2apic
xsave
```

13.3.3. 古い CPU モデルの設定

HyperConverged カスタムリソース (CR) を編集して、古い CPU モデルのリストを設定できます。

手順

- 古い CPU モデルを **obsoleteCPUs** 配列で指定して、**HyperConverged** カスタムリソースを編集します。以下に例を示します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
```

```
spec:
  obsoleteCPUs:
    cpuModels: 1
      - "<obsolete_cpu_1>"
      - "<obsolete_cpu_2>"
    minCPUModel: "<minimum_cpu_model>" 2
```

1 **cpuModels** 配列のサンプル値を、古くなった CPU モデルに置き換えます。指定した値はすべて、古くなった CPU モデルの事前定義リストに追加されます。事前定義されたリストは CR に表示されません。

2 この値を、基本的な CPU 機能に使用する最小 CPU モデルに置き換えます。値を指定しない場合は、デフォルトで **Penryn** が使用されます。

13.4. ノードの調整の防止

skip-node アノテーションを使用して、**node-labeller** がノードを調整できないようにします。

13.4.1. skip-node アノテーションの使用

node-labeller でノードを省略するには、**oc** CLI を使用してそのノードにアノテーションを付けます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

- 以下のコマンドを実行して、スキップするノードにアノテーションを付けます。

```
$ oc annotate node <node_name> node-labeller.kubevirt.io/skip-node=true 1
```

1 **<node_name>** は、スキップする関連ノードの名前に置き換えます。

調整は、ノードアノテーションが削除されるか、false に設定された後に、次のサイクルで再開されます。

13.4.2. 関連情報

- 古い CPU モデルのノードラベルの管理

第14章 SUPPORT

14.1. サポートの概要

以下のツールを使用して、環境に関するデータを収集し、クラスターと仮想マシン (VM) の状態を監視し、OpenShift 仮想化リソースのトラブルシューティングを行うことができます。

14.1.1. Web コンソール

OpenShift Container Platform Web コンソールには、クラスターと OpenShift Virtualization コンポーネントおよびリソースのリソース使用量、アラート、イベント、および傾向が表示されます。

表14.1 監視とトラブルシューティングのための Web コンソールページ

ページ	説明
Overview ページ	クラスターの詳細、ステータス、アラート、インベントリー、およびリソースの使用状況
Virtualization → Overview タブ	OpenShift 仮想化のリソース、使用状況、アラート、およびステータス
Virtualization → Top consumers タブ	CPU、メモリー、およびストレージの上位コンシューマー
Virtualization → Migrations タブ	ライブマイグレーションの進捗
VirtualMachines → **VirtualMachine** → **VirtualMachine details** → Metrics タブ	VM リソースの使用状況、ストレージ、ネットワーク、および移行
VirtualMachines → **VirtualMachine** → **VirtualMachine details** → Events タブ	VM イベントリスト
VirtualMachines → **VirtualMachine** → **VirtualMachine details** → Diagnostics タブ	仮想マシンのステータス条件とボリュームスナップショットのステータス

14.1.2. Red Hat サポート用のデータ収集

Red Hat サポートに サポートケース を送信する場合、デバッグ情報を提供すると役立ちます。次の手順を実行して、デバッグ情報を収集できます。

環境に関するデータの収集

Prometheus および Alertmanager を設定し、OpenShift Container Platform および OpenShift Virtualization の **must-gather** データを収集します。

VM に関するデータの収集

must-gather データとメモリーダンプを VM から収集します。

OpenShift 仮想化のための must-gather ツール

must-gather ツールを設定および使用します。

14.1.3. モニタリング

クラスターと VM の正常性を監視できます。モニタリングツールの詳細については、モニタリングの概要 を参照してください。

14.1.4. トラブルシューティング

OpenShift Virtualization コンポーネントと VM のトラブルシューティングを行い、Web コンソールでアラートをトリガーする問題を解決します。

イベント

VM、namespace、およびリソースの重要なライフサイクル情報を表示します。

ログ

OpenShift Virtualization コンポーネントおよび VM のログを表示および設定します。

ランブック

Web コンソールで OpenShift Virtualization アラートをトリガーする問題を診断および解決します。

データボリュームのトラブルシューティング

条件とイベントを分析して、データボリュームのトラブルシューティングを行います。

14.2. RED HAT サポート用のデータ収集

Red Hat サポートに サポートケース を送信する際に、以下のツールを使用して OpenShift Container Platform と OpenShift Virtualization のデバッグ情報を提供すると役立ちます。

must-gather ツール

must-gather ツールは、リソース定義やサービスログなどの診断情報を収集します。

Prometheus

Prometheus は Time Series を使用するデータベースであり、メトリクスのルール評価エンジンです。Prometheus は処理のためにアラートを Alertmanager に送信します。

Alertmanager

Alertmanager サービスは、Prometheus から送信されるアラートを処理します。また、Alertmanager は外部の通知システムにアラートを送信します。

OpenShift Container Platform モニタリングスタックの詳細は、OpenShift Container Platform モニタリングについて を参照してください。

14.2.1. 環境に関するデータの収集

環境に関するデータを収集すると、根本原因の分析および特定に必要な時間が最小限に抑えられます。

前提条件

- Prometheus メトリクスデータの保持期間を最低 7 日間に設定する。
- クラスターの外部で表示および永続化できるように、Alertmanager を設定して、関連するアラートをキャプチャーし、アラート通知を専用のメールボックスに送信 する。
- 影響を受けるノードおよび仮想マシンの正確な数を記録する。

手順

1. クラスターの must-gather データを収集 します。
2. 必要に応じて、Red Hat OpenShift Data Foundation の must-gather データを収集 します。
3. OpenShift Virtualization の必須データを収集 します。
4. クラスターの Prometheus メトリクスを収集 します。

14.2.2. 仮想マシンに関するデータの収集

仮想マシン (VM) の誤動作に関するデータを収集することで、根本原因の分析および特定に必要な時間を最小限に抑えることができます。

前提条件

- Linux VM: 最新の QEMU ゲストエージェントをインストール します。
- Windows 仮想マシン:
  - Windows パッチ更新の詳細を記録します。
  - 最新の VirtIO ドライバーをインストール します。
  - 最新の QEMU ゲストエージェントをインストール します。
  - リモートデスクトッププロトコル (RDP) が有効になっている場合は、Web コンソール または コマンドライン を使用して RDP で VM に接続し、接続ソフトウェアに問題があるかどうかを確認してください。

手順

1. **/usr/bin/gather** スクリプトを使用して、VM の必須収集データを収集 します。
2. VM を再起動する **前** に、クラッシュした VM のスクリーンショットを収集します。
3. 修復を試みる **前** に、VM からメモリーダンプを収集 します。
4. 誤動作している仮想マシンに共通する要因を記録します。たとえば、仮想マシンには同じホストまたはネットワークがあります。

14.2.3. OpenShift Virtualization の must-gather ツールの使用

OpenShift Virtualization イメージで **must-gather** コマンドを実行することにより、OpenShift Virtualization リソースに関するデータを収集できます。

デフォルトのデータ収集には、次のリソースに関する情報が含まれています。

- 子オブジェクトを含む OpenShift Virtualization Operator namespace
- すべての OpenShift Virtualization カスタムリソース定義
- 仮想マシンを含むすべての namespace
- 基本的な仮想マシン定義

手順

- 以下のコマンドを実行して、OpenShift Virtualization に関するデータを収集します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.13.11 \
  -- /usr/bin/gather
```

14.2.3.1. must-gather ツールオプション

次のオプションに対して、スクリプトおよび環境変数の組み合わせを指定できます。

- namespace から詳細な仮想マシン (VM) 情報の収集する
- 特定の仮想マシンに関する詳細情報の収集
- image、image-stream、および image-stream-tags 情報の収集
- **must-gather** ツールが使用する並列プロセスの最大数の制限

14.2.3.1.1. パラメーター

環境変数

互換性のあるスクリプトの環境変数を指定できます。

NS=<namespace_name>

指定した namespace から **virt-launcher** Pod の詳細を含む仮想マシン情報を収集します。**VirtualMachine** および **VirtualMachineInstance** CR データはすべての namespace で収集されます。

VM=<vm_name>

特定の仮想マシンに関する詳細を収集します。このオプションを使用するには、**NS** 環境変数を使用して namespace も指定する必要があります。

PROS=<number_of_processes>

must-gather ツールが使用する並列処理の最大数を変更します。デフォルト値は **5** です。

重要

並列処理が多すぎると、パフォーマンスの問題が発生する可能性があります。並列処理の最大数を増やすことは推奨されません。

スクリプト

各スクリプトは、特定の環境変数の組み合わせとのみ互換性があります。

/usr/bin/gather

デフォルトの **must-gather** スクリプトを使用します。すべての namespace からクラスターデータが収集され、基本的な仮想マシン情報のみが含まれます。このスクリプトは、**PROS** 変数とのみ互換性があります。

/usr/bin/gather --vms_details

OpenShift Virtualization リソースに属する VM ログファイル、VM 定義、コントロールプレーンログ、および namespace を収集します。namespace の指定には、その子オブジェクトが含まれます。namespace または仮想マシンを指定せずにこのパラメーターを使用する場合、**must-gather**

ツールはクラスター内のすべての仮想マシンについてこのデータを収集します。このスクリプトはすべての環境変数と互換性がありますが、**VM** 変数を使用する場合は namespace を指定する必要があります。

/usr/bin/gather --images

image、image-stream、および image-stream-tags カスタムリソース情報を収集します。このスクリプトは、**PROS** 変数とのみ互換性があります。

14.2.3.1.2. 使用方法および例

環境変数はオプションです。スクリプトは、単独で実行することも、1 つ以上の互換性のある環境変数を使用して実行することもできます。

表14.2 互換性のあるパラメーター

スクリプト	互換性のある環境変数
/usr/bin/gather	• **PROS=<number_of_processes>**
/usr/bin/gather --vms_details	• namespace の場合: **NS=<namespace_name>** • 仮想マシンの場合: **VM=<vm_name> NS=<namespace_name>** • **PROS=<number_of_processes>**
/usr/bin/gather --images	• **PROS=<number_of_processes>**

構文

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.13.11 \
  -- <environment_variable_1> <environment_variable_2> <script_name>
```

デフォルトのデータ収集の並列プロセス

デフォルトでは、5 つのプロセスを並行して実行します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.13.11 \
  -- PROS=5 /usr/bin/gather 1
```

1 デフォルトを変更することで、並列プロセスの数を変更できます。

詳細な仮想マシン情報

次のコマンドは、**mynamespace** namespace にある **my-vm** 仮想マシンの詳細な仮想マシン情報を収集します。

-

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.13.11 \
  -- NS=mynamespace VM=my-vm /usr/bin/gather --vms_details 1
```

1 **VM** 環境変数を使用する場合、**NS** 環境変数は必須です。

image、image-stream、および image-stream-tags 情報

以下のコマンドは、クラスターからイメージ、image-stream、および image-stream-tags 情報を収集します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.13.11 \
  /usr/bin/gather --images
```

14.3. モニタリング

14.3.1. モニタリングの概要

次のツールを使用して、クラスターと仮想マシン (VM) の正常性を監視できます。

OpenShift Container Platform クラスター診断フレームワーク

OpenShift Container Platform クラスター診断フレームワークを使用してクラスターに対する自動テストを実行し、以下の状態を確認します。

- セカンダリーネットワークインターフェイスに接続された 2 つの仮想マシン間のネットワーク接続とレイテンシー
- パケット損失ゼロで Data Plane Development Kit (DPDK) ワークロードを実行する仮想マシン

重要

OpenShift Container Platform クラスターチェックアップフレームワークは、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

仮想リソースの Prometheus クエリー

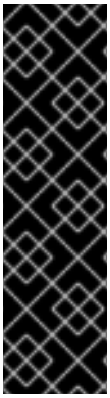
vCPU、ネットワーク、ストレージ、およびゲストメモリースワッピングの使用状況とライブマイグレーションの進行状況をクエリーします。

VM カスタムメトリクス

内部 VM メトリクスとプロセスを公開するように、**node-exporter** サービスを設定します。

xref:[VM ヘルスチェック]

レディネス、ライブネス、ゲストエージェントの ping プローブ、および VM のウォッチドッグを設定します。



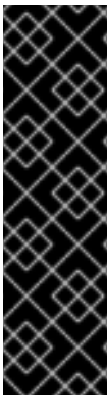
重要

ゲストエージェント ping プローブは、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

14.3.2. OpenShift Container Platform クラスター診断フレームワーク

OpenShift Virtualization には、クラスターのメンテナンスとトラブルシューティングに使用できる定義済みのチェックアップが含まれています。



重要

OpenShift Container Platform クラスターチェックアップフレームワークは、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

14.3.2.1. OpenShift Container Platform クラスターチェックアップフレームワークについて

チェックアップは、特定のクラスター機能が期待どおりに機能するかどうかを確認できる自動テストワークロードです。クラスター検査フレームワークは、ネイティブの Kubernetes リソースを使用して、チェックアップを設定および実行します。

事前定義されたチェックアップを使用することで、クラスター管理者はクラスターの保守性を向上させ、予期しない動作をトラブルシューティングし、エラーを最小限に抑え、時間を節約できます。また、チェックアップの結果を確認し、専門家と共有してさらに分析することもできます。ベンダーは、提供する機能やサービスのチェックアップを作成して公開し、顧客環境が正しく設定されていることを確認できます。

既存の namespace で事前定義されたチェックアップを実行するには、チェックアップ用のサービスアカウントの設定、サービスアカウント用の **Role** および **RoleBinding** オブジェクトの作成、チェックアップのパーミッションの有効化、入力 config map とチェックアップジョブの作成が含まれます。チェックアップは複数回実行できます。

重要

以下が常に必要になります。

- チェックアップイメージを適用する前に、信頼できるソースからのものであることを確認します。
- **Role** および **RoleBinding** オブジェクトを作成する前に、チェックアップパーミッションを確認してください。

14.3.2.2. 仮想マシンのレイテンシーチェックアップ

定義済みのチェックアップを使用して、ネットワーク接続を確認し、セカンダリーネットワークインターフェイスに接続されている 2 つの仮想マシン (VM) 間の待機時間を測定します。レイテンシーチェックアップでは ping ユーティリティーを使用します。

次の手順でレイテンシーチェックアップを実行します。

1. サービスアカウント、ロール、ロールバインディングを作成して、レイテンシーチェックアップへのクラスターアクセス権を提供します。
2. config map を作成し、チェックアップを実行して結果を保存するための入力を行います。
3. チェックアップを実行するジョブを作成します。
4. config map で結果を確認します。
5. オプション: チェックアップを再実行するには、既存の config map とジョブを削除し、新しい config map とジョブを作成します。
6. 完了したら、レイテンシーチェックアップリソースを削除します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスターには少なくとも 2 つのワーカーノードがあります。
- Multus Container Network Interface (CNI) プラグインがクラスターにインストールされている。
- namespace のネットワーク接続定義を設定しました。

手順

1. レイテンシーチェックアップ用の **ServiceAccount**、**Role**、**RoleBinding** マニフェストを作成します。

例14.1 ロールマニフェストファイルの例

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: vm-latency-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: Role
metadata:
  name: kubevirt-vm-latency-checker
rules:
- apiGroups: ["kubevirt.io"]
  resources: ["virtualmachineinstances"]
  verbs: ["get", "create", "delete"]
- apiGroups: ["subresources.kubevirt.io"]
  resources: ["virtualmachineinstances/console"]
  verbs: ["get"]
- apiGroups: ["k8s.cni.cncf.io"]
  resources: ["network-attachment-definitions"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kubevirt-vm-latency-checker
subjects:
- kind: ServiceAccount
  name: vm-latency-checkup-sa
roleRef:
  kind: Role
  name: kubevirt-vm-latency-checker
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kiagnose-configmap-access
rules:
- apiGroups: [ "" ]
  resources: [ "configmaps" ]
  verbs: ["get", "update"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kiagnose-configmap-access
subjects:
- kind: ServiceAccount
  name: vm-latency-checkup-sa
roleRef:
  kind: Role
  name: kiagnose-configmap-access
  apiGroup: rbac.authorization.k8s.io
```

2. **ServiceAccount**、**Role**、**RoleBinding** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <latency_sa_roles_rolebinding>.yaml 1
```

1 **<target_namespace>** は、チェックアップを実行する namespace です。これは、**NetworkAttachmentDefinition** オブジェクトが存在する既存の namespace である必要があります。

3. チェックアップの入力パラメーターを含む **ConfigMap** マニフェストを作成します。

入力 config map の例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-vm-latency-checkup-config
data:
  spec.timeout: 5m
  spec.param.networkAttachmentDefinitionNamespace: <target_namespace>
  spec.param.networkAttachmentDefinitionName: "blue-network" (1)
  spec.param.maxDesiredLatencyMilliseconds: "10" (2)
  spec.param.sampleDurationSeconds: "5" (3)
  spec.param.sourceNode: "worker1" (4)
  spec.param.targetNode: "worker2" (5)
```

(1) **NetworkAttachmentDefinition** オブジェクトの名前。

(2) オプション: 仮想マシン間の必要な最大待機時間 (ミリ秒単位)。測定されたレイテンシーがこの値を超えると、チェックアップは失敗します。

(3) オプション: レイテンシーチェックの継続時間 (秒単位)。

(4) オプション: 指定すると、このノードからターゲットノードまでの待ち時間が測定されます。ソースノードが指定されている場合、**spec.param.targetNode** フィールドは空にできません。

(5) オプション: 指定すると、ソースノードからこのノードまでの待ち時間が測定されます。

4. ターゲット namespace に config map マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <latency_config_map>.yaml
```

5. チェックアップを実行する **Job** マニフェストを作成します。

ジョブマニフェストの例

```
apiVersion: batch/v1
kind: Job
metadata:
  name: kubevirt-vm-latency-checkup
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccountName: vm-latency-checkup-sa
      restartPolicy: Never
      containers:
        - name: vm-latency-checkup
          image: registry.redhat.io/container-native-virtualization/vm-network-latency-checkup-
rhel9:v4.13.0
          securityContext:
            allowPrivilegeEscalation: false
```

```
        capabilities:
          drop: ["ALL"]
        runAsNonRoot: true
        seccompProfile:
          type: "RuntimeDefault"
      env:
        - name: CONFIGMAP_NAMESPACE
          value: <target_namespace>
        - name: CONFIGMAP_NAME
          value: kubevirt-vm-latency-checkup-config
        - name: POD_UID
          valueFrom:
            fieldRef:
              fieldPath: metadata.uid
```

6. **Job** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <latency_job>.yaml
```

7. ジョブが完了するまで待ちます。

```
$ oc wait job kubevirt-vm-latency-checkup -n <target_namespace> --for condition=complete --timeout 6m
```

8. 以下のコマンドを実行して、レイテンシーチェックアップの結果を確認します。測定された最大レイテンシーが **spec.param.maxDesiredLatencyMilliseconds** 属性の値よりも大きい場合、チェックアップは失敗し、エラーが返されます。

```
$ oc get configmap kubevirt-vm-latency-checkup-config -n <target_namespace> -o yaml
```

出力 config map の例 (成功)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-vm-latency-checkup-config
  namespace: <target_namespace>
data:
  spec.timeout: 5m
  spec.param.networkAttachmentDefinitionNamespace: <target_namespace>
  spec.param.networkAttachmentDefinitionName: "blue-network"
  spec.param.maxDesiredLatencyMilliseconds: "10"
  spec.param.sampleDurationSeconds: "5"
  spec.param.sourceNode: "worker1"
  spec.param.targetNode: "worker2"
  status.succeeded: "true"
  status.failureReason: ""
  status.completionTimestamp: "2022-01-01T09:00:00Z"
  status.startTimestamp: "2022-01-01T09:00:07Z"
  status.result.avgLatencyNanoSec: "177000"
  status.result.maxLatencyNanoSec: "244000" (1)
  status.result.measurementDurationSec: "5"
```

```
status.result.minLatencyNanoSec: "135000"
status.result.sourceNode: "worker1"
status.result.targetNode: "worker2"
```

1 測定された最大レイテンシー (ナノ秒)。

9. オプション: チェックアップが失敗した場合に詳細なジョブログを表示するには、次のコマンドを使用します。

```
$ oc logs job.batch/kubevirt-vm-latency-checkup -n <target_namespace>
```

10. 以下のコマンドを実行して、以前に作成したジョブおよび config map を削除します。

```
$ oc delete job -n <target_namespace> kubevirt-vm-latency-checkup
```

```
$ oc delete config-map -n <target_namespace> kubevirt-vm-latency-checkup-config
```

11. オプション: 別のチェックアップを実行する予定がない場合は、ロールマニフェストを削除します。

```
$ oc delete -f <latency_sa_roles_rolebinding>.yaml
```

14.3.2.3. DPDK チェックアップ

事前定義されたチェックアップを使用して、OpenShift Container Platform クラスターノードが Data Plane Development Kit (DPDK) ワークロードがある仮想マシン (VM) をパケット損失ゼロで実行できるか確認します。DPDK チェックアップは、トラフィックジェネレーター Pod とテスト DPDK アプリケーションを実行している 仮想マシン間でトラフィックを実行します。

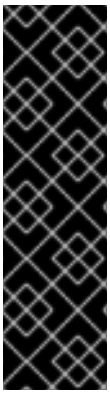
次の手順で DPDK チェックアップを実行します。

1. DPDK チェックアップ用のサービスアカウント、ロール、ロールバインディング、およびトラフィックジェネレーター Pod のサービスアカウントを作成します。
2. トラフィックジェネレーター Pod のセキュリティーコンテキスト制約リソースを作成します。
3. config map を作成し、チェックアップを実行して結果を保存するための入力を行います。
4. チェックアップを実行するジョブを作成します。
5. config map で結果を確認します。
6. オプション: チェックアップを再実行するには、既存の config map とジョブを削除し、新しい config map とジョブを作成します。
7. 完了したら、DPDK チェックリソースを削除します。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

- VM でパケット損失なしで DPDK アプリケーションを実行するようにコンピュートノードを設定しました。



重要

チェックによって作成されたトラフィックジェネレーター Pod には、昇格した権限があります。

- root として実行されます。
- ノードのファイルシステムにバインドマウントがあります。

トラフィックジェネレーターのコンテナーイメージは、アップストリームの Project Quay コンテナーレジストリーから取得されます。

手順

1. DPDK チェックアップとトラフィックジェネレーター Pod 用の **ServiceAccount**、**role**、**roleBinding** マニフェストを作成します。

例14.2 サービスアカウント、ロール、ロールバインディングマニフェストファイルの例

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dpdk-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kiagnose-configmap-access
rules:
  - apiGroups: [ "" ]
    resources: [ "configmaps" ]
    verbs: [ "get", "update" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kiagnose-configmap-access
subjects:
  - kind: ServiceAccount
    name: dpdk-checkup-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kiagnose-configmap-access
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kubevirt-dpdk-checker
rules:
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachineinstances" ]
```

```
    verbs: [ "create", "get", "delete" ]
  - apiGroups: [ "subresources.kubevirt.io" ]
    resources: [ "virtualmachineinstances/console" ]
    verbs: [ "get" ]
  - apiGroups: [ "" ]
    resources: [ "pods" ]
    verbs: [ "create", "get", "delete" ]
  - apiGroups: [ "" ]
    resources: [ "pods/exec" ]
    verbs: [ "create" ]
  - apiGroups: [ "k8s.cni.cncf.io" ]
    resources: [ "network-attachment-definitions" ]
    verbs: [ "get" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kubevirt-dpdk-checker
subjects:
  - kind: ServiceAccount
    name: dpdk-checkup-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kubevirt-dpdk-checker
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dpdk-checkup-traffic-gen-sa
```

2. **ServiceAccount**、**Role**、**RoleBinding** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <dpdk_sa_roles_rolebinding>.yaml
```

3. トラフィックジェネレーター Pod の **SecurityContextConstraints** マニフェストを作成します。

セキュリティーコンテキスト制約マニフェストの例

```
apiVersion: security.openshift.io/v1
kind: SecurityContextConstraints
metadata:
  name: dpdk-checkup-traffic-gen
allowHostDirVolumePlugin: true
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: false
allowPrivilegedContainer: false
allowedCapabilities:
- IPC_LOCK
- NET_ADMIN
```

```
- NET_RAW
- SYS_RESOURCE
defaultAddCapabilities: null
fsGroup:
  type: RunAsAny
groups: []
readOnlyRootFilesystem: false
requiredDropCapabilities: null
runAsUser:
  type: RunAsAny
seLinuxContext:
  type: RunAsAny
seccompProfiles:
- runtime/default
- unconfined
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:dpdk-checkup-ns:dpdk-checkup-traffic-gen-sa
```

4. **SecurityContextConstraints** マニフェストを適用します。

```
$ oc apply -f <dpdk_scc>.yaml
```

5. チェックアップの入力パラメーターを含む **ConfigMap** マニフェストを作成します。

入力 config map の例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dpdk-checkup-config
data:
  spec.timeout: 10m
  spec.param.networkAttachmentDefinitionName: <network_name> 1
  spec.param.trafficGeneratorRuntimeClassName: <runtimeclass_name> 2
  spec.param.trafficGeneratorImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-traffic-gen:v0.1.1" 3
  spec.param.vmContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-vm:v0.1.1" 4
```

1. **NetworkAttachmentDefinition** オブジェクトの名前。
2. トラフィックジェネレーター Pod が使用する **RuntimeClass** リソース。
3. トラフィックジェネレーターのコンテナーイメージ。この例では、アップストリームの Project Quay Container Registry からイメージをプルしています。
4. VM のコンテナーディスクイメージ。この例では、アップストリームの Project Quay Container Registry からイメージをプルしています。

6. ターゲット namespace に **ConfigMap** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <dpdk_config_map>.yaml
```

7. チェックアップを実行する **Job** マニフェストを作成します。

ジョブマニフェストの例

```
apiVersion: batch/v1
kind: Job
metadata:
  name: dpdk-checkup
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccountName: dpdk-checkup-sa
      restartPolicy: Never
      containers:
        - name: dpdk-checkup
          image: registry.redhat.io/container-native-virtualization/kubevirt-dpdk-checkup-
rhel9:v4.13.0
          imagePullPolicy: Always
          securityContext:
            allowPrivilegeEscalation: false
            capabilities:
              drop: ["ALL"]
            runAsNonRoot: true
            seccompProfile:
              type: "RuntimeDefault"
          env:
            - name: CONFIGMAP_NAMESPACE
              value: <target-namespace>
            - name: CONFIGMAP_NAME
              value: dpdk-checkup-config
            - name: POD_UID
              valueFrom:
                fieldRef:
                  fieldPath: metadata.uid
```

8. **Job** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <dpdk_job>.yaml
```

9. ジョブが完了するまで待ちます。

```
$ oc wait job dpdk-checkup -n <target_namespace> --for condition=complete --timeout 10m
```

10. 次のコマンドを実行して、検査の結果を確認します。

```
$ oc get configmap dpdk-checkup-config -n <target_namespace> -o yaml
```

出力 config map の例 (成功)

```
apiVersion: v1
kind: ConfigMap
metadata:
```

```
  name: dpdk-checkup-config
data:
  spec.timeout: 1h2m
  spec.param.NetworkAttachmentDefinitionName: "mlx-dpdk-network-1"
  spec.param.trafficGeneratorRuntimeClassName: performance-performance-zeus10
  spec.param.trafficGeneratorImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-traffic-gen:v0.1.1"
  spec.param.vmContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-vm:v0.1.1"
  status.succeeded: true
  status.failureReason: " "
  status.startTimestamp: 2022-12-21T09:33:06+00:00
  status.completionTimestamp: 2022-12-21T11:33:06+00:00
  status.result.actualTrafficGeneratorTargetNode: worker-dpdk1
  status.result.actualDPDKVMTargetNode: worker-dpdk2
  status.result.dropRate: 0
```

11. 以下のコマンドを実行して、以前に作成したジョブおよび config map を削除します。

```
$ oc delete job -n <target_namespace> dpdk-checkup
```

```
$ oc delete config-map -n <target_namespace> dpdk-checkup-config
```

12. オプション: 別のチェックアップを実行する予定がない場合は、**ServiceAccount**、**Role**、**RoleBinding** マニフェストを削除します。

```
$ oc delete -f <dpdk_sa_roles_rolebinding>.yaml
```

14.3.2.3.1. DPDK チェックアップ config map パラメーター

次の表は、クラスター DPDK 準備状況チェックを実行するときに、入力 **ConfigMap** マニフェストの **data** スタンザに設定できる必須パラメーターとオプションパラメーターを示しています。

表14.3 DPDK チェックアップ config map パラメーター

パラメーター	説明	必須かどうか
spec.timeout	チェックアップが失敗するまでの時間 (分単位)。	True
spec.param.networkAttachmentDefinitionName	接続されている SR-IOV NIC の **NetworkAttachmentDefinition** オブジェクトの名前。	True
spec.param.trafficGeneratorRuntimeClassName	トラフィックジェネレーター Pod が使用する RuntimeClass リソース。	True
spec.param.trafficGeneratorImage	トラフィックジェネレーターのコンテナーイメージ。デフォルト値は **quay.io/kiagnose/kubevirt-dpdk-checkup-traffic-gen:main** です。	False

パラメーター	説明	必須かどうか
spec.param.trafficGenerator NodeSelector	トラフィックジェネレーター Pod がスケジュールされるノード。DPDK トラフィックを許可するようにノードを設定する必要があります。	False
spec.param.trafficGenerator PacketsPerSecond	1 秒あたりのパケット数 (キロ (k) または 100 万 (m) 単位)。デフォルト値は 14m です。	False
spec.param.trafficGenerator EastMacAddress	トラフィックジェネレーター Pod または VM に接続されている NIC の MAC アドレス。デフォルト値は、**50:xx:xx:xx:xx:01** の形式のランダムな MAC アドレスです。	False
spec.param.trafficGenerator WestMacAddress	トラフィックジェネレーター Pod または VM に接続されている NIC の MAC アドレス。デフォルト値は、**50:xx:xx:xx:xx:02** の形式のランダムな MAC アドレスです。	False
spec.param.vmContainerDiskImage	VM のコンテナーディスクイメージ。デフォルト値は **quay.io/kiagnose/kubevirt-dpdk-checkup-vm:main** です。	False
spec.param.DPDKLabelSelector	VM が実行されているノードのラベル。DPDK トラフィックを許可するようにノードを設定する必要があります。	False
spec.param.DPDKEastMacAddress	VM に接続されている NIC の MAC アドレス。デフォルト値は、**60:xx:xx:xx:xx:01** の形式のランダムな MAC アドレスです。	False
spec.param.DPDKWestMacAddress	VM に接続されている NIC の MAC アドレス。デフォルト値は、**60:xx:xx:xx:xx:02** の形式のランダムな MAC アドレスです。	False

パラメーター	説明	必須かどうか
spec.param.testDuration	トラフィックジェネレーターが実行される期間 (分単位)。デフォルト値は 5 分です。	False
spec.param.portBandwidthGB	SR-IOV NIC の最大帯域幅。デフォルト値は 10GB です。	False
spec.param.verbose	**true** に設定すると、検査ログの詳細度が上がります。デフォルト値は **false** です。	False

14.3.2.3.2. RHEL 仮想マシン用コンテナーディスクイメージのビルド

カスタムの Red Hat Enterprise Linux (RHEL) 8 OS イメージを **qcow2** 形式でビルドし、それを使用してコンテナーディスクイメージを作成できます。クラスターからアクセス可能なレジストリーにコンテナーディスクイメージを保存し、DPDK チェック config map の **spec.param.vmContainerDiskImage** 属性でイメージの場所を指定できます。

コンテナーディスクイメージをビルドするには、Image Builder 仮想マシン (VM) を作成する必要があります。**Image Builder 仮想マシン** は、カスタム RHEL イメージのビルドに使用できる RHEL 8 仮想マシンです。

前提条件

- Image Builder 仮想マシンは RHEL 8.7 を実行でき、**/var** ディレクトリーに少なくとも 2 つの CPU コア、4 GiB RAM、20 GB の空き領域がある。
- Image Builder ツールとその CLI (**composer-cli**) が仮想マシンにインストールされている。
- **virt-customize** ツールがインストールされている。

```
# dnf install libguestfs-tools
```

- Podman CLI ツール (**podman**) がインストールされている。

手順

1. RHEL 8.7 イメージをビルドできることを確認します。

```
# composer-cli distros list
```

注記

非 root として **composer-cli** コマンドを実行するには、ユーザーを **weldr** または **root** グループに追加します。

```
# usermod -a -G weldr user
```

```
$ newgrp weldr
```

2. 次のコマンドを入力して、インストールするパッケージ、カーネルのカスタマイズ、起動時に無効化するサービスを含むイメージブループリントファイルを TOML 形式で作成します。

```
$ cat << EOF > dpdk-vm.toml
name = "dpdk_image"
description = "Image to use with the DPDK checkup"
version = "0.0.1"
distro = "rhel-87"

[[packages]]
name = "dpdk"

[[packages]]
name = "dpdk-tools"

[[packages]]
name = "driverctl"

[[packages]]
name = "tuned-profiles-cpu-partitioning"

[customizations.kernel]
append = "default_hugepagesz=1GB hugepagesz=1G hugepages=8 isolcpus=2-7"

[customizations.services]
disabled = ["NetworkManager-wait-online", "sshd"]
EOF
```

3. 次のコマンドを実行して、ブループリントファイルを Image Builder ツールにプッシュします。

```
# composer-cli blueprints push dpdk-vm.toml
```

4. ブループリント名と出力ファイル形式を指定して、システムイメージを生成します。作成プロセスを開始すると、イメージのユニバーサル一意識別子 (UUID) が表示されます。

```
# composer-cli compose start dpdk_image qcow2
```

5. 作成プロセスが完了するまで待ちます。作成ステータスが **FINISHED** になると次のステップに進めます。

```
# composer-cli compose status
```

6. 次のコマンドを入力し、UUID を指定して **qcow2** イメージファイルをダウンロードします。

```
# composer-cli compose image <UUID>
```

7. 次のコマンドを実行して、カスタマイズスクリプトを作成します。

```
$ cat <<EOF >customize-vm
echo  isolated_cores=2-7 > /etc/tuned/cpu-partitioning-variables.conf
tuned-adm profile cpu-partitioning
echo "options vfio enable_unsafe_noiommu_mode=1" > /etc/modprobe.d/vfio-noiommu.conf
EOF
```

```
$ cat <<EOF >first-boot
driverctl set-override 0000:06:00.0 vfio-pci
driverctl set-override 0000:07:00.0 vfio-pci

mkdir /mnt/huge
mount /mnt/huge --source nodev -t hugetlbfs -o pagesize=1GB
EOF
```

8. **virt-customize** ツールを使用して、Image Builder ツールによって生成されたイメージをカスタマイズします。

```
$ virt-customize -a <UUID>.qcow2 --run=customize-vm --firstboot=first-boot --selinux-relabel
```

9. コンテナーディスクイメージのビルドに必要なすべてのコマンドを含む Dockerfile を作成するには、次のコマンドを入力します。

```
$ cat << EOF > Dockerfile
FROM scratch
COPY <uuid>-disk.qcow2 /disk/
EOF
```

ここでは、以下のようになります。

<uuid>-disk.qcow2
: カスタムイメージの名前を **qcow2** 形式で指定します。

10. 次のコマンドを実行し、コンテナーをビルドしてタグを追加します。

```
$ podman build . -t dpdk-rhel:latest
```

11. 次のコマンドを実行して、クラスターからアクセスできるレジストリーにコンテナーディスクイメージをプッシュします。

```
$ podman push dpdk-rhel:latest
```

12. DPDK チェックアップ config map の **spec.param.vmContainerDiskImage** 属性で、コンテナーディスクイメージへのリンクを指定します。

14.3.2.4. 関連情報

- 仮想マシンの複数ネットワークへの割り当て
- Intel NIC を使用した DPDK モードでの Virtual Function の使用
- SR-IOV と Node Tuning Operator を使用した DPDK ラインレートの実現
- Image Builder のインストール
- How to register and subscribe a RHEL system to the Red Hat Customer Portal using Red Hat Subscription Manager

14.3.3. 仮想リソースの Prometheus クエリー

OpenShift Virtualization は、vCPU、ネットワーク、ストレージ、ゲストメモリースワッピングなどのクラスターインフラストラクチャーリソースの消費を監視するために使用できるメトリックを提供します。メトリックを使用して、ライブマイグレーションのステータスを照会することもできます。

OpenShift Container Platform モニタリングダッシュボードを使用して仮想化メトリックをクエリーします。

14.3.3.1. 前提条件

- vCPU メトリックを使用するには、**schedstats=enable** カーネル引数を **MachineConfig** オブジェクトに適用する必要があります。このカーネル引数を使用すると、デバッグとパフォーマンスチューニングに使用されるスケジューラーの統計が有効になり、スケジューラーに小規模な負荷を追加できます。詳細は、ノードへのカーネル引数の追加 を参照してください。
- ゲストメモリースワップクエリーがデータを返すには、仮想ゲストでメモリースワップを有効にする必要があります。

14.3.3.2. メトリクスのクエリー

OpenShift Container Platform モニタリングダッシュボードでは、Prometheus のクエリー言語 (PromQL) クエリーを実行し、プロットに可視化されるメトリクスを検査できます。この機能により、クラスターの状態と、モニターしているユーザー定義のワークロードに関する情報が提供されます。

クラスター管理者は、すべての OpenShift Container Platform のコアプロジェクトおよびユーザー定義プロジェクトのメトリクスをクエリーできます。

開発者として、メトリクスのクエリー時にプロジェクト名を指定する必要があります。選択したプロジェクトのメトリクスを表示するには、必要な権限が必要です。

14.3.3.2.1. クラスター管理者としてのすべてのプロジェクトのメトリクスのクエリー

クラスター管理者またはすべてのプロジェクトの表示権限を持つユーザーとして、メトリクス UI ですべてのデフォルト OpenShift Container Platform およびユーザー定義プロジェクトのメトリクスにアクセスできます。

前提条件

- **cluster-admin** クラスターロールまたはすべてのプロジェクトの表示権限を持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブから、**Observe → Metrics** を選択します。
2. 1 つ以上のクエリーを追加するには、次のいずれかを実行します。

オプション	説明

オプション	説明
カスタムクエリーを作成します。	Prometheus Query Language (PromQL) クエリーを **Expression** フィールドに追加します。 PromQL 式を入力すると、オートコンプリートの提案がドロップダウンリストに表示されます。これらの提案には、関数、メトリクス、ラベル、および時間トークンが含まれます。キーボードの矢印を使用して提案された項目のいずれかを選択し、Enter を押して項目を式に追加できます。また、マウスポインターを推奨項目の上に移動して、その項目の簡単な説明を表示することもできます。
複数のクエリーを追加します。	**クエリーの追加** を選択します。
既存のクエリーを複製します。	オプションメニューを選択します  クエリーの横にある **Duplicate query** を選択します。
クエリーの実行を無効にします。	オプションメニューを選択します  クエリーの横にある **Disable query** を選択します。

3. 作成したクエリーを実行するには、**Run queries** を選択します。クエリーからのメトリクスはプロットで可視化されます。クエリーが無効な場合は、UI にエラーメッセージが表示されます。

注記

大量のデータで動作するクエリーは、時系列グラフの描画時にタイムアウトするか、ブラウザーをオーバーロードする可能性があります。これを回避するには、**Hide graph** を選択し、メトリクステーブルのみを使用してクエリーを調整します。次に、使用できるクエリーを確認した後に、グラフを描画できるようにプロットを有効にします。

注記

デフォルトでは、クエリーテーブルに、すべてのメトリクスとその現在の値をリスト表示する拡張ビューが表示されます。˅ を選択すると、クエリーの拡張ビューを最小にすることができます。

4. オプション: ページ URL には、実行したクエリーが含まれます。このクエリーのセットを再度使用できるようにするには、この URL を保存します。

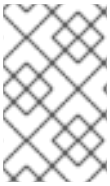
5. 視覚化されたメトリクスを調べます。最初に、有効な全クエリーの全メトリクスがプロットに表示されます。次のいずれかを実行して、表示するメトリクスを選択できます。

オプション	説明
クエリーからすべてのメトリクスを非表示にします。	オプションメニューをクリックします クエリーを選択し、**Hide all series** をクリックします。
特定のメトリクスを非表示にします。	クエリーテーブルに移動し、メトリクス名の近くにある色付きの四角形をクリックします。
プロットを拡大し、時間範囲を変更します。	次のいずれかになります。 ● プロットを水平にクリックし、ドラッグして、時間範囲を視覚的に選択します。 ● 左上隅のメニューを使用して、時間範囲を選択します。
時間範囲をリセットします。	**Reset zoom** を選択します。
特定の時点でのすべてのクエリーの出力を表示します。	その時点でプロット上にマウスカーソルを置きます。クエリーの出力はポップアップに表示されます。
プロットを非表示にします。	**Hide graph** を選択します。

14.3.3.2.2. 開発者が行うユーザー定義プロジェクトのメトリクスのクエリー

ユーザー定義のプロジェクトのメトリクスには、開発者またはプロジェクトの表示権限を持つユーザーとしてアクセスできます。

Developer パースペクティブには、選択したプロジェクトの事前に定義された CPU、メモリー、帯域幅、およびネットワークパケットのクエリーが含まれます。また、プロジェクトの CPU、メモリー、帯域幅、ネットワークパケット、およびアプリケーションメトリクスについてカスタム Prometheus Query Language (PromQL) クエリーを実行することもできます。



注記

開発者は **Developer** パースペクティブのみを使用でき、**Administrator** パースペクティブは使用できません。開発者は、1 度に 1 つのプロジェクトのメトリクスのみをクエリーできます。

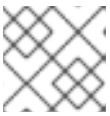
前提条件

- 開発者として、またはメトリクスで表示しているプロジェクトの表示権限を持つユーザーとしてクラスターへのアクセスがある。
- ユーザー定義プロジェクトのモニタリングが有効化されている。
- ユーザー定義プロジェクトにサービスをデプロイしている。

- サービスのモニター方法を定義するために、サービスの **ServiceMonitor** カスタムリソース定義 (CRD) を作成している。

手順

1. OpenShift Container Platform Web コンソールの **Developer** パースペクティブから、**Observe → Metrics** を選択します。
2. **Project:** 一覧でメトリクスで表示するプロジェクトを選択します。
3. **Select query** 一覧からクエリーを選択するか、**Show PromQL** を選択して、選択したクエリーに基づいてカスタム PromQL クエリーを作成します。クエリーからのメトリクスはプロットで可視化されます。



注記

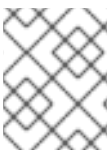
Developer パースペクティブでは、1 度に 1 つのクエリーのみを実行できます。

4. 次のいずれかを実行して、視覚化されたメトリクスを調べます。

オプション	説明
プロットを拡大し、時間範囲を変更します。	次のいずれかになります。 ● プロットを水平にクリックし、ドラッグして、時間範囲を視覚的に選択します。 ● 左上隅のメニューを使用して、時間範囲を選択します。
時間範囲をリセットします。	**Reset zoom** を選択します。
特定の時点でのすべてのクエリーの出力を表示します。	その時点でプロット上にマウスカーソルを置きます。クエリーの出力はポップアップに表示されます。

14.3.3.3. 仮想化メトリクス

以下のメトリクスの記述には、Prometheus Query Language (PromQL) クエリーのサンプルが含まれます。これらのメトリクスは API ではなく、バージョン間で変更される可能性があります。



注記

以下の例では、期間を指定する **topk** クエリーを使用します。その期間中に仮想マシンが削除された場合でも、クエリーの出力に依然として表示されます。

14.3.3.3.1. vCPU メトリック

以下のクエリーは、入出力 I/O) を待機している仮想マシンを特定します。

kubevirt_vmi_vcpu_wait_seconds

仮想マシンの vCPU の待機時間 (秒単位) を返します。タイプ: カウンター。

'0' より大きい値は、仮想 CPU は実行する用意ができているが、ホストスケジューラーがこれをまだ実行できないことを意味します。実行できない場合には I/O に問題があることを示しています。

注記

vCPU メトリックをクエリーするには、最初に **schedstats=enable** カーネル引数を **MachineConfig** オブジェクトに適用する必要があります。このカーネル引数を使用すると、デバッグとパフォーマンスチューニングに使用されるスケジューラーの統計が有効になり、スケジューラーに小規模な負荷を追加できます。

vCPU 待機時間クエリーの例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_vcpu_wait_seconds[6m]))) > 0 1
```

1 このクエリーは、6 分間の任意の全タイミングで I/O を待機する上位 3 の仮想マシンを返します。

14.3.3.3.2. ネットワークメトリック

以下のクエリーは、ネットワークを飽和状態にしている仮想マシンを特定できます。

kubevirt_vmi_network_receive_bytes_total

仮想マシンのネットワークで受信したトラフィックの合計量 (バイト単位) を返します。タイプ: カウンター。

kubevirt_vmi_network_transmit_bytes_total

仮想マシンのネットワーク上で送信されるトラフィックの合計量 (バイト単位) を返します。タイプ: カウンター。

ネットワークトラフィッククエリーの例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_network_receive_bytes_total[6m])) + sum by (name, namespace) (rate(kubevirt_vmi_network_transmit_bytes_total[6m]))) > 0 1
```

1 このクエリーは、6 分間の任意のタイミングで最大のネットワークトラフィックを送信する上位 3 の仮想マシンを返します。

14.3.3.3.3. ストレージメトリクス

14.3.3.3.3.1. ストレージ関連のトラフィック

以下のクエリーは、大量のデータを書き込んでいる仮想マシンを特定できます。

kubevirt_vmi_storage_read_traffic_bytes_total

仮想マシンのストレージ関連トラフィックの合計量 (バイト単位) を返します。タイプ: カウンター。

kubevirt_vmi_storage_write_traffic_bytes_total

仮想マシンのストレージ関連トラフィックのストレージ書き込みの合計量 (バイト単位) を返します。タイプ: カウンター。

ストレージ関連のトラフィッククエリーの例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_storage_read_traffic_bytes_total[6m])) + sum by (name, namespace) (rate(kubevirt_vmi_storage_write_traffic_bytes_total[6m]))) > 0 1
```

1 上記のクエリーは、6 分間の任意のタイミングで最も大きなストレージトラフィックを送信する上位 3 の仮想マシンを返します。

14.3.3.3.3.2. ストレージスナップショットデータ

kubevirt_vmsnapshot_disks_restored_from_source_total

ソース仮想マシンから復元された仮想マシンディスクの総数を返します。タイプ: ゲージ。

kubevirt_vmsnapshot_disks_restored_from_source_bytes

ソース仮想マシンから復元された容量をバイト単位で返します。タイプ: ゲージ。

ストレージスナップショットデータクエリーの例

```
kubevirt_vmsnapshot_disks_restored_from_source_total{vm_name="simple-vm", vm_namespace="default"} 1
```

1 このクエリーは、ソース仮想マシンから復元された仮想マシンディスクの総数を返します。

```
kubevirt_vmsnapshot_disks_restored_from_source_bytes{vm_name="simple-vm", vm_namespace="default"} 1
```

1 このクエリーは、ソース仮想マシンから復元された容量をバイト単位で返します。

14.3.3.3.3.3. I/O パフォーマンス

以下のクエリーで、ストレージデバイスの I/O パフォーマンスを判別できます。

kubevirt_vmi_storage_iops_read_total

仮想マシンが実行している 1 秒あたりの書き込み I/O 操作の量を返します。タイプ: カウンター。

kubevirt_vmi_storage_iops_write_total

仮想マシンが実行している 1 秒あたりの読み取り I/O 操作の量を返します。タイプ: カウンター。

I/O パフォーマンスクエリーの例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_storage_iops_read_total[6m])) + sum by (name, namespace) (rate(kubevirt_vmi_storage_iops_write_total[6m]))) > 0 1
```

1 上記のクエリーは、6 分間の任意のタイミングで最も大きな I/O 操作を実行している上位 3 の仮想マシンを返します。

14.3.3.3.4. ゲストメモリーのスワップメトリクス

以下のクエリーにより、メモリースワップを最も多く実行しているスワップ対応ゲストを特定できます。

kubevirt_vmi_memory_swap_in_traffic_bytes_total

仮想ゲストがスワップされているメモリーの合計量 (バイト単位) を返します。タイプ: ゲージ。

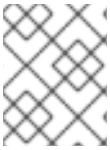
kubevirt_vmi_memory_swap_out_traffic_bytes_total

仮想ゲストがスワップアウトされているメモリーの合計量 (バイト単位) を返します。タイプ: ゲージ。

メモリースワップクエリーの例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_memory_swap_in_traffic_bytes_total[6m])) +
sum by (name, namespace) (rate(kubevirt_vmi_memory_swap_out_traffic_bytes_total[6m]))) > 0 (1)
```

(1) 上記のクエリーは、6 分間の任意のタイミングでゲストが最も大きなメモリースワップを実行している上位 3 の仮想マシンを返します。



注記

メモリースワップは、仮想マシンがメモリー不足の状態にあることを示します。仮想マシンのメモリー割り当てを増やすと、この問題を軽減できます。

14.3.3.3.5. ライブマイグレーションのメトリクス

次のメトリックをクエリーして、ライブマイグレーションのステータスを表示できます。

kubevirt_migrate_vmi_data_processed_bytes

新しい仮想マシン (VM) に移行されたゲストオペレーティングシステムデータの量。タイプ: ゲージ。

kubevirt_migrate_vmi_data_remaining_bytes

移行されていないゲストオペレーティングシステムデータの量。タイプ: ゲージ。

kubevirt_migrate_vmi_dirty_memory_rate_bytes

ゲスト OS でメモリーがダーティーになる速度。ダーティメモリーとは、変更されたがまだディスクに書き込まれていないデータです。タイプ: ゲージ。

kubevirt_migrate_vmi_pending_count

保留中の移行の数。タイプ: ゲージ。

kubevirt_migrate_vmi_scheduling_count

スケジュール移行の数。タイプ: ゲージ。

kubevirt_migrate_vmi_running_count

実行中の移行の数。タイプ: ゲージ。

kubevirt_migrate_vmi_succeeded

正常に完了した移行の数。タイプ: ゲージ。

kubevirt_migrate_vmi_failed

失敗した移行の数。タイプ: ゲージ。

14.3.3.4. 関連情報

- モニタリングの概要
- Querying Prometheus

- Prometheus クエリーの例

14.3.4. 仮想マシンのカスタムメトリクスの公開

OpenShift Container Platform には、コアプラットフォームコンポーネントのモニタリングを提供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含まれます。このモニタリングスタックは、Prometheus モニタリングシステムをベースにしています。Prometheus は Time Series を使用するデータベースであり、メトリクスのルール評価エンジンです。

OpenShift Container Platform モニタリングスタックの使用のほかに、CLI を使用してユーザー定義プロジェクトのモニタリングを有効にし、**node-exporter** サービスで仮想マシン用に公開されるカスタムメトリックをクエリーできます。

14.3.4.1. ノードエクスポーターサービスの設定

node-exporter エージェントは、メトリクスを収集するクラスター内のすべての仮想マシンにデプロイされます。node-exporter エージェントをサービスとして設定し、仮想マシンに関連付けられた内部メトリクスおよびプロセスを公開します。

前提条件

- OpenShift Container Platform CLI (**oc**) がインストールされている。
- **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。
- **cluster-monitoring-config ConfigMap** オブジェクトを **openshift-monitoring** プロジェクトに作成する。
- **enableUserWorkload** を **true** に設定して、**user-workload-monitoring-config ConfigMap** オブジェクトを **openshift-user-workload-monitoring** プロジェクトに設定する。

手順

1. **Service** YAML ファイルを作成します。以下の例では、このファイルは **node-exporter-service.yaml** という名前です。

```
kind: Service
apiVersion: v1
metadata:
  name: node-exporter-service 1
  namespace: dynamation 2
  labels:
    servicetype: metrics 3
spec:
  ports:
    - name: exmet 4
      protocol: TCP
      port: 9100 5
      targetPort: 9100 6
  type: ClusterIP
  selector:
    monitor: metrics 7
```

1 仮想マシンからメトリクスを公開する node-exporter サービス。

2 サービスが作成される namespace。

3 サービスのラベル。**ServiceMonitor** はこのラベルを使用してこのサービスを照会します。

4 **ClusterIP** サービスのポート 9100 でメトリクスを公開するポートに指定された名前。

5 リクエストをリッスンするために **node-exporter-service** によって使用されるターゲットポート。

6 **monitor** ラベルが設定された仮想マシンの TCP ポート番号。

7 仮想マシンの Pod を照会するために使用されるラベル。この例では、ラベル **monitor** のある仮想マシンの Pod と、**metrics** の値がマッチします。

2. node-exporter サービスを作成します。

```
$ oc create -f node-exporter-service.yaml
```

14.3.4.2. ノードエクスポーターサービスが設定された仮想マシンの設定

node-exporter ファイルを仮想マシンにダウンロードします。次に、仮想マシンの起動時に node-exporter サービスを実行する **systemd** サービスを作成します。

前提条件

- コンポーネントの Pod は **openshift-user-workload-monitoring** プロジェクトで実行されます。
- このユーザー定義プロジェクトをモニターする必要のあるユーザーに **monitoring-edit** ロールを付与します。

手順

1. 仮想マシンにログインします。
2. **node-exporter** ファイルのバージョンに適用されるディレクトリーパスを使用して、**node-exporter** ファイルを仮想マシンにダウンロードします。

```
$ wget
https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporter-
1.3.1.linux-amd64.tar.gz
```

3. 実行ファイルを展開して、**/usr/bin** ディレクトリーに配置します。

```
$ sudo tar xvf node_exporter-1.3.1.linux-amd64.tar.gz \
    --directory /usr/bin --strip 1 "*/node_exporter"
```

4. ディレクトリーのパス**/etc/systemd/system**に **node_exporter.service** ファイルを作成します。この **systemd** サービスファイルは、仮想マシンの再起動時に node-exporter サービスを実行します。

```
[Unit]
Description=Prometheus Metrics Exporter
```

```
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
User=root
ExecStart=/usr/bin/node_exporter

[Install]
WantedBy=multi-user.target
```

5. **systemd** サービスを有効にし、起動します。

```
$ sudo systemctl enable node_exporter.service
$ sudo systemctl start node_exporter.service
```

検証

- node-exporter エージェントが仮想マシンからのメトリクスを報告していることを確認します。

```
$ curl http://localhost:9100/metrics
```

出力例

```
go_gc_duration_seconds{quantile="0"} 1.5244e-05
go_gc_duration_seconds{quantile="0.25"} 3.0449e-05
go_gc_duration_seconds{quantile="0.5"} 3.7913e-05
```

14.3.4.3. 仮想マシンのカスタムモニタリングラベルの作成

単一サービスから複数の仮想マシンに対するクエリーを有効にするには、仮想マシンの YAML ファイルにカスタムラベルを追加します。

前提条件

- OpenShift Container Platform CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- 仮想マシンを停止および再起動するための Web コンソールへのアクセス権限がある。

手順

1. 仮想マシン設定ファイルの **template** spec を編集します。この例では、ラベル **monitor** の値が **metrics** になります。

```
spec:
  template:
    metadata:
```

```
    labels:
      monitor: metrics
```

2. 仮想マシンを停止して再起動し、**monitor** ラベルに指定されたラベル名を持つ新しい Pod を作成します。

14.3.4.3.1. メトリクスを取得するための node-exporter サービスのクエリー

仮想マシンのメトリクスは、**/metrics** の正規名の下に HTTP サービスエンドポイント経由で公開されます。メトリクスのクエリー時に、Prometheus は仮想マシンによって公開されるメトリクスエンドポイントからメトリクスを直接収集し、これらのメトリクスを確認用に表示します。

前提条件

- **cluster-admin** 権限を持つユーザーまたは **monitoring-edit** ロールを持つユーザーとしてクラスターにアクセスできる。
- node-exporter サービスを設定して、ユーザー定義プロジェクトのモニタリングを有効にしている。

手順

1. サービスの namespace を指定して、HTTP サービスエンドポイントを取得します。

```
$ oc get service -n <namespace> <node-exporter-service>
```

2. node-exporter サービスの利用可能なすべてのメトリクスを一覧表示するには、**metrics** リソースをクエリーします。

```
$ curl http://<172.30.226.162:9100>/metrics | grep -vE "^#|^$"
```

出力例

```
node_arp_entries{device="eth0"} 1
node_boot_time_seconds 1.643153218e+09
node_context_switches_total 4.4938158e+07
node_cooling_device_cur_state{name="0",type="Processor"} 0
node_cooling_device_max_state{name="0",type="Processor"} 0
node_cpu_guest_seconds_total{cpu="0",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="0",mode="user"} 0
node_cpu_seconds_total{cpu="0",mode="idle"} 1.10586485e+06
node_cpu_seconds_total{cpu="0",mode="iowait"} 37.61
node_cpu_seconds_total{cpu="0",mode="irq"} 233.91
node_cpu_seconds_total{cpu="0",mode="nice"} 551.47
node_cpu_seconds_total{cpu="0",mode="softirq"} 87.3
node_cpu_seconds_total{cpu="0",mode="steal"} 86.12
node_cpu_seconds_total{cpu="0",mode="system"} 464.15
node_cpu_seconds_total{cpu="0",mode="user"} 1075.2
node_disk_discard_time_seconds_total{device="vda"} 0
node_disk_discard_time_seconds_total{device="vdb"} 0
node_disk_discarded_sectors_total{device="vda"} 0
node_disk_discarded_sectors_total{device="vdb"} 0
node_disk_discards_completed_total{device="vda"} 0
node_disk_discards_completed_total{device="vdb"} 0
```

```
node_disk_discards_merged_total{device="vda"} 0
node_disk_discards_merged_total{device="vdb"} 0
node_disk_info{device="vda",major="252",minor="0"} 1
node_disk_info{device="vdb",major="252",minor="16"} 1
node_disk_io_now{device="vda"} 0
node_disk_io_now{device="vdb"} 0
node_disk_io_time_seconds_total{device="vda"} 174
node_disk_io_time_seconds_total{device="vdb"} 0.054
node_disk_io_time_weighted_seconds_total{device="vda"} 259.79200000000003
node_disk_io_time_weighted_seconds_total{device="vdb"} 0.039
node_disk_read_bytes_total{device="vda"} 3.71867136e+08
node_disk_read_bytes_total{device="vdb"} 366592
node_disk_read_time_seconds_total{device="vda"} 19.128
node_disk_read_time_seconds_total{device="vdb"} 0.039
node_disk_reads_completed_total{device="vda"} 5619
node_disk_reads_completed_total{device="vdb"} 96
node_disk_reads_merged_total{device="vda"} 5
node_disk_reads_merged_total{device="vdb"} 0
node_disk_write_time_seconds_total{device="vda"} 240.66400000000002
node_disk_write_time_seconds_total{device="vdb"} 0
node_disk_writes_completed_total{device="vda"} 71584
node_disk_writes_completed_total{device="vdb"} 0
node_disk_writes_merged_total{device="vda"} 19761
node_disk_writes_merged_total{device="vdb"} 0
node_disk_written_bytes_total{device="vda"} 2.007924224e+09
node_disk_written_bytes_total{device="vdb"} 0
```

14.3.4.4. ノードエクスポーターサービスの ServiceMonitor リソースの作成

Prometheus クライアントライブラリーを使用し、/**metrics** エンドポイントからメトリクスを収集して、node-exporter サービスが公開するメトリクスにアクセスし、表示できます。**ServiceMonitor** カスタムリソース定義 (CRD) を使用して、ノードエクスポーターサービスをモニターします。

前提条件

- **cluster-admin** 権限を持つユーザーまたは **monitoring-edit** ロールを持つユーザーとしてクラスターにアクセスできる。
- node-exporter サービスを設定して、ユーザー定義プロジェクトのモニタリングを有効にしている。

手順

1. **ServiceMonitor** リソース設定の YAML ファイルを作成します。この例では、サービスモニターはラベル **metrics** が指定されたサービスとマッチし、30 秒ごとに **exmet** ポートをクエリーします。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: node-exporter-metrics-monitor
  name: node-exporter-metrics-monitor (1)
  namespace: dynamation (2)
spec:
```

```
  endpoints:
  - interval: 30s 3
    port: exmet 4
    scheme: http
  selector:
    matchLabels:
      servicetype: metrics
```

1. **ServiceMonitor** の名前。
2. **ServiceMonitor** が作成される namespace。
3. ポートをクエリーする間隔。
4. 30 秒ごとにクエリーされるポートの名前

2. node-exporter サービスの **ServiceMonitor** 設定を作成します。

```
$ oc create -f node-exporter-metrics-monitor.yaml
```

14.3.4.4.1. クラスター外のノードエクスポーターサービスへのアクセス

クラスター外の node-exporter サービスにアクセスし、公開されるメトリクスを表示できます。

前提条件

- **cluster-admin** 権限を持つユーザーまたは **monitoring-edit** ロールを持つユーザーとしてクラスターにアクセスできる。
- node-exporter サービスを設定して、ユーザー定義プロジェクトのモニタリングを有効にしている。

手順

1. node-exporter サービスを公開します。

```
$ oc expose service -n <namespace> <node_exporter_service_name>
```

2. ルートの FQDN(完全修飾ドメイン名) を取得します。

```
$ oc get route -o=custom-columns=NAME:.metadata.name,DNS:.spec.host
```

出力例

```
NAME                 DNS
node-exporter-service   node-exporter-service-dynamation.apps.cluster.example.org
```

3. **curl** コマンドを使用して、node-exporter サービスのメトリクスを表示します。

```
$ curl -s http://node-exporter-service-dynamation.apps.cluster.example.org/metrics
```

出力例

```
go_gc_duration_seconds{quantile="0"} 1.5382e-05
go_gc_duration_seconds{quantile="0.25"} 3.1163e-05
go_gc_duration_seconds{quantile="0.5"} 3.8546e-05
go_gc_duration_seconds{quantile="0.75"} 4.9139e-05
go_gc_duration_seconds{quantile="1"} 0.000189423
```

14.3.4.5. 関連情報

- モニタリングスタックの設定
- ユーザー定義プロジェクトのモニタリングの有効化
- メトリックの管理
- モニタリングダッシュボードの確認
- ヘルスチェックの使用によるアプリケーションの正常性の監視
- 設定マップの作成および使用
- 仮想マシンの状態の制御

14.3.5. 仮想マシンのヘルスチェック

VirtualMachine リソースで readiness プローブと liveness プローブを定義することにより、仮想マシン (VM) のヘルスチェックを設定できます。

14.3.5.1. readiness および liveness プローブについて

readiness プローブと liveness プローブを使用して、異常な仮想マシン (VM) を検出および処理します。VM の仕様に 1 つ以上のプローブを含めて、準備ができていない VM にトラフィックが到達しないようにし、VM が応答しなくなったときに新しい VM が作成されるようにすることができます。

readiness プローブ は、VM がサービス要求を受け入れることができるかどうかを判断します。プローブが失敗すると、VM は、準備状態になるまで、利用可能なエンドポイントのリストから削除されます。

liveness プローブ は、VM が応答しているかどうかを判断します。プローブが失敗すると、VM は削除され、応答性を復元するために、新しい VM が作成されます。

VirtualMachine オブジェクトの **spec.readinessProbe** フィールドと **spec.livenessProbe** フィールドを設定することで、readiness プローブと liveness プローブを設定できます。これらのフィールドは、以下のテストをサポートします。

HTTP GET

プローブは、Web フックを使用して VM の正常性を判断します。このテストは、HTTP の応答コードが 200 から 399 までの値の場合に正常と見なされます。完全に初期化されている場合に、HTTP ステータスコードを返すアプリケーションで HTTP GET テストを使用できます。

TCP ソケット

プローブは、VM へのソケットを開こうとします。プローブが接続を確立できる場合のみ、VM は正常であると見なされます。TCP ソケットテストは、初期化が完了するまでリスニングを開始しないアプリケーションで使用できます。

ゲストエージェントの ping

プローブは、**guest-ping** コマンドを使用して、QEMU ゲストエージェントが仮想マシンで実行されているかどうかを判断します。

14.3.5.1.1. HTTP readiness プローブの定義

仮想マシン (VM) 設定の **spec.readinessProbe.httpGet** フィールドを設定して、HTTP readiness プローブを定義します。

手順

1. VM 設定ファイルに readiness プローブの詳細を含めます。

HTTP GET テストを使用した readiness プローブの例

```
# ...
spec:
  readinessProbe:
    httpGet: 1
      port: 1500 2
      path: /healthz 3
      httpHeaders:
      - name: Custom-Header
        value: Awesome
    initialDelaySeconds: 120 4
    periodSeconds: 20 5
    timeoutSeconds: 10 6
    failureThreshold: 3 7
    successThreshold: 3 8
# ...
```

1. VM に接続するために実行する HTTP GET 要求。
2. プローブがクエリーする VM のポート。上記の例では、プローブはポート 1500 をクエリーします。
3. HTTP サーバーでアクセスするパス。上記の例では、サーバーの /healthz パスのハンドラーが成功コードを返した場合、VM は正常であると見なされます。ハンドラーが失敗コードを返した場合、VM は使用可能なエンドポイントのリストから削除されます。
4. VM が起動してから準備プローブが開始されるまでの時間 (秒単位)。
5. プローブの実行間の遅延 (秒単位)。デフォルトの遅延は 10 秒です。この値は **timeoutSeconds** よりも大きくなければなりません。
6. プローブがタイムアウトになり、VM が失敗したと見なされるまでの非アクティブな秒数。デフォルト値は 1 です。この値は **periodSeconds** 未満である必要があります。
7. プローブが失敗できる回数。デフォルトは 3 です。指定された試行回数になると、Pod には **Unready** というマークが付けられます。
8. 成功とみなされるまでにプローブが失敗後に成功を報告する必要のある回数。デフォルトでは 1 回です。

2. 次のコマンドを実行して VM を作成します。

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

14.3.5.1.2. TCP readiness プローブの定義

仮想マシン (VM) 設定の **spec.readinessProbe.tcpSocket** フィールドを設定して、TCP readiness プローブを定義します。

手順

1. TCP readiness プローブの詳細を VM 設定ファイルに追加します。

TCP ソケットテストを含む readiness プローブの例

```
# ...
spec:
  readinessProbe:
    initialDelaySeconds: 120 1
    periodSeconds: 20 2
    tcpSocket: 3
      port: 1500 4
    timeoutSeconds: 10 5
# ...
```

1. VM が起動してから準備プローブが開始されるまでの時間 (秒単位)。
2. プローブの実行間の遅延 (秒単位)。デフォルトの遅延は 10 秒です。この値は **timeoutSeconds** よりも大きくなければなりません。
3. 実行する TCP アクション。
4. プローブがクエリーする VM のポート。
5. プローブがタイムアウトになり、VM が失敗したと見なされるまでの非アクティブな秒数。デフォルト値は 1 です。この値は **periodSeconds** 未満である必要があります。

2. 次のコマンドを実行して VM を作成します。

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

14.3.5.1.3. HTTP liveness プローブの定義

仮想マシン (VM) 設定の **spec.livenessProbe.httpGet** フィールドを設定して、HTTP liveness プローブを定義します。readiness プローブと同様に、liveness プローブの HTTP および TCP テストの両方を定義できます。この手順では、HTTP GET テストを使用して liveness プローブのサンプルを設定します。

手順

1. VM 設定ファイルに HTTP liveness プローブの詳細を含めます。

HTTP GET テストを使用した liveness プローブの例

```
# ...
```

```
spec:
  livenessProbe:
    initialDelaySeconds: 120 1
    periodSeconds: 20 2
    httpGet: 3
      port: 1500 4
      path: /healthz 5
      httpHeaders:
      - name: Custom-Header
        value: Awesome
    timeoutSeconds: 10 6
# ...
```

1. VM が起動してから liveness プローブが開始されるまでの時間 (秒単位)。
2. プローブの実行間の遅延 (秒単位)。デフォルトの遅延は 10 秒です。この値は **timeoutSeconds** よりも大きくなければなりません。
3. VM に接続するために実行する HTTP GET 要求。
4. プローブがクエリーする VM のポート。上記の例では、プローブはポート 1500 をクエリーします。VM は、cloud-init を介してポート 1500 に最小限の HTTP サーバーをインストールして実行します。
5. HTTP サーバーでアクセスするパス。上記の例では、サーバーの **/healthz** パスのハンドラーが成功コードを返した場合、VM は正常であると見なされます。ハンドラーが失敗コードを返した場合、VM は削除され、新しい VM が作成されます。
6. プローブがタイムアウトになり、VM が失敗したと見なされるまでの非アクティブな秒数。デフォルト値は 1 です。この値は **periodSeconds** 未満である必要があります。

2. 次のコマンドを実行して VM を作成します。

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

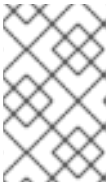
14.3.5.2. ウォッチドッグの定義

次の手順を実行して、ゲスト OS の正常性を監視するウォッチドッグを定義できます。

1. 仮想マシン (VM) のウォッチドッグデバイスを設定します。
2. ゲストにウォッチドッグエージェントをインストールします。

ウォッチドッグデバイスはエージェントを監視し、ゲストオペレーティングシステムが応答しない場合、次のいずれかのアクションを実行します。

- **poweroff**: VM の電源がすぐにオフになります。**spec.running** が **true** に設定されているか、**spec.runStrategy** が **manual** に設定されていない場合、VM は再起動します。
- **reset**: VM はその場で再起動し、ゲストオペレーティングシステムは反応できません。



注記

再起動時間が原因で liveness プローブがタイムアウトする場合があります。クラスターレベルの保護が liveness プローブの失敗を検出すると、VM が強制的に再スケジュールされ、再起動時間が長くなる可能性があります。

- **shutdown**: すべてのサービスを停止することで、VM の電源を正常にオフにします。

注記

ウォッチドッグは、Windows VM では使用できません。

14.3.5.2.1. 仮想マシンのウォッチドッグデバイスの設定

仮想マシン (VM) のウォッチドッグデバイスを設定するとします。

前提条件

- VM には、**i6300esb** ウォッチドッグデバイスのカーネルサポートが必要です。Red Hat Enterprise Linux(RHEL) イメージが、**i6300esb** をサポートしている。

手順

1. 次の内容で **YAML** ファイルを作成します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm2-rhel84-watchdog
  name: <vm-name>
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm2-rhel84-watchdog
    spec:
      domain:
        devices:
          watchdog:
            name: <watchdog>
            i6300esb:
              action: "poweroff" (1)
# ...
```

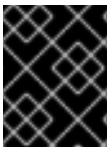
(1) **poweroff**、**reset**、または **shutdown** を指定します。

上記の例では、電源オフアクションを使用して、RHEL8 VM で **i6300esb** ウォッチドッグデバイスを設定し、デバイスを **/dev/watchdog** として公開します。

このデバイスは、ウォッチドッグバイナリーで使用できるようになりました。

2. 以下のコマンドを実行して、YAML ファイルをクラスターに適用します。

```
$ oc apply -f <file_name>.yaml
```



重要

この手順は、ウォッチドッグ機能をテストするためにのみ提供されており、実稼働マシンでは実行しないでください。

1. 以下のコマンドを実行して、VM がウォッチドッグデバイスに接続されていることを確認します。

```
$ lspci | grep watchdog -i
```

2. 以下のコマンドのいずれかを実行して、ウォッチドッグがアクティブであることを確認します。

- カーネルパニックをトリガーします。

```
# echo c > /proc/sysrq-trigger
```

- ウォッチドッグサービスを停止します。

```
# pkill -9 watchdog
```

14.3.5.2.2. ゲストへのウォッチドッグエージェントのインストール

ゲストにウォッチドッグエージェントをインストールし、**watchdog** サービスを開始します。

手順

1. root ユーザーとして仮想マシンにログインします。
2. **watchdog** ドッグパッケージとその依存関係をインストールします。

```
# yum install watchdog
```

3. **/etc/watchdog.conf** ファイルの次の行のコメントを外し、変更を保存します。

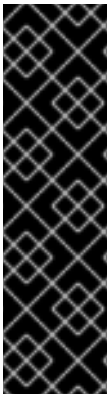
```
#watchdog-device = /dev/watchdog
```

4. 起動時に **watchdog** サービスを開始できるようにします。

```
# systemctl enable --now watchdog.service
```

14.3.5.3. ゲストエージェントの ping プローブの定義

仮想マシン (VM) 設定の **spec.readinessProbe.guestAgentPing** フィールドを設定して、ゲストエージェント ping プローブを定義します。



重要

ゲストエージェント ping プローブは、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

前提条件

- 仮想マシンに QEMU ゲストエージェントをインストールして有効にする必要があります。

手順

1. VM 設定ファイルにゲストエージェント ping プローブの詳細を含めます。以下に例を示します。

ゲストエージェント ping プローブの例

```
# ...
spec:
  readinessProbe:
    guestAgentPing: {} (1)
    initialDelaySeconds: 120 (2)
    periodSeconds: 20 (3)
    timeoutSeconds: 10 (4)
    failureThreshold: 3 (5)
    successThreshold: 3 (6)
# ...
```

(1) VM に接続するためのゲストエージェント ping プローブ。

(2) オプション: VM が起動してからゲストエージェントプローブが開始されるまでの時間 (秒単位)。

(3) オプション: プローブを実行する間の遅延 (秒単位)。デフォルトの遅延は 10 秒です。この値は **timeoutSeconds** よりも大きくなければなりません。

(4) オプション: プローブがタイムアウトになり、VM が失敗したと見なされるまでの非アクティブの秒数。デフォルト値は 1 です。この値は **periodSeconds** 未満である必要があります。

(5) オプション: プローブが失敗できる回数。デフォルトは 3 です。指定された試行回数になると、Pod には **Unready** というマークが付けられます。

(6) オプション: 成功とみなされるまでにプローブが失敗後に成功を報告する必要のある回数。デフォルトでは 1 回です。

2. 次のコマンドを実行して VM を作成します。

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

14.3.5.4. 関連情報

- ヘルスチェックの使用によるアプリケーションの正常性の監視

14.4. トラブルシューティング

OpenShift Virtualization は、仮想マシンと仮想化コンポーネントのトラブルシューティングに使用するツールとログを提供します。

OpenShift Virtualization コンポーネントのトラブルシューティングは、Web コンソールで提供されるツール または **oc** CLI ツールを使用して実行できます。

14.4.1. イベント

OpenShift Container Platform イベント は重要なライフサイクル情報の記録であり、仮想マシン、namespace、リソース問題のモニタリングおよびトラブルシューティングに役立ちます。

- 仮想マシンイベント: Web コンソールで **VirtualMachine details** ページの Events タブに移動します。

  namespace イベント

  namespace イベントを表示するには、次のコマンドを実行します。

  ```
  $ oc get events -n <namespace>
  ```

  特定のイベントの詳細は、イベントのリスト を参照してください。

  リソースイベント

  リソースイベントを表示するには、次のコマンドを実行します。

  ```
  $ oc describe <resource> <resource_name>
  ```

14.4.2. ログ

トラブルシューティングのために、次のログを確認できます。

- 仮想マシン
- OpenShift Virtualization Pod
- OpenShift Virtualization ログの集約

14.4.2.1. Web コンソールを使用した仮想マシンログの表示

OpenShift Container Platform Web コンソールで仮想マシンのログを表示できます。

手順

1. **Virtualization → VirtualMachines** に移動します。

2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Details** タブで、Pod 名をクリックして **Pod details** ページを開きます。
4. **Logs** タブをクリックして、ログを表示します。

14.4.2.2. OpenShift Virtualization Pod のログ表示

oc CLI ツールを使用して、OpenShift Virtualization Pod のログを表示できます。

HyperConverged カスタムリソース (CR) を編集することで、ログの詳細レベルを設定できます。

14.4.2.2.1. CLI を使用した OpenShift Virtualization Pod ログの表示

oc CLI ツールを使用して、OpenShift Virtualization Pod のログを表示できます。

手順

1. 以下のコマンドを実行して、OpenShift Virtualization の namespace 内の Pod のリストを表示します。

```
$ oc get pods -n openshift-cnv
```

例14.3 出力例

```
NAME                               READY   STATUS    RESTARTS   AGE
disks-images-provider-7gqbc        1/1     Running   0          32m
disks-images-provider-vg4kx        1/1     Running   0          32m
virt-api-57fcc4497b-7qfmc          1/1     Running   0          31m
virt-api-57fcc4497b-tx9nc          1/1     Running   0          31m
virt-controller-76c784655f-7fp6m   1/1     Running   0          30m
virt-controller-76c784655f-f4pbd   1/1     Running   0          30m
virt-handler-2m86x                 1/1     Running   0          30m
virt-handler-9qs6z                 1/1     Running   0          30m
virt-operator-7ccfdbf65f-q5snk     1/1     Running   0          32m
virt-operator-7ccfdbf65f-vllz8     1/1     Running   0          32m
```

2. Pod ログを表示するには、次のコマンドを実行します。

```
$ oc logs -n openshift-cnv <pod_name>
```

注記

Pod の起動に失敗した場合は、**--previous** オプションを使用して、最後の試行からのログを表示できます。

ログ出力をリアルタイムで監視するには、**-f** オプションを使用します。

例14.4 出力例

```
{"component":"virt-handler","level":"info","msg":"set verbosity to 2","pos":"virt-handler.go:453","timestamp":"2022-04-17T08:58:37.373695Z"}
```

{"component":"virt-handler","level":"info","msg":"set verbosity to 2","pos":"virt-handler.go:453","timestamp":"2022-04-17T08:58:37.373726Z"}
{"component":"virt-handler","level":"info","msg":"setting rate limiter to 5 QPS and 10 Burst","pos":"virt-handler.go:462","timestamp":"2022-04-17T08:58:37.373782Z"}
{"component":"virt-handler","level":"info","msg":"CPU features of a minimum baseline CPU model: map[apic:true clflush:true cmov:true cx16:true cx8:true de:true fpu:true fxsr:true lahf_lm:true lm:true mca:true mce:true mmx:true msr:true mtrr:true nx:true pae:true pat:true pge:true pni:true pse:true pse36:true sep:true sse:true sse2:true sse4.1:true ssse3:true syscall:true tsc:true]","pos":"cpu_plugin.go:96","timestamp":"2022-04-17T08:58:37.390221Z"}
{"component":"virt-handler","level":"warning","msg":"host model mode is expected to contain only one model","pos":"cpu_plugin.go:103","timestamp":"2022-04-17T08:58:37.390263Z"}
{"component":"virt-handler","level":"info","msg":"node-labeller is running","pos":"node_labeller.go:94","timestamp":"2022-04-17T08:58:37.391011Z"}

14.4.2.2.2. OpenShift Virtualization Pod ログの詳細レベルの設定

HyperConverged カスタムリソース (CR) を編集することで、OpenShift Virtualization Pod ログの詳細レベルを設定できます。

手順

1. 特定のコンポーネントのログの詳細度を設定するには、次のコマンドを実行して、デフォルトのテキストエディターで **HyperConverged** CR を開きます。

```
$ oc edit hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **spec.logVerbosityConfig** スタンザを編集して、1 つ以上のコンポーネントのログレベルを設定します。以下に例を示します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  logVerbosityConfig:
    kubevirt:
      virtAPI: 5 (1)
      virtController: 4
      virtHandler: 3
      virtLauncher: 2
      virtOperator: 6
```

(1) ログの詳細度の値は **1** ~ **9** の範囲の整数である必要があり、数値が大きいほど詳細なログであることを示します。この例では、**virtAPI** コンポーネントのログは、優先度が **5** 以上の場合に公開されます。

3. エディターを保存して終了し、変更を適用します。

14.4.2.2.3. 一般的なエラーメッセージ

以下のエラーメッセージが OpenShift Virtualization ログに表示される場合があります。

ErrImagePull または ImagePullBackOff

デプロイメント設定が正しくないか、参照されているイメージに問題があることを示します。

14.4.2.3. LokiStack を使用した OpenShift Virtualization 集約ログの表示

Web コンソールで LokiStack を使用すると、OpenShift Virtualization Pod およびコンテナーの集約ログを表示できます。

前提条件

- LokiStack をデプロイしている。

手順

1. Web コンソールで **Observe → Logs** に移動します。
2. ログタイプのリストから、**virt-launcher** Pod ログの場合は **application** を選択し、OpenShift Virtualization コントロールプレーン Pod およびコンテナーの場合は **infrastructure** を選択します。
3. **Show Query** をクリックしてクエリーフィールドを表示します。
4. クエリーフィールドに LogQL クエリーを入力し、**Run Query** をクリックしてフィルタリングされたログを表示します。

14.4.2.3.1. OpenShift Virtualization LogQL クエリー

Web コンソールの **Observe → Logs** ページで Loki Query Language (LogQL) クエリーを実行することで、OpenShift Virtualization コンポーネントの集約ログを表示およびフィルタリングできます。

デフォルトのログタイプは **infrastructure** です。**virt-launcher** のログタイプは **application** です。

オプション: 行フィルター式を使用して、文字列または正規表現の追加や除外を行えます。

注記

クエリーが多数のログに一致する場合、クエリーがタイムアウトになる可能性があります。

表14.4 OpenShift Virtualization LogQL クエリーの例

コンポーネント	LogQL クエリー
すべて	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"`

コンポーネント	LogQL クエリー
cdi-apiserver **cdi-deployment** **cdi-operator**	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="storage"
hco-operator	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="deployment"
kubemacpool	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="network"
virt-api **virt-controller** **virt-handler** **virt-operator**	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="compute"
ssp-operator	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="schedule"
Container	{log_type=~".+",kubernetes_container_name=~"<container>|<container>"} (1) |json|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"

(1) 1つ以上のコンテナーを縦線記号 (|) で区切って指定します。

コンポーネント	LogQL クエリー
virt-launcher	このクエリーを実行する前に、ログタイプのリストから **application** を選択する必要があります。 `{log_type=~".+", kubernetes_container_name="compute"}|json` `|!= "custom-ga-command"` (1) (1) **|!= "custom-ga-command"** は、**custom-ga-command** の文字列を含む libvirt ログを除外します。(BZ#2177684)

行フィルター式を使用して、文字列や正規表現を追加または除外するようにログ行をフィルタリングできます。

表14.5 行フィルター式

行フィルター式	説明
|= "<string>"	ログ行に文字列が含まれています
!= "<string>"	ログ行に文字列は含まれていません
|~ "<regex>"	ログ行に正規表現が含まれています
!~ "<regex>"	ログ行に正規表現は含まれていません

行フィルター式の例

```
{log_type=~".+"}|json
|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"
|= "error" != "timeout"
```

14.4.2.3.2. LokiStack および LogQL の関連情報

- ログストレージについて
- LokiStack のデプロイ
- Grafana ドキュメントの LogQL log queries

14.4.3. データボリュームのトラブルシューティング

DataVolume オブジェクトの **Conditions** および **Events** セクションを確認して、問題を分析および解決できます。

14.4.3.1. データボリュームの条件とイベントについて

コマンドによって生成された **Conditions** および **Events** セクションの出力を調べることで、データボリュームの問題を診断できます。

```
$ oc describe dv <DataVolume>
```

Conditions セクションには、次の **Types** が表示されます。

- **Bound**
- **running**
- **Ready**

Events セクションでは、以下の追加情報を提供します。

- イベントの **Type**
- ロギングの **Reason**
- イベントの **Source**
- 追加の診断情報が含まれる **Message**

oc describe からの出力には常に **Events** が含まれるとは限りません。

Status、**Reason**、または **Message** が変化すると、イベントが生成されます。状態およびイベントはどちらもデータボリュームの状態の変更に対応します。

たとえば、インポート操作中に URL のスペルを誤ると、インポートにより 404 メッセージが生成されます。メッセージの変更により、理由と共にイベントが生成されます。**Conditions** セクションの出力も更新されます。

14.4.3.2. データボリュームの状態とイベントの分析

describe コマンドで生成される **Conditions** セクションおよび **Events** セクションを検査することにより、永続ボリューム要求 (PVC) に関連してデータボリュームの状態を判別します。また、操作がアクティブに実行されているか、または完了しているかどうかを判断します。また、データボリュームのステータスに関する特定の詳細、およびどのように現在の状態になったかに関する情報を提供するメッセージを受信する可能性があります。

状態の組み合わせは多数あります。それぞれは一意のコンテキストで評価される必要があります。

各種の組み合わせの例を以下に示します。

- **Bound** - この例では、正常にバインドされた PVC が表示されます。
  Type は **Bound** であるため、**Status** は **True** になります。PVC がバインドされていない場合、**Status** は **False** になります。

  PVC がバインドされると、PVC がバインドされていることを示すイベントが生成されます。この場合、**Reason** は **Bound** で、**Status** は **True** です。**Message** はデータボリュームを所有する PVC を示します。

  Events セクションの **Message** では、PVC がバインドされている期間 (**Age**) およびどのリソース (**From**) によってバインドされているか、**datavolume-controller** に関する詳細が提供されます。

  出力例

```
Status:
  Conditions:
    Last Heart Beat Time:  2020-07-15T03:58:24Z
    Last Transition Time:  2020-07-15T03:58:24Z
    Message:               PVC win10-rootdisk Bound
    Reason:                Bound
    Status:                True
    Type:                  Bound
...
  Events:
    Type    Reason     Age    From                   Message
    ----    ------     ----   ----                   -------
    Normal  Bound      24s    datavolume-controller  PVC example-dv Bound
```

- **Running** - この場合、**Type** が **Running** で、**Status** が **False** であることに注意してください。これは、試行された操作が失敗する原因となったイベントが発生し、Status が **True** から **False** に変化したことを示しています。
ただし、**Reason** が **Completed** であり、**Message** フィールドには **Import Complete** が表示されることに注意してください。

  Events セクションには、**Reason** および **Message** に失敗した操作に関する追加のトラブルシューティング情報が含まれます。この例では、**Events** セクションの最初の **Warning** に一覧表示される **Message** に、**404** によって接続できないことが示唆されます。

  この情報から、インポート操作が実行されており、データボリュームにアクセスしようとしている他の操作に対して競合が生じることを想定できます。

  出力例

```
Status:
  Conditions:
    Last Heart Beat Time:  2020-07-15T04:31:39Z
    Last Transition Time:  2020-07-15T04:31:39Z
    Message:               Import Complete
    Reason:                Completed
    Status:                False
    Type:                  Running
...
  Events:
    Type    Reason      Age              From                   Message
    ----    ------      ----             ----                   -------
    Warning  Error      12s (x2 over 14s)  datavolume-controller  Unable to connect
    to http data source: expected status code 200, got 404. Status: 404 Not Found
```

- **Ready**: **Type** が **Ready** であり、**Status** が **True** の場合、以下の例のようにデータボリュームは使用可能な状態になります。データボリュームが使用可能な状態にない場合、**Status** は **False** になります。

  出力例

```
Status:
  Conditions:
    Last Heart Beat Time: 2020-07-15T04:31:39Z
```

```
Last Transition Time:  2020-07-15T04:31:39Z
Status:                True
Type:                  Ready
```

14.5. OPENSHIFT VIRTUALIZATION の RUNBOOK

OpenShift Virtualization Operator の runbook は、openshift/runbooks Git リポジトリーで管理されており、GitHub で表示できます。OpenShift Virtualization アラート をトリガーする問題を診断して解決するには、runbook の手順に従います。

OpenShift Virtualization のアラートは、Web コンソールの **Virtualization → Overview** タブに表示されます。

14.5.1. CDIDataImportCronOutdated

- **CDIDataImportCronOutdated** アラートの runbook を表示 します。

14.5.2. CDIDataVolumeUnusualRestartCount

- **CDIDataVolumeUnusualRestartCount** アラートの runbook を表示 します。

14.5.3. CDIDefaultStorageClassDegraded

- **CDIDefaultStorageClassDegraded** アラートの runbook を表示 します。

14.5.4. CDIMultipleDefaultVirtStorageClasses

- **CDIMultipleDefaultVirtStorageClasses** アラートの runbook を表示 します。

14.5.5. CDINoDefaultStorageClass

- **CDINoDefaultStorageClass** アラートの runbook を表示 します。

14.5.6. CDINotReady

- **CDINotReady** アラートの runbook を表示 します。

14.5.7. CDIOperatorDown

- **CDIOperatorDown** アラートの runbook を表示 します。

14.5.8. CDIStorageProfilesIncomplete

- **CDIStorageProfilesIncomplete** アラートの runbook を表示 します。

14.5.9. CnaoDown

- **CnaoDown** アラートの runbook を表示 します。

14.5.10. CnaoNMstateMigration

- **CnaoNMstateMigration** アラートの runbook を表示 します。

14.5.11. HCOInstallationIncomplete

- **HCOInstallationIncomplete** アラートの runbook を表示 します。

14.5.12. HPPNotReady

- **HPPNotReady** アラートの runbook を表示します。

14.5.13. HPPOperatorDown

- **HPPOperatorDown** アラートの runbook を表示 します。

14.5.14. HPPSharingPoolPathWithOS

- **HPPSharingPoolPathWithOS** アラートの runbook を表示 します。

14.5.15. KubemacpoolDown

- **KubemacpoolDown** アラートの runbook を表示 します。

14.5.16. KubeMacPoolDuplicateMacsFound

- **KubeMacPoolDuplicateMacsFound** アラートの runbook を表示 します。

14.5.17. KubeVirtComponentExceedsRequestedCPU

- **KubeVirtComponentExceedsRequestedCPU** アラートが 非推奨 になりました。

14.5.18. KubeVirtComponentExceedsRequestedMemory

- **KubeVirtComponentExceedsRequestedMemory** アラートは 非推奨 になりました。

14.5.19. KubeVirtCRModified

- **KubeVirtCRModified** アラートの runbook を表示 します。

14.5.20. KubeVirtDeprecatedAPIRequested

- **KubeVirtDeprecatedAPIRequested** アラートの runbook を表示 します。

14.5.21. KubeVirtNoAvailableNodesToRunVMs

- **KubeVirtNoAvailableNodesToRunVMs** アラートの runbook を表示 します。

14.5.22. KubevirtVmHighMemoryUsage

- **KubevirtVmHighMemoryUsage** アラートの runbook を表示 します。

14.5.23. KubeVirtVMIExcessiveMigrations

- **KubeVirtVMIExcessiveMigrations** アラートの runbook を表示 します。

14.5.24. LowKVMNodesCount

- **LowKVMNodesCount** アラートの runbook を表示 します。

14.5.25. LowReadyVirtControllersCount

- **LowReadyVirtControllersCount** アラートの runbook を表示 します。

14.5.26. LowReadyVirtOperatorsCount

- **LowReadyVirtOperatorsCount** アラートの runbook を表示 します。

14.5.27. LowVirtAPICount

- **LowVirtAPICount** アラートの runbook を表示 します。

14.5.28. LowVirtControllersCount

- **LowVirtControllersCount** アラートの runbook を表示 します。

14.5.29. LowVirtOperatorCount

- **LowVirtOperatorCount** アラートの runbook を表示 します。

14.5.30. NetworkAddonsConfigNotReady

- **NetworkAddonsConfigNotReady** アラートの runbook を表示 します。

14.5.31. NoLeadingVirtOperator

- **NoLeadingVirtOperator** アラートの runbook を表示 します。

14.5.32. NoReadyVirtController

- **NoReadyVirtController** アラートの runbook を表示 します。

14.5.33. NoReadyVirtOperator

- **NoReadyVirtOperator** アラートの runbook を表示 します。

14.5.34. OrphanedVirtualMachineInstances

- **OrphanedVirtualMachineInstances** アラートの runbook を表示 します。

14.5.35. OutdatedVirtualMachineInstanceWorkloads

- **OutdatedVirtualMachineInstanceWorkloads** アラートの runbook を表示 します。

14.5.36. SingleStackIPv6Unsupported

- **SingleStackIPv6Unsupported** アラートの runbook を表示 します。

14.5.37. SSPCommonTemplatesModificationReverted

- **SSPCommonTemplatesModificationReverted** アラートの runbook を表示 します。

14.5.38. SSPDown

- **SSPDown** アラートの runbook を表示 します。

14.5.39. SSPFailingToReconcile

- **SSPFailingToReconcile** アラートの runbook を表示します。

14.5.40. SSPHighRateRejectedVms

- **SSPHighRateRejectedVms** アラートの runbook を表示 します。

14.5.41. SSPTemplateValidatorDown

- **SSPTemplateValidatorDown** アラートの runbook を表示 します。

14.5.42. UnsupportedHCOModification

- **UnsupportedHCOModification** アラートの runbook を表示 します。

14.5.43. VirtAPIDown

- **VirtAPIDown** アラートの runbook を表示 します。

14.5.44. VirtApiRESTErrorsBurst

- **VirtApiRESTErrorsBurst** アラートの runbook を表示 します。

14.5.45. VirtApiRESTErrorsHigh

- **VirtApiRESTErrorsHigh** アラートの runbook を表示 します。

14.5.46. VirtControllerDown

- **VirtControllerDown** アラートの runbook を表示 します。

14.5.47. VirtControllerRESTErrorsBurst

- **VirtControllerRESTErrorsBurst** アラートの runbook を表示 します。

14.5.48. VirtControllerRESTErrorsHigh

- **VirtControllerRESTErrorsHigh** アラートの runbook を表示 します。

14.5.49. VirtHandlerDaemonSetRolloutFailing

- **VirtHandlerDaemonSetRolloutFailing** アラートの runbook を表示 します。

14.5.50. VirtHandlerRESTErrorsBurst

- **VirtHandlerRESTErrorsBurst** アラートの runbook を表示 します。

14.5.51. VirtHandlerRESTErrorsHigh

- **VirtHandlerRESTErrorsHigh** アラートの runbook を表示 します。

14.5.52. VirtOperatorDown

- **VirtOperatorDown** アラートの runbook を表示 します。

14.5.53. VirtOperatorRESTErrorsBurst

- **VirtOperatorRESTErrorsBurst** アラートの runbook を表示 します。

14.5.54. VirtOperatorRESTErrorsHigh

- **VirtOperatorRESTErrorsHigh** アラートの runbook を表示 します。

14.5.55. VirtualMachineCRCErrors

- **VirtualMachineCRCErrors** アラートの runbook は、アラートの名前が **VMStorageClassWarning** に変更されたため非推奨になりました。
    - **VMStorageClassWarning** アラートの runbook を表示 します。

14.5.56. VMCannotBeEvicted

- **VMCannotBeEvicted** アラートの runbook を表示 します。

14.5.57. VMStorageClassWarning

- **VMStorageClassWarning** アラートの runbook を表示 します。

第15章 バックアップおよび復元

15.1. OADP のインストールおよび設定

クラスター管理者は、OADP Operator をインストールして、OpenShift API for Data Protection (OADP) をインストールします。Operator は Velero 1.14 をインストールします。

バックアップストレージプロバイダーのデフォルトの **Secret** を作成し、Data Protection Application をインストールします。

15.1.1. OADP Operator のインストール

Operator Lifecycle Manager (OLM) を使用して、OpenShift Container Platform 4.13 に OpenShift API for Data Protection (OADP) オペレーターをインストールします。

OADP Operator は Velero 1.14 をインストールします。

前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** をクリックします。
2. **Filter by keyword** フィールドを使用して、**OADP Operator** を検索します。
3. **OADP Operator** を選択し、**Install** をクリックします。
4. **Install** をクリックして、**openshift-adp** プロジェクトに Operator をインストールします。
5. **Operators → Installed Operators** をクリックして、インストールを確認します。

15.1.2. バックアップおよびスナップショットの場所、ならびにそのシークレットについて

DataProtectionApplication カスタムリソース (CR) で、バックアップおよびスナップショットの場所、ならびにそのシークレットを指定します。

バックアップの場所
Multicloud Object Gateway、Red Hat Container Storage、Ceph RADOS Gateway、Ceph Object Gateway、{odf-full}、または MinIO などのバックアップの場所として AWS S3 互換オブジェクトストレージを指定します。

Velero は、オブジェクトストレージのアーカイブファイルとして、OpenShift Container Platform リソース、Kubernetes オブジェクト、および内部イメージをバックアップします。

スナップショットの場所
クラウドプロバイダーのネイティブスナップショット API を使用して永続ボリュームをバックアップする場合、クラウドプロバイダーをスナップショットの場所として指定する必要があります。

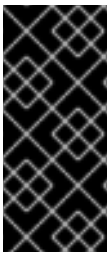
Container Storage Interface (CSI) スナップショットを使用する場合、CSI ドライバーを登録するために **VolumeSnapshotClass** CR を作成するため、スナップショットの場所を指定する必要はありません。

File System Backup (FSB) を使用する場合、FSB がオブジェクトストレージ上にファイルシステムをバックアップするため、スナップショットの場所を指定する必要はありません。

シークレット
バックアップとスナップショットの場所が同じ認証情報を使用する場合、またはスナップショットの場所が必要ない場合は、デフォルトの **Secret** を作成します。

バックアップとスナップショットの場所で異なる認証情報を使用する場合は、次の 2 つの secret オブジェクトを作成します。

- **DataProtectionApplication** CR で指定する、バックアップの場所用のカスタム **Secret**。
- **DataProtectionApplication** CR で参照されない、スナップショットの場所用のデフォルト **Secret**。



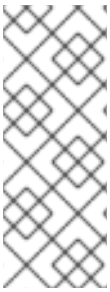
重要

Data Protection Application には、デフォルトの **Secret** が必要です。作成しないと、インストールは失敗します。

インストール中にバックアップまたはスナップショットの場所を指定したくない場合は、空の **credentials-velero** ファイルを使用してデフォルトの **Secret** を作成できます。

15.1.2.1. デフォルト Secret の作成

バックアップとスナップショットの場所が同じ認証情報を使用する場合、またはスナップショットの場所が必要ない場合は、デフォルトの **Secret** を作成します。



注記

DataProtectionApplication カスタムリソース (CR) にはデフォルトの **Secret** が必要です。作成しないと、インストールは失敗します。バックアップの場所の **Secret** の名前が指定されていない場合は、デフォルトの名前が使用されます。

インストール時にバックアップの場所の認証情報を使用しない場合は、空の **credentials-velero** ファイルを使用してデフォルト名前で **Secret** を作成できます。

前提条件

- オブジェクトストレージとクラウドストレージがある場合は、同じ認証情報を使用する必要があります。
- Velero のオブジェクトストレージを設定する必要があります。
- オブジェクトストレージ用の **credentials-velero** ファイルを適切な形式で作成する必要があります。

手順

- デフォルト名で **Secret** を作成します。

```
$ oc create secret generic cloud-credentials -n openshift-adp --from-file cloud=credentials-velero
```

Secret は、Data Protection Application をインストールするときに、**DataProtectionApplication** CR の **spec.backupLocations.credential** ブロックで参照されます。

15.1.3. Data Protection Application の設定

Velero リソースの割り当てを設定するか、自己署名 CA 証明書を有効にして、Data Protection Application を設定できます。

15.1.3.1. Velero の CPU とメモリーのリソース割り当てを設定

DataProtectionApplication カスタムリソース (CR) マニフェストを編集して、**Velero** Pod の CPU およびメモリーリソースの割り当てを設定します。

前提条件

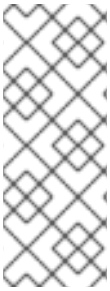
- OpenShift API for Data Protection (OADP) Operator がインストールされている必要があります。

手順

- 次の例のように、**DataProtectionApplication** CR マニフェストの **spec.configuration.velero.podConfig.ResourceAllocations** ブロックの値を編集します。

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
spec:
...
  configuration:
    velero:
      podConfig:
        nodeSelector: <node_selector> 1
        resourceAllocations: 2
          limits:
            cpu: "1"
            memory: 1024Mi
          requests:
            cpu: 200m
            memory: 256Mi
```

1. Velero podSpec に提供されるノードセレクターを指定します。
2. リストされている **resourceAllocations** は、平均使用量です。



注記

Kopia は OADP 1.3 以降のリリースで選択できます。Kopia はファイルシステムのバックアップに使用できます。組み込みの Data Mover を使用する Data Mover の場合は、Kopia が唯一の選択肢になります。

Kopia は Restic よりも多くのリソースを消費するため、それに応じて CPU とメモリーの要件を調整しなければならない場合があります。

15.1.3.2. 自己署名 CA 証明書の有効化

certificate signed by unknown authority エラーを防ぐために、**DataProtectionApplication** カスタムリソース (CR) マニフェストを編集して、オブジェクトストレージの自己署名 CA 証明書を有効にする必要があります。

前提条件

- OpenShift API for Data Protection (OADP) Operator がインストールされている必要があります。

手順

- **DataProtectionApplication** CR マニフェストの **spec.backupLocations.velero.objectStorage.caCert** パラメーターと **spec.backupLocations.velero.config** パラメーターを編集します。

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
spec:
...
  backupLocations:
    - name: default
      velero:
        provider: aws
        default: true
        objectStorage:
          bucket: <bucket>
          prefix: <prefix>
          caCert: <base64_encoded_cert_string> 1
        config:
          insecureSkipTLSVerify: "false" 2
...
```

1. Base64 でエンコードされた CA 証明書文字列を指定します。
2. **insecureSkipTLSVerify** 設定は、**"true"** または **"false"** のいずれかに設定できます。**"true"** に設定すると、SSL/TLS セキュリティーが無効になります。**"false"** に設定すると、SSL/TLS セキュリティーが有効になります。

15.1.3.2.1. Velero デプロイメント用のエイリアス化した velero コマンドで CA 証明書を使用する

Velero CLI のエイリアスを作成することで、システムにローカルにインストールせずに Velero CLI を使用できます。

前提条件

- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform クラスターにログインしている。
- OpenShift CLI (**oc**) がインストールされている。

1. エイリアス化した Velero コマンドを使用するには、次のコマンドを実行します。

```
$ alias velero='oc -n openshift-adp exec deployment/velero -c velero -it -- ./velero'
```

2. 次のコマンドを実行して、エイリアスが機能していることを確認します。

   例

```
$ velero version
Client:
 Version: v1.12.1-OADP
 Git commit: -
Server:
 Version: v1.12.1-OADP
```

3. このコマンドで CA 証明書を使用するには、次のコマンドを実行して証明書を Velero デプロイメントに追加できます。

```
$ CA_CERT=$(oc -n openshift-adp get dataprotectionapplications.oadp.openshift.io
<dpa-name> -o jsonpath='{.spec.backupLocations[0].velero.objectStorage.caCert}')

$ [[ -n $CA_CERT ]] && echo "$CA_CERT" | base64 -d | oc exec -n openshift-adp -i
deploy/velero -c velero -- bash -c "cat > /tmp/your-cacert.txt" || echo "DPA BSL has no
caCert"
```

```
$ velero describe backup <backup_name> --details --cacert /tmp/<your_cacert>.txt
```

4. バックアップログを取得するために、次のコマンドを実行します。

```
$ velero backup logs  <backup_name>  --cacert /tmp/<your_cacert.txt>
```

   このログを使用して、バックアップできないリソースの障害と警告を表示できます。

5. Velero Pod が再起動すると、**/tmp/your-cacert.txt** ファイルが消去されます。そのため、前の手順のコマンドを再実行して **/tmp/your-cacert.txt** ファイルを再作成する必要があります。

6. 次のコマンドを実行すると、**/tmp/your-cacert.txt** ファイルを保存した場所にファイルがまだ存在するかどうかを確認できます。

```
$ oc exec -n openshift-adp -i deploy/velero -c velero -- bash -c "ls /tmp/your-cacert.txt"
/tmp/your-cacert.txt
```

OpenShift API for Data Protection (OADP) の今後のリリースでは、この手順が不要になるように証明書を Velero Pod にマウントする予定です。

15.1.4. Data Protection Application 1.2 以前のインストール

DataProtectionApplication API のインスタンスを作成して、Data Protection Application (DPA) をインストールします。

前提条件

- OADP Operator をインストールする。
- オブジェクトストレージをバックアップロケーションとして設定する必要がある。
- スナップショットを使用して PV をバックアップする場合、クラウドプロバイダーはネイティブスナップショット API または Container Storage Interface (CSI) スナップショットのいずれかをサポートする必要がある。
- バックアップとスナップショットの場所で同じ認証情報を使用する場合は、デフォルトの名前である **cloud-credentials** を使用して **Secret** を作成する必要がある。

注記

インストール中にバックアップまたはスナップショットの場所を指定したくない場合は、空の **credentials-velero** ファイルを使用してデフォルトの **Secret** を作成できます。デフォルトの **Secret** がない場合、インストールは失敗します。

注記

Velero は、OADP namespace に **velero-repo-credentials** という名前のシークレットを作成します。これには、デフォルトのバックアップリポジトリーパスワードが含まれます。バックアップリポジトリーを対象とした最初のバックアップを実行する **前** に、base64 としてエンコードされた独自のパスワードを使用してシークレットを更新できます。更新するキーの値は **Data[repository-password]** です。

DPA を作成した後、バックアップリポジトリーを対象としたバックアップを初めて実行するときに、Velero はシークレットが **velero-repo-credentials** のバックアップリポジトリーを作成します。これには、デフォルトのパスワードまたは置き換えたパスワードが含まれます。最初のバックアップの **後** にシークレットパスワードを更新すると、新しいパスワードが **velero-repo-credentials** のパスワードと一致しなくなり、Velero は古いバックアップに接続できなくなります。

手順

1. **Operators → Installed Operators** をクリックして、OADP Operator を選択します。
2. **Provided APIs** で、**DataProtectionApplication** ボックスの **Create instance** をクリックします。
3. **YAML View** をクリックして、**DataProtectionApplication** マニフェストのパラメーターを更新します。
4. **Create** をクリックします。

検証

1. 次のコマンドを実行して OpenShift API for Data Protection (OADP) リソースを表示し、インストールを検証します。

```
$ oc get all -n openshift-adp
```

出力例

```
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/oadp-operator-controller-manager-67d9494d47-6l8z8   2/2     Running   0          2m8s
pod/restic-9cq4q                                     1/1     Running   0          94s
pod/restic-m4lts                                     1/1     Running   0          94s
pod/restic-pv4kr                                     1/1     Running   0          95s
pod/velero-588db7f655-n842v                          1/1     Running   0          95s

NAME                                                     TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)    AGE
service/oadp-operator-controller-manager-metrics-service   ClusterIP   172.30.70.140
<none>        8443/TCP   2m8s

NAME                  DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
daemonset.apps/restic   3         3         3       3            3           <none>          96s

NAME                                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/oadp-operator-controller-manager   1/1     1            1           2m9s
deployment.apps/velero                             1/1     1            1           96s

NAME                                                    DESIRED   CURRENT   READY   AGE
replicaset.apps/oadp-operator-controller-manager-67d9494d47   1         1         1       2m9s
replicaset.apps/velero-588db7f655                       1         1         1       96s
```

2. 次のコマンドを実行して、**DataProtectionApplication** (DPA) が調整されていることを確認します。

```
$ oc get dpa dpa-sample -n openshift-adp -o jsonpath='{.status}'
```

出力例

```
{"conditions":[{"lastTransitionTime":"2023-10-27T01:23:57Z","message":"Reconcile
complete","reason":"Complete","status":"True","type":"Reconciled"}]}
```

3. **type** が **Reconciled** に設定されていることを確認します。

4. 次のコマンドを実行して、バックアップ保存場所を確認し、**PHASE** が **Available** であることを確認します。

```
$ oc get backupstoragelocations.velero.io -n openshift-adp
```

出力例

```
NAME          PHASE      LAST VALIDATED   AGE     DEFAULT
dpa-sample-1  Available  1s               3d16h   true
```

15.1.5. Data Protection Application のインストール

DataProtectionApplication API のインスタンスを作成して、Data Protection Application (DPA) をインストールします。

前提条件

- OADP Operator をインストールする。
- オブジェクトストレージをバックアップロケーションとして設定する必要がある。
- スナップショットを使用して PV をバックアップする場合、クラウドプロバイダーはネイティブスナップショット API または Container Storage Interface (CSI) スナップショットのいずれかをサポートする必要がある。
- バックアップとスナップショットの場所で同じ認証情報を使用する場合は、デフォルトの名前である **cloud-credentials** を使用して **Secret** を作成する必要がある。

注記

インストール中にバックアップまたはスナップショットの場所を指定したくない場合は、空の **credentials-velero** ファイルを使用してデフォルトの **Secret** を作成できます。デフォルトの **Secret** がない場合、インストールは失敗します。

手順

1. **Operators → Installed Operators** をクリックして、OADP Operator を選択します。
2. **Provided APIs** で、**DataProtectionApplication** ボックスの **Create instance** をクリックします。
3. **YAML View** をクリックして、**DataProtectionApplication** マニフェストのパラメーターを更新します。
4. **Create** をクリックします。

検証

1. 次のコマンドを実行して OpenShift API for Data Protection (OADP) リソースを表示し、インストールを検証します。

```
$ oc get all -n openshift-adp
```

出力例

```
NAME                                                READY   STATUS    RESTARTS   AGE
pod/oadp-operator-controller-manager-67d9494d47-6l8z8   2/2     Running   0          2m8s
pod/node-agent-9cq4q                                1/1     Running   0          94s
pod/node-agent-m4lts                                1/1     Running   0          94s
pod/node-agent-pv4kr                                1/1     Running   0          95s
pod/velero-588db7f655-n842v                         1/1     Running   0          95s
```

```
NAME                                                 TYPE       CLUSTER-IP       EXTERNAL-IP
PORT(S)    AGE
service/oadp-operator-controller-manager-metrics-service   ClusterIP   172.30.70.140
<none>        8443/TCP   2m8s
service/openshift-adp-velero-metrics-svc                   ClusterIP   172.30.10.0      <none>
8085/TCP   8h

NAME                  DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
daemonset.apps/node-agent    3         3         3       3            3          <none>          96s

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/oadp-operator-controller-manager    1/1     1            1           2m9s
deployment.apps/velero                              1/1     1            1           96s

NAME                                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/oadp-operator-controller-manager-67d9494d47    1         1         1       2m9s
replicaset.apps/velero-588db7f655                              1         1         1       96s
```

2. 次のコマンドを実行して、**DataProtectionApplication** (DPA) が調整されていることを確認します。

```
$ oc get dpa dpa-sample -n openshift-adp -o jsonpath='{.status}'
```

出力例

```
{"conditions":[{"lastTransitionTime":"2023-10-27T01:23:57Z","message":"Reconcile
complete","reason":"Complete","status":"True","type":"Reconciled"}]}
```

3. **type** が **Reconciled** に設定されていることを確認します。

4. 次のコマンドを実行して、バックアップ保存場所を確認し、**PHASE** が **Available** であることを確認します。

```
$ oc get backupstoragelocations.velero.io -n openshift-adp
```

出力例

```
NAME           PHASE       LAST VALIDATED   AGE     DEFAULT
dpa-sample-1   Available   1s               3d16h   true
```

15.1.5.1. DataProtectionApplication CR で CSI を有効にする

CSI スナップショットを使用して永続ボリュームをバックアップするには、**DataProtectionApplication** カスタムリソース (CR) で Container Storage Interface (CSI) を有効にします。

前提条件

- クラウドプロバイダーは、CSI スナップショットをサポートする必要があります。

手順

- 次の例のように、**DataProtectionApplication** CR を編集します。

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
...
spec:
  configuration:
    velero:
      defaultPlugins:
      - openshift
      - csi (1)
```

(1) **csi** デフォルトプラグインを追加します。

15.1.6. OADP のアンインストール

OpenShift API for Data Protection (OADP) をアンインストールするには、OADP Operator を削除します。詳細は、クラスターからの Operators の削除 を参照してください。

15.2. 仮想マシンのバックアップと復元

重要

OpenShift Virtualization の OADP は、テクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

OpenShift API for Data Protection (OADP) を使用して、仮想マシンをバックアップおよび復元します。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. ストレージプロバイダーの指示に従って、OADP Operator をインストールします。
2. **kubevirt** および **openshift** プラグイン を使用して データ保護アプリケーション をインストールします。
3. **Backup** カスタムリソース (CR) を作成して、仮想マシンをバックアップします。
4. **Restore** CR を作成して、**Backup** CR を復元します。

15.2.1. 関連情報

- OADP features and plugins
- トラブルシューティング

15.3. 仮想マシンのバックアップ

重要

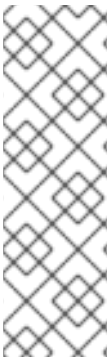
OpenShift Virtualization の OADP は、テクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

OpenShift API for Data Protection (OADP) の **Backup** カスタムリソース (CR) を作成して、仮想マシン (VM) をバックアップします。

Backup CR は以下のアクションを実行します。

- Multicloud Object Gateway、Noobaa、または Minio などの S3 互換オブジェクトストレージにアーカイブファイルを作成して、OpenShift Virtualization リソースをバックアップします。
- 以下のオプションのいずれかを使用して、仮想マシンディスクをバックアップします。
  - Ceph RBD または Ceph FS などの CSI 対応クラウドストレージ上の Container Storage Interface (CSI) スナップショット。
  - ファイルシステムバックアップを使用してアプリケーションをバックアップします。オブジェクトストレージ上の Kopia または Restic。



注記

OADP はバックアップフックを提供し、バックアップ操作の前に仮想マシンのファイルシステムをフリーズし、バックアップの完了時にフリーズを解除します。

kubevirt-controller は、バックアップ操作の前後に Velero が **virt-freezer** バイナリーを実行できるようにするアノテーションで **virt-launcher** Pod を作成します。

freeze および **unfreeze** API は、仮想マシンスナップショット API のサブリソースです。詳細は、仮想マシンのスナップショットについてを参照してください。

フック を**Backup** CR に追加して、バックアップ操作の前後に特定の仮想マシンでコマンドを実行できます。

Backup CR の代わりに **Schedule** CR を作成することにより、バックアップをスケジュールします。

15.3.1. Backup CR の作成

Kubernetes リソース、内部イメージ、および永続ボリューム(PV)をバックアップするには、Backup カスタムリソース(CR)を作成します。

前提条件

- OpenShift API for Data Protection (OADP) Operator をインストールしている。
- **DataProtectionApplication** CR が **Ready** 状態である。
- バックアップロケーションの前提条件:
  - Velero 用に S3 オブジェクトストレージを設定する必要があります。
  - **DataProtectionApplication** CR でバックアップの場所を設定する必要があります。
- スナップショットの場所の前提条件:
  - クラウドプロバイダーには、ネイティブスナップショット API が必要であるか、Container Storage Interface (CSI) スナップショットをサポートしている必要があります。
  - CSI スナップショットの場合、CSI ドライバーを登録するために **VolumeSnapshotClass** CR を作成する必要があります。
  - **DataProtectionApplication** CR でボリュームの場所を設定する必要があります。

手順

1. 次のコマンドを入力して、**backupStorageLocations** CR を取得します。

```
$ oc get backupstoragelocations.velero.io -n openshift-adp
```

出力例

```
NAMESPACE       NAME              PHASE       LAST VALIDATED   AGE   DEFAULT
openshift-adp   velero-sample-1   Available   11s              31m
```

2. 次の例のように、**Backup** CR を作成します。

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup>
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  hooks: {}
  includedNamespaces:
  - <namespace> 1
  includedResources: [] 2
  excludedResources: [] 3
  storageLocation: <velero-sample-1> 4
  ttl: 720h0m0s
  labelSelector: 5
    matchLabels:
      app=<label_1>
      app=<label_2>
      app=<label_3>
```

```
    orLabelSelectors: 6
    - matchLabels:
        app=<label_1>
        app=<label_2>
        app=<label_3>
```

1. バックアップする namespace の配列を指定します。
2. オプション: バックアップに含めるリソースの配列を指定します。リソースは、省略語 ('pods' は 'po' など) または完全修飾の場合があります。指定しない場合、すべてのリソースが含まれます。
3. オプション: バックアップから除外するリソースの配列を指定します。リソースは、省略語 ('pods' は 'po' など) または完全修飾の場合があります。
4. **backupStorageLocations** CR の名前を指定します。
5. 指定したラベルを **すべて** 持つバックアップリソースの {key,value} ペアのマップ。
6. 指定したラベルを **1つ以上** 持つバックアップリソースの {key,value} ペアのマップ。

3. **Backup** CR のステータスが **Completed** したことを確認します。

```
$ oc get backups.velero.io -n openshift-adp <backup> -o jsonpath='{.status.phase}'
```

15.3.1.1. CSI スナップショットを使用した永続ボリュームのバックアップ

Backup CR を作成する前に、クラウドストレージの **VolumeSnapshotClass** カスタムリソース (CR) を編集して、Container Storage Interface (CSI) スナップショットを使用して永続ボリュームをバックアップします。

前提条件

- クラウドプロバイダーは、CSI スナップショットをサポートする必要があります。
- **DataProtectionApplication** CR で CSI を有効にする必要があります。

手順

- **metadata.labels.velero.io/csi-volumesnapshot-class: "true"** のキー: 値ペアを **VolumeSnapshotClass** CR に追加します。

  設定ファイルのサンプル

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: <volume_snapshot_class_name>
  labels:
    velero.io/csi-volumesnapshot-class: "true" 1
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: true 2
driver: <csi_driver>
deletionPolicy: <deletion_policy_type> 3
```

-

1. **true** に設定する必要があります。
2. **true** に設定する必要があります。
3. OADP は、CSI および Data Mover のバックアップと復元に対して、**Retain** および **Delete** 削除ポリシータイプをサポートしています。OADP 1.2 Data Mover の場合、削除ポリシータイプを **Retain** に設定します。

次のステップ

- これで、**Backup** CR を作成できます。

15.3.1.2. Restic を使用したアプリケーションのバックアップ

Backup カスタムリソース (CR) を編集して、Restic を使用して Kubernetes リソース、内部イメージ、および永続ボリュームをバックアップします。

DataProtectionApplication CR でスナップショットの場所を指定する必要はありません。

重要

Restic は、**hostPath** ボリュームのバックアップをサポートしません。詳細は、追加の Restic 制限事項 を参照してください。

前提条件

- OpenShift API for Data Protection (OADP) Operator をインストールしている。
- **DataProtectionApplication** CR で **spec.configuration.restic.enable** を **false** に設定して、デフォルトの Restic インストールを無効にしていない。
- **DataProtectionApplication** CR が **Ready** 状態である。

手順

- 次の例のように、**Backup** CR を編集します。

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup>
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  defaultVolumesToFsBackup: true 1
...
```

1. OADP バージョン 1.2 以降では、**defaultVolumesToFsBackup: true** 設定を **spec** ブロック内に追加します。OADP バージョン 1.1 では、**defaultVolumesToRestic: true** を追加します。

15.3.1.3. バックアップフックの作成

Backup カスタムリソース (CR) を編集して、Pod 内のコンテナーでコマンドを実行するためのバックアップフックを作成します。

プレ フックは、Pod のバックアップが作成される前に実行します。**ポスト** フックはバックアップ後に実行します。

手順

- 次の例のように、**Backup** CR の **spec.hooks** ブロックにフックを追加します。

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup>
  namespace: openshift-adp
spec:
  hooks:
    resources:
      - name: <hook_name>
        includedNamespaces:
        - <namespace> 1
        excludedNamespaces: 2
        - <namespace>
        includedResources: []
        - pods 3
        excludedResources: [] 4
        labelSelector: 5
          matchLabels:
            app: velero
            component: server
        pre: 6
          - exec:
              container: <container> 7
              command:
              - /bin/uname 8
              - -a
              onError: Fail 9
              timeout: 30s 10
        post: 11
...
```

1. オプション: フックが適用される namespace を指定できます。この値が指定されていない場合、フックはすべてのネームスペースに適用されます。
2. オプション: フックが適用されない namespace を指定できます。
3. 現在、Pod は、フックを適用できる唯一のサポート対象リソースです。
4. オプション: フックが適用されないリソースを指定できます。
5. オプション: このフックは、ラベルに一致するオブジェクトにのみ適用されます。この値が指定されていない場合、フックはすべてのネームスペースに適用されます。

6 バックアップの前に実行するフックの配列。

7 オプション: コンテナーが指定されていない場合、コマンドは Pod の最初のコンテナーで実行されます。

8 これは、追加される init コンテナーのエントリーポイントです。

9 エラー処理に許可される値は、**Fail** と **Continue** です。デフォルトは **Fail** です。

10 オプション: コマンドの実行を待機する時間。デフォルトは **30s** です。

11 このブロックでは、バックアップ後に実行するフックの配列を、バックアップ前のフックと同じパラメーターで定義します。

15.3.2. 関連情報

- CSI ボリュームスナップショットの概要

15.4. 仮想マシンの復元

Restore CR を作成して、OpenShift API for Data Protection (OADP) の **Backup** カスタムリソース (CR) を復元します。

フック を **Restore** CR に追加して、アプリケーションコンテナーの起動前に init コンテナーでコマンドを実行するか、アプリケーションコンテナー自体でコマンドを実行することができます。

15.4.1. 復元 CR の作成

Restore CR を作成して、**Backup** カスタムリソース (CR) を復元します。

前提条件

- OpenShift API for Data Protection (OADP) Operator をインストールしている。
- **DataProtectionApplication** CR が **Ready** 状態である。
- Velero **Backup** CR がある。
- 永続ボリューム (PV) の容量は、バックアップ時に要求されたサイズと一致する必要があります。必要に応じて、要求されたサイズを調整します。

手順

1. 次の例のように、**Restore** CR を作成します。

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: <restore>
  namespace: openshift-adp
spec:
  backupName: <backup> 1
  includedResources: [] 2
  excludedResources:
```

```
    - nodes
    - events
    - events.events.k8s.io
    - backups.velero.io
    - restores.velero.io
    - resticrepositories.velero.io
  restorePVs: true (3)
```

(1) **Backup** CR の名前

(2) オプション: 復元プロセスに含めるリソースの配列を指定します。リソースは、ショートカット (**Pods** は **po** など) または完全修飾の場合があります。指定しない場合、すべてのリソースが含まれます。

(3) オプション: **restorePVs** パラメーターを **false** に設定すると、コンテナーストレージインターフェイス (CSI) スナップショットの **VolumeSnapshot** から、または **VolumeSnapshotLocation** が設定されている場合はネイティブスナップショットからの **PersistentVolumes** の復元をオフにすることができます。

2. 次のコマンドを入力して、**Restore** CR のステータスが **Completed** であることを確認します。

```
$ oc get restores.velero.io -n openshift-adp <restore> -o jsonpath='{.status.phase}'
```

3. 次のコマンドを入力して、バックアップリソースが復元されたことを確認します。

```
$ oc get all -n <namespace> (1)
```

(1) バックアップした namespace。

4. ボリュームを使用して **DeploymentConfig** を復元する場合、または復元後のフックを使用する場合は、次のコマンドを入力して **dc-post-restore.sh** クリーンアップスクリプトを実行します。

```
$ bash dc-restic-post-restore.sh -> dc-post-restore.sh
```

注記

復元プロセス中に、OADP Velero プラグインは **DeploymentConfig** オブジェクトをスケールダウンし、Pod をスタンドアロン Pod として復元します。これは、クラスターが復元された **DeploymentConfig** Pod を復元時にすぐに削除することを防ぎ、復元フックと復元後のフックが復元された Pod 上でアクションを完了できるようにするために行われます。以下に示すクリーンアップスクリプトは、これらの切断された Pod を削除し、**DeploymentConfig** オブジェクトを適切な数のレプリカにスケールアップします。

例15.1 dc-restic-post-restore.sh → dc-post-restore.sh クリーンアップスクリプト

```
#!/bin/bash
set -e

# if sha256sum exists, use it to check the integrity of the file
if command -v sha256sum >/dev/null 2>&1; then
```

```
  CHECKSUM_CMD="sha256sum"
else
  CHECKSUM_CMD="shasum -a 256"
fi

label_name () {
    if [ "${#1}" -le "63" ]; then
 echo $1
 return
    fi
    sha=$(echo -n $1|$CHECKSUM_CMD)
    echo "${1:0:57}${sha:0:6}"
}

if [[ $# -ne 1 ]]; then
    echo "usage: ${BASH_SOURCE} restore-name"
    exit 1
fi

echo "restore: $1"

label=$(label_name $1)
echo "label:   $label"

echo Deleting disconnected restore pods
oc delete pods --all-namespaces -l oadp.openshift.io/disconnected-from-dc=$label

for dc in $(oc get dc --all-namespaces -l oadp.openshift.io/replicas-modified=$label -o
jsonpath='{range .items[*]}{.metadata.namespace}{","}{.metadata.name}{","}
{.metadata.annotations.oadp\.openshift\.io/original-replicas}{","}
{.metadata.annotations.oadp\.openshift\.io/original-paused}{"\n"}')
do
    IFS=',' read -ra dc_arr <<< "$dc"
    if [ ${#dc_arr[0]} -gt 0 ]; then
 echo Found deployment ${dc_arr[0]}/${dc_arr[1]}, setting replicas: ${dc_arr[2]}, paused:
${dc_arr[3]}
 cat <<EOF | oc patch dc  -n ${dc_arr[0]} ${dc_arr[1]} --patch-file /dev/stdin
spec:
  replicas: ${dc_arr[2]}
  paused: ${dc_arr[3]}
EOF
    fi
done
```

15.4.1.1. 復元フックの作成

Restore カスタムリソース (CR) を編集して、Pod 内のコンテナーでコマンドを実行する復元フックを作成します。

2 種類の復元フックを作成できます。

- **init** フックは、init コンテナーを Pod に追加して、アプリケーションコンテナーが起動する前にセットアップタスクを実行します。

Restic バックアップを復元する場合は、復元フック init コンテナーの前に **restic-wait** init コンテナーが追加されます。

- **exec** フックは、復元された Pod のコンテナーでコマンドまたはスクリプトを実行します。

手順

- 次の例のように、**Restore CR** の **spec.hooks** ブロックにフックを追加します。

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: <restore>
  namespace: openshift-adp
spec:
  hooks:
    resources:
      - name: <hook_name>
        includedNamespaces:
        - <namespace> 1
        excludedNamespaces:
        - <namespace>
        includedResources:
        - pods 2
        excludedResources: []
        labelSelector: 3
          matchLabels:
            app: velero
            component: server
        postHooks:
        - init:
            initContainers:
            - name: restore-hook-init
              image: alpine:latest
              volumeMounts:
              - mountPath: /restores/pvc1-vm
                name: pvc1-vm
              command:
              - /bin/ash
              - -c
            timeout: 4
        - exec:
            container: <container> 5
            command:
            - /bin/bash 6
            - -c
            - "psql < /backup/backup.sql"
            waitTimeout: 5m 7
            execTimeout: 1m 8
            onError: Continue 9
```

1 オプション: フックが適用される namespace の配列。この値が指定されていない場合、フックはすべてのネームスペースに適用されます。

2 現在、Pod は、フックを適用できる唯一のサポート対象リソースです。

3 オプション: このフックは、ラベルセレクターに一致するオブジェクトにのみ適用されます。

4 オプション: Timeout は、**initContainers** が完了するまで Velero が待機する最大時間を指定します。

5 オプション: コンテナーが指定されていない場合、コマンドは Pod の最初のコンテナーで実行されます。

6 これは、追加される init コンテナーのエントリーポイントです。

7 オプション: コンテナーの準備が整うまでの待機時間。これは、コンテナーが起動して同じコンテナー内の先行するフックが完了するのに十分な長さである必要があります。設定されていない場合、復元プロセスの待機時間は無期限になります。

8 オプション: コマンドの実行を待機する時間。デフォルトは **30s** です。

9 エラー処理に許可される値は、**Fail** および **Continue** です。

- **Continue**: コマンドの失敗のみがログに記録されます。
- **Fail**: Pod 内のコンテナーで復元フックが実行されなくなりました。**Restore** CR のステータスは **PartiallyFailed** になります。