OpenShift Container Platform 4.10

仮想化

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: 2023-10-18

OpenShift Container Platform 4.10 仮想化

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 仮想マシンの作成と管理
- 各種コンソールおよび CLI ツールの使用による仮想マシンへの接続
- 既存の仮想マシンのインポートおよびクローン作成
- ネットワークインターフェ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。

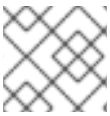
OpenShift Virtualization は、OVN-Kubernetes、OpenShiftSDN、または 認定 OpenShift CNI プラグイン に一記載されている他の認定デフォルトの Container Network Interface (CNI) ネットワークプロバイダーの 1 つと使用できます。

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

OpenShift Virtualization 4.10 は OpenShift Container Platform 4.10 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。

第2章 OPENSHIFT VIRTUALIZATION の開始

基本的な OpenShift Virtualization 環境をインストールして設定し、そ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。

2.1. 作業を開始する前に

- インストール要件 を確認します。
- クローン作成、スナップショット、およびライブマイグレーションに必要な ストレージ機能 を確認します。詳細については、CSI 対応のストレージ プロバイダーの使用 を参照してください。
- OpenShift Virtualization Operator をインストールします。
- **virtctl** ツール をインストールします。

2.2. スタートガイド

仮想マシンを作成します。

- ウィザードを使用して RHEL 仮想マシン を作成します。
- Windows 仮想マシンを作成します。
  - Windows ブート ソース を作成してカスタマイズします。
  - ウィザードを使用して Windows 仮想マシン を作成します。
  - VirtIO ドライバーと QEMU ゲストエージェント を Windows 仮想マシンにインストールします。

仮想マシンに接続する

- Web コンソールを使用して、仮想マシンの シリアルコンソール 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。

仮想マシンの管理

- Web コンソールから 仮想マシンを停止、開始、一時停止、再起動します。
- 仮想マシンを管理し、ポートを公開し、コマンド ラインから**virtctl** を使用して仮想マシンのシリアルコンソールに接続します。

2.3. 次のステップ

仮想マシンをセカンダリーネットワークに接続する

- 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。

OpenShift Virtualization 環境を監視する

- **仮想化の概要**ページ で、リソース、詳細、ステータス、上位のコンシューマーを監視します。
- **仮想マシン**ダッシュボード で、仮想マシンに関する概要情報を表示します。
- 仮想マシンの ログ を表示します。

展開を自動化する

- Ansible を使用して 仮想マシンのデプロイを自動化 します。
- **sysprep** を使用して Windows 仮想マシンのデプロイメントを自動化 します。

2.4. 関連情報

- Kubernetes NMState Operator について
- 仮想マシンのノードの指定
- ライブマイグレーション
- 仮想マシンテンプレート
- ローカルストレージの設定
- バックアップおよび復元

第3章 OPENSHIFT VIRTUALIZATION リリースノート

3.1. 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のいずれかで OpenShift Virtualization を使用できます。

OpenShift Virtualization の機能 について参照してください。

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

OpenShift Virtualization 4.10 は OpenShift Container Platform 4.10 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。

3.1.2. 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Certified Guest Operating Systems in Red Hat OpenStack Platform, Red Hat Virtualization and OpenShift Virtualization参照してください。

3.2. 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、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

3.3. 新機能および変更された機能

- 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 8** という名前が付けられます。
  - Intel および AMD CPU。
- OpenShift Virtualization が OpenShift Service Mesh に統合されるようになりました。仮想マシンをサービスメッシュに接続 することで、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。

- OpenShift Virtualization は、事前に定義されたブートソースの自動インポートおよび更新 用に統一された API を提供するようになりました。

3.3.1. 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 **virtual machine** 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。

3.3.2. インストール

- **virt-launcher** Pod などの OpenShift Virtualization ワークロードは、ライブマイグレーションをサポートしている場合に自動更新されるようになりました。**HyperConverged** カスタムリソース (CR) を編集して、ワークロードの更新ストラテジーを設定 したり、今後の自動更新をオプトアウトしたりできます。

- Single Node OpenShift (SNO) とも呼ばれるシングルノードクラスターで OpenShift Virtualization を使用できるようになりました。

注記

シングルノードクラスターは高可用性操作用に設定されていないため、OpenShift Virtualization の動作が大幅に変更されます。

- リソース要求および優先順位クラスは、すべての OpenShift Virtualization コントロールプレーンコンポーネントに対して定義されるようになりました。

3.3.3. ネットワーク

- 単一の **NodeNetworkConfigurationPolicy** マニフェストを使用して、複数の nmstate 対応ノードを同時に設定 できるようになりました。

- SR-IOV 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に対して、ライブマイグレーションがデフォルトでサポートされるようになりました。

3.3.4. ストレージ

- ホットプラグされた仮想ディスクの仮想マシンで、オンラインスナップショットがサポートされます。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。

- ホストパスプロビジョナー (HPP) で Kubernetes Container Storage Interface(CSI) ドライバーを使用し、仮想マシンのローカルストレージを設定できます。CSI ドライバーを使用すると、ローカルストレージの設定時に既存の OpenShift Container Platform ノードおよびクラスターの中断を最小限に抑えることができます。

3.3.5. Web コンソール

- OpenShift Virtualization ダッシュボードでは、仮想マシンおよび関連付けられた Pod のリソース消費に関するデータが得られます。OpenShift Virtualization ダッシュボードに表示される可視化メトリクスは、Prometheus Query Language(PromQL) クエリー に基づいています。

3.4. 非推奨および削除された機能

3.4.1. 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。

- 今後のリリースでは、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非推奨になります。OpenShift Virtualization 4.10 以降、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。Operator は、引き続き HPP カスタムリソースの既存の (レガシー) 形式および関連付けられたストレージクラスをサポートします。HPP Operator を使用する場合、移行ストラテジーの一部として CSI ドライバーのストレージクラスを作成する ことを計画してください。

3.4.2. 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。

- このリリースでは、VM Import Operator が OpenShift Virtualization から削除されました。これは、 Migration Toolkit for Virtualization に置き換えられました。
- 本リリースでは、2021 年 12 月 31 日に End of Life (EOL) に到達した CentOS Linux 8 のテンプレートが削除されました。ただし、OpenShift Container Platform には CentOS Stream 8 および CentOS Stream 9 のテンプレートが含まれるようになりました。

注記

CentOS ディストリビューションはすべてコミュニティーでサポートされています。

3.5. 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。これらの機能については、Red Hat カスタマーポータルのTechnology Preview Features Support Scope を参照してください。

- Red Hat Enterprise Linux 9 Beta テンプレートを使用し、仮想マシンを作成できます。
- OpenShift Virtualization を AWS ベアメタルノードにデプロイ できるようになりました。
- OpenShift Virtualization の重大なアラート には、早急な対応が必要な問題に対する説明、各アラートが発生する理由、問題の原因を診断するためのトラブルシューティングプロセス、および各アラートを解決する手順が含まれるようになりました。
- クラスター管理者は、OpenShift Virtualization プラグインで OpenShift API for Data Protection を使用して、仮想マシンが含まれる namespace をバックアップできるようになりました。
- 管理者は、**HyperConverged**CR を編集することにより、宣言的に、仮想グラフィックス処理ユニット (vGPU) などの仲介デバイスを作成および公開できるようになりました。仮想マシンの所有者は、これらのデバイスを仮想マシンに割り当てることができます。
- 単一の**NodeNetworkConfigurationPolicy**マニフェストをクラスターに適用することにより、ブリッジに接続されている NIC の静的 IP 設定を転送 できます。

- OpenShift Virtualization を IBM Cloud ベアメタルサーバーにインストール できるようになりました。他のクラウドプロバイダーが提供するベアメタルサーバーはサポートされません。

3.6. バグ修正

- クローンソースが利用可能になる前にクローン操作を開始すると、クローン操作は回避策を使用せずに正常に完了するようになりました。(BZ#1855182)

- バージョン 4.8 より前の Open Shift Virtualization によって提供された削除済みテンプレートを仮想マシンが参照している場合、仮想マシンの編集は失敗します。Open Shift Virtualization 4.8 以降では、削除された Open Shift Virtualization が提供するテンプレートは、Open Shift Virtualization Operator によって自動的に再作成されます。(BZ#1929165)

- VNC コンソールで仮想マシンを使用する場合は、**Send Keys** および **Disconnect** ボタンを正常に使用できるようになりました。(BZ#1964789)

- 仮想マシンを作成すると、一意の完全修飾ドメイン名 (FQDN) にクラスターのドメイン名が含まれるようになりました。(BZ#1998300)

- 仮想ディスクをホットプラグしてから **virt-launcher** Pod を強制的に削除した場合に、データが失われることはなくなりました。(BZ#2007397)

- OpenShift Virtualization は、他の重要なコンポーネントとファイルシステムを共有するパスにホストパスプロビジョナー (HPP) をインストールしようとすると、HPPSharingPoolPathWithOS アラートを発行するようになりました。
HPP を使用して仮想マシンディスクのストレージを提供するには、ノードのルートファイルシステムとは別の専用のストレージで設定します。そうしない場合には、ノードはストレージが不足し、機能しなくなる可能性があります。(BZ#2038985)

- 仮想マシンディスクをプロビジョニングする場合、OpenShift Virtualization は、各仮想マシンディスク PVC に KubePersistentVolumeFillingUp アラートを発行するのではなく、要求されるディスクサイズに対応するのに十分な大きさの永続ボリューム要求 (PVC) を割り当てるようになりました。仮想マシン内からディスク使用量をモニタリングできます。(BZ#2039489)

- ホットプラグされたディスクを持つ仮想マシンの仮想マシンスナップショットを作成できるようになりました。(BZ#2042908)

- クラスター全体のプロキシー設定を使用する場合に、仮想マシンイメージを正常にインポートできるようになりました。(BZ#2046271)

3.7. 既知の問題

- 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。

- OpenShift Virtualization 4.10.5 に更新すると、一部の仮想マシン (VM) がライブマイグレーションループでスタックします。これは、仮想マシンマニフェストの **spec.volumes.containerDisk.path** フィールドが相対パスに設定されている場合に発生しま

す。

- 回避策として、仮想マシンマニフェストを削除して再作成し、**spec.volumes.containerDisk.path** フィールドの値を絶対パスに設定します。その後、OpenShift Virtualization を更新できます。

- 単一ノードに 50 を超えるイメージが含まれている場合、Pod のスケジューリングがノード間で不均衡になる可能性があります。これは、ノード上のイメージのリストがデフォルトで 50 に短縮されているためです。(BZ#1984442)

  - 回避策として、**KubeletConfig** オブジェクトを編集し、**nodeStatusMaxImages** の値を **-1** に設定することで、イメージの制限を無効にすることができます。

- 42 文字を超える完全修飾ドメイン名 (FQDN) を持つノードがあるクラスターに ホストパスプロビジョナー をデプロイする場合、プロビジョナーは PVC をバインドできません。(BZ#2057157)

  エラーメッセージの例

  ```
  E0222 17:52:54.088950       1 reflector.go:138] k8s.io/client-go/informers/factory.go:134: Failed to watch *v1beta1.CSIStorageCapacity: failed to list *v1beta1.CSIStorageCapacity: unable to parse requirement: values[0][csi.storage.k8s.io/managed-by]: Invalid value: "external-provisioner-<node_FQDN>": must be no more than 63 characters (1)
  ```

  (1) エラーメッセージの最大文字数は 63 文字ですが、これにはノードの FQDN の先頭に付く **external-provisioner-** の文字列が含まれます。

  - 回避策として、以下のコマンドを実行して、ホストパスプロビジョナー CSI ドライバーの **storageCapacity** オプションを無効にします。

    ```
    $ oc patch csidriver kubevirt.io.hostpath-provisioner --type merge --patch '{"spec": {"storageCapacity": false}}'
    ```

- 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。

- ライブマイグレーションを実行できない仮想マシンを実行すると、OpenShift Container Platform クラスターのアップグレードがブロックされる可能性があります。これには、hostpath-provisioner ストレージまたは SR-IOV ネットワークインターフェイスを使用する仮想マシンが含まれます。

  - 回避策として、仮想マシンを再設定し、クラスターのアップグレード時にそれらの電源をオフにするようにできます。仮想マシン設定ファイルの **spec** セクションで、以下を実行します。

    1. **evictionStrategy** および **runStrategy** フィールドを変更します。

       a. **evictionStrategy: LiveMigrate** フィールドを削除します。エビクションストラテジーの設定方法についての詳細は、仮想マシンのエビクションストラテジーの設定を参照してください。

b. **runStrategy** フィールドを **Always** に設定します。

2. 以下のコマンドを実行して、デフォルトの CPU モデルを設定します。

注記

ライブマイグレーションをサポートする仮想マシンを起動する前に、この変更を行う必要があります。

```
$ oc annotate --overwrite -n openshift-cnv hyperconverged kubevirt-hyperconverged
kubevirt.kubevirt.io/jsonpatch='[
  {
    "op": "add",
    "path": "/spec/configuration/cpuModel",
    "value": "<cpu_model>" 1
  }
]'
```

1 **<cpu-model>** を実際の CPU モデルの値に置き換えます。すべてのノードに **oc describe node <node>** を実行し、**cpu-model-<name>** ラベルを確認してこの値を判別できます。すべてのノードに存在する CPU モデルを選択します。

- Red Hat Ceph Storage または Red Hat OpenShift Data Foundation Storage を使用する場合は、一度に 100 台以上の仮想マシンのクローンを作成できない場合があります。(BZ#1989527)
  - 回避策として、ストレージプロファイルマニフェストに **spec.cloneStrategy: copy** を設定して、ホスト支援コピーを実行できます。以下に例を示します。

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <provisioner_class>
#   ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce
    volumeMode: Filesystem
  cloneStrategy: copy 1
status:
  provisioner: <provisioner>
  storageClass: <provisioner_class>
```

1 デフォルトのクローン作成方法は、**copy** として設定されます。

- 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スにします。

- 大規模なクラスターでは、OpenShift Virtualization MAC プールマネージャーの起動に時間がかかりすぎる可能性があり、OpenShift Virtualization が準備状態にならない可能性があります。(BZ#2035344)

  - 回避策として、MAC プーリング機能が必要ない場合には、以下のコマンドを実行してこのサブコンポーネントを無効にします。

```
$ oc annotate --overwrite -n openshift-cnv hco kubevirt-hyperconverged
'networkaddonsconfigs.kubevirt.io/jsonpatch=[
 {
   "op": "replace"
   "path": "/spec/kubeMacPool"
   "value": null
 }
]'
```

- 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。

- シャットダウン中に仮想マシンがクラッシュしたりハングアップしたりした場合に、新しいシャットダウン要求は仮想マシンを停止しません。(BZ#2040766)

- ドライバーをインストールする前に仲介デバイスを有効にするように **HyperConverged** カスタムリソース (CR) を設定する場合は、仲介デバイスの有効化は発生しません。この問題は更新によってトリガーされます。たとえば、NVIDIA ドライバーをインストールする **daemonset** の前に **virt-handler** を更新した場合、ノードは仮想マシン GPU を提供することができません。(BZ#2046298)

  - 回避策として、以下を実施します。

    1. **HyperConverged** CR から **mediatedDevicesConfiguration** および **permittedHostDevices** を削除します。

    2. 使用する設定で、**mediatedDevicesConfiguration** および **permittedHostDevices** スタンザの両方を更新します。

- 仮想マシンウィザードの YAML サンプルはハードコーディングされており、常に最新のアップストリーム変更が含まれるわけではありません。(BZ#2055492)

- **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)

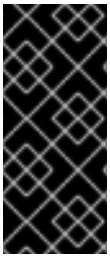
  - 回避策として、**ceph-mgr** を再起動して仮想マシンのクローンをパージすることができます。

- 非特権ユーザーは、**VM Network Interfaces** タブの Add Network Interface ボタンを使用できません。(BZ#2056420)
  - 回避策として、非特権ユーザーは、仮想マシンウィザードを使用して仮想マシンを作成する間に追加のネットワークインターフェイスを追加できます。
- 非特権ユーザーは、RBAC ルールにより仮想マシンにディスクを追加できません。(BZ#2056421)
  - 回避策として、特定のユーザーがディスクを追加できるように RBAC ルールを手動で追加します。
- Web コンソールには、カスタム namespace にデプロイされた仮想マシンテンプレートが表示されません。Web コンソールには、デフォルトの namespace にデプロイされたテンプレートしか表示されません。(BZ#2054650)
  - 回避策として、カスタム namespace にテンプレートをデプロイすることは避けてください。
- Single Node OpenShift (SNO) クラスターでは、VMI の **spec.evictionStrategy** フィールドが **LiveMigrate** に設定されている場合、クラスターの更新が失敗します。ライブマイグレーションを成功させるには、クラスターに複数のワーカー ノードが必要です。(BZ#2073880)
  - 次の 2 つの回避策があります。
    - 仮想マシン宣言から **spec.evictionStrategy** フィールドを削除します。
    - OpenShift Container Platform を更新する前に、仮想マシンを手動で停止します。

第4章 INSTALLING

4.1. 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、または支援付き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。

単一ノードの Openshift の違い

単一ノードのクラスターに OpenShift Virtualization をインストールできます。詳細は、単一ノードの Openshift について を参照してください。単一ノードの Openshift は高可用性をサポートしていないため、次の違いがあります。

- Pod disruption budgets はサポートされていません。
- ライブマイグレーション には対応していません。
- 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テンプレートまたは仮想マシンには、**evictionStrategy** が設定されていてはなりません。

FIPS モード

クラスターを FIPS モード でインストールする場合、OpenShift Virtualization に追加の設定は必要ありません。

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**)

4.1.1. 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) 8 でサポート
- 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化されている。
- 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。

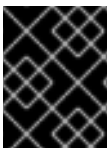
- クラスターが CPU の異なるワーカーノードを使用している場合、CPU ごとに能力が異なるため、ライブマイグレーションの失敗が発生する可能性があります。このような失敗を回避するには、各ノードに適切な能力の CPU を使用し、仮想マシンにノードアフィニティーを設定して、移行が成功するようにします。詳細は、必須のノードのアフィニティールールの設定 を参照してください。

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

4.1.2. 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

本書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。

4.1.2.1. 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) + 146 MiB  \
              + 8 MiB * (number of vCPUs) \ 1
             + 16 MiB * (number of graphics devices) 2
```

1 仮想マシンが要求する仮想 CPU の数

2 仮想マシンが要求する仮想グラフィックスカードの数

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。

4.1.2.2. 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。

4.1.2.3. 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ついての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。

4.1.2.4. 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。

4.1.3. 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

4.1.4. 制限されたネットワーク環境

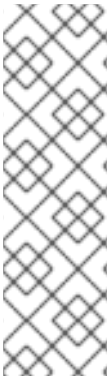
インターネット接続のない制限された環境に OpenShift Virtualization をインストールする場合は、制限されたネットワーク用に Operator Lifecycle Manager を設定 する必要があります。

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。

4.1.5. 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。

4.1.6. スナップショットとクローン作成

スナップショットとクローン作成の要件は、OpenShift Virtualization ストレージ機能 を参照してください。

4.1.7. 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。

- インストーラーによってプロビジョニングされたインフラストラクチャー (IPI) の 自動高可用性は、マシンの可用性チェック 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 RunStrategies 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識別し、セルフノード修復 Operator を使用し

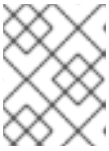
て異常なノードを修復します。

重要

Node Health Check Operator 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。

- 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。

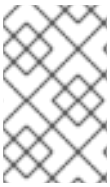


注記

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。

4.2. 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。

4.2.1. 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。

4.2.1.1. 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、一覧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。

4.2.1.2. 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.v4.10.10
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。

4.2.1.3. 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。

4.2.1.4. 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。

4.2.1.5. 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

4.2.2. 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。

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

4.2.2.1.1. 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/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.10.10
  channel: "stable"
  config:
    nodeSelector:
      example.io/example-infra-key: example-infra-value
```

4.2.2.1.2. 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/v1alpha1
kind: Subscription
metadata:
  name: hco-operatorhub
  namespace: openshift-cnv
spec:
  source: redhat-operators
```

```
  sourceNamespace: openshift-marketplace
  name: kubevirt-hyperconverged
  startingCSV: kubevirt-hyperconverged-operator.v4.10.10
  channel: "stable"
  config:
    tolerations:
    - key: "key"
      operator: "Equal"
      value: "virtualization"
      effect: "NoSchedule"
```

4.2.2.2. HyperConverged オブジェクト

4.2.2.2.1. 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
```

4.2.2.2.2. 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
```

4.2.2.2.3. 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"
```

4.2.2.3. HostPathProvisioner オブジェクト

4.2.2.3.1. 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
```

4.3. 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 4.10 web console 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。

4.3.1. 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 4.10 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** フィールドに **OpenShift Virtualization** を入力します。
3. **OpenShift Virtualization** タイルを選択します。
4. Operator についての情報を確認してから、**Install** をクリックします。
5. **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お勧めします。
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、お勧めできません。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。

4.3.2. 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。

4.4. 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。

4.4.1. 前提条件

- OpenShift Container Platform 4.10 をクラスターにインストールしていること。
- OpenShift 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。

4.4.2. 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.v4.10.10
  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。

4.4.3. 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.v4.10.10   OpenShift Virtualization   4.10.10
Succeeded
```

4.4.4. 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。

4.5. VIRTCTL クライアントの有効化

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。これは、Linux、macOS、および Windows ディストリビューションで利用できます。

4.5.1. virtctl クライアントのダウンロードおよびインストール

4.5.1.1. virtctl クライアントのダウンロード

ConsoleCLIDownload カスタムリソース (CR) で提供されるリンクを使用して **virtctl** クライアントをダウンロードします。

手順

1. 以下のコマンドを実行して **ConsoleCLIDownload** オブジェクトを表示します。

```
$ oc get ConsoleCLIDownload virtctl-clidownloads-kubevirt-hyperconverged -o yaml
```

2. お使いのディストリビューションに一覧表示されているリンクを使用して **virtctl** クライアントをダウンロードします。

4.5.1.2. virtctl クライアントのインストール

オペレーティングシステムに適した場所からダウンロードした後に、**virtctl** クライアントを展開し、インストールします。

前提条件

- **virtctl** クライアントをダウンロードしている。

手順

- Linux の場合
  1. tarball を展開します。以下の CLI コマンドは、tarball と同じディレクトリーに展開します。

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

  2. 展開したフォルダー階層に移動し、以下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 <virtctl-file-name>
```

  3. **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。

4. パスを確認するには、以下のコマンドを実行します。

```
$ echo $PATH
```

- Windows ユーザーの場合:

1. アーカイブを展開し、解凍します。
2. 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。
3. **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。
4. パスを確認するには、以下のコマンドを実行します。

```
C:\> path
```

- macOS ユーザーの場合:

1. アーカイブを展開し、解凍します。
2. **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。
3. パスを確認するには、以下のコマンドを実行します。

```
echo $PATH
```

4.5.2. その他の設定オプション

4.5.2.1. yum ユーティリティーを使用した virtctl クライアントのインストール

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

手順

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

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

4.5.2.2. OpenShift Virtualization リポジトリーの有効化

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

- Red Hat Enterprise Linux 8 リポジトリー: **cnv-4.10-for-rhel-8-x86_64-rpms**
- Red Hat Enterprise Linux 7 リポジトリー: **rhel-7-server-cnv-4.10-rpms**

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

手順

- 以下のコマンドを実行して、お使いのシステムに適した OpenShift Virtualization リポジトリーを有効にします。

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

4.5.3. 関連情報

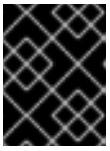
- OpenShift Virtualization の CLI ツールの使用。

4.6. WEB コンソールを使用した OPENSHIFT VIRTUALIZATION のアンインストール

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

4.6.1. 前提条件

- OpenShift Virtualization 4.10 がインストールされていること。
- すべての 仮想マシン、仮想マシンインスタンス、および データボリューム を削除する必要がある。



重要

これらのオブジェクトを削除せずに OpenShift Virtualization のアンインストールを試みると失敗します。

4.6.2. OpenShift Virtualization Operator Deployment 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、まず **OpenShift Virtualization Operator Deployment** カスタムリソースを削除する必要がある。

前提条件

- **OpenShift Virtualization Operator Deployment** カスタムリソースを作成すること。

手順

1. OpenShift Container Platform Web コンソールから、**Projects** 一覧より **openshift-cnv** を選択します。
2. **Operators → Installed Operators** ページに移動します。
3. **OpenShift Virtualization** をクリックします。
4. **OpenShift Virtualization Operator Deployment** タブをクリックします。
5. Options メニュー  を **kubevirt-hyperconverged** カスタムリソースを含む行でクリックします。拡張されたメニューで、**Delete HyperConverged Cluster** をクリックします。
6. 確認ウィンドウで **Delete** をクリックします。

7. **Workloads → Pods** ページに移動し、Operator Pod のみが実行中であることを確認します。
8. ターミナルウィンドウを開き、以下のコマンドを実行して残りのリソースをクリーンアップします。

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

4.6.3. OpenShift Virtualization カタログサブスクリプションの削除

OpenShift Virtualization のアンインストールを終了するには、**OpenShift Virtualization** カタログサブスクリプションを削除します。

前提条件

- **OpenShift Virtualization** カタログの有効なサブスクリプション。

手順

1. **Operators → OperatorHub** ページに移動します。
2. **OpenShift Virtualization** を検索し、これを選択します。
3. **Uninstall** をクリックします。

注記

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

4.6.4. Web コンソールを使用した namespace の削除

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

注記

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

手順

1. **Administration → Namespaces** に移動します。
2. namespace の一覧で削除する必要のある namespace を見つけます。
3. namespace の一覧の右端で、Options メニュー  から **Delete Namespace** を選択します。
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。

4.7. CLI を使用した OPENSHIFT VIRTUALIZATION のアンインストール

OpenShift Container Platform CLI を使用して OpenShift Virtualization をアンインストールできます。

4.7.1. 前提条件

- OpenShift Virtualization 4.10 がインストールされていること。
- すべての 仮想マシン、仮想マシンインスタンス、および データボリューム を削除する必要がある。

重要

これらのオブジェクトを削除せずに OpenShift Virtualization のアンインストールを試みると失敗します。

4.7.2. OpenShift Virtualization の削除

CLI を使用して OpenShift Virtualization を削除できます。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Virtualization クラスターにアクセスできる。

注記

CLI を使用して OLM で OpenShift Virtualization Operator のサブスクリプションを削除すると、**ClusterServiceVersion** (CSV) オブジェクトはクラスターから削除されません。OpenShift Virtualization を完全にアンインストールするには、CSV を明示的に削除する必要があります。

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. Operator Lifecycle Manager (OLM) で OpenShift Virtualization Operator の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 の Cluster Service Version (CSV) 名を環境変数として設定します。

```
$ CSV_NAME=$(oc get csv -n openshift-cnv -o=jsonpath="{.items[0].metadata.name}")
```

4. 直前の手順で CSV 名を指定して、OpenShift Virtualization クラスターから CSV を削除します。

```
$ oc delete csv ${CSV_NAME} -n openshift-cnv
```

OpenShift Virtualization は、CSV が正常に削除されたことを示す確認メッセージが表示される際にアンインストールされます。

出力例

```
clusterserviceversion.operators.coreos.com "kubevirt-hyperconverged-operator.v4.10.10"
deleted
```

第5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。

5.1. 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。

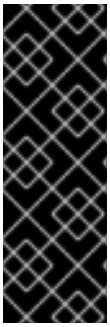
- サブスクリプションの承認ストラテジーが **Automatic** に設定されている場合に、更新プロセスは、Operator の新規バージョンが **stable** チャネルで利用可能になるとすぐに開始します。サポート可能な環境を確保するために、**自動** 承認ストラテジーを使用することを強く推奨します。OpenShift Virtualization の各マイナーバージョンは、対応する OpenShift Container Platform バージョンを実行する場合にのみサポートされます。たとえば、OpenShift Virtualization 4.10 は OpenShift Container Platform 4.10 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。

- Open Shift 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 (PVC) は更新時に保持されます。



重要

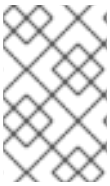
ホストパスプロビジョナーストレージを使用する仮想マシンを実行している場合、それらをライブマイグレーションすることはできず、Open Shift Container Platform クラスターの更新をブロックする可能性があります。

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。

5.2. ワークロードの自動更新の設定

5.2.1. 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

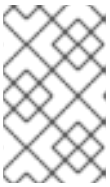
各試行は、移行オブジェクトに対応します。直近の 5 回の試行のみがバッファーに保持されます。これにより、デバッグ用の情報を保持しながら、移行オブジェクトがシステムに蓄積されるのを防ぎます。

5.2.2. 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。

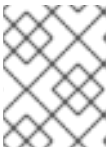
2 中断を最小限に抑えた更新メソッド。ライブマイグレーションをサポートする VMI は、仮想マシン (VM) ゲストを更新されたコンポーネントが有効なっている新規 Pod に移行することで更新されます。**LiveMigrate** がリストされている唯一のワークロード更新メソッドである場合には、ライブマイグレーションをサポートしない VMI は中断または更新されません。

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

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。

5.3. 保留中の OPERATOR 更新の承認

5.3.1. 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、**アップグレードステータス** 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。

5.4. 更新ステータスの監視

5.4.1. 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すること。
- 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
```

5.4.2. 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一覧を表示できます。

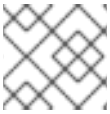
注記

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。

5.5. 関連情報

- 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6章 KUBEVIRT-CONTROLLER および VIRT-LAUNCHER に付与される追加のセキュリティー権限

kubevirt-controller および virt-launcher Pod には、通常の Pod 所有者の権限に加えて一部の SELinux ポリシーおよび SCC (Security Context Constraints) 権限が付与されます。これらの権限により、仮想マシンは OpenShift Virtualization 機能を使用できます。

6.1. VIRT-LAUNCHER POD の拡張 SELINUX ポリシー

virt-launcher Pod の **container_t** SELinux ポリシーが拡張され、OpenShift 仮想化の重要な機能が有効になります。

- ネットワークマルチキューには次のポリシーが必要です。これにより、使用可能な vCPU の数が増加するにつれてネットワークパフォーマンスを拡張できます。
  - **allow process self (tun_socket (relabelfrom relabelto attach_queue))**
- 次のポリシーによって、**virt-launcher** が **/proc/cpuinfo** および **/proc/uptime** を含む **/proc** ディレクトリーの下のファイルを読み取ることができます。
  - **allow process proc_type (file (getattr open read))**
- 次のポリシーによって、**libvirtd** がネットワーク関連のデバッグメッセージをリレーできます。
  - **allow process self (netlink_audit_socket (nlmsg_relay))**

注記

このポリシーがない場合、ネットワークデバッグメッセージをリレーしようとする試みはブロックされます。これにより、ノードの監査ログが SELinux 拒否でいっぱいになる可能性があります。

- 次のポリシーによって、**libvirtd** が **hugetblfs** にアクセスできます。これは、巨大なページをサポートするために必要です。
  - **allow process hugetlbfs_t (dir (add_name create write remove_name rmdir setattr))**
  - **allow process hugetlbfs_t (file (create unlink))**
- 次のポリシーによって、**virtiofs** がファイルシステムをマウントし、NFS にアクセスできます。
  - **allow process nfs_t (dir (mounton))**
  - **allow process proc_t (dir (mounton))**
  - **allow process proc_t (filesystem (mount unmount))**

6.2. 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 の実行についての条件の一覧を定義することができます。

kubevirt-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 virt-launcher 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。

6.2.1. kubevirt-controller サービスアカウントに付与される追加の SCC

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{"NET_ADMIN", "NET_RAW", "SYS_NICE"}**
  This provides the following additional Linux capabilities **NET_ADMIN**, **NET_RAW**, and **SYS_NICE**.

6.2.2. 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
```

6.3. 関連情報

- SSC (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
- Red Hat Enterprise Linux (RHEL) ドキュメントの 仮想マシンネットワークパフォーマンスの最適化
- 仮想マシンでの Huge Page の使用
- RHEL ドキュメントの huge ページの設定

第7章 CLI ツールの使用

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

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

7.1. 前提条件

- **virtctl** クライアントを有効にする必要がある。

7.2. OPENSHIFT CONTAINER PLATFORM クライアントコマンド

OpenShift Container Platform **oc** クライアントは、**VirtualMachine** (**vm**) および **VirtualMachineInstance** (**vmi**) オブジェクトタイプを含む、OpenShift Container Platform リソースを管理するためのコマンドラインユーティリティーです。

注記

-n <namespace> フラグを使用して、別のプロジェクトを指定できます。

表7.1 oc コマンド

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

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

7.3. VIRTCTL クライアントコマンド

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。

virtctlコマンドのリストを表示するには、次のコマンドを実行します。

```
$ virtctl help
```

特定のコマンドで使用できるオプションの一覧を表示するには、これを **-h** または **--help** フラグを指定して実行します。以下に例を示します。

```
$ virtctl image-upload -h
```

任意の**virtctl**コマンドで使用できるグローバルコマンドオプションのリストを表示するには、次のコマンドを実行します。

```
$ virtctl options
```

以下の表には、OpenShift Virtualization のドキュメント全体で使用されている **virtctl** コマンドが記載されています。

表7.2 virtctl クライアントコマンド

コマンド	Description
virtctl start <vm_name>	仮想マシンを起動します。
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。
virtctl pause vm|vmi <object_name>	仮想マシンまたは仮想マシンインスタンスを一時停止します。マシンの状態がメモリーに保持されます。
virtctl unpause vm|vmi <object_name>	仮想マシンまたは仮想マシンインスタンス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。
virtctl expose <vm_name>	仮想マシンまたは仮想マシンインスタンスの指定されたポートを転送するサービスを作成し、このサービスをノードの指定されたポートで公開します。
virtctl console <vmi_name>	仮想マシンインスタンスのシリアルコンソールに接続します。

コマンド	Description
virtctl vnc --kubeconfig=$KUBECONFIG <vmi_name>	VNC (仮想ネットワーククライアント) の仮想マシンインスタンスへの接続を開きます。ローカルマシンでリモートビューアーを必要とする VNC を使用して仮想マシンインスタンスのグラフィカルコンソールにアクセスします。
virtctl vnc --kubeconfig=$KUBECONFIG --proxy-only=true <vmi-name>	VNC 接続からビューアーを使用してポート番号を表示し、仮想マシンインスタンスに手動で接続します。
virtctl vnc --kubeconfig=$KUBECONFIG --port=<port-number> <vmi-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。
virtctl image-upload dv <datavolume_name> --image-path=</path/to/image> --no-create	仮想マシンイメージをすでに存在する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>	仮想マシンイメージを新規データボリューム にアップロードします。
virtctl version	クライアントおよびサーバーのバージョン情報を表示します。
virtctl fslist <vmi_name>	ゲストマシンで利用可能なファイルシステムの詳細な一覧を返します。
virtctl guestosinfo <vmi_name>	オペレーティングシステムに関するゲストエージェント情報を返します。
virtctl userlist <vmi_name>	ゲストマシンでログインしているユーザーの詳細な一覧を返します。

7.4. VIRTCTL GUESTFS を使用したコンテナーの作成

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。

7.5. LIBGUESTFS ツールおよび VIRTCTL GUESTFS

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一覧を表示するには、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。

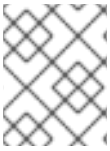
コマンド	Description
virt-edit -a /dev/vda /etc/motd	ターミナルでファイルを対話的に編集します。
virt-customize -a /dev/vda --ssh-inject root:string:<public key example>	ゲストに ssh キーを挿入し、ログインを作成します。
virt-df -a /dev/vda -h	仮想マシンによって使用されるディスク容量を確認します。
virt-customize -a /dev/vda --run-command 'rpm -qa > /rpm-list'	詳細の一覧を含む出力ファイルを作成して、ゲストにインストールされたすべての RPM の詳細一覧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 の出力ファイルの一覧を表示します。
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	Description
--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。

フラグオプション	Description
--image string	**libguestfs-tools** コンテナーイメージを一覧表示します。 **--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。
--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。

7.6. 関連情報

- Libguestfs: 仮想マシン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。

第8章 仮想マシン

8.1. 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。

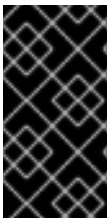
- クイックスタートのガイド付きツアー
- ウィザードの実行
- 仮想マシンウィザードによる事前に設定された YAML ファイルの貼り付け
- CLI の使用

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。仮想マシンにブートソースを追加する 手順を完了すれば、これらのテンプレートを使用することができます。



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。

8.1.1. 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。

8.1.2. 仮想マシンウィザードの実行による仮想マシンの作成

Web コンソールは、仮想マシンテンプレートの選択と仮想マシンの作成プロセスをガイドするウィザードを特長としています。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。テンプレートがブートソースで設定される場合、それらのテンプレートにはカスタマイズされたラベルテキストテキストまたはデフォルトのラベルテキスト (**Available boot source**) のラベルが付けられます。その後、これらのテンプレートは仮想マシンの作成に使用する準備が整います。

事前に設定されたテンプレートの一覧からテンプレートを選択し、設定を確認し、 **Create virtual machine from template** ウィザードで仮想マシンを作成できます。仮想マシンのカスタマイズを選択した場合には、ウィザードが **General**、**Networking**、**Storage**、**Advanced**、および **Review** の手順をガイドします。ウィザードに表示されるすべての必須フィールドには * のマークが付けられます。

ネットワークインターフェイスコントローラー (NIC) およびストレージディスクを作成し、それらを仮想マシンに割り当てます。

手順

1. サイドメニューから **Workloads → Virtualization** をクリックします。
2. **Virtual Machines** タブまたは **Templates** タブで、**Create** をクリックし、**Virtual Machine with Wizard** を選択します。
3. ブートソースで設定したテンプレートを選択します。
4. **Next** をクリックして **Review and create** ステップに移動します。
5. 仮想マシンをすぐに起動する必要がない場合は、**Start this virtual machine after creation** チェックボックスをクリアします。
6. **Create virtual machine** をクリックし、ウィザードを終了するか、ウィザードを継続して使用し、仮想マシンをカスタマイズします。
7. **Customize virtual machine** をクリックして **General** ステップに移動します。
   a. オプション: **Name** フィールドを編集して、仮想マシンのカスタム名を指定します。
   b. オプション: **Description** フィールドに説明を追加します。

8. **Next** をクリックして **Networking** ステップに進みます。デフォルトで **nic0** NIC が割り当てられます。

   a. オプション: **Add Network Interface** をクリックし、追加の NIC を作成します。

   b. オプション: すべての NIC の削除は、Options メニュー をクリックし、**Delete** を選択して実行できます。仮想マシンの作成において、NIC が割り当てられている必要はありません。NIC は仮想マシンの作成後に作成することができます。

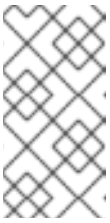
9. **Next** をクリックして **Storage** ステップに進みます。

   a. オプション: **Add Disk** をクリックして追加のディスクを作成します。これらのディスクの削除は、Options メニュー をクリックし、**Delete** を選択して実行できます。

   b. オプション: Options メニュー をクリックし、ディスクを編集して変更内容を保存します。

10. **Next** をクリックして **Advanced** ステップに移動し、以下のいずれかのオプションを選択します。

    a. Linux テンプレートを選択して仮想マシンを作成した場合は、**Cloud-init** の詳細を確認し、SSH アクセスを設定します。

    注記

    cloud-init またはウィザードでカスタムスクリプトを使用して、SSH キーを静的に挿入します。これにより、仮想マシンを安全に、かつリモートで管理し、情報を管理し、転送することができます。この手順は、仮想マシンのセキュリティーを保護するために実行することを強く推奨します。

    b. Windows テンプレートを選択して仮想マシンを作成した場合、**SysPrep** セクションを使用して、自動化された Windows 設定用に XML 形式で回答ファイルをアップロードします。

11. **Next** をクリックして **Review** ステップに移動し、仮想マシンの設定を確認します。

12. **Create Virtual Machine** をクリックします。

13. **See virtual machine details** をクリックして、この仮想マシンの **Overview** を表示します。仮想マシンは **Virtual Machines** タブに一覧表示されます。

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

8.1.2.1. 仮想マシンウィザードのフィールド

名前	パラメーター	説明

名前	パラメーター	説明
名前		この名前には、小文字 (**a-z**)、数字 (**0-9**)、およびハイフン (**-**) を含めることができ、最大 253 文字を使用できます。最初と最後の文字は英数字にする必要があります。この名前には、大文字、スペース、ピリオド (**.**)、または特殊文字を使用できません。
説明		オプションの説明フィールド。
オペレーティングシステム		テンプレートで仮想マシン用に選択されるオペレーティングシステム。テンプレートから仮想マシンを作成する場合、このフィールドを編集することはできません。
Boot Source	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。例: オペレーティングシステムイメージのある Web ページから URL リンクを取得します。
	クローン (PVC を作成)	クラスターで利用可能な既存の永続ボリューム要求 (PVC) を選択し、これをクローンします。
	レジストリー (PVC を作成)	クラスターからアクセスできるレジストリーの起動可能なオペレーティングシステムコンテナーから仮想マシンをプロビジョニングします。例: **kubevirt/cirros-registry-disk-demo**
	PXE (ネットワークブート: ネットワークインターフェイスの追加)	ネットワークのサーバーからオペレーティングシステムを起動します。PXE ブート可能なネットワーク接続定義が必要です。
永続ボリューム要求 (PVC) のプロジェクト		PVC のクローン作成に使用するプロジェクト名。
永続ボリューム要求 (PVC) の名前		既存の PVC のクローンを作成する場合にこの仮想マシンテンプレートに適用する必要のある PVC 名。

名前	パラメーター	説明
これを CD-ROM ブートソースとしてマウントする		CD-ROM には、オペレーティングシステムをインストールするための追加のディスクが必要です。チェックボックスを選択して、ディスクを追加し、後でカスタマイズします。
Flavor	Tiny、Small、Medium、Large、Custom	仮想マシンテンプレートの CPU およびメモリーの容量を、そのテンプレートに関連付けられたオペレーティングシステムに応じて、仮想マシンに割り当てられる事前に定義された値で事前設定します。 デフォルトのテンプレートを選択する場合は、カスタム値を使用して、テンプレートの **cpus** および **memsize** の値を上書きしてカスタムテンプレートを作成できます。または、**Template details** ページの **General** タブで **cpus** と **memsize** の値を変更して、カスタムテンプレートを作成することもできます。
Workload Type **注記** 誤った **Workload Type** を選択した場合は、パフォーマンスまたはリソースの使用状況の問題が発生することがあります (UI の速度低下など)。	デスクトップ	デスクトップで使用するための仮想マシン設定。小規模な環境での使用に適しています。Web コンソールでの使用に推奨されます。このテンプレートクラスまたはサーバーテンプレートクラスを使用して、**保証された** 仮想マシンのパフォーマンスよりも VM の密度を優先します。
	Server	パフォーマンスのバランスを図り、さまざまなサーバーのワークロードと互換性があります。このテンプレートクラスまたはデスクトップテンプレートクラスを使用して、**保証された** 仮想マシンのパフォーマンスよりも VM の密度を優先します。
	高パフォーマンス (CPU マネージャーが必要)	高パフォーマンスのワークロードに対して最適化された仮想マシン設定。このテンプレートクラスを使用して、仮想マシンの密度よりも **保証された** 仮想マシンのパフォーマンスを優先します。

名前	パラメーター	説明
作成後にこの仮想マシンを起動します。		このチェックボックスはデフォルトで選択され、仮想マシンは作成後に実行を開始します。仮想マシンの作成時に起動する必要がない場合は、チェックボックスをクリアします。

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。

8.1.2.1.1. ネットワークフィールド

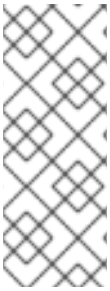
Name	Description
Name	ネットワークインターフェイスコントローラーの名前。
モデル	ネットワークインターフェイスコントローラーのモデルを示します。サポートされる値は **e1000e** および **virtio** です。
ネットワーク	利用可能なネットワーク接続定義の一覧。
Type	利用可能なバインディングメソッドの一覧。ネットワークインターフェイスに適したバインド方法を選択します。 • デフォルトの Pod ネットワーク: **masquerade** • Linux ブリッジネットワーク: **bridge** • SR-IOV ネットワーク: **SR-IOV**
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。

8.1.2.2. 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。

Name	選択	Description
	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)。
Type		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)	ボリュームは単一ノードで読み取り/書き込みとしてマウントできます。
		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。

8.1.2.3. Cloud-init フィールド

Name	説明
Hostname	仮想マシンの特定のホスト名を設定します。

Name	説明
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。

8.1.2.4. 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** ページに一覧表示されます。

8.1.3. 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8.1 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>
```

8.1.4. 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。

8.1.5. 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。

8.1.6. 関連情報

- KubeVirt v0.49.0 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のより範囲の広い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。

- **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。
- マシンのヘルスチェックの導入と有効化の詳細は、マシンの可用性チェックの導入 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

8.2. 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。

8.2.1. Web コンソールでの仮想マシンの編集

関連するフィールドの横にある鉛筆アイコンをクリックして、Web コンソールで仮想マシンの選択する値 (select values) を編集します。他の値は、CLI を使用して編集できます。

ラベルとアノテーションは、事前に設定された Red Hat テンプレートとカスタム仮想マシンテンプレートの両方について編集されます。その他のすべての値は、ユーザーが Red Hat テンプレートまたは **Create Virtual Machine Template**ウィザードを使用して作成したカスタム仮想マシンテンプレートについてのみ編集されます。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. オプション: **Filter** ドロップダウン メニューを使用して、ステータス、テンプレート、ノード、またはオペレーティング システム (OS) などの属性で仮想マシンのリストを並べ替えます。
3. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
4. 鉛筆アイコンをクリックして、フィールドを編集可能にします。
5. 関連する変更を加え、**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一覧が表示されます。

8.2.1.1. 仮想マシンフィールド

以下の表には、OpenShift Container Platform Web コンソールで編集できる仮想マシンのフィールドが記載されています。

表8.1 仮想マシンフィールド

タブ	フィールドまたは機能

タブ	フィールドまたは機能
Details	• ラベル • アノテーション • 説明 • CPU/メモリー • 起動モード • ブート順序 • GPU デバイス • ホストデバイス • SSH アクセス
YAML	• カスタムリソースを表示、編集、またはダウンロードします。
スケジューリング	• Node selector • 容認 • アフィニティールール • 専用リソース • エビクションストラテジー • Descheduler 設定
Network Interfaces	• ネットワークインターフェイスを追加、編集、または削除します。
ディスク	• ディスクを追加、編集、または削除します。
スクリプト	• cloud-init 設定
スナップショット	• 仮想マシンのスナップショットを追加、復元、または削除します。

8.2.2. 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。

8.2.3. 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>
```

8.2.4. 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** 画面を開きます。
3. **Disks** タブをクリックし、**Add disk** をクリックします。
4. **Add disk** ウィンドウで、**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。

8.2.4.1. VirtualMachine の CD-ROM の編集

以下の手順を使用して、仮想マシンの CD-ROM を編集します。

手順

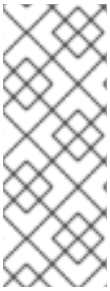
1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** 画面を開きます。
3. **Disks** タブをクリックします。
4. 編集する CD-ROM の Options メニュー をクリックし、**Edit** を選択します。
5. **Edit CD-ROM** ウィンドウで、**Source**、**Persistent Volume Claim**、**Name**、**Type**、および **Interface** フィールドを編集します。

6. **Save** をクリックします。

8.2.4.2. 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)。
Type		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。

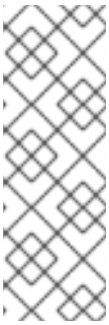
ストレージの詳細設定
以下のストレージの詳細設定はオプションであり、**Blank**、**Import via URLURL**、および **Clone existing PVC** ディスクで利用できます。OpenShift Virtualization 4.11 より前では、これらのパラメーターを指定しない場合、システムは **kubevirt-storage-class-defaults** 設定マップのデフォルト値を使用します。OpenShift Virtualization 4.11 以降では、システムは ストレージプロファイル のデフォルト値を使用します。



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)	ボリュームは単一ノードで読み取り/書き込みとしてマウントできます。
		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。

8.2.5. 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** 画面を開きます。
3. **Network Interfaces** タブをクリックします。
4. **Add Network Interface** をクリックします。
5. **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。
6. **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一覧が表示されます。

8.2.5.1. ネットワークフィールド

Name	Description
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。
ネットワーク	利用可能なネットワーク接続定義の一覧。
Type	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**

Name	Description
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。

8.2.6. 関連情報

- ストレージプロファイルのカスタマイズ

8.3. 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。

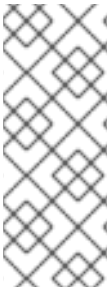
- ディスクまたはネットワークインターフェイスコントローラー (NIC) を選択し、これをブート順序の一覧に追加します。
- ブート順序の一覧でディスクまたは NIC の順序を編集します。
- ブート順序の一覧からディスクまたは NIC を削除して、起動可能なソースのインベントリーに戻します。

8.3.1. Web コンソールでのブート順序一覧への項目の追加

Web コンソールを使用して、ブート順序一覧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、これがブート順序一覧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一覧が表示されます。

8.3.2. Web コンソールでのブート順序一覧の編集

Web コンソールで起動順序一覧を編集します。

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. ブート順序一覧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、ブート順序一覧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一覧が表示されます。

8.3.3. YAML 設定ファイルでのブート順序一覧の編集

CLI を使用して、YAML 設定ファイルのブート順序の一覧を編集します。

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一覧に適用します。

8.3.4. Web コンソールでのブート順序一覧からの項目の削除

Web コンソールを使用して、ブート順序の一覧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一覧から削除され、利用可能なブートソースの一覧に保存されます。ブート順序一覧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一覧が表示されます。

8.4. 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。

8.4.1. 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。

注記

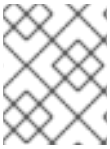
仮想マシンを削除する際に、これが使用するデータボリュームは自動的に削除されます。

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. 削除する仮想マシンの 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. 確認のポップアップウィンドウで、**Delete** をクリックし、仮想マシンを永続的に削除します。

8.4.2. 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各種のアクションを実行できます。



注記

仮想マシンを削除する際に、これが使用するデータボリュームは自動的に削除されます。

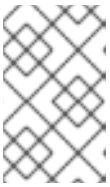
前提条件

- 削除する仮想マシンの名前を特定すること。

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に存在するオブジェクトのみを削除します。削除する必要のあるオブジェクトが別のプロジェクトまたは namespace にある場合、**-n <project_name>** オプションを指定します。

8.5. 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。

8.5.1. 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 とそれらの詳細を一覧表示します。
- 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 の一覧を表示します。次に、未処理の VMI を削除します。

8.5.2. CLI を使用した仮想マシンインスタンスの一覧表示

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一覧を表示できます。

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
```

8.5.3. 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一覧表示

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) の一覧を表示できます。

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 を一覧表示するには、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。

8.5.4. 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。

8.5.5. 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 の名前を特定すること。

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>
```

8.5.6. 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. 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. **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。

8.6. 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。

8.6.1. 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. ユースケースに適した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。

8.6.2. 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. ユースケースに適した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。

8.6.3. 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. ユースケースに適した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。

8.6.4. 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. ユースケースに適した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。

8.7. 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。

8.7.1. 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。

8.7.1.1. 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。

8.7.1.2. 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。

8.7.1.3. RDP を使用した Windows 仮想マシンへの接続

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

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

前提条件

- QEMU ゲストエージェントがインストールされた実行中の Windows 仮想マシン。**qemu-guest-agent** は VirtIO ドライバーに含まれています。
- 仮想マシンに接続された layer-2 NIC。
- 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. **Console** 一覧で、 **Desktop Viewer** を選択します。
5. **Network Interface** 一覧で、 layer-2 NIC を選択します。
6. **Launch Remote Desktop** をクリックし、 **console.rdp** ファイルをダウンロードします。
7. RDP クライアントを開き、**console.rdp** ファイルを参照します。たとえば、**remmina** を使用します。

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

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

8.7.2. 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

8.7.2.1. SSH 経由での仮想マシンインスタンスへのアクセス

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

virtctl expose コマンドは、仮想マシンインスタンス (VMI) のポートをノードポートに転送し、有効にされたアクセスのサービスを作成します。以下の例では、**fedora-vm-ssh** サービスを作成します。このサービスは、クラスターノードの特定のポートから **<fedora-vm>** 仮想マシンのポート 22 にトラフィックを転送します。

前提条件

- VMI と同じプロジェクトを使用する。
- アクセスする VMI は、**masquerade** バインディング方法を使用してデフォルトの Pod ネットワークに接続されている。
- アクセスする VMI が実行中であること。
- OpenShift CLI (**oc**) をインストールしている。

手順

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

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

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

2. サービスをチェックし、サービスが取得したポートを見つけます。

```
$ oc get svc
```

-

出力例

```
NAME           TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
fedora-vm-ssh  NodePort   127.0.0.1      <none>        22:32551/TCP     6s
```

この例では、サービスは **32551** ポートを取得しています。

3. SSH 経由で VMI にログインします。クラスターノードの **ipAddress** および直前の手順で確認したポートを使用します。

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

8.7.2.2. YAML 設定を使用した SSH での仮想マシンへのアクセス

virtctl expose コマンドを実行する必要なしに、仮想マシン (VM) への SSH 接続を有効にすることができます。仮想マシンの YAML ファイルおよびサービスの YAML ファイルが設定され、適用されると、サービスは SSH トラフィックを仮想マシンに転送します。

以下の例は、仮想マシンの YAML ファイルおよびサービス YAML ファイルの設定を示しています。

前提条件

- OpenShift CLI (**oc**) をインストールしている。
- **oc create namespace** コマンドを使用し、namespace の名前を指定して仮想マシンの YAML ファイルの namespace を作成します。

手順

1. 仮想マシンの YAML ファイルで、SSH 接続のサービスを公開するためのラベルおよび値を追加します。インターフェイスの **masquerade** 機能を有効にします。

VirtualMachine 定義の例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  namespace: ssh-ns 1
  name: vm-ssh
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-ssh
        special: vm-ssh 2
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: containerdisk
```

```
        - disk:
            bus: virtio
          name: cloudinitdisk
        interfaces:
        - masquerade: {} 3
          name: testmasquerade 4
        rng: {}
      machine:
        type: ""
      resources:
        requests:
          memory: 1024M
    networks:
    - name: testmasquerade
      pod: {}
    volumes:
    - name: containerdisk
      containerDisk:
        image: kubevirt/fedora-cloud-container-disk-demo
    - name: cloudinitdisk
      cloudInitNoCloud:
        userData: |
          #cloud-config
          user: fedora
          password: fedora
          chpasswd: {expire: False}
# ...
```

1. **oc create namespace** コマンドで作成される namespace の名前。
2. SSH トラフィック接続に対して有効にされた仮想マシンインスタンスを識別するためにサービスによって使用されるラベル。ラベルには、この YAML ファイルに **label** として追加される任意の **key:value** ペアを使用でき、サービス YAML ファイルの **selector** として使用できます。
3. インターフェイスタイプは **masquerade** です。
4. このインターフェイスの名前は **testmasquerade** です。

2. 仮想マシンを作成します。

```
$ oc create -f <path_for_the_VM_YAML_file>
```

3. 仮想マシンを起動します。

```
$ virtctl start vm-ssh
```

4. サービスの YAML ファイルで、サービス名、ポート番号、およびターゲットポートを指定します。

Service 定義の例。

```
apiVersion: v1
kind: Service
```

```
metadata:
  name: svc-ssh 1
  namespace: ssh-ns 2
spec:
  ports:
  - targetPort: 22 3
    protocol: TCP
    port: 27017
  selector:
    special: vm-ssh 4
  type: NodePort
# ...
```

1. SSH サービスの名前。
2. **oc create namespace** コマンドで作成される namespace の名前。
3. SSH 接続のターゲットポート番号。
4. セレクター名と値は仮想マシンの YAML ファイルに指定されるラベルと一致する必要があります。

5. サービスを作成します。

```
$ oc create -f <path_for_the_service_YAML_file>
```

6. 仮想マシンが実行されていることを確認します。

```
$ oc get vmi
```

出力例

```
NAME    AGE     PHASE       IP             NODENAME
vm-ssh 6s       Running     10.244.196.152  node01
```

7. サービスをチェックし、サービスが取得したポートを見つけます。

```
$ oc get svc
```

出力例

```
NAME          TYPE       CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE
svc-ssh     NodePort      10.106.236.208 <none>       27017:30093/TCP   22s
```

この例では、サービスはポート番号 30093 を取得しています。

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  v1.23.0  192.168.55.101   <none>
```

9. 仮想マシンが実行されているノードの IP アドレスとポート番号を指定して、SSH 経由で仮想マシンにログインします。**oc get svc** コマンドで表示されるポート番号および **oc get node** コマンドで表示されるノードの IP アドレスを使用します。以下の例は、ユーザー名、ノードの IP アドレス、およびポート番号を指定した **ssh** コマンドを示しています。

```
$ ssh fedora@192.168.55.101 -p 30093
```

8.7.2.3. 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>
```

8.7.2.4. 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
```

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

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

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

前提条件

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

手順

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

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

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

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

出力例

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

```
      mac: 00:17:a4:77:77:25
      name: bridge-net
...
```

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

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

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

8.8. 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。

8.8.1. 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。

8.8.2. 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。

8.8.3. 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。

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. **Disks** タブをクリックします。
4. **sysprep** ディスクの Options メニュー  をクリックし、**Detach** を選択します。
5. **デタッチ** 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。

8.8.4. 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。

8.8.5. 関連情報

- 仮想マシンの作成
- Microsoft、Sysprep (Generalize) a Windows installation
- Microsoft、generalize
- Microsoft、specialize

8.9. 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 (VM) は正常なノードに自動的に移動しません。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。

8.9.1. 前提条件

- 仮想マシンが実行されているノードには **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。

8.9.2. 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。

8.9.3. 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。

8.9.3.1. CLI を使用した仮想マシンインスタンスの一覧表示

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一覧を表示できます。

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
```

8.10. QEMU ゲストエージェントの仮想マシンへのインストール

QEMU ゲストエージェント は、仮想マシンで実行され、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。

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

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

仮想マシン (VM) に QEMU ゲストエージェントがインストールされ、実行されているかどうかを確認するには、**AgentConnected** が VM 仕様に表示されていることを確認します。

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
```

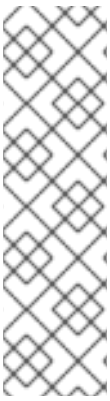
3. サービスに永続性があることを確認し、これを起動します。

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

8.10.2. 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。

仮想マシン (VM) に QEMU ゲストエージェントがインストールされ、実行されているかどうかを確認するには、**AgentConnected** が VM 仕様に表示されていることを確認します。



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。

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

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

注記

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

手順

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

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

9. 仮想マシンを再起動してドライバーのインストールを完了します。

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

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

注記

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

手順

1. 仮想マシンを起動し、グラフィカルコンソールに接続します。

2. Windows インストールプロセスを開始します。

3. **Advanced** インストールを選択します。

4. ストレージの宛先は、ドライバーがロードされるまで認識されません。**Load driver** をクリックします。

5. ドライバーは SATA CD ドライブとして割り当てられます。**OK** をクリックし、CD ドライバーでロードするストレージドライバーを参照します。ドライバーは、ドライバーのタイプ、オペレーティングシステム、および CPU アーキテクチャー別に階層的に編成されます。

6. 必要なすべてのドライバーについて直前の 2 つの手順を繰り返します。

7. Windows インストールを完了します。

8.11. 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。

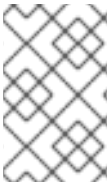
8.11.1. 前提条件

- 仮想マシンに QEMU ゲストエージェントを インストールします。

8.11.2. Web コンソールでの QEMU ゲストエージェント情報について

QEMU ゲストエージェントがインストールされると、**VirtualMachine details** ページの **Overview** タブおよび **Details** タブに、ホスト名、オペレーティングシステム、タイムゾーン、およびログインユーザーに関する情報が表示されます。

VirtualMachine details ページには、仮想マシンにインストールされているゲストオペレーティングシステムに関する情報が表示されます。**Details** タブには、ログインユーザーの情報が含まれる表が表示されます。**Disks** タブには、ファイルシステムの情報が含まれる表が表示されます。



注記

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。

8.11.3. 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。

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. **Disks** タブをクリックして、ファイルシステムについての情報を表示します。

8.12. 仮想マシンでの CONFIG MAP、シークレット、およびサービスアカウントの管理

シークレット、config map、およびサービスアカウントを使用して設定データを仮想マシンに渡すことができます。たとえば、以下を実行できます。

- シークレットを仮想マシンに追加して認証情報を必要とするサービスに仮想マシンのアクセスを付与します。
- Pod または別のオブジェクトがデータを使用できるように、機密データではない設定データを config map に保存します。
- サービスアカウントをそのコンポーネントに関連付けることにより、コンポーネントが API サーバーにアクセスできるようにします。

注記

OpenShift Virtualization はシークレット、設定マップ、およびサービスアカウントを仮想マシンディスクとして公開し、追加のオーバーヘッドなしにプラットフォーム全体でそれらを使用できるようにします。

8.12.1. 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、変更内容は仮想マシンが再起動されるまで反映されません。新規に追加されたリソースは、ページ上部の **Pending Changes** バナーの **Environment** および **Disks** タブの両方に保留中のリソース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. **Environment** タブで、**Add Config Map, Secret or Service Account**をクリックします。
4. **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。
5. 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。
6. **Save** をクリックします。

検証

1. **VirtualMachine details** ページで、**Disks** タブをクリックし、シークレット、config map、またはサービスアカウントがディスクのリストに含ま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。

8.12.2. 仮想マシンからのシークレット、設定マップ、またはサービスアカウントの削除

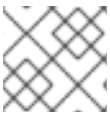
OpenShift Container Platform Web コンソールを使用して、シークレット、設定マップ、またはサービスアカウントを仮想マシンから削除します。

前提条件

- 仮想マシンに割り当てられるシークレット、設定マップ、またはサービスアカウントが少なくとも 1 つ必要である。

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. **Environment** タブをクリックします。
4. 一覧で削除する項目を見つけ、項目の右上にある **Remove** ⛔ をクリックします。
5. **Save** をクリックします。



注記

Reload をクリックし、最後に保存された状態にフォームをリセットできます。

検証

1. **VirtualMachine details** ページで、**Disks** タブをクリックします。
2. 削除したシークレット、設定マップ、またはサービスアカウントがディスクの一覧に含まれていないことを確認します。

8.12.3. 関連情報

- 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
- 設定マップについて

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

8.13.1. VirtIO ドライバーについて

VirtIO ドライバーは、Microsoft Windows 仮想マシンが 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。

新しい Windows 仮想マシンに Virtio ドライバーをインストールする も参照してください。

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

表8.2 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	Description
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。

ドライバー名	ハードウェア ID	Description
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。

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

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。

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

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

注記

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

手順

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

8.13.5. 仮想マシンからの VirtIO コンテナーディスクの削除

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

手順

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

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

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

2. 変更を有効にするために仮想マシンを再起動します。

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

8.14.1. 前提条件

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

8.14.2. 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。

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

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

表8.3 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	Description
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	ブロックドライバー。**Other devices** グループの **SCSI Controller** として表示される場合があります。

ドライバー名	ハードウェア ID	Description
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。

8.14.4. 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。

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

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

注記

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

手順

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

8.14.6. 仮想マシンからの VirtIO コンテナーディスクの削除

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

手順

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

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

```
spec:
  domain:
```

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

2. 変更を有効にするために仮想マシンを再起動します。

8.15. 高度な仮想マシン管理

8.15.1. 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

8.15.1.1. 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。

8.15.1.2. 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

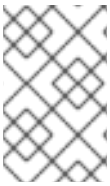
8.15.2. 仮想マシンのノードの指定

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。

8.15.2.1. 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。

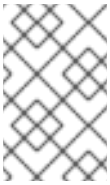
以下のルールタイプは、**VirtualMachine** マニフェストの **spec** フィールドで使用できます。

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、一覧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。

8.15.2.2. 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。

8.15.2.2.1. 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
...
```

8.15.2.2.2. 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:
  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。

8.15.2.2.3. 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:
  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。

8.15.2.2.4. 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"
...
```

8.15.2.3. 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

8.15.3. 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。

8.15.3.1. 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。

8.15.3.2. 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。

8.15.4. 管理タスクの自動化

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

8.15.4.1. Red Hat Ansible Automation について

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

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

8.15.4.2. 仮想マシン作成の自動化

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

前提条件

- Red Hat Ansible Engine バージョン 2.8 以降。

手順

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

```
kubevirt_vm:
  namespace:
  name:
  cpu_cores:
  memory:
  disks:
```

```
    - name:
      volume:
        containerDisk:
          image:
      disk:
        bus:
```

注記

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

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

```
kubevirt_vm:
  namespace: default
  name: vm1
  cpu_cores: 1
  memory: 64Mi
  disks:
    - name: containerdisk
      volume:
        containerDisk:
          image: kubevirt/cirros-container-disk-demo:latest
      disk:
        bus: virtio
```

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

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

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

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

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

5. 出力を確認し、プレイが正常に実行されたかどうかを確認します。

出力例

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

PLAY RECAP
```

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

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

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

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

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

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

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

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

追加情報

- Playbook の概要
- Playbook の検証ツール

8.15.5. 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。

8.15.5.1. 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。

8.15.5.2. 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
```

8.15.6. 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。

8.15.6.1. 前提条件

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

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

手順

1. クラスターに PXE ネットワークを設定します。
   a. PXE ネットワーク **pxe-net-conf** の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

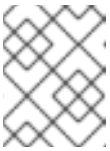
仮想マシンインスタンスは、必要な VLAN のアクセスポートでブリッジ **br1** に割り当てられます。

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
```

8.15.6.3. 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。

8.15.7. 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。

8.15.7.1. 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。

8.15.7.2. 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。

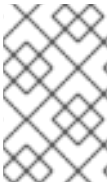
Huge Page とは、4Ki より大きいメモリーページのことです。x86_64 アーキテクチャーでは、2Mi と 1Gi の 2 つが一般的な Huge Page サイズです。別のアーキテクチャーではサイズは異なります。Huge Page を使用するには、アプリケーションが認識できるようにコードを書き込む必要があります。Transparent Huge Pages (THP) は、アプリケーションによる認識なしに、Huge Page の管理を自動化しようとしますが、制約があります。特に、ページサイズは 2Mi に制限されます。THP では、THP のデフラグが原因で、メモリー使用率が高くなり、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。

8.15.7.3. 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ついてのものです。

```
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
```

8.15.8. 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。

8.15.8.1. 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。

8.15.8.2. 前提条件

- CPU マネージャー がノードで設定されている。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。

8.15.8.3. 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. **Scheduling** タブで、**Dedicated Resources** の横にある鉛筆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。

8.15.9. 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。

8.15.9.1. 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	Description
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。

ポリシー属性	Description
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。

8.15.9.2. 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。

8.15.9.3. 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。

8.15.9.4. 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 (VM) 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。

8.15.10. 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。

8.15.10.1. 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** の一覧は空になります。

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。

8.15.10.1.1. 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
```

8.15.10.1.2. 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、これらの値で一覧を作成します。一覧を **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: 4.10.0
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。複数のデバイスの一覧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
```

8.15.10.1.3. 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 デバイスの一覧。
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
```

8.15.10.1.4. 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
```

8.15.10.2. 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。

8.15.10.2.1. 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)
```

8.15.10.3. 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

8.15.11. 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。

8.15.11.1. 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。

8.15.11.2. 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

8.15.12. 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。

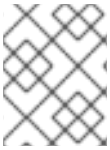
重要

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。

8.15.12.1. 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、および自動ノードラベル付け、監視などを可能にする NVIDIA ドライバーが含まれます。



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 が GPU を OpenShift Container Platform で実行されている仮想化されたワークロードに公開できるようにする Kubernetes Operator です。これにより、ユーザーは GPU 対応の仮想マシンを簡単にプロビジョニン

グおよび管理できるようになり、他のワークロードと同じプラットフォームで複雑な人工知能/機械学習 (AI/ML) ワークロードを実行できるようになります。また、インフラストラクチャーの GPU 容量を簡単にスケーリングできるようになり、GPU ベースのワークロードが急激に増加しても対応できます。

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。

8.15.12.2. 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、Open Shift 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については、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。

8.15.12.2.1. 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。

8.15.12.2.2. 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"))'
```

8.15.12.2.3. 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。

8.15.12.2.4. 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。

8.15.12.2.5. 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。

8.15.12.2.5.1. 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
```

8.15.12.2.6. 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。

8.15.12.2.6.1. 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>
```

8.15.12.2.6.2. 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。

8.15.12.3. 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。

8.15.12.3.1. 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>
```

8.15.12.4. 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

8.15.13. ウォッチドッグの設定

ウォッチドッグデバイスに仮想マシン (VM) を設定し、ウォッチドッグをインストールして、ウォッチドッグサービスを開始することで、ウォッチドッグを公開します。

8.15.13.1. 前提条件

- 仮想マシンで **i6300esb** ウォッチドッグデバイスのカーネルサポートが含まれている。Red Hat Enterprise Linux(RHEL) イメージが、**i6300esb** をサポートしている。

8.15.13.2. ウォッチドッグデバイスの定義

オペレーティングシステム (OS) が応答しなくなるときにウォッチドッグがどのように進行するかを定義します。

表8.4 利用可能なアクション

poweroff	仮想マシン (VM) の電源がすぐにオフになります。**spec.running** が **true** に設定されている場合や、**spec.runStrategy** が **manual** に設定されていない場合には、仮想マシンは再起動します。
reset	VM はその場で再起動し、ゲスト OS は反応できません。ゲスト OS の再起動に必要な時間の長さにより liveness プローブのタイムアウトが生じる可能性があるため、このオプションの使用は推奨されません。このタイムアウトにより、クラスターレベルの保護が liveness プローブの失敗に気づき、強制的に再スケジュールした場合に、VM の再起動にかかる時間が長くなる可能性があります。
shutdown	VM は、すべてのサービスを停止することにより、正常に電源を切ります。

手順

1. 以下の内容を含む 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. **watchdog** アクション (**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終了します。

```
# pkill -9 watchdog
```

8.15.13.3. ウォッチドッグデバイスのインストール

仮想マシンに **watchdog** パッケージをインストールして、ウォッチドッグサービスを起動します。

手順

1. root ユーザーとして、**watchdog** パッケージおよび依存関係をインストールします。

```
# yum install watchdog
```

2. **/etc/watchdog.conf** ファイルの以下の行のコメントを解除して、変更を保存します。

```
#watchdog-device = /dev/watchdog
```

3. ウォッチドッグサービスが起動時に開始できるように有効化します。

```
# systemctl enable --now watchdog.service
```

8.15.13.4. 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

8.15.14. 事前定義済みのブートソースの自動インポートおよび更新

システム定義 で OpenShift Virtualization に含まれるブートソース、または作成した **ユーザー定義** のブートソースを使用できます。システム定義のブートソースのインポートおよび更新は、製品の機能ゲートによって制御されます。機能ゲートを使用して、更新を有効、無効、または再度有効にすることができます。ユーザー定義のブートソースは、製品機能ゲートによって制御されないため、自動インポートおよび更新をオプトインまたはオプトアウトするには、個別に管理する必要があります。

重要

ブートソースの自動インポートおよび更新のために、デフォルトのストレージクラスを設定する必要があります。

8.15.14.1. ブートソースの自動更新の有効化

OpenShift Virtualization 4.9 からの事前定義済みのブートソースがある場合は、手動でブートソースの自動更新を選択する必要があります。OpenShift Virtualization 4.10 以降からのすべての事前定義済みブートソースは、デフォルトで自動的に更新されます。

手順

- 以下のコマンドを使用して **dataImportCron** ラベルをデータソースに適用します。

```
$ oc label --overwrite DataSource rhel8 -n openshift-virtualization-os-images
cdi.kubevirt.io/dataImportCron=true
```

8.15.14.2. ブートソースの自動更新の無効化

非接続環境のログ数を減らしたり、リソースの使用量を減らしたりできます。そのためには、事前定義済みブートソースの自動インポートと更新を無効にします。**HyperConverged** カスタムリソース (CR) の **spec.featureGates.enableCommonBootImageImport** フィールドを **false** に設定します。

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

- 以下のコマンドを使用して自動更新を無効にします。

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

8.15.14.3. ブートソースの自動更新の再有効化

以前にブートソースの自動更新を無効にしている場合は、この機能を手動で再度有効にする必要があります。**HyperConverged** カスタムリソース (CR) の **spec.featureGates.enableCommonBootImageImport** フィールドを **true** に設定します。

手順

- 以下のコマンドを使用して自動更新を再度有効にします。

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

8.15.14.4. カスタムブートソースでの自動更新の有効化

OpenShift Virtualization はデフォルトで事前に定義された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、カスタムブートソースで自動インポートおよび更新を手動で有効にする必要があります。

手順

1. 以下のコマンドを使用して、編集するために **HyperConverged** CR を開きます。

```
$ oc edit -n openshift-cnv HyperConverged
```

2. 適切なテンプレートおよびブートソースを **dataImportCronTemplates** セクションで指定して、**HyperConverged** CR を編集します。以下に例を示します。

CentOS 7 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

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

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。

8.15.15. 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。

8.15.15.1. 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。

8.15.15.2. 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。

前提条件

- クラスター管理者の権限。
- OpenShift Container Platform Web コンソールにアクセスします。

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. **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。

8.15.15.3. 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
```

Descheduler が仮想マシンで有効になりました。

8.15.15.4. 関連情報

- Descheduler を使用した Pod のエビクト

8.16. 仮想マシンのインポート

8.16.1. データボリュームインポートの TLS 証明書

8.16.1.1. 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>
```

8.16.1.2. 例: TLS 証明書から作成される設定マップ

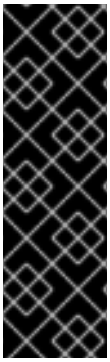
以下は、**ca.pem** TLS 証明書で作成される設定マップの例です。

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

8.16.2. 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

Containerized Data Importer (CDI) を使用し、データボリュームを使用して仮想マシンイメージを永続ボリューム要求 (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。

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。

8.16.2.1. 前提条件

- エンドポイントに TLS 証明書が必要な場合、証明書はデータボリュームと同じ namespace の設定マップ に組み込む 必要があり、これはデータボリューム設定で参照されること。
- 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。

8.16.2.2. 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。

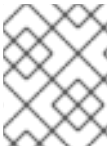
コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
KubeVirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ 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。

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

8.16.2.4. 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:
        resources:
          requests:
            storage: 10Gi
        storageClassName: local
      source:
        http: (3)
          url: "https://mirror.arizona.edu/fedora/linux/releases/35/Cloud/x86_64/images/Fedora-
Cloud-Base-35-1.2.x86_64.qcow2" (4)
          secretRef: endpoint-secret (5)
          certConfigMap: "" (6)
    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) HTTP または HTTPS エンドポイントに **http** を指定します。レジストリーからインポートされたコンテナーディスクイメージの **registry** を指定します。

それたコンテナーディスクイメージの **registry** を指定します。

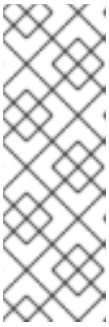
4 インポートする仮想マシンイメージの URL またはレジストリーエンドポイントを指定します。この例では、HTTPS エンドポイントで仮想マシンイメージを参照します。コンテナーレジストリーエンドポイントのサンプルは **url: "docker://kubevirt/fedora-cloud-container-disk-demo:latest"** です。

5 データソースの **Secret** を作成した場合は、**Secret** 名を指定します。

6 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
```

8.16.2.5. 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。

8.16.3. データボリュームの使用による仮想マシンイメージのブロックストレージへのインポート

既存の仮想マシンイメージは OpenShift Container Platform クラスターにインポートできます。OpenShift Virtualization はデータボリュームを使用してデータのインポートおよび基礎となる永続ボリューム要求 (PVC) の作成を自動化します。

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。

8.16.3.1. 前提条件

- CDI でサポートされる操作マトリックス に応じてスクラッチ領域が必要な場合は、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。

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

8.16.3.3. 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。

8.16.3.4. ローカルブロック永続ボリュームの作成

ファイルにデータを設定し、これをループデバイスとしてマウントすることにより、ノードでローカルブロック永続ボリューム (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。

8.16.3.5. データボリュームを使用して仮想マシンイメージをブロック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ブロックストレージにインポートできます。仮想マシンを作成する前に、**VirtualMachine** マニフェストでデータボリュームを参照します。

前提条件

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

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. データ **DataVolume** マニフェストを作成し、仮想マシンイメージのデータソースと **storage.volumeMode** の **Block** を指定します。

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: import-pv-datavolume 1
spec:
  storageClassName: local 2
    source:
      http:
        url: "https://mirror.arizona.edu/fedora/linux/releases/35/Cloud/x86_64/images/Fedora-Cloud-Base-35-1.2.x86_64.qcow2" 3
        secretRef: endpoint-secret 4
  storage:
    volumeMode: Block 5
    resources:
      requests:
        storage: 10Gi
```

1. データボリュームの名前を指定します。
2. オプション: ストレージクラスを設定するか、これを省略してクラスターのデフォルトを受け入れます。
3. インポートするイメージの HTTP または HTTPS URL を指定します。
4. データソースの **Secret** を作成した場合は、**Secret** 名を指定します。
5. ボリュームモードとアクセスモードは、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。それ以外の場合は、**Block** を指定します。

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

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

仮想マシンを作成する前に、**VirtualMachine** マニフェストでこのデータボリュームを参照できます。

8.16.3.6. 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。

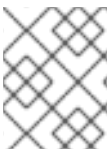
コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
KubeVirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* □ GZ □ XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* □ GZ □ XZ	✓ RAW* ✓ GZ* ✓ XZ*

✓ 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



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。

8.16.3.7. 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。

8.17. 仮想マシンのクローン作成

8.17.1. 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。

8.17.1.1. 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。

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

8.17.1.3. 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。

8.17.2. 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 について参照してください。

8.17.2.1. 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 が必要である。

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

8.17.2.3. 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。

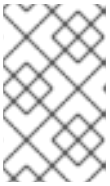
```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: <cloner-datavolume> 1
spec:
  source:
    pvc:
      namespace: "<source-namespace>" 2
      name: "<my-favorite-vm-disk>" 3
  pvc:
    accessModes:
      - ReadWriteOnce
```

```
  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。

8.17.2.4. 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

8.17.3. 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 について参照してください。

8.17.3.1. 前提条件

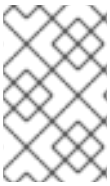
- ユーザーは、仮想マシンディスクの PVC のクローンを別の namespace に作成するために 追加のパーミッション が必要である。

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

8.17.3.3. 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
```

8.17.3.4. 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

8.17.4. 新規ブロックストレージ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、新規ブロック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 について参照してください。

8.17.4.1. 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 が必要である。

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

8.17.4.3. 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。

8.17.4.4. ローカルブロック永続ボリュームの作成

ファイルにデータを設定し、これをループデバイスとしてマウントすることにより、ノードでローカルブロック永続ボリューム (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。

8.17.4.5. 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。

8.17.4.6. 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

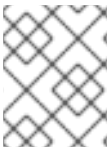
* スクラッチ領域が必要

**カスタム認証局が必要な場合にスクラッチ領域が必要

8.18. 仮想マシンのネットワーク

8.18.1. 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。

8.18.1.1. 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
```

8.18.1.2. 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) ネットワークプロバイダー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}"
```

8.18.2. 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。

8.18.2.1. サービスについて

Kubernetes **サービス** は、一連の Pod で実行されるアプリケーションをネットワークサービスとして公開するための抽象的な方法です。サービスを使用すると、アプリケーションがトラフィックを受信できます。サービスは、**Service** オブジェクトに **spec.type** を指定して複数の異なる方法で公開できます。

ClusterIP
: クラスター内の内部 IP アドレスでサービスを公開します。**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。

8.18.2.1.1. 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]**

8.18.2.2. 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。

8.18.2.3. 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

8.18.3. 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。

8.18.3.1. ネットワーク接続定義によるネットワークへの接続

8.18.3.1.1. 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。

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。

8.18.3.2. 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。

8.18.3.2.1. 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

ネットワーク管理者は、ネットワーク接続定義を作成して layer-2 ネットワークを Pod および仮想マシンに提供できます。

手順

1. Web コンソールで、**Networking → Network Attachment Definitions** 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** 一覧をクリックし、**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。

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。

8.18.3.2.2. 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": 1 7
  }'
```

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

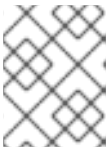
3 設定の名前。設定名をネットワーク接続定義の **name** 値に一致させることが推奨されます。

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。



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>
```

8.18.3.3. Linux ブリッジネットワーク用の仮想マシンの設定

8.18.3.3.1. 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. **Network Interfaces** タブをクリックして、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。

8.18.3.3.2. ネットワークフィールド

Name	Description
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。
ネットワーク	利用可能なネットワーク接続定義の一覧。
Type	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。

8.18.3.3.3. 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。

8.18.4. 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。

8.18.4.1. 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 がインストールされていること。

8.18.4.2. 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こと。
- 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こと。

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: 仮想機能 (VF) 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 **128** 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ついての 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について、詳しく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。

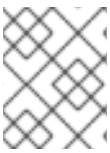
4. SR-IOV ネットワークデバイスが設定されていることを確認するには、以下のコマンドを実行します。**<node_name>** を、設定したばかりの SR-IOV ネットワークデバイスを持つノードの名前に置き換えます。

```
$ oc get sriovnetworknodestates -n openshift-sriov-network-operator <node_name> -o
jsonpath='{.status.syncStatus}'
```

8.18.4.3. 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>
```

8.18.4.4. 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。

8.18.5. 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。

8.18.5.1. 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。

8.18.5.2. 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。

8.18.6. 仮想マシンの IP アドレスの設定

動的または静的のいずれかでプロビジョニングされた仮想マシンの IP アドレスを設定できます。

前提条件

- 仮想マシンは、外部ネットワーク に接続する必要があります。
- 仮想マシンの動的 IP を設定するには、追加のネットワークで使用可能な DHCP サーバーが必要です。

8.18.6.1. cloud-init を使用した新規仮想マシンの IP アドレスの設定

仮想マシンの作成時に cloud-init を使用して 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。

手順

- 仮想マシン設定を作成し、仮想マシン設定の **spec.volumes.cloudInitNoCloud.networkData** フィールドに cloud-init ネットワークの詳細を追加します。

  a. 動的 IP を設定するには、インターフェイス名と **dhcp4** ブール値を指定します。

```
kind: VirtualMachine
spec:
...
  volumes:
  - cloudInitNoCloud:
      networkData: |
        version: 2
        ethernets:
          eth1: 1
            dhcp4: true 2
```

1 インターフェイスの名前。

2 DHCP を使用して IPv4 アドレスをプロビジョニングします。

  b. 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:
...
  volumes:
  - cloudInitNoCloud:
      networkData: |
        version: 2
        ethernets:
          eth1: 1
            addresses:
            - 10.10.10.14/24 2
```

1 インターフェイスの名前。

2 仮想マシンの静的 IP アドレス。

8.18.7. 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。

8.18.7.1. 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。

8.18.7.2. 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
```

8.18.7.3. 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。

8.18.8. 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。

8.18.8.1. 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。

8.18.8.2. 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。

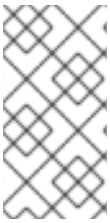
手順

- **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
```

8.18.8.3. 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-
```

8.19. 仮想マシンディスク

8.19.1. ストレージ機能

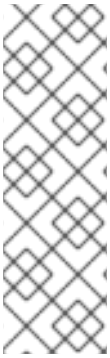
以下の表を使用して、OpenShift Virtualization のローカルおよび共有の永続ストレージ機能の可用性を確認できます。

8.19.1.1. OpenShift Virtualization ストレージ機能マトリクス

表8.5 OpenShift Virtualization ストレージ機能マトリクス

	仮想マシンのライブマイグレーション	ホスト支援型仮想マシンディスクのクローン作成	ストレージ支援型仮想マシンディスクのクローン作成	仮想マシンのスナップショット
OpenShift Data Foundation: RBD ブロックモードボリューム	はい	○	○	はい
OpenShift Virtualization ホストパスプロビジョナー	いいえ	はい	×	いいえ
他の複数ノードの書き込み可能なストレージ	はい [1]	はい	はい [2]	はい [2]
他の単一ノードの書き込み可能なストレージ	いいえ	はい	はい [2]	はい [2]

1. PVC は ReadWriteMany アクセスモードを要求する必要があります。
2. ストレージプロバイダーが Kubernetes および CSI スナップショット API の両方をサポートする必要があります。



注記

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クラス
- 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。

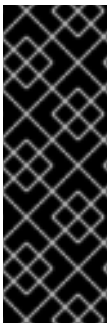
8.19.2. 仮想マシンのローカルストレージの設定

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。

8.19.2.1. ホストパスプロビジョナーについて

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。



重要

OpenShift Virtualization 4.10 では、HPP Operator は Kubernetes CSI ドライバーを設定します。Operator は、HPP CR の既存の (レガシー) フォーマットも認識します。

従来の HPP と Container Storage Interface (CSI) ドライバーは、多くのリリースで並行してサポートされています。ただし、ある時点では、従来の HPP はサポートされなくなります。HPP を使用する場合は、移行ストラテジーの一部として CSI ドライバーのストレージクラスを作成することを計画してください。

既存のクラスターで OpenShift Virtualization バージョン 4.10 にアップグレードする場合、HPP Operator はアップグレードされ、システムは以下の操作を実行します。

- CSI ドライバーがインストールされる。
- CSI ドライバーは、レガシー HPP CR の内容で設定されます。

OpenShift Virtualization バージョン 4.10 を新規クラスターにインストールする場合、以下のアクションを実行する必要があります。

- 基本ストレージプールを使用して HPP CR を作成します。
- CSI ドライバーのストレージクラスを作成します。

オプション: 複数の HPP ボリューム用の PVC テンプレートを使用してストレージプールを作成できます。

8.19.2.2. 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。

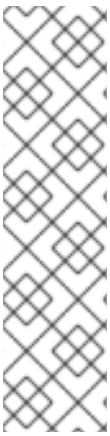
3. 次のコマンドを実行して HPP を作成します。

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

8.19.2.3. 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。

8.19.2.3.1. 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。

前提条件

- OpenShift Virtualization 4.10 以降が必要です。

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 (1)
provisioner: kubevirt.io.hostpath-provisioner
reclaimPolicy: Delete (2)
volumeBindingMode: WaitForFirstConsumer (3)
parameters:
  storagePool: my-storage-pool (4)
```

1. 意味のある名前をストレージクラスに割り当てます。この例では、**csi** は、クラスがレガシープロビジョナーではなく CSI プロビジョナーを使用することを指定するために使用されます。レガシーまたは CSI ドライバープロビジョニングに基づいてストレージクラスの説明的な名を選択することで、移行ストラテジーの実装が容易になります。
2. **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。
3. **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。
4. 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。

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、**StorageClass** オブジェクトを作成します。

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

8.19.2.3.2. レガシーのホストパスプロビジョナーのストレージクラスの作成

storagePool パラメーターを指定せずに **StorageClass** オブジェクトを作成することにより、従来のホストパスプロビジョナー (HPP) のストレージ クラスを作成します。

手順

1. **storageclass.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-provisioner
provisioner: kubevirt.io/hostpath-provisioner
reclaimPolicy: Delete (1)
volumeBindingMode: WaitForFirstConsumer (2)
```

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、ストレージクラスはデフォルトで **Delete** に設定されます。
2. **volumeBindingMode** 値は、動的プロビジョニングおよびボリューム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** の値を指定して、永続ボ

行されるノードセンタを決定します。**WaitForFirstConsumer** の値を指定して、永続ボリューム要求 (PVC) を使用する Pod が作成されるまで永続ボリューム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。

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、**StorageClass** オブジェクトを作成します。

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

関連情報

- ストレージクラス

8.19.2.4. 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。

8.19.2.4.1. 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。

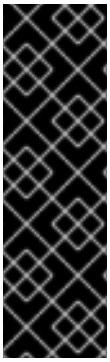
```
$ oc create -f hpp_pvc_template_pool.yaml
```

関連情報

- ストレージプロファイルのカスタマイズ

8.19.3. データボリュームの作成

データボリュームの作成時に、Containerized Data Importer (CDI) は永続ボリューム要求 (PVC) を作成し、PVC にデータを入力します。データボリュームは、スタンドアロンリソースとして、または仮想マシン仕様で **dataVolumeTemplate** リソースを使用することで、作成できます。PVC API 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します。



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。

8.19.3.1. 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。

8.19.3.2. 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**)

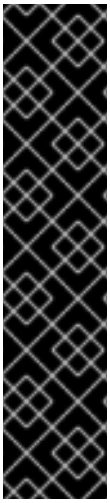
- **storage** の **capacity** (例: **5Gi**)

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。

8.19.3.3. 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。

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。

注記

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。

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。

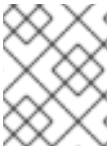
8.19.3.3.1. ストレージプロファイルを使用したデフォルトのクローンストラテジーの設定

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**: この方法は、スナップショットが設定されている場合にデフォルトで使用されます。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 CSI スナップショットをサポートする必要があります。
- **Copy**: この方法では、ソース Pod およびターゲット 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**: この方法は、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) 選択する **accessModes**

(2) 選択する **volumeMode**。

(3) 選択するデフォルトのクローン作成方法。この例では、CSI ボリュームのクローン作成が指定されています。

8.19.3.4. 関連情報

- ストレージクラスの作成について
- smart-cloning を使用したデータボリュームのクローン作成

8.19.4. 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。

8.19.4.1. 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。

8.19.4.2. 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。

8.19.4.3. 関連情報

- プロジェクトごとのリソースクォータ

8.19.5. 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。

8.19.5.1. 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

8.19.6. 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。

8.19.6.1. 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。

8.19.6.2. 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。

8.19.7. 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。

8.19.7.1. 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。

8.19.7.2. 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*

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 (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

8.19.7.3. 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. 一覧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**一覧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。

8.19.7.4. 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。

8.19.8. 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データボリュームにアップロードできます。

8.19.8.1. 前提条件

- **kubevirt-virtctl** パッケージを 有効 にすること。
- 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。

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

8.19.8.3. 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
```

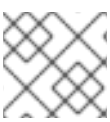
1. データボリュームの名前。
2. データボリュームのサイズ。この値がアップロードするディスクのサイズ以上であることを確認します。

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
```

8.19.8.4. 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。

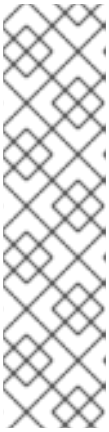
- **kubevirt-virtctl** パッケージがクライアントマシンにインストールされていること。
- クライアントマシンが OpenShift Container Platform ルーターの証明書を信頼するように設定されていること。

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
```

8.19.8.5. 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

8.19.8.6. 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。

8.19.9. ブロックストレージデータボリュームへのローカルディスクイメージのアップロード

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

8.19.9.1. 前提条件

- **kubevirt-virtctl** パッケージを 有効 にすること。
- 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。

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

8.19.9.3. 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。

8.19.9.4. ローカルブロック永続ボリュームの作成

ファイルにデータを設定し、これをループデバイスとしてマウントすることにより、ノードでローカルブロック永続ボリューム (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。

8.19.9.5. 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
```

8.19.9.6. 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
```

8.19.9.7. 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

8.19.9.8. 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。

8.19.10. 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。

8.19.10.1. 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。そのような場合には、スナップショットの作成前に仮想マシンファイルシステムをフリーズして、スナップショットの作成後にファイルシステムをロールアウトします。

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一覧表示
- 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

8.19.10.1.1. 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。

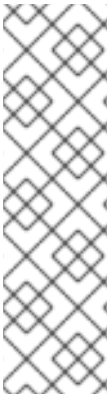
- **VirtualMachineSnapshot**: スナップショットを作成するユーザー要求を表します。これには、仮想マシンの現在の状態に関する情報が含まれます。
- **VirtualMachineSnapshotContent**: クラスター上のプロビジョニングされたリソース (スナップショット) を表します。これは、仮想マシンのスナップショットコントローラーによって作成され、仮想マシンの復元に必要なすべてのリソースへの参照が含まれます。
- **VirtualMachineRestore**: スナップショットから仮想マシンを復元するユーザー要求を表します。

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。

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

仮想マシン (VM) に QEMU ゲストエージェントがインストールされ、実行されているかどうかを確認するには、**AgentConnected** が VM 仕様に表示されていることを確認します。



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
```

8.19.10.3. 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。

仮想マシン (VM) に QEMU ゲストエージェントがインストールされ、実行されているかどうかを確認するには、**AgentConnected** が VM 仕様に表示されていることを確認します。

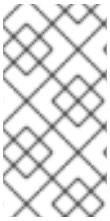
注記

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。

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

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

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

注記

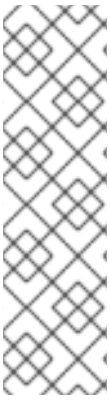
この手順では、Windows インストールの汎用的なアプローチを使用しますが、インストール方法は Windows のバージョンごとに異なる可能性があります。インストールする Windows のバージョンについてのドキュメントを参照してください。

手順

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

8.19.10.4. 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。

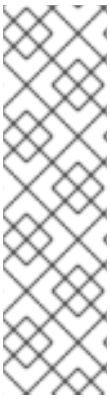
手順

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。

8.19.10.5. 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する。
- 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ること。

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/v1alpha1
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/v1alpha1
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/v1alpha1/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。

8.19.10.6. 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。

8.19.10.7. 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一覧が表示されます。
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。

8.19.10.8. 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/v1alpha1
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/v1alpha1
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/v1alpha1/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。

8.19.10.9. 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。

8.19.10.10. 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
```

8.19.10.11. 関連情報

- CSI ボリュームスナップショット

8.19.11. 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。

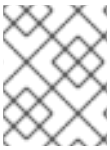
以下の理由により、仮想マシンを特定のノードに移動する場合があります。

- 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 について参照してください。



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。

8.19.11.1. 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。

8.19.12. 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。

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

8.19.12.2. 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:
    # Optional: Set the storage class or omit to accept the default
    # storageClassName: "hostpath"
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: 500Mi
```

2. 以下のコマンドを実行して、空のディスクイメージを作成します。

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

8.19.12.3. 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。

8.19.13. 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。

8.19.13.1. 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。

8.19.13.2. 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。

8.19.13.3. 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

8.19.14. 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。

8.19.14.1. 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、以前の既定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。

8.19.14.2. 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** 一覧から **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一覧は、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。

8.19.14.3. 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。

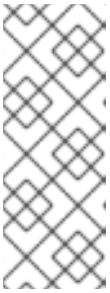
- **ローカルファイルのアップロード (PVC の作成)**
- **URL (PVC を作成)**
- **クローン (PVC を作成)**
- **レジストリー (PVC を作成)**

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)** または **Use an existing 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** または **CD-ROM** 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、以前の既定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** ページに一覧表示されています。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。

8.19.14.4. 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。

8.19.14.5. カスタムブートソースの作成

既存のディスクイメージに基づいて、ブートソースとして使用するカスタムディスクイメージを準備できます。

この手順を使用して、以下のタスクを実行します。

- カスタムディスクイメージの準備
- カスタムディスクイメージからのブートソースの作成
- ブートソースのカスタムテンプレートへの割り当て

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. カスタマイズするテンプレートの **ブートソース** 列のリンクをクリックします。ウィンドウが表示され、テンプレートに現在 **定義されている** ソースがあることを示します。

3. ウィンドウで **Customize source** リンクをクリックします。

4. ブートソースのカスタマイズプロセスについて提供された情報を読んだ後、**About boot source customization** ウィンドウで **Continue** をクリックして、カスタマイズを続行します。

5. **Prepare boot source customization** ページの **Define new template** セクションで、以下を実行します。

    a. **New template namespace** フィールドを選択してから、プロジェクトを選択します。

    b. **New template name** フィールドに、カスタムテンプレートの名前を入力します。

    c. **New template provider** フィールドに、テンプレートプロバイダーの名前を入力します。

    d. **New template support** フィールドを選択してから、作成するカスタムテンプレートのサポートの連絡先を示す適切な値を選択します。

    e. **New template flavor** フィールドを選択してから、作成するカスタムイメージの適切な CPU およびメモリーの値を選択します。

6. **Prepare boot source for customize** セクションで、必要に応じてログイン認証情報を定義する **cloud-init** YAML スクリプトをカスタマイズします。それ以外の場合は、スクリプトによりデフォルトの認証情報が生成されます。

7. **Start Customization** をクリックします。カスタマイズプロセスが開始され、**Preparing boot source customization** ページが表示され、その後に **Customize boot source** ページが表示されます。**Customize boot source** ページには、実行中のスクリプトの出力が表示されます。スクリプトが完了したら、カスタムイメージが利用可能になります。

8. **VNC console** で、**Guest login credentials** セクションの **show password** をクリックします。ログイン認証情報が表示されます。

9. ログインのイメージの準備ができたら、**Guest login credentials** セクションに表示されるユーザー名およびパスワードを指定して、**VNC Console** でサインインします。

10. カスタムイメージが期待どおりに機能することを確認します。機能する場合は、**Make this boot source available** をクリックします。

11. **Finish customization and make template available** ウィンドウで、**I have sealed the boot source so it can be used as a template** を選択してから、**Apply** をクリックします。

12. **Finishing boot source customization** ページで、テンプレート作成プロセスが完了するまで待ちます。**Navigate to template details** または **Navigate to template list** をクリックして、カスタムブートソースから作成したカスタマイズされたテンプレートを表示します。

8.19.14.6. 関連情報

- 仮想マシンテンプレートの作成
- クラウドイメージからの Microsoft Windows ブートソースの作成
- OpenShift Container Platform での既存の Microsoft Windows ブートソースのカスタマイズ
- CLI を使用した Microsoft Windows テンプレートのブートソースとしての PVC の設定

- 自動スクリプトを使用したブートソースの作成
- Pod 内でのブートソースの自動作成

8.19.15. 仮想ディスクのホットプラグ

仮想マシンまたは仮想マシンインスタンスを停止せずに追加または削除する場合に、ホットプラグおよびホットアンプラグを行います。この機能は、ダウンタイムなしに、実行中の仮想マシンにストレージを追加する必要がある場合に便利です。

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** する場合、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またはホットアンプラグはできません。

8.19.15.1. 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。

8.19.15.2. 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>
```

8.19.15.3. 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。

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. 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. **Disks** タブで、**Add disk** をクリックします。
4. **Add disk** 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。
5. **Add** をクリックします。

8.19.15.4. 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) に割り当てる必要のある仮想ディスクのホットプラグを解除します。

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アタッチされている。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 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. **Disks** タブで、ホットアンプラグする仮想ディスクの Options メニュー をクリックします。

4. **Delete** をクリックします。

8.19.16. 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

8.19.16.1. 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。

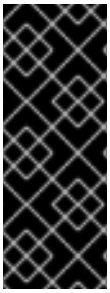
8.19.16.1.1. データボリュームの使用によるコンテナーディスクの PVC へのインポート

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。

8.19.16.1.2. 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。



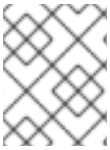
重要

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。

8.19.16.2. 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。

8.19.16.3. 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) この一覧のサンプルを、有効なレジストリーホスト名に置き換えます。

8.19.16.4. 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。

8.19.17. CDI のスクラッチ領域の用意

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

8.19.17.2. 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。

8.19.17.3. スクラッチ領域を必要とする CDI 操作

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

8.19.17.4. 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 (PVC) 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。

8.19.17.5. 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

8.19.17.6. 関連情報

- 動的プロビジョニング

8.19.18. 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。

8.19.18.1. 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。

8.19.18.2. 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

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。

8.19.19. 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。

8.19.19.1. 仮想マシンディスクの拡張

仮想マシンディスクの拡大により、仮想マシンに対して追加の領域が利用可能になります。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. **spec.resource.requests.storage** 属性の値を大きなサイズに変更します。

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: vm-disk-expand
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 3Gi (1)
...
```

(1) 拡張できる仮想マシンのディスクサイズ

8.19.19.2. 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

8.19.20. データボリュームの削除

oc コマンドラインインターフェイスを使用して、データボリュームを手動で削除できます。

注記

仮想マシンを削除する際に、これが使用するデータボリュームは自動的に削除されます。

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

8.19.20.2. すべてのデータボリュームの一覧表示

oc コマンドラインインターフェイスを使用して、クラスターのデータボリュームを一覧表示できます。

手順

- 以下のコマンドを実行して、データボリュームを一覧表示します。

```
$ oc get dvs
```

8.19.20.3. データボリュームの削除

oc コマンドラインインターフェイス (CLI) を使用してデータボリュームを削除できます。

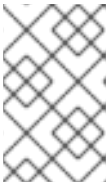
前提条件

- 削除する必要のあるデータボリュームの名前を特定すること。

手順

- 以下のコマンドを実行し、データボリューム を削除します。

```
$ oc delete dv <datavolume_name>
```



注記

このコマンドは、現在のプロジェクトに存在するオブジェクトのみを削除します。削除する必要のあるオブジェクトが別のプロジェクトまたは namespace にある場合、**-n <project_name>** オプションを指定します。

第9章 仮想マシンテンプレート

9.1. 仮想マシンテンプレートの作成

9.1.1. 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** ページに一覧表示されます。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。

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。

9.1.2. 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、以前の既定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。

9.1.3. 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. **Create Template** をクリックします。
3. 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。
4. **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。
5. 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。

9.1.4. 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)** または **Use an existing 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** または **CD-ROM** 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、以前の既定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** ページに一覧表示されています。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。

9.1.4.1. 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。

9.1.5. 仮想マシンテンプレートをお気に入りとしてマーク

よく使うテンプレートをお気に入りに登録できます。

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. テンプレートの横にあるスターアイコンをクリックして、お気に入りとしてマークします。お気に入りのテンプレートは、テンプレートリストの上部に表示されます。

9.1.6. プロバイダーによる仮想マシンテンプレート一覧のフィルター

Templates タブで、**Search by name** フィールドを使用し、テンプレートの名前、またはテンプレートを特定するラベルのいずれかを指定して仮想マシンテンプレートを検索できます。プロバイダーでテンプレートをフィルターし、フィルター条件を満たすテンプレートのみを表示することもできます。

手順

1. OpenShift Virtualization コンソールのサイドメニューから **Workloads → Virtualization** をクリックします。
2. **Templates** タブをクリックします。
3. テンプレートをフィルターするには、**Filter** をクリックします。
4. 一覧から該当するチェックボックスを選択してテンプレートをフィルターします: **Red Hat Supported**、**User Supported**、**Red Hat Provided**、および **User Provided**。

9.1.7. 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

9.2. 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。

9.2.1. Web コンソールでの仮想マシンテンプレートの編集

関連するフィールドの横にある鉛筆アイコンをクリックして、Web コンソールの仮想マシンテンプレートの選択する値 (select values) を編集します。他の値は、CLI を使用して編集できます。

ラベルとアノテーションは、事前に設定された Red Hat テンプレートとカスタム仮想マシンテンプレートの両方について編集されます。その他のすべての値は、ユーザーが Red Hat テンプレートまたは **Create Virtual Machine Template**ウィザードを使用して作成したカスタム仮想マシンテンプレートについてのみ編集されます。

手順

1. サイドメニューから **Virtualization → Templates** をクリックします。
2. オプション: **Filter** ドロップダウンメニューを使用して、ステータス、テンプレート、ノード、またはオペレーティングシステム (OS) などの属性で仮想マシンテンプレートのリストを並べ替えます。
3. 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。
4. 鉛筆アイコンをクリックして、フィールドを編集可能にします。
5. 関連する変更を加え、**Save** をクリックします。

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。

9.2.1.1. 仮想マシンテンプレートフィールド

以下の表には、OpenShift Container Platform Web コンソールで編集できる仮想マシンテンプレートのフィールドが記載されています。

表9.1 仮想マシンテンプレートフィールド

タブ	フィールドまたは機能
Details	● ラベル ● アノテーション ● 表示名 ● 説明 ● Workload profile ● CPU/メモリー ● 起動モード ● GPU デバイス ● ホストデバイス
YAML	● カスタムリソースを表示、編集、またはダウンロードします。
スケジューリング	● Node selector ● 容認 ● アフィニティールール ● 専用リソース ● エビクションストラテジー ● Descheduler 設定
Network Interfaces	● ネットワークインターフェイスを追加、編集、または削除します。
ディスク	● ディスクを追加、編集、または削除します。
スクリプト	● cloud-init 設定

タブ	フィールドまたは機能
パラメーター (オプション)	• Virtual machine name • cloud-user パスワード

9.2.1.2. 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** 画面を開きます。
3. **Network Interfaces** タブをクリックします。
4. **Add Network Interface** をクリックします。
5. **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。
6. **Add** をクリックします。

9.2.1.3. 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** 画面を開きます。
3. **Disks** タブをクリックし、**Add disk** をクリックします。
4. **Add disk** ウィンドウで、**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。

9.2.1.4. テンプレートの CD-ROM の編集

以下の手順を使用して、仮想マシンテンプレートの CD-ROM を設定します。

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** 画面を開きます。
3. **Disks** タブをクリックします。
4. 編集する CD-ROM の Options メニュー  をクリックし、**Edit** を選択します。
5. **Edit CD-ROM** ウィンドウで、**Source**、**Persistent Volume Claim**、**Name**、**Type**、および **Interface** フィールドを編集します。
6. **Save** をクリックします。

9.3. 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。

9.3.1. 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。

9.3.2. 前提条件

- CPU マネージャー がノードで設定されている。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。

9.3.3. 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** の横にある鉛筆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。

9.4. 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** ページに一覧表示されています。

9.4.1. 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>
```

9.4.2. 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一覧を表示します。

```
$ 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。

9.4.2.1. 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
```

9.4.2.2. 関連情報

- 仮想マシンテンプレートの作成

9.5. 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。

9.5.1. 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。

第10章 ライブマイグレーション

10.1. 仮想マシンのライブマイグレーション

10.1.1. 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。

10.1.2. 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制限
- ストレージプロファイルのカスタマイズ

10.2. 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。

10.2.1. 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。

10.2.2. 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

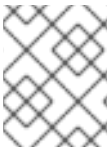
表10.1 移行パラメーター

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	それぞれの移行の帯域幅 (MiB/s)。	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。

10.3. 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。

10.3.1. 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。

10.3.2. 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。

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モニター
- 仮想マシンインスタンスのライブマイグレーションの取り消し

10.4. 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。

10.4.1. 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用に専用のセカンダリーネットワークを設定するには、まず CLI を使用して namespace のブリッジネットワーク接続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
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", (2)
    "mode": "bridge",
```

```
    "ipam": {
      "type": "whereabouts", 3
      "range": "10.200.5.0/24" 4
    }
  }'
```

1. **NetworkAttachmentDefinition** オブジェクトの名前。
2. ライブマイグレーションに使用する NIC の名前。
3. 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。
4. 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}'
```

10.4.2. 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

10.5. 仮想マシンインスタンスのライブマイグレーションのモニター

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

10.5.1. 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モニター

移行期間中、仮想マシンのステータスは **Migrating** になります。このステータスは、**VirtualMachines** ページまたは移行中の仮想マシンの **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. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。

10.5.2. 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
```

10.6. 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。

10.6.1. 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。

10.6.2. 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
```

10.7. 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。

10.7.1. 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>
```

第11章 ノードのメンテナンス

11.1. ノードのメンテナンスについて

11.1.1. 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。

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。

OpenShift Virtualization の一部としてインストールされる場合、Node Maintenance Operator は新規または削除された **NodeMaintenance** CR の有無を監視します。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。

11.1.2. 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についての詳細は、メンテナンス時に提供されます。

関連情報:

- 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

11.2. ノードのメンテナンスモードへの設定

Web コンソール、CLI、または **NodeMaintenance** カスタムリソースを使用してノードをメンテナンス状態にします。

11.2.1. Web コンソールでのノードのメンテナンスモードへの設定

Compute → Nodes 一覧で各ノードにある Options メニュー  を使用するフィルタリング**Node Details** 画面の **Actions** コントロールを使用してノードをメンテナンスモードに設定します。

手順

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

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

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

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

11.2.2. CLI でのノードのメンテナンスモードへの設定

ノードをスケジュール対象外としてマークし、**oc adm drain** コマンドを使用してノードから Pod をエビクトまたは削除して、ノードをメンテナンスモードに設定します。

手順

1. ノードにスケジュール対象外 (unschedulable) のマークを付けます。ノードのステータスが **NotReady,SchedulingDisabled** に切り替わります。

```
$ oc adm cordon <node1>
```

2. メンテナンスの準備のためにノードをドレイン (解放) します。ノードは、**LiveMigratable** の状態が **True** に設定され、 **spec:evictionStrategy** フィールドが **LiveMigrate** に設定された仮想マシンインスタンスのライブマイグレーションを行います。このノードの他のすべての Pod および仮想マシンは削除され、別のノードで再作成されます。

```
$ oc adm drain <node1> --delete-emptydir-data --ignore-daemonsets=true --force
```

- **--delete-emptydir-data** フラグは、**emptyDir** ボリュームを使用するノード上の仮想マシンインスタンスを削除します。これらのボリュームのデータは一時的なものであり、終了後に削除しても問題はありません。
- **--ignore-daemonsets=true** フラグは、デーモンセットは無視され、Pod のエビクションが正常に続行されるようにします。
- **--force** フラグは、レプリカセットまたはデーモンセットコントローラーによって管理されない Pod を削除するために必要です。

11.2.3. NodeMaintenance カスタムリソースを使用したノードのメンテナンスモードへの設定

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できます。**NodeMaintenance** CR を適用する場合、許可されるすべての Pod はエビクトされ、ノードはシャットダウンします。エビクトされた Pod は、クラスター内の別のノードに移動するようにキューに置かれます。

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クラスターにログインしている。

手順

1. 以下のノードメンテナンス CR を作成し、そのファイルを **nodemaintenance-cr.yaml** として保存します。

```
apiVersion: nodemaintenance.kubevirt.io/v1beta1
kind: NodeMaintenance
metadata:
  name: maintenance-example 1
spec:
  nodeName: node-1.example.com 2
  reason: "Node maintenance" 3
```

1. ノードのメンテナンス CR 名
2. メンテナンスモードに設定するノードの名前
3. メンテナンスの理由のプレーンテキストの説明

2. 以下のコマンドを実行してノードのメンテナンススケジュールを適用します。

```
$ oc apply -f nodemaintenance-cr.yaml
```

3. 以下のコマンドを実行して、**<node-name>** をノードの名前に置き換えて、メンテナンスタスクの進捗を確認します。

```
$ oc describe node <node-name>
```

出力例

```
Events:
  Type     Reason                  Age                From     Message
  ----     ------                  ----               ----     -------
  Normal   NodeNotSchedulable      61m                kubelet  Node node-1.example.com
status is now: NodeNotSchedulable
```

11.2.3.1. 現在の NodeMaintenance CR タスクのステータスの確認

現在の **NodeMaintenance** CR タスクのステータスを確認できます。

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

- 以下のコマンドを実行して、現在のノードのメンテナンスタスクのステータスを確認します。

```
$ oc get NodeMaintenance -o yaml
```

出力例

```
apiVersion: v1
items:
- apiVersion: nodemaintenance.kubevirt.io/v1beta1
  kind: NodeMaintenance
  metadata:
...
  spec:
    nodeName: node-1.example.com
    reason: Node maintenance
  status:
    evictionPods: 3 (1)
    pendingPods:
    - pod-example-workload-0
    - httpd
    - httpd-manual
    phase: Running
    lastError: "Last failure message" (2)
    totalpods: 5
...
```

(1) **evictionPods** はエビクションにスケジュールされる Pod 数です。

(2) **lastError** は、最新のエビクションエラーが存在する場合は、最新のエビクションエラーを記録します。

関連情報:

- メンテナンスモードからのノードの再開
- 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

11.3. メンテナンスモードからのノードの再開

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

Web コンソール、CLI を使用するフィルタリング**NodeMaintenance** カスタムリソースを削除して、メンテナンスモードからノードを再開します。

11.3.1. Web コンソールでのメンテナンスモードからのノードの再開

Options メニュー  (**Compute → Nodes** 一覧の各ノードにある) を使用するフィルタリング**Node Details** 画面の **Actions** コントロールを使用してノードをメンテナンスモードから再開します。

手順

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

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

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

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

11.3.2. CLI でのメンテナンスモードからのノードの再開

ノードを再度スケジュール可能にすることで、メンテナンスモードのノードを再開します。

手順

- ノードにスケジュール可能のマークを付けます。次に、ノードで新規ワークロードのスケジューリングを再開できます。

```
$ oc adm uncordon <node1>
```

11.3.3. NodeMaintenance CR を使用して開始されたメンテナンスモードからのノードの再起動

NodeMaintenance CR を削除してノードを再起動できます。

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クラスターにログインしている。

手順

- ノードのメンテナンスタスクが完了したら、アクティブな **NodeMaintenance** CR を削除します。

```
$ oc delete -f nodemaintenance-cr.yaml
```

出力例

```
nodemaintenance.nodemaintenance.kubevirt.io "maintenance-example" deleted
```

11.4. 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。

11.4.1. 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

11.5. 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。

11.5.1. 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一覧を使用して、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一覧から削除されます。

例11.1 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。この一覧から 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一覧に追加することはできます。

11.5.2. 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一覧から削除されます。

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 機能の一覧と比較されます。

  例11.2 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11.3 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11.4 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
```

11.5.3. 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 モデルの一覧を設定できます。

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 モデルの事前定義一覧に追加されます。事前定義された一覧は 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。

11.6. 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。

11.6.1. 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。

11.6.2. 関連情報

- 古い CPU モデルのノードラベルの管理

第12章 ノードのネットワーク

12.1. ノードのネットワーク状態の確認

ノードのネットワーク状態は、クラスター内のすべてのノードのネットワーク設定です。

12.1.1. nmstate について

OpenShift Virtualization は **nmstate** を使用してノードネットワークの状態を報告し、設定します。これにより、単一の設定マニフェストをクラスターに適用して、すべてのノードに Linux ブリッジを作成するなどして、ネットワークポリシーの設定を変更することができます。

ノードのネットワークは、以下のオブジェクトによって監視され更新されます。

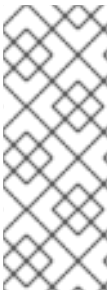
NodeNetworkState
: そのノード上のネットワークの状態を報告します。

NodeNetworkConfigurationPolicy
: ノードで要求されるネットワーク設定について説明します。**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。

NodeNetworkConfigurationEnactment
: 各ノードに制定されたネットワークポリシーを報告します。

OpenShift Virtualization は以下の nmstate インターフェイスタイプの使用をサポートします。

- Linux Bridge
- VLAN
- Bond
- イーサネット



注記

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。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。

12.1.2. ノードのネットワーク状態の表示

NodeNetworkState オブジェクトはクラスター内のすべてのノードにあります。このオブジェクトは定期的に更新され、ノードのネットワークの状態を取得します。

手順

1. クラスターのすべての **NodeNetworkState** オブジェクトを一覧表示します。

```
$ oc get nns
```

2. **NodeNetworkState** オブジェクトを検査して、そのノードにネットワークを表示します。この例の出力は、明確にするために編集されています。

```
$ oc get nns node01 -o yaml
```

出力例

```
apiVersion: nmstate.io/v1
kind: NodeNetworkState
metadata:
  name: node01 (1)
status:
  currentState: (2)
    dns-resolver:
...
    interfaces:
...
    route-rules:
...
    routes:
...
  lastSuccessfulUpdateTime: "2020-01-31T12:14:00Z" (3)
```

(1) **NodeNetworkState** オブジェクトの名前はノードから取られています。

(2) **currentState** には、DNS、インターフェイス、およびルートを含む、ノードの完全なネットワーク設定が含まれます。

(3) 最後に成功した更新のタイムスタンプ。これは、ノードが到達可能であり、レポートの鮮度の評価に使用できる限り定期的に更新されます。

12.2. ノードのネットワーク設定の更新

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用して、ノードからのインターフェイスの追加または削除など、ノードネットワーク設定を更新できます。

12.2.1. nmstate について

OpenShift Virtualization は **nmstate** を使用してノードネットワークの状態を報告し、設定します。これにより、単一の設定マニフェストをクラスターに適用して、すべてのノードに Linux ブリッジを作成するなどして、ネットワークポリシーの設定を変更することができます。

ノードのネットワークは、以下のオブジェクトによって監視され更新されます。

NodeNetworkState

そのノード上のネットワークの状態を報告します。

NodeNetworkConfigurationPolicy

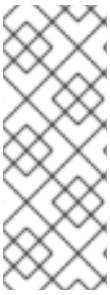
ノードで要求されるネットワーク設定について説明します。**NodeNetworkConfigurationPolicy** マニフェストをクラスターに適用して、インターフェイスの追加および削除など、ノードネットワーク設定を更新します。

NodeNetworkConfigurationEnactment

各ノードに制定されたネットワークポリシーを報告します。

OpenShift Virtualization は以下の nmstate インターフェイスタイプの使用をサポートします。

- Linux Bridge
- VLAN
- Bond
- イーサネット



注記

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。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。

12.2.2. ノード上でのインターフェイスの作成

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノード上にインターフェイスを作成します。マニフェストには、インターフェイスの要求された設定の詳細が含まれます。

デフォルトでは、マニフェストはクラスター内のすべてのノードに適用されます。インターフェイスを特定ノードに追加するには、ノードセレクターの **spec: nodeSelector** パラメーターおよび適切な **<key>:<value>** を追加します。

複数の nmstate 対応ノードを同時に設定できます。この設定は、並列のノードの 50% に適用されます。このストラテジーでは、ネットワーク接続に障害が発生した場合にクラスター全体が使用できなくなるのを回避します。クラスターの特定の部分に、ポリシーの並行設定を適用するには、**maxUnavailable**フィールドを使用します。

手順

1. **NodeNetworkConfigurationPolicy** マニフェストを作成します。以下の例は、すべてのワーカーノードで Linux ブリッジを設定し、DNS リゾルバーを設定します。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy (1)
spec:
  nodeSelector: (2)
    node-role.kubernetes.io/worker: "" (3)
  maxUnavailable: 3 (4)
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port (5)
        type: linux-bridge
        state: up
```

```
      ipv4:
        dhcp: true
        enabled: true
        auto-dns: false
      bridge:
        options:
          stp:
            enabled: false
        port:
          - name: eth1
    dns-resolver: 6
      config:
        search:
        - example.com
        - example.org
        server:
        - 8.8.8.8
```

1 ポリシーの名前。

2 オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。

3 この例では **node-role.kubernetes.io/worker: ""** ノードセレクターを使用し、クラスター内のすべてのワーカーノードを選択します。

4 オプション: ポリシー設定を同時に適用できる nmstate 対応ノードの最大数を指定します。このパラメーターは、 **10%**などのパーセンテージ値 (文字列)、または**3**などの絶対値 (数値) のいずれかに設定できます。

5 オプション: インターフェイスの人間が判読できる説明。

6 オプション:DNS サーバーの検索およびサーバー設定を指定します。

2. ノードのネットワークポリシーを作成します。

```
$ oc apply -f br1-eth1-policy.yaml 1
```

1 ノードネットワーク設定ポリシーマニフェストのファイル名。

関連情報

- 異なるインターフェイスのポリシー設定の例
- 同じポリシーで複数のインターフェイスを作成する例
- ポリシーの各種 IP の管理方法の例

12.2.3. ノード上でのノードネットワークポリシー更新の確認

NodeNetworkConfigurationPolicy マニフェストは、クラスターのノードについて要求されるネットワーク設定を記述します。ノードネットワークポリシーには、要求されたネットワーク設定と、クラスター全体でのポリシーの実行ステータスが含まれます。

ノードネットワークポリシーを適用する際に、**NodeNetworkConfigurationEnactment** オブジェクトがクラスター内のすべてのノードについて作成されます。ノードネットワーク設定の enactment (実行) は、そのノードでのポリシーの実行ステータスを表す読み取り専用オブジェクトです。ポリシーがノードに適用されない場合、そのノードの enactment (実行) にはトラブルシューティングのためのトレースバックが含まれます。

手順

1. ポリシーがクラスターに適用されていることを確認するには、ポリシーとそのステータスを一覧表示します。

```
$ oc get nncp
```

2. オプション: ポリシーの設定に想定されている以上の時間がかかる場合は、特定のポリシーの要求される状態とステータスの状態を検査できます。

```
$ oc get nncp <policy> -o yaml
```

3. オプション: ポリシーのすべてのノード上での設定に想定されている以上の時間がかかる場合は、クラスターの enactment (実行) のステータスを一覧表示できます。

```
$ oc get nnce
```

4. オプション: 特定の enactment (実行) の設定 (失敗した設定のエラーレポートを含む) を表示するには、以下を実行します。

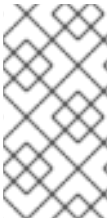
```
$ oc get nnce <node>.<policy> -o yaml
```

12.2.4. ノードからインターフェイスの削除

NodeNetworkConfigurationPolicy オブジェクトを編集し、インターフェイスの **state** を **absent** に設定して、クラスターの 1 つ以上のノードからインターフェイスを削除できます。

ノードからインターフェイスを削除しても、ノードのネットワーク設定は以前の状態に自動的に復元されません。以前の状態に復元する場合、そのノードのネットワーク設定をポリシーで定義する必要があります。

ブリッジまたはボンディングインターフェイスを削除すると、そのブリッジまたはボンディングインターフェイスに以前に接続されたか、それらの下位にあるノード NIC は **down** の状態になり、到達できなくなります。接続が失われないようにするには、同じポリシーでノード NIC を設定し、ステータスを **up** にし、DHCP または静的 IP アドレスのいずれかになるようにします。



注記

インターフェイスを追加したポリシーを削除しても、ノード上のポリシーの設定は変更されません。**NodeNetworkConfigurationPolicy** はクラスターのオブジェクトですが、これは要求される設定のみを表します。
同様に、インターフェイスを削除してもポリシーは削除されません。

手順

1. インターフェイスの作成に使用する **NodeNetworkConfigurationPolicy** マニフェストを更新します。以下の例は Linux ブリッジを削除し、接続が失われないように DHCP で **eth1** NIC を設定します。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> (1)
spec:
  nodeSelector: (2)
    node-role.kubernetes.io/worker: "" (3)
  desiredState:
    interfaces:
    - name: br1
      type: linux-bridge
      state: absent (4)
    - name: eth1 (5)
      type: ethernet (6)
      state: up (7)
      ipv4:
        dhcp: true (8)
        enabled: true (9)
```

(1) ポリシーの名前。

(2) オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。

(3) この例では **node-role.kubernetes.io/worker: ""** ノードセレクターを使用し、クラスター内のすべてのワーカーノードを選択します。

(4) 状態を **absent** に変更すると、インターフェイスが削除されます。

(5) ブリッジインターフェイスから接続が解除されるインターフェイスの名前。

(6) インターフェイスのタイプ。この例では、イーサネットネットワークインターフェイスを作成します。

(7) インターフェイスの要求された状態。

(8) オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。

(9) この例では **ipv4** を有効にします。

2. ノード上でポリシーを更新し、インターフェイスを削除します。

```
$ oc apply -f <br1-eth1-policy.yaml> (1)
```

(1) ポリシーマニフェストのファイル名。

12.2.5. 異なるインターフェイスのポリシー設定の例

12.2.5.1. 例: Linux ブリッジインターフェイスノードネットワーク設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノード上に Linux ブリッジインターフェイスを作成します。

以下の YAML ファイルは、Linux ブリッジインターフェイスのマニフェストの例です。これには、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:
  nodeSelector: (2)
    kubernetes.io/hostname: <node01> (3)
  desiredState:
    interfaces:
      - name: br1 (4)
        description: Linux bridge with eth1 as a port (5)
        type: linux-bridge (6)
        state: up (7)
        ipv4:
          dhcp: true (8)
          enabled: true (9)
        bridge:
          options:
            stp:
              enabled: false (10)
          port:
            - name: eth1 (11)
```

1. ポリシーの名前。
2. オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
3. この例では、**hostname** ノードセレクターを使用します。
4. インターフェイスの名前。
5. オプション: 人間が判読できるインターフェイスの説明。
6. 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。
7. 作成後のインターフェイスの要求された状態。
8. オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。
9. この例では **ipv4** を有効にします。
10. この例では **stp** を無効にします。
11. ブリッジが接続されるノードの NIC。

12.2.5.2. 例: VLAN インターフェイスノードネットワークの設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノード上に VLAN インターフェイスを作成します。

以下の YAML ファイルは、VLAN インターフェイスのマニフェストの例です。これには、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: vlan-eth1-policy (1)
spec:
  nodeSelector: (2)
    kubernetes.io/hostname: <node01> (3)
  desiredState:
    interfaces:
    - name: eth1.102 (4)
      description: VLAN using eth1 (5)
      type: vlan (6)
      state: up (7)
      vlan:
        base-iface: eth1 (8)
        id: 102 (9)
```

1. ポリシーの名前。
2. オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
3. この例では、**hostname** ノードセレクターを使用します。
4. インターフェイスの名前。
5. オプション: 人間が判読できるインターフェイスの説明。
6. インターフェイスのタイプ。以下の例では VLAN を作成します。
7. 作成後のインターフェイスの要求された状態。
8. VLAN が接続されているノードの NIC。
9. VLAN タグ。

12.2.5.3. 例: ボンドインターフェイスノードネットワークの設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してノード上にボンドインターフェイスを作成します。

注記

OpenShift Virtualization は以下のボンドモードのみをサポートします。

- mode=1 active-backup
- mode=2 balance-xor
- mode=4 802.3ad
- mode=5 balance-tlb
- mode=6 balance-alb

以下の YAML ファイルは、ボンドインターフェイスのマニフェストの例です。これには、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: bond0-eth1-eth2-policy (1)
spec:
  nodeSelector: (2)
    kubernetes.io/hostname: <node01> (3)
  desiredState:
    interfaces:
    - name: bond0 (4)
      description: Bond with ports eth1 and eth2 (5)
      type: bond (6)
      state: up (7)
      ipv4:
        dhcp: true (8)
        enabled: true (9)
      link-aggregation:
        mode: active-backup (10)
        options:
          miimon: '140' (11)
        port: (12)
        - eth1
        - eth2
      mtu: 1450 (13)
```

(1) ポリシーの名前。

(2) オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。

(3) この例では、**hostname** ノードセレクターを使用します。

(4) インターフェイスの名前。

(5) オプション: 人間が判読できるインターフェイスの説明。

(6) インターフェイスのタイプ。この例では、ボンドを作成します。

7 作成後のインターフェイスの要求された状態。

8 オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。

9 この例では **ipv4** を有効にします。

10 ボンドのドライバーモード。この例では、アクティブなバックアップモードを使用します。

11 オプション: この例では、miimon を使用して 140ms ごとにボンドリンクを検査します。

12 ボンド内の下位ノードの NIC。

13 オプション: ボンドの Maximum transmission unit (MTU)指定がない場合、この値はデフォルトで **1500** に設定されます。

12.2.5.4. 例: イーサネットインターフェイスノードネットワークの設定ポリシー

NodeNetworkConfigurationPolicy マニフェストをクラスターに適用してクラスター内のノードにイーサネットインターフェイスを作成します。

以下の YAML ファイルは、イーサネットインターフェイスのマニフェストの例です。これには、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: eth1-policy 1
spec:
  nodeSelector: 2
    kubernetes.io/hostname: <node01> 3
  desiredState:
    interfaces:
    - name: eth1 4
      description: Configuring eth1 on node01 5
      type: ethernet 6
      state: up 7
      ipv4:
        dhcp: true 8
        enabled: true 9
```

1 ポリシーの名前。

2 オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。

3 この例では、**hostname** ノードセレクターを使用します。

4 インターフェイスの名前。

5 オプション: 人間が判読できるインターフェイスの説明。

6 インターフェイスのタイプ。この例では、イーサネットネットワークインターフェイスを作成します。

7 作成後のインターフェイスの要求された状態。

8 オプション: **dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。

9 この例では **ipv4** を有効にします。

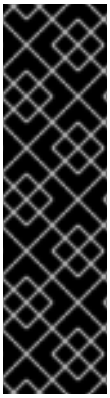
12.2.5.5. 例: 同じノードネットワーク設定ポリシーでの複数のインターフェイス

同じノードネットワーク設定ポリシーで複数のインターフェイスを作成できます。これらのインターフェイスは相互に参照でき、単一のポリシーマニフェストを使用してネットワーク設定をビルドし、デプロイできます。

以下のスニペット例では、2 つの NIC 間に **bond10** という名前のボンドと、ボンドに接続する **br1** という名前の Linux ブリッジを作成します。

```
#...
    interfaces:
    - name: bond10
      description: Bonding eth2 and eth3 for Linux bridge
      type: bond
      state: up
      link-aggregation:
        port:
        - eth2
        - eth3
    - name: br1
      description: Linux bridge on bond
      type: linux-bridge
      state: up
      bridge:
        port:
        - name: bond10
#...
```

12.2.6. ブリッジに接続された NIC の静的 IP の取得



重要

NIC の静的 IP のキャプチャーは、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。

12.2.6.1. 例: ブリッジに接続された NIC から静的 IP アドレスを継承する Linux ブリッジインターフェイスノードネットワーク設定ポリシー

クラスター内のノードに Linux ブリッジインターフェイスを作成し、単一の **NodeNetworkConfigurationPolicy** マニフェストをクラスターに適用して NIC の静的 IP 設定をブリッジに転送します。

以下の YAML ファイルは、Linux ブリッジインターフェイスのマニフェストの例です。これには、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-copy-ipv4-policy (1)
spec:
  nodeSelector: (2)
    node-role.kubernetes.io/worker: ""
  capture:
    eth1-nic: interfaces.name=="eth1" (3)
    eth1-routes: routes.running.next-hop-interface=="eth1"
    br1-routes: capture.eth1-routes | routes.running.next-hop-interface := "br1"
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port
        type: linux-bridge (4)
        state: up
        ipv4: "{{ capture.eth1-nic.interfaces.0.ipv4 }}" (5)
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: eth1 (6)
    routes:
      config: "{{ capture.br1-routes.routes.running }}"
```

(1) ポリシーの名前。

(2) オプション: **nodeSelector** パラメーターを含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。この例では **node-role.kubernetes.io/worker: ""** ノードセレクターを使用し、クラスター内のすべてのワーカーノードを選択します。

(3) ブリッジを接続するノード NIC への参照。

(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) ブリッジインターフェイスの IP アドレス。この値は、**spec.capture.eth1-nic** エントリーにより参照される NIC の IP アドレスと一致します。

(6) ブリッジが接続されるノードの NIC。

関連情報

- The NMPolicy project - Policy syntax

12.2.7. 例: IP 管理

以下の設定スニペットの例は、さまざまな IP 管理方法を示しています。

これらの例では、**ethernet** インターフェイスタイプを使用して、ポリシー設定に関連するコンテキストを表示しつつ、サンプルを単純化します。これらの IP 管理のサンプルは、他のインターフェイスタイプでも使用できます。

12.2.7.1. 静的

以下のスニペットは、イーサネットインターフェイスで IP アドレスを静的に設定します。

```
...
    interfaces:
    - name: eth1
      description: static IP on eth1
      type: ethernet
      state: up
      ipv4:
        dhcp: false
        address:
        - ip: 192.168.122.250 1
          prefix-length: 24
        enabled: true
...
```

1 この値を、インターフェイスの静的 IP アドレスに置き換えます。

12.2.7.2. IP アドレスなし

以下のスニペットでは、インターフェイスに IP アドレスがないことを確認できます。

```
...
    interfaces:
    - name: eth1
      description: No IP on eth1
      type: ethernet
      state: up
      ipv4:
        enabled: false
...
```

12.2.7.3. 動的ホストの設定

以下のスニペットは、動的 IP アドレス、ゲートウェイアドレス、および DNS を使用するイーサネットインターフェイスを設定します。

```
...
    interfaces:
    - name: eth1
      description: DHCP on eth1
      type: ethernet
      state: up
      ipv4:
```

```
      dhcp: true
      enabled: true
...
```

以下のスニペットは、動的 IP アドレスを使用しますが、動的ゲートウェイアドレスまたは DNS を使用しないイーサネットインターフェイスを設定します。

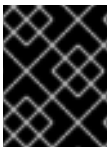
```
...
  interfaces:
  - name: eth1
    description: DHCP without gateway or DNS on eth1
    type: ethernet
    state: up
    ipv4:
      dhcp: true
      auto-gateway: false
      auto-dns: false
      enabled: true
...
```

12.2.7.4. DNS

DNS 設定の定義は、**/etc/resolv.conf** ファイルの変更に類似しています。以下のスニペットは、ホストに DNS 設定を定義します。

```
...
  interfaces: 1
    ...
    ipv4:
      ...
      auto-dns: false
      ...
  dns-resolver:
    config:
      search:
      - example.com
      - example.org
      server:
      - 8.8.8.8
...
```

1 Kubernetes NMState がカスタム DNS 設定を保存するためには、インターフェイスを **auto-dns: false** で設定するか、インターフェイスで静的 IP 設定を使用する必要があります。



重要

DNS リゾルバーの設定時に、インターフェイスとして OVNKubernetes が管理する Open vSwitch ブリッジである **br-ex** を使用することはできません。

12.2.7.5. 静的ルーティング

以下のスニペットは、インターフェイス **eth1** に静的ルートおよび静的 IP を設定します。

```
...
    interfaces:
    - name: eth1
      description: Static routing on eth1
      type: ethernet
      state: up
      ipv4:
        dhcp: false
        address:
        - ip: 192.0.2.251 1
          prefix-length: 24
        enabled: true
    routes:
      config:
      - destination: 198.51.100.0/24
        metric: 150
        next-hop-address: 192.0.2.1 2
        next-hop-interface: eth1
        table-id: 254
...
```

1 イーサネットインターフェイスの静的 IP アドレス。

2 ノードトラフィックのネクストホップアドレス。これは、イーサネットインターフェイスに設定される IP アドレスと同じサブネットにある必要があります。

12.3. ノードのネットワーク設定のトラブルシューティング

ノードのネットワーク設定で問題が発生した場合には、ポリシーが自動的にロールバックされ、enactment (実行) レポートは失敗します。これには、以下のような問題が含まれます。

- ホストで設定を適用できません。
- ホストはデフォルトゲートウェイへの接続を失います。
- ホストは API サーバーへの接続を失います。

12.3.1. 正確でないノードネットワーク設定のポリシー設定のトラブルシューティング

ノードネットワーク設定ポリシーを適用し、クラスター全体でノードのネットワーク設定への変更を適用することができます。正確でない設定を適用する場合、以下の例を使用して、失敗したノードネットワークポリシーのトラブルシューティングと修正を行うことができます。

この例では、Linux ブリッジポリシーは、3 つのコントロールプレーンノード (マスター) と 3 つのコンピュート (ワーカー) ノードを持つクラスターのサンプルに適用されます。ポリシーは正しくないインターフェイスを参照するために、適用することができません。エラーを確認するには、利用可能な NMState リソースを調べます。その後に、正しい設定でポリシーを更新できます。

手順

1. ポリシーを作成し、これをクラスターに適用します。以下の例では、**ens01** インターフェイスに単純なブリッジを作成します。

```
apiVersion: nmstate.io/v1
```

```
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ens01-bridge-testfail
spec:
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with the wrong port
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: ens01
```

```
$ oc apply -f ens01-bridge-testfail.yaml
```

出力例

```
nodenetworkconfigurationpolicy.nmstate.io/ens01-bridge-testfail created
```

2. 以下のコマンドを実行してポリシーのステータスを確認します。

```
$ oc get nncp
```

この出力は、ポリシーが失敗したことを示しています。

出力例

```
NAME                  STATUS
ens01-bridge-testfail   FailedToConfigure
```

ただし、ポリシーのステータスのみでは、すべてのノードで失敗したか、ノードのサブセットで失敗したかを確認することはできません。

3. ノードのネットワーク設定の enactment (実行) を一覧表示し、ポリシーがいずれかのノードで成功したかどうかを確認します。このポリシーがノードのサブセットに対してのみ失敗した場合は、問題が特定のノード設定にあることが示唆されます。このポリシーがすべてのノードで失敗した場合には、問題はポリシーに関連するものであることが示唆されます。

```
$ oc get nnce
```

この出力は、ポリシーがすべてのノードで失敗したことを示しています。

出力例

```
NAME                                STATUS
control-plane-1.ens01-bridge-testfail       FailedToConfigure
```

```
control-plane-2.ens01-bridge-testfail       FailedToConfigure
control-plane-3.ens01-bridge-testfail       FailedToConfigure
compute-1.ens01-bridge-testfail             FailedToConfigure
compute-2.ens01-bridge-testfail             FailedToConfigure
compute-3.ens01-bridge-testfail             FailedToConfigure
```

4. 失敗した enactment (実行) のいずれかを表示し、トレースバックを確認します。以下のコマンドは、出力ツール **jsonpath** を使用して出力をフィルターします。

```
$ oc get nnce compute-1.ens01-bridge-testfail -o jsonpath='{.status.conditions[?
(@.type=="Failing")].message}'
```

このコマンドは、簡潔にするために編集されている大きなトレースバックを返します。

出力例

```
error reconciling NodeNetworkConfigurationPolicy at desired state apply: , failed to execute
nmstatectl set --no-commit --timeout 480: 'exit status 1' "
...
libnmstate.error.NmstateVerificationError:
desired
=======
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
    stp:
      enabled: false
      forward-delay: 15
      hello-time: 2
      max-age: 20
      priority: 32768
  port:
  - name: ens01
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

current
=======
---
```

```
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
    stp:
      enabled: false
      forward-delay: 15
      hello-time: 2
      max-age: 20
      priority: 32768
  port: []
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

difference
==========
--- desired
+++ current
@@ -13,8 +13,7 @@
       hello-time: 2
       max-age: 20
       priority: 32768
-  port:
-  - name: ens01
+  port: []
 description: Linux bridge with the wrong port
 ipv4:
   address: []
  line 651, in _assert_interfaces_equal\n
current_state.interfaces[ifname],\nlibnmstate.error.NmstateVerificationError:
```

NmstateVerificationError は、**desired** ポリシー設定、ノード上のポリシーの **current** 設定、および一致しないパラメーターを強調表示する **difference** を一覧表示します。この例では、**port** は **difference** に組み込まれ、これは問題がポリシーのポート設定に関連するものであることを示唆します。

5. ポリシーが適切に設定されていることを確認するには、**NodeNetworkState** オブジェクトを要求して、1つまたはすべてのノードのネットワーク設定を表示します。以下のコマンドは、**control-plane-1** ノードのネットワーク設定を返します。

```
$ oc get nns control-plane-1 -o yaml
```

出力は、ノード上のインターフェイス名は **ens1** であるものの、失敗したポリシーが **ens01** を誤って使用していることを示します。

出力例

```
  - ipv4:
...
    name: ens1
    state: up
    type: ethernet
```

6. 既存のポリシーを編集してエラーを修正します。

```
$ oc edit nncp ens01-bridge-testfail
```

```
...
        port:
          - name: ens1
```

ポリシーを保存して修正を適用します。

7. ポリシーのステータスをチェックして、更新が正常に行われたことを確認します。

```
$ oc get nncp
```

出力例

```
NAME                    STATUS
ens01-bridge-testfail   SuccessfullyConfigured
```

更新されたポリシーは、クラスターのすべてのノードで正常に設定されました。

第13章 ロギング、イベント、およびモニタリング

13.1. 仮想化の概要の確認

Virtualization Overview ページには、仮想化リソース、詳細、ステータス、および上位消費項目の包括的なビューが提供されます。OpenShift Virtualization の全体的な健全性に関する洞察を得ることで、データを調べることで特定される特定の問題を解決するために介入が必要かどうかを判断できます。

Getting Started リソースを使用して、クイックスタートにアクセスし、仮想化に関する最新ブログを読み、Operator の使用方法を学習します。アラート、イベント、インベントリー、および仮想マシンのステータスに関する詳細情報を取得します。**Top Consumer** カードをカスタマイズして、プロジェクト、仮想マシン、またはノード別に特定のリソースの高使用率に関するデータを取得します。**View virtualization dashboard** をクリックして、Dashboards ページにすばやくアクセスできます。

13.1.1. 前提条件

Top Consumers カードで **vCPU wait** メトリクスを使用するには、**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。カーネル引数の適用に関する詳細は、OpenShift Container Platform マシン設定タスク のドキュメントを参照してください。

13.1.2. Virtualization Overview ページでアクティブに監視されるリソース

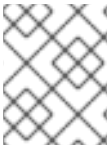
以下の表に、**Virtualization Overview** ページでアクティブに監視されるリソース、メトリクス、およびフィールドを示します。この情報は、関連データを取得し、問題を解決するために介入する必要がある場合に役立ちます。

監視されるリソース、フィールド、およびメトリクス	説明
Details	**OpenShift Virtualization** のサービスの概要とバージョン情報。
Status	仮想化およびネットワークに関するアラート。
Activity	仮想マシンの現在のイベント。メッセージは、Pod の作成または別のホストへの仮想マシンの移行など、クラスター内の直近のアクティビティーに関連します。
Running VMs by Template	ドーナツチャートは、各仮想マシンテンプレートに固有の色を示し、各テンプレートを使用する実行中の仮想マシンの数を表示します。
Inventory	アクティブな仮想マシン、テンプレート、ノード、ネットワークの合計数。
Status of VMs	仮想マシンの現在の状態: **running**、**provisioning**、**starting**、**migrating**、**paused**、**stopping**、**terminating**、および **unknown**

Permissions	パーミッションにより機能を有効にするタスク: **Access to public templates**、**Access to public boot sources**、**Clone a VM**、**Attach VM to multiple networks**、**Upload a base image from local disk**、および **Share templates**

13.1.3. 上位の消費が監視されるリソース

Virtualization Overview ページの **Top Consumers** カードには、リソースの最大消費と共にプロジェクト、仮想マシン、またはノードが表示されます。プロジェクト、仮想マシン、またはノードを選択し、特定リソースの上位 5 または上位 10 の消費項目を表示できます。



注記

最大リソース消費の表示は、各 **Top Consumers** カード内で上位 5 または上位 10 の消費項目に制限されます。

以下の表に、上位消費項目が監視されるリソースを示します。

上位の消費が監視されるリソース	説明
CPU	最も多くの CPU を使用するプロジェクト、仮想マシン、またはノード。
メモリー	最も多くのメモリー (バイト単位) を使用するプロジェクト、仮想マシン、またはノード。表示の単位 (MiB または GiB など) は、リソース消費のサイズにより決定されます。
使用済みのファイルシステム	ファイルシステムの消費 (バイト単位) が最も多いプロジェクト、仮想マシン、またはノード。表示の単位 (MiB または GiB など) は、リソース消費のサイズにより決定されます。
メモリースワップ	メモリーがスワップされる際に、メモリープレッシャーが最も多いプロジェクト、仮想マシン、またはノード。
vCPU 待機時間	仮想 CPU の待ち時間 (秒単位) が最も長いプロジェクト、仮想マシン、またはノード。
ストレージスループット	ストレージメディアとの間のデータ転送率 (mbps 単位) が最も高いプロジェクト、仮想マシン、またはノード。
ストレージ IOPS	一定期間におけるストレージ IOPS(input/output operations per second) が最も高いプロジェクト、仮想マシン、またはノード。

13.1.4. 上位消費プロジェクト、仮想マシン、およびノードの確認

Virtualization Overview ページで、選択したプロジェクト、仮想マシン、またはノードのリソースの上位消費項目を表示できます。

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アクセスできる。

手順

1. OpenShift Virtualization Web コンソールの **Administrator** パースペクティブで、**Virtualization → Overview** に移動します。
2. **Top Consumers** カードに移動します。
3. ドロップダウンメニューから、**Show top 5** または **Show top 10** を選択します。
4. **Top Consumer** カードで、ドロップダウンメニューからリソースのタイプを選択します (**CPU**、**Memory**、**Used Filesystem**、**Memory Swap**、**vCPU Wait**、または **Storage Throughput**)。
5. **By Project**、**By VM**、または **By Node** を選択します。選択したリソースの上位 5 または上位 10 消費項目の一覧が表示されます。

13.1.5. 関連情報

- モニタリングの概要
- モニタリングダッシュボードの確認
- ダッシュボード

13.2. 仮想マシンログの表示

13.2.1. 仮想マシンのログについて

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

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

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

警告

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

13.2.2. CLI での仮想マシンログの表示

仮想マシンランチャー Pod から仮想マシンログを取得します。

手順

- 以下のコマンドを使用します。

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

13.2.3. Web コンソールでの仮想マシンログの表示

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

手順

1. 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. **Details** タブをクリックします。
4. **Pod** セクションの **virt-launcher-<name>** Pod をクリックして、**Pod details** ページを開きます。
5. **Logs** タブをクリックして、Pod のログを表示します。

13.3. イベントの表示

13.3.1. 仮想マシンイベントについて

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

OpenShift Virtualization は、仮想マシンおよび仮想マシンインスタンスについてのイベントを追加します。これらは Web コンソールまたは CLI のいずれかで表示できます。

OpenShift Container Platform クラスター内のシステムイベント情報の表示 も参照してください。

13.3.2. Web コンソールでの仮想マシンのイベントの表示

Web コンソールの **VirtualMachine details** ページで、実行中の仮想マシンのストリーミング イベントを表示できます。

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. **Events** タブをクリックして、仮想マシンのストリーミングイベントを表示します。
   - ▮▮ ボタンはイベントストリームを一時停止します。
   - ▶ ボタンは、一時停止したイベントストリームを再開します。

13.3.3. CLI での namespace イベントの表示

OpenShift Container Platform クライアントを使用して namespace のイベントを取得します。

手順

- namespace で、**oc get** コマンドを使用します。

```
$ oc get events
```

13.3.4. CLI でのリソースイベントの表示

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

手順

- namespace で、**oc describe** コマンドを使用します。以下の例は、仮想マシン、仮想マシンインスタンス、および仮想マシンの virt-launcher Pod のイベント取得する方法を示しています。

```
$ oc describe vm <vm>
```

```
$ oc describe vmi <vmi>
```

```
$ oc describe pod virt-launcher-<name>
```

13.4. イベントおよび状態を使用したデータボリュームの診断

oc describe コマンドを使用してデータボリュームの問題を分析し、解決できるようにします。

13.4.1. 状態およびイベントについて

コマンドで生成される **Conditions** および **Events** セクションの出力を検査してデータボリュームの問題を診断します。

```
$ oc describe dv <DataVolume>
```

表示される **Conditions** セクションには、3 つの **Types** があります。

- **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。

イベントは **Status**、**Reason**、または **Message** のいずれかの変更時に生成されます。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。

13.4.2. 状態およびイベントを使用したデータボリュームの分析

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、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ついての特定の詳細、およびどのように現在の状態になったかについての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** であり、**Status** が **False** であることに注意してください。これは、操作の失敗の原因となったイベントが発生したことを示しています。ステータスを **True** から **False** に変更します。
  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
```

13.5. 仮想マシンのワークロードに関する情報の表示

OpenShift Container Platform Web コンソールで **Virtual Machines** ダッシュボードを使用して、仮想マシンについての概要を表示できます。

13.5.1. Virtual Machines ダッシュボードについて

Virtualization → VirtualMachines ページに移動し、仮想マシン (VM) をクリックして **VirtualMachine details** ページを表示することにより、OpenShift Container Platform Web コンソールから仮想マシン (VM) にアクセスします。

Overview タブには、次のカードが表示されます。

- **Details**: 以下を含む、仮想マシンについての識別情報を提供します。
  - 名前
  - namespace
  - 作成日
  - ノード名
  - IP アドレス
- **Inventory**: 以下を含む、仮想マシンのリソースを一覧表示します。
  - ネットワークインターフェイスコントローラー (NIC)
  - ディスク
- **Status** には、以下が含まれます。
  - 仮想マシンの現在の状態
  - QEMU ゲストエージェントが仮想マシンにインストールされているかどうかを示す注記
- **Utilization**: 以下の使用状況についてのデータを表示するチャートが含まれます。
  - CPU
  - メモリー
  - Filesystem
  - ネットワーク転送

注記

ドロップダウンリストを使用して、使用状況データの期間を選択します。選択できるオプションは、**1 Hour**、**6 Hours**、および **24 Hours** です。

- **Events**: 過去 1 時間における仮想マシンのアクティビティーについてのメッセージを一覧表示します。追加のイベントを表示するには、**View all** をクリックします。

13.6. 仮想マシンの正常性のモニタリング

仮想マシンインスタンス (VMI) は、接続の損失、デッドロック、または外部の依存関係に関する問題など、一時的な問題が原因で正常でなくなることがあります。ヘルスチェックは、readiness および liveness プローブの組み合わせを使用して VMI で診断を定期的に実行します。

13.6.1. readiness および liveness プローブについて

readiness および liveness プローブを使用して、正常でない仮想マシンインスタンス (VMI) を検出して処理します。VMI の仕様に 1 つ以上のプローブを追加して、トラフィックが準備状態にない VMI に到達せず、VMI が応答不能になると新規インスタンスが作成されるようにすることができます。

readiness プローブ は、VMI 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別します。プローブに失敗すると、VMI は準備状態になるまで、利用可能なエンドポイントの一覧から削除されます。

liveness プローブは、VMI が応答しているかどうかを判断します。プローブに失敗すると、VMI が削除され、新規インスタンスが作成されて応答性を復元します。

VirtualMachineInstance オブジェクトの **spec.readinessProbe** と **spec.livenessProbe** フィールドを設定して、readiness および 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 VMI の正常性を判別します。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、VMI に対してソケットを開くことを試行します。VMI はプローブで接続を確立できる場合にのみ正常であるとみなされます。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。

13.6.2. HTTP readiness プローブの定義

仮想マシンインスタンス (VMI) 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. VMI 設定ファイルに readiness プローブの詳細を追加します。

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. VMI への接続に使用する HTTP GET 要求。
2. プローブがクエリーする VMI 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す場合に、VMI は正常であるとみなされます。ハンドラーが失敗コードを返すと、VMI は利用可能なエンドポイントの一覧から削除されます。
4. VMI が起動してから readiness 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し、VMI が失敗したと想定されてから非アクティブになるまでの時間 (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. 以下のコマンドを実行して VMI を作成します。

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

13.6.3. TCP readiness プローブの定義

仮想マシンインスタンス (VMI) 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 VMI 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. VMI が起動してから readiness 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 VMI のポート。
5. プローブがタイムアウトし、VMI が失敗したと想定されてから非アクティブになるまでの時間 (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. 以下のコマンドを実行して VMI を作成します。

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

13.6.4. HTTP liveness プローブの定義

仮想マシンインスタンス (VMI) 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. HTTP liveness プローブの詳細を VMI 設定ファイルに追加します。

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. VMI 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. VMI への接続に使用する HTTP GET 要求。
4. プローブがクエリーする VMI 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。VMI は、cloud-init 経由でポート 1500 に最小限の HTTP サーバーをインストールし、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す場合に、VMI は正常であるとみなされます。ハンドラーが失敗コードを返すと、VMI が削除され、新規インスタンスが作成されます。

6 プローブがタイムアウトし、VMI が失敗したと想定されてから非アクティブになるまでの時間 (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. 以下のコマンドを実行して VMI を作成します。

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

13.6.5. テンプレート: ヘルスチェックを定義するための仮想マシンの設定ファイル

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    special: vm-fedora
  name: vm-fedora
spec:
  template:
    metadata:
      labels:
        special: vm-fedora
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: containerdisk
          - disk:
              bus: virtio
            name: cloudinitdisk
        resources:
          requests:
            memory: 1024M
      readinessProbe:
        httpGet:
          port: 1500
        initialDelaySeconds: 120
        periodSeconds: 20
        timeoutSeconds: 10
        failureThreshold: 3
        successThreshold: 3
      terminationGracePeriodSeconds: 180
      volumes:
      - name: containerdisk
        containerDisk:
          image: kubevirt/fedora-cloud-registry-disk-demo
      - cloudInitNoCloud:
          userData: |-
```

```
        #cloud-config
        password: fedora
        chpasswd: { expire: False }
        bootcmd:
          - setenforce 0
          - dnf install -y nmap-ncat
          - systemd-run --unit=httpserver nc -klp 1500 -e '/usr/bin/echo -e HTTP/1.1 200 OK\\n\\nHello
World!'
      name: cloudinitdisk
```

13.6.6. 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

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

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

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

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

OpenShift Container Platform ダッシュボードは以下のカードで設定されます。

- **Details** は、クラスターの詳細情報の概要を表示します。
  ステータスには、**ok**、**error**、**warning**、**in progress**、および **unknown** が含まれます。リソースでは、カスタムのステータス名を追加できます。
  - クラスター
  - プロバイダー
  - バージョン
- **Cluster Inventory** は、リソースの数および関連付けられたステータスの詳細を表示します。これは、問題の解決に介入が必要な場合に役立ちます。以下についての情報が含まれます。
  - ノード数
  - Pod 数
  - 永続ストレージボリューム要求
  - 仮想マシン (OpenShift Virtualization がインストールされている場合に利用可能)
  - クラスター内のベアメタルホスト。これらはステータス別に一覧表示されます (**metal3** 環境でのみ利用可能)。
- **Cluster Health** では、関連するアラートおよび説明を含む、クラスターの現在の健全性についてのサマリーを表示します。OpenShift Virtualization がインストールされている場合、OpenShift Virtualization の健全性についても全体的に診断されます。複数のサブシステムが存

在する場合は、**See All** をクリックして、各サブシステムのステータスを表示します。

- **Cluster Capacity** グラフは、管理者が追加リソースがクラスターで必要になるタイミングを把握するのに役立ちます。このグラフには、現在の消費量を表示する内側の円が含まれ、外側の円には、以下の情報を含む、リソースに対して設定されたしきい値が表示されます。
  - CPU 時間
  - メモリー割り当て
  - 消費されるストレージ
  - 消費されるネットワークリソース
- **Cluster Utilization** は指定された期間における各種リソースの容量を表示します。これは、管理者がリソースの高い消費量の規模および頻度を理解するのに役立ちます。
- **Events** は、Pod の作成または別のホストへの仮想マシンの移行などのクラスター内の最近のアクティビティーに関連したメッセージを一覧表示します。
- **Top Consumers** は、管理者がクラスターリソースの消費状況を把握するのに役立ちます。リソースをクリックし、指定されたクラスターリソース (CPU、メモリー、またはストレージ) の最大量を消費する Pod およびノードを一覧表示する詳細ページに切り替えます。

13.8. 仮想マシンによるリソース使用率の確認

OpenShift Container Platform Web コンソールのダッシュボードには、クラスターの状態をすぐに理解できるようにするクラスターのメトリックの視覚的な表示が含まれます。ダッシュボードは、コアプラットフォームコンポーネントを監視する Monitoring overview に属します。

OpenShift Virtualization ダッシュボードでは、仮想マシンおよび関連付けられた Pod のリソース消費に関するデータが得られます。OpenShift Virtualization ダッシュボードに表示される可視化メトリクスは、Prometheus Query Language(PromQL) クエリー に基づいています。

OpenShift Virtualization ダッシュボードでユーザー定義の namespace をモニターするには、モニタリングロール が必要です。

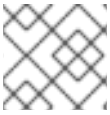
13.8.1. トップコンシューマーの確認について

OpenShift Virtualization ダッシュボードでは、特定の期間を選択して、その期間内のリソースのトップコンシューマーを表示できます。トップコンシューマーとは、最も多くリソースを消費する仮想マシンまたは **virt-launcher** Pod のことです。

以下の表は、ダッシュボードでモニターされたリソースと、トップコンシューマーの各リソースに関連付けられたメトリックを示しています。

モニタリングされたリソース	Description
メモリースワップトラフィック	仮想マシンは、メモリーのスワップ時に最も多くのメモリーを消費します。
vCPU 待機時間	vCPU の待機時間 (秒単位) が最大の仮想マシン。

Pod 別の CPU 使用率	CPU を最も使用している **virt-launcher** Pod。
ネットワークトラフィック	最も多くのネットワークトラフィック (バイト単位) を受信することによってネットワークを飽和させている仮想マシン。
ストレージトラフィック	ストレージ関連のトラフィック量 (バイト単位) が最大の仮想マシン。
ストレージ IOPS	一定期間における 1 秒あたりの I/O 操作が最も多い仮想マシン。
メモリー使用率	メモリー (バイト単位) を最も使用している **virt-launcher** Pod。



注記

リソース消費の最大値の表示は、トップコンシューマー 5 件に制限されます。

13.8.2. トップコンシューマーの確認

Administrator パースペクティブでは、リソースのトップコンシューマーが表示される OpenShift Virtualization ダッシュボードを確認できます。

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アクセスできる。

手順

1. OpenShift Virtualization Web コンソールの **Administrator** パースペクティブで、**Observe → Dashboards** に移動します。
2. **Dashboard** 一覧から **KubeVirt/Infrastructure Resources/Top Consumers** ダッシュボードを選択します。
3. **Period** のドロップダウンメニューから事前に定義された期間を選択します。表でトップコンシューマーのデータを確認できます。
4. オプション:**Inspect** をクリックして、テーブルの上部コンシューマーに関連付けられた Prometheus クエリー言語 (PromQL) クエリーを表示または編集します。

13.8.3. 関連情報

- モニタリングの概要
- モニタリングダッシュボードの確認

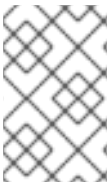
13.9. OPENSHIFT CONTAINER PLATFORM クラスターモニタリング、ロギング、および TELEMETRY

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

13.9.1. OpenShift Container Platform モニタリングについて

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。OpenShift Container Platform は、追加設定が不要のモニタリングのベストプラクティスを提供します。クラスター管理者にクラスターの問題について即時に通知するアラートのセットがデフォルトで含まれます。OpenShift Container Platform Web コンソールのデフォルトのダッシュボードには、クラスターの状態をすぐに理解できるようにするクラスターのメトリックの視覚的な表示が含まれます。

OpenShift Container Platform 4.10 のインストール後に、クラスター管理者はオプションで**ユーザー定義プロジェクトのモニタリング**を有効にできます。この機能を使用することで、クラスター管理者、開発者、および他のユーザーは、サービスと Pod を独自のプロジェクトでモニターする方法を指定できます。次に、OpenShift Container Platform Web コンソールでメトリックのクエリー、ダッシュボードの確認、および独自のプロジェクトのアラートルールおよびサイレンスを管理できます。



注記

クラスター管理者は、開発者およびその他のユーザーに、独自のプロジェクトをモニターするパーミッションを付与できます。事前に定義されたモニタリングロールのいずれかを割り当てると、特権が付与されます。

13.9.2. サブシステムコンポーネントのロギングについて

ロギングシステムコンポーネントには、すべてのノードおよびコンテナーログを収集し、それらをログストアに書き込む OpenShift Container Platform クラスターの各ノードにデプロイされるコレクターが含まれます。一元化された Web UI を使用し、集計されたデータを使用して高度な可視化 (visualization) およびダッシュボードを作成できます。

ロギングサブシステムの主なコンポーネントは次のとおりです。

- collection: これは、クラスターからログを収集し、それらをフォーマットし、ログストアに転送するコンポーネントです。現在の実装は Fluentd です。
- log store: これはログが保存される場所です。デフォルトの実装は Elasticsearch です。デフォルトの Elasticsearch ログストアを使用、またはログを外部ログストアに転送できます。デフォルトのログストアは、短期の保存について最適化され、テストされています。
- visualization: これは、ログ、グラフ、グラフなどを表示するために使用される UI コンポーネントです。現在の実装は Kibana です。

OpenShift Logging の詳細は、OpenShift Logging のドキュメント を参照してください。

13.9.3. Telemetry について

Telemetry は厳選されたクラスターモニタリングメトリクスのサブセットを Red Hat に送信します。Telemeter Client はメトリクスの値を 4 分 30 秒ごとに取得し、データを Red Hat にアップロードします。これらのメトリクスについては、このドキュメントで説明しています。

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

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

13.9.3.1. Telemetry で収集される情報

以下の情報は、Telemetry によって収集されます。

13.9.3.1.1. システム情報

- OpenShift Container Platform クラスターのバージョン情報、および更新バージョンの可用性を特定するために使用されるインストールの更新の詳細を含むバージョン情報
- クラスターごとに利用可能な更新の数、更新に使用されるチャネルおよびイメージリポジトリー、更新の進捗情報、および更新で発生するエラーの数などの更新情報
- インストール時に生成される一意でランダムな識別子
- クラウドインフラストラクチャーレベルのノード設定、ホスト名、IP アドレス、Kubernetes Pod 名、namespace、およびサービスなど、Red Hat サポートがお客様にとって有用なサポートを提供するのに役立つ設定の詳細
- クラスターにインストールされている OpenShift Container Platform フレームワークコンポーネントおよびそれらの状態とステータス
- 動作が低下した Operator の関連オブジェクトとして一覧表示されるすべての namespace のイベント
- 動作が低下したソフトウェアに関する情報
- 証明書の有効性についての情報
- OpenShift Container Platform がデプロイされているプラットフォームの名前およびデータセンターの場所

13.9.3.1.2. サイジング情報

- CPU コアの数およびそれぞれに使用される RAM の容量を含む、クラスター、マシンタイプ、およびマシンについてのサイジング情報
- クラスター内での実行中の仮想マシンインスタンスの数
- etcd メンバーの数および etcd クラスターに保存されるオブジェクトの数
- ビルドストラテジータイプ別のアプリケーションビルドの数

13.9.3.1.3. 使用情報

- コンポーネント、機能および拡張機能に関する使用率の情報
- テクノロジープレビューおよびサポート対象外の設定に関する使用率の詳細

Telemetry は、ユーザー名やパスワードなどの識別情報を収集しません。Red Hat は、意図的な個人情報の収集は行いません。誤って個人情報を受信したことが明らかになった場合、Red Hat はその情報を削除します。Telemetry データが個人データを設定する場合において、Red Hat のプライバシー方針に

ついては、Red Hat Privacy Statement を参照してください。

13.9.4. CLI のトラブルシューティングおよびデバッグコマンド

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

13.10. 仮想リソースの PROMETHEUS クエリー

OpenShift Virtualization は、インフラストラクチャーリソースがクラスターで消費される方法を監視するためのメトリックを提供します。メトリックでは以下のリソースを対象とします。

- vCPU
- ネットワーク
- ストレージ
- ゲストメモリーのスワップ

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。

13.10.1. 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。カーネル引数の適用に関する詳細は、OpenShift Container Platform マシン設定タスク のドキュメント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。

13.10.2. メトリクスのクエリー

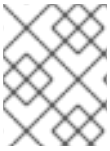
OpenShift Container Platform モニタリングダッシュボードでは、Prometheus のクエリー言語 (PromQL) クエリーを実行し、プロットに可視化されるメトリクスを検査できます。この機能により、クラスターの状態と、モニターしているユーザー定義のワークロードに関する情報が提供されます。

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ックをクエリーできます。

開発者 として、メトリック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ックを表示するには、必要な権限が必要です。

13.10.2.1. 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ックのクエリー

クラスター管理者またはすべてのプロジェクトの表示パーミッション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。



注記

クラスター管理者のみが、OpenShift Container Platform Monitoring で提供されるサードパーティーの UI 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パーミッション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 コンソール内の **Administrator** パースペクティブで、**Observe → Metrics** を選択します。
2. **Insert Metric at Cursor** を選択し、事前に定義されたクエリーの一覧を表示します。
3. カスタムクエリーを作成するには、Prometheus クエリー言語 (PromQL) のクエリーを **Expression** フィールドに追加します。
4. 複数のクエリーを追加するには、**Add Query** を選択します。
5. クエリーを削除するには、クエリーの横にある  を選択してから **Delete query** を選択します。
6. クエリーの実行を無効にするには、クエリーの横にある  を選択してから **Disable query** を選択します。
7. **Run Queries** を選択し、作成したクエリーを実行します。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、メトリック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。

8. 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。

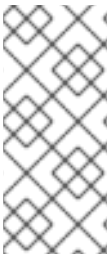
関連情報

- PromQL クエリーの作成に関する詳細は、Prometheus クエリーについてのドキュメント を参照してください。

13.10.2.2. 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ックには、開発者またはプロジェクトの表示パーミッション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ックについてカスタム Prometheus Query Language (PromQL) クエリーを実行することもできます。



注記

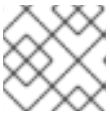
開発者は **Developer** パースペクティブのみを使用でき、**Administrator** パースペクティブは使用できません。開発者は、1 度に 1 つのプロジェクトのメトリクスのみをクエリーできます。開発者はコアプラットフォームコンポーネント用の OpenShift Container Platform モニタリングで提供されるサードパーティーの UI にアクセスできません。その代わりとして、ユーザー定義プロジェクトにメトリクス UI を使用します。

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パーミッション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を有効にしている。
- 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:** 一覧でメトリック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** を選択してカスタム PromQL クエリーを実行します。



注記

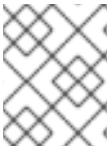
Developer パースペクティブでは、1 度に 1 つのクエリーのみを実行できます。

関連情報

- PromQL クエリーの作成に関する詳細は、Prometheus クエリーについてのドキュメント を参照してください。

13.10.3. 仮想化メトリクス

以下のメトリックの記述には、Prometheus Query Language (PromQL) クエリーのサンプルが含まれます。これらのメトリックは 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。

13.10.3.1. 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。

0 より大きい値は、vCPU 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。

13.10.3.2. 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。

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。

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。

13.10.3.3. ストレージメトリック

13.10.3.3.1. 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。

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。

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。

13.10.3.3.2. 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。

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。

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。

13.10.3.4. ゲストメモリーのスワップメトリック

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。

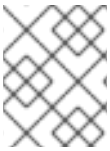
kubevirt_vmi_memory_swap_out_traffic_bytes_total

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。

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。

13.10.4. 関連情報

- モニタリングの概要

13.11. 仮想マシンのカスタムメトリックの公開

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、メトリック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。

13.11.1. 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、メトリック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ックおよびプロセスを公開します。

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します。
- **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します。

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ックを公開する 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 でメトリック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
```

13.11.2. 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ック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
```

13.11.3. 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**) 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。
- 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。

13.11.3.1. メトリックを取得するための node-exporter サービスのクエリー

仮想マシンのメトリックは、/**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。メトリックのクエリー時に、Prometheus は仮想マシンによって公開されるメトリックエンドポイントからメトリックを直接収集し、これらのメトリック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ック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
```

13.11.4. 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ックを収集して、node-exporter サービスが公開するメトリックにアクセスし、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
```

13.11.4.1. 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、公開されるメトリックを表示できます。

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 サービスのメトリックを表示します。

```
$ curl -s http://node-exporter-service-dynamation.apps.cluster.example.org/metrics
```

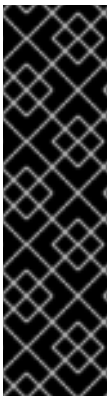
出力例

```
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
```

13.11.5. 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

13.12. OPENSHIFT VIRTUALIZATION の重大なアラート



重要

OpenShift Virtualization の重大なアラートは、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 Virtualization には、問題が発生したときに通知するアラートがあります。重大なアラートには早急な対応が必要です。

各アラートには、問題に対する説明、アラートが発生した理由、問題の原因を診断するためのトラブルシューティングプロセス、およびアラートの解決手順が含まれます。

13.12.1. ネットワークアラート

ネットワークアラートは、OpenShift Virtualization Network Operator の問題についての情報を提供します。

13.12.1.1. KubeMacPoolDown アラート

説明

KubeMacPool コンポーネントは MAC アドレスを割り当て、MAC アドレスの競合を防ぎます。

理由

KubeMacPool-manager Pod が停止すると、**VirtualMachine** オブジェクトの作成に失敗します。

トラブルシューティング

1. Kubemacpool-manager Pod の namespace および名前を把握します。

```
$ export KMP_NAMESPACE="$(oc get pod -A --no-headers -l control-plane=mac-controller-manager | awk '{print $1}')"
```

```
$ export KMP_NAME="$(oc get pod -A --no-headers -l control-plane=mac-controller-manager | awk '{print $2}')"
```

2. Kubemacpool-manager Pod の説明およびログを確認して、問題の原因を判断します。

```
$ oc describe pod -n $KMP_NAMESPACE $KMP_NAME
```

```
$ oc logs -n $KMP_NAMESPACE $KMP_NAME
```

解決方法

サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.2. SSP アラート

SSP アラートは、OpenShift Virtualization SSP Operator の問題についての情報を提供します。

13.12.2.1. SSPFailingToReconcile アラート

説明

SSP Operator の Pod が起動しているが、Pod の調整サイクルが必ず失敗する。この失敗には、該当するリソースの更新の失敗、テンプレートバリデータのデプロイの失敗、共通テンプレートのデプロイまたは更新の失敗が含まれます。

理由

SSP Operator が調整に失敗すると、依存するコンポーネントのデプロイメントに失敗するか、コンポーネント変更の調整に失敗するか、あるいはその両方に失敗します。さらに、共通テンプレートおよびテンプレートバリデーターの更新がリセットされ、失敗します。

トラブルシューティング

1. ssp-operator Pod のログでエラーの有無を確認します。

```
$ export NAMESPACE="$(oc get deployment -A | grep ssp-operator | awk '{print $1}')"
```

```
$ oc -n $NAMESPACE describe pods -l control-plane=ssp-operator
```

```
$ oc -n $NAMESPACE logs --tail=-1 -l control-plane=ssp-operator
```

2. テンプレートバリデーターが起動していることを確認します。テンプレートバリデーターが起動していない場合は、Pod のログでエラーの有無を確認します。

```
$ export NAMESPACE="$($ oc get deployment -A | grep ssp-operator | awk '{print $1}')"
```

```
$ oc -n $NAMESPACE get pods -l name=virt-template-validator
```

```
$ oc -n $NAMESPACE describe pods -l name=virt-template-validator
```

```
$ oc -n $NAMESPACE logs --tail=-1 -l name=virt-template-validator
```

解決方法

サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.2.2. SSPOperatorDown アラート

説明

SSP Operator は、共通テンプレートおよびテンプレートバリデーターをデプロイし、調整します。

理由

SSP Operator が停止するると、依存するコンポーネントのデプロイメントに失敗するか、コンポーネント変更の調整に失敗するか、あるいはその両方に失敗します。さらに、共通テンプレートおよびテンプレートバリデーターの更新がリセットされ、失敗します。

トラブルシューティング

1. ssp-operator の Pod の namespace を確認します。

```
$ export NAMESPACE="$(oc get deployment -A | grep ssp-operator | awk '{print $1}')"
```

2. ssp-operator の Pod が現在ダウンしていることを確認します。

```
$ oc -n $NAMESPACE get pods -l control-plane=ssp-operator
```

3. ssp-operator の Pod の説明およびログを確認します。

```
$ oc -n $NAMESPACE describe pods -l control-plane=ssp-operator
```

```
$ oc -n $NAMESPACE logs --tail=-1 -l control-plane=ssp-operator
```

解決方法

サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.2.3. SSPTemplateValidatorDown アラート

説明

テンプレートバリデーターは、仮想マシン (VM) が割り当てられたテンプレートに違反していないことを検証します。

理由

すべてのテンプレートバリデーター Pod がダウンしている場合、テンプレートバリデーターは仮想マシンを割り当てられたテンプレートに対して検証するのに失敗します。

トラブルシューティング

1. ssp-operator Pod および virt-template-validator Pod の namespace を確認します。

```
$ export NAMESPACE_SSP="$(oc get deployment -A | grep ssp-operator | awk '{print $1}')"
```

```
$ export NAMESPACE="$(oc get deployment -A | grep virt-template-validator | awk '{print $1}')"
```

2. virt-template-validator の Pod が現在ダウンしていることを確認します。

```
$ oc -n $NAMESPACE get pods -l name=virt-template-validator
```

3. ssp-operator および virt-template-validator の Pod の説明およびログを確認します。

```
$ oc -n $NAMESPACE_SSP describe pods -l name=ssp-operator
```

```
$ oc -n $NAMESPACE_SSP logs --tail=-1 -l name=ssp-operator
```

```
$ oc -n $NAMESPACE describe pods -l name=virt-template-validator
```

```
$ oc -n $NAMESPACE logs --tail=-1 -l name=virt-template-validator
```

解決方法

サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.3. virt アラート

virt アラートは、OpenShift Virtualization Virt Operator の問題についての情報を提供します。

13.12.3.1. NoLeadingVirtOperator アラート

説明

過去 10 分間、1 つまたは複数の virt-operator Pod が **Ready** 状態にあるにもかかわらず、どの virt-operator Pod もリーダーリースを保持しない。アラートは動作している virt-operator Pod が存在しないことを示唆します。

理由

virt-operator は、OpenShift Container Platform クラスターでアクティブな最初の Kubernetes Operator です。その主なロールは以下のとおりです。

- インストール
- ライブ更新
- クラスターのライブアップグレード
- virt-controller、virt-handler、virt-launcher などの最上位のコントローラーのライフサイクルの監視
- 最上位のコントローラーの調整の管理

さらに、virt-operator は、証明書のローテーションや一部のインフラストラクチャー管理などのクラスター全体のタスクを行います。

virt-operator デプロイメントには、2 つの Pod のデフォルトレプリカが設定されます。1 つのリーダー Pod はリーダーリースを保持し、動作中の virt-operator Pod であることを示します。

このアラートは、クラスターレベルでの失敗を示します。証明書のローテーション、アップグレード、およびコントローラーの調整などの重要なクラスター全体の管理機能は、一時的に利用できなくなる可能性があります。

トラブルシューティング

Pod のログから virt-operator Pod のリーダーのステータスを判断します。**Started leading** および **acquire leader** が含まれるログメッセージは、その virt-operator Pod のリーダーステータスを示します。

さらに、以下のコマンドで、動作中の virt-operator Pod の有無、および Pod のステータスを常に確認します。

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns="":.metadata.namespace)"
```

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-operator
```

```
$ oc -n $NAMESPACE logs <pod-name>
```

```
$ oc -n $NAMESPACE describe pod <pod-name>
```

リーダー Pod の例:

```
$ oc -n $NAMESPACE logs <pod-name> |grep lead
```

出力例

```
{"component":"virt-operator","level":"info","msg":"Attempting to acquire leader
status","pos":"application.go:400","timestamp":"2021-11-30T12:15:18.635387Z"}
I1130 12:15:18.635452       1 leaderelection.go:243] attempting to acquire leader lease
<namespace>/virt-operator...
I1130 12:15:19.216582       1 leaderelection.go:253] successfully acquired lease <namespace>/virt-
operator
```

```
{"component":"virt-operator","level":"info","msg":"Started
leading","pos":"application.go:385","timestamp":"2021-11-30T12:15:19.216836Z"}
```

リーダー以外の Pod の例:

```
$ oc -n $NAMESPACE logs <pod-name> |grep lead
```

出力例

```
{"component":"virt-operator","level":"info","msg":"Attempting to acquire leader
status","pos":"application.go:400","timestamp":"2021-11-30T12:15:20.533696Z"}
I1130 12:15:20.533792       1 leaderelection.go:243] attempting to acquire leader lease
<namespace>/virt-operator...
```

解決方法

さまざまな理由により、1 つまたは複数の virt-operator Pod が **Ready** 状態にあるにもかかわらず、どの virt-operator Pod もリーダーリースを保持しない状況になります。根本原因を特定し、適切なアクションを実行します。

それ以外には、サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.3.2. NoReadyVirtController アラート

説明

virt-controller は、仮想マシンインスタンス (VMI) を監視します。virt-controller は、VMI オブジェクトに関連付けられた Pod のライフサイクルを作成し、管理して、関連付けられた Pod の管理も行います。

VMI オブジェクトは、有効期間中に常に Pod に関連付けられます。ただし、Pod インスタンスは VMI の移行により時間の経過とともに変更される可能性があります。

このアラートは、準備ができている virt-controller が 5 分間検出されなかった場合に発生します。

理由

virt-controller が失敗すると、仮想マシンのライフサイクル管理は完全に失敗します。ライフサイクルの管理タスクには、新規 VMI の起動や既存の VMI のシャットダウンなどが含まれます。

トラブルシューティング

1. virt-controller のデプロイメントステータスで、利用可能なレプリカおよび条件を確認します。

   ```
   $ oc -n $NAMESPACE get deployment virt-controller -o yaml
   ```

2. virt-controller Pod が存在するかどうかを確認し、それらのステータスを確認します。

   ```
   get pods -n $NAMESPACE |grep virt-controller
   ```

3. virt-controller Pod のイベントを確認します。

   ```
   $ oc -n $NAMESPACE describe pods <virt-controller pod>
   ```

4. virt-controller Pod のログを確認します。

```
$ oc -n $NAMESPACE logs <virt-controller pod>
```

5. ノードが **NotReady** 状態にあるなど、ノードに問題があるかどうかを確認します。

```
$ oc get nodes
```

解決方法

Ready 状態にある virt-controller Pod が存在しない理由はいくつかあります。根本原因を特定し、適切なアクションを実行します。

それ以外には、サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.3.3. NoReadyVirtOperator アラート

説明

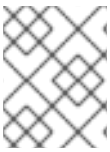
過去 10 分間、**Ready** 状態の virt-operator Pod が検出されない。virt-operator デプロイメントには、2 つの Pod のデフォルトレプリカが設定されます。

理由

virt-operator は、OpenShift Container Platform クラスターでアクティブな最初の Kubernetes Operator です。その主なロールは以下のとおりです。

- インストール
- ライブ更新
- クラスターのライブアップグレード
- virt-controller、virt-handler、virt-launcher などの最上位のコントローラーのライフサイクルの監視
- 最上位のコントローラーの調整の管理

さらに、virt-operator は、証明書のローテーションや一部のインフラストラクチャー管理などのクラスター全体のタスクを行います。



注記

virt-operator には、クラスター内の仮想マシンに対する直接のロールりはありません。virt-operator が使用できないことは、カスタムワークロードには影響しません。

このアラートは、クラスターレベルでの失敗を示します。証明書のローテーション、アップグレード、およびコントローラーの調整などの重要なクラスター全体の管理機能は、一時的に利用できなくなります。

トラブルシューティング

1. virt-operator のデプロイメントステータスで、利用可能なレプリカおよび条件を確認します。

```
$ oc -n $NAMESPACE get deployment virt-operator -o yaml
```

2. virt-controller Pod のイベントを確認します。

```
$ oc -n $NAMESPACE describe pods <virt-operator pod>
```

3. virt-operator Pod のログを確認します。

```
$ oc -n $NAMESPACE logs <virt-operator pod>
```

4. **NotReady** 状態にあるなど、コントロールプレーンおよびマスターのノードに問題があるかどうかを確認します。

```
$ oc get nodes
```

解決方法

Ready 状態にある virt-operator Pod が存在しない理由はいくつかあります。根本原因を特定し、適切なアクションを実行します。

それ以外には、サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.3.4. VirtAPIDown アラート

説明

すべての OpenShift Container Platform API サーバーが停止している。

理由

すべての OpenShift Container Platform API サーバーがダウンしている場合、OpenShift Container Platform エンティティーの API 呼び出しは行われません。

トラブルシューティング

1. 環境変数 **NAMESPACE** を変更します。

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns="":.metadata.namespace)"
```

2. 動作中の virt-api Pod があるかどうかを確認します。

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-api
```

3. **oc logs**を使用して Pod のログを表示し、**oc describe** を使用して Pod のステータスを表示します。

4. virt-api デプロイメントのステータスを確認します。以下のコマンドを使用して関連するイベントについて確認し、イメージのプルに関する問題、クラッシュしている Pod、またはその他の同様の問題の有無を表示します。

```
$ oc -n $NAMESPACE get deployment virt-api -o yaml
```

```
$ oc -n $NAMESPACE describe deployment virt-api
```

5. ノードが **NotReady** 状態にあるなど、ノードに問題があるかどうかを確認します。

```
$ oc get nodes
```

解決方法

virt-api Pod は、いくつかの理由でダウンする可能性があります。根本原因を特定し、適切なアクションを実行します。

それ以外には、サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.3.5. VirtApiRESTErrorsBurst アラート

説明

過去 5 分間、virt-api で 80% を超える REST 呼び出しが失敗する。

理由

virt-api への REST 呼び出しの失敗率が非常に高くなると、応答が遅くなるか、API 呼び出しの実行が遅くなるか、API 呼び出しがすべて破棄されます。

トラブルシューティング

1. 環境変数 **NAMESPACE** を変更します。

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns="":.metadata.namespace)"
```

2. 動作中の virt-api Pod がいくつあるかを確認します。

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-api
```

3. **oc logs**を使用して Pod のログを表示し、**oc describe** を使用して Pod のステータスを表示します。

4. virt-api デプロイメントのステータスをチェックして、詳細情報を確認します。以下のコマンドは関連するイベントを提供し、イメージのプルに関する問題またはクラッシュしている Pod があるかどうかを表示します。

```
$ oc -n $NAMESPACE get deployment virt-api -o yaml
```

```
$ oc -n $NAMESPACE describe deployment virt-api
```

5. ノードがオーバーロード状態にある、または **NotReady** 状態にあるなど、ノードに問題があるかどうかを確認します。

```
$ oc get nodes
```

解決方法

REST 呼び出しの失敗率が高い理由はいくつかあります。根本原因を特定し、適切なアクションを実行します。

- ノードリソースの枯渇
- クラスターに十分なメモリーがない
- ノードがダウンしている
- スケジューラーが 100% 利用できない場合など、API サーバーのオーバーロード
- ネットワークの問題

それ以外には、サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.3.6. VirtControllerDown アラート

説明

過去 5 分間 virt-controller が検出されない場合、virt-controller デプロイメントには 2 つの Pod のデフォルトレプリカが設定されます。

理由

virt-controller が失敗すると、新規 VMI の起動や既存 VMI のシャットダウンなどの仮想マシンのライフサイクル管理タスクが完全に失敗します。

トラブルシューティング

1. 環境変数 **NAMESPACE** を変更します。

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns="":.metadata.namespace)"
```

2. virt-controller デプロイメントのステータスを確認します。

```
$ oc get deployment -n $NAMESPACE virt-controller -o yaml
```

3. virt-controller Pod のイベントを確認します。

```
$ oc -n $NAMESPACE describe pods <virt-controller pod>
```

4. virt-controller Pod のログを確認します。

```
$ oc -n $NAMESPACE logs <virt-controller pod>
```

5. マネージャー Pod のログを確認して、virt-controller Pod の作成に失敗した理由を判断します。

```
$ oc get logs <virt-controller-pod>
```

ログの virt-controller Pod 名の例は **virt-controller-7888c64d66-dzc9p** です。ただし、virt-controller を実行する Pod が複数存在する場合があります。

解決方法

動作中の virt-controller が検出されない既知の理由がいくつかあります。考えられる理由の一覧から根本原因を特定し、適切なアクションを実行します。

- ノードリソースの枯渇
- クラスターに十分なメモリーがない
- ノードがダウンしている
- スケジューラーが 100% 利用できない場合など、API サーバーのオーバーロード
- ネットワークの問題

それ以外には、サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.3.7. VirtControllerRESTErrorsBurst アラート

説明

過去 5 分間、virt-controller で 80% を超える REST 呼び出しが失敗する。

理由

virt-controller が、API サーバーへの接続を完全に失った可能性があります。接続の喪失は実行中のワークロードには影響しませんが、ステータス更新の反映や移行などのアクションは実行できません。

トラブルシューティング

virt-controller REST 呼び出しの失敗には、以下の 2 つの一般的なエラータイプがあります。

- API サーバーのオーバーロードにより、タイムアウトを引き起こす。応答時間や全体の呼び出しなど、API サーバーメトリクスと詳細を確認します。
- virt-controller Pod が API サーバーに到達できない。一般的な原因は以下のとおりです。
  - ノード上の DNS の問題
  - ネットワーク接続の問題

解決方法

virt-controller ログをチェックして、virt-controller Pod が API サーバーにまったく接続できないかどうかを判断します。その場合、Pod を削除して強制的に再起動します。

さらに、ノードリソースが使い切られるか、クラスターに十分なメモリーがないため、接続に失敗しているかどうかを確認します。

通常、問題は、このアラートの範囲外の DNS または CNI の問題に関連しています。

それ以外には、サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.3.8. VirtHandlerRESTErrorsBurst アラート

説明

過去 5 分間、virt-handler で 80% を超える REST 呼び出しが失敗する。

理由

virt-handler が API サーバーへの接続を失った。影響を受けるノードで実行中のワークロードは実行し続けますが、ステータスの更新を反映できず、移行などのアクションを実行できません。

トラブルシューティング

virt-operator REST 呼び出しの失敗には、以下の 2 つの一般的なエラータイプがあります。

- API サーバーのオーバーロードにより、タイムアウトを引き起こす。応答時間や全体の呼び出しなど、API サーバーメトリクスと詳細を確認します。
- virt-operator Pod が API サーバーに到達できない。一般的な原因は以下のとおりです。
  - ノード上の DNS の問題
  - ネットワーク接続の問題

解決方法

virt-handler が API サーバーに接続できない場合、Pod を削除して強制的に再起動します。通常、問題は、このアラートの範囲外の DNS または CNI の問題に関連しています。根本原因を特定し、適切なアクションを実行します。

それ以外には、サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.3.9. VirtOperatorDown アラート

説明

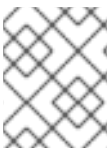
このアラートは、過去 10 分間 **Running** 状態の virt-operator Pod が存在しない場合に発生します。virt-operator デプロイメントには、2 つの Pod のデフォルトレプリカが設定されます。

理由

virt-operator は、OpenShift Container Platform クラスターでアクティブな最初の Kubernetes Operator です。その主なロールは以下のとおりです。

- インストール
- ライブ更新
- クラスターのライブアップグレード
- virt-controller、virt-handler、virt-launcher などの最上位のコントローラーのライフサイクルの監視
- 最上位のコントローラーの調整の管理

さらに、virt-operator は、証明書のローテーションや一部のインフラストラクチャー管理などのクラスター全体のタスクを行います。



注記

virt-operator には、クラスター内の仮想マシンに対する直接のロールりはありません。virt-operator が使用できないことは、カスタムワークロードには影響しません。

このアラートは、クラスターレベルでの失敗を示します。証明書のローテーション、アップグレード、およびコントローラーの調整などの重要なクラスター全体の管理機能は、一時的に利用できなくなります。

トラブルシューティング

1. 環境変数 **NAMESPACE** を変更します。

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns="":.metadata.namespace)"
```

2. virt-operator デプロイメントのステータスを確認します。

```
$ oc get deployment -n $NAMESPACE virt-operator -o yaml
```

3. virt-operator Pod のイベントを確認します。

```
$ oc -n $NAMESPACE describe pods <virt-operator pod>
```

4. virt-operator Pod のログを確認します。

```
$ oc -n $NAMESPACE logs <virt-operator pod>
```

5. マネージャー Pod のログを確認して、virt-operator Pod の作成に失敗した理由を判断します。

```
$ oc get logs <virt-operator-pod>
```

ログの virt-operator Pod 名の例は **virt-operator-7888c64d66-dzc9p** です。ただし、virt-operator を実行する Pod が複数存在する場合があります。

解決方法

動作中の virt-operator が検出されない既知の理由がいくつかあります。考えられる理由の一覧から根本原因を特定し、適切なアクションを実行します。

- ノードリソースの枯渇
- クラスターに十分なメモリーがない
- ノードがダウンしている
- スケジューラーが 100% 利用できない場合など、API サーバーのオーバーロード
- ネットワークの問題

それ以外には、サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.3.10. VirtOperatorRESTErrorsBurst アラート

説明

過去 5 分間、virt-operator で 80% を超える REST 呼び出しが失敗する。

理由

virt-operator が API サーバーへの接続を失った。アップグレードおよびコントローラーの調整などのクラスターレベルのアクションは機能しません。仮想マシンや VMI などのお客様のワークロードには影響がありません。

トラブルシューティング

virt-operator REST 呼び出しの失敗には、以下の 2 つの一般的なエラータイプがあります。

- API サーバーのオーバーロードにより、タイムアウトを引き起こす。応答時間や全体の呼び出しなど、API サーバーメトリクスと詳細を確認します。
- virt-operator Pod が API サーバーに到達できない。一般的な原因は、ネットワークの接続性の問題や、ノードでの DNS の問題です。virt-operator ログをチェックして、Pod が API サーバーに接続できることを確認します。

```
$ export NAMESPACE="$(oc get kubevirt -A -o custom-columns="":.metadata.namespace)"
```

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-operator
```

```
$ oc -n $NAMESPACE logs <pod-name>
```

```
$ oc -n $NAMESPACE describe pod <pod-name>
```

解決方法

virt-operator が API サーバーに接続できない場合、Pod を削除して強制的に再起動します。通常、問題は、このアラートの範囲外の DNS または CNI の問題に関連しています。根本原因を特定し、適切なアクションを実行します。

それ以外には、サポートケースを作成し、トラブルシューティングのプロセスで収集された情報を提供します。

13.12.4. 関連情報

- サポート

13.13. 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、以下のツールを使用して OpenShift Container Platform および 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、メトリック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。

13.13.1. 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 メトリックデータの保持期間を最低 7 日間に設定します。
- Alertmanager を設定して、関連するアラートを取得して、それらを専用のメールボックスに送信して、クラスター外で表示および保持できるようにします。
- 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します。

手順

1. デフォルトの **must-gather** イメージを使用して、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 **must-gather** イメージを使用して、OpenShift Virtualization の **must-gather** データを収集します。
4. クラスターの Prometheus メトリックを収集します。

13.13.1.1. 関連情報

- Prometheus メトリクスデータの 保持期間 の設定
- アラート通知 を外部システムに送信するための Alertmanager の設定
- OpenShift Container Platform の **must-gather** データの収集
- Red Hat OpenShift Data Foundation の **must-gather** データの収集
- OpenShift Virtualization の **must-gather** データの収集
- クラスター管理者として すべてのプロジェクト の Prometheus メトリクスの収集

13.13.2. 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

- Windows 仮想マシン:
  - Red Hat サポート用に Windows パッチ更新の詳細を記録します。
  - VirtIO ドライバーの最新バージョンをインストールします。VirtIO ドライバーには、QEMU ゲストエージェントが含まれています。
  - 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、RDP を使用して仮想マシンに接続し、接続ソフトウェアに問題があるかどうかを判断します。

手順

1. 誤動作している仮想マシンに関する詳細な **must-gather** を収集します。
2. 仮想マシンを再起動する前に、クラッシュした仮想マシンのスクリーンショットを収集します。
3. 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。

13.13.2.1. 関連情報

- Windows VM への VirtIO ドライバー のインストール
- ホストアクセスなしで Windows VM に VirtIO ドライバー をダウンロードしてインストール
- Web コンソール または コマンドライン を使用して RDP で Windows 仮想マシンに接続する
- 仮想マシン に関する **must-gather** データの収集

13.13.3. 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 (CRD)
- 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-stream=openshift/must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.10.10
```

13.13.3.1. 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
- イメージおよびイメージストリーム情報の収集
- **must-gather** ツールが使用する並列プロセスの最大数の制限

13.13.3.1.1. 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。

gather_vms_details

OpenShift Virtualization リソースに属する仮想マシンログファイル、仮想マシン定義、ならびに namespace (およびそれらのサブオブジェクト) を収集します。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。

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。

gather_images

イメージおよびイメージストリームの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。

13.13.3.1.2. 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13.1 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

スクリプト	互換性のある環境変数
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>**
gather	• **PROS=<number_of_processes>**
gather_images	• **PROS=<number_of_processes>**

must-gather が収集するデータをカスタマイズするには、コマンドに二重ダッシュ (**--**) を追加し、その後にスペースと 1 つ以上の互換性のあるパラメーターを追加します。

構文

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.10.10 \
  -- <environment_variable_1> <environment_variable_2> <script_name>
```

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-rhel8:v4.10.10 \
  -- NS=mynamespace VM=my-vm 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。

3 つの並列プロセスに限定されたデフォルトのデータ収集

以下のコマンドは、最大 3 つの並列処理を使用して、デフォルトの **must-gather** 情報を収集します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.10.10 \
  -- PROS=3 gather
```

イメージおよびイメージストリーム情報

以下のコマンドは、クラスターからイメージおよびイメージストリームの情報を収集します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8:v4.10.10 \
  -- gather_images
```

13.13.3.2. 関連情報

- **must-gather** ツールについて

第14章 バックアップと復元

14.1. 仮想マシンのバックアップと復元

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。

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** 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、**Backup** CR を復元します。

14.1.1. 関連情報

- OADP features and plugins
- トラブルシューティング