OpenShift Container Platform 4.16

Virtualization

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

Last Updated: 2024-09-10

OpenShift Container Platform 4.16 Virtualization

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

法律上の通知

概要

このドキュメントでは、OpenShift Container Platform で OpenShift Virtualization を使用する方法に関する情報を提供します。

目次

第1章 概要

1.1. 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。

1.1.1. OpenShift Virtualization の機能

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

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

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

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

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

重要

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

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

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

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

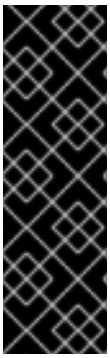
OpenShift Virtualization 4.16 は、OpenShift Container Platform 4.16 クラスターでの使用がサポートされます。OpenShift Virtualization の最新の z-stream リリースを使用するには、最初に OpenShift Container Platform の最新バージョンにアップグレードする必要があります。

1.1.2. 仮想マシンディスクのボリュームとアクセスモードについて

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

最良の結果を得るには、**ReadWriteMany** (RWX) アクセスモードと **Block** ボリュームモードを使用してください。これは、以下の理由により重要です。

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



重要

次の設定の仮想マシンをライブマイグレーションすることはできません。

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

これらの仮想マシンの **evictionStrategy** フィールドを **None** に設定します。**None** ストラテジーでは、ノードの再起動中に仮想マシンの電源がオフになります。

1.1.3. シングルノード OpenShift の違い

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

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

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

1.1.4. 関連情報

- OpenShift Container Platform ストレージの共通用語集
- 単一ノード OpenShift について
- Assisted installer
- Pod の Disruption Budget (停止状態の予算)
- ライブマイグレーションについて
- エビクションストラテジー

- Tuning & Scaling Guide
- Supported limits for OpenShift Virtualization 4.x

1.2. セキュリティーポリシー

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

主なポイント

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

1.2.1. ワークロードのセキュリティーについて

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

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

1.2.2. TLS 証明書

OpenShift Virtualization コンポーネントの TLS 証明書は更新され、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。

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

1.2.3. 認可

OpenShift Virtualization は、ロールベースのアクセス制御 (RBAC) を使用して、人間のユーザーとサービスアカウントの権限を定義します。サービスアカウントに定義された権限は、OpenShift Virtualization コンポーネントが実行できるアクションを制御します。

RBAC ロールを使用して、仮想化機能へのユーザーアクセスを管理することもできます。たとえば管理者は、仮想マシンの起動に必要な権限を提供する RBAC ロールを作成できます。管理者は、ロールを特定のユーザーにバインドすることでアクセスを制限できます。

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

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

表1.1 OpenShift Virtualization のクラスターロール

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

1.2.3.2. OpenShift Virtualization のストレージ機能の RBAC ロール

cdi-operator および **cdi-controller** サービスアカウントを含む、次のパーミッションがコンテナー化データインポーター (CDI) に付与されます。

1.2.3.2.1. クラスター全体の RBAC のロール

表1.2 cdi.kubevirt.io API グループの集約されたクラスターロール

CDI クラスターのロール	リソース	動詞
cdi.kubevirt.io:admin	**datavolumes**、**uploadtokenrequests**	* (すべて)

CDI クラスターのロール	リソース	動詞
	datavolumes/source	create
cdi.kubevirt.io:edit	datavolumes、uploadtokenrequests	*
	datavolumes/source	create
cdi.kubevirt.io:view	cdiconfigs、dataimportcrons、datasources、datavolumes、objecttransfers、storageprofiles、volumeimportsources、volumeuploadsources、volumeclonesources	get, list, watch
	datavolumes/source	create
cdi.kubevirt.io:config-reader	cdiconfigs、storageprofiles	get, list, watch

表1.3 cdi-operator サービスアカウントのクラスター全体のロール

API グループ	リソース	動詞
rbac.authorization.k8s.io	clusterrolebindings、clusterroles	get、list、watch、create、update、delete
security.openshift.io	securitycontextconstraints	get、list、watch、update、create
apiextensions.k8s.io	customresourcedefinitions、customresourcedefinitions/status	get、list、watch、create、update、delete
cdi.kubevirt.io	*	*
upload.cdi.kubevirt.io	*	*
admissionregistration.k8s.io	validatingwebhookconfigurations、mutatingwebhookconfigurations	create、list、watch

API グループ	リソース	動詞
admissionregistration.k8s.io	**validatingwebhookconfigurations** 許可リスト: **cdi-api-dataimportcron-validate, cdi-api-populator-validate, cdi-api-datavolume-validate, cdi-api-validate, objecttransfer-api-validate**	**get、update、delete**
admissionregistration.k8s.io	**mutatingwebhookconfigurations** 許可リスト: **cdi-api-datavolume-mutate**	**get、update、delete**
apiregistration.k8s.io	**apiservices**	**get、list、watch、create、update、delete**

表1.4 cdi-controller サービスアカウントのクラスター全体のロール

API グループ	リソース	動詞
"" (core)	**events**	**create、patch**
"" (core)	**persistentvolumeclaims**	**get、list、watch、create、update、delete、deletecollection、patch**
"" (core)	**persistentvolumes**	**get、list、watch、update**
"" (core)	**persistentvolumeclaims/finalizers、pods/finalizers**	**update**
"" (core)	**pods、services**	**get、list、watch、create、delete**
"" (core)	**configmaps**	**get、create**
storage.k8s.io	**storageclasses、csidrivers**	**get, list, watch**
config.openshift.io	**proxies**	**get, list, watch**
cdi.kubevirt.io	*	*

API グループ	リソース	動詞
snapshot.storage.k8s.io	**volumesnapshots、volumesnapshotclasses、volumesnapshotcontents**	**get、list、watch、create、delete**
snapshot.storage.k8s.io	**volumesnapshots**	**update、deletecollection**
apiextensions.k8s.io	**customresourcedefinitions**	**get, list, watch**
scheduling.k8s.io	**priorityclasses**	**get, list, watch**
image.openshift.io	**imagestreams**	**get, list, watch**
"" (core)	**secrets**	**create**
kubevirt.io	**virtualmachines/finalizers**	**update**

1.2.3.2.2. namespace 付きの RBAC ロール

表1.5 **cdi-operator** サービスアカウントの namespace ロール

API グループ	リソース	動詞
rbac.authorization.k8s.io	**rolebindings、roles**	**get、list、watch、create、update、delete**
"" (core)	**serviceaccounts、configmaps、events、secrets、services**	**get, list, watch, create, update, patch, delete**
apps	**deployments、deployments/finalizers**	**get、list、watch、create、update、delete**
route.openshift.io	**routes、routes/custom-host**	**get、list、watch、create、update**
config.openshift.io	**proxies**	**get, list, watch**
monitoring.coreos.com	**servicemonitors、prometheusrules**	**get、list、watch、create、delete、update、patch**
coordination.k8s.io	**leases**	**get、create、update**

表1.6 cdi-controller サービスアカウントの namespace ロール

API グループ	リソース	動詞
"" (core)	**configmaps**	**get、list、watch、create、update、delete**
"" (core)	**secrets**	**get**, **list**, **watch**
batch	**cronjobs**	**get、list、watch、create、update、delete**
batch	**jobs**	**create、delete、list、watch**
coordination.k8s.io	**leases**	**get、create、update**
networking.k8s.io	**ingresses**	**get**, **list**, **watch**
route.openshift.io	**routes**	**get**, **list**, **watch**

1.2.3.3. kubevirt-controller サービスアカウントの追加の SCC とパーミッション

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

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

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

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

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

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
```

1.2.4. 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
- クラスターロールの作成
- クラスターのロールバインディングコマンド
- 複数の namespace 間でデータボリュームをクローン作成するためのユーザーパーミッションの有効化

1.3. OPENSHIFT VIRTUALIZATION アーキテクチャー

Operator Lifecycle Manager (OLM) は、OpenShift Virtualization の各コンポーネントのオペレーター Pod をデプロイします。

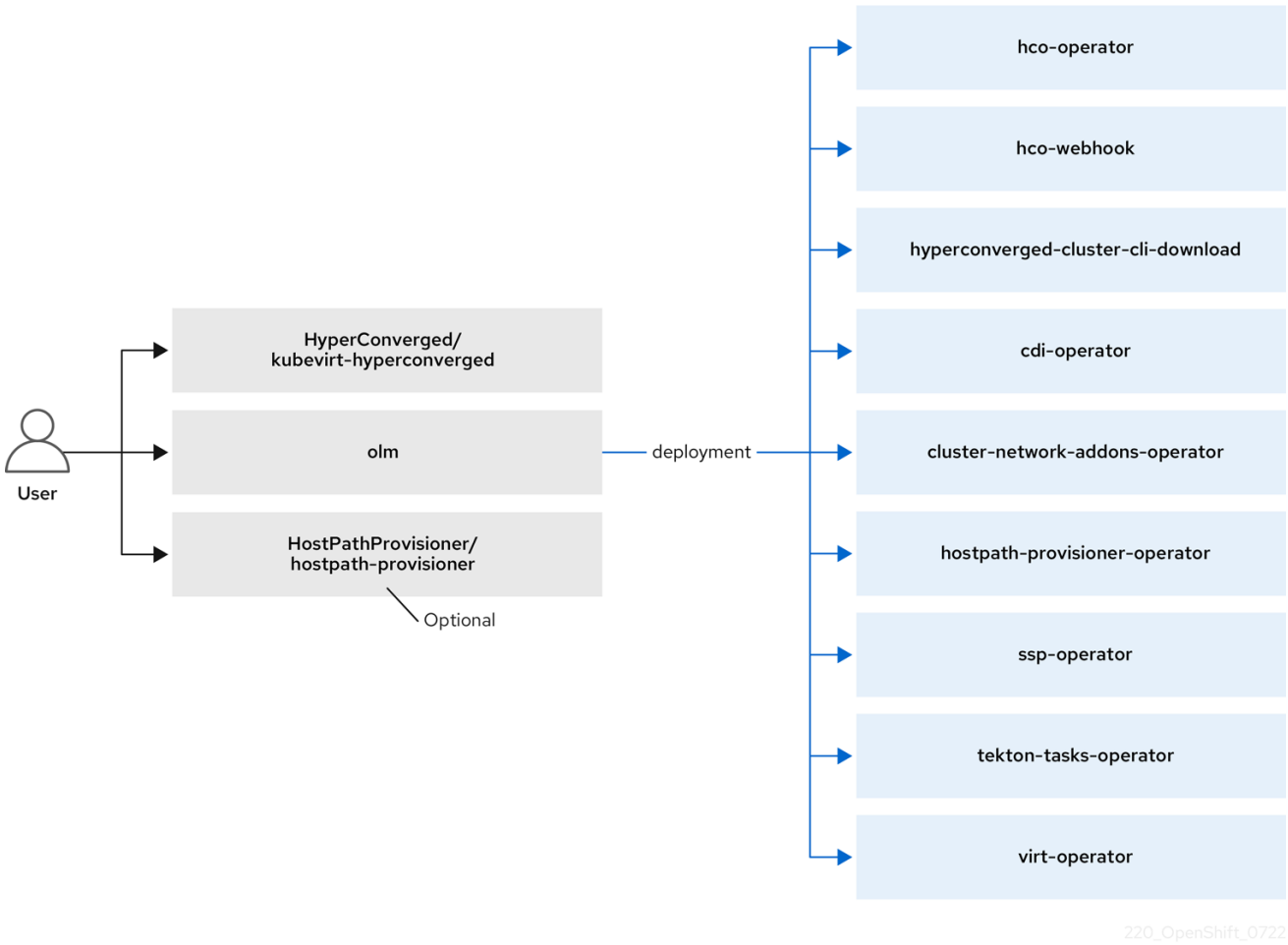
- コンピューティング: **virt-operator**
- ストレージ: **cdi-operator**
- ネットワーク: **cluster-network-addons-operator**
- スケーリング: **ssp-operator**

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

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

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

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



- Virtctl クライアントコマンド

1.3.1. HyperConverged Operator (HCO) について

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

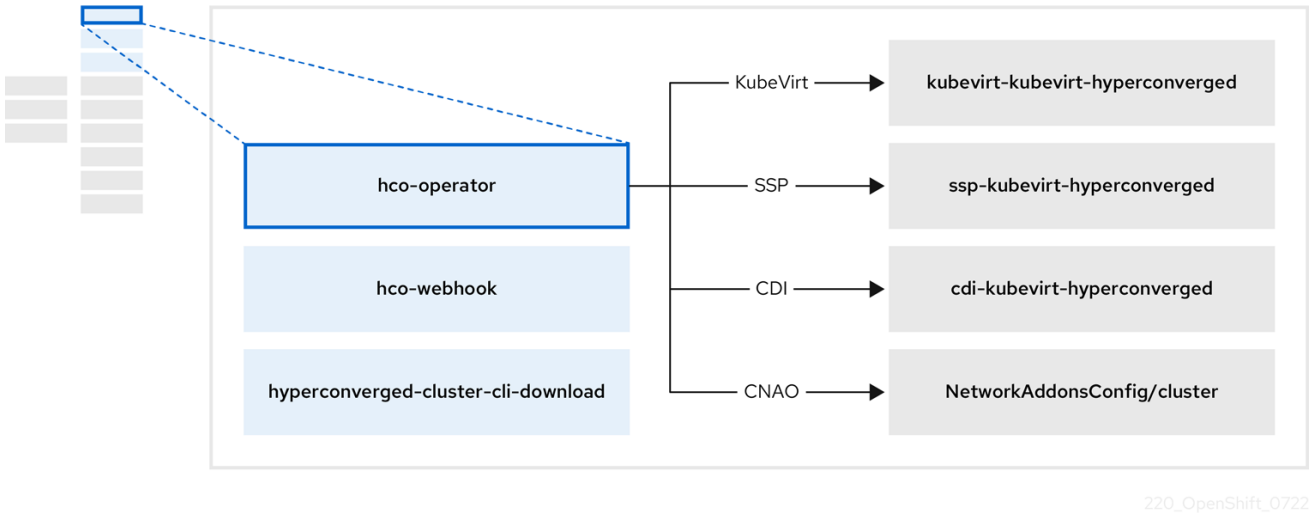


表1.7 HyperConverged Operator コンポーネント

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

1.3.2. Containerized Data Importer (CDI) Operator について

CDI Operator **cdi-operator** は、CDI とその関連リソースを管理し、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。

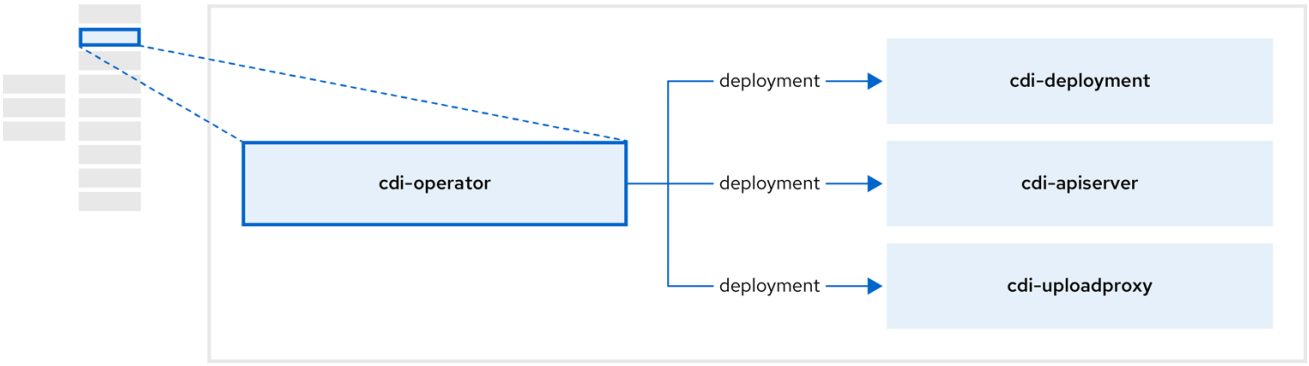


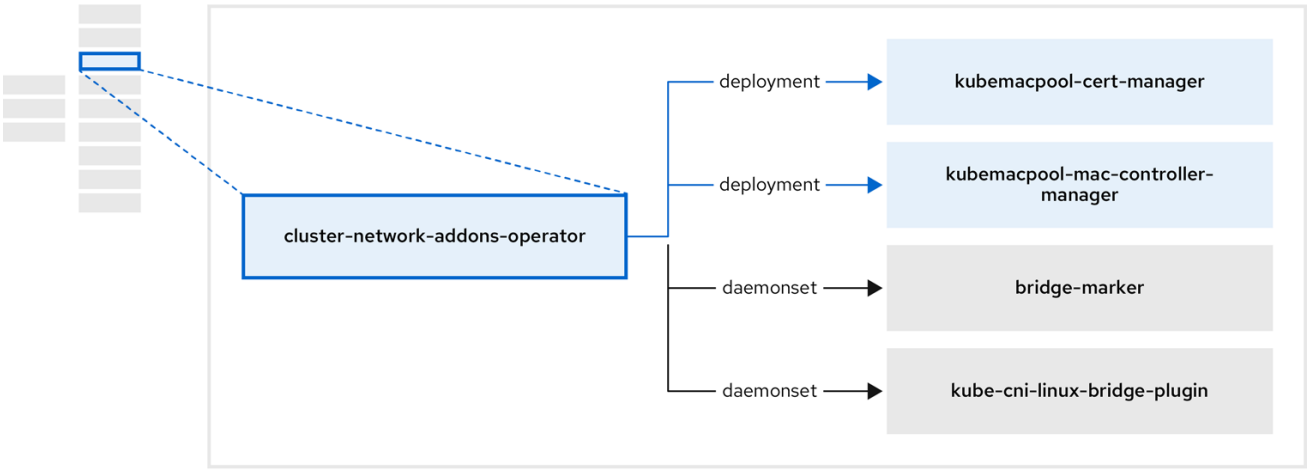
表1.8 CDI Operator コンポーネント

コンポーネント	説明
deployment/cdi-apiserver	安全なアップロードトークンを発行して、VM ディスクを PVC にアップロードするための承認を管理します。

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

1.3.3. Cluster Network Addons Operator について

Cluster Network Addons Operator (**cluster-network-addons-operator**) は、クラスターにネットワークコンポーネントをデプロイし、拡張ネットワーク機能の関連リソースを管理します。



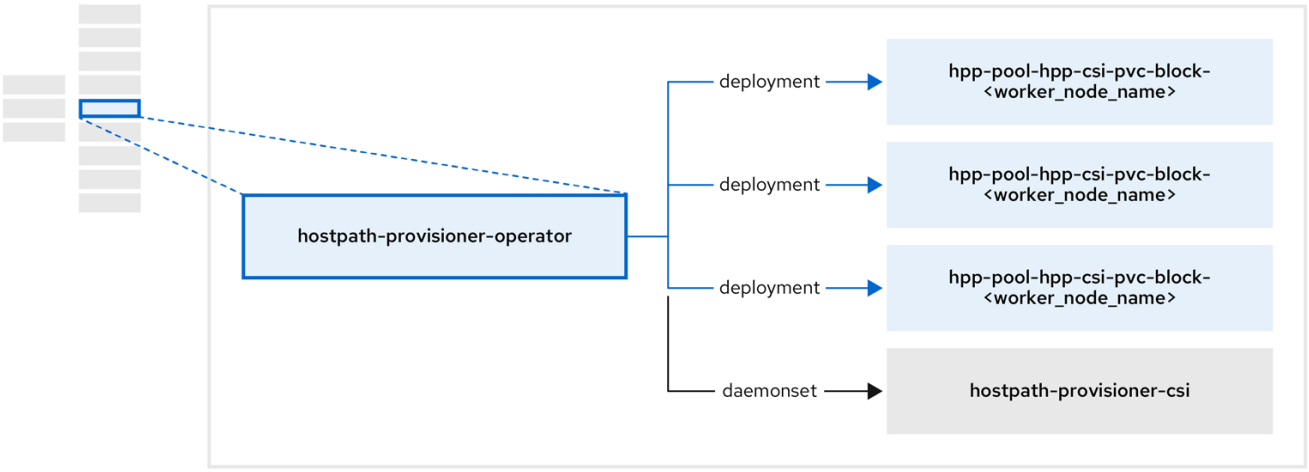
220_OpenShift_0722

表1.9 Cluster Network Addons Operator コンポーネント

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

1.3.4. Hostpath Provisioner (HPP) Operator について

HPP オペレーター **hostpath-provisioner-operator** は、マルチノード HPP および関連リソースをデプロイおよび管理します。



220_OpenShift_0622

表1.10 HPP Operator コンポーネント

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

1.3.5. Scheduling, Scale, and Performance (SSP) Operator について

SSP オペレーター **ssp-operator** は、共通テンプレート、関連するデフォルトのブートソース、パイプラインタスク、およびテンプレートバリデーターをデプロイします。

1.3.6. OpenShift Virtualization Operator について

OpenShift Virtualization Operator **virt-operator** は、現在の仮想マシンワークロードを中断することなく OpenShift Virtualization をデプロイ、アップグレード、管理します。さらに、OpenShift Virtualization Operator は、共通のインスタンスタイプと共通の設定をデプロイします。

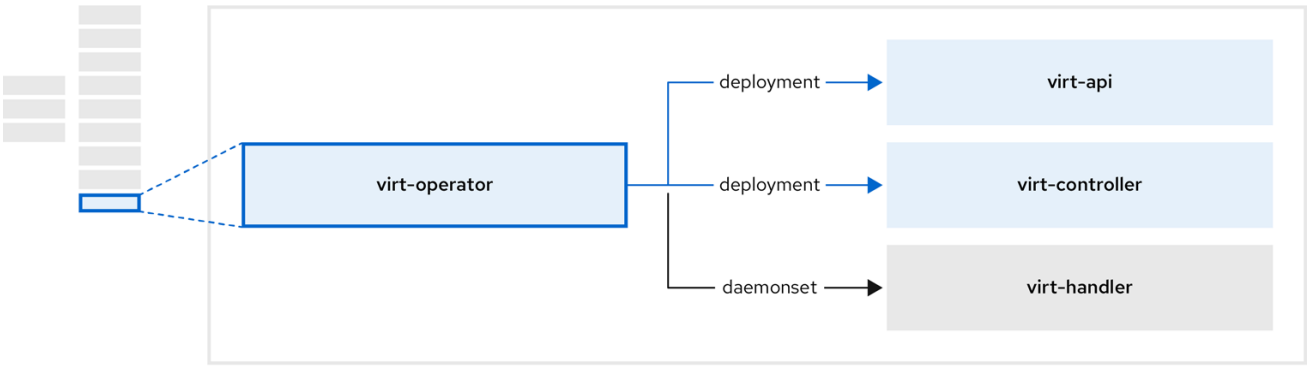


表1.11 virt-operator コンポーネント

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

第2章 リリースノート

2.1. OPENSHIFT VIRTUALIZATION リリースノート

2.1.1. ドキュメントに関するフィードバックの提供

エラーを報告したり、ドキュメントを改善したりするには、Red Hat Jira アカウント にログインし、Jira issue を送信してください。

2.1.2. Red Hat OpenShift Virtualization について

Red Hat OpenShift Virtualization を使用すると、従来の仮想マシン (VM) を OpenShift Container Platform に導入し、コンテナーと一緒に実行できます。OpenShift Virtualization では、仮想マシンとは OpenShift Container Platform Web コンソールまたはコマンドラインを使用して管理できるネイティブ Kubernetes オブジェクトです。

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

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

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

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

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

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

OpenShift Virtualization 4.16 は、OpenShift Container Platform 4.16 クラスターでの使用がサポートされます。OpenShift Virtualization の最新の z-stream リリースを使用するには、最初に OpenShift 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。

2.1.2.2. サポート対象のゲストオペレーティングシステム

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

2.1.2.3. Microsoft Windows SVVP 認定

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 認定は以下に適用されます。

- Red Hat Enterprise Linux CoreOS ワーカー。Microsoft SVVP Catalog では、**Red Hat OpenShift Container Platform 4 on RHEL CoreOS 9** という名前が付けられます。
- Intel および AMD CPU。

2.1.3. 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 Container Platform Web 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** フィールドにキーワードとして **virtualization** を入力すると、利用可能なツアーをフィルタリングできます。

2.1.4. 新機能および変更された機能

このリリースでは、次のコンポーネントと概念に関連する新機能と機能拡張が追加されています。

2.1.4.1. インストールおよび更新

- OpenShift Virtualization 4.16 にアップグレードすると、ガベージコレクションにより削除されていたデータボリュームが再作成される可能性があります。この動作は想定されています。データボリュームのガベージコレクションは無効になったため、再作成されたデータボリュームは無視できます。

2.1.4.2. Virtualization

- Windows 10 仮想マシンは、TPM を使用した UEFI を使用して起動するようになりました。
- **AutoResourceLimits** フィーチャーゲートを有効にすると、仮想マシンの CPU とメモリーの制限が自動的に管理されます。
- KubeVirt Tekton タスクは、OpenShift Container Platform Pipelines catalog の一部として出荷されるようになりました。

2.1.4.3. ネットワーク

- ヘッドレスサービスを使用して、安定した完全修飾ドメイン名 (FQDN) 上のデフォルトの内部 Pod ネットワークに接続されている 仮想マシンにアクセスできる ようになりました。

2.1.4.4. Web コンソール

- 仮想マシンのホットプラグが一般利用可能になりました。仮想マシンをホットプラグできない場合、**RestartRequired** の条件が VM に適用されます。この状態は、Web コンソールの **Diagnostics** タブで確認できます。
- インスタンスタイプから Microsoft Windows 仮想マシンを作成するときに、**sysprep** オプションを選択できるようになりました。以前は、仮想マシンを作成した後にカスタマイズして **sysprep** オプションを設定する必要がありました。

2.1.4.5. モニタリング

- 管理者は、**downwardMetrics** フィーチャーゲートを有効にし、**downwardMetrics** デバイスを設定することで、OpenShift Virtualization の virtio-serial ポートを介してホストおよび仮想マシン (VM) メトリクスの限定セットをゲスト仮想マシンに公開 できるようになりました。ユーザーは、**vm-dump-metrics** ツールまたはコマンドラインからメトリクスを取得します。Red Hat Enterprise Linux (RHEL) 9 では、コマンドラインを使用して 下向きメトリクスを表示します。**vm-dump-metrics** ツールは、Red Hat Enterprise Linux (RHEL) 9 プラットフォームではサポートされていません。

2.1.4.6. 主な技術上の変更点

- 仮想マシンでは、メモリーホットプラグを有効にするために、少なくとも 1 GiB のメモリー割り当てが必要です。仮想マシンに割り当てられたメモリーが 1 GiB 未満の場合、メモリーホットプラグは無効になります。

- OpenShift Virtualization アラートの runbook は、**openshift/runbooks** git repository 内でのみ管理されるようになりました。削除された runbook の代わりに、runbook ソースファイルへのリンク が利用できるようになりました。

2.1.5. 非推奨の機能と削除された機能

2.1.5.1. 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、引き続きサポートされています。ただし、非推奨の機能は今後のリリースで削除されるため、新しいデプロイメントには推奨されません。

- **tekton-tasks-operator** は非推奨になり、Tekton タスクとサンプルパイプラインは **ssp-operator** によってデプロイされるようになりました。

- **copy-template**、**modify-vm-template**、および **create-vm-from-template** タスクは非推奨になりました。

- Windows Server 2012 R2 テンプレートのサポートは廃止されました。

- **KubeVirtComponentExceedsRequestedMemory** アラートと **KubeVirtComponentExceedsRequestedCPU** アラートは非推奨になりました。安全に サイレント にできます。

2.1.5.2. 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。

2.1.6. テクノロジープレビュー機能

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

テクノロジープレビュー機能のサポート範囲

- クラスター全体 の 仮想マシンエビクションストラテジー を設定できるようになりました。

- OpenShift Virtualization ホストでネストされた仮想化 を有効化できるようになりました。

- クラスター管理者は、OpenShift Container Platform Web コンソールの **Overview → Settings → Preview features** で、namespace の CPU リソース制限を有効にできるようになりました。

- クラスター管理者は **wasp-agent** ツールを使用して、RAM 内のメモリー量をオーバーコミットし、スワップリソースを仮想マシンワークロードに割り当てることで、クラスター内の 仮想マシンワークロード密度を高く設定できる ようになりました。

- OpenShift Virtualization は、Red Hat OpenShift Data Foundation (ODF) Regional Disaster Recovery との互換性をサポートするようになりました。

2.1.7. 既知の問題

モニタリング

- Pod Disruption Budget (PDB) は、移行可能な仮想マシンイメージに関する 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。(CNV-33834)
    - 回避策として、アラートの通知を解除 します。

ノード

- OpenShift Virtualization をアンインストールしても、OpenShift Virtualization によって作成された **feature.node.kubevirt.io** ノードラベルは削除されません。ラベルは手動で削除する必要があります。(CNV-38543)

- コンピュートノードが異なる異種クラスターでは、HyperV reenlightenment が有効になっている仮想マシンは、タイムスタンプカウンタースケーリング(TSC)をサポートしていないノード、TSC 頻度が非対応ノードでスケジュールできません。(BZ#2151169)

ストレージ

- AWS 上のストレージソリューションとして Portworx を使用し、仮想マシンのディスクイメージを作成する場合、ファイルシステムのオーバーヘッドが 2 回考慮されるため、作成されるイメージは予想よりも小さくなる可能性があります。(CNV-40217)
    - 回避策として、最初のプロビジョニングプロセスが完了した後に、永続ボリューム要求 (PVC) を手動で拡張して利用可能なスペースを増やせます。

- 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。(CNV-13500)
    - 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。

- **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。(CNV-23501)
    - 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。

仮想化

- CPU タイプが混在するクラスターでは、仮想マシンの移行が失敗する可能性があります。(CNV-43195)
    - 回避策として、仮想マシン仕様レベル または クラスターレベル で CPU モデルを設定できます。

- 仮想 Trusted Platform Module (vTPM) デバイスを Windows 仮想マシンに追加すると、vTPM デバイスが永続的でない場合でも、BitLocker ドライブ暗号化システムチェックに合格します。これは、**virt-launcher** Pod の存続期間中、永続的ではない vTPM デバイスが一時ストレージを使用して暗号化キーを保存および復元するためです。仮想マシンが移行するか、シャットダウンして再起動すると、vTPM データは失われます。(CNV-36448)

- 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。(CNV-33835)

    - 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。

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

    - 回避策として、レガシー OpenSSL クライアントを TLS 1.3 をサポートするバージョンに更新し、FIPS モードの場合に OpenShift Virtualization が **Modern** TLS セキュリティープロファイル型で TLS 1.3 を使用するようにを設定します。

Web コンソール

- **cluster-admin** 権限がない場合、OpenShift Container Platform クラスターの初回デプロイ時に、Web コンソールを使用してテンプレートまたはインスタンスタイプから仮想マシンを作成すると失敗します。

    - 回避策として、クラスター管理者は最初に config map を作成 し、他のユーザーがテンプレートとインスタンスタイプを使用して仮想マシンを作成できるようにする必要があります。(**CNV-38284**)

- Web コンソールの **Create PersistentVolumeClaim** リストから **With Data upload form** を選択して永続ボリューム要求 (PVC) を作成する際に、**Upload Data** フィールドを使用して PVC にデータをアップロードすると失敗します。(**CNV-37607**)

第3章 スタートガイド

3.1. OPENSHIFT VIRTUALIZATION の開始

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

注記

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

3.1.1. OpenShift Virtualization の計画とインストール

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

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

計画およびインストールのリソース

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

3.1.2. 仮想マシンの作成と管理

仮想マシンを作成します。

- Red Hat イメージから仮想マシンを作成します。
  Red Hat テンプレートまたは インスタンスタイプ を使用して仮想マシンを作成できます。
- カスタムイメージから仮想マシンを作成します。
  仮想マシンを作成するには、コンテナーレジストリーまたは Web ページからカスタムイメージをインポートするか、ローカルマシンからイメージをアップロードするか、永続ボリューム要求 (PVC) を複製することによって実行できます。

仮想マシンをセカンダリーネットワークに接続します。

- Linux ブリッジネットワーク。
- オープン仮想ネットワーク (OVN)- Kubernetes セカンダリーネットワーク。

- シングルルート I/O 仮想化 (SR-IOV) ネットワーク。

注記

VM はデフォルトで Pod ネットワークに接続されます。

仮想マシンに接続します。

- 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。
- Windows 仮想マシンのデスクトップビューアーに接続します。

仮想マシンを管理します。

- Web コンソールを使用して仮想マシンを管理します。
- **virtctl** CLI ツールを使用して仮想マシンを管理します。
- 仮想マシンをエクスポートします。

3.1.3. 次のステップ

- インストール後の設定オプションを確認します。
- ストレージオプションとブートソースの自動更新を設定します。
- モニタリングとヘルスチェックについて学びます。
- ライブマイグレーションについて学びます。
- 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。
- クラスターのチューニングとスケーリング

3.2. CLI ツールの使用

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

libguestfs コマンドラインツールを使用すると、仮想マシンのディスクイメージにアクセスして変更できます。**libguestfs** をデプロイするには、**virtctl libguestfs** コマンドを使用します。

3.2.1. virtctl のインストール

Red Hat Enterprise Linux (RHEL) 9、Linux、Windows、および MacOS オペレーティングシステムに **virtctl** をインストールするには、**virtctl** バイナリーファイルをダウンロードしてインストールします。

RHEL 8 に **virtctl** をインストールするには、OpenShift Virtualization リポジトリーを有効にしてから、**kubevirt-virtctl** パッケージをインストールします。

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

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

手順

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

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

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

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

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

     ```
     $ echo $PATH
     ```

     d. **KUBECONFIG** 環境変数を設定します。

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

   - Windows の場合:
     a. アーカイブファイルを展開します。
     b. 展開したフォルダー階層に移動し、**virtctl** 実行可能ファイルをダブルクリックしてクライアントをインストールします。
     c. **virtctl** バイナリーを **PATH 環境変数** にあるディレクトリーに移動します。
     次のコマンドを実行して、パスを確認できます。

     ```
     C:\> path
     ```

   - macOS の場合:
     a. アーカイブファイルを展開します。
     b. **virtctl** バイナリーを **PATH 環境変数** にあるディレクトリーに移動します。
     次のコマンドを実行して、パスを確認できます。

     ```
     echo $PATH
     ```

3.2.1.2. RHEL 8 への virtctl RPM のインストール

OpenShift Virtualization リポジトリーを有効にし、**kubevirt-virtctl** パッケージをインストールすることで、Red Hat Enterprise Linux (RHEL) 8 に **virtctl** RPM をインストールできます。

前提条件

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

手順

1. **subscription-manager** CLI ツールを使用して次のコマンドを実行し、OpenShift Virtualization リポジトリーを有効にします。

```
# subscription-manager repos --enable cnv-4.16-for-rhel-8-x86_64-rpms
```

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

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

3.2.2. virtctl コマンド

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

注記

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

3.2.2.1. virtctl 情報コマンド

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

表3.1 情報コマンド

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

3.2.2.2. 仮想マシン情報コマンド

virtctl を使用すると、仮想マシンおよび仮想マシンインスタンス (VMI) に関する情報を表示できます。

表3.2 仮想マシン情報コマンド

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

3.2.2.3. 仮想マシンマニフェスト作成コマンド

virtctl create コマンドを使用して、仮想マシン、インスタンスタイプ、および設定のマニフェストを作成できます。

表3.3 仮想マシンマニフェスト作成コマンド

コマンド	設定
virtctl create vm	**VirtualMachine** (VM) マニフェストを作成します。
virtctl create vm --name <vm_name>	仮想マシンの名前を指定して、仮想マシンのマニフェストを作成します。
virtctl create vm --instancetype <instancetype_name>	既存のクラスター全体のインスタンスの種類を使用する仮想マシンのマニフェストを作成します。
virtctl create vm --instancetype=virtualmachineinstancetype/<instancetype_name>	既存の namespaced インスタンスタイプを使用する仮想マシンのマニフェストを作成します。
virtctl create instancetype --cpu <cpu_value> --memory <memory_value> --name <instancetype_name>	クラスター全体のインスタンスタイプのマニフェストを作成します。
virtctl create instancetype --cpu <cpu_value> --memory <memory_value> --name <instancetype_name> --namespace <namespace_value>	namespaced インスタンスタイプのマニフェストを作成します。
virtctl create preference --name <preference_name>	設定の名前を指定して、クラスター全体の仮想マシン設定のマニフェストを作成します。
virtctl create preference --namespace <namespace_value>	namespace 付き仮想マシン設定のマニフェストを作成します。

3.2.2.4. 仮想マシン管理コマンド

virtctl 仮想マシン管理コマンドを使用して、仮想マシンおよび仮想マシンインスタンスを管理および移行します。

表3.4 仮想マシン管理コマンド

コマンド	説明
virtctl start <vm_name>	仮想マシンを開始します。
virtctl start --paused <vm_name>	仮想マシンを一時停止状態で起動します。このオプションを使用すると、VNC コンソールからブートプロセスを中断できます。
virtctl stop <vm_name>	仮想マシンを停止します。
virtctl stop <vm_name> --grace-period 0 --force	仮想マシンを強制停止します。このオプションは、データの不整合またはデータ損失を引き起こす可能性があります。
virtctl pause vm <vm_name>	仮想マシンを一時停止します。マシンの状態がメモリーに保持されます。
virtctl unpause vm <vm_name>	仮想マシンの一時停止を解除します。
virtctl migrate <vm_name>	仮想マシンを移行します。
virtctl migrate-cancel <vm_name>	仮想マシンの移行をキャンセルします。
virtctl restart <vm_name>	仮想マシンを再起動します。

3.2.2.5. 仮想マシン接続コマンド

virtctl 接続コマンドを使用してポートを公開し、仮想マシンおよび仮想マシンインスタンスに接続します。

表3.5 仮想マシン接続コマンド

コマンド	説明
virtctl console <vm_name>	仮想マシンのシリアルコンソールに接続します。
virtctl expose vm <vm_name> --name <service_name> --type <ClusterIP|NodePort|LoadBalancer> --port <port>	仮想マシンの指定されたポートを転送するサービスを作成し、ノードの指定されたポートでサービスを公開します。 例: **virtctl expose vm rhel9_vm --name rhel9-ssh --type NodePort --port 22**
virtctl scp -i <ssh_key> <file_name> <user_name>@<vm_name>	マシンから仮想マシンにファイルをコピーします。このコマンドは、SSH キーペアの秘密キーを使用します。仮想マシンは公開キーを使用して設定する必要があります。

コマンド	説明
virtctl scp -i <ssh_key> <user_name@<vm_name>: <file_name> .	仮想マシンからマシンにファイルをコピーします。このコマンドは、SSH キーペアの秘密キーを使用します。仮想マシンは公開キーを使用して設定する必要があります。
virtctl ssh -i <ssh_key> <user_name>@<vm_name>	仮想マシンとの SSH 接続を開きます。このコマンドは、SSH キーペアの秘密キーを使用します。仮想マシンは公開キーを使用して設定する必要があります。
virtctl vnc <vm_name>	仮想マシンの VNC コンソールに接続します。 **virt-viewer** がインストールされている必要があります。
virtctl vnc --proxy-only=true <vm_name>	ポート番号を表示し、VNC 接続を介してビューアーを使用して手動で VM に接続します。
virtctl vnc --port=<port-number> <vm_name>	ポートが利用可能な場合、その指定されたポートでプロキシーを実行するためにポート番号を指定します。 ポート番号が指定されていない場合、プロキシーはランダムポートで実行されます。

3.2.2.6. 仮想マシンエクスポートコマンド

virtctl vmexport コマンドを使用して、仮想マシン、仮想マシンスナップショット、または永続ボリューム要求 (PVC) からエクスポートされたボリュームを作成、ダウンロード、または削除できます。特定のマニフェストには、OpenShift Virtualization が使用できる形式でディスクイメージをインポートするためのエンドポイントへのアクセスを許可するヘッダーシークレットも含まれています。

表3.6 仮想マシンエクスポートコマンド

コマンド	説明
virtctl vmexport create <vmexport_name> --vm|snapshot|pvc= <object_name>	仮想マシン、仮想マシンスナップショット、または PVC からボリュームをエクスポートするには、**VirtualMachineExport** カスタムリソース (CR) を作成します。 ● **--vm**: 仮想マシンの PVC をエクスポートします。 ● **--snapshot**: **VirtualMachineSnapshot** CR に含まれる PVC をエクスポートします。 ● **--pvc**: PVC をエクスポートします。 ● オプション: **--ttl=1h** は存続時間を指定します。デフォルトの期間は 2 時間です。
virtctl vmexport delete <vmexport_name>	**VirtualMachineExport** CR を手動で削除します。

コマンド	説明
virtctl vmexport download <vmexport_name> --output=<output_file> --volume=<volume_name>	**VirtualMachineExport** CR で定義されたボリュームをダウンロードします。 ● **--output** はファイル形式を指定します。例: **disk.img.gz**。 ● **--volume** は、ダウンロードするボリュームを指定します。使用可能なボリュームが 1 つだけの場合、このフラグはオプションです。 オプション: ● **--keep-vme** は、ダウンロード後に **VirtualMachineExport** CR を保持します。デフォルトの動作では、ダウンロード後に **VirtualMachineExport** CR を削除します。 ● **--insecure** は、安全でない HTTP 接続を有効にします。
virtctl vmexport download <vmexport_name> --<vm|snapshot|pvc>=<object_name> --output=<output_file> --volume=<volume_name>	**VirtualMachineExport** CR を作成し、CR で定義されたボリュームをダウンロードします。
virtctl vmexport download export --manifest	既存のエクスポートのマニフェストを取得します。マニフェストにはヘッダーシークレットが含まれていません。
virtctl vmexport download export --manifest --vm=example	仮想マシンサンプルの仮想マシンエクスポートを作成し、マニフェストを取得します。マニフェストにはヘッダーシークレットが含まれていません。
virtctl vmexport download export --manifest --snap=example	仮想マシンスナップショットの例の仮想マシンエクスポートを作成し、マニフェストを取得します。マニフェストにはヘッダーシークレットが含まれていません。
virtctl vmexport download export --manifest --include-secret	既存のエクスポートのマニフェストを取得します。マニフェストにはヘッダーシークレットが含まれています。
virtctl vmexport download export --manifest --manifest-output-format=json	既存のエクスポートのマニフェストを json 形式で取得します。マニフェストにはヘッダーシークレットが含まれていません。
virtctl vmexport download export --manifest --include-secret --output=manifest.yaml	既存のエクスポートのマニフェストを取得します。マニフェストにはヘッダーシークレットが含まれており、指定されたファイルにそれを書き込みます。

3.2.2.7. 仮想マシンメモリーダンプコマンド

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

前提条件

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

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

メモリーダンプのダウンロード

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

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

表3.7 仮想マシンメモリーダンプコマンド

コマンド	説明
virtctl memory-dump get <vm_name> --claim-name= <pvc_name>	仮想マシンのメモリーダンプを PVC に保存します。メモリーダンプのステータスは、**VirtualMachine** リソースの **status** セクションに表示されます。 オプション: ● **--create-claim** は、適切なサイズで新しい PVC を作成します。このフラグには次のオプションがあります。 ○ **--storage-class=<storage_class>**: PVC のストレージクラスを指定します。 ○ **--access-mode=<access_mode>**: **ReadWriteOnce** または **ReadWriteMany** を指定します。
virtctl memory-dump get <vm_name>	同じ PVC で **virtctl memory-dump** コマンドを再実行します。 このコマンドは、以前のメモリーダンプを上書きします。

コマンド	説明
virtctl memory-dump remove <vm_name>	メモリーダンプを削除します。 ターゲット PVC を変更する場合は、メモリーダンプを手動で削除する必要があります。 このコマンドは、**VirtualMachine** リソースの **status** セクションにメモリーダンプが表示されないように、仮想マシンと PVC の間の関連付けを削除します。PVC は影響を受けません。

3.2.2.8. ホットプラグおよびホットアンプラグコマンド

virtctl を使用して、実行中の仮想マシンおよび仮想マシンインスタンス (VMI) にリソースを追加または削除します。

表3.8 ホットプラグおよびホットアンプラグコマンド

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

3.2.2.9. イメージアップロードコマンド

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

表3.9 イメージアップロードコマンド

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

3.2.3. virtctl を使用した libguestfs のデプロイ

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

手順

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

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

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

3.2.3.1. Libguestfs および virtctl guestfs コマンド

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

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

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

コマンド	説明
virt-cat -a /dev/vda /rpm-list	ターミナルで **virt-customize -a /dev/vda --run-command 'rpm -qa > /rpm-list'** コマンドを使用して作成されたすべての RPM の出力ファイルのリストを表示します。
virt-sysprep -a /dev/vda	テンプレートとして使用する仮想マシンディスクイメージをシールします。

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

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

3.2.4. Ansible の使用

OpenShift Virtualization 用の Ansible コレクションを使用するには、Red Hat Ansible Automation Hub (Red Hat Hybrid Cloud Console) を参照してください。

第4章 インストール

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

インストール方法の考慮事項

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

Red Hat OpenShift Data Foundation

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。

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。

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。

4.1.1. サポート対象のプラットフォーム

OpenShift Virtualization では、以下のプラットフォームを使用できます。

- オンプレミスのベアメタルサーバー。OpenShift Virtualization で使用するベアメタルクラスターの計画 を参照してください。
- Amazon Web Services ベアメタルインスタンス。カスタマイズを使用して AWS にクラスターをインストールする を参照してください。
- IBM Cloud® ベアメタルサーバー。Deploy OpenShift Virtualization on IBM Cloud® Bare Metal nodes を参照してください。

重要

IBM Cloud® ベアメタルサーバーへの 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の機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

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

4.1.1.1. AWS ベアメタル上の OpenShift Virtualization

OpenShift Virtualization は、Amazon Web Services (AWS) ベアメタル OpenShift Container Platform クラスターで実行できます。

注記

OpenShift Virtualization は、AWS ベアメタルクラスターと同じ設定要件を持つ Red Hat OpenShift Service on AWS (ROSA) Classic クラスターでもサポートされています。

クラスターを設定する前に、サポート対象の機能と制限に関する以下の要約を確認してください。

インストール

- 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、ワーカーノードにベアメタルインスタンスタイプを指定する必要があります。たとえば、x86_64 アーキテクチャーをベースとするマシンの **c5n.metal** タイプの値を使用できます。**install-config.yaml** ファイルを編集して、ベアメタルインスタンスタイプを指定します。
  詳細は、AWS へのインストールに関する OpenShift Container Platform ドキュメントを参照してください。

仮想マシン (VM) へのアクセス

- **virtctl** CLI ツールまたは OpenShift Container Platform Web コンソールを使用して仮想マシンにアクセスする方法に変更はありません。
- **NodePort** または **LoadBalancer** サービスを使用して、仮想マシンを公開できます。
  - OpenShift Container Platform は AWS でロードバランサーを自動的に作成し、そのライフサイクルを管理するため、ロードバランサーのアプローチが推奨されます。また、セキュリティーグループはロードバランサー用にも作成され、アノテーションを使用して既存のセキュリティーグループをアタッチできます。サービスを削除すると、OpenShift Container Platform はロードバランサーとそれに関連付けられたリソースを削除します。

ネットワーク

- Single Root I/O Virtualization (SR-IOV) またはブリッジ Container Network Interface (CNI) ネッ

トワーク (仮想 LAN (VLAN) を含む) は使用できません。アプリケーションにフラットレイヤー 2 ネットワークが必要な場合や、IP プールを制御する必要がある場合は、OVN-Kubernetes セカンダリーオーバーレイネットワークを使用することを検討してください。

ストレージ

- 基盤となるプラットフォームとの連携がストレージベンダーによって認定されている任意のストレージソリューションを使用できます。

重要

AWS ベアメタルクラスターと ROSA クラスターでは、サポートされているストレージソリューションが異なる場合があります。ストレージベンダーにサポートを確認してください。

- OpenShift Virtualization で Amazon Elastic File System (EFS) または Amazon Elastic Block Store (EBS) を使用すると、パフォーマンスと機能が制限される可能性があります。ライブマイグレーション、高速仮想マシンの作成、仮想マシンスナップショット機能の有効化を行うには、ReadWriteMany (RWX)、クローン作成、スナップショットをサポートする CSI ストレージの使用を検討してください。

Hosted Control Plane (HCP)

- OpenShift Virtualization の HCP は現在、AWS インフラストラクチャーではサポートされていません。

関連情報

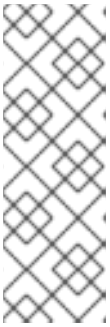
- OVN-Kubernetes セカンダリーネットワークへの仮想マシンの接続
- サービスを使用して仮想マシンを公開する

4.1.2. 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。

4.1.2.1. CPU の要件

- Red Hat Enterprise Linux (RHEL) 9 でサポート。
サポートされている CPU の Red Hat Ecosystem Catalog を参照してください。



注記

ワーカーノードの CPU が異なる場合は、CPU ごとに機能が異なるため、ライブマイグレーションが失敗する可能性があります。この問題は、ワーカーノードに適切な容量の CPU が搭載されていることを確認し、仮想マシンのノードアフィニティールールを設定することで軽減できます。

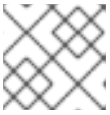
詳細は、ノードアフィニティーの required (必須) ルールの設定 を参照してください。

- AMD および Intel 64 ビットアーキテクチャー (x86-64-v2) のサポート。

- Intel 64 または AMD64 CPU 拡張機能のサポート。
- Intel VT または AMD-V ハードウェア仮想化拡張機能が有効化されている。
- NX (実行なし) フラグが有効。

4.1.2.2. 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)。
  詳細は、RHCOS について を参照してください。



注記

RHEL ワーカーノードはサポートされていません。

4.1.2.3. ストレージ要件

- OpenShift Container Platform によるサポート。ストレージの最適化 を参照してください。
- デフォルトの OpenShift Virtualization または OpenShift Container Platform ストレージクラスを作成する必要があります。これは、仮想マシンワークロード固有のストレージニーズに対応し、最適化されたパフォーマンス、信頼性、ユーザーエクスペリエンスを提供することを目的としています。OpenShift Virtualization と OpenShift Container Platform の両方にデフォルトのストレージクラスが存在する場合、仮想マシンディスクの作成時には OpenShift Virtualization のクラスが優先されます。

注記

ストレージクラスを仮想化ワークロードのデフォルトとしてマークするには、アノテーション **storageclass.kubevirt.io/is-default-virt-class** を **"true"** に設定します。

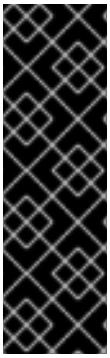
- ストレージプロビジョナーがスナップショットをサポートしている場合は、**VolumeSnapshotClass** オブジェクトをデフォルトのストレージクラスに関連付ける必要があります。

4.1.2.3.1. 仮想マシンディスクのボリュームとアクセスモードについて

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

最良の結果を得るには、**ReadWriteMany** (RWX) アクセスモードと **Block** ボリュームモードを使用してください。これは、以下の理由により重要です。

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



重要

次の設定の仮想マシンをライブマイグレーションすることはできません。

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

これらの仮想マシンの **evictionStrategy** フィールドを **None** に設定します。**None** ストラテジーでは、ノードの再起動中に仮想マシンの電源がオフになります。

4.1.3. ライブマイグレーションの要件

- **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

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。

- 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。
- ライブマイグレーション 専用の Multus ネットワーク を強く推奨します。専用ネットワークは、移行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。

4.1.4. 物理リソースのオーバーヘッド要件

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

重要

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

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

クラスターメモリーのオーバーヘッド

> Memory overhead per infrastructure node ≈ 150 MiB

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

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

仮想マシンのメモリーオーバーヘッド

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

1. **virt-launcher** Pod で実行されるプロセスに必要です。
2. 仮想マシンが要求する仮想 CPU の数。
3. 仮想マシンが要求する仮想グラフィックスカードの数。
4. 追加のメモリーオーバーヘッド:
    - お使いの環境に Single Root I/O Virtualization (SR-IOV) ネットワークデバイスまたは Graphics Processing Unit (GPU) が含まれる場合、それぞれのデバイスに 1 GiB の追加のメモリーオーバーヘッドを割り当てます。
    - Secure Encrypted Virtualization (SEV) が有効な場合は、256 MiB を追加します。
    - Trusted Platform Module (TPM) が有効な場合は、53 MiB を追加します。

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。

ストレージのオーバーヘッド

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

クラスターストレージオーバーヘッド

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

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

仮想マシンのストレージオーバーヘッド

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

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関する CPU の影響は最小 2 コアで示されます。

4.1.5. シングルノード OpenShift の違い

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

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

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

関連情報

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

4.1.6. 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

4.1.7. クラスターの高可用性オプション

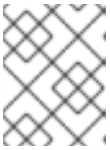
クラスターには、次の高可用性 (HA) オプションのいずれかを設定できます。

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

注記

インストーラー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を使用し、適切に設定された **MachineHealthCheck** リソースを使用してインストールされた OpenShift Container Platform クラスターでは、ノードがマシンヘルスチェックに失敗し、クラスターで使用できなくなった場合、そのノードはリサイクルされます。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、実行戦略がそれらの結果にどのように影響するかに関する詳細は、戦略の実行 を参照してください。

- IPI と非 IPI の両方の自動高可用性は、OpenShift Container Platform クラスターで **Node Health Check Operator** を使用して **NodeHealthCheck** コントローラーをデプロイすることで利用できます。コントローラーは異常なノードを特定し、Self Node Remediation Operator や Fence Agents Remediation Operator などの修復プロバイダーを使用して異常なノードを修復します。ノードの修復、フェンシング、メンテナンスの詳細は、Red Hat OpenShift のワークロードの可用性 を参照してください。
- モニタリングシステムまたは有資格者を使用してノードの可用性をモニターすることにより、あらゆるプラットフォームの高可用性を利用できます。ノードが失われた場合は、これをシャットダウンして **oc delete node <lost_node>** を実行します。



注記

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

4.2. 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。

重要

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 (OLM) 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は、OLM でプロキシーサポートを設定 して OperatorHub にアクセスできます。

4.2.1. OpenShift Virtualization Operator のインストール

OpenShift Container Platform Web コンソールまたはコマンドラインを使用して、OpenShift Virtualization Operator をインストールします。

4.2.1.1. Web コンソールを使用した OpenShift Virtualization Operator のインストール

OpenShift Container Platform Web コンソールを使用して、OpenShift Virtualization Operator をデプロイできます。

前提条件

- OpenShift Container Platform 4.16 をクラスターにインストールしている。

- **cluster-admin** パーミッションを持つユーザーとして OpenShift Container Platform Web コンソールにログインすること。

手順

1. **Administrator** パースペクティブから、**Operators → OperatorHub** をクリックします。
2. **Filter by keyword** に **Virtualization** と入力します。
3. **Red Hat** ソースラベルが示されている **OpenShift Virtualization Operator** タイルを選択します。
4. Operator に関する情報を確認してから、**Install** をクリックします。
5. **Install Operator** ページで以下を行います。
   a. 選択可能な **Update Channel** オプションの一覧から **stable** を選択します。これにより、OpenShift Container Platform バージョンと互換性がある OpenShift Virtualization のバージョンをインストールすることができます。
   b. **インストールされた namespace** の場合、**Operator recommended namespace** オプションが選択されていることを確認します。これにより、Operator が必須の **openshift-cnv** namespace にインストールされます。この namespace は存在しない場合は、自動的に作成されます。

警告

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

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

警告

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

6. **Install** をクリックし、Operator を **openshift-cnv** namespace で利用可能にします。

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

検証

- **Workloads → Pods** ページに移動して、OpenShift Virtualization Pod がすべて **Running** 状態になるまでこれらの Pod をモニターします。すべての Pod で **Running** 状態が表示された後に、OpenShift Virtualization を使用できます。

4.2.1.2. コマンドラインを使用した OpenShift Virtualization Operator のインストール

OpenShift Virtualization カタログをサブスクライブし、クラスターにマニフェストを適用して OpenShift Virtualization Operator をインストールします。

4.2.1.2.1. 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。

前提条件

- OpenShift Container Platform 4.16 をクラスターにインストールしている。
- 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.v4.16.2
  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。

4.2.1.2.2. 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 カタログへのサブスクリプション。
- **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.v4.16.2   OpenShift Virtualization   4.16.2
Succeeded
```

4.2.2. 次のステップ

- 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。

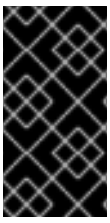
4.3. OPENSHIFT VIRTUALIZATION のアンインストール

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

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

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

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



重要

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

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

4.3.1.1. HyperConverged カスタムリソースの削除

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

前提条件

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

手順

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

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

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

前提条件

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

手順

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

注記

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

4.3.1.3. Web コンソールを使用した namespace の削除

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

前提条件

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

手順

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

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

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

前提条件

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

手順

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

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

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

前提条件

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

手順

1. **HyperConverged** カスタムリソースを削除します。

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

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

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

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

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

4. OpenShift Virtualization namespace を削除します。

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

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

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

出力例

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

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

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

関連情報

- 仮想マシンの削除
- 仮想マシンインスタンスの削除

第5章 インストール後の設定

5.1. インストール後の設定

通常、次の手順は OpenShift Virtualization のインストール後に実行されます。環境に関連するコンポーネントを設定できます。

- OpenShift Virtualization Operator、ワークロード、およびコントローラーのノード配置ルール
- ネットワーク設定:
  - Kubernetes NMState および SR-IOV Operator のインストール
  - 仮想マシンへの外部アクセスのための Linux ブリッジネットワークの設定
  - 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
  - SR-IOV ネットワークの設定
  - OpenShift Container Platform Web コンソールを使用したロードバランサーサービスの作成の有効化
- ストレージの設定:
  - Container Storage Interface (CSI) のデフォルトのストレージクラスの定義
  - ホストパスプロビジョナー (HPP) を使用したローカルストレージの設定

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

ベアメタルノード上の仮想マシンのデフォルトのスケジューリングは適切です。任意で、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できます。

注記

OpenShift Virtualization のインストール後に一部のコンポーネントに対してノード配置ルールを設定できますが、ワークロードに対してノード配置ルールを設定する場合は仮想マシンが存在できません。

5.2.1. OpenShift Virtualization コンポーネントのノード配置ルールについて

ノード配置ルールは次のタスクに使用できます。

- 仮想マシンは、仮想化ワークロードを対象としたノードにのみデプロイしてください。
- Operator はインフラストラクチャーノードにのみデプロイメントします。
- ワークロード間の分離を維持します。

オブジェクトに応じて、以下のルールタイプを 1 つ以上使用できます。

nodeSelector

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

affinity

より表現的な構文を使用して、ノードと Pod に一致するルールを設定できます。アフィニティーを使用すると、ルールの適用方法に追加のニュアンスを持たせることができます。たとえば、ルールが要件ではなく設定であると指定できます。ルールが優先の場合、ルールが満たされていない場合でも Pod はスケジュールされます。

toleration

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。

5.2.2. ノード配置ルールの適用

コマンドラインを使用して **Subscription**、**HyperConverged**、または **HostPathProvisioner** オブジェクトを編集することで、ノード配置ルールを適用できます。

前提条件

- **oc** CLI ツールがインストールされている。
- クラスター管理者の権限でログインしています。

手順

1. 次のコマンドを実行して、デフォルトのエディターでオブジェクトを編集します。

```
$ oc edit <resource_type> <resource_name> -n {CNVNamespace}
```

2. 変更を適用するためにファイルを保存します。

5.2.3. ノード配置ルールの例

Subscription、**HyperConverged**、または **HostPathProvisioner** オブジェクトを編集することで、OpenShift Virtualization コンポーネントのノード配置ルールを指定できます。

5.2.3.1. サブスクリプションオブジェクトノード配置ルールの例

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。

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。

Subscription オブジェクトは、**affinity** ノードの配置ルールをサポートしていません。

nodeSelector ルールを使用した Subscription オブジェクトの例

```
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.v4.16.2
  channel: "stable"
  config:
    nodeSelector:
      example.io/example-infra-key: example-infra-value (1)
```

(1) OLM は、**example.io/example-infra-key = example-infra-value** というラベルのノードに OpenShift Virtualization Operator をデプロイします。

tolerations ルールを備えた Subscription オブジェクトの例

```
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.v4.16.2
  channel: "stable"
  config:
    tolerations:
    - key: "key"
      operator: "Equal"
      value: "virtualization" (1)
      effect: "NoSchedule"
```

(1) OLM は、**key = virtualization:NoSchedule** テイントというラベルの付いたノードに OpenShift Virtualization Operator をデプロイします。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。

5.2.3.2. HyperConverged オブジェクトノード配置ルールの例

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 カスタムリソース (CR) ファイルに **nodePlacement** オブジェクトを編集できます。

nodeSelector ルールを使用した HyperConverged オブジェクトの例

```
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 1
  workloads:
    nodePlacement:
      nodeSelector:
        example.io/example-workloads-key: example-workloads-value 2
```

1 インフラストラクチャーリソースは、**example.io/example-infra-key = example-infra-value** というラベルの付いたノードに配置されます。

2 ワークロードは、**example.io/example-workloads-key = example-workloads-value** というラベルの付いたノードに配置されます。

affinity ルールを使用した HyperConverged オブジェクトの例

```
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 1
  workloads:
    nodePlacement:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-workloads-key 2
                operator: In
                values:
                - example-workloads-value
          preferredDuringSchedulingIgnoredDuringExecution:
          - weight: 1
            preference:
              matchExpressions:
              - key: example.io/num-cpus
                operator: Gt
                values:
                - 8 3
```

1. インフラストラクチャーリソースは、**example.io/example-infra-key = example-value** というラベルの付いたノードに配置されます。

2. ワークロードは、**example.io/example-workloads-key = example-workloads-value** というラベルの付いたノードに配置されます。

3. 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。

tolerations ルールを備えた HyperConverged オブジェクトの例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  workloads:
    nodePlacement:
      tolerations: (1)
      - key: "key"
        operator: "Equal"
        value: "virtualization"
        effect: "NoSchedule"
```

1. OpenShift Virtualization コンポーネント用に予約されたノードには、**key = virtualization:NoSchedule** テイントのラベルが付けられます。許容範囲が一致する Pod のみが予約済みノードでスケジュールされます。

5.2.3.3. HostPathProvisioner オブジェクトノード配置ルールの例

HostPathProvisioner オブジェクトは、直接編集することも、Web コンソールを使用して編集することもできます。

> **警告**
>
> ホストパスプロビジョナーと OpenShift Virtualization コンポーネントを同じノード上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。仮想マシンを実行することはできません。

ホストパスプロビジョナー (HPP) ストレージクラスを使用して仮想マシン (VM) をデプロイした後、ノードセレクターを使用して同じノードからホストパスプロビジョナー Pod を削除できます。ただし、少なくともその特定のノードについては、まずその変更を元に戻し、仮想マシンを削除しようとする前に Pod が実行されるのを待つ必要があります。

ノード配置ルールを設定するには、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** フィールドに **nodeSelector**、**affinity**、または **tolerations** を指定します。

nodeSelector ルールを使用した HostPathProvisioner オブジェクトの例

```
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 (1)
```

(1) ワークロードは、**example.io/example-workloads-key = example-workloads-value** というラベルの付いたノードに配置されます。

5.2.4. 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 taint を使用した Pod 配置の制御

5.3. インストール後のネットワーク設定

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。

OpenShift Virtualization をインストールした後、ネットワーク Operator をインストールし、追加のネットワークを設定できます。

5.3.1. ネットワーキングオペレーターのインストール

ライブマイグレーションまたは仮想マシン (VM) への外部アクセス用に Linux ブリッジネットワークを設定するには、Kubernetes NMState Operator をインストールする必要があります。インストール手順については、Web コンソールを使用した Kubernetes NMState Operator のインストール を参照してください。

SR-IOV Operator をインストールして、SR-IOV ネットワークデバイスとネットワークアタッチメントを管理できます。インストール手順については、SR-IOV Network Operator のインストール を参照してください。

MetalLB Operator を追加すると、クラスター上の MetalLB インスタンスのライフサイクルを管理できます。インストール手順については、Web コンソールを使用した OperatorHub からの MetalLB Operator のインストール を参照してください。

5.3.2. Linux ブリッジネットワークの設定

Kubernetes NMState Operator をインストールしたら、ライブマイグレーションまたは仮想マシン (VM) への外部アクセス用に Linux ブリッジネットワークを設定できます。

5.3.2.1. Linux ブリッジ NNCP の作成

Linux ブリッジネットワークの **NodeNetworkConfigurationPolicy** (NNCP) マニフェストを作成できます。

前提条件

- Kubernetes NMState Operator がインストールされている。

手順

- **NodeNetworkConfigurationPolicy** マニフェストを作成します。この例には、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy (1)
spec:
  desiredState:
    interfaces:
      - name: br1 (2)
        description: Linux bridge with eth1 as a port (3)
        type: linux-bridge (4)
        state: up (5)
        ipv4:
          enabled: false (6)
        bridge:
          options:
            stp:
              enabled: false (7)
          port:
            - name: eth1 (8)
```

(1) ポリシーの名前。

(2) インターフェイスの名前。

(3) オプション: 人間が判読できるインターフェイスの説明。

(4) インターフェイスのタイプ。この例では、ブリッジを作成します。

(5) 作成後のインターフェイスの要求された状態。

(6) この例では IPv4 を無効にします。

(7) この例では STP を無効にします。

(8) ブリッジが接続されているノード NIC。

5.3.2.2. Web コンソールを使用した Linux ブリッジ NAD の作成

OpenShift Container Platform Web コンソールを使用して、ネットワーク接続定義 (NAD) を作成して、Pod および仮想マシンに layer-2 ネットワークを提供できます。

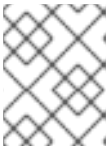
Linux ブリッジネットワーク接続定義は、仮想マシンを VLAN に接続するための最も効率的な方法です。

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。

手順

1. Web コンソールで、**Networking → NetworkAttachmentDefinitions** をクリックします。
2. **Create Network Attachment Definition**をクリックします。



注記

ネットワーク接続定義は Pod または仮想マシンと同じ namespace にある必要があります。

3. 一意の **Name** およびオプションの **Description** を入力します。
4. **Network Type** リストから **CNV Linux bridge**を選択します。
5. **Bridge Name** フィールドにブリッジの名前を入力します。
6. オプション: リソースに VLAN ID が設定されている場合、**VLAN Tag Number** フィールドに ID 番号を入力します。
7. オプション: **MAC Spoof Check**を選択して、MAC スプーフフィルタリングを有効にします。この機能により、Pod を終了するための MAC アドレスを 1 つだけ許可することで、MAC スプーフィング攻撃に対してセキュリティーを確保します。
8. **Create** をクリックします。

次のステップ

- Linux ブリッジネットワークへの仮想マシンの接続

5.3.3. ライブマイグレーション用のネットワークの設定

Linux ブリッジネットワークを設定した後、ライブマイグレーション用の専用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。

5.3.3.1. 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。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。
- 各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) があります。
- ライブマイグレーション用の NIC は同じ VLAN に接続されます。

手順

1. 次の例に従って、**NetworkAttachmentDefinition** マニフェストを作成します。

   設定ファイルのサンプル

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: my-secondary-network 1
  namespace: openshift-cnv 2
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", 3
    "mode": "bridge",
    "ipam": {
      "type": "whereabouts", 4
      "range": "10.200.5.0/24" 5
    }
  }'
```

   1. **NetworkAttachmentDefinition** オブジェクトの名前を指定します。
   2 3. ライブマイグレーションに使用する NIC の名前を指定します。
   4. NAD にネットワークを提供する CNI プラグインの名前を指定します。
   5. セカンダリーネットワークの IP アドレス範囲を指定します。この範囲は、メインネットワークの IP アドレスと重複してはなりません。

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。

HyperConverged マニフェストの例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  liveMigrationConfig:
    completionTimeoutPerGiB: 800
    network: <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を指定します。

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}'
```

5.3.3.2. Web コンソールを使用して専用ネットワークを選択する

OpenShift Container Platform Web コンソールを使用して、ライブマイグレーション用の専用ネットワークを選択できます。

前提条件

- ライブマイグレーション用に Multus ネットワークが設定されている。

手順

1. OpenShift Container Platform Web コンソールで **Virtualization** > **Overview** に移動します。
2. **Settings** タブをクリックし、**Live migration** をクリックします。
3. **Live migration network** リストからネットワークを選択します。

5.3.4. SR-IOV ネットワークの設定

SR-IOV Operator をインストールした後、SR-IOV ネットワークを設定できます。

5.3.4.1. 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: Virtual Function の最大転送単位 (MTU) の値を指定します。MTU の最大値は NIC モデルによって異なります。

7 SR-IOV 物理ネットワークデバイス用に作成する仮想機能 (VF) の数を指定します。Intel ネットワークインターフェイスコントローラー (NIC) の場合、VF の数はデバイスがサポートする VF の合計よりも大きくすることはできません。Mellanox NIC の場合、VF の数は **127** よりも大きくすることはできません。

8 **nicSelector** マッピングは、Operator が設定するイーサネットデバイスを選択します。すべてのパラメーターの値を指定する必要はありません。意図せずにイーサネットデバイスを選択する可能性を最低限に抑えるために、イーサネットアダプターを正確に特定できるようにすることが推奨されます。**rootDevices** を指定する場合、**vendor**、**deviceID**、または **pfName** の値も指定する必要があります。**pfNames** と **rootDevices** の両方を同時に指定する場合、それらが同一のデバイスをポイントすることを確認します。

9 オプション: SR-IOV ネットワークデバイスのベンダー 16 進コードを指定します。許可される値は **8086** または **15b3** のいずれかのみになります。

10 オプション: SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。許可される値は **158b**、**1015**、**1017** のみになります。

11 オプション: このパラメーターは、1 つ以上のイーサネットデバイスの物理機能 (PF) 名の配列を受け入れます。

12 このパラメーターは、イーサネットデバイスの物理機能に関する 1 つ以上の PCI バスアドレスの配列を受け入れます。以下の形式でアドレスを指定します: **0000:02:00.1**

13 OpenShift Virtualization の仮想機能には、**vfio-pci** ドライバータイプが必要です。

14 オプション: Remote Direct Memory Access (RDMA) モードを有効にするかどうかを指定します。Mellanox カードの場合、**isRdma** を **false** に設定します。デフォルト値は **false** です。

注記

isRDMA フラグが **true** に設定される場合、引き続き RDMA 対応の VF を通常のネットワークデバイスとして使用できます。デバイスはどちらのモードでも使用できます。

2. オプション: SR-IOV 対応のクラスターノードにまだラベルが付いていない場合は、**SriovNetworkNodePolicy.Spec.NodeSelector** でラベルを付けます。ノードのラベル付けの詳細は、「ノードのラベルを更新する方法について」を参照してください。

3. **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f <name>-sriov-node-network.yaml
```

ここで、**<name>** はこの設定の名前を指定します。

設定の更新が適用された後に、**sriov-network-operator** namespace のすべての Pod が **Running** ステータスに移行します。

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

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

次のステップ

- 仮想マシンの SR-IOV ネットワークへの割り当て

5.3.5. Web コンソールを使用したロードバランサーサービスの作成の有効化

OpenShift Container Platform Web コンソールを使用して、仮想マシン (VM) のロードバランサーサービスの作成を有効にすることができます。

前提条件

- クラスターのロードバランサーが設定されました。
- **cluster-admin** ロールを持つユーザーとしてログインしている。

手順

1. **Virtualization → Overview** に移動します。
2. **Settings** タブで、**Cluster** をクリックします。
3. Expand **General settings** と **SSH configuration** を展開します。
4. **SSH over LoadBalancer service** をオンに設定します。

5.4. インストール後のストレージ設定

次のストレージ設定タスクは必須です。

- クラスターの 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。そうしないと、クラスターは自動ブートソース更新を受信できません。
- ストレージプロバイダーが CDI によって認識されない場合は、storage profiles を設定する必要があります。ストレージプロファイルは、関連付けられたストレージクラスに基づいて推奨されるストレージ設定を提供します。

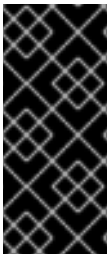
オプション: ホストパスプロビジョナー (HPP) を使用して、ローカルストレージを設定できます。

Containerized Data Importer (CDI)、データボリューム、自動ブートソース更新の設定など、その他のオプションについては、ストレージ設定の概要 を参照してください。

5.4.1. HPP を使用したローカルストレージの設定

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 Operator は HPP プロビジョナーを作成します。

HPP 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。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。



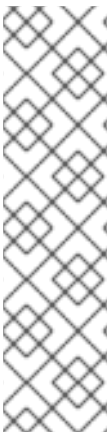
重要

HPP 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、ストレージプールがオペレーティングシステムパーティションをいっぱいにする可能性があります。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ると、パフォーマンスに影響が生じたり、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。

5.4.1.1. 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) を使用するには、コンテナーストレージインターフェイス (CSI) ドライバーに関連するストレージクラスを作成する必要があります。

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。



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。

手順

1. **storageclass_csi.yaml** ファイルを作成して、ストレージクラスを定義します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hostpath-csi
provisioner: kubevirt.io.hostpath-provisioner
reclaimPolicy: Delete (1)
volumeBindingMode: WaitForFirstConsumer (2)
parameters:
  storagePool: my-storage-pool (3)
```

1. **reclaimPolicy** には、**Delete** および **Retain** の 2 つの値があります。値を指定しない場合、デフォルト値は **Delete** です。
2. **volumeBindingMode** パラメーターは、動的プロビジョニングとボリュームのバインディングが実行されるタイミングを決定します。**WaitForFirstConsumer** を指定して、永続ボリューム要求 (PVC) を使用する Pod が作成されるまで PV のバインディングおよびプロビジョニングを遅延させます。これにより、PV が Pod のスケジュール要件を満たすようになります。
3. HPP CR で定義されているストレージプールの名前を指定します。

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
```

5.5. より高い仮想マシンワークロード密度の設定

仮想マシンの数を増やすには、メモリー (RAM) の量をオーバーコミットして、クラスター内の仮想マシンワークロード密度を高く設定します。

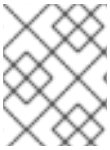
重要

より高いワークロード密度を設定する機能は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

次のワークロードは、高いワークロード密度に特に適しています。

- 多数の類似ワークロード
- 過少使用されたワークロード



注記

メモリーがオーバーコミットされると、ワークロード密度が高くなりますが、使用率の高いシステムのワークロードパフォーマンスが低下する可能性もあります。

5.5.1. wasp-agent を使用して仮想マシンワークロード密度を高く設定する

wasp-agent コンポーネントにより、OpenShift Container Platform クラスターはスワップリソースを仮想マシンワークロードに割り当てることができます。スワップの使用は、ワーカーノードでのみサポートされます。

重要

スワップリソースは、**Burstable** Quality of Service (QoS) クラスの仮想マシンワークロード (VM Pod) にのみ割り当てることができます。**Guaranteed** QoS クラスの仮想マシン Pod と、仮想マシンに属していない任意の QoS クラスの Pod は、リソースをスワップできません。

QoS クラスの説明については、Configure Quality of Service for Pods (Kubernetes ドキュメント) を参照してください。

前提条件

- **oc** ツールが利用できる。
- cluster-admin ロールでクラスターにログイン済みである。
- メモリーのオーバーコミット率が定義済みである。
- ノードはワーカープールに属している。

手順

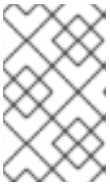
1. 次のコマンドを入力して、特権サービスアカウントを作成します。

```
$ oc adm new-project wasp
```

```
$ oc create sa -n wasp wasp
```

```
$ oc create clusterrolebinding wasp --clusterrole=cluster-admin --serviceaccount=wasp:wasp
```

```
$ oc adm policy add-scc-to-user -n wasp privileged -z wasp
```



注記

wasp-agent コンポーネントは、OCI フックをデプロイして、ノードレベルでコンテナーのスワップ使用を有効にします。低レベルの性質上、**DaemonSet** オブジェクトには特権が必要です。

2. 次のように **DaemonSet** オブジェクトを作成して **wasp-agent** をデプロイします。

```
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: wasp-agent
  namespace: wasp
  labels:
    app: wasp
    tier: node
spec:
  selector:
    matchLabels:
      name: wasp
  template:
```

```
    metadata:
      annotations:
        description: >-
          Configures swap for workloads
      labels:
          name: wasp
    spec:
      serviceAccountName: wasp
      hostPID: true
      hostUsers: true
      terminationGracePeriodSeconds: 5
      containers:
        - name: wasp-agent
          image: >-
            registry.redhat.io/container-native-virtualization/wasp-agent-rhel9:v4.16
          imagePullPolicy: Always
          env:
          - name: "FSROOT"
            value: "/host"
          resources:
            requests:
              cpu: 100m
              memory: 50M
          securityContext:
            privileged: true
          volumeMounts:
          - name: host
            mountPath: "/host"
      volumes:
      - name: host
        hostPath:
          path: "/"
      priorityClassName: system-node-critical
  updateStrategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 10%
      maxSurge: 0
status: {}
```

3. スワップを許可するように **kubelet** サービスを設定します。

    a. 例に示すように、**KubeletConfiguration** ファイルを作成します。

    KubeletConfiguration ファイルの例

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config
spec:
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ''  # MCP
      #machine.openshift.io/cluster-api-machine-role: worker # machine
      #node-role.kubernetes.io/worker: '' # node
```

```
    kubeletConfig:
      failSwapOn: false
      evictionSoft:
        memory.available: "1Gi"
      evictionSoftGracePeriod:
        memory.available: "10s"
```

クラスターが既存の **KubeletConfiguration** ファイルをすでに使用している場合は、**spec** セクションに以下を追加します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: custom-config
# ...
spec
# ...
    kubeletConfig:
      evictionSoft:
        memory.available: 1Gi
      evictionSoftGracePeriod:
        memory.available: 1m30s
      failSwapOn: false
```

b. 以下のコマンドを実行します。

```
$ oc wait mcp worker --for condition=Updated=True
```

4. 次のとおり、スワップをプロビジョニングするための **MachineConfig** オブジェクトを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 90-worker-swap
spec:
  config:
    ignition:
      version: 3.4.0
    systemd:
      units:
      - contents: |
          [Unit]
          Description=Provision and enable swap
          ConditionFirstBoot=no

          [Service]
          Type=oneshot
          Environment=SWAP_SIZE_MB=5000
          ExecStart=/bin/sh -c "sudo dd if=/dev/zero of=/var/tmp/swapfile
count=${SWAP_SIZE_MB} bs=1M && \
          sudo chmod 600 /var/tmp/swapfile && \
          sudo mkswap /var/tmp/swapfile && \
```

```
          sudo swapon /var/tmp/swapfile && \
          free -h && \
          sudo systemctl set-property --runtime system.slice MemorySwapMax=0
IODeviceLatencyTargetSec=\"/ 50ms\""

          [Install]
          RequiredBy=kubelet-dependencies.target
        enabled: true
        name: swap-provision.service
```

最悪のシナリオに備えて十分なスワップ領域を確保するには、オーバーコミットされた RAM と同量以上のスワップ領域をプロビジョニングしておく必要があります。次の式を使用して、ノードにプロビジョニングするスワップ領域の量を計算します。

```
NODE_SWAP_SPACE = NODE_RAM * (MEMORY_OVER_COMMIT_PERCENT / 100% -
1)
```

以下に例を示します。

```
NODE_SWAP_SPACE = 16 GB * (150% / 100% - 1)
                = 16 GB * (1.5 - 1)
                = 16 GB * (0.5)
                =  8 GB
```

5. アラートルールを次のようにデプロイします。

```
apiVersion: monitoring.openshift.io/v1
kind: AlertingRule
metadata:
  name: wasp-alerts
  namespace: openshift-monitoring
spec:
  groups:
  - name: wasp.rules
    rules:
    - alert: NodeSwapping
      annotations:
        description: Node {{ $labels.instance }} is swapping at a rate of {{ printf "%.2f" $value }}
MB/s
        runbook_url: https://github.com/openshift-virtualization/wasp-
agent/tree/main/runbooks/alerts/NodeSwapping.md
        summary: A node is swapping memory pages
      expr: |
        # In MB/s
        irate(node_memory_SwapFree_bytes{job="node-exporter"}[5m]) / 1024^2 > 0
      for: 1m
      labels:
        severity: critical
```

6. OpenShift Container Platform Web コンソールを使用するか、次の例のとおり HyperConverged カスタムリソース (CR) ファイルを編集して、メモリーオーバーコミットを使用するように OpenShift Virtualization 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
  namespace: openshift-cnv
spec:
  higherWorkloadDensity:
    memoryOvercommitPercentage: 150
```

7. 次のコマンドを入力して、クラスター内のコンピュートノードにすべての設定を適用します。

```
$ oc patch --type=merge \
  -f <../manifests/hco-set-memory-overcommit.yaml> \
  --patch-file <../manifests/hco-set-memory-overcommit.yaml>
```

注記

すべての設定を適用すると、すべての **MachineConfigPool** ロールアウトの完了後にのみスワップ機能が完全に利用可能になります。

検証

1. **wasp-agent** のデプロイメントを確認するには、次のコマンドを実行します。

```
$  oc rollout status ds wasp-agent -n wasp
```

デプロイメントが成功すると、次のメッセージが表示されます。

```
daemon set "wasp-agent" successfully rolled out
```

2. スワップが正しくプロビジョニングされていることを確認するには、次の手順を実行します。
   a. 以下のコマンドを実行します。

```
$ oc get nodes -l node-role.kubernetes.io/worker
```

   b. 提供されたリストからノードを選択し、次のコマンドを実行します。

```
$ oc debug node/<selected-node> -- free -m
```

スワップが正しくプロビジョニングされている場合、次のようにゼロより大きい値が表示されます。

	total	used	free	shared	buff/cache	available
Mem:	31846	23155	1044	6014	14483	8690
Swap:	8191	2337	5854			

3. 次のコマンドを実行して、OpenShift Virtualization のメモリーオーバーコミットメント設定を確認します。

```
$ oc get -n openshift-cnv HyperConverged kubevirt-hyperconverged -o jsonpath="
{.spec.higherWorkloadDensity.memoryOvercommitPercentage}"
150
```

返される値 (たとえば **150**) は、以前に設定した値と一致する必要があります。

第6章 更新

6.1. 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。

6.1.1. RHEL 9 上の OpenShift Virtualization

OpenShift Virtualization 4.16 は、Red Hat Enterprise Linux (RHEL) 9 をベースにしています。標準の OpenShift Virtualization 更新手順に従って、RHEL 8 をベースとするバージョンから OpenShift Virtualization 4.16 に更新できます。追加の手順は必要ありません。

以前のバージョンと同様に、実行中のワークロードを中断することなく更新を実行できます。OpenShift Virtualization 4.16 では、RHEL 8 ノードから RHEL 9 ノードへのライブマイグレーションがサポートされています。

6.1.1.1. RHEL 9 マシンタイプ

OpenShift Virtualization に含まれるすべての仮想マシンテンプレートは、デフォルトで RHEL 9 マシンタイプ **machineType: pc-q35-rhel9.<y>.0** を使用するようになりました。この場合の **<y>** は RHEL 9 の最新のマイナーバージョンに対応する 1 桁の数字です。たとえば、RHEL 9.2 の場合は **pc-q35-rhel9.2.0** の値が使用されます。

OpenShift Virtualization を更新しても、既存仮想マシンの **machineType** 値は変更されません。これらの仮想マシンは、引き続き更新前と同様に機能します。RHEL 9 での改良を反映するために、オプションで仮想マシンのマシンタイプを変更できます。

重要

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

6.1.2. OpenShift Virtualization の更新について

- Operator Lifecycle Manager(OLM) は OpenShift Virtualization Operator のライフサイクルを管理します。OpenShift Container Platform のインストール時にデプロイされる Marketplace Operator により、クラスターで外部 Operator が利用できるようになります。
- OLM は、OpenShift Virtualization の z-stream およびマイナーバージョンの更新を提供します。OpenShift Container Platform を次のマイナーバージョンに更新すると、マイナーバージョンの更新が利用可能になります。OpenShift Container Platform を最初に更新しない限り、OpenShift Virtualization を次のマイナーバージョンに更新できません。
- OpenShift Virtualization サブスクリプションは、**stable** という名前の単一の更新チャネルを使用します。**stable** チャネルでは、OpenShift Virtualization および OpenShift Container Platform バージョンとの互換性が確保されます。
- サブスクリプションの承認ストラテジーが **Automatic** に設定されている場合に、更新プロセスは、Operator の新規バージョンが **stable** チャネルで利用可能になるとすぐに開始します。サポート可能な環境を確保するために、**自動** 承認ストラテジーを使用することを強く推奨します。OpenShift Virtualization の各マイナーバージョンは、対応する OpenShift Container Platform バージョンを実行する場合にのみサポートされます。たとえば、OpenShift Virtualization 4.16 は OpenShift Container Platform 4.16 で実行する必要があります。

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

- 更新の完了までにかかる時間は、ネットワーク接続によって異なります。ほとんどの自動更新は 15 分以内に完了します。

- OpenShift Virtualization を更新しても、ネットワーク接続が中断されることはありません。

- データボリュームおよびその関連付けられた永続ボリューム要求は更新時に保持されます。

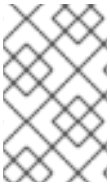
重要

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

回避策として、仮想マシンを再設定し、クラスターの更新時にそれらの電源を自動的にオフになるようにできます。**evictionStrategy** フィールドを **None** に、**runStrategy** フィールドを **Always** に設定します。

6.1.2.1. 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: Always** に設定された **VirtualMachine** オブジェクトによって制御される場合、新規の VMI は、更新されたコンポーネントを使用して新規 Pod に作成されます。

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。

6.1.2.2. EUS から EUS への更新について

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

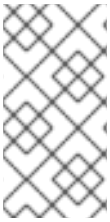
ソース EUS バージョンから次の奇数番号のマイナーバージョンに更新した後、更新パス上にあるそのマイナーバージョンのすべての z-stream リリースに OpenShift Virtualization を順次更新する必要があります。適用可能な最新の z-stream バージョンにアップグレードしたら、OpenShift Container Platform をターゲットの EUS マイナーバージョンに更新できます。

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

6.1.2.2.1. 更新の準備中

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

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



注記

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

EUS から EUS への更新の実行 の詳細を参照してください。

6.1.3. EUS から EUS への更新中のワークロード更新の防止

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

前提条件

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

手順

1. 次のコマンドを実行して、**workloadUpdateMethods** 設定を記録します。

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

2. 次のコマンドを実行して、すべてのワークロード更新方法をオフにします。

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

出力例

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

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

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

例6.1 出力例

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

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

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

```
$ oc adm upgrade
```

検証

- 次のコマンドを実行して、現在のバージョンを確認します。

```
$ oc get clusterversion
```

注記

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

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

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

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

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

出力例

```
[
  {
    "name": "operator",
    "version": "4.16.2"
  }
]
```

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

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

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

```
$ oc get clusterversion
```

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

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

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

14. 次のコマンドを使用して、手順 1 から記録した **workloadUpdateMethods** 設定を復元します。

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

出力例

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

検証

- 次のコマンドを実行して、VM 移行のステータスを確認します。

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

次のステップ

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

6.1.4. ワークロードの更新方法の設定

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

前提条件

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

注記

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

手順

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

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

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

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

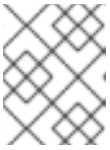
1. ワークロードの自動更新を実行するのに使用できるメソッド。設定可能な値は **LiveMigrate** および **Evict** です。上記の例のように両方のオプションを有効にした場合に、ライブマイグレーションをサポートする VMI には **LiveMigrate** を、ライブマイグレーションをサポートしない VMI には **Evict** を、更新に使用します。ワークロードの自動更新を無効にするには、**workloadUpdateStrategy**スタンザを削除するか、**workloadUpdateMethods: []** を設定して配列を空のままにします。

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

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

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

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



注記

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

3. 変更を適用するには、エディターを保存し、終了します。

6.1.5. 保留中の Operator 更新の承認

6.1.5.1. 保留中の Operator 更新の手動による承認

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

前提条件

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

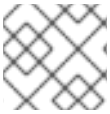
手順

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

6.1.6. 更新ステータスの監視

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

OpenShift Virtualization Operator のアップグレードのステータスをモニターするには、クラスターサービスバージョン (CSV) **PHASE** を監視します。Web コンソールを使用するか、ここに提供されているコマンドを実行して CSV の状態をモニターすることもできます。



注記

PHASE および状態の値は利用可能な情報に基づく近似値になります。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにログインすること。
- 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 hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  -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
```

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

CLI を使用して、以前のワークロードのリストを表示できます。

注記

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

手順

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

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

注記

ワークロードの更新を設定して、VMI が自動的に更新されるようにします。

6.1.7. 関連情報

- EUS から EUS への更新の実行
- Operator について
- Operator Lifecycle Manager の概念およびリソース

- Cluster Service Version (CSV)
- ライブマイグレーションについて
- エビクション戦略の設定
- 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7章 仮想マシン

7.1. RED HAT イメージからの仮想マシンの作成

7.1.1. Red Hat イメージからの仮想マシン作成の概要

Red Hat イメージは ゴールデンイメージ です。これらは、安全なレジストリー内のコンテナーディスクとして公開されます。Containerized Data Importer (CDI) は、コンテナーディスクをポーリングしてクラスターにインポートし、スナップショットまたは永続ボリュームクレーム (PVC) として **openshift-virtualization-os-images** プロジェクトに保存します。

Red Hat イメージは自動的に更新されます。これらのイメージの自動更新を無効にして再度有効にすることができます。Red Hat ブートソースの更新の管理 を参照してください。

クラスター管理者は、OpenShift Virtualization Web コンソール で Red Hat Enterprise Linux (RHEL) 仮想マシンの自動サブスクリプションを有効にできるようになりました。

次のいずれかの方法を使用して、Red Hat が提供するオペレーティングシステムイメージから仮想マシンを作成できます。

- Web コンソールを使用して、テンプレートから仮想マシンを作成します。
- Web コンソールを使用して、インスタンスタイプから仮想マシンを作成します。
- コマンドラインを使用して、**VirtualMachine** マニフェストから仮想マシンを作成します。

重要

デフォルトの **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。

7.1.1.1. ゴールデンイメージについて

ゴールデンイメージは、新しい仮想マシンをデプロイメントするためのリソースとして使用できる、仮想マシンの事前設定されたスナップショットです。たとえば、ゴールデンイメージを使用すると、同じシステム環境を一貫してプロビジョニングし、システムをより迅速かつ効率的にデプロイメントできます。

7.1.1.1.1. ゴールデンイメージはどのように機能しますか?

ゴールデンイメージは、リファレンスマシンまたは仮想マシンにオペレーティングシステムとソフトウェアアプリケーションをインストールして設定することによって作成されます。これには、システムのセットアップ、必要なドライバーのインストール、パッチと更新の適用、特定のオプションと環境設定の設定が含まれます。

ゴールデンイメージは、作成後、複数のクラスターに複製してデプロイできるテンプレートまたはイメージファイルとして保存されます。ゴールデンイメージは、メンテナーによって定期的に更新されて、必要なソフトウェア更新とパッチが組み込まれるため、イメージが最新かつ安全な状態に保たれ、新しく作成された仮想マシンはこの更新されたイメージに基づいています。

7.1.1.1.2. Red Hat のゴールデンイメージの実装

Red Hat は、Red Hat Enterprise Linux (RHEL) のバージョンのレジストリー内のコンテナーディスクと

してゴールデンイメージを公開します。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イメージです。公開されたイメージは、OpenShift Virtualization のインストール後に、接続されたクラスターで自動的に使用できるようになります。イメージがクラスター内で使用可能になると、仮想マシンの作成に使用できるようになります。

7.1.1.2. 仮想マシンブートソースについて

仮想マシンは、仮想マシン定義と、データボリュームによってバックアップされる 1 つ以上のディスクで構成されます。仮想マシンテンプレートを使用すると、事前定義された仕様を使用して仮想マシンを作成できます。

すべてのテンプレートにはブートソースが必要です。ブートソースは、設定されたドライバーを含む完全に設定されたディスクイメージです。各テンプレートには、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ています。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。自動更新されるブートソースの場合、永続ボリュームクレーム (PVC) とボリュームスナップショットはクラスターのデフォルトストレージクラスで作成されます。設定後に別のデフォルトストレージクラスを選択した場合は、以前のデフォルトストレージクラスで設定されたクラスター namespace 内の既存のブートソースを削除する必要があります。

7.1.2. インスタンスタイプからの仮想マシンの作成

OpenShift Container Platform Web コンソールまたは CLI を使用して仮想マシンを作成する場合でも、インスタンスタイプを使用することで仮想マシン (仮想マシン) の作成を簡素化できます。

7.1.2.1. インスタンスタイプについて

インスタンスタイプは、新しい仮想マシンに適用するリソースと特性を定義できる再利用可能なオブジェクトです。カスタムインスタンスタイプを定義したり、OpenShift Virtualization のインストール時に含まれるさまざまなインスタンスタイプを使用したりできます。

新しいインスタンスタイプを作成するには、まず手動で、または **virtctl** CLI ツールを使用してマニフェストを作成する必要があります。次に、マニフェストをクラスターに適用してインスタンスタイプオブジェクトを作成します。

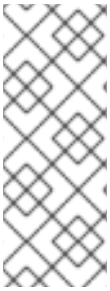
OpenShift Virtualization は、インスタンスタイプを設定するための 2 つの CRD を提供します。

- namespace 付きジェクト: **VirtualMachineInstancetype**
- クラスター全体のオブジェクト: **VirtualMachineClusterInstancetype**

これらのオブジェクトは同じ **VirtualMachineInstancetypeSpec** を使用します。

7.1.2.1.1. 必須の属性

インスタンスタイプを設定するときは、**cpu** および **memory** 属性を定義する必要があります。その他の属性はオプションです。



注記

インスタンスタイプから仮想マシンを作成する場合は、インスタンスタイプで定義されているパラメーターをオーバーライドすることはできません。

インスタンスタイプには定義された CPU およびメモリー属性が必要であるため、OpenShift Virtualization は、インスタンスタイプから仮想マシンを作成するときに、これらのリソースに対する追加の要求を常に拒否します。

インスタンスタイプマニフェストを手動で作成できます。以下に例を示します。

必須フィールドを含む YAML ファイルの例

```
apiVersion: instancetype.kubevirt.io/v1beta1
kind: VirtualMachineInstancetype
metadata:
  name: example-instancetype
spec:
  cpu:
    guest: 1 (1)
  memory:
    guest: 128Mi (2)
```

(1) 必須。ゲストに割り当てる vCPU の数を指定します。

(2) 必須。ゲストに割り当てるメモリーの量を指定します。

virtctl CLI ユーティリティーを使用してインスタンスタイプマニフェストを作成できます。以下に例を示します。

必須フィールドを含む virtctl コマンドの例

```
$ virtctl create instancetype --cpu 2 --memory 256Mi
```

ここでは、以下のようになります。

--cpu <value>
ゲストに割り当てる vCPU の数を指定します。必須。

--memory <value>
ゲストに割り当てるメモリーの量を指定します。必須。

ヒント

次のコマンドを実行すると、新しいマニフェストからオブジェクトをすぐに作成できます。

```
$ virtctl create instancetype --cpu 2 --memory 256Mi | oc apply -f -
```

7.1.2.1.2. オプション属性

必須の **cpu** および **memory** 属性に加えて、**VirtualMachineInstancetypeSpec** に次のオプション属性を含めることができます。

annotations

仮想マシンに適用するアノテーションをリスト表示します。

gpus

パススルー用の vGPU をリスト表示します。

hostDevices

パススルー用のホストデバイスをリスト表示します。

ioThreadsPolicy

専用ディスクアクセスを管理するための IO スレッドポリシーを定義します。

launchSecurity

セキュア暗号化仮想化 (SEV) を設定します。

nodeSelector

この仮想マシンがスケジュールされているノードを制御するためのノードセレクターを指定します。

schedulerName

デフォルトのスケジューラーの代わりに、この仮想マシンに使用するカスタムスケジューラーを定義します。

7.1.2.2. 定義済みのインスタンスタイプ

OpenShift Virtualization には、**common-instancetypes** と呼ばれる事前定義されたインスタンスタイプのセットが含まれています。特定のワークロードに特化したものもあれば、ワークロードに依存しないものもあります。

これらのインスタンスタイプリソースは、シリーズ、バージョン、サイズに応じて名前が付けられます。サイズの値は **.** 区切り文字に続き、範囲は **nano** から **8xlarge** です。

表7.1 common-instancetypes シリーズの比較

ユースケース	シリーズ	特徴	vCPU とメモリーの比率	リソースの例
Universal	U	• バースト可能な CPU パフォーマンス	1:4	**u1.medium** • 仮想 CPU 1 個 • 4Gi メモリー
過剰コミットメント	O	• 過剰にコミットされたメモリー • バースト可能な CPU パフォーマンス	1:4	**o1.small** • 1 vCPU • 2Gi メモリー

ユースケース	シリーズ	特徴	vCPU とメモリーの比率	リソースの例
コンピュート専用	CX	• hugepage • 専用 CPU • 分離されたエミュレータースレッド • vNUMA	1:2	**cx1.2xlarge** • 仮想 CPU 8 個 • 16Gi メモリー
NVIDIA GPU	GN	• NVIDIA GPU Operator が提供する GPU を使用する仮想マシンの場合 • 定義済みの GPU がある • バースト可能な CPU パフォーマンス	1:4	**gn1.8xlarge** • 仮想 CPU 32 個 • 128Gi メモリー
メモリー集約型	M	• hugepage • バースト可能な CPU パフォーマンス	1:8	**m1.large** • 仮想 CPU 2 個 • 16Gi メモリー
ネットワーク集約型	N	• hugepage • 専用 CPU • 分離されたエミュレータースレッド • DPDK ワークロードを実行できるノードが必要	1:2	**n1.medium** • 仮想 CPU 4 個 • 4Gi メモリー

7.1.2.3. virtctl ツールを使用したマニフェストの作成

virtctl CLI ユーティリティーを使用すると、仮想マシン、仮想マシンインスタンスタイプ、および仮想マシン設定のマニフェストの作成を簡素化できます。詳細は、仮想マシンマニフェスト作成コマンド を参照してください。

VirtualMachine マニフェストがある場合は、コマンドライン から仮想マシンを作成できます。

7.1.2.4. Web コンソールを使用して、インスタンスタイプから仮想マシンを作成します。

OpenShift Container Platform Web コンソールを使用して、インスタンスタイプから仮想マシンを作成できます。Web コンソールを使用して、既存のスナップショットをコピーするか仮想マシンを複製して、仮想マシンを作成することもできます。

使用可能な起動可能なボリュームのリストから仮想マシンを作成できます。Linux ベースまたは Windows ベースのボリュームをリストに追加できます。

手順

1. Web コンソールで、**Virtualization → Catalog** に移動します。
InstanceTypes タブがデフォルトで開きます。
2. 次のオプションのいずれかを選択します。
   - リストから適切な起動可能なボリュームを選択します。リストが切り捨てられている場合は、**Show all** ボタンをクリックしてリスト全体を表示します。

     注記

     ブート可能ボリュームテーブルには、**openshift-virtualization-os-images** namespace 内の **instancetype.kubevirt.io/default-preference** ラベルを持つボリュームのみリストされます。

     - オプション: 星アイコンをクリックして、ブート可能ボリュームをお気に入りとして指定します。星付きのブート可能ボリュームは、ボリュームリストの最初に表示されます。
   - 新しいボリュームをアップロードするか、既存の永続ボリューム要求 (PVC)、ボリュームスナップショット、または **containerDisk** ボリュームを使用するには **Add volume** をクリックします。**Save** をクリックします。
     クラスターで使用できないオペレーティングシステムのロゴは、リストの下部に表示されます。**Add volume** リンクをクリックすると、必要なオペレーティングシステムのボリュームを追加できます。

     さらに、**Create a Windows boot source**クイックスタートへのリンクもあります。**Select volume to boot from** の横にある疑問符アイコンの上にポインターを置くと、ポップオーバーに同じリンクが表示されます。

     環境をインストールした直後、または環境が切断された直後は、起動元のボリュームのリストは空になります。その場合、Windows、RHEL、Linux の 3 つのオペレーティングシステムのロゴが表示されます。**Add volume** ボタンをクリックすると、要件を満たす新しいボリュームを追加できます。
3. インスタンスタイプのタイルをクリックし、ワークロードに適したリソースサイズを選択します。
4. オプション: 起動元のボリュームに適用される仮想マシンの詳細 (仮想マシンの名前を含む) を選択します。
   - Linux ベースのボリュームの場合は、次の手順に従って SSH を設定します。

a. 公開 SSH キーをまだプロジェクトに追加していない場合は、**VirtualMachine details** セクションの **Authorized SSH key** の横にある編集アイコンをクリックします。

b. 以下のオプションのいずれかを選択します。

- **Use existing**: シークレットリストからシークレットを選択します。
- **Add new**: 以下の手順に従ってください。
  i. 公開 SSH キーファイルを参照するか、ファイルをキーフィールドに貼り付けます。
  ii. シークレット名を入力します。
  iii. オプション: **Automatically apply this key to any new VirtualMachine you create in this project** を選択します。

c. **Save** をクリックします。

- Windows ボリュームの場合は、次のいずれかの手順に従って sysprep オプションを設定します。
  - Windows ボリュームに sysprep オプションをまだ追加していない場合は、次の手順に従います。
    i. **VirtualMachine details** セクションの **Sysprep** の横にある編集アイコンをクリックします。
    ii. **Autoattend.xml** アンサーファイルを追加します。
    iii. **Unattend.xml** アンサーファイルを追加します。
    iv. **Save** をクリックします。
  - Windows ボリュームに既存の sysprep オプションを使用する場合は、次の手順に従います。
    i. **既存の sysprep を添付** をクリックします。
    ii. 既存の sysprep **Unattend.xml** アンサーファイルの名前を入力します。
    iii. **Save** をクリックします。

5. オプション: Windows 仮想マシンを作成する場合は、Windows ドライバーディスクをマウントできます。
   a. **Customize VirtualMachine** ボタンをクリックします。
   b. **VirtualMachine details** ページで、**Storage** をクリックします。
   c. **Mount Windows drivers disk** チェックボックスを選択します。
6. オプション: **View YAML & CLI** をクリックして YAML ファイルを表示します。**CLI** をクリックして CLI コマンドを表示します。YAML ファイルの内容または CLI コマンドをダウンロードまたはコピーすることもできます。
7. **Create VirtualMachine** をクリックします。

仮想マシンの作成後、**VirtualMachine details** ページでステータスを監視できます。

7.1.3. テンプレートからの仮想マシンの作成

OpenShift Container Platform Web コンソールを使用して、Red Hat テンプレートから仮想マシン (VM) を作成できます。

7.1.3.1. 仮想マシンテンプレートについて

ブートソース

利用可能なブートソースを持つテンプレートを使用すると、仮想マシンの作成を効率化できます。
ブートソースを含むテンプレートには、カスタムラベルがない場合、**Available boot source** ラベルが付けられます。
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。カスタムイメージからの仮想マシンの作成 を参照してください。

カスタマイズ

仮想マシンを起動する前に、ディスクソースと仮想マシンパラメーターをカスタマイズできます。

ディスクソース設定の詳細は、ストレージボリュームタイプ と ストレージフィールド を参照してください。

注記

すべてのラベルとアノテーションとともに仮想マシンテンプレートをコピーすると、新しいバージョンの Scheduling、Scale、and Performance (SSP) Operator がデプロイされたときに、そのバージョンのテンプレートが非推奨に指定されます。この指定は削除できます。Web コンソールを使用した仮想マシンテンプレートのカスタマイズ 参照してください。

シングルノード OpenShift

ストレージの動作の違いにより、一部のテンプレートはシングルノード OpenShift と互換性がありません。互換性を確保するには、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テンプレートまたは仮想マシンに **evictionStrategy** フィールドを設定しないでください。

7.1.3.2. テンプレートから仮想マシンを作成する

OpenShift Container Platform Web コンソールを使用して、使用可能なブートソースを持つテンプレートから仮想マシンを作成できます。

オプション: 仮想マシンを起動する前に、データソース、cloud-init、SSH キーなどのテンプレートまたは仮想マシンパラメーターをカスタマイズできます。

手順

1. Web コンソールで **Virtualization → Catalog** に移動します。
2. **利用可能なブートソース** をクリックして、テンプレートをブートソースでフィルタリングします。
   カタログにはデフォルトのテンプレートが表示されます。**All Items** をクリックして、フィルターに使用できるすべてのテンプレートを表示します。
3. テンプレートタイルをクリックして詳細を表示します。

4. オプション: Windows テンプレートを使用している場合は、**Mount Windows drivers disk** チェックボックスを選択することで、Windows ドライバーディスクをマウントできます。

5. テンプレートまたは仮想マシンパラメーターをカスタマイズする必要がない場合は、**Quick create VirtualMachine** をクリックして、テンプレートから仮想マシンを作成します。
テンプレートまたは仮想マシンパラメーターをカスタマイズする必要がある場合は、次の手順を実行します。

   a. **Customize VirtualMachine** をクリックします。

   b. **Storage** または **Optional parameters** をデプロイメントして、データソース設定を編集します。

   c. **VirtualMachine パラメーターのカスタマイズ** をクリックします。
Customize and create VirtualMachine ペインには、**Overview**、**YAML**、**Scheduling**、**Environment**、**Network interfaces**、**Disks**、**Scripts**、および **Metadata** タブが表示されます。

   d. 仮想マシンの起動前に設定する必要があるパラメーター (cloud-init や静的 SSH キーなど) を編集します。

   e. **Create VirtualMachine** をクリックします。
VirtualMachine details ページには、プロビジョニングステータスが表示されます。

7.1.3.2.1. ストレージボリュームタイプ

表7.2 ストレージボリュームタイプ

タイプ	説明
ephemeral	ネットワークボリュームを読み取り専用のバッキングストアとして使用するローカルの copy-on-write (COW) イメージ。バッキングボリュームは **PersistentVolumeClaim** である必要があります。一時イメージは仮想マシンの起動時に作成され、すべての書き込みをローカルに保存します。一時イメージは、仮想マシンの停止、再起動または削除時に破棄されます。バッキングボリューム (PVC) はいずれの方法でも変更されません。
persistentVolumeClaim	利用可能な PV を仮想マシンに割り当てます。PV の割り当てにより、仮想マシンデータのセッション間での永続化が可能になります。 CDI を使用して既存の仮想マシンディスクを PVC にインポートし、PVC を仮想マシンインスタンスに割り当てる方法は、既存の仮想マシンを OpenShift Container Platform にインポートするための推奨される方法です。ディスクを PVC 内で使用できるようにするためのいくつかの要件があります。
dataVolume	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。

タイプ	説明
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。
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。

7.1.3.2.2. ストレージフィールド

フィールド	説明
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。

フィールド	説明
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)。
タイプ	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 URL**、および **Clone existing PVC** ディスクで利用できます。

これらのパラメーターを指定しない場合、システムはデフォルトのストレージプロファイル値を使用します。

パラメーター	オプション	パラメーターの説明
ボリュームモード	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	ReadWriteOnce (RWO)	ボリュームはシングルノードで読み取り/書き込みとしてマウントできます。

パラメーター	オプション	パラメーターの説明
	ReadWriteMany (RWX)	ボリュームは、一度に多くのノードで読み取り/書き込みとしてマウントできます。 **注記** このモードはライブマイグレーションに必要です。

7.1.3.2.3. Web コンソールを使用した仮想マシンテンプレートのカスタマイズ

仮想マシンを起動する前に、データソース、cloud-init、SSH キーなど、仮想マシンまたはテンプレートのパラメーターを変更することで、既存の仮想マシン (VM) テンプレートをカスタマイズできます。テンプレートをコピーし、すべてのラベルとアノテーションを含めてテンプレートをカスタマイズすると、新しいバージョンの Scheduling、Scale、and Performance (SSP) Operator がデプロイされたときに、カスタマイズしたテンプレートが非推奨に指定されます。

この非推奨の指定は、カスタマイズしたテンプレートから削除できます。

手順

1. Web コンソールで **Virtualization → Templates** に移動します。
2. 仮想マシンテンプレートのリストから、非推奨とマークされているテンプレートをクリックします。
3. **Labels** の近くにある鉛筆アイコンの横にある **Edit** をクリックします。
4. 次の 2 つのラベルを削除します。
   - **template.kubevirt.io/type: "base"**
   - **template.kubevirt.io/version: "version"**
5. **Save** をクリックします。
6. 既存の **Annotations** の数の横にある鉛筆アイコンをクリックします。
7. 次のアノテーションを削除します。
   - **template.kubevirt.io/deprecated**
8. **Save** をクリックします。

7.1.4. コマンドラインからの仮想マシンの作成

VirtualMachine マニフェストを編集または作成することで、コマンドラインから仮想マシン (VM) を作成できます。仮想マシンマニフェストで インスタンスタイプ を使用すると、仮想マシン設定を簡素化できます。

注記

Web コンソールを使用してインスタンスタイプから仮想マシンを作成する こともできます。

7.1.4.1. virtctl ツールを使用したマニフェストの作成

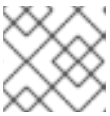
virtctl CLI ユーティリティーを使用すると、仮想マシン、仮想マシンインスタンスタイプ、および仮想マシン設定のマニフェストの作成を簡素化できます。詳細は、仮想マシンマニフェスト作成コマンド を参照してください。

7.1.4.2. VirtualMachine マニフェストからの仮想マシンの作成

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。次の例では、Red Hat Enterprise Linux (RHEL) 仮想マシンを設定します。



注記

このサンプルマニフェストでは、仮想マシン認証は設定されません。

RHEL 仮想マシンのマニフェストの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
 name: rhel-9-minimal
spec:
 dataVolumeTemplates:
  - metadata:
      name: rhel-9-minimal-volume
    spec:
      sourceRef:
        kind: DataSource
        name: rhel9 (1)
        namespace: openshift-virtualization-os-images (2)
      storage: {}
 instancetype:
  name: u1.medium (3)
 preference:
  name: rhel.9 (4)
 running: true
 template:
  spec:
    domain:
      devices: {}
    volumes:
      - dataVolume:
          name: rhel-9-minimal-volume
        name: rootdisk
```

1. **rhel9** ゴールデンイメージは、RHEL 9 をゲストオペレーティングシステムとしてインストールするために使用されます。
2. ゴールデンイメージは、**openshift-virtualization-os-images** namespace に保存されます。
3. **u1.medium** インスタンスタイプは、仮想マシンに 1 つの vCPU と 4Gi のメモリーを要求します。これらのリソース値は仮想マシン内で上書きできません。
4. **rhel.9** 設定は、RHEL 9 ゲストオペレーティングシステムをサポートする追加属性を指定します。

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: 仮想マシンを起動します。

```
$ virtctl start <vm_name> -n <namespace>
```

次のステップ

- 仮想マシンへの SSH アクセスの設定

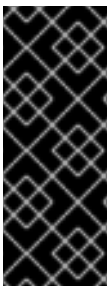
7.2. カスタムイメージからの仮想マシンの作成

7.2.1. カスタムイメージからの仮想マシン作成の概要

次のいずれかの方法を使用して、カスタムオペレーティングシステムイメージから仮想マシンを作成できます。

- レジストリーからイメージをコンテナーディスクとしてインポートします。
  オプション: コンテナーディスクの自動更新を有効にすることができます。詳細は、ブートソースの自動更新の管理 を参照してください。
- Web ページからイメージをインポートします。
- ローカルマシンからイメージをアップロードします。
- イメージを含む永続ボリューム要求 (PVC) のクローンを作成します。

Containerized Data Importer (CDI) は、データボリュームを使用してイメージを PVC にインポートします。OpenShift Container Platform Web コンソールまたはコマンドラインを使用して、PVC を仮想マシンに追加します。



重要

Red Hat が提供していないオペレーティングシステムイメージから作成された仮想マシンには、QEMU ゲストエージェント をインストールする必要があります。

Windows 仮想マシンにも VirtIO ドライバー をインストールする必要があります。

QEMU ゲストエージェントは Red Hat イメージに含まれています。

7.2.2. コンテナーディスクを使用した仮想マシンの作成

オペレーティングシステムイメージから構築されたコンテナーディスクを使用して、仮想マシンを作成できます。

コンテナーディスクの自動更新を有効にすることができます。詳細は、ブートソースの自動更新の管理を参照してください。

重要

コンテナーディスクが大きい場合は、I/O トラフィックが増加し、ワーカーノードが使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、次のタスクを実行できます。

- **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

次の手順を実行して、コンテナーディスクから仮想マシンを作成します。

1. オペレーティングシステムイメージをコンテナーディスクに構築し、それをコンテナーレジストリーにアップロードします。
2. コンテナーレジストリーに TLS がない場合は、レジストリーの TLS を無効にするように環境を設定します。
3. Web コンソール または コマンドライン を使用して、コンテナーディスクをディスクソースとして使用する仮想マシンを作成します。

重要

Red Hat が提供していないオペレーティングシステムイメージから作成された仮想マシンには、QEMU ゲストエージェント をインストールする必要があります。

7.2.2.1. コンテナーディスクの構築とアップロード

仮想マシンイメージをコンテナーディスクに構築し、レジストリーにアップロードできます。

コンテナーディスクのサイズは、コンテナーディスクがホストされてい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る必要があります。
- QCOW2 または RAW イメージファイルが必要です。

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 **107** を持つ QEMU によって所有され、コンテナー内の **/disk/** ディレクトリーに配置さ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 **<vm_image>** は、QCOW2 または RAW 形式のイメージです。リモートイメージを使用する場合は、**<vm_image>.qcow2** を完全な 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
```

7.2.2.2. コンテナーレジストリーの TLS を無効にする

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

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

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

2. セキュアでないレジストリーのリストを **spec.storageImport.insecureRegistries** フィールドに追加します。

HyperConverged カスタムリソースの例

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

```
insecureRegistries: 1
  - "private-registry-example-1:5000"
  - "private-registry-example-2:5000"
```

1 このリストのサンプルを、有効なレジストリーホスト名に置き換えます。

7.2.2.3. Web コンソールを使用してコンテナーディスクから仮想マシンを作成する

OpenShift Container Platform Web コンソールを使用してコンテナーレジストリーからコンテナーディスクをインポートすることで、仮想マシンを作成できます。

手順

1. Web コンソールで **Virtualization → Catalog** に移動します。
2. 使用可能なブートソースのないテンプレートタイルをクリックします。
3. **Customize VirtualMachine** をクリックします。
4. **Customize template parameters** ページで、**Storage** を展開し、**Disk source** リストから **Registry (creates PVC)** を選択します。
5. コンテナーイメージの URL を入力します。例: **https://mirror.arizona.edu/fedora/linux/releases/38/Cloud/x86_64/images/Fedora-Cloud-Base-38-1.6.x86_64.qcow2**
6. ディスクサイズを設定します。
7. **Next** をクリックします。
8. **Create VirtualMachine** をクリックします。

7.2.2.4. コマンドラインを使用したコンテナーディスクからの仮想マシンの作成

コマンドラインを使用して、コンテナーディスクから仮想マシンを作成できます。

仮想マシンが作成されると、コンテナーディスクを含むデータボリュームが永続ストレージにインポートされます。

前提条件

- コンテナーディスクを含むコンテナーレジストリーへのアクセス認証情報が必要です。

手順

1. コンテナーレジストリーで認証が必要な場合は、認証情報を指定して **Secret** マニフェストを作成し、**data-source-secret.yaml** ファイルとして保存します。

```
apiVersion: v1
kind: Secret
metadata:
  name: data-source-secret
  labels:
    app: containerized-data-importer
```

```
type: Opaque
data:
  accessKeyId: "" 1
  secretKey:   "" 2
```

1 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。

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

```
$ oc apply -f data-source-secret.yaml
```

3. 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、仮想マシンと同じ namespace に config map を作成します。

```
$ oc create configmap tls-certs 1
  --from-file=</path/to/file/ca.pem> 2
```

1 config map 名を指定します。

2 CA 証明書へのパスを指定します。

4. **VirtualMachine** マニフェストを編集し、**vm-fedora-datavolume.yaml** ファイルとして保存します。

```
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 3
        storageClassName: <storage_class> 4
      source:
        registry:
          url: "docker://kubevirt/fedora-cloud-container-disk-demo:latest" 5
          secretRef: data-source-secret 6
          certConfigMap: tls-certs 7
    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. データボリュームに要求されるストレージのサイズを指定します。
4. オプション: ストレージクラスを指定しない場合は、デフォルトのストレージクラスが適用されます。
5. コンテナーレジストリーの URL を指定します。
6. オプション: コンテナーレジストリーアクセス認証情報のシークレットを作成した場合は、シークレット名を指定します。
7. オプション: CA 証明書 config map を指定します。

5. 次のコマンドを実行して VM を作成します。

```
$ oc create -f vm-fedora-datavolume.yaml
```

oc create コマンドは、データボリュームと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 からコンテナーディスクをダウンロードし、プロビジョニングされた永続ボリュームに保存します。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
```

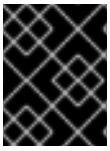
1 **VirtualMachine** マニフェストで定義したデータボリューム名を指定します。

3. プロビジョニングが完了し、シリアルコンソールにアクセスして仮想マシンが起動したことを確認します。

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

7.2.3. Web ページからイメージをインポートして仮想マシンを作成する

Web ページからオペレーティングシステムイメージをインポートすることで、仮想マシンを作成できます。



重要

Red Hat が提供していないオペレーティングシステムイメージから作成された仮想マシンには、QEMU ゲストエージェント をインストールする必要があります。

7.2.3.1. Web コンソールを使用して Web ページ上のイメージから仮想マシンを作成する

OpenShift Container Platform Web コンソールを使用して Web ページからイメージをインポートすることで、仮想マシンを作成できます。

前提条件

- イメージを含む Web ページにアクセスできる。

手順

1. Web コンソールで **Virtualization → Catalog** に移動します。
2. 使用可能なブートソースのないテンプレートタイルをクリックします。
3. **Customize VirtualMachine** をクリックします。
4. **Customize template parameters** ページで、**Storage** を展開し、**Disk source** リストから **URL (creates PVC)** を選択します。
5. イメージの URL を入力します。例: **https://access.redhat.com/downloads/content/69/ver=/rhel---7/7.9/x86_64/product-software**

6. コンテナーイメージの URL を入力します。例: **https://mirror.arizona.edu/fedora/linux/releases/38/Cloud/x86_64/images/Fedora-Cloud-Base-38-1.6.x86_64.qcow2**
7. ディスクサイズを設定します。
8. **Next** をクリックします。
9. **Create VirtualMachine** をクリックします。

7.2.3.2. コマンドラインを使用して Web ページ上のイメージから仮想マシンを作成する

コマンドラインを使用して、Web ページ上のイメージから仮想マシンを作成できます。

仮想マシンが作成されると、イメージを含むデータボリュームが永続ストレージにインポートされます。

前提条件

- イメージを含む Web ページへのアクセス認証情報が必要です。

手順

1. Web ページで認証が必要な場合は、認証情報を指定して **Secret** マニフェストを作成し、**data-source-secret.yaml** ファイルとして保存します。

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

1 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。

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

```
$ oc apply -f data-source-secret.yaml
```

3. 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、仮想マシンと同じ namespace に config map を作成します。

```
$ oc create configmap tls-certs 1
  --from-file=</path/to/file/ca.pem> 2
```

1 config map 名を指定します。

2 CA 証明書へのパスを指定します。

4. **VirtualMachine** マニフェストを編集し、**vm-fedora-datavolume.yaml** ファイルとして保存します。

```
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 3
        storageClassName: <storage_class> 4
      source:
        http:
          url: "https://mirror.arizona.edu/fedora/linux/releases/35/Cloud/x86_64/images/Fedora-Cloud-Base-35-1.2.x86_64.qcow2" 5
        registry:
          url: "docker://kubevirt/fedora-cloud-container-disk-demo:latest" 6
          secretRef: data-source-secret 7
          certConfigMap: tls-certs 8
    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. データボリュームに要求されるストレージのサイズを指定します。
4. オプション: ストレージクラスを指定しない場合は、デフォルトのストレージクラスが適用されます。
5. 6. Web ページの URL を指定します。
7. オプション: Web ページのアクセス認証情報のシークレットを作成した場合は、シークレット名を指定します。
8. オプション: CA 証明書 config map を指定します。

5. 次のコマンドを実行して VM を作成します。

```
$ oc create -f vm-fedora-datavolume.yaml
```

oc create コマンドは、データボリュームと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 からイメージをダウンロードし、プロビジョニングされた永続ボリュームに保存します。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. プロビジョニングが完了し、シリアルコンソールにアクセスして仮想マシンが起動したことを確認します。

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

7.2.4. イメージをアップロードして仮想マシンを作成する

ローカルマシンからオペレーティングシステムイメージをアップロードすることで、仮想マシンを作成できます。

Windows イメージを PVC にアップロードすることで、Windows 仮想マシンを作成できます。次に、仮想マシンの作成時に PVC のクローンを作成します。

重要

Red Hat が提供していないオペレーティングシステムイメージから作成された仮想マシンには、QEMU ゲストエージェント をインストールする必要があります。

Windows 仮想マシンにも VirtIO ドライバー をインストールする必要があります。

7.2.4.1. Web コンソールを使用して、アップロードされたイメージから仮想マシンを作成する

OpenShift Container Platform Web コンソールを使用して、アップロードされたオペレーティングシステムイメージから仮想マシンを作成できます。

前提条件

- **IMG**、**ISO**、または **QCOW2** イメージファイルが必要です。

手順

1. Web コンソールで **Virtualization → Catalog** に移動します。
2. 使用可能なブートソースのないテンプレートタイルをクリックします。
3. **Customize VirtualMachine** をクリックします。
4. **Customize template parameters** ページで、**Storage** を展開し、**Disk source** リストから **Upload (Upload a new file to a PVC)**を選択します。
5. ローカルマシン上のイメージを参照し、ディスクサイズを設定します。
6. **Customize VirtualMachine** をクリックします。
7. **Create VirtualMachine** をクリックします。

7.2.4.2. Windows 仮想マシンの作成

Windows 仮想マシンを作成するには、Windows イメージを永続ボリューム要求 (PVC) にアップロードし、OpenShift Container Platform Web コンソールを使用して仮想マシンを作成するときに PVC のクローンを作成します。

前提条件

- Windows Media Creation Tool を使用して Windows インストール DVD または USB が作成されている。Microsoft ドキュメントの Windows 10 インストールメディアの作成 を参照してください。
- **autounattend.xml**アンサーファイルを作成しました。Microsoft ドキュメントの アンサーファイル (unattend.xml) を参照してください。

手順

1. Windows イメージを新しい PVC としてアップロードします。
   a. Web コンソールで **Storage → PersistentVolumeClaims** に移動します。
   b. **Create PersistentVolumeClaim → With Data upload form** をクリックします。
   c. Windows イメージを参照して選択します。
   d. PVC 名を入力し、ストレージクラスとサイズを選択して、**Upload** をクリックします。Windows イメージは PVC にアップロードされます。
2. アップロードされた PVC のクローンを作成して、新しい仮想マシンを設定します。
   a. **Virtualization → Catalog** に移動します。
   b. Windows テンプレートタイルを選択し、**Customize VirtualMachine** をクリックします。
   c. **Disk source** リストから **Clone (clone PVC)** を選択します。
   d. PVC プロジェクト、Windows イメージ PVC、およびディスクサイズを選択します。
3. アンサーファイルを仮想マシンに適用します。
   a. **VirtualMachine パラメーターのカスタマイズ** をクリックします。
   b. **Scripts** タブの **Sysprep** セクションで、**Edit** をクリックします。
   c. **autounattend.xml** 応答ファイルを参照し、**Save** をクリックします。
4. 仮想マシンの実行戦略を設定します。
   a. 仮想マシンがすぐに起動しないように、**Start this VirtualMachine after creation** をオフにします。
   b. **Create VirtualMachine** をクリックします。
   c. **YAML** タブで、**running:false** を **runStrategy: RerunOnFailure** に置き換え、**Save** をクリックします。
5. オプションメニュー ⋮ をクリックして **Start** を選択します。
   仮想マシンは、**autounattend.xml** アンサーファイルを含む **sysprep** ディスクから起動します。

7.2.4.2.1. Windows 仮想マシンイメージの一般化

Windows オペレーティングシステムイメージを一般化して、イメージを使用して新しい仮想マシンを作成する前に、システム固有の設定データをすべて削除できます。

仮想マシンを一般化する前に、Windows の無人インストール後に**sysprep**ツールが応答ファイルを検出できないことを確認する必要があります。

前提条件

- QEMU ゲストエージェントがインストールされた実行中の Windows 仮想マシン。

手順

1. OpenShift Container Platform コンソールで、**Virtualization → VirtualMachines** をクリックします。
2. Windows 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration → Disks** をクリックします。
4. **sysprep** ディスクの横にある Options メニュー  をクリックし、**Detach** を選択します。
5. **Detach** をクリックします。
6. **sysprep**ツールによる検出を回避するために、**C:\Windows\Panther\unattend.xml**の名前を変更します。
7. 次のコマンドを実行して、**sysprep** プログラムを開始します。

```
%WINDIR%\System32\Sysprep\sysprep.exe /generalize /shutdown /oobe /mode:vm
```

8. **sysprep**ツールが完了すると、Windows 仮想マシンがシャットダウンします。これで、仮想マシンのディスクイメージを Windows 仮想マシンのインストールイメージとして使用できるようになりました。

これで、仮想マシンを特殊化できます。

7.2.4.2.2. Windows 仮想マシンイメージの特殊化

Windows 仮想マシンを特殊化すると、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 PVC プロジェクトと PVC 名を選択します。
5. **VirtualMachine パラメーターのカスタマイズ** 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。

Windows 仮想マシンを作成するための関連情報

- Microsoft、Sysprep (Generalize) a Windows installation
- Microsoft、generalize
- Microsoft、specialize

7.2.4.3. コマンドラインを使用して、アップロードされたイメージから仮想マシンを作成する

virtctl コマンドラインツールを使用して、オペレーティングシステムイメージをアップロードできます。既存のデータボリュームを使用することも、イメージ用に新しいデータボリュームを作成することもできます。

前提条件

- **ISO**、**IMG**、または **QCOW2** オペレーティングシステムイメージファイルがある。
- 最高のパフォーマンスを得るには、virt-sparsify ツールまたは **xz** ユーティリティーまたは **gzip** ユーティリティーを使用してイメージファイルを圧縮しておく。
- **virtctl** がインストールされている。
- 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. **virtctl image-upload** コマンドを実行してイメージをアップロードします。

```
$ 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. 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。

2. 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
```

7.2.5. QEMU ゲストエージェントと VirtIO ドライバーのインストール

QEMU ゲストエージェントは、仮想マシンで実行され、仮想マシン、ユーザー、ファイルシステム、およびセカンダリーネットワークに関する情報をホストに渡すデーモンです。

Red Hat が提供していないオペレーティングシステムイメージから作成された仮想マシンには、QEMU ゲストエージェントをインストールする必要があります。

7.2.5.1. QEMU ゲストエージェントのインストール

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

qemu-guest-agent は広範な使用が可能で、Red Hat Enterprise Linux (RHEL) 仮想マシン (VM) においてデフォルトで使用できます。このエージェントをインストールし、サービスを起動します。

注記

整合性が最も高いオンライン (実行状態) の仮想マシンのスナップショットを作成するには、QEMU ゲストエージェントをインストールします。

QEMU ゲストエージェントは、システムのワークロードに応じて、可能な限り仮想マシンのファイルシステムの休止しようとすることで一貫性のあるスナップショットを取得します。これにより、スナップショットの作成前にインフライトの I/O がディスクに書き込まれるようになります。ゲストエージェントが存在しない場合は、休止はできず、ベストエフォートスナップショットが作成されます。スナップショットの作成条件は、Web コンソールまたは CLI に表示されるスナップショットの指示に反映されます。

手順

1. コンソールまたは SSH を使用して仮想マシンにログインします。
2. 次のコマンドを実行して、QEMU ゲストエージェントをインストールします。

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

3. サービスに永続性があることを確認し、これを起動します。

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

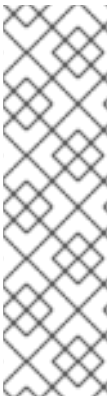
検証

- 次のコマンドを実行して、**AgentConnected** が VM 仕様にリストされていることを確認します。

```
$ oc get vm <vm_name>
```

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

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。ドライバーは、Windows のインストール中または既存の Windows 仮想マシンにインストールできます。



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。

手順

1. Windows Guest Operating System で、**File Explorer** を使用して、**virtio-win** CD ドライブの **guest-agent** ディレクトリーに移動します。
2. **qemu-ga-x86_64.msi** インストーラーを実行します。

検証

1. 次のコマンドを実行して、ネットワークサービスのリストを取得します。

```
$ net start
```

2. 出力に **QEMU Guest Agent** が含まれていることを確認します。

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

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。ドライバーは残りのイメージと同梱されるされるため、個別にダウンロードする必要はありません。

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。VirtIO ドライバーは、Windows のインストール中に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。

表7.3 サポートされているドライバー

ドライバー名	ハードウェア ID	説明
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	ブロックドライバー。**Other devices** グループの **SCSI Controller** としてラベル付け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** としてラベル付け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** としてラベル付け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。

7.2.5.2.1. インストール中に VirtIO コンテナーディスクを Windows 仮想マシンにアタッチする

必要な Windows ドライバーをインストールするには、VirtIO コンテナーディスクを Windows 仮想マシンにアタッチする必要があります。これは、仮想マシンの作成時に実行できます。

手順

1. テンプレートから Windows 仮想マシンを作成する場合は、**Customize VirtualMachine** をクリックします。
2. **Mount Windows drivers disk** を選択します。
3. **Customize VirtualMachine parameters** をクリックします。
4. **Create VirtualMachine** をクリックします。

仮想マシンの作成後、**virtio-win** SATA CD ディスクが仮想マシンにアタッチされます。

7.2.5.2.2. VirtIO コンテナーディスクを既存の Windows 仮想マシンにアタッチする

必要な Windows ドライバーをインストールするには、VirtIO コンテナーディスクを Windows 仮想マシンにアタッチする必要があります。これは既存の仮想マシンに対して実行できます。

手順

1. 既存の Windows 仮想マシンに移動し、**Actions → Stop** をクリックします。
2. **VM Details → Configuration → Disks** に移動し、**Add disk** をクリックします。

3. コンテナーソースから **windows-driver-disk** を追加し、**Type** を **CD-ROM** に設定して、**Interface** を **SATA** に設定します。
4. **Save** をクリックします。
5. 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。

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

仮想マシンに Windows をインストールする際に VirtIO ドライバーをインストールできます。

注記

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

前提条件

- **virtio** ドライバーを含むストレージデバイスを仮想マシンに接続している。

手順

1. Windows オペレーティングシステムでは、**File Explorer** を使用して **virtio-win** CD ドライブに移動します。
2. ドライブをダブルクリックして、仮想マシンに適切なインストーラーを実行します。64 ビット vCPU の場合は、**virtio-win-gt-x64** インストーラーを選択します。32 ビット vCPU はサポート対象外になりました。
3. オプション: インストーラーの **Custom Setup** 手順で、インストールするデバイスドライバーを選択します。デフォルトでは、推奨ドライバーセットが選択されています。
4. インストールが完了したら、**Finish** を選択します。
5. 仮想マシンを再起動します。

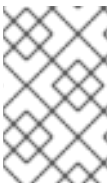
検証

1. PC でシステムディスクを開きます。通常は **(C:)** です。
2. **Program Files → Virtio-Win** に移動します。

Virtio-Win ディレクトリーが存在し、各ドライバーのサブディレクトリーが含まれていればインストールは成功です。

7.2.5.2.4. 既存の Windows 仮想マシン上の SATA CD ドライブから VirtIO ドライバーのインストール

VirtIO ドライバーは、SATA CD ドライブから既存の Windows 仮想マシンにインストールできます。



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。特定のインストール手順は、お使いの Windows バージョンに関するインストールドキュメントを参照してください。

前提条件

- virtio ドライバーを含むストレージデバイスは、SATA CD ドライブとして仮想マシンに接続する必要があります。

手順

1. 仮想マシンを起動し、グラフィカルコンソールに接続します。
2. Windows ユーザーセッションにログインします。
3. **Device Manager** を開き、**Other devices** を拡張して、**Unknown device** をリスト表示します。
   a. **Device Properties** を開いて、不明なデバイスを特定します。
   b. デバイスを右クリックし、**Properties** を選択します。
   c. **Details** タブをクリックし、**Property** リストで **Hardware Ids** を選択します。
   d. **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。

7.2.5.2.5. SATA CD ドライブとして追加されたコンテナーディスクからの VirtIO ドライバーのインストール

Windows 仮想マシンに SATA CD ドライブとして追加するコンテナーディスクから VirtIO ドライバーをインストールできます。

ヒント

コンテナーディスクがクラスター内に存在しない場合、コンテナーディスクは Red Hat レジストリーからダウンロードされるため、Red Hat エコシステムカタログ からの **container-native-virtualization/virtio-win** コンテナーディスクのダウンロードは必須ではありません。ただし、ダウンロードするとインストール時間が短縮されます。

前提条件

- 制限された環境では、Red Hat レジストリー、またはダウンロードされた **container-native-virtualization/virtio-win** コンテナーディスクにアクセスできる必要があります。

手順

1. **VirtualMachine** マニフェストを編集して、**container-native-virtualization/virtio-win** コンテナーディスクを CD ドライブとして追加します。

```
# ...
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** マニフェストで定義された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起動する他の仮想マシン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することができます。ディスクのブート順序を設定する場合は、他のディスクのブート順序も設定する必要があります。

2. 変更を適用します。
   - 仮想マシンを実行していない場合は、次のコマンドを実行します。

```
$ virtctl start <vm> -n <namespace>
```

   - 仮想マシンが実行中の場合は、仮想マシンを再起動するか、次のコマンドを実行します。

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

3. 仮想マシンが起動したら、SATA CD ドライブから VirtIO ドライバーをインストールします。

7.2.5.3. VirtIO ドライバーの更新

7.2.5.3.1. Windows 仮想マシンでの VirtIO ドライバーの更新

Windows Update サービスを使用して、Windows 仮想マシン上の **virtio** ドライバーを更新します。

前提条件

- クラスターはインターネットに接続されている必要があります。切断されたクラスターは Windows Update サービスにアクセスできません。

手順

1. Windows ゲストオペレーティングシステムで、**Windows** キーをクリックし、**Settings** を選択します。
2. **Windows Update → Advanced Options → Optional Updates** に移動します。

3. **Red Hat, Inc.** からのすべての更新をインストールします。
4. 仮想マシンを再起動します。

検証

1. Windows 仮想マシンで、**Device Manager** に移動します。
2. デバイスを選択します。
3. **Driver** タブを選択します。
4. **Driver Details** をクリックし、**virtio** ドライバーの詳細に正しいバージョンが表示されていることを確認します。

7.2.6. 仮想マシンのクローン作成

仮想マシンのクローンを作成するか、スナップショットから新しい仮想マシンを作成できます。

7.2.6.1. Web コンソールを使用して仮想マシンを複製する

Web コンソールを使用して、既存の仮想マシンを複製できます。

手順

1. Web コンソールで **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Actions** をクリックします。
4. **Clone** を選択します。
5. **Clone VirtualMachine** ページで、新しい仮想マシンの名前を入力します。
6. (オプション) **Start cloned VM**チェックボックスを選択して、複製された仮想マシンを開始します。
7. **Clone** をクリックします。

7.2.6.2. Web コンソールを使用して既存のスナップショットから仮想マシンを作成する

既存のスナップショットをコピーすることで、新しい仮想マシンを作成できます。

手順

1. Web コンソールで **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Snapshots** タブをクリックします。
4. コピーするスナップショットのアクションメニュー  をクリックします。

5. **Create VirtualMachine** を選択します。
6. 仮想マシンの名前を入力します。
7. (オプション) **Start this VirtualMachine after creation** チェックボックスを選択して、新しい仮想マシンを起動します。
8. **Create** をクリックします。

7.2.6.3. 関連情報

- PVC のクローン作成による仮想マシンの作成

7.2.7. PVC のクローン作成による仮想マシンの作成

カスタムイメージを使用して既存の永続ボリューム要求 (PVC) のクローンを作成することで、仮想マシンを作成できます。

Red Hat が提供していないオペレーティングシステムイメージから作成された仮想マシンには、QEMU ゲストエージェント をインストールする必要があります。

PVC のクローンを作成するには、ソース PVC を参照するデータボリュームを作成します。

7.2.7.1. クローン作成について

データボリュームのクローンを作成する場合、Containerized Data Importer (CDI) は、次の Container Storage Interface (CSI) クローンメソッドのいずれかを選択します。

- CSI ボリュームのクローン作成
- スマートクローン作成

CSI ボリュームのクローン作成方法とスマートクローン作成方法はどちらも効率的ですが、使用するには特定の要件があります。要件が満たされていない場合、CDI はホスト支援型クローン作成を使用します。ホスト支援型クローン作成は、最も時間がかかり、最も効率の悪いクローン作成方法ですが、他の 2 つのクローン作成方法よりも要件の数が少ないです。

7.2.7.1.1. CSI ボリュームのクローン作成

Container Storage Interface (CSI) のクローン作成では、CSI ドライバー機能を使用して、ソースデータボリュームのクローンをより効率的に作成します。

CSI ボリュームのクローン作成には次の要件があります。

- 永続ボリューム要求 (PVC) のストレージクラスをサポートする CSI ドライバーは、ボリュームのクローン作成をサポートする必要があります。
- CDI によって認識されないプロビジョナーの場合、対応するストレージプロファイルの **cloneStrategy** が CSI Volume Cloning に設定されている必要があります。
- ソース PVC とターゲット PVC は、同じストレージクラスとボリュームモードを持つ必要があります。
- データボリュームを作成する場合は、ソース namespace に **datavolumes/source** リソースを作成するパーミッションが必要です。

- ソースボリュームは使用されていない状態である必要があります。

7.2.7.1.2. スマートクローン作成

スナップショット機能を備えた Container Storage Interface (CSI) プラグインが使用可能な場合、Containerized Data Importer (CDI) はスナップショットから永続ボリューム要求 (PVC) を作成し、これにより、追加の PVC の効率的なクローン作成を可能にします。

スマートクローン作成には次の要件があります。

- ストレージクラスに関連付けられたスナップショットクラスが存在する必要があります。
- ソース PVC とターゲット PVC は、同じストレージクラスとボリュームモードを持つ必要があります。
- データボリュームを作成する場合は、ソース namespace に **datavolumes/source** リソースを作成するパーミッションが必要です。
- ソースボリュームは使用されていない状態である必要があります。

7.2.7.1.3. ホスト支援型クローン作成

Container Storage Interface (CSI) ボリュームのクローン作成もスマートクローン作成の要件も満たされていない場合、ホスト支援型クローン作成がフォールバック方法として使用されます。ホスト支援型クローン作成は、他の 2 つのクローン作成方法と比べると効率が悪いです。

ホスト支援型クローン作成では、ソース Pod とターゲット Pod を使用して、ソースボリュームからターゲットボリュームにデータをコピーします。ターゲットの永続ボリューム要求 (PVC) には、ホスト支援型クローン作成が使用された理由を説明するフォールバック理由のアノテーションが付けられ、イベントが作成されます。

PVC ターゲットアノテーションの例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  annotations:
    cdi.kubevirt.io/cloneFallbackReason: The volume modes of source and target are incompatible
    cdi.kubevirt.io/clonePhase: Succeeded
    cdi.kubevirt.io/cloneType: copy
```

イベント例

```
NAMESPACE   LAST SEEN   TYPE      REASON                  OBJECT                          MESSAGE
test-ns     0s          Warning   IncompatibleVolumeModes persistentvolumeclaim/test-target   The volume modes of source and target are incompatible
```

7.2.7.2. Web コンソールを使用した PVC からの仮想マシンの作成

OpenShift Container Platform Web コンソールを使用して Web ページからイメージをインポートすることで、仮想マシンを作成できます。OpenShift Container Platform Web コンソールを使用して永続ボリューム要求 (PVC) のクローンを作成することにより、仮想マシンを作成できます。

前提条件

- イメージを含む Web ページにアクセスできる。
- ソース PVC を含む namespace にアクセスできる。

手順

1. Web コンソールで **Virtualization → Catalog** に移動します。
2. 使用可能なブートソースのないテンプレートタイルをクリックします。
3. **Customize VirtualMachine** をクリックします。
4. **テンプレートパラメーターのカスタマイズ** ページで、**Storage** を展開し、**Disk source** リストから **PVC (clone PVC)** を選択します。
5. イメージの URL を入力します。例:
https://access.redhat.com/downloads/content/69/ver=/rhel---7/7.9/x86_64/product-software
6. コンテナーイメージの URL を入力します。例:
https://mirror.arizona.edu/fedora/linux/releases/38/Cloud/x86_64/images/Fedora-Cloud-Base-38-1.6.x86_64.qcow2
7. PVC プロジェクトと PVC 名を選択します。
8. ディスクサイズを設定します。
9. **Next** をクリックします。
10. **Create VirtualMachine** をクリックします。

7.2.7.3. コマンドラインを使用した PVC からの仮想マシンの作成

コマンドラインを使用して既存の仮想マシンの永続ボリューム要求 (PVC) のクローンを作成することで、仮想マシンを作成できます。

次のオプションのいずれかを使用して、PVC のクローンを作成できます。

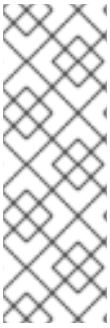
- PVC を新しいデータボリュームに複製します。
この方法では、ライフサイクルが元の仮想マシンから独立したデータボリュームが作成されます。元の仮想マシンを削除しても、新しいデータボリュームやそれに関連付けられた PVC には影響しません。
- **dataVolumeTemplates** スタンザを含む **VirtualMachine** マニフェストを作成して、PVC を複製します。
この方法では、ライフサイクルが元の仮想マシンに依存するデータボリュームが作成されます。元の仮想マシンを削除すると、クローン作成されたデータボリュームとそれに関連付けられた PVC も削除されます。

7.2.7.3.1. データボリュームへの PVC のクローン作成

コマンドラインを使用して、既存の仮想マシンディスクの永続ボリューム要求 (PVC) のクローンをデータボリュームに作成できます。

元のソース PVC を参照するデータボリュームを作成します。新しいデータボリュームのライフサイクルは、元の仮想マシンから独立しています。元の仮想マシンを削除しても、新しいデータボリュームやそれに関連付けられた PVC には影響しません。

異なるボリュームモード間のクローン作成は、ソース PV とターゲット PV が **kubevirt** コンテンツタイプに属している限り、ブロック永続ボリューム (PV) からファイルシステム PV へのクローン作成など、ホスト支援型クローン作成でサポートされます。



注記

スマートクローン作成は、スナップショットを使用して PVC のクローンを作成するため、ホスト支援型クローン作成よりも高速かつ効率的です。スマートクローン作成は、Red Hat OpenShift Data Foundation など、スナップショットをサポートするストレージプロバイダーによってサポートされています。

異なるボリュームモード間のクローン作成は、スマートクローン作成ではサポートされていません。

前提条件

- ソース PVC を含む仮想マシンの電源をオフにする必要があります。
- PVC を別の namespace に複製する場合は、ターゲットの namespace にリソースを作成するパーミッションが必要です。
- スマートクローン作成の追加の前提条件:
  - 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 PVC とターゲット PVC には、同じストレージプロバイダーとボリュームモードがある必要があります。
  - 次の例に示すように、**VolumeSnapshotClass** オブジェクトの **driver** キーの値は、**StorageClass** オブジェクトの **provisioner** キーの値と一致する必要があります。

    VolumeSnapshotClass オブジェクトの例

    ```
    kind: VolumeSnapshotClass
    apiVersion: snapshot.storage.k8s.io/v1
    driver: openshift-storage.rbd.csi.ceph.com
    # ...
    ```

    StorageClass オブジェクトの例

    ```
    kind: StorageClass
    apiVersion: storage.k8s.io/v1
    # ...
    provisioner: openshift-storage.rbd.csi.ceph.com
    ```

手順

1. 次の例に示すように、**DataVolume** マニフェストを作成します。

    ```
    apiVersion: cdi.kubevirt.io/v1beta1
    kind: DataVolume
    ```

```
metadata:
  name: <datavolume> 1
spec:
  source:
    pvc:
      namespace: "<source_namespace>" 2
      name: "<my_vm_disk>" 3
  storage: {}
```

1 新しいデータボリュームの名前を指定します。

2 ソース PVC の namespace を指定します。

3 ソース PVC の名前を指定します。

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 <datavolume>.yaml
```

注記

データ量により、PVC が準備される前に仮想マシンが起動できなくなります。PVC のクローン作成中に、新しい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。

7.2.7.3.2. データボリュームテンプレートを使用したクローン PVC からの仮想マシンの作成

データボリュームテンプレートを使用して、既存の仮想マシンの永続ボリューム要求 (PVC) のクローンを作成する仮想マシンを作成できます。

この方法では、ライフサイクルが元の仮想マシンに依存するデータボリュームが作成されます。元の仮想マシンを削除すると、クローン作成されたデータボリュームとそれに関連付けられた PVC も削除されます。

前提条件

- ソース PVC を含む仮想マシンの電源をオフにする必要があります。

手順

1. 次の例に示すように、**VirtualMachine** マニフェストを作成します。

```
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> 2
        name: "<source_pvc>" 3
```

1 仮想マシンの名前を指定します。

2 ソース PVC の namespace を指定します。

3 ソース PVC の名前を指定します。

2. 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
```

7.3. 仮想マシンコンソールへの接続

次のコンソールに接続して、実行中の仮想マシンにアクセスできます。

- VNC コンソール
- Serial Console
- Windows 仮想マシン用のデスクトップビューアー

7.3.1. VNC コンソールへの接続

OpenShift Container Platform Web コンソールまたは **virtctl** コマンドラインツールを使用して、仮想マシンの VNC コンソールに接続できます。

7.3.1.1. Web コンソールを使用した VNC コンソールへの接続

OpenShift Container Platform Web コンソールを使用して、仮想マシンの VNC コンソールに接続できます。

注記

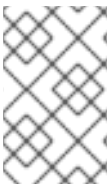
vGPU が仲介デバイスとして割り当てられている Windows 仮想マシンに接続すると、デフォルトの表示と vGPU 表示を切り替えることができます。

手順

1. **Virtualization → VirtualMachines** ページで仮想マシンをクリックして、**VirtualMachine details** ページを開きます。
2. **Console** タブをクリックします。VNC コンソールセッションが自動的に開始します。
3. オプション: Windows 仮想マシンの vGPU 表示に切り替えるには、**Send key** リストから **Ctl + Alt + 2** を選択します。
   - デフォルトの表示に戻すには、**Send key** リストから **Ctl + Alt + 1**を選択します。
4. コンソールセッションを終了するには、コンソールペインの外側をクリックし、**Disconnect** をクリックします。

7.3.1.2. virtctl を使用した VNC コンソールへの接続

virtctl コマンドラインツールを使用して、実行中の仮想マシンの VNC コンソールに接続できます。



注記

SSH 接続経由でリモートマシン上で **virtctl vnc** コマンドを実行する場合は、**-X** フラグまたは **-Y** フラグを指定して **ssh** コマンドを実行して、X セッションをローカルマシンに転送する必要があります。

前提条件

- **virt-viewer** パッケージをインストールする必要があります。

手順

1. 次のコマンドを実行して、コンソールセッションを開始します。

```
$ virtctl vnc <vm_name>
```

2. 接続に失敗した場合は、次のコマンドを実行してトラブルシューティング情報を収集します。

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

7.3.1.3. VNC コンソールの一時トークンの生成

Kubernetes API が仮想マシン (VM) の VNC にアクセスするための一時的な認証ベアラートークンを生成します。

注記

Kubernetes は、curl コマンドを変更することで、ベアラートークンの代わりにクライアント証明書を使用した認証もサポートします。

前提条件

- OpenShift Virtualization 4.14 以降および **ssp-Operator** 4.14 以降を搭載した仮想マシンが稼働している。

手順

1. HyperConverged (**HCO**) カスタムリソース (CR) の機能ゲートを有効にします。

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

2. 次のコマンドを入力してトークンを生成します。

```
$ curl --header "Authorization: Bearer ${TOKEN}" \
    "https://api.
<cluster_fqdn>/apis/token.kubevirt.io/v1alpha1/namespaces/<namespace>/virtualmachines/<vm
_name>/vnc?duration=<duration>"
```

<duration> パラメーターは時間と分で設定でき、最小期間は 10 分です。たとえば、**5h30m** です。このパラメーターが設定されていない場合、トークンはデフォルトで 10 分間有効です。

出力サンプル

```
{ "token": "eyJhb..." }
```

3. オプション: 出力で提供されたトークンを使用して変数を作成します。

```
$ export VNC_TOKEN="<token>"
```

これで、トークンを使用して、仮想マシンの VNC コンソールにアクセスできるようになります。

検証

1. 次のコマンドを入力してクラスターにログインします。

```
$ oc login --token ${VNC_TOKEN}
```

2. **virtctl** コマンドを使用して、仮想マシンの VNC コンソールへのアクセスをテストします。

```
$ virtctl vnc <vm_name> -n <namespace>
```

警告

現在、特定のトークンを取り消すことはできません。

トークンを取り消すには、トークンの作成に使用したサービスアカウントを削除する必要があります。ただし、これにより、サービスアカウントを使用して作成された他のすべてのトークンも取り消されます。次のコマンドは注意して使用してください。

```
$ virtctl delete serviceaccount --namespace "<namespace>" "<vm_name>-vnc-access"
```

7.3.1.3.1. クラスターロールを使用して VNC コンソールにトークン生成権限を付与する

クラスター管理者は、クラスターロールをインストールし、それをユーザーまたはサービスアカウントにバインドして、VNC コンソールのトークンを生成するエンドポイントへのアクセスを許可できます。

手順

- クラスターロールをユーザーまたはサービスアカウントにバインドすることを選択します。
  - クラスターロールをユーザーにバインドするには、次のコマンドを実行します。

    ```
    $ kubectl create rolebinding "${ROLE_BINDING_NAME}" --
    clusterrole="token.kubevirt.io:generate" --user="${USER_NAME}"
    ```

  - 以下のコマンドを実行してクラスターロールをサービスアカウントにバインドします。

    ```
    $ kubectl create rolebinding "${ROLE_BINDING_NAME}" --
    clusterrole="token.kubevirt.io:generate" --
    serviceaccount="${SERVICE_ACCOUNT_NAME}"
    ```

7.3.2. シリアルコンソールへの接続

OpenShift Container Platform Web コンソールまたは **virtctl** コマンドラインツールを使用して、仮想マシンのシリアルコンソールに接続できます。

注記

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

7.3.2.1. Web コンソールを使用したシリアルコンソールへの接続

OpenShift Container Platform 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に接続できます。

手順

1. **Virtualization → VirtualMachines** ページで仮想マシンをクリックして、**VirtualMachine details** ページを開きます。
2. **Console** タブをクリックします。VNC コンソールセッションが自動的に開始します。
3. **Disconnect** をクリックして、VNC コンソールセッションを終了します。それ以外の場合、VNC コンソールセッションは引き続きバックグラウンドで実行されます。
4. コンソールリストから **Serial console** を選択します。
5. コンソールセッションを終了するには、コンソールペインの外側をクリックし、**Disconnect** をクリックします。

7.3.2.2. virtctl を使用したシリアルコンソールへの接続

virtctl コマンドラインツールを使用して、実行中の仮想マシンのシリアルコンソールに接続できます。

手順

1. 次のコマンドを実行して、コンソールセッションを開始します。

```
$ virtctl console <vm_name>
```

2. **Ctrl+]** を押してコンソールセッションを終了します。

7.3.3. デスクトップビューアーに接続する

デスクトップビューアーとリモートデスクトッププロトコル (RDP) を使用して、Windows 仮想マシンに接続できます。

7.3.3.1. Web コンソールを使用したデスクトップビューアーへの接続

OpenShift Container Platform Web コンソールを使用して、Windows 仮想マシンのデスクトップビューアーに接続できます。

前提条件

- QEMU ゲストエージェントが Windows 仮想マシンにインストールされている。
- RDP クライアントがインストールされている。

手順

1. **Virtualization → VirtualMachines** ページで仮想マシンをクリックして、**VirtualMachine details** ページを開きます。
2. **Console** タブをクリックします。VNC コンソールセッションが自動的に開始します。
3. **Disconnect** をクリックして、VNC コンソールセッションを終了します。それ以外の場合、VNC コンソールセッションは引き続きバックグラウンドで実行されます。
4. コンソールのリストから **Desktop viewer** を選択します。
5. **RDP サービスの作成** をクリックして、**RDP サービス** ダイアログを開きます。

6. **Expose RDP Service** を選択し、**Save** をクリックしてノードポートサービスを作成します。
7. **Launch Remote Desktop** をクリックして **.rdp** ファイルをダウンロードし、デスクトップビューアーを起動します。

7.4. 仮想マシンへの SSH アクセスの設定

次の方法を使用して、仮想マシンへの SSH アクセスを設定できます。

- **virtctl ssh コマンド**
  SSH キーペアを作成し、公開キーを仮想マシンに追加し、秘密キーを使用して **virtctl ssh** コマンドを実行して仮想マシンに接続します。

  cloud-init データソースを使用して設定できるゲストオペレーティングシステムを使用して、実行時または最初の起動時に Red Hat Enterprise Linux (RHEL) 9 仮想マシンに公開 SSH キーを追加できます。

- **virtctl port-forward コマンド**
  virtctl port-foward コマンドを **.ssh/config** ファイルに追加し、OpenSSH を使用して仮想マシンに接続します。

- サービス
  サービスを作成し、そのサービスを仮想マシンに関連付け、サービスによって公開されている IP アドレスとポートに接続します。

- セカンダリーネットワーク
  セカンダリーネットワークを設定し、仮想マシンをセカンダリーネットワークインターフェイスに接続し、DHCP によって割り当てられた IP アドレスに接続します。

7.4.1. アクセス設定の考慮事項

仮想マシンへのアクセスを設定する各方法には、トラフィックの負荷とクライアントの要件に応じて利点と制限があります。

サービスは優れたパフォーマンスを提供するため、クラスターの外部からアクセスされるアプリケーションに推奨されます。

内部クラスターネットワークがトラフィック負荷を処理できない場合は、セカンダリーネットワークを設定できます。

virtctl ssh および virtctl port-forwarding コマンド

- 設定が簡単。
- 仮想マシンのトラブルシューティングに推奨されます。
- Ansible を使用した仮想マシンの自動設定には、**virtctl port-forwarding** が推奨されます。
- 動的公開 SSH キーを使用して、Ansible で仮想マシンをプロビジョニングできます。
- API サーバーに負担がかかるため、Rsync やリモートデスクトッププロトコルなどの高トラフィックのアプリケーションには推奨されません。
- API サーバーはトラフィック負荷を処理できる必要があります。
- クライアントは API サーバーにアクセスできる必要があります。

- クライアントはクラスターへのアクセス認証情報を持っている必要があります。

クラスター IP サービス

- 内部クラスターネットワークはトラフィック負荷を処理できる必要があります。
- クライアントは内部クラスター IP アドレスにアクセスできる必要があります。

ノードポートサービス

- 内部クラスターネットワークはトラフィック負荷を処理できる必要があります。
- クライアントは少なくとも 1 つのノードにアクセスできる必要があります。

ロードバランサーサービス

- ロードバランサーを設定する必要があります。
- 各ノードは、1 つ以上のロードバランサーサービスのトラフィック負荷を処理できなければなりません。

セカンダリーネットワーク

- トラフィックが内部クラスターネットワークを経由しないため、優れたパフォーマンスが得られます。
- ネットワークトポロジーへの柔軟なアプローチを可能にします。
- 仮想マシンはセカンダリーネットワークに直接公開されるため、ゲストオペレーティングシステムは適切なセキュリティーを備えて設定する必要があります。仮想マシンが侵害されると、侵入者がセカンダリーネットワークにアクセスする可能性があります。

7.4.2. virtctl ssh の使用

virtctl ssh コマンドを実行することで、公開 SSH キーを仮想マシンに追加し、仮想マシンに接続できます。

この方法の設定は簡単です。ただし、API サーバーに負担がかかるため、トラフィック負荷が高い場合には推奨されません。

7.4.2.1. 静的および動的 SSH キー管理について

公開 SSH キーは、最初の起動時に静的に、または実行時に動的に仮想マシンに追加できます。

注記

Red Hat Enterprise Linux (RHEL) 9 のみが動的キーインジェクションをサポートしています。

静的 SSH キー管理
cloud-init データソースを使用して、設定をサポートするゲストオペレーティングシステムを備えた仮想マシンに静的に管理された SSH キーを追加できます。キーは、最初の起動時に仮想マシンに追加されます。

次のいずれかの方法を使用してキーを追加できます。

- Web コンソールまたはコマンドラインを使用して単一の仮想マシンを作成する場合は、その仮想マシンにキーを追加します。
- Web コンソールを使用してプロジェクトにキーを追加します。その後、このプロジェクトで作成した仮想マシンにキーが自動的に追加されます。

ユースケース

- 仮想マシン所有者は、新しく作成したすべての仮想マシンを 1 つのキーでプロビジョニングできます。

動的な SSH キー管理
Red Hat Enterprise Linux (RHEL) 9 がインストールされている仮想マシンに対して動的 SSH キー管理を有効にすることができます。その後、実行時にキーを更新できます。キーは、Red Hat ブートソースとともにインストールされる QEMU ゲストエージェントによって追加されます。

セキュリティー上の理由から、動的キー管理を無効にできます。その後、仮想マシンは作成元のイメージのキー管理設定を継承します。

ユースケース

- 仮想マシンへのアクセスの付与または取り消し: クラスター管理者は、namespace 内のすべての仮想マシンに適用される **Secret** オブジェクトに個々のユーザーのキーを追加または削除することで、リモート仮想マシンアクセスを付与または取り消すことができます。
- ユーザーアクセス: 作成および管理するすべての仮想マシンにアクセス認証情報を追加できます。
- Ansible プロビジョニング:
  - 運用チームのメンバーは、Ansible プロビジョニングに使用されるすべてのキーを含む 1 つのシークレットを作成できます。
  - 仮想マシン所有者は、仮想マシンを作成し、Ansible プロビジョニングに使用されるキーをアタッチできます。
- キーのローテーション:
  - クラスター管理者は、namespace 内の仮想マシンによって使用される Ansible プロビジョナーキーをローテーションできます。
  - ワークロード所有者は、管理する仮想マシンのキーをローテーションできます。

7.4.2.2. 静的キー管理

OpenShift Container Platform Web コンソールまたはコマンドラインを使用して仮想マシンを作成するときに、静的に管理される公開 SSH キーを追加できます。このキーは、仮想マシンが初めて起動するときに、cloud-init データソースとして追加されます。

Web コンソールを使用して仮想マシンを作成するときに、公開 SSH 鍵をプロジェクトに追加することもできます。鍵はシークレットとして保存され、作成するすべての仮想マシンに自動的に追加されます。

注記

シークレットは namespace リソースであるため、シークレットをプロジェクトに追加し、その後仮想マシンを削除しても、シークレットは保持されます。シークレットは、手動で削除する必要があります。

7.4.2.2.1. テンプレートから仮想マシンを作成するときにキーを追加する

OpenShift Container Platform Web コンソールを使用して仮想マシンを作成するときに、静的に管理される公開 SSH キーを追加できます。キーは、最初の起動時に cloud-init データソースとして仮想マシンに追加されます。このメソッドは、cloud-init ユーザーデータには影響しません。

オプション: プロジェクトにキーを追加できます。その後、このキーはプロジェクトで作成した仮想マシンに自動的に追加されます。

前提条件

- **ssh-keygen** コマンドを実行して、SSH 鍵ペアを生成しました。

手順

1. Web コンソールで **Virtualization → Catalog** に移動します。
2. テンプレートタイルをクリックします。
   ゲストオペレーティングシステムは、cloud-init データソースからの設定をサポートする必要があります。
3. **Customize VirtualMachine** をクリックします。
4. **Next** をクリックします。
5. **Scripts** タブをクリックします。
6. 公開 SSH キーをまだプロジェクトに追加していない場合は、**Authorized SSH key** の横にある編集アイコンをクリックし、次のオプションのいずれかを選択します。
   - **Use existing**: シークレットリストからシークレットを選択します。
   - **Add new**:
     a. SSH キーファイルを参照するか、ファイルをキーフィールドに貼り付けます。
     b. シークレット名を入力します。
     c. オプション: **Automatically apply this key to any new VirtualMachine you create in this project** を選択します。
7. **Save** をクリックします。
8. **Create VirtualMachine** をクリックします。
   VirtualMachine details ページには、仮想マシン作成の進行状況が表示されます。

検証

- **Configuration** タブの **Scripts** タブをクリックします。
  シークレット名は **Authorized SSH key** セクションに表示されます。

7.4.2.2.2. Web コンソールを使用してインスタンスタイプから仮想マシンを作成するときにキーを追加する

OpenShift Container Platform Web コンソールを使用して、インスタンスタイプから仮想マシンを作成できます。Web コンソールを使用して、既存のスナップショットをコピーするか仮想マシンを複製して、仮想マシンを作成することもできます。

使用可能な起動可能なボリュームのリストから仮想マシンを作成できます。Linux ベースまたは Windows ベースのボリュームをリストに追加できます。

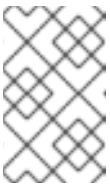
OpenShift Container Platform Web コンソールを使用してインスタンスタイプから仮想マシンを作成するときに、静的に管理される SSH キーを追加できます。キーは、最初の起動時に cloud-init データソースとして仮想マシンに追加されます。このメソッドは、cloud-init ユーザーデータには影響しません。

手順

1. Web コンソールで、**Virtualization → Catalog** に移動します。
   InstanceTypes タブがデフォルトで開きます。

2. 次のオプションのいずれかを選択します。

   - リストから適切な起動可能なボリュームを選択します。リストが切り捨てられている場合は、**Show all** ボタンをクリックしてリスト全体を表示します。

     注記

     ブート可能ボリュームテーブルには、**openshift-virtualization-os-images** namespace 内の **instancetype.kubevirt.io/default-preference** ラベルを持つボリュームのみリストされます。

     - オプション: 星アイコンをクリックして、ブート可能ボリュームをお気に入りとして指定します。星付きのブート可能ボリュームは、ボリュームリストの最初に表示されます。

   - 新しいボリュームをアップロードするか、既存の永続ボリューム要求 (PVC)、ボリュームスナップショット、または **containerDisk** ボリュームを使用するには **Add volume** をクリックします。**Save** をクリックします。
     クラスターで使用できないオペレーティングシステムのロゴは、リストの下部に表示されます。**Add volume** リンクをクリックすると、必要なオペレーティングシステムのボリュームを追加できます。

     さらに、**Create a Windows boot source**クイックスタートへのリンクもあります。**Select volume to boot from** の横にある疑問符アイコンの上にポインターを置くと、ポップオーバーに同じリンクが表示されます。

     環境をインストールした直後、または環境が切断された直後は、起動元のボリュームのリストは空になります。その場合、Windows、RHEL、Linux の 3 つのオペレーティングシステムのロゴが表示されます。**Add volume** ボタンをクリックすると、要件を満たす新しいボリュームを追加できます。

3. インスタンスタイプのタイルをクリックし、ワークロードに適したリソースサイズを選択します。

4. オプション: 起動元のボリュームに適用される仮想マシンの詳細 (仮想マシンの名前を含む) を選択します。

- Linux ベースのボリュームの場合は、次の手順に従って SSH を設定します。
  a. 公開 SSH キーをまだプロジェクトに追加していない場合は、**VirtualMachine details** セクションの **Authorized SSH key** の横にある編集アイコンをクリックします。
  b. 以下のオプションのいずれかを選択します。
    - **Use existing**: シークレットリストからシークレットを選択します。
    - **Add new**: 以下の手順に従ってください。
      i. 公開 SSH キーファイルを参照するか、ファイルをキーフィールドに貼り付けます。
      ii. シークレット名を入力します。
      iii. オプション: **Automatically apply this key to any new VirtualMachine you create in this project** を選択します。
  c. **Save** をクリックします。
- Windows ボリュームの場合は、次のいずれかの手順に従って sysprep オプションを設定します。
  - Windows ボリュームに sysprep オプションをまだ追加していない場合は、次の手順に従います。
    i. **VirtualMachine details** セクションの **Sysprep** の横にある編集アイコンをクリックします。
    ii. **Autoattend.xml** アンサーファイルを追加します。
    iii. **Unattend.xml** アンサーファイルを追加します。
    iv. **Save** をクリックします。
  - Windows ボリュームに既存の sysprep オプションを使用する場合は、次の手順に従います。
    i. **既存の sysprep を添付** をクリックします。
    ii. 既存の sysprep **Unattend.xml** アンサーファイルの名前を入力します。
    iii. **Save** をクリックします。

5. オプション: Windows 仮想マシンを作成する場合は、Windows ドライバーディスクをマウントできます。
   a. **Customize VirtualMachine** ボタンをクリックします。
   b. **VirtualMachine details** ページで、**Storage** をクリックします。
   c. **Mount Windows drivers disk** チェックボックスを選択します。
6. オプション: **View YAML & CLI** をクリックして YAML ファイルを表示します。**CLI** をクリックして CLI コマンドを表示します。YAML ファイルの内容または CLI コマンドをダウンロードまたはコピーすることもできます。
7. **Create VirtualMachine** をクリックします。

仮想マシンの作成後、**VirtualMachine details** ページでステータスを監視できます。

7.4.2.2.3. コマンドラインを使用して仮想マシンを作成するときにキーを追加する

コマンドラインを使用して仮想マシンを作成するときに、静的に管理される公開 SSH キーを追加できます。キーは最初の起動時に仮想マシンに追加されます。

キーは、cloud-init データソースとして仮想マシンに追加されます。このメソッドは、cloud-init ユーザーデータ内のアプリケーションデータからアクセス認証情報を分離します。このメソッドは、cloud-init ユーザーデータには影響しません。

前提条件

- **ssh-keygen** コマンドを実行して、SSH 鍵ペアを生成しました。

手順

1. **VirtualMachine** オブジェクトと **Secret** オブジェクトのマニフェストファイルを作成します。

マニフェストの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  dataVolumeTemplates:
    - metadata:
        name: example-vm-volume
      spec:
        sourceRef:
          kind: DataSource
          name: rhel9
          namespace: openshift-virtualization-os-images
        storage:
          resources: {}
  instancetype:
    name: u1.medium
  preference:
    name: rhel.9
  running: true
  template:
    spec:
      domain:
        devices: {}
      volumes:
        - dataVolume:
            name: example-vm-volume
          name: rootdisk
        - cloudInitNoCloud: 1
            userData: |-
              #cloud-config
              user: cloud-user
          name: cloudinitdisk
```

```
      accessCredentials:
       - sshPublicKey:
           propagationMethod:
             noCloud: {}
           source:
             secret:
               secretName: authorized-keys 2
---
apiVersion: v1
kind: Secret
metadata:
  name: authorized-keys
data:
  key: c3NoLXJzYSB... 3
```

1. **cloudInitNoCloud** データソースを指定します。
2. **Secret** オブジェクト名を指定します。
3. 公開 SSH キーを貼り付けます。

2. 次のコマンドを実行して、**VirtualMachine** オブジェクトと **Secret** オブジェクトを作成します。

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

3. 次のコマンドを実行して VM を起動します。

```
$ virtctl start vm example-vm -n example-namespace
```

検証

- 仮想マシン設定を取得します。

```
$ oc describe vm example-vm -n example-namespace
```

出力例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  template:
    spec:
      accessCredentials:
       - sshPublicKey:
           propagationMethod:
             noCloud: {}
           source:
```

```
            secret:
              secretName: authorized-keys
# ...
```

7.4.2.3. 動的なキー管理

OpenShift Container Platform Web コンソールまたはコマンドラインを使用して、仮想マシンの動的キーインジェクションを有効にできます。その後、実行時にキーを更新できます。

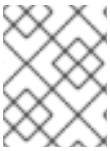
注記

Red Hat Enterprise Linux (RHEL) 9 のみが動的キーインジェクションをサポートしています。

動的キーインジェクションを無効にすると、仮想マシンは作成元のイメージのキー管理方法を継承します。

7.4.2.3.1. テンプレートから仮想マシンを作成するときに動的キーインジェクションを有効にする

OpenShift Container Platform Web コンソールを使用してテンプレートから仮想マシンを作成するときに、動的パブリック SSH キーインジェクションを有効にすることができます。その後、実行時にキーを更新できます。



注記

Red Hat Enterprise Linux (RHEL) 9 のみが動的キーインジェクションをサポートしています。

キーは、RHEL 9 とともにインストールされる QEMU ゲストエージェントによって仮想マシンに追加されます。

前提条件

- **ssh-keygen** コマンドを実行して、SSH 鍵ペアを生成しました。

手順

1. Web コンソールで **Virtualization → Catalog** に移動します。
2. **Red Hat Enterprise Linux 9 VM**タイルをクリックします。
3. **Customize VirtualMachine** をクリックします。
4. **Next** をクリックします。
5. **Scripts** タブをクリックします。
6. 公開 SSH キーをまだプロジェクトに追加していない場合は、**Authorized SSH key** の横にある編集アイコンをクリックし、次のオプションのいずれかを選択します。
   - **Use existing**: シークレットリストからシークレットを選択します。
   - **Add new**:

a. SSH キーファイルを参照するか、ファイルをキーフィールドに貼り付けます。

b. シークレット名を入力します。

c. オプション: **Automatically apply this key to any new VirtualMachine you create in this project** を選択します。

7. **Dynamic SSH key injection** をオンに設定します。

8. **Save** をクリックします。

9. **Create VirtualMachine** をクリックします。
VirtualMachine details ページには、仮想マシン作成の進行状況が表示されます。

検証

- **Configuration** タブの **Scripts** タブをクリックします。
シークレット名は **Authorized SSH key** セクションに表示されます。

7.4.2.3.2. Web コンソールを使用してインスタンスタイプから仮想マシンを作成するときに動的キーインジェクションを有効にする

OpenShift Container Platform Web コンソールを使用して、インスタンスタイプから仮想マシンを作成できます。Web コンソールを使用して、既存のスナップショットをコピーするか仮想マシンを複製して、仮想マシンを作成することもできます。

使用可能な起動可能なボリュームのリストから仮想マシンを作成できます。Linux ベースまたは Windows ベースのボリュームをリストに追加できます。

OpenShift Container Platform Web コンソールを使用してインスタンスタイプから仮想マシンを作成するときに、動的 SSH キーインジェクションを有効にできます。その後、実行時にキーを追加または取り消すことができます。

注記

Red Hat Enterprise Linux (RHEL) 9 のみが動的キーインジェクションをサポートしています。

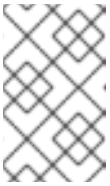
キーは、RHEL 9 とともにインストールされる QEMU ゲストエージェントによって仮想マシンに追加されます。

手順

1. Web コンソールで、**Virtualization → Catalog** に移動します。
InstanceTypes タブがデフォルトで開きます。

2. 次のオプションのいずれかを選択します。

   - リストから適切な起動可能なボリュームを選択します。リストが切り捨てられている場合は、**Show all** ボタンをクリックしてリスト全体を表示します。



注記

ブート可能ボリュームテーブルには、**openshift-virtualization-os-images** namespace 内の **instancetype.kubevirt.io/default-preference** ラベルを持つボリュームのみリストされます。

- オプション: 星アイコンをクリックして、ブート可能ボリュームをお気に入りとして指定します。星付きのブート可能ボリュームは、ボリュームリストの最初に表示されます。

- 新しいボリュームをアップロードするか、既存の永続ボリューム要求 (PVC)、ボリュームスナップショット、または **containerDisk** ボリュームを使用するには **Add volume** をクリックします。**Save** をクリックします。
  クラスターで使用できないオペレーティングシステムのロゴは、リストの下部に表示されます。**Add volume** リンクをクリックすると、必要なオペレーティングシステムのボリュームを追加できます。

  さらに、**Create a Windows boot source**クイックスタートへのリンクもあります。**Select volume to boot from** の横にある疑問符アイコンの上にポインターを置くと、ポップオーバーに同じリンクが表示されます。

  環境をインストールした直後、または環境が切断された直後は、起動元のボリュームのリストは空になります。その場合、Windows、RHEL、Linux の 3 つのオペレーティングシステムのロゴが表示されます。**Add volume** ボタンをクリックすると、要件を満たす新しいボリュームを追加できます。

3. インスタンスタイプのタイルをクリックし、ワークロードに適したリソースサイズを選択します。

4. **Red Hat Enterprise Linux 9 VM**タイルをクリックします。

5. オプション: 起動元のボリュームに適用される仮想マシンの詳細 (仮想マシンの名前を含む) を選択します。

   - Linux ベースのボリュームの場合は、次の手順に従って SSH を設定します。

     a. 公開 SSH キーをまだプロジェクトに追加していない場合は、**VirtualMachine details** セクションの **Authorized SSH key**の横にある編集アイコンをクリックします。

     b. 以下のオプションのいずれかを選択します。

        - **Use existing**: シークレットリストからシークレットを選択します。

        - **Add new**: 以下の手順に従ってください。

          i. 公開 SSH キーファイルを参照するか、ファイルをキーフィールドに貼り付けます。

          ii. シークレット名を入力します。

          iii. オプション: **Automatically apply this key to any new VirtualMachine you create in this project** を選択します。

     c. **Save** をクリックします。

   - Windows ボリュームの場合は、次のいずれかの手順に従って sysprep オプションを設定します。

- Windows ボリュームに sysprep オプションをまだ追加していない場合は、次の手順に従います。
  i. **VirtualMachine details** セクションの **Sysprep** の横にある編集アイコンをクリックします。
  ii. **Autoattend.xml** アンサーファイルを追加します。
  iii. **Unattend.xml** アンサーファイルを追加します。
  iv. **Save** をクリックします。
- Windows ボリュームに既存の sysprep オプションを使用する場合は、次の手順に従います。
  i. **既存の sysprep を添付** をクリックします。
  ii. 既存の sysprep **Unattend.xml** アンサーファイルの名前を入力します。
  iii. **Save** をクリックします。

6. **VirtualMachine details** セクションで **Dynamic SSH key injection** をオンに設定します。
7. オプション: Windows 仮想マシンを作成する場合は、Windows ドライバーディスクをマウントできます。
   a. **Customize VirtualMachine** ボタンをクリックします。
   b. **VirtualMachine details** ページで、**Storage** をクリックします。
   c. **Mount Windows drivers disk** チェックボックスを選択します。
8. オプション: **View YAML & CLI** をクリックして YAML ファイルを表示します。**CLI** をクリックして CLI コマンドを表示します。YAML ファイルの内容または CLI コマンドをダウンロードまたはコピーすることもできます。
9. **Create VirtualMachine** をクリックします。

仮想マシンの作成後、**VirtualMachine details** ページでステータスを監視できます。

7.4.2.3.3. Web コンソールを使用した動的 SSH キーインジェクションの有効化

OpenShift Container Platform Web コンソールを使用して、仮想マシンの動的キーインジェクションを有効にできます。その後、実行時に公開 SSH キーを更新できます。

キーは、Red Hat Enterprise Linux (RHEL) 9 とともにインストールされる QEMU ゲストエージェントによって仮想マシンに追加されます。

前提条件

- ゲスト OS は RHEL 9 です。

手順

1. Web コンソールで **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。

3. **Configuration** タブで、**Scripts** をクリックします。
4. 公開 SSH キーをまだプロジェクトに追加していない場合は、**Authorized SSH key** の横にある編集アイコンをクリックし、次のオプションのいずれかを選択します。
   - **Use existing**: シークレットリストからシークレットを選択します。
   - **Add new**:
     a. SSH キーファイルを参照するか、ファイルをキーフィールドに貼り付けます。
     b. シークレット名を入力します。
     c. オプション: **Automatically apply this key to any new VirtualMachine you create in this project** を選択します。
5. **Dynamic SSH key injection** をオンに設定します。
6. **Save** をクリックします。

7.4.2.3.4. コマンドラインを使用して動的キーインジェクションを有効にする

コマンドラインを使用して、仮想マシンの動的キーインジェクションを有効にすることができます。その後、実行時に公開 SSH キーを更新できます。

注記

Red Hat Enterprise Linux (RHEL) 9 のみが動的キーインジェクションをサポートしています。

キーは QEMU ゲストエージェントによって仮想マシンに追加され、RHEL 9 とともに自動的にインストールされます。

前提条件

- **ssh-keygen** コマンドを実行して、SSH 鍵ペアを生成しました。

手順

1. **VirtualMachine** オブジェクトと **Secret** オブジェクトのマニフェストファイルを作成します。

マニフェストの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  dataVolumeTemplates:
    - metadata:
        name: example-vm-volume
      spec:
        sourceRef:
          kind: DataSource
```

```
        name: rhel9
        namespace: openshift-virtualization-os-images
      storage:
        resources: {}
  instancetype:
    name: u1.medium
  preference:
    name: rhel.9
  running: true
  template:
    spec:
      domain:
        devices: {}
      volumes:
        - dataVolume:
            name: example-vm-volume
          name: rootdisk
        - cloudInitNoCloud: 1
            userData: |-
              #cloud-config
              runcmd:
              - [ setsebool, -P, virt_qemu_ga_manage_ssh, on ]
          name: cloudinitdisk
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              qemuGuestAgent:
                users: ["cloud-user"]
            source:
              secret:
                secretName: authorized-keys 2
---
apiVersion: v1
kind: Secret
metadata:
  name: authorized-keys
data:
  key: c3NoLXJzYSB... 3
```

1. **cloudInitNoCloud** データソースを指定します。
2. **Secret** オブジェクト名を指定します。
3. 公開 SSH キーを貼り付けます。

2. 次のコマンドを実行して、**VirtualMachine** オブジェクトと **Secret** オブジェクトを作成します。

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

3. 次のコマンドを実行して VM を起動します。

```
$ virtctl start vm example-vm -n example-namespace
```

検証

- 仮想マシン設定を取得します。

```
$ oc describe vm example-vm -n example-namespace
```

出力例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  template:
    spec:
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              qemuGuestAgent:
                users: ["cloud-user"]
            source:
              secret:
                secretName: authorized-keys
# ...
```

7.4.2.4. virtctl ssh コマンドの使用

virtcl ssh コマンドを使用して、実行中の仮想マシンにアクセスできます。

前提条件

- **virtctl** コマンドラインツールがインストールされました。
- 公開 SSH キーを仮想マシンに追加しました。
- SSH クライアントがインストールされています。
- **virtctl** ツールがインストールされている環境には、仮想マシンへのアクセスに必要なクラスターパーミッションがある。たとえば、**oc login** を実行するか、**KUBECONFIG** 環境変数を設定します。

手順

- **virtctl ssh** コマンドを実行します。

```
$ virtctl -n <namespace> ssh <username>@example-vm -i <ssh_key> 1
```

1 namespace、ユーザー名、SSH 秘密キーを指定します。デフォルトの SSH キーの場所は **/home/user/.ssh** です。キーを別の場所に保存する場合は、パスを指定する必要があります。

例

```
$ virtctl -n my-namespace ssh cloud-user@example-vm -i my-key
```

ヒント

VirtualMachines ページの仮想マシンの横にあるオプション  メニューから、**Copy SSH command** を選択すると、Web コンソールで **virtctl ssh** コマンドをコピーできます。

7.4.3. virtctl port-forward コマンドの使用

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

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

前提条件

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

手順

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

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

2. 次のコマンドを実行して、仮想マシンに接続します。

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

7.4.4. SSH アクセス用のサービスを使用する

仮想マシンのサービスを作成し、サービスによって公開される IP アドレスとポートに接続できます。

サービスは優れたパフォーマンスを提供するため、クラスターの外部またはクラスター内からアクセスされるアプリケーションに推奨されます。受信トラフィックはファイアウォールによって保護されます。

クラスターネットワークがトラフィック負荷を処理できない場合は、仮想マシンアクセスにセカンダリーネットワークを使用することを検討してください。

7.4.4.1. サービスについて

Kubernetes サービスは一連の Pod で実行されているアプリケーションへのクライアントのネットワークアクセスを公開します。サービスは抽象化、負荷分散を提供し、タイプ **NodePort** と **LoadBalancer** の場合は外部世界への露出を提供します。

ClusterIP

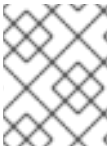
内部 IP アドレスでサービスを公開し、クラスター内の他のアプリケーションに DNS 名として公開します。1 つのサービスを複数の仮想マシンにマッピングできます。クライアントがサービスに接続しようとすると、クライアントのリクエストは使用可能なバックエンド間で負荷分散されます。**ClusterIP** はデフォルトのサービスタイプです。

NodePort

クラスター内の選択した各ノードの同じポートでサービスを公開します。**NodePort** は、ノード自体がクライアントから外部にアクセスできる限り、クラスターの外部からポートにアクセスできるようにします。

LoadBalancer

現在のクラウド（サポートされている場合）に外部ロードバランサーを作成し、固定の外部 IP アドレスをサービスに割り当てます。



注記

オンプレミスクラスターの場合、MetalLB Operator をデプロイすることで負荷分散サービスを設定できます。

7.4.4.2. サービスの作成

OpenShift Container Platform Web コンソール、**virtctl** コマンドラインツール、または YAML ファイルを使用して、仮想マシンを公開するサービスを作成できます。

7.4.4.2.1. Web コンソールを使用したロードバランサーサービスの作成の有効化

OpenShift Container Platform Web コンソールを使用して、仮想マシン (VM) のロードバランサーサービスの作成を有効にすることができます。

前提条件

- クラスターのロードバランサーが設定されました。
- **cluster-admin** ロールを持つユーザーとしてログインしている。

手順

1. **Virtualization → Overview** に移動します。
2. **Settings** タブで、**Cluster** をクリックします。
3. Expand **General settings** と **SSH configuration** を展開します。
4. **SSH over LoadBalancer service** をオンに設定します。

7.4.4.2.2. Web コンソールを使用したサービスの作成

OpenShift Container Platform Web コンソールを使用して、仮想マシンのノードポートまたはロードバランサーサービスを作成できます。

前提条件

前提条件

- ロードバランサーまたはノードポートをサポートするようにクラスターネットワークが設定されている。
- ロードバランサーサービスを作成するためにロードバランサーサービスの作成が有効化されている。

手順

1. **VirtualMachines** に移動し、仮想マシンを選択して、**VirtualMachine details** ページを表示します。
2. **Details** タブで、**SSH service type** リストから **SSH over LoadBalancer** を選択します。
3. オプション: コピーアイコンをクリックして、**SSH** コマンドをクリップボードにコピーします。

検証

- **Details** タブの **Services** ペインをチェックして、新しいサービスを表示します。

7.4.4.2.3. virtctl を使用したサービスの作成

virtctl コマンドラインツールを使用して、仮想マシンのサービスを作成できます。

前提条件

- **virtctl** コマンドラインツールがインストールされました。
- サービスをサポートするようにクラスターネットワークを設定しました。
- **virtctl** をインストールした環境には、仮想マシンにアクセスするために必要なクラスター権限があります。たとえば、**oc login** を実行するか、**KUBECONFIG** 環境変数を設定します。

手順

- 次のコマンドを実行してサービスを作成します。

```
$ virtctl expose vm <vm_name> --name <service_name> --type <service_type> --port <port>
```
(1)

(1) **ClusterIP**、**NodePort**、または **LoadBalancer** サービスタイプを指定します。

例

```
$ virtctl expose vm example-vm --name example-service --type NodePort --port 22
```

検証

- 以下のコマンドを実行してサービスを確認します。

```
$ oc get service
```

次のステップ

virtctl を使用してサービスを作成した後、**VirtualMachine** マニフェストの **spec.template.metadata.labels** スタンザに **special: key** を追加する必要があります。コマンドラインを使用したサービスの作成 を参照してください。

7.4.4.2.4. コマンドラインを使用したサービスの作成

コマンドラインを使用して、サービスを作成し、それを仮想マシンに関連付けることができます。

前提条件

- サービスをサポートするようにクラスターネットワークを設定しました。

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

(1) **special: key** を **spec.template.metadata.labels** スタンザに追加します。

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: example-service
  namespace: example-namespace
spec:
# ...
  selector:
    special: key (1)
  type: NodePort (2)
  ports: (3)
```

```
    protocol: TCP
    port: 80
    targetPort: 9376
    nodePort: 30000
```

1. **VirtualMachine** マニフェストの **spec.template.metadata.labels** スタンザに追加したラベルを指定します。
2. **ClusterIP**、**NodePort**、または **LoadBalancer** を指定します。
3. 仮想マシンから公開するネットワークポートとプロトコルのコレクションを指定します。

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 example-service.yaml
```

6. 仮想マシンを再起動して変更を適用します。

検証

- **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
```

7.4.4.3. SSH を使用してサービスによって公開される仮想マシンに接続する

SSH を使用して、サービスによって公開されている仮想マシンに接続できます。

前提条件

- 仮想マシンを公開するサービスを作成しました。
- SSH クライアントがインストールされています。
- クラスターにログインしている。

手順

- 次のコマンドを実行して仮想マシンにアクセスします。

```
$ ssh <user_name>@<ip_address> -p <port> 1
```

1. クラスター IP サービスの場合はクラスター IP、ノードポートサービスの場合はノード IP、またはロードバランサーサービスの場合は外部 IP アドレスを指定します。

7.4.5. SSH アクセスにセカンダリーネットワークを使用する

SSH を使用して、セカンダリーネットワークを設定し、仮想マシンをセカンダリーネットワークインターフェイスに接続し、DHCP によって割り当てられた IP アドレスに接続できます。

重要

セカンダリーネットワークは、トラフィックがクラスターネットワークスタックによって処理されないため、優れたパフォーマンスを提供します。ただし、仮想マシンはセカンダリーネットワークに直接公開されており、ファイアウォールによって保護されません。仮想マシンが侵害されると、侵入者がセカンダリーネットワークにアクセスする可能性があります。この方法を使用する場合は、仮想マシンのオペレーティングシステム内で適切なセキュリティーを設定する必要があります。

ネットワークオプションの詳細はOpenShift Virtualization Tuning & Scaling Guide の Multus および SR-IOV ドキュメントを参照してください。

前提条件

- Linux ブリッジ や SR-IOV などのセカンダリーネットワークを設定しました。
- Linux ブリッジネットワーク のネットワークアタッチメント定義を作成したか、**SriovNetwork** オブジェクトの作成時に SR-IOV ネットワークオペレータが ネットワークアタッチメント定義 を作成しました。

7.4.5.1. Web コンソールを使用した仮想マシンネットワークインターフェイスの設定

OpenShift Container Platform Web コンソールを使用して、仮想マシンのネットワークインターフェイスを設定できます。

前提条件

- ネットワークのネットワーク接続定義を作成しました。

手順

1. **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンをクリックして、**VirtualMachine details** ページを表示します。
3. **Configuration** タブで、**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. インターフェイス名を入力し、**Network** リストからネットワーク接続定義を選択します。
6. **Save** をクリックします。
7. 仮想マシンを再起動して変更を適用します。

7.4.5.2. SSH を使用したセカンダリーネットワークに接続された仮想マシンへの接続

SSH を使用して、セカンダリーネットワークに接続されている仮想マシンに接続できます。

前提条件

- DHCP サーバーを使用して仮想マシンをセカンダリーネットワークに接続しました。
- SSH クライアントがインストールされています。

手順

1. 次のコマンドを実行して、仮想マシンの IP アドレスを取得します。

```
$ oc describe vm <vm_name> -n <namespace>
```

出力例

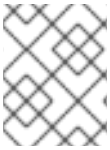
```
# ...
Interfaces:
  Interface Name:  eth0
  Ip Address:      10.244.0.37/24
  Ip Addresses:
    10.244.0.37/24
    fe80::858:aff:fef4:25/64
  Mac:             0a:58:0a:f4:00:25
  Name:            default
# ...
```

2. 次のコマンドを実行して、仮想マシンに接続します。

```
$ ssh <user_name>@<ip_address> -i <ssh_key>
```

例

```
$ ssh cloud-user@10.244.0.37 -i ~/.ssh/id_rsa_cloud-user
```



注記

クラスター FQDN を使用して、セカンダリーネットワークインターフェイスに接続された仮想マシンにアクセスする こともできます。

7.5. 仮想マシンの編集

OpenShift Container Platform Web コンソールを使用して、仮想マシン設定を更新できます。YAML ファイルまたは **VirtualMachine details** ページを更新できます。

コマンドラインを使用して仮想マシンを編集することもできます。

仮想マシンを編集して仮想ディスクまたは LUN を用いたディスク共有を設定する場合は、仮想マシンの共有ボリュームの設定 を参照してください。

7.5.1. コマンドラインを使用した仮想マシンの編集

コマンドラインを使用して仮想マシンを編集できます。

前提条件

- **oc** CLI がインストールされている。

手順

1. 次のコマンドを実行して、仮想マシンの設定を取得します。

```
$ oc edit vm <vm_name>
```

2. YAML 設定を編集します。
3. 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。

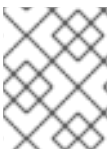
     ```
     $ oc apply vm <vm_name> -n <namespace>
     ```

7.5.2. 仮想マシンへのディスクの追加

OpenShift Container Platform Web コンソールを使用して、仮想ディスクを仮想マシンに追加できます。

手順

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



注記

仮想マシンが実行中の場合は、仮想マシンを再起動して変更を適用する必要があります。

7.5.2.1. ストレージフィールド

フィールド	説明
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。

フィールド	説明
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)。
タイプ	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 URL**、および **Clone existing PVC** ディスクで利用できます。

これらのパラメーターを指定しない場合、システムはデフォルトのストレージプロファイル値を使用します。

パラメーター	オプション	パラメーターの説明
ボリュームモード	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	ReadWriteOnce (RWO)	ボリュームはシングルノードで読み取り/書き込みとしてマウントできます。

パラメーター	オプション	パラメーターの説明
	ReadWriteMany (RWX)	ボリュームは、一度に多くのノードで読み取り/書き込みとしてマウントできます。  **注記** このモードはライブマイグレーションに必要です。

7.5.3. 仮想マシンに Windows ドライバーディスクをマウントする

OpenShift Container Platform Web コンソールを使用して、Windows ドライバーディスクを仮想マシン (VM) にマウントできます。

手順

1. **Virtualization → VirtualMachines** に移動します。
2. 目的の仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration** タブで、**Storage** をクリックします。
4. **Mount Windows drivers disk** チェックボックスを選択します。
   マウント済みディスクのリストに、Windows ドライバーディスクが表示されます。

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

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

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

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

前提条件

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

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration → Environment** をクリックします。
4. **Add Config Map, Secret or Service Account**をクリックします。

5. **Select a resource** をクリックし、リストから resource を選択します。6 文字のシリアル番号が、選択したリソースについて自動的に生成されます。
6. オプション: **Reload** をクリックして、環境を最後に保存した状態に戻します。
7. **Save** をクリックします。

検証

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

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

config map、シークレット、サービスアカウントの追加リソース

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

7.6. ブート順序の編集

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

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

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

7.6.1. Web コンソールでのブート順序リストへの項目の追加

Web コンソールを使用して、ブート順序リストに項目を追加します。

手順

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

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

注記

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

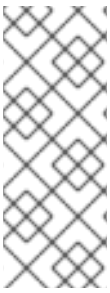
Boot Order フィールドの右側にある **View Pending Changes** をクリックして、保留中の変更を表示できます。ページ上部の **Pending Changes** バナーには、仮想マシンの再起動時に適用されるすべての変更のリストが表示されます。

7.6.2. Web コンソールでのブート順序リストの編集

Web コンソールで起動順序リストを編集します。

手順

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



注記

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

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

7.6.3. YAML 設定ファイルでのブート順序リストの編集

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

手順

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

```
$ oc edit vm <vm_name> -n <namespace>
```

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。

7.6.4. Web コンソールでのブート順序リストからの項目の削除

Web コンソールを使用して、ブート順序のリストから項目を削除します。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Details** タブをクリックします。
4. **Boot Order** の右側にある鉛筆アイコンをクリックします。
5. 項目の横にある **Remove** アイコン をクリックします。この項目はブート順序のリストから削除され、利用可能なブートソースのリストに保存されます。ブート順序リストからすべての項目を削除する場合、以下のメッセージが表示されます。**No resource selected.仮想マシンは、YAML ファイルでの出現順にディスクからの起動を試行します。**

注記

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

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

7.7. 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。

7.7.1. Web コンソールの使用による仮想マシンの削除

仮想マシンを削除すると、仮想マシンはクラスターから永続的に削除されます。

手順

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

7.7.2. CLI の使用による仮想マシンの削除

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

前提条件

- 削除する仮想マシンの名前を特定すること。

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

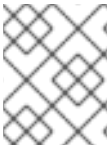
このコマンドは、現在のプロジェクト内の VM のみを削除します。削除する仮想マシンが別のプロジェクトまたは namespace にある場合は、**-n <project_name>** オプションを指定します。

7.8. 仮想マシンのエクスポート

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

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

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



注記

Migration Toolkit for Virtualization を使用して、OpenShift Virtualization クラスター間で仮想マシンを移行できます。

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

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

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

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

エクスポートサーバーは、次のファイル形式をサポートしています。

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

前提条件

- 仮想マシンをエクスポートするには、仮想マシンをシャットダウンする必要があります。

手順

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

   VirtualMachineExport の例

   ```
   apiVersion: export.kubevirt.io/v1alpha1
   ```

```
kind: VirtualMachineExport
metadata:
  name: example-export
spec:
  source:
    apiGroup: "kubevirt.io" 1
    kind: VirtualMachine 2
    name: example-vm
  ttlDuration: 1h 3
```

1 適切な API グループを指定します。

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

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

3 オプション: デフォルトの期間は 2 時間です。

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

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

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

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

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

出力例

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

```
    lastTransitionTime: "2022-06-21T14:09:02Z"
    reason: pvcBound
    status: "True"
    type: PVCReady
  links:
    external: (1)
      cert: |-
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
      volumes:
      - formats:
        - format: raw
          url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/volumes/example-disk/disk.img
        - format: gzip
          url: https://vmexport-
proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/exam
ple-export/volumes/example-disk/disk.img.gz
        name: example-disk
    internal: (2)
      cert: |-
        -----BEGIN CERTIFICATE-----
        ...
        -----END CERTIFICATE-----
      volumes:
      - formats:
        - format: raw
          url: https://virt-export-example-export.example.svc/volumes/example-disk/disk.img
        - format: gzip
          url: https://virt-export-example-export.example.svc/volumes/example-disk/disk.img.gz
        name: example-disk
  phase: Ready
  serviceName: virt-export-example-export
```

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

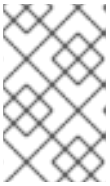
(2) 内部リンクは、クラスター内でのみ有効です。

7.8.2. エクスポートされた仮想マシンマニフェストへのアクセス

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

前提条件

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



注記

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

手順

1. マニフェストにアクセスするには、まず証明書をソースクラスターからターゲットクラスターにコピーする必要があります。
   a. ソースクラスターにログインします。
   b. 次のコマンドを実行して、証明書を **cacert.crt** ファイルに保存します。

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

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

   c. **cacert.crt** ファイルをターゲットクラスターにコピーします。
2. 次のコマンドを実行して、ソースクラスター内のトークンをデコードし、**token_decode** ファイルに保存します。

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

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

3. **token_decode** ファイルをターゲットクラスターにコピーします。
4. 次のコマンドを実行して、**VirtualMachineExport** カスタムリソースを取得します。

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

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

出力例

```
apiVersion: export.kubevirt.io/v1alpha1
kind: VirtualMachineExport
metadata:
  name: example-export
spec:
  source:
    apiGroup: "kubevirt.io"
    kind: VirtualMachine
    name: example-vm
  tokenSecretRef: example-token
```

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

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

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

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

6. ターゲットクラスターにログインします。

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

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

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

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

以下に例を示します。

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

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

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

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

以下に例を示します。

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

次のステップ

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

7.9. 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。

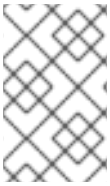
7.9.1. 仮想マシンインスタンスについて

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

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

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

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



注記

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

仮想マシンを編集すると、一部の設定が再起動なしで VMI に動的に適用される場合があります。VMI に動的に適用できない仮想マシンオブジェクトを変更すると、**RestartRequired** 仮想マシン条件がトリガーされます。変更は次回の再起動時に有効になり、条件は削除されます。

7.9.2. CLI を使用した仮想マシンインスタンスのリスト表示

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

手順

- 以下のコマンドを実行して、すべての VMI のリストを表示します。

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

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

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

注記

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

手順

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

7.9.4. 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。

7.9.5. 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>
```

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

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

手順

1. OpenShift Container Platform Web コンソールで、サイドメニューから**Virtualization → VirtualMachines**をクリックします。
2. **Actions → Delete VirtualMachineInstance** をクリックします。
3. 確認のポップアップウィンドウで、**Delete** をクリックし、スタンドアロン VMI を永続的に削除します。

7.10. 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。

7.10.1. 仮想マシンの起動

Web コンソールから仮想マシンを起動できます。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 起動する仮想マシンが含まれる行を見つけます。
3. ユースケースに応じて適切なメニューに移動します。
   - このページに留まり、複数の仮想マシンに対して操作を実行するには、次の手順を実行します。
     a. 行の右端にある Options メニュー  をクリックして、**Start VirtualMachine** 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 → Start** をクリックします。

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。

7.10.2. 仮想マシンの停止

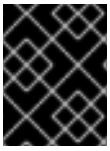
Web コンソールから仮想マシンを停止できます。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 停止する仮想マシンが含まれる行を見つけます。
3. ユースケースに応じて適切なメニューに移動します。
   - このページに留まり、複数の仮想マシンに対して操作を実行するには、次の手順を実行します。
     a. 行の右端にある Options メニュー  をクリックして、**Stop VirtualMachine** 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。

7.10.3. 仮想マシンの再起動

Web コンソールから実行中の仮想マシンを再起動できます。



重要

エラーを回避するには、ステータスが **Importing** の仮想マシンは再起動しないでください。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 再起動する仮想マシンが含まれる行を見つけます。

3. ユースケースに応じて適切なメニューに移動します。
   - このページに留まり、複数の仮想マシンに対して操作を実行するには、次の手順を実行します。
     a. 行の右端にある Options メニュー  をクリックして、**Restart** 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。

7.10.4. 仮想マシンの一時停止

Web コンソールから仮想マシンを一時停止できます。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 一時停止する仮想マシンが含まれている行を見つけます。
3. ユースケースに応じて適切なメニューに移動します。
   - このページに留まり、複数の仮想マシンに対して操作を実行するには、次の手順を実行します。
     a. 行の右端にある Options メニュー  をクリックして、**Pause VirtualMachine** をクリックします。
   - 選択した仮想マシンを一時停止する前に、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 → Pause** をクリックします。

7.10.5. 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。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 一時停止を解除する仮想マシンが含まれる行を見つけます。

3. ユースケースに応じて適切なメニューに移動します。

   - このページに留まり、複数の仮想マシンに対して操作を実行するには、次の手順を実行します。

     a. 行の右端にある Options メニュー  をクリックして、**Unpause VirtualMachine** 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. **Actions → Unpause** をクリックします。

7.11. 仮想 TRUSTED PLATFORM MODULE デバイスの使用

VirtualMachine (VM) または **VirtualMachineInstance** (VMI) マニフェストを編集して、仮想 Trusted Platform Module (vTPM) デバイスを新規または既存の仮想マシンに追加します。

7.11.1. vTPM デバイスについて

仮想トラステッドプラットフォームモジュール (vTPM) デバイスは、物理トラステッドプラットフォームモジュール (TPM) ハードウェアチップのように機能します。

vTPM デバイスはどのオペレーティングシステムでも使用できますが、Windows 11 をインストールまたは起動するには TPM チップが必要です。vTPM デバイスを使用すると、Windows 11 イメージから作成された VM を物理 TPM チップなしで機能させることができます。

vTPM を有効にしないと、ノードに TPM デバイスがある場合でも、VM は TPM デバイスを認識しません。

また、vTPM デバイスは、物理ハードウェアを使用せずにシークレットを保存することで仮想マシンを保護します。OpenShift Virtualization は、仮想マシンの永続ボリューム要求 (PVC) を使用して、vTPM デバイス状態の永続化をサポートします。**HyperConverged** カスタムリソース (CR) で **vmStateStorageClass** 属性を設定することにより、PVC が使用するストレージクラスを指定する必要があります。

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  vmStateStorageClass: <storage_class_name>

# ...
```

注記

ストレージクラスは **Filesystem** タイプであり、**ReadWriteMany** (RWX) アクセスモードをサポートしている必要があります。

7.11.2. 仮想マシンへの vTPM デバイスの追加

仮想トラステッドプラットフォームモジュール (vTPM) デバイスを仮想マシン (VM) に追加すると、物理 TPM デバイスなしで Windows 11 イメージから作成された仮想マシンを実行できます。vTPM デバイスには、その仮想マシンのシークレットも保存されます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- **ReadWriteMany** (RWX) アクセスモードをサポートする **Filesystem** タイプのストレージクラスを使用するように永続ボリューム要求 (PVC) を設定しました。これは、仮想マシンの再起動後も vTPM デバイスデータを維持するために必要です。

手順

1. 次のコマンドを実行して、仮想マシン設定を更新します。

```
$ oc edit vm <vm_name> -n <namespace>
```

2. 仮想マシン仕様を編集して vTPM デバイスを追加します。以下に例を示します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          tpm: 1
            persistent: true 2
# ...
```

1. vTPM デバイスを仮想マシンに追加します。
2. 仮想マシンがシャットダウンされた後も vTPM デバイスの状態が維持されるように指定します。デフォルト値は **false** です。

3. 変更を適用するには、エディターを保存し、終了します。
4. オプション: 実行中の仮想マシンを編集している場合は、変更を有効にするためにこれを再起動する必要があります。

7.12. OPENSHIFT PIPELINES を使用した仮想マシンの管理

Red Hat OpenShift Pipelines は、開発者が独自のコンテナーで CI/CD パイプラインの各ステップを設計および実行できるようにする、Kubernetes ネイティブの CI/CD フレームワークです。

Scheduling、Scale、and Performance (SSP) Operator は、OpenShift Virtualization を OpenShift Pipelines と統合します。SSP Operator には、次のことを可能にするタスクとサンプルパイプラインが含まれています。

- 仮想マシン (VM)、永続ボリューム要求 (PVC)、およびデータボリュームの作成と管理

- 仮想マシンでコマンドを実行する
- **libguestfs** ツールを使用してディスクイメージを操作する

7.12.1. 前提条件

- **cluster-admin** 権限を使用して OpenShift Container Platform クラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Pipeline がインストールされている。

7.12.2. SSP Operator によってサポートされる仮想マシンのタスク

次の表は、SSP Operator の一部として含まれるタスクを示しています。

表7.4 SSP Operator によってサポートされる仮想マシンのタスク

タスク	説明
create-vm-from-manifest	提供されたマニフェストから、または **virtctl** を使用して仮想マシンを作成します。
create-vm-from-template	テンプレートからの仮想マシンの作成
copy-template	仮想マシンテンプレートをコピーします。
modify-vm-template	仮想マシンテンプレートを変更します。
modify-data-object	データボリュームまたはデータソースを作成または削除します。
cleanup-vm	仮想マシンでスクリプトまたはコマンドを実行し、後で仮想マシンを停止または削除します。
disk-virt-customize	**virt-customize** ツールを使用して、ターゲット PVC でカスタマイズスクリプトを実行します。
disk-virt-sysprep	**virt-sysprep** ツールを使用して、ターゲット PVC で sysprep スクリプトを実行します。
wait-for-vmi-status	仮想マシンインスタンスの特定のステータスを待機し、ステータスに基づいて失敗または成功します。

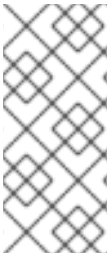
注記

パイプラインでの仮想マシンの作成では、非推奨になったテンプレートベースのタスクではなく、**ClusterInstanceType** と **ClusterPreference** が使用されるようになりました。**create-vm-from-template**、**copy-template**、および **modify-vm-template** コマンドは引き続き使用できますが、デフォルトのパイプラインタスクでは使用されません。

7.12.3. Windows EFI インストーラーパイプライン

Web コンソールまたは CLI を使用して、Windows EFI installer pipeline を実行できます。

Windows EFI インストーラーパイプラインは、Windows インストールイメージ (ISO ファイル) から Windows 10、Windows 11、または Windows Server 2022 を新しいデータボリュームにインストールします。インストールプロセスの実行には、カスタムアンサーファイルが使用されます。



注記

Windows EFI インストーラーパイプラインは、OpenShift Container Platform により事前定義された、Microsoft ISO ファイルに適した **sysprep** を含む config map ファイルを使用します。さまざまな Windows エディションに関連する ISO ファイルの場合は、システム固有の **sysprep** 定義を使用して新しい config map ファイルを作成することが必要になる場合があります。

7.12.3.1. Web コンソールを使用してサンプルパイプラインを実行する

サンプルパイプラインは、Web コンソールの **Pipelines** メニューから実行できます。

手順

1. サイドメニューの **Pipelines** → **Pipelines** をクリックします。
2. パイプラインを選択して、**Pipeline details** ページを開きます。
3. **Actions** リストから、**Start** を選択します。**Start Pipeline** ダイアログが表示されます。
4. パラメーターのデフォルト値を保持し、**Start** をクリックしてパイプラインを実行します。**Details** タブでは、各タスクの進行状況が追跡され、パイプラインのステータスが表示されます。

7.12.3.2. CLI を使用してサンプルパイプラインを実行する

PipelineRun リソースを使用して、サンプルパイプラインを実行します。**PipelineRun** オブジェクトは、パイプラインの実行中のインスタンスです。これは、クラスター上の特定の入力、出力、および実行パラメーターで実行されるパイプラインをインスタンス化します。また、パイプライン内のタスクごとに **TaskRun** オブジェクトを作成します。

手順

1. Windows 10 インストーラーパイプラインを実行するには、次の **PipelineRun** マニフェストを作成します。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: windows10-installer-run-
  labels:
    pipelinerun: windows10-installer-run
spec:
  params:
  - name: winImageDownloadURL
    value: <link_to_windows_10_iso> (1)
  pipelineRef:
```

```
    name: windows10-installer
  taskRunSpecs:
    - pipelineTaskName: copy-template
      taskServiceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template
      taskServiceAccountName: modify-vm-template-task
    - pipelineTaskName: create-vm-from-template
      taskServiceAccountName: create-vm-from-template-task
    - pipelineTaskName: wait-for-vmi-status
      taskServiceAccountName: wait-for-vmi-status-task
    - pipelineTaskName: create-base-dv
      taskServiceAccountName: modify-data-object-task
    - pipelineTaskName: cleanup-vm
      taskServiceAccountName: cleanup-vm-task
  status: {}
```

1 Windows 10 64 ビット ISO ファイルの URL を指定します。製品の言語は英語 (米国) でなければなりません。

2. **PipelineRun** マニフェストを適用します。

```
$ oc apply -f windows10-installer-run.yaml
```

3. Windows 10 カスタマイズパイプラインを実行するには、次の **PipelineRun** マニフェストを作成します。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: windows10-customize-run-
  labels:
    pipelinerun: windows10-customize-run
spec:
  params:
    - name: allowReplaceGoldenTemplate
      value: true
    - name: allowReplaceCustomizationTemplate
      value: true
  pipelineRef:
    name: windows10-customize
  taskRunSpecs:
    - pipelineTaskName: copy-template-customize
      taskServiceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template-customize
      taskServiceAccountName: modify-vm-template-task
    - pipelineTaskName: create-vm-from-template
      taskServiceAccountName: create-vm-from-template-task
    - pipelineTaskName: wait-for-vmi-status
      taskServiceAccountName: wait-for-vmi-status-task
    - pipelineTaskName: create-base-dv
      taskServiceAccountName: modify-data-object-task
    - pipelineTaskName: cleanup-vm
      taskServiceAccountName: cleanup-vm-task
    - pipelineTaskName: copy-template-golden
      taskServiceAccountName: copy-template-task
```

```
    - pipelineTaskName: modify-vm-template-golden
      taskServiceAccountName: modify-vm-template-task
status: {}
```

4. **PipelineRun** マニフェストを適用します。

```
$ oc apply -f windows10-customize-run.yaml
```

7.12.4. 関連情報

- Red Hat OpenShift Pipelines を使用してアプリケーションの CI/CD ソリューションを作成する
- Windows 仮想マシンの作成

7.13. 高度な仮想マシン管理

7.13.1. 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

7.13.1.1. 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。

7.13.1.2. 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

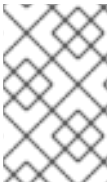
7.13.2. 仮想マシンのノードの指定

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。

7.13.2.1. 仮想マシンのノード配置について

仮想マシン (VM) が適切なノードで実行されるようにするには、ノードの配置ルールを設定できます。以下の場合にこれを行うことができます。

- 仮想マシンが複数ある。フォールトトレランスを確保するために、これらを異なるノードで実行する必要がある。
- 2 つの相互間のネットワークトラフィックの多い chatty VM がある。冗長なノード間のルーティングを回避するには、仮想マシンを同じノードで実行します。
- 仮想マシンには、利用可能なすべてのノードにない特定のハードウェア機能が必要です。
- 機能をノードに追加する Pod があり、それらの機能を使用できるように仮想マシンをそのノードに配置する必要があります。



注記

仮想マシンの配置は、ワークロードの既存のノードの配置ルールに基づきます。ワークロードがコンポーネントレベルの特定のノードから除外される場合、仮想マシンはそれらのノードに配置できません。

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

nodeSelector

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

affinity

より表現的な構文を使用して、ノードと仮想マシンに一致するルールを設定できます。たとえば、ルールがハード要件ではなく基本設定になるように指定し、ルールの条件が満たされない場合も仮想マシンがスケジュールされるようにすることができます。Pod のアフィニティー、Pod の非アフィニティー、およびノードのアフィニティーは仮想マシンの配置でサポートされます。Pod のアフィニティーは仮想マシンに対して動作します。**VirtualMachine** ワークロードタイプは **Pod** オブジェクトに基づくためです。

toleration

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。

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。

7.13.2.2. 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。

7.13.2.2.1. 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
# ...
```

7.13.2.2.2. 例: Pod のアフィニティーおよび Pod の非アフィニティーによる仮想マシンノードの配置

この例では、仮想マシンはラベル **example-key-1 = example-value-1** を持つ実行中の Pod のあるノードでスケジュールされる必要があります。このようなノードで実行中の Pod がない場合、仮想マシンはスケジュールされません。

可能な場合に限り、仮想マシンはラベル **example-key-2 = example-value-2** を持つ Pod のあるノードではスケジュールされません。ただし、すべての候補となるノードにこのラベルを持つ Pod がある場合、スケジューラーはこの制約を無視します。

仮想マシンマニフェストの例

```
metadata:
  name: example-vm-pod-affinity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      affinity:
        podAffinity:
          requiredDuringSchedulingIgnoredDuringExecution: 1
```

```
        - labelSelector:
            matchExpressions:
            - key: example-key-1
              operator: In
              values:
              - example-value-1
          topologyKey: kubernetes.io/hostname
      podAntiAffinity:
        preferredDuringSchedulingIgnoredDuringExecution: 2
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchExpressions:
              - key: example-key-2
                operator: In
                values:
                - example-value-2
            topologyKey: kubernetes.io/hostname
# ...
```

1 **requiredDuringSchedulingIgnoredDuringExecution** ルールタイプを使用する場合、制約を満たさない場合には仮想マシンはスケジュールされません。

2 **preferredDuringSchedulingIgnoredDuringExecution** ルールタイプを使用する場合、この制約を満たさない場合でも、必要なすべての制約を満たす場合に仮想マシンは依然としてスケジュールされます。

7.13.2.2.3. 例: ノードのアフィニティーによる仮想マシンノードの配置

この例では、仮想マシンはラベル **example.io/example-key = example-value-1** またはラベル **example.io/example-key = example-value-2** を持つノードでスケジュールされる必要があります。この制約は、ラベルのいずれかがノードに存在する場合に満たされます。いずれのラベルも存在しない場合、仮想マシンはスケジュールされません。

可能な場合、スケジューラーはラベル **example-node-label-key = example-node-label-value** を持つノードを回避します。ただし、すべての候補となるノードにこのラベルがある場合、スケジューラーはこの制約を無視します。

仮想マシンマニフェストの例

```
metadata:
  name: example-vm-node-affinity
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution: 1
            nodeSelectorTerms:
            - matchExpressions:
              - key: example.io/example-key
                operator: In
```

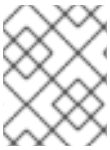
```
            values:
            - example-value-1
            - example-value-2
        preferredDuringSchedulingIgnoredDuringExecution: 2
        - weight: 1
          preference:
            matchExpressions:
            - key: example-node-label-key
              operator: In
              values:
              - example-node-label-value
# ...
```

1 **requiredDuringSchedulingIgnoredDuringExecution** ルールタイプを使用する場合、制約を満たさない場合には仮想マシンはスケジュールされません。

2 **preferredDuringSchedulingIgnoredDuringExecution** ルールタイプを使用する場合、この制約を満たさない場合でも、必要なすべての制約を満たす場合に仮想マシンは依然としてスケジュールされます。

7.13.2.2.4. 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"
# ...
```

7.13.2.3. 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 taint を使用した Pod 配置の制御

7.13.3. kernel samepage merging (KSM) のアクティブ化

OpenShift Virtualization は、ノードが過負荷になると kernel samepage merging (KSM) をアクティブ化できます。KSM は、仮想マシンのメモリーページにある同一データの重複を排除します。非常によく似た仮想マシンがある場合に KSM を使用すると、シングルノード上で多くの仮想マシンをスケジュールできるようになります。

重要

KSM は、必ず信頼できるワークロードでのみ使用してください。

7.13.3.1. 前提条件

- OpenShift Virtualization が KSM をアクティブ化するノード上で、管理者が KSM サポートを設定していることを確認する。

7.13.3.2. OpenShift Virtualization を使用して KSM をアクティブ化する

ノードでメモリーの過負荷が発生した場合に kernel samepage merging (KSM) をアクティブ化するように、OpenShift Virtualization を設定できます。

7.13.3.2.1. 設定方法

OpenShift Container Platform Web コンソールを使用するか、**HyperConverged** カスタムリソース (CR) を編集することで、すべてのノードの KSM アクティブ化機能を有効または無効にできます。**HyperConverged** CR は、より詳細な設定をサポートしています。

CR 設定
HyperConverged CR の **spec.configuration.ksmConfiguration** スタンザを編集することで、KSM アクティブ化機能を設定できます。

- この機能を有効にして設定するには、**ksmConfiguration** スタンザを編集します。
- この機能を無効にするには、**ksmConfiguration** スタンザを削除します。
- ノード選択構文を **ksmConfiguration.nodeLabelSelector** フィールドに追加すると、OpenShift Virtualization はノードのサブセットのみで KSM を有効化できます。

注記

管理者は、OpenShift Virtualization で KSM アクティブ化機能が無効になっている場合でも、それをサポートするノードでは KSM を有効化できます。

7.13.3.2.2. KSM ノードのラベル

OpenShift Virtualization は、KSM をサポートするように設定されたノードを識別し、次のノードラベルを適用します。

kubevirt.io/ksm-handler-managed: "false"
メモリーの過負荷が発生しているノード上で OpenShift Virtualization が KSM をアクティブ化すると、このラベルが **"true"** に設定されます。管理者が KSM をアクティブ化した場合、このラベルは **"true"** に設定されません。

kubevirt.io/ksm-enabled: "false"

OpenShift Virtualization が KSM をアクティブ化しなかった場合でも、KSM がノード上でアクティブ化されると、このラベルは **"true"** に設定されます。

このラベルは、KSM をサポートしていないノードには適用されません。

7.13.3.3. Web コンソールを使用して KSM のアクティブ化を設定する

OpenShift Container Platform Web コンソールを使用して、OpenShift Virtualization がクラスター内のすべてのノードで kernel samepage merging (KSM) をアクティブ化できるように設定できます。

手順

1. サイドメニューから、**Virtualization → Overview** をクリックします。
2. **Settings** タブを選択します。
3. **Cluster** タブを選択します。
4. **Resource management** を展開します。
5. すべてのノードの機能を有効または無効にします。
   - **Kernel Samepage Merging (KSM)** をオンに設定します。
   - **Kernel Samepage Merging (KSM)** をオフに設定します。

7.13.3.4. CLI を使用して KSM のアクティブ化を設定する

HyperConverged カスタムリソース (CR) を編集することで、OpenShift Virtualization の kernel samepage merging (KSM) アクティブ化機能を有効または無効にできます。OpenShift Virtualization がノードのサブセットのみで KSM をアクティブ化するように設定する場合は、この方法を使用します。

手順

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

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

2. **ksmConfiguration** スタンザを編集します。
   - すべてのノードで KSM アクティブ化機能を有効にするには、**nodeLabelSelector** の値を **{}** に設定します。以下に例を示します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  configuration:
    ksmConfiguration:
      nodeLabelSelector: {}
# ...
```

- ノードのサブセットで KSM アクティブ化機能を有効にするには、**nodeLabelSelector** フィールドを編集します。OpenShift Virtualization が KSM を有効にするノードに一致する構文を追加します。たとえば次の設定では、OpenShift Virtualization は **<first_example_key>** と **<second_example_key>** の両方が **"true"** に設定されているノードで KSM を有効にできます。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  configuration:
    ksmConfiguration:
      nodeLabelSelector:
        matchLabels:
          <first_example_key>: "true"
          <second_example_key>: "true"
# ...
```

- KSM アクティブ化機能を無効にするには、**ksmConfiguration** スタンザを削除します。以下に例を示します。

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

3. ファイルを保存します。

7.13.3.5. 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
- Red Hat Enterprise Linux (RHEL) ドキュメントの kernel samepage merging の管理

7.13.4. 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。

7.13.4.1. 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 hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

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。

7.13.4.2. 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。

7.13.5. デフォルトの CPU モデルの設定

HyperConverged カスタムリソース (CR) の **defaultCPUModel** 設定を使用して、クラスター全体のデフォルト CPU モデルを定義します。

仮想マシン (VM) の CPU モデルは、仮想マシンおよびクラスター内の CPU モデルの可用性によって異なります。

- 仮想マシンに定義された CPU モデルがない場合:
  - **defaultCPUModel** は、クラスター全体のレベルで定義された CPU モデルを使用して自動的に設定されます。
- 仮想マシンとクラスターの両方に CPU モデルが定義されている場合:
  - 仮想マシンの CPU モデルが優先されます。
- 仮想マシンにもクラスターにも CPU モデルが定義されていない場合:
  - ホストモデルは、ホストレベルで定義された CPU モデルを使用して自動的に設定されます。

7.13.5.1. デフォルトの CPU モデルの設定

HyperConverged カスタムリソース (CR) を更新して、 **defaultCPUModel** を設定します。OpenShift Virtualization の実行中に、**defaultCPUModel** を変更できます。

注記

defaultCPUModel では、大文字と小文字が区別されます。

前提条件

- OpenShift CLI (oc) のインストール。

手順

1. 以下のコマンドを実行して **HyperConverged** CR を開きます。

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

2. CR に **defaultCPUModel** フィールドを追加し、値をクラスター内に存在する CPU モデルの名前に設定します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
```

```
metadata:
 name: kubevirt-hyperconverged
 namespace: openshift-cnv
spec:
  defaultCPUModel: "EPYC"
```

3. YAML ファイルをクラスターに適用します。

7.13.6. 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。

7.13.6.1. 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。

7.13.6.2. 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
```

7.13.6.3. 永続的な EFI の有効化

クラスターレベルで RWX ストレージクラスを設定し、仮想マシンの EFI セクションで設定を調整することで、仮想マシンで EFI 永続性を有効にできます。

前提条件

- クラスター管理者の権限がある。
- RWX アクセスモードと FS ボリュームモードをサポートするストレージクラスが必要です。

手順

- 次のコマンドを実行して、**VMPersistentState** フィーチャーゲートを有効にします。

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

7.13.6.4. 永続的な EFI を使用した仮想マシンの設定

マニフェストファイルを編集して、EFI の永続性を有効にするように仮想マシンを設定できます。

前提条件

- **VMPersistentState** フィーチャーゲートが有効になっている。

手順

- 仮想マシンマニフェストファイルを編集して保存し、設定を適用します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm
spec:
  template:
    spec:
      domain:
        firmware:
          bootloader:
            efi:
              persistent: true
# ...
```

7.13.7. 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。

7.13.7.1. 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。

7.13.7.2. 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
```

5. 仮想マシンインスタンスの実行を待機します。

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

6. VNC を使用して仮想マシンインスタンスを表示します。

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

7. ブート画面で、PXE ブートが正常に実行されていることを確認します。

8. 仮想マシンインスタンスにログインします。

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

検証

1. 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
```

7.13.7.3. OpenShift Virtualization ネットワークの用語集

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)

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)

Multus プロジェクトによって導入された CRD。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接続できるようにします。

ノードネットワーク設定ポリシー (NNCP)

nmstate プロジェクトによって導入された CRD。ノード上で要求されるネットワーク設定を表します。**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。

7.13.8. 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。

7.13.8.1. 前提条件

- ノードに 事前に割り当てられた huge page が設定されている 。

7.13.8.2. Huge Page の機能

メモリーは Page と呼ばれるブロックで管理されます。多くのシステムでは、1 ページは 4Ki です。メモリー 1Mi は 256 ページに、メモリー 1Gi は 256,000 ページに相当します。CPU には、内蔵のメモリー管理ユニットがあり、ハードウェアでこのようなページリストを管理します。トランスレーションルックアサイドバッファー (TLB: Translation Lookaside Buffer) は、仮想から物理へのページマッピングの小規模なハードウェアキャッシュのことです。ハードウェアの指示で渡された仮想アドレスが TLB にあれば、マッピングをすばやく決定できます。そうでない場合には、TLB ミスが発生し、システムは速度が遅く、ソフトウェアベースのアドレス変換にフォールバックされ、パフォーマンスの問題が発生します。TLB のサイズは固定されているので、TLB ミスの発生率を減らすには Page サイズを大きくする必要があります。

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

フラグが原因で、メモリー使用率が高くなり、断片化が起こり、パフォーマンスの低下につながり、メモリーページがロックされてしまう可能性があります。このような理由から、アプリケーションは THP ではなく、事前割り当て済みの Huge Page を使用するように設計 (また推奨) される場合があります。

OpenShift Virtualization では、事前に割り当てられた Huge Page を使用できるように仮想マシンを設定できます。

7.13.8.3. 仮想マシンの Huge Page の設定

memory.hugepages.pageSize および **resources.requests.memory** パラメーターを仮想マシン設定に組み込み、仮想マシンを事前に割り当てられた Huge Page を使用するように設定できます。

メモリー要求はページサイズ別に分ける必要があります。たとえば、ページサイズ **1Gi** の場合に **500Mi** メモリーを要求することはできません。

注記

ホストおよびゲスト OS のメモリーレイアウトには関連性がありません。仮想マシンマニフェストで要求される Huge Page が QEMU に適用されます。ゲスト内の Huge Page は、仮想マシンインスタンスの利用可能なメモリー量に基づいてのみ設定できます。

実行中の仮想マシンを編集する場合は、変更を有効にするために仮想マシンを再起動する必要があります。

前提条件

- ノードには、事前に割り当てられた Huge Page が設定されている必要がある。

手順

1. 仮想マシン設定で、**resources.requests.memory** および **memory.hugepages.pageSize** パラメーターを **spec.domain** に追加します。以下の設定スニペットは、ページサイズが **1Gi** の合計 **4Gi** メモリーを要求する仮想マシンに関するものです。

```
kind: VirtualMachine
# ...
spec:
  domain:
    resources:
      requests:
        memory: "4Gi" (1)
    memory:
      hugepages:
        pageSize: "1Gi" (2)
# ...
```

(1) 仮想マシンに要求されるメモリーの合計量。この値はページサイズで分ける必要があります。

(2) 各 Huge Page のサイズ。x86_64 アーキテクチャーの有効な値は **1Gi** および **2Mi** です。ページサイズは要求されたメモリーよりも小さくなければなりません。

2. 仮想マシン設定を適用します。

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

7.13.9. 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。

7.13.9.1. 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。

7.13.9.2. 前提条件

- CPU マネージャー がノードに設定されている。仮想マシンのワークロードをスケジュールする前に、ノードに **cpumanager = true** ラベルが設定されていることを確認する。
- 仮想マシンの電源がオフになっている。

7.13.9.3. 仮想マシンの専用リソースの有効化

Details タブで、仮想マシンの専用リソースを有効にすることができます。Red Hat テンプレートから作成された仮想マシンは、専用のリソースで設定できます。

手順

1. OpenShift Container Platform コンソールで、サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Configuration → Scheduling** タブで、**Dedicated Resources** の横にある編集アイコンをクリックします。
4. **Schedule this workload with dedicated resources (guaranteed policy)**を選択します。
5. **Save** をクリックします。

7.13.10. 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。

7.13.10.1. ポリシー属性

仮想マシン (VM) をスケジュールするには、ポリシー属性と、仮想マシンがノードでスケジュールされる際の互換性について一致する CPU 機能を指定します。仮想マシンに指定されるポリシー属性は、その仮想マシンをノードにスケジュールする方法を決定します。

ポリシー属性	説明

ポリシー属性	説明
force	仮想マシンは強制的にノードでスケジュールされます。これは、ホストの CPU が仮想マシンの CPU に対応していない場合でも該当します。
require	仮想マシンが特定の CPU モデルおよび機能仕様で設定されていない場合に仮想マシンに適用されるデフォルトのポリシー。このデフォルトポリシー属性または他のポリシー属性のいずれかを持つ CPU ノードの検出をサポートするようにノードが設定されていない場合、仮想マシンはそのノードでスケジュールされません。ホストの CPU が仮想マシンの CPU をサポートしているか、ハイパーバイザーが対応している CPU モデルをエミュレートできる必要があります。
optional	仮想マシンがホストの物理マシンの CPU でサポートされている場合は、仮想マシンがノードに追加されます。
disable	仮想マシンは CPU ノードの検出機能と共にスケジュールすることはできません。
forbid	この機能がホストの CPU でサポートされ、CPU ノード検出が有効になっている場合でも、仮想マシンはスケジュールされません。

7.13.10.2. 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。

7.13.10.3. 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。

7.13.10.4. ホストモデルでの仮想マシンのスケジューリング

仮想マシン (VM) の CPU モデルが **host-model** に設定されている場合、仮想マシンはスケジュールされているノードの CPU モデルを継承します。

手順

- 仮想マシン設定ファイルの **domain** 仕様を編集します。以下の例は、仮想マシンに指定される **host-model** を示しています。

```
apiVersion: kubevirt/v1alpha3
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: host-model 1
```

1 スケジュールされるノードの CPU モデルを継承する仮想マシン。

7.13.10.5. カスタムスケジューラーを使用した仮想マシンのスケジュール設定

カスタムスケジューラーを使用して、ノード上の仮想マシンをスケジュールできます。

前提条件

- セカンダリースケジューラーがクラスター用に設定されています。

手順

- **VirtualMachine** マニフェストを編集して、カスタムスケジューラーを仮想マシン設定に追加します。以下に例を示します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
spec:
  running: true
  template:
    spec:
      schedulerName: my-scheduler (1)
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
# ...
```

(1) カスタムスケジューラーの名前。**schedulerName** 値が既存のスケジューラーと一致しない場合、**virt-launcher** Pod は、指定されたスケジューラーが見つかるまで **Pending** 状態のままになります。

検証

- **virt-launcher** Pod イベントをチェックして、仮想マシンが **VirtualMachine** マニフェストで指定されたカスタムスケジューラーを使用していることを確認します。

  a. 次のコマンドを入力して、クラスター内の Pod のリストを表示します。

  ```
  $ oc get pods
  ```

  出力例

  ```
  NAME                            READY   STATUS    RESTARTS   AGE
  virt-launcher-vm-fedora-dpc87   2/2     Running   0          24m
  ```

  b. 次のコマンドを実行して Pod イベントを表示します。

  ```
  $ oc describe pod virt-launcher-vm-fedora-dpc87
  ```

  出力の **From** フィールドの値により、スケジューラー名が **VirtualMachine** マニフェストで指定されたカスタムスケジューラーと一致することが検証されます。

  出力例

  ```
  [...]
  Events:
  ```

```
  Type    Reason     Age   From            Message
  ----    ------     ----  ----            -------
  Normal  Scheduled  21m   my-scheduler  Successfully assigned default/virt-launcher-
vm-fedora-dpc87 to node01
[...]
```

関連情報

- セカンダリースケジューラーのデプロイ

7.13.11. 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。

7.13.11.1. GPU パススルー用のノードの準備

GPU パススルー用に指定したワーカーノードに GPU オペランドがデプロイされないようにすることができます。

7.13.11.1.1. NVIDIA GPU オペランドがノードにデプロイメントされないようにする

クラスター内で NVIDIA GPU Operator を使用する場合は、GPU または vGPU オペランド用に設定したくないノードに **nvidia.com/gpu.deploy.operands=false** ラベルを適用できます。このラベルは、GPU または vGPU オペランドを設定する Pod の作成を防止し、Pod がすでに存在する場合は終了します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

- 次のコマンドを実行して、ノードのラベルを付けます。

```
$ oc label node <node_name> nvidia.com/gpu.deploy.operands=false 1
```

1. **<node_name>** を、NVIDIA GPU オペランドをインストールしないノードの名前に置き換えます。

検証

1. 次のコマンドを実行して、ラベルがノードに追加されたことを確認します。

```
$ oc describe node <node_name>
```

2. オプション: GPU オペランドが以前にノードにデプロイされていた場合は、それらの削除を確認します。

a. 次のコマンドを実行して、**nvidia-gpu-operator** namespace 内の Pod のステータスを確認します。

```
$ oc get pods -n nvidia-gpu-operator
```

出力例

```
NAME                            READY   STATUS        RESTARTS   AGE
gpu-operator-59469b8c5c-hw9wj    1/1     Running       0          8d
nvidia-sandbox-validator-7hx98   1/1     Running       0          8d
nvidia-sandbox-validator-hdb7p   1/1     Running       0          8d
nvidia-sandbox-validator-kxwj7   1/1     Terminating   0          9d
nvidia-vfio-manager-7w9fs        1/1     Running       0          8d
nvidia-vfio-manager-866pz        1/1     Running       0          8d
nvidia-vfio-manager-zqtck        1/1     Terminating   0          9d
```

b. **Termination** ステータスの Pod が削除されるまで、Pod のステータスを監視します。

```
$ oc get pods -n nvidia-gpu-operator
```

出力例

```
NAME                            READY   STATUS    RESTARTS   AGE
gpu-operator-59469b8c5c-hw9wj    1/1     Running   0          8d
nvidia-sandbox-validator-7hx98   1/1     Running   0          8d
nvidia-sandbox-validator-hdb7p   1/1     Running   0          8d
nvidia-vfio-manager-7w9fs        1/1     Running   0          8d
nvidia-vfio-manager-866pz        1/1     Running   0          8d
```

7.13.11.2. PCI パススルー用のホストデバイスの準備

7.13.11.2.1. PCI パススルー用ホストデバイスの準備について

CLI を使用して PCI パススルー用にホストデバイスを準備するには、**MachineConfig** オブジェクトを作成し、カーネル引数を追加して、Input-Output Memory Management Unit (IOMMU) を有効にします。PCI デバイスを Virtual Function I/O (VFIO) ドライバーにバインドしてから、**HyperConverged** カスタムリソース (CR) の **permittedHostDevices** フィールドを編集してクラスター内で公開します。OpenShift Virtualization Operator を最初にインストールする場合、**permittedHostDevices** のリストは空になります。

CLI を使用してクラスターから PCI ホストデバイスを削除するには、**HyperConverged** CR から PCI デバイス情報を削除します。

7.13.11.2.2. IOMMU ドライバーを有効にするためのカーネル引数の追加

カーネルで IOMMU 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

- クラスター管理者パーミッションがある。
- CPU ハードウェアは Intel または AMD です。

- BIOS で Directed I/O 拡張機能または AMD IOMMU 用の Intel Virtualization Technology を有効にしました。

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
```

7.13.11.2.3. PCI デバイスの VFIO ドライバーへのバインディング

PCI デバイスを VFIO (Virtual Function I/O) ドライバーにバインドするには、各デバイスから **vendor-ID** および **device-ID** の値を取得し、これらの値でリストを作成します。リストを **MachineConfig** オブジェクトに追加します。**MachineConfig** Operator は、PCI デバイスを持つノードで **/etc/modprobe.d/vfio.conf** を生成し、PCI デバイスを VFIO ドライバーにバインドします。

前提条件

- カーネル引数を CPU の IOMMU を有効にするために追加している。

手順

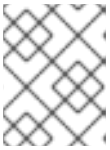
1. **lspci** コマンドを実行して、PCI デバイスの **vendor-ID** および **device-ID** を取得します。

```
$ lspci -nnv | grep -i nvidia
```

出力例

```
02:01.0 3D controller [0302]: NVIDIA Corporation GV100GL [Tesla V100 PCIe 32GB]
[10de:1eb8] (rev a1)
```

2. Butane 設定ファイル **100-worker-vfiopci.bu** を作成し、PCI デバイスを VFIO ドライバーにバインドします。



注記

Butane の詳細は、「Butane を使用したマシン設定の作成」を参照してください。

例

```
variant: openshift
version: 4.16.0
metadata:
  name: 100-worker-vfiopci
  labels:
    machineconfiguration.openshift.io/role: worker (1)
storage:
  files:
  - path: /etc/modprobe.d/vfio.conf
    mode: 0644
    overwrite: true
    contents:
      inline: |
        options vfio-pci ids=10de:1eb8 (2)
  - path: /etc/modules-load.d/vfio-pci.conf (3)
    mode: 0644
    overwrite: true
    contents:
      inline: vfio-pci
```

(1) 新しいカーネル引数をワーカーノードのみに適用します。

(2) 以前に決定された **vendor-ID** 値 (**10de**) と **device-ID** 値 (**1eb8**) を指定して、単一のデバイスを VFIO ドライバーにバインドします。複数のデバイスのリストをベンダーおよびデバイス情報とともに追加できます。

(3) ワーカーノードで vfio-pci カーネルモジュールを読み込むファイル。

3. Butane を使用して、ワーカーノードに配信される設定を含む **MachineConfig** オブジェクトファイル (**100-worker-vfiopci.yaml**) を生成します。

```
$ butane 100-worker-vfiopci.bu -o 100-worker-vfiopci.yaml
```

4. **MachineConfig** オブジェクトをワーカーノードに適用します。

```
$ oc apply -f 100-worker-vfiopci.yaml
```

5. **MachineConfig** オブジェクトが追加されていることを確認します。

```
$ oc get MachineConfig
```

出力例

```
NAME                             GENERATEDBYCONTROLLER                      IGNITIONVERSION
AGE
00-master                        d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0             25h
00-worker                        d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0             25h
01-master-container-runtime      d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-master-kubelet                d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-worker-container-runtime      d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
01-worker-kubelet                d3da910bfa9f4b599af4ed7f5ac270d55950a3a1   3.2.0
25h
100-worker-iommu                                                            3.2.0             30s
100-worker-vfiopci-configuration                                            3.2.0             30s
```

検証

- VFIO ドライバーがロードされていることを確認します。

```
$ lspci -nnk -d 10de:
```

この出力では、VFIO ドライバーが使用されていることを確認します。

出力例

```
04:00.0 3D controller [0302]: NVIDIA Corporation GP102GL [Tesla P40] [10de:1eb8] (rev a1)
        Subsystem: NVIDIA Corporation Device [10de:1eb8]
        Kernel driver in use: vfio-pci
        Kernel modules: nouveau
```

7.13.11.2.4. CLI を使用したクラスターでの PCI ホストデバイスの公開

クラスターで PCI ホストデバイスを公開するには、PCI デバイスの詳細を **HyperConverged** カスタムリソース (CR) の **spec.permittedHostDevices.pciHostDevices** 配列に追加します。

手順

1. 以下のコマンドを実行して、デフォルトエディターで **HyperConverged** CR を編集します。

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

2. PCI デバイス情報を **spec.permittedHostDevices.pciHostDevices** 配列に追加します。以下に例を示します。

設定ファイルのサンプル

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  permittedHostDevices: 1
    pciHostDevices: 2
    - pciDeviceSelector: "10DE:1DB6" 3
      resourceName: "nvidia.com/GV100GL_Tesla_V100" 4
    - pciDeviceSelector: "10DE:1EB8"
      resourceName: "nvidia.com/TU104GL_Tesla_T4"
    - pciDeviceSelector: "8086:6F54"
      resourceName: "intel.com/qat"
      externalResourceProvider: true 5
# ...
```

1. クラスターでの使用が許可されているホストデバイス。
2. ノードで利用可能な PCI デバイスのリスト。
3. PCI デバイスを識別するために必要な **vendor-ID** および **device-ID**。
4. PCI ホストデバイスの名前。
5. オプション: このフィールドを **true** に設定すると、リソースが外部デバイスプラグインにより提供されることを示します。OpenShift Virtualization はクラスターでこのデバイスの使用を許可しますが、割り当ておよびモニタリングを外部デバイスプラグインに残します。

注記

上記のスニペットの例は、**nvidia.com/GV100GL_Tesla_V100** および **nvidia.com/TU104GL_Tesla_T4** という名前の 2 つの PCI ホストデバイスが、**HyperConverged** CR の許可されたホストデバイスの一覧に追加されたことを示しています。これらのデバイスは、OpenShift Virtualization と動作することがテストおよび検証されています。

3. 変更を保存し、エディターを終了します。

検証

- 以下のコマンドを実行して、PCI ホストデバイスがノードに追加されたことを確認します。この出力例は、各デバイスが **nvidia.com/GV100GL_Tesla_V100**、**nvidia.com/TU104GL_Tesla_T4**、および **intel.com/qat** のリソース名にそれぞれ関連付けられたデバイスが 1 つあることを示しています。

```
$ oc describe node <node_name>
```

出力例

-

```
Capacity:
  cpu:                          64
  devices.kubevirt.io/kvm:      110
  devices.kubevirt.io/tun:      110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:            915128Mi
  hugepages-1Gi:                0
  hugepages-2Mi:                0
  memory:                       131395264Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                1
  pods:                         250
Allocatable:
  cpu:                          63500m
  devices.kubevirt.io/kvm:      110
  devices.kubevirt.io/tun:      110
  devices.kubevirt.io/vhost-net:  110
  ephemeral-storage:            863623130526
  hugepages-1Gi:                0
  hugepages-2Mi:                0
  memory:                       130244288Ki
  nvidia.com/GV100GL_Tesla_V100   1
  nvidia.com/TU104GL_Tesla_T4     1
  intel.com/qat:                1
  pods:                         250
```

7.13.11.2.5. 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
```

7.13.11.3. 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。

7.13.11.3.1. 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)
```

7.13.11.4. 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
- ファイル権限の管理
- マシン設定の概要

7.13.12. 仮想 GPU の設定

グラフィックスプロセッシングユニット (GPU) カードがある場合、OpenShift Virtualization は仮想マシンに割り当てることができる仮想 GPU (vGPU) を自動的に作成できます。

7.13.12.1. OpenShift Virtualization での仮想 GPU の使用について

一部のグラフィックス処理ユニット (GPU) カードは、仮想 GPU (vGPU) の作成をサポートしています。管理者が **HyperConverged** カスタムリソース (CR) で設定の詳細を提供すると、OpenShift Virtualization は仮想 GPU およびその他の仲介デバイスを自動的に作成できます。この自動化は、大規模なクラスターで特に役立ちます。

注記

機能とサポートの詳細は、ハードウェアベンダーのドキュメントを参照してください。

仲介デバイス

1 つまたは複数の仮想デバイスに分割された物理デバイス。仮想 GPU は、仲介デバイス (mdev) の一種です。物理 GPU のパフォーマンスが、仮想デバイス間で分割されます。仲介デバイスを 1 つまたは複数の仮想マシン (VM) に割り当てることができますが、ゲストの数は GPU と互換性がある必要があります。一部の GPU は複数のゲストをサポートしていません。

7.13.12.2. 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。

7.13.12.2.1. IOMMU ドライバーを有効にするためのカーネル引数の追加

カーネルで IOMMU 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

- クラスター管理者パーミッションがある。
- CPU ハードウェアは Intel または AMD です。
- BIOS で Directed I/O 拡張機能または AMD IOMMU 用の Intel Virtualization Technology を有効にしました。

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
```

7.13.12.3. NVIDIA GPU Operator の設定

NVIDIA GPU Operator を使用して、OpenShift Virtualization で GPU 高速化仮想マシンを実行するためのワーカーノードをプロビジョニングできます。

注記

NVIDIA GPU Operator は、NVIDIA によってのみサポートされています。詳細は、Red Hat ナレッジベースの Obtaining Support from NVIDIA を参照してください。

7.13.12.3.1. NVIDIA GPU Operator の使用について

NVIDIA GPU Operator と OpenShift Virtualization を使用すると、GPU 対応の仮想マシンを実行するためのワーカーノードを迅速にプロビジョニングできます。NVIDIA GPU Operator は、OpenShift Container Platform クラスター内の NVIDIA GPU リソースを管理し、GPU ワークロード用にノードを準備するときに必要なタスクを自動化します。

アプリケーションワークロードを GPU リソースにデプロイする前に、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や監視などのその他の機能を有効にする NVIDIA ドライバーなどのコンポーネントをインストールする必要があります。これらのタスクを自動化することで、インフラストラクチャーの GPU 容量を迅速に拡張できます。NVIDIA GPU Operator は、複雑な人工知能および機械学習 (AI/ML) ワークロードのプロビジョニングを特に容易にします。

7.13.12.3.2. 仲介デバイスを設定するためのオプション

NVIDIA GPU Operator を使用すると、仲介デバイスを設定するには 2 つの方法があります。Red Hat がテストする方法では、OpenShift Virtualization 機能を使用して仲介デバイスをスケジュールしますが、NVIDIA の方法では GPU Operator のみを使用します。

NVIDIA GPU Operator を使用した仲介デバイスの設定

この方法では、NVIDIA GPU Operator のみを使用して仲介デバイスを設定します。この方法を使用するには、NVIDIA ドキュメントの NVIDIA GPU Operator with OpenShift Virtualization を参照してください。

OpenShift Virtualization を使用した仲介デバイスの設定

この方法は Red Hat によってテストされており、OpenShift Virtualization の機能を使用して仲介デバイスを設定します。この場合、NVIDIA GPU Operator は、NVIDIA vGPU Manager でドライバーをインストールするためにのみ使用されます。GPU Operator は仲介デバイスを設定しません。OpenShift 仮想化方式を使用する場合でも、NVIDIA ドキュメント に従って GPU Operator を設定します。ただし、この方法は次の点で NVIDIA ドキュメントとは異なります。

- **HyperConverged** カスタムリソース (CR) のデフォルトの **disableMDEVConfiguration: false** 設定を上書きしないでください。

重要

NVIDIA ドキュメント の説明に従ってこの機能ゲートを設定すると、OpenShift Virtualization が仲介デバイスを設定できなくなります。

- 次の例と一致するように **ClusterPolicy** マニフェストを設定する必要があります。

マニフェストの例

```
kind: ClusterPolicy
apiVersion: nvidia.com/v1
metadata:
  name: gpu-cluster-policy
spec:
  operator:
    defaultRuntime: crio
    use_ocp_driver_toolkit: true
    initContainer: {}
  sandboxWorkloads:
    enabled: true
    defaultWorkload: vm-vgpu
  driver:
    enabled: false (1)
  dcgmExporter: {}
  dcgm:
    enabled: true
  daemonsets: {}
  devicePlugin: {}
  gfd: {}
  migManager:
    enabled: true
  nodeStatusExporter:
    enabled: true
  mig:
    strategy: single
  toolkit:
    enabled: true
  validator:
    plugin:
      env:
        - name: WITH_WORKLOAD
          value: "true"
  vgpuManager:
    enabled: true (2)
    repository: <vgpu_container_registry> (3)
    image: <vgpu_image_name>
    version: nvidia-vgpu-manager
  vgpuDeviceManager:
    enabled: false (4)
    config:
      name: vgpu-devices-config
      default: default
  sandboxDevicePlugin:
```

```
    enabled: false 5
  vfioManager:
    enabled: false 6
```

1. この値を **false** に設定します。仮想マシンには必要ありません。
2. この値を **true** に設定します。仮想マシンで vGPU を使用する場合に必要です。
3. **<vgpu_container_registry>** をレジストリー値に置き換えます。
4. この値を **false** に設定すると、OpenShift Virtualization が NVIDIA GPU Operator の代わりに仲介デバイスを設定できるようになります。
5. vGPU デバイスの検出と kubelet へのアドバタイズを防止するには、この値を **false** に設定します。
6. **vfio-pci** ドライバーがロードされないようにするには、この値を **false** に設定します。代わりに、OpenShift Virtualization のドキュメントに従って PCI パススルーを設定します。

関連情報

- PCI パススルーの設定

7.13.12.4. 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:
  mediatedDeviceTypes:
  - 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、**mediatedDeviceTypes** 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。次の **mediatedDeviceTypes** リストが設定されています。

```
# ...
mediatedDevicesConfiguration:
  mediatedDeviceTypes:
  - 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。

7.13.12.5. 仲介されたデバイスの管理

仲介されたデバイスを仮想マシンに割り当てる前に、デバイスを作成してクラスターに公開する必要があります。仲介されたデバイスを再設定および削除することもできます。

7.13.12.5.1. 仲介デバイスの作成および公開

管理者は、**HyperConverged** カスタムリソース (CR) を編集することで、仲介デバイスを作成し、クラスターに公開できます。

前提条件

- 入出力メモリー管理ユニット (IOMMU) ドライバーを有効にしました。
- 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、NVIDIA GRID ドライバー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、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
```

例7.1 仲介デバイスが設定された設定ファイルの例

```
apiVersion: hco.kubevirt.io/v1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  mediatedDevicesConfiguration:
    mediatedDeviceTypes:
    - nvidia-231
    nodeMediatedDeviceTypes:
    - mediatedDeviceTypes:
      - 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
# ...
```

2. 仲介デバイスを **spec.mediatedDevicesConfiguration** スタンザに追加して作成します。

YAML スニペットの例

```
# ...
spec:
  mediatedDevicesConfiguration:
    mediatedDeviceTypes: 1
    - <device_type>
    nodeMediatedDeviceTypes: 2
    - mediatedDeviceTypes: 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 **mediatedDeviceTypes** 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 **mediatedDeviceTypes** 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。

重要

OpenShift Virtualization 4.14 より前では、**mediatedDeviceTypes** フィールドの名前は **mediatedDevicesTypes** でした。仲介デバイスを設定するときは、必ず正しいフィールド名を使用してください。

3. クラスターに公開するデバイスの名前セレクターとリソース名の値を特定します。次のステップで、これらの値を **HyperConverged** CR に追加します。

   a. 次のコマンドを実行して、**resourceName** 値を見つけます。

   ```
   $ oc get $NODE -o json \
     | jq '.status.allocatable \
       | with_entries(select(.key | startswith("nvidia.com/"))) \
       | with_entries(select(.value != "0"))'
   ```

   b. **/sys/bus/pci/devices/<slot>:<bus>:<domain>.<function>/mdev_supported_types/<type>/name** の内容を表示して **mdevNameSelector** 値を見つけ、システムの正しい値に置き換えます。
   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。

4. **mdevNameSelector** と **resourceName** の値を **HyperConverged** CR の **spec.permittedHostDevices.mediatedDevices** スタンザに追加することで、仲介されたデバイスをクラスターに公開します。

   YAML スニペットの例

   ```
   # ...
     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。

   2 ノードに割り当てられているリソース名と一致します。

5. 変更を保存し、エディターを終了します。

検証

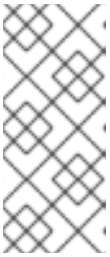
- オプション: 次のコマンドを実行して、デバイスが特定のノードに追加されたことを確認します。

  ```
  $ oc describe node <node_name>
  ```

7.13.12.5.2. 仲介デバイスの変更および削除について

仲介されたデバイスは、いくつかの方法で再設定または削除できます。

- **HyperConverged** CR を編集し、**mediatedDeviceTypes** 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.mediatedDevicesConfiguration** 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。

7.13.12.5.3. 仲介されたデバイスをクラスターから削除する

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:
    mediatedDeviceTypes: 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 **mediatedDeviceTypes** 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。

7.13.12.6. 仲介デバイスの使用

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。

7.13.12.6.1. CLI を使用した仮想マシンへの vGPU 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。
- 仮想マシンが停止しています。

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_T4-2Q
        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。

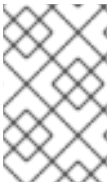
検証

- 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>
```

7.13.12.6.2. Web コンソールを使用した仮想マシンへの vGPU の割り当て

OpenShift Container Platform Web コンソールを使用して、仮想 GPU を仮想マシンに割り当てることができます。



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。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。

前提条件

- vGPU は、クラスター内の仲介デバイスとして設定されます。
  - クラスターに接続されているデバイスを表示するには、サイドメニューから **Compute** → **Hardware Devices** をクリックします。
- 仮想マシンが停止しています。

手順

1. OpenShift Container Platform Web コンソールで、サイドメニューから**Virtualization** → **VirtualMachines**をクリックします。
2. 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。
4. **Add GPU device** をクリックします。
5. **Name** フィールドに識別値を入力します。
6. **Device name** リストから、仮想マシンに追加するデバイスを選択します。
7. **Save** をクリックします。

検証

- デバイスが仮想マシンに追加されたことを確認するには、**YAML** タブをクリックし、**VirtualMachine** 設定を確認します。仲介されたデバイスは **spec.domain.devices** スタンザに追加されます。

7.13.12.7. 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

7.13.13. 仮想マシンでの Descheduler エビクションの有効化

Descheduler を使用して Pod を削除し、Pod をより適切なノードに再スケジュールできます。Pod が仮想マシンの場合、Pod の削除により、仮想マシンが別のノードにライブマイグレーションされます。

重要

仮想マシンの Descheduler エビクションはテクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

7.13.13.1. Descheduler プロファイル

テクノロジープレビューの **DevPreviewLongLifecycle** プロファイルを使用して、仮想マシンで Descheduler を有効にします。これは、現在 OpenShift Virtualization で利用可能な唯一の Descheduler プロファイルです。適切なスケジューリングを確保するには、予想される負荷に応じた CPU およびメモリー要求で仮想マシンを作成します。

DevPreviewLongLifecycle

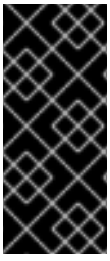
このプロファイルは、ノード間のリソース使用率のバランスを取り、以下のストラテジーを有効にします。

- **RemovePodsHavingTooManyRestarts**: コンテナーが何度も再起動された Pod、およびすべてのコンテナー (Init コンテナーを含む) の再起動の合計が 100 を超える Pod を削除します。仮想マシンのゲストオペレーティングシステムを再起動しても、この数は増えません。
- **LowNodeUtilization**: 使用率の低いノードがある場合に、使用率の高いノードから Pod をエビクトします。エビクトされた Pod の宛先ノードはスケジューラーによって決定されます。
  - ノードは、使用率がすべてのしきい値 (CPU、メモリー、Pod の数) について 20% 未満の場合に使用率が低いと見なされます。
  - ノードは、使用率がすべてのしきい値 (CPU、メモリー、Pod の数) について 50% を超える場合に過剰に使用されていると見なされます。

7.13.13.2. Descheduler のインストール

Descheduler はデフォルトで利用できません。Descheduler を有効にするには、Kube Descheduler Operator を OperatorHub からインストールし、1 つ以上の Descheduler プロファイルを有効にする必要があります。

デフォルトで、Descheduler は予測モードで実行されます。つまり、これは Pod エビクションのみをシミュレートします。Pod エビクションを実行するには、Descheduler のモードを automatic に変更する必要があります。



重要

クラスターでホストされたコントロールプレーンを有効にしている場合は、カスタム優先度のしきい値を設定して、ホストされたコントロールプレーンの namespace の Pod が削除される可能性を下げます。ホストされたコントロールプレーンの優先度クラスの中で優先度値が最も低い (**100000000**) ため、優先度しきい値クラス名を **hypershift-control-plane** に設定します。

前提条件

- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform にログインしている。
- OpenShift Container Platform Web コンソールにアクセスできる。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. Kube Descheduler Operator に必要な namespace を作成します。
   a. **Administration → Namespaces** に移動し、**Create Namespace** をクリックします。
   b. **Name** フィールドに **openshift-kube-descheduler-operator** を入力し、**Labels** フィールドに **openshift.io/cluster-monitoring=true** を入力して Descheduler メトリックを有効にし、**Create** をクリックします。
3. Kube Descheduler Operator をインストールします。
   a. **Operators → OperatorHub** に移動します。
   b. **Kube Descheduler Operator** をフィルターボックスに入力します。
   c. **Kube Descheduler Operator** を選択し、**Install** をクリックします。
   d. **Install Operator** ページで、**A specific namespace on the cluster**を選択します。ドロップダウンメニューから **openshift-kube-descheduler-operator** を選択します。
   e. **Update Channel** および **Approval Strategy** の値を必要な値に調整します。
   f. **Install** をクリックします。
4. Descheduler インスタンスを作成します。
   a. **Operators → Installed Operators** ページから、**Kube Descheduler Operator** をクリックします。
   b. **Kube Descheduler** タブを選択し、**Create KubeDescheduler**をクリックします。
   c. 必要に応じて設定を編集します。
      i. エビクションをシミュレーションせずに Pod をエビクトするには、**Mode** フィールドを **Automatic** に変更します。
      ii. **Profiles** セクションを展開し、**DevPreviewLongLifecycle** を選択します。**AffinityAndTaints** プロファイルがデフォルトで有効になっています。

重要

OpenShift Virtualization で現在利用できるプロファイルは **DevPreviewLongLifecycle** のみです。

また、後で OpenShift CLI (**oc**) を使用して、Descheduler のプロファイルおよび設定を設定することもできます。

7.13.13.3. 仮想マシン (VM) での Descheduler エビクションの有効化

Descheduler のインストール後に、アノテーションを **VirtualMachine** カスタムリソース (CR) に追加して Descheduler エビクションを仮想マシンで有効にできます。

前提条件

- Descheduler を OpenShift Container Platform Web コンソールまたは OpenShift CLI (**oc**) にインストールしている。
- 仮想マシンが実行されていないことを確認します。

手順

1. 仮想マシンを起動する前に、**descheduler.alpha.kubernetes.io/evict** アノテーションを **VirtualMachine** CR に追加します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
spec:
  template:
    metadata:
      annotations:
        descheduler.alpha.kubernetes.io/evict: "true"
```

2. インストール時に Web コンソールで **DevPreviewLongLifecycle** プロファイルをまだ設定していない場合は、**KubeDescheduler** オブジェクトの **spec.profile** セクションに **DevPreviewLongLifecycle** を指定します。

```
apiVersion: operator.openshift.io/v1
kind: KubeDescheduler
metadata:
  name: cluster
  namespace: openshift-kube-descheduler-operator
spec:
  deschedulingIntervalSeconds: 3600
  profiles:
  - DevPreviewLongLifecycle
  mode: Predictive (1)
```

(1) デフォルトでは、Descheduler は Pod をエビクトしません。Pod をエビクトするには、**mode** を **Automatic** に設定します。

Descheduler が仮想マシンで有効になりました。

7.13.13.4. 関連情報

- デスケジューラーの概要

7.13.14. 仮想マシンの高可用性について

障害が発生したノードを手動で削除して仮想マシンフェイルオーバーをトリガーするか、修復ノードを設定することによって、仮想マシンの高可用性を有効にすることができます。

障害が発生したノードを手動で削除する

ノードに障害が発生し、マシンヘルスチェックがクラスターにデプロイされていない場合、**runStrategy: Always** が設定された仮想マシンは正常なノードに自動的に移動しません。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。

障害が発生したノードを削除して仮想マシンのフェイルオーバーをトリガーする を参照してください。

修復ノードの設定

OperatorHub から Self Node Remediation Operator をインストールし、マシンの健全性チェックまたはノード修復チェックを有効にすることで、修復ノードを設定できます。

ノードの修復、フェンシング、メンテナンスの詳細は、Red Hat OpenShift のワークロードの可用性 を参照してください。

7.13.15. 仮想マシンのコントロールプレーンのチューニング

OpenShift Virtualization は、コントロールプレーンレベルで次のチューニングオプションを提供します。

- **highBurst** プロファイルは、固定 **QPS** と **burst** レートを使用して、1 つのバッチで数百の仮想マシンを作成します。
- ワークロードの種類に基づいた移行設定の調整

7.13.15.1. highBurst プロファイルの設定

highBurst プロファイルを使用して、1 つのクラスター内に多数の仮想マシンを作成および維持します。

手順

- 次のパッチを適用して、**highBurst** チューニングポリシープロファイルを有効にします。

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

検証

- 次のコマンドを実行して、**highBurst** チューニングポリシープロファイルが有効になっていることを確認します。

```
$ oc get kubevirt.kubevirt.io/kubevirt-kubevirt-hyperconverged \
  -n openshift-cnv -o go-template --template='{{range $config, \
  $value := .spec.configuration}} {{if eq $config "apiConfiguration" \
  "webhookConfiguration" "controllerConfiguration" "handlerConfiguration"}} \
  {{"\n"}} {{$config}} = {{$value}} {{end}} {{end}} {{"\n"}}
```

7.13.16. コンピュートリソースの割り当て

OpenShift Virtualization では、仮想マシン (VM) に割り当てられたコンピュートリソースは、Guaranteed CPU またはタイムスライスされた CPU シェアのいずれかによってサポートされます。

Guaranteed CPU (CPU 予約とも呼ばれる) は、CPU コアまたはスレッドを特定のワークロード専用にし、他のワークロードでは使用できなくなります。Guaranteed CPU を仮想マシンに割り当てると、仮想マシンは予約された物理 CPU に単独でアクセスできるようになります。仮想マシンの専用リソースを有効化 し、Guaranteed CPU を使用します。

タイムスライス CPU は、共有物理 CPU 上の時間のスライスを各ワークロード専用にします。スライスのサイズは、仮想マシンの作成中、または仮想マシンがオフラインのときに指定できます。デフォルトでは、各 vCPU は 100 ミリ秒、つまり 1/10 秒の物理 CPU 時間を受け取ります。

CPU 予約のタイプは、インスタンスのタイプまたは仮想マシン設定によって異なります。

7.13.16.1. CPU リソースのオーバーコミット

タイムスライシングにより、複数の仮想 CPU (vCPU) が 1 つの物理 CPU を共有できます。これは **CPU オーバーコミットメント** として知られています。Guaranteed 仮想マシンをオーバーコミットすることはできません。

CPU を仮想マシンに割り当てるときに、パフォーマンスよりも仮想マシン密度を優先するように CPU オーバーコミットメントを設定します。vCPU の CPU オーバーコミットメントが高くなると、より多くの仮想マシンが特定のノードに適合します。

7.13.16.2. CPU 割り当て率の設定

CPU 割り当て率は、vCPU を物理 CPU のタイムスライスにマッピングすることにより、オーバーコミットメントの程度を指定します。

たとえば、10:1 のマッピングまたは比率は、タイムスライスを使用して、10 個の仮想 CPU を 1 個の物理 CPU にマッピングします。

各物理 CPU にマップされる vCPU のデフォルトの数を変更するには、**HyperConverged** CR で **vmiCPUAllocationRatio** 値を設定します。Pod CPU リクエストは、vCPU の数に CPU 割り当て率の逆数を乗算して計算されます。たとえば、**vmiCPUAllocationRatio** が 10 に設定されている場合、OpenShift Virtualization はその仮想マシンの Pod 上で 10 分の 1 少ない CPU を要求します。

手順

HyperConverged CR で **vmiCPUAllocationRatio** 値を設定して、ノードの CPU 割り当て率を定義します。

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

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

2. **vmiCPUAllocationRatio** を設定します。

```
...
spec:
  resourceRequirements:
    vmiCPUAllocationRatio: 1 (1)
# ...
```

(1) **vmiCPUAllocationRatio** が **1** に設定されている場合、Pod に対して最大量の vCPU が要求されます。

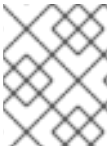
7.13.16.3. 関連情報

- Pod Quality of Service Classes

7.13.17. マルチキュー機能について

マルチキュー機能を使用して、複数の vCPU を備えた仮想マシン (VM) のネットワークスループットとパフォーマンスを拡張します。

デフォルトでは、ドメイン XML から派生した **queueCount** 値は、仮想マシンに割り当てられた vCPU の数によって決まります。vCPU の数が増えても、ネットワークパフォーマンスは向上しません。さらに、virtio-net には Tx キューと Rx キューが 1 つしかないため、ゲストはパケットを並行して送信または取得できません。



注記

ゲストインスタンス内の vNIC の数が vCPU の数に比例する場合、virtio-net マルチキューを有効にしても大きな改善は得られません。

7.13.17.1. 既知の制限

- virtio-net マルチキューがホストで有効になっているが、管理者によってゲストオペレーティングシステムで有効になっていない場合でも、MSI ベクトルは消費されます。
- 各 virtio-net キューは、vhost ドライバー用に 64 KiB のカーネルメモリーを消費します。
- **networkInterfaceMultiqueue** が 'true' に設定されている場合、CPU が 16 個を超えて搭載されている仮想マシンを起動すると接続できなくなります (CNV-16107)。

7.13.17.2. マルチキュー機能の有効化

VirtIO モデルで設定されたインターフェイスのマルチキュー機能を有効にします。

手順

1. マルチキュー機能を有効にするには、仮想マシンの **VirtualMachine** マニフェストファイルで **networkInterfaceMultiqueue** 値を **true** に設定します。

```
apiVersion: kubevirt.io/v1
kind: VM
spec:
  domain:
    devices:
      networkInterfaceMultiqueue: true
```

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。

7.14. 仮想マシンディスク

7.14.1. 仮想マシンディスクのホットプラグ

仮想マシン (VM) または仮想マシンインスタンス (VMI) を停止することなく、仮想ディスクを追加または削除できます。

データボリュームおよび永続ボリューム要求 (PVC) のみをホットプラグおよびホットアンプラグできます。コンテナーディスクのホットプラグおよびホットアンプラグはできません。

再起動後にもホットプラグされたディスクは仮想マシンに接続されたままです。仮想マシンからディスクを削除するには、ディスクを切断する必要があります。

ホットプラグされたディスクを永続的に作成して、仮想マシンに永続的にマウントできます。

注記

各仮想マシンには **virtio-scsi** コントローラーがあり、ホットプラグされたディスクが **SCSI** バスを使用できるようになります。**virtio-scsi** コントローラーは、パフォーマンス上の利点を維持しながら、**virtio** の制限を克服します。スケーラビリティーが高く、400 万台を超えるディスクのホットプラグをサポートします。

通常の **virtio** はスケーラブルではないため、ホットプラグされたディスクには使用できません。各 **virtio** ディスクは、仮想マシン内の限られた PCI Express (PCIe) スロットの 1 つを使用します。PCIe スロットは他のデバイスでも使用されるため、事前に予約する必要があります。したがって、スロットはオンデマンドで利用できない場合があります。

7.14.1.1. Web コンソールを使用したディスクのホットプラグとホットアンプラグ

OpenShift Container Platform Web コンソールを使用して、仮想マシンの実行中にディスクを仮想マシンに接続することで、ディスクをホットプラグできます。

ホットプラグされたディスクは、プラグを抜くまで仮想マシンに接続されたままになります。

ホットプラグされたディスクを永続的に作成して、仮想マシンに永続的にマウントできます。

前提条件

- ホットプラグに使用できるデータボリュームまたは永続ボリュームクレーム (PVC) が必要です。

手順

1. Web コンソールで **Virtualization → VirtualMachines** に移動します。
2. 実行中の仮想マシンを選択して、その詳細を表示します。
3. **VirtualMachine details** ページで、**Configuration → Disks** をクリックします。
4. ホットプラグされたディスクを追加します。
   a. **ディスクの追加**をクリックします。
   b. **Add disk (hot plugged)** ウィンドウで、**Source** リストからディスクを選択し、**Save** をクリックします。
5. オプション: ホットプラグされたディスクを取り外します。
   a. ディスクの横にあるオプションメニュー ⋮ をクリックし、**Detach** を選択します。
   b. **Detach** をクリックします。

6. オプション: ホットプラグされたディスクを永続的にします。

   a. ディスクの横にあるオプションメニュー  をクリックし、**Make persistent** を選択します。

   b. 仮想マシンを再起動して変更を適用します。

7.14.1.2. コマンドラインを使用したディスクのホットプラグとホットアンプラグ

コマンドラインを使用して、仮想マシンの実行中にディスクのホットプラグおよびホットアンプラグを行うことができます。

ホットプラグされたディスクを永続的に作成して、仮想マシンに永続的にマウントできます。

前提条件

- 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

- 次のコマンドを実行して、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。

- 次のコマンドを実行して、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>
```

7.14.2. 仮想マシンディスクの拡張

ディスクの永続ボリューム要求 (PVC) を拡張することで、仮想マシンディスクのサイズを増やすことができます。

ストレージプロバイダーがボリューム拡張をサポートしていない場合は、空のデータボリュームを追加することで、仮想マシンの利用可能な仮想ストレージを拡張できます。

仮想マシンディスクのサイズを減らすことはできません。

7.14.2.1. 仮想マシンディスク PVC の拡張

ディスクの永続ボリューム要求 (PVC) を拡張することで、仮想マシンディスクのサイズを増やすことができます。

PVC がファイルシステムボリュームモードを使用する場合、ディスクイメージファイルは、ファイルシステムのオーバーヘッド用にスペースを確保しながら、利用可能なサイズまで拡張されます。

手順

1. 拡張する必要のある仮想マシンディスクの **PersistentVolumeClaim** マニフェストを編集します。

```
$ oc edit pvc <pvc_name>
```

2. ディスクサイズを更新します。

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

(1) 新しいディスクサイズを指定します。

ボリューム拡張のための関連情報

- 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

7.14.2.2. 空のデータボリュームを追加して利用可能な仮想ストレージを拡張する

空のデータボリュームを追加することで、仮想マシンの利用可能なストレージを拡張できます。

前提条件

- 少なくとも 1 つの永続ボリュームが必要です。

手順

1. 次の例に示すように、**DataVolume** マニフェストを作成します。

DataVolume マニフェストの例

-

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: DataVolume
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  storage:
    resources:
      requests:
        storage: <2Gi> 1
  storageClassName: "<storage_class>" 2
```

1 データボリュームに要求される利用可能なスペースの量を指定します。

2 オプション: ストレージクラスを指定しない場合は、デフォルトのストレージクラスが適用されます。

2. 以下のコマンドを実行してデータボリュームを作成します。

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

データボリューム用の関連情報

- データボリュームの事前割り当てモードの設定
- データボリュームアノテーションの管理

7.14.3. 仮想マシンの共有ボリュームの設定

共有ディスクを設定して、複数の仮想マシン (VM) が同じ基礎となるストレージを共有できるようにすることができます。共有ディスクのボリュームは、ブロックモードである必要があります。

ディスク共有を設定するには、ストレージを次のいずれかのタイプとして公開します。

- 通常の仮想マシンディスク
- 共有ボリュームの Windows フェイルオーバークラスタリングに必要な、SCSI 接続と RAW デバイスマッピングを備えた論理ユニット番号 (LUN) ディスク

ディスク共有を設定するだけでなく、通常の仮想マシンディスクまたは LUN ディスクごとにエラーポリシーを設定することもできます。エラーポリシーは、ディスクの読み取りまたは書き込み時に入出力エラーが発生した場合のハイパーバイザーの動作を制御します。

7.14.3.1. 仮想マシンディスクを使用してディスク共有を設定する

複数の仮想マシン (VM) がストレージを共有できるようにブロックボリュームを設定できます。

ゲストオペレーティングシステムで実行されているアプリケーションに応じて、仮想マシンに設定する必要があるストレージオプションが決まります。**disk** タイプのディスクは、ボリュームを通常のディスクとして仮想マシンに公開します。

ディスクごとにエラーポリシーを設定できます。エラーポリシーは、ディスクの書き込み中または読み取り中に入出力エラーが発生した場合のハイパーバイザーの動作を制御します。デフォルトの動作では、仮想マシンが停止され、Kubernetes イベントが生成されます。

デフォルトの動作を受け入れることも、エラーポリシーを次のいずれかのオプションに設定することもできます。

- **report**、ゲスト内のエラーを報告します。
- **ignore**、エラーを無視します。読み取りまたは書き込みの障害は検出されません。
- **enospace**、ディスク容量が不足していることを示すエラーを生成します。

前提条件

- ディスクを共有している仮想マシンが異なるノードで稼働している場合、ボリュームアクセスモードは **ReadWriteMany** (RWX) である必要があります。
  ディスクを共有している仮想マシンが同じノード上で稼働している場合、ボリュームアクセスモードは **ReadWriteOnce** (RWO) で問題ありません。
- ストレージプロバイダーは、必要な Container Storage Interface (CSI) ドライバーをサポートする必要があります。

手順

1. 次の例のように、仮想マシンの **VirtualMachine** マニフェストを作成して必要な値を設定します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <vm_name>
spec:
  template:
# ...
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: rootdisk
            errorPolicy: report (1)
            disk1: disk_one (2)
          - disk:
              bus: virtio
            name: cloudinitdisk
            disk2: disk_two
            shareable: true (3)
          interfaces:
          - masquerade: {}
            name: default
```

(1) エラーポリシーを識別します。

2 デバイスをディスクとして識別します。

3 共有ディスクを識別します。

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。

7.14.3.2. LUN を使用してディスク共有を設定する

仮想マシン上のデータを外部からのアクセスから保護するには、SCSI 永続予約を有効にし、LUN でバックアップされた仮想マシンディスクを複数の仮想マシン間で共有するように設定します。共有オプションを有効にすると、Windows フェイルオーバークラスタリング実装に必要なコマンドなどの高度な SCSI コマンドを使用して、基盤となるストレージを管理できます。

ストレージボリュームが **LUN** ディスクタイプとして設定されていると、仮想マシンはボリュームを論理ユニット番号 (LUN) デバイスとして使用できます。その結果、仮想マシンは SCSI コマンドを使用してディスクをデプロイおよび管理できるようになります。

SCSI 永続予約オプションを使用して LUN を予約します。予約を有効にするには、以下を行います。

1. フィーチャーゲートオプションを設定する。
2. LUN ディスク上のフィーチャーゲートオプションをアクティブにして、仮想マシンに必要な SCSI デバイス固有の入出力制御 (IOCTL) を発行する。

各 LUN ディスクに対してエラーポリシーを設定できます。エラーポリシーは、ディスクの読み取りまたは書き込み時に入出力エラーが発生した場合のハイパーバイザーの動作を制御します。デフォルトの動作では、ゲストが停止され、Kubernetes イベントが生成されます。

共有ボリュームの Windows フェイルオーバークラスタリングに必要な、iSCSi 接続と永続的な予約を持つ LUN ディスクの場合は、エラーポリシーを **report** に設定します。

前提条件

- フィーチャーゲートオプションの設定に必要なクラスター管理者権限がある。
- ディスクを共有している仮想マシンが異なるノードで稼働している場合、ボリュームアクセスモードは **ReadWriteMany** (RWX) である必要があります。
  ディスクを共有している仮想マシンが同じノード上で稼働している場合、ボリュームアクセスモードは **ReadWriteOnce** (RWO) で問題ありません。
- ストレージプロバイダーは、SCSI プロトコルを使用する Container Storage Interface (CSI) ドライバーをサポートする必要があります。
- クラスター管理者が LUN を使用してディスク共有を設定する場合は、**HyperConverged** カスタムリソース (CR) でクラスターのフィーチャーゲートを有効にする必要があります。
- 共有するディスクはブロックモードである必要があります。

手順

1. 次の例のように、仮想マシンの **VirtualMachine** マニフェストを編集または作成して、必要な値を設定します。

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

```
metadata:
  name: vm-0
spec:
  template:
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: sata
            name: rootdisk
          - errorPolicy: report (1)
            lun: (2)
              bus: scsi
              reservation: true (3)
            name: na-shared
            serial: shared1234
      volumes:
      - dataVolume:
          name: vm-0
        name: rootdisk
      - name: na-shared
        persistentVolumeClaim:
          claimName: pvc-na-share
```

1. エラーポリシーを識別します。
2. LUN ディスクを識別します。
3. 永続予約が有効であることを識別します。

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。

7.14.3.2.1. LUN と Web コンソールを使用してディスク共有を設定する

OpenShift Container Platform Web コンソールで、LUN を使用してディスク共有を設定できます。

前提条件

- クラスター管理者は、**persistentreservation** フィーチャーゲート設定を有効にする必要があります。

手順

1. Web コンソールで、**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. **Storage** を展開します。
4. **Disks** タブで、**Add disk** をクリックします。
5. **Name**、**Source**、**Size**、**Interface**、**Storage Class** を指定します。
6. **Type** として **LUN** を選択します。

7. **Access Mode** として **Shared access (RWX)** を選択します。
8. **Volume Mode** として **Block** を選択します。
9. **Advanced Settings** を展開し、両方のチェックボックスをオンにします。
10. **Save** をクリックします。

7.14.3.2.2. LUN とコマンドラインを使用してディスク共有を設定する

コマンドラインで LUN を使用してディスク共有を設定できます。

手順

1. 次の例のように、仮想マシンの **VirtualMachine** マニフェストを編集または作成して、必要な値を設定します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-0
spec:
  template:
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: sata
            name: rootdisk
          - errorPolicy: report
            lun: (1)
              bus: scsi
              reservation: true (2)
            name: na-shared
            serial: shared1234
      volumes:
      - dataVolume:
          name: vm-0
        name: rootdisk
      - name: na-shared
        persistentVolumeClaim:
          claimName: pvc-na-share
```

(1) LUN ディスクを識別します。

(2) 永続予約が有効であることを識別します。

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。

7.14.3.3. PersistentReservation フィーチャーゲートの有効化

SCSI **persistentReservation** フィーチャーゲートを有効にして、LUN バックアップのブロックモードの仮想マシン (VM) ディスクを、複数の仮想マシン間で共有できます。

デフォルトで、**persistentReservation** フィーチャーゲートは無効になっています。Web コンソールまたはコマンドラインを使用して、**persistentReservation** フィーチャーゲートを有効にできます。

前提条件

- クラスター管理者権限が必要です。
- ディスクを共有している仮想マシンが異なるノードで稼働している場合、ボリュームアクセスモードは **ReadWriteMany** (RWX) である必要があります。ディスクを共有している仮想マシンが同じノード上で稼働している場合、ボリュームアクセスモードは **ReadWriteOnce** (RWO) で問題ありません。
- ストレージプロバイダーは、SCSI プロトコルを使用する Container Storage Interface (CSI) ドライバーをサポートする必要があります。

7.14.3.3.1. Web コンソールを使用して PersistentReservation フィーチャーゲートを有効にする

LUN バックアップのブロックモード仮想マシンディスクを複数の仮想マシン間で共有できるようにするには、PersistentReservation フィーチャーゲートを有効にする必要があります。フィーチャーゲートを有効にするには、クラスター管理者権限が必要です。

手順

1. Web コンソールで **Virtualization → Overview** をクリックします。
2. **Settings** タブをクリックします。
3. **Cluster** を選択します。
4. **SCSI persistent reservation** を展開し、**Enable persistent reservation** をオンに設定します。

7.14.3.3.2. コマンドラインを使用して PersistentReservation フィーチャーゲートを有効にする

コマンドラインを使用して、**persistentReservation** フィーチャーゲートを有効にできます。フィーチャーゲートを有効にするには、クラスター管理者権限が必要です。

手順

1. 次のコマンドを実行して、**persistentReservation** フィーチャーゲートを有効にします。

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

関連情報

- Persistent reservation helper protocol
- Failover Clustering in Windows Server and Azure Stack HCI

第8章 ネットワーク

8.1. ネットワークの概要

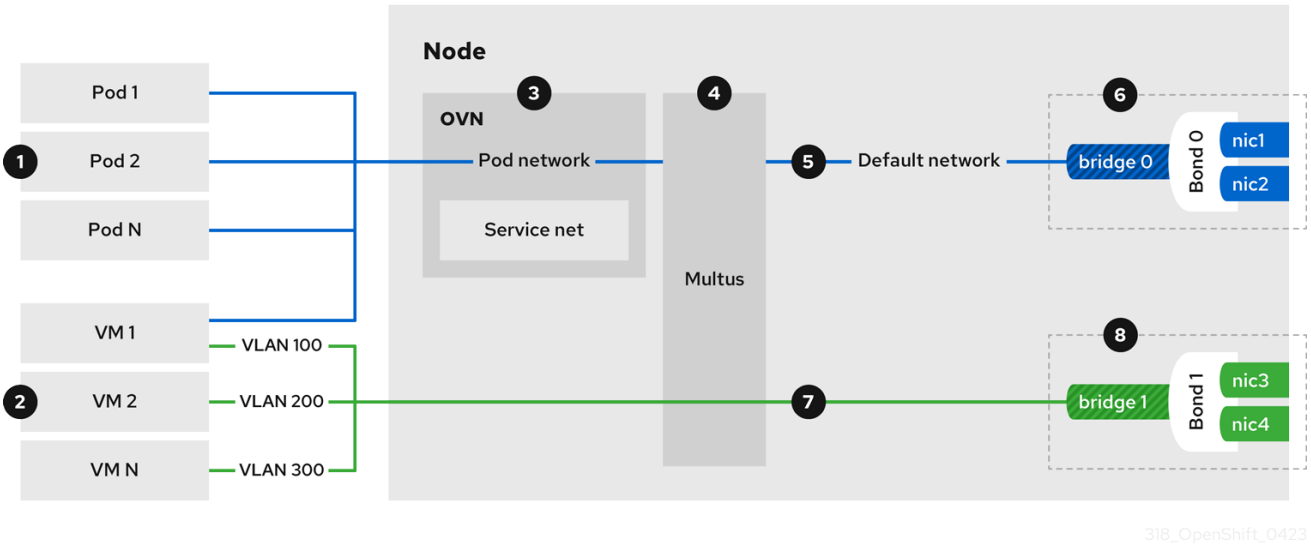
OpenShift Virtualization は、カスタムリソースおよびプラグインを使用して高度なネットワーク機能を提供します。仮想マシン (VM) は、OpenShift Container Platform のネットワークおよびエコシステムと統合されています。

注記

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。

次の図は、OpenShift Virtualization の一般的なネットワーク設定を示しています。他の設定も可能です。

図8.1 OpenShift Virtualization ネットワークの概要



1 Pod と仮想マシンは同じネットワークインフラストラクチャー上で実行するため、コンテナー化されたワークロードと仮想化されたワークロードを簡単に接続できます。

2 仮想マシンをデフォルトの Pod ネットワークおよび任意の数のセカンダリーネットワークに接続できます。

3 デフォルトの Pod ネットワークは、すべてのメンバー間の接続、サービス抽象化、IP 管理、マイクロセグメンテーション、およびその他の機能を提供します。

4 Multus は、互換性のある他の CNI プラグインを使用して、Pod または仮想マシンが追加のネットワークインターフェイスに接続できるようにする "メタ" CNI プラグインです。

5 デフォルトの Pod ネットワークはオーバーレイベースであり、基盤となるマシンネットワークを介してトンネリングされます。

6 マシンネットワークは、選択したネットワークインターフェイスコントローラー (NIC) のセットを介して定義できます。

7 セカンダリー仮想マシンネットワークは通常、VLAN カプセル化の有無にかかわらず、物理ネットワークに直接ブリッジされます。

8 セカンダリー仮想マシンネットワークは、図 1 に示すように専用の NIC セットで定義することも、マシンネットワークを使用することもできます。

8.1.1. OpenShift Virtualization ネットワークの用語集

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)

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)

Multus プロジェクトによって導入された CRD。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接続できるようにします。

ノードネットワーク設定ポリシー (NNCP)

nmstate プロジェクトによって導入された CRD。ノード上で要求されるネットワーク設定を表します。**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。

8.1.2. デフォルトの Pod ネットワークの使用

仮想マシンをデフォルトの Pod ネットワークに接続する

各仮想マシンは、デフォルトの内部 Pod ネットワークにデフォルトで接続されます。仮想マシン仕様を編集することで、ネットワークインターフェイスを追加または削除できます。

仮想マシンをサービスとして公開する

Service オブジェクトを作成することで、クラスター内またはクラスター外に仮想マシンを公開できます。オンプレミスクラスターの場合、MetalLB Operator を使用して負荷分散サービスを設定できます。OpenShift Container Platform Web コンソールまたは CLI を使用して、MetalLB Operator をインストール できます。

8.1.3. 仮想マシンのセカンダリーネットワークインターフェイスの設定

Linux ブリッジ、SR-IOV、OVN-Kubernetes CNI プラグインを使用して、仮想マシンをセカンダリーネットワークに接続できます。仮想マシン仕様では、複数のセカンダリーネットワークとインターフェイスをリストできます。セカンダリーネットワークインターフェイスに接続する場合、仮想マシン仕様でプライマリー Pod ネットワークを指定する必要はありません。

Linux ブリッジネットワークへの仮想マシンの割り当て

Kubernetes NMState Operator をインストール して、セカンダリーネットワークの Linux ブリッジ、VLAN、およびボンディングを設定します。
次の手順を実行すると、Linux ブリッジネットワークを作成し、そのネットワークに仮想マシンを接続できます。

1. **NodeNetworkConfigurationPolicy** カスタムリソース定義 (CRD) を作成して、 Linux ブリッジネットワークデバイスを設定 します。
2. **NetworkAttachmentDefinition** CRD を作成して、 Linux ブリッジネットワークを設定 します。
3. 仮想マシン設定にネットワークの詳細を追加して、仮想マシンを Linux ブリッジネットワークに接続 します。

仮想マシンの SR-IOV ネットワークへの接続

高帯域幅または低レイテンシーを必要とするアプリケーション向けに、ベアメタルまたは Red Hat OpenStack Platform (RHOSP) インフラストラクチャーにインストールされた OpenShift Container Platform クラスター上の追加ネットワークで、Single Root I/O Virtualization (SR-IOV) ネットワークデバイスを使用できます。
SR-IOV ネットワークデバイスとネットワーク接続を管理するには、クラスターに SR-IOV Network Operator をインストール する必要があります。

次の手順を実行すると、仮想マシンを SR-IOV ネットワークに接続できます。

1. **SriovNetworkNodePolicy** CRD を作成して、 SR-IOV ネットワークデバイスを設定 します。
2. **SriovNetwork** オブジェクトを作成して、 SR-IOV ネットワークを設定 します。
3. 仮想マシン設定にネットワークの詳細を追加して、仮想マシンを SR-IOV ブリッジネットワークに接続 します。

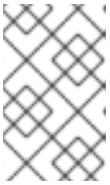
OVN-Kubernetes セカンダリーネットワークへの仮想マシンの接続

仮想マシンは Open Virtual Network (OVN)-Kubernetes セカンダリーネットワークに接続できます。OpenShift Virtualization は、OVN-Kubernetes のレイヤー 2 およびローカルネットトポロジーをサポートします。

- レイヤー 2 トポロジーは、クラスター全体の論理スイッチでワークロードを接続します。OVN-Kubernetes Container Network Interface (CNI) プラグインは、Geneve (Generic Network Virtualization Encapsulation) プロトコルを使用してノード間にオーバーレイネットワークを作成します。このオーバーレイネットワークを使用すると、追加の物理ネットワークインフラストラクチャーを設定することなく、さまざまなノード上の仮想マシンを接続できます。
- ローカルネットトポロジーは、セカンダリーネットワークを物理アンダーレイに接続します。これにより、east-west クラスタートラフィックとクラスター外で実行されているサービスへのアクセスの両方が可能になります。ただし、クラスターノード上の基盤となる Open vSwitch (OVS) システムの追加設定が必要です。

OVN-Kubernetes セカンダリーネットワークを設定し、そのネットワークに仮想マシンを接続するには、次の手順を実行します。

1. ネットワークアタッチメント定義 (NAD) を作成して、OVN-Kubernetes セカンダリーネットワークの設定 を行います。



注記

ローカルネットトポロジーの場合は、NAD を作成する前に **NodeNetworkConfigurationPolicy** オブジェクトを作成して OVS ブリッジを設定する 必要があります。

2. ネットワークの詳細を仮想マシン仕様に追加して、仮想マシンを OVN-Kubernetes セカンダリーネットワークに接続 します。

8.1.3.1. Linux ブリッジ CNI と OVN-Kubernetes ローカルネットトポロジーの比較

次の表は、Linux ブリッジ CNI を使用する場合と OVN-Kubernetes プラグインのローカルネットトポロジーを使用する場合に利用できる機能の比較を示しています。

表8.1 Linux ブリッジ CNI と OVN-Kubernetes ローカルネットトポロジーの比較

機能	Linux ブリッジ CNI で利用可能	OVN-Kubernetes ローカルネットで利用可能
アンダーレイネイティブネットワークへのレイヤー 2 アクセス	セカンダリーネットワークインターフェイスコントローラー (NIC) のみ	はい
アンダーレイ VLAN へのレイヤー 2 アクセス	はい	はい
ネットワークポリシー	いいえ	はい
管理された IP プール	いいえ	いいえ
MAC スプーフィングフィルタリング	はい	はい

ホットプラグ対応のセカンダリーネットワークインターフェイス

仮想マシンを停止せずに、セカンダリーネットワークインターフェイスを追加または削除できます。OpenShift Virtualization は、VirtIO デバイスドライバーを使用する Linux ブリッジインターフェイスのホットプラグとホットアンプラグをサポートしています。

SR-IOV での DPDK の使用

Data Plane Development Kit (DPDK) は、高速パケット処理用のライブラリーとドライバーのセットを提供するものです。SR-IOV ネットワーク上で DPDK ワークロードを実行するようにクラスターと仮想マシンを設定できます。

ライブマイグレーション用の専用ネットワークの設定

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。

クラスター FQDN を使用した仮想マシンへのアクセス

完全修飾ドメイン名 (FQDN) を使用して、クラスターの外部からセカンダリーネットワークインターフェイスに接続されている仮想マシンにアクセスできます。

IP アドレスの設定と表示

仮想マシンを作成するときに、セカンダリーネットワークインターフェイスの IP アドレスを設定できます。IP アドレスは、cloud-init でプロビジョニングされます。仮想マシンの IP アドレスは、OpenShift Container Platform Web コンソールまたはコマンドラインを使用して表示できます。ネットワーク情報は QEMU ゲストエージェントによって収集されます。

8.1.4. OpenShift Service Mesh との統合

仮想マシンのサービスメッシュへの接続

OpenShift Virtualization は OpenShift Service Mesh と統合されています。Pod と仮想マシン間のトラフィックを監視、視覚化、制御できます。

8.1.5. MAC アドレスプールの管理

ネットワークインターフェイスの MAC アドレスプールの管理

KubeMacPool コンポーネントは、共有 MAC アドレスプールから仮想マシンネットワークインターフェイスの MAC アドレスを割り当てます。これにより、各ネットワークインターフェイスに一意の MAC アドレスが確実に割り当てられます。その仮想マシンから作成された仮想マシンインスタンスは、再起動後も割り当てられた MAC アドレスを保持します。

8.1.6. SSH アクセスの設定

仮想マシンへの SSH アクセスの設定

次の方法を使用して、仮想マシンへの SSH アクセスを設定できます。

- **virtctl ssh コマンド**
  SSH キーペアを作成し、公開キーを仮想マシンに追加し、秘密キーを使用して **virtctl ssh** コマンドを実行して仮想マシンに接続します。

  cloud-init データソースを使用して設定できるゲストオペレーティングシステムを使用して、実行時または最初の起動時に Red Hat Enterprise Linux (RHEL) 9 仮想マシンに公開 SSH キーを追加できます。

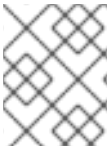
- **virtctl port-forward コマンド**
  virtctl port-foward コマンドを **.ssh/config** ファイルに追加し、OpenSSH を使用して仮想マシンに接続します。

- **サービス**
  サービスを作成し、そのサービスを仮想マシンに関連付け、サービスによって公開されている IP アドレスとポートに接続します。

- **セカンダリーネットワーク**
  セカンダリーネットワークを設定し、仮想マシンをセカンダリーネットワークインターフェイスに接続し、割り当てられた IP アドレスに接続します。

8.2. 仮想マシンをデフォルトの POD ネットワークに接続する

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

ネットワークインターフェイスを通過してデフォルトの Pod ネットワークに到達するトラフィックは、ライブマイグレーション中に中断されます。

8.2.1. 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。

手順

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

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    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
```

8.2.2. デュアルスタック (IPv4 および IPv6) でのマスカレードモードの設定

cloud-init を使用して、新規仮想マシン (VM) を、デフォルトの Pod ネットワークで IPv6 と IPv4 の両方を使用するように設定できます。

仮想マシンインスタンス設定の **Network.pod.vmIPv6NetworkCIDR** フィールドは、VM の静的 IPv6 アドレスとゲートウェイ IP アドレスを決定します。これらは IPv6 トラフィックを仮想マシンにルーティングするために virt-launcher Pod で使用され、外部では使用されません。**Network.pod.vmIPv6NetworkCIDR** フィールドは、Classless Inter-Domain Routing (CIDR) 表記で IPv6 アドレスブロックを指定します。デフォルト値は **fd10:0:2::2/120** です。この値は、ネットワーク要件に基づいて編集できます。

仮想マシンが実行されている場合、仮想マシンの送受信トラフィックは、virt-launcher Pod の IPv4 アドレスと固有の IPv6 アドレスの両方にルーティングされます。次に、virt-launcher Pod は IPv4 トラフィックを仮想マシンの DHCP アドレスにルーティングし、IPv6 トラフィックを仮想マシンの静的に設定された IPv6 アドレスにルーティングします。

前提条件

- OpenShift Container Platform クラスターは、デュアルスタック用に設定された OVN-Kubernetes Container Network Interface (CNI) ネットワークプラグインを使用する必要があります。

手順

1. 新規の仮想マシン設定では、**masquerade** を指定したインターフェイスを追加し、cloud-init を使用して IPv6 アドレスとデフォルトゲートウェイを設定します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm-ipv6
spec:
  template:
    spec:
      domain:
        devices:
          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}"
```

8.2.3. ジャンボフレームのサポートについて

OVN-Kubernetes CNI プラグインを使用すると、デフォルトの Pod ネットワークに接続されている 2 つの仮想マシン (VM) 間でフラグメント化されていないジャンボフレームパケットを送信できます。ジャンボフレームの最大伝送単位 (MTU) 値は 1500 バイトを超えています。

VM は、クラスター管理者が設定したクラスターネットワークの MTU 値を、次のいずれかの方法で自動的に取得します。

- **libvirt**: ゲスト OS に VirtIO ドライバーの最新バージョンがあり、エミュレーションされたデバイスの Peripheral Component Interconnect (PCI) 設定レジスターを介して着信データを解釈できる場合。
- DHCP: ゲスト DHCP クライアントが DHCP サーバーの応答から MTU 値を読み取ることができる場合。

注記

VirtIO ドライバーを持たない Windows VM の場合、**netsh** または同様のツールを使用して MTU を手動で設定する必要があります。これは、Windows DHCP クライアントが MTU 値を読み取らないためです。

8.2.4. 関連情報

- クラスターネットワークの MTU 変更
- ネットワークでの MTU の最適化

8.3. サービスを使用して仮想マシンを公開する

Service オブジェクトを作成して、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。

8.3.1. サービスについて

Kubernetes サービスは一連の Pod で実行されているアプリケーションへのクライアントのネットワークアクセスを公開します。サービスは抽象化、負荷分散を提供し、タイプ **NodePort** と **LoadBalancer** の場合は外部世界への露出を提供します。

ClusterIP

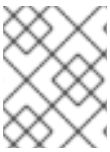
内部 IP アドレスでサービスを公開し、クラスター内の他のアプリケーションに DNS 名として公開します。1 つのサービスを複数の仮想マシンにマッピングできます。クライアントがサービスに接続しようとすると、クライアントのリクエストは使用可能なバックエンド間で負荷分散されます。**ClusterIP** はデフォルトのサービスタイプです。

NodePort

クラスター内の選択した各ノードの同じポートでサービスを公開します。**NodePort** は、ノード自体がクライアントから外部にアクセスできる限り、クラスターの外部からポートにアクセスできるようにします。

LoadBalancer

現在のクラウド（サポートされている場合）に外部ロードバランサーを作成し、固定の外部 IP アドレスをサービスに割り当てます。



注記

オンプレミスクラスターの場合、MetalLB Operator をデプロイすることで負荷分散サービスを設定できます。

関連情報

- MetalLB Operator のインストール
- MetalLB を使用するためのサービスの設定

8.3.2. 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]**

8.3.3. コマンドラインを使用したサービスの作成

コマンドラインを使用して、サービスを作成し、それを仮想マシンに関連付けることができます。

前提条件

- サービスをサポートするようにクラスターネットワークを設定しました。

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

1 **special: key** を **spec.template.metadata.labels** スタンザに追加します。

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: example-service
  namespace: example-namespace
spec:
# ...
  selector:
    special: key 1
  type: NodePort 2
  ports: 3
    protocol: TCP
    port: 80
    targetPort: 9376
    nodePort: 30000
```

1. **VirtualMachine** マニフェストの **spec.template.metadata.labels** スタンザに追加したラベルを指定します。
2. **ClusterIP**、**NodePort**、または **LoadBalancer** を指定します。
3. 仮想マシンから公開するネットワークポートとプロトコルのコレクションを指定します。

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 example-service.yaml
```

6. 仮想マシンを再起動して変更を適用します。

検証

- **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
```

8.3.4. 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

8.4. 内部 FQDN を使用した仮想マシンへのアクセス

ヘッドレスサービスを使用すると、安定した完全修飾ドメイン名 (FQDN) 上のデフォルトの内部 Pod ネットワークに接続されている仮想マシン (VM) にアクセスできます。

Kubernetes **ヘッドレスサービス** は、Pod のセットの表現にクラスター IP アドレスを割り当てないサービス形式です。ヘッドレスサービスは、サービスに単一の仮想 IP アドレスを提供する代わりに、サービスに関連付けられた Pod ごとに DNS レコードを作成します。特定の TCP ポートまたは UDP ポートを公開しなくても、FQDN を通じて仮想マシンを公開できます。

重要

OpenShift Container Platform Web コンソールを使用して VM を作成した場合、**VirtualMachine details** ページの **Overview** タブの **Network** タイルにその内部 FQDN が表示されます。VM への接続の詳細は、内部 FQDN を使用した仮想マシンへの接続 を参照してください。

8.4.1. CLI を使用したプロジェクトでのヘッドレスサービスの作成

namespace にヘッドレスサービスを作成するには、サービス YAML 定義に **clusterIP: None** パラメーターを追加します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. 次の例のように、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: mysubdomain (1)
spec:
  selector:
    expose: me (2)
  clusterIP: None (3)
  ports: (4)
  - protocol: TCP
    port: 1234
    targetPort: 1234
```

(1) サービスの名前。これは、**VirtualMachine** マニフェストファイルの **spec.subdomain** 属性と同じである必要があります。

(2) このサービスセレクターは、**VirtualMachine** マニフェストファイルの **expose:me** ラベルと一致する必要があります。

(3) ヘッドレスサービスを指定します。

(4) サービスによって公開されるポートのリスト。少なくとも 1 つのポートを定義する必要があります。これはヘッドレスサービスに影響を与えないため、任意の値にすることができます。

2. **サービス** マニフェストファイルを保存します。
3. 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 headless_service.yaml
```

8.4.2. CLI を使用した仮想マシンのヘッドレスサービスへのマッピング

内部の完全修飾ドメイン名 (FQDN) を使用してクラスター内から仮想マシン (VM) に接続するには、まず仮想マシンをヘッドレスサービスにマップする必要があります。仮想マシン設定ファイルに **spec.hostname** および **spec.subdomain** パラメーターを設定します。

サブドメインと名前が一致するヘッドレスサービスが存在する場合、**<vm.spec.hostname>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local** の形式で仮想マシンに対して一意の DNS A レコードが作成されます。

手順

1. 次のコマンドを実行して、**VirtualMachine** マニフェストを編集し、サービスセレクターラベルとサブドメインを追加します。

```
$ oc edit vm <vm_name>
```

VirtualMachine マニフェストファイルの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
spec:
  template:
    metadata:
      labels:
        expose: me (1)
    spec:
      hostname: "myvm" (2)
      subdomain: "mysubdomain" (3)
# ...
```

(1) **expose:me** ラベルは、以前に作成した **Service** マニフェストの **spec.selector** 属性と一致する必要があります。

(2) この属性が指定されていない場合、結果の DNS A レコードは **<vm.metadata.name>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local** の形式になります。

(3) **spec.subdomain** 属性は **Service** オブジェクトの **metadata.name** 値と一致する必要があります。

2. 変更を保存し、エディターを終了します。
3. 仮想マシンを再起動して変更を適用します。

8.4.3. 内部 FQDN を使用した仮想マシンへの接続

内部の完全修飾ドメイン名 (FQDN) を使用して仮想マシン (VM) に接続できます。

前提条件

- **virtctl** ツールがインストールされている。

- Web コンソールから仮想マシンの内部 FQDN を特定するか、仮想マシンをヘッドレスサービスにマッピングしている。内部 FQDN の形式は **<vm.spec.hostname>.<vm.spec.subdomain>.<vm.metadata.namespace>.svc.cluster.local** です。

手順

1. 次のコマンドを入力して仮想マシンコンソールに接続します。

```
$ virtctl console vm-fedora
```

2. 要求された FQDN を使用して仮想マシンに接続するには、次のコマンドを実行します。

```
$ ping myvm.mysubdomain.<namespace>.svc.cluster.local
```

出力例

```
PING myvm.mysubdomain.default.svc.cluster.local (10.244.0.57) 56(84) bytes of data.
64 bytes from myvm.mysubdomain.default.svc.cluster.local (10.244.0.57): icmp_seq=1 ttl=64
time=0.029 ms
```

上記の例では、**myvm.mysubdomain.default.svc.cluster.local** の DNS エントリーは **10.244.0.57** を指しています。これは、現在仮想マシンに割り当てられているクラスター IP アドレスです。

8.4.4. 関連情報

- サービスを使用した仮想マシンの公開

8.5. 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。

次の手順を実行すると、Linux ブリッジネットワークを作成し、そのネットワークに仮想マシンを接続できます。

1. Linux ブリッジノードネットワーク設定ポリシー (NNCP) を作成します。
2. Web コンソール または コマンドライン を使用して、Linux ブリッジネットワークアタッチメント定義 (NAD) を作成します。
3. Web コンソール または コマンドライン を使用して、NAD を認識するように仮想マシンを設定します。

8.5.1. Linux ブリッジ NNCP の作成

Linux ブリッジネットワークの **NodeNetworkConfigurationPolicy** (NNCP) マニフェストを作成できます。

前提条件

- Kubernetes NMState Operator がインストールされている。

手順

- **NodeNetworkConfigurationPolicy** マニフェストを作成します。この例には、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: br1-eth1-policy (1)
spec:
  desiredState:
    interfaces:
      - name: br1 (2)
        description: Linux bridge with eth1 as a port (3)
        type: linux-bridge (4)
        state: up (5)
        ipv4:
          enabled: false (6)
        bridge:
          options:
            stp:
              enabled: false (7)
          port:
            - name: eth1 (8)
```

(1) ポリシーの名前。

(2) インターフェイスの名前。

(3) オプション: 人間が判読できるインターフェイスの説明。

(4) インターフェイスのタイプ。この例では、ブリッジを作成します。

(5) 作成後のインターフェイスの要求された状態。

(6) この例では IPv4 を無効にします。

(7) この例では STP を無効にします。

(8) ブリッジが接続されているノード NIC。

8.5.2. Linux ブリッジ NAD の作成

OpenShift Container Platform Web コンソールまたはコマンドラインを使用して、Linux ブリッジネットワーク接続定義 (NAD) を作成できます。

8.5.2.1. Web コンソールを使用した Linux ブリッジ NAD の作成

OpenShift Container Platform Web コンソールを使用して、ネットワーク接続定義 (NAD) を作成して、Pod および仮想マシンに layer-2 ネットワークを提供できます。

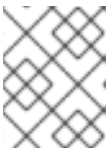
Linux ブリッジネットワーク接続定義は、仮想マシンを VLAN に接続するための最も効率的な方法です。

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。

手順

1. Web コンソールで、**Networking → NetworkAttachmentDefinitions** をクリックします。
2. **Create Network Attachment Definition**をクリックします。



注記

ネットワーク接続定義は Pod または仮想マシンと同じ namespace にある必要があります。

3. 一意の **Name** およびオプションの **Description** を入力します。
4. **Network Type** リストから **CNV Linux bridge** を選選択します。
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。

8.5.2.2. コマンドラインを使用した Linux ブリッジ NAD の作成

コマンドラインを使用してネットワークアタッチメント定義 (NAD) を作成し、Pod および仮想マシンに layer-2 ネットワークを提供できます。

NAD と仮想マシンは同じ namespace に存在する必要があります。

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。

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. 次の例のように、仮想マシンを **NetworkAttachmentDefinition** 設定に追加します。

```
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": false, (6)
    "vlan": 100, (7)
    "disableContainerInterface": "true",
    "preserveDefaultVlan": false (8)
  }'
```

(1) **NetworkAttachmentDefinition** オブジェクトの名前。

(2) オプション: ノード選択のアノテーションのキーと値のペア。**bridge-interface** は一部のノードに設定されるブリッジの名前に一致する必要があります。このアノテーションをネットワーク接続定義に追加する場合、仮想マシンインスタンスは **bridge-interface** ブリッジが接続されているノードでのみ実行されます。

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

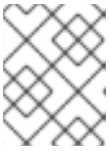
(4) このネットワーク接続定義のネットワークを提供する Container Network Interface (CNI) プラグインの実際の名前。異なる CNI を使用するのでない限り、このフィールドは変更しないでください。

(5) ノードに設定される Linux ブリッジの名前。

(6) オプション: MAC スプーフィングチェックを有効にするフラグ。**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。この属性により、Pod から出ることができる MAC アドレスは 1 つだけになり、MAC スプーフィング攻撃に対するセキュリティーが確保されます。

(7) オプション: VLAN タグ。ノードのネットワーク設定ポリシーでは、追加の VLAN 設定は必要ありません。

(8) オプション: 仮想マシンがデフォルト VLAN 経由でブリッジに接続するかどうかを示します。デフォルト値は **true** です。



注記

Linux ブリッジネットワーク接続定義は、仮想マシンを VLAN に接続するための最も効率的な方法です。

2. 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
```

8.5.3. 仮想マシンネットワークインターフェイスの設定

OpenShift Container Platform Web コンソールまたはコマンドラインを使用して、仮想マシンネットワークインターフェイスを設定できます。

8.5.3.1. Web コンソールを使用した仮想マシンネットワークインターフェイスの設定

OpenShift Container Platform Web コンソールを使用して、仮想マシンのネットワークインターフェイスを設定できます。

前提条件

- ネットワークのネットワーク接続定義を作成しました。

手順

1. **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンをクリックして、**VirtualMachine details** ページを表示します。
3. **Configuration** タブで、**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. インターフェイス名を入力し、**Network** リストからネットワーク接続定義を選択します。
6. **Save** をクリックします。
7. 仮想マシンを再起動して変更を適用します。

ネットワークフィールド

Name	説明

Name	説明
Name	ネットワークインターフェイスコントローラーの名前。
モデル	ネットワークインターフェイスコントローラーのモデルを示します。サポートされる値は **e1000e** および **virtio** です。
Network	利用可能なネットワーク接続定義のリスト。
タイプ	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。

8.5.3.2. コマンドラインを使用した仮想マシンネットワークインターフェイスの設定

コマンドラインを使用して、ブリッジネットワークの仮想マシンネットワークインターフェイスを設定できます。

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. 次の例のように、ブリッジインターフェイスとネットワークアタッチメント定義を仮想マシン設定に追加します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          interfaces:
            - bridge: {}
```

```
          name: bridge-net (1)
# ...
    networks:
      - name: bridge-net (2)
        multus:
          networkName: 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。

2. 設定を適用します。

```
$ oc apply -f example-vm.yaml
```

3. 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。

8.6. 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。

- SR-IOV ネットワークデバイスの設定
- SR-IOV ネットワークの設定
- 仮想マシンを SR-IOV ネットワークに接続する

8.6.1. 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: Virtual Function の最大転送単位 (MTU) の値を指定します。MTU の最大値は NIC モデルによって異なります。
7. SR-IOV 物理ネットワークデバイス用に作成する仮想機能 (VF) の数を指定します。Intel ネットワークインターフェイスコントローラー (NIC) の場合、VF の数はデバイスがサポートする VF の合計よりも大きくすることはできません。Mellanox NIC の場合、VF の数は **127** よりも大きくすることはできません。

8 **nicSelector** マッピングは、Operator が設定するイーサネットデバイスを選択します。すべてのパラメーターの値を指定する必要はありません。意図せずにイーサネットデバイスを選択する可能性を最低限に抑えるために、イーサネットアダプターを正確に特定できるようにすることが推奨されます。**rootDevices** を指定する場合、**vendor**、**deviceID**、または **pfName** の値も指定する必要があります。**pfNames** と **rootDevices** の両方を同時に指定する場合、それらが同一のデバイスをポイントすることを確認します。

9 オプション: SR-IOV ネットワークデバイスのベンダー 16 進コードを指定します。許可される値は **8086** または **15b3** のいずれかのみになります。

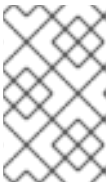
10 オプション: SR-IOV ネットワークデバイスのデバイス 16 進コードを指定します。許可される値は **158b**、**1015**、**1017** のみになります。

11 オプション: このパラメーターは、1 つ以上のイーサネットデバイスの物理機能 (PF) 名の配列を受け入れます。

12 このパラメーターは、イーサネットデバイスの物理機能に関する 1 つ以上の PCI バスアドレスの配列を受け入れます。以下の形式でアドレスを指定します: **0000:02:00.1**

13 OpenShift Virtualization の仮想機能には、**vfio-pci** ドライバータイプが必要です。

14 オプション: Remote Direct Memory Access (RDMA) モードを有効にするかどうかを指定します。Mellanox カードの場合、**isRdma** を **false** に設定します。デフォルト値は **false** です。



注記

isRDMA フラグが **true** に設定される場合、引き続き RDMA 対応の VF を通常のネットワークデバイスとして使用できます。デバイスはどちらのモードでも使用できます。

2. オプション: SR-IOV 対応のクラスターノードにまだラベルが付いていない場合は、**SriovNetworkNodePolicy.Spec.NodeSelector** でラベルを付けます。ノードのラベル付けの詳細は、「ノードのラベルを更新する方法について」を参照してください。

3. **SriovNetworkNodePolicy** オブジェクトを作成します。

```
$ oc create -f <name>-sriov-node-network.yaml
```

ここで、**<name>** はこの設定の名前を指定します。

設定の更新が適用された後に、**sriov-network-operator** namespace のすべての Pod が **Running** ステータスに移行します。

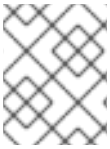
4. SR-IOV ネットワークデバイスが設定されていることを確認するには、以下のコマンドを実行します。**<node_name>** を、設定したばかりの SR-IOV ネットワークデバイスを持つノードの名前に置き換えます。

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

8.6.2. 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>
```

8.6.3. コマンドラインを使用して仮想マシンを SR-IOV ネットワークに接続する

仮想マシン (VM) 設定にネットワークの詳細を追加することで、仮想マシンを SR-IOV ネットワークに接続できます。

手順

1. 次の例のように、SR-IOV ネットワークの詳細を仮想マシン設定の **spec.domain.devices.interfaces** スタンザと **spec.networks** スタンザに追加します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  domain:
    devices:
      interfaces:
      - name: nic1 (1)
        sriov: {}
  networks:
  - name: nic1 (2)
    multus:
        networkName: sriov-network (3)
# ...
```

(1) SR-IOV インターフェイスの一意の名前を指定します。

(2) SR-IOV インターフェイスの名前を指定します。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。

(3) SR-IOV ネットワーク割り当て定義の名前を指定します。

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。

8.6.4. Web コンソールを使用して仮想マシンを SR-IOV ネットワークに接続する

仮想マシン設定にネットワークの詳細を追加することで、仮想マシンを SR-IOV ネットワークに接続できます。

前提条件

- ネットワークのネットワークアタッチメント定義を作成した。

手順

1. **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンをクリックして、**VirtualMachine details** ページを表示します。
3. **Configuration** タブで、**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. インターフェイス名を入力します。

6. **Network** リストから SR-IOV ネットワークアタッチメント定義を選択します。
7. **Type** リストから **SR-IOV** を選択します。
8. オプション: ネットワーク **Model** または **Mac address** を追加します。
9. **Save** をクリックします。
10. 仮想マシンを再起動またはライブマイグレーションして、変更を適用します。

8.6.5. 関連情報

- パフォーマンスを向上させるための DPDK ワークロードの設定

8.7. SR-IOV での DPDK の使用

Data Plane Development Kit (DPDK) は、高速パケット処理用のライブラリーとドライバーのセットを提供するものです。

SR-IOV ネットワーク上で DPDK ワークロードを実行するようにクラスターと仮想マシンを設定できます。

8.7.1. DPDK ワークロード用のクラスター設定

ネットワークパフォーマンスを向上させるために、Data Plane Development Kit (DPDK) ワークロードを実行するように OpenShift Container Platform クラスターを設定できます。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- SR-IOV Network Operator がインストールされている。
- Node Tuning Operator がインストールされている。

手順

1. コンピュートノードトポロジーのマッピングを実行し、DPDK アプリケーション用に分離する Non-Uniform Memory Access (NUMA) CPU と、オペレーティングシステム (OS) 用に予約する NUMA CPU を決定します。
2. コンピュートノードのサブセットにカスタムロール (例: **worker-dpdk**) のラベルを追加します。

```
$ oc label node <node_name> node-role.kubernetes.io/worker-dpdk=""
```

3. **spec.machineConfigSelector** オブジェクトに **worker-dpdk** ラベルを含む新しい **MachineConfigPool** マニフェストを作成します。

   MachineConfigPool マニフェストの例

```
apiVersion: machineconfiguration.openshift.io/v1
```

```
kind: MachineConfigPool
metadata:
  name: worker-dpdk
  labels:
    machineconfiguration.openshift.io/role: worker-dpdk
spec:
  machineConfigSelector:
    matchExpressions:
      - key: machineconfiguration.openshift.io/role
        operator: In
        values:
          - worker
          - worker-dpdk
  nodeSelector:
    matchLabels:
      node-role.kubernetes.io/worker-dpdk: ""
```

4. 前の手順で作成したラベル付きノードとマシン設定プールに適用する **PerformanceProfile** マニフェストを作成します。パフォーマンスプロファイルは、DPDK アプリケーション用に分離された CPU とハウスキーピング用に予約された CPU を指定します。

   PerformanceProfile マニフェストの例

```
apiVersion: performance.openshift.io/v2
kind: PerformanceProfile
metadata:
  name: profile-1
spec:
  cpu:
    isolated: 4-39,44-79
    reserved: 0-3,40-43
  globallyDisableIrqLoadBalancing: true
  hugepages:
    defaultHugepagesSize: 1G
    pages:
    - count: 8
      node: 0
      size: 1G
  net:
    userLevelNetworking: true
  nodeSelector:
    node-role.kubernetes.io/worker-dpdk: ""
  numa:
    topologyPolicy: single-numa-node
```

注記

MachineConfigPool マニフェストと **PerformanceProfile** マニフェストを適用すると、コンピュートノードが自動的に再起動します。

5. **PerformanceProfile** オブジェクトの **status.runtimeClass** フィールドから、生成された **RuntimeClass** リソースの名前を取得します。

```
$ oc get performanceprofiles.performance.openshift.io profile-1 -
o=jsonpath='{.status.runtimeClass}{"\n"}'
```

6. **HyperConverged** カスタムリソース (CR) を編集して、以前に取得した **RuntimeClass** 名を **virt-launcher** Pod のデフォルトのコンテナーランタイムクラスとして設定します。

```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
    --type='json' -p='[{"op": "add", "path": "/spec/defaultRuntimeClass", "value":"<runtimeclass-
name>"}]'
```

注記

HyperConverged CR を編集すると、変更の適用後に作成されるすべての仮想マシンに影響するグローバル設定が変更されます。

7. DPDK 対応コンピュートノードが同時マルチスレッド (SMT) を使用している場合は、**HyperConverged** CR を編集して **AlignCPUs** イネーブラーを有効にします。

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

注記

AlignCPUs を有効にすると、OpenShift Virtualization は、エミュレーターのスレッド分離を使用する場合に CPU 総数を偶数パリティーにするために、追加で最大 2 つの専用 CPU を要求できるようになります。

8. **spec.deviceType** フィールドを **vfio-pci** に設定して **SriovNetworkNodePolicy** オブジェクトを作成します。

SriovNetworkNodePolicy マニフェストの例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: policy-1
  namespace: openshift-sriov-network-operator
spec:
  resourceName: intel_nics_dpdk
  deviceType: vfio-pci
  mtu: 9000
  numVfs: 4
  priority: 99
  nicSelector:
    vendor: "8086"
    deviceID: "1572"
    pfNames:
      - eno3
    rootDevices:
      - "0000:19:00.2"
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
```

関連情報

- CPU マネージャーおよび Topology Manager の使用
- Huge Page の設定
- カスタムマシン設定プールの作成

8.7.2. DPDK ワークロード用のプロジェクト設定

SR-IOV ハードウェアで DPDK ワークロードを実行するプロジェクトを設定できます。

前提条件

- DPDK ワークロードを実行するようにクラスターが設定されている。

手順

1. DPDK アプリケーションの namespace を作成します。

```
$ oc create ns dpdk-checkup-ns
```

2. **SriovNetworkNodePolicy** オブジェクトを参照する **SriovNetwork** オブジェクトを作成します。**SriovNetwork** オブジェクトの作成時に、SR-IOV Network Operator は **NetworkAttachmentDefinition** オブジェクトを自動的に作成します。

   SriovNetwork マニフェストの例

```
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetwork
metadata:
  name: dpdk-sriovnetwork
  namespace: openshift-sriov-network-operator
spec:
  ipam: |
    {
      "type": "host-local",
      "subnet": "10.56.217.0/24",
      "rangeStart": "10.56.217.171",
      "rangeEnd": "10.56.217.181",
      "routes": [{
        "dst": "0.0.0.0/0"
      }],
      "gateway": "10.56.217.1"
    }
  networkNamespace: dpdk-checkup-ns (1)
  resourceName: intel_nics_dpdk (2)
  spoofChk: "off"
  trust: "on"
  vlan: 1019
```

(1) **NetworkAttachmentDefinition** オブジェクトがデプロイされる namespace。

(2)

DPDK ワークロード用クラスターの設定時に作成された **SriovNetworkNodePolicy** オブジェクトの **spec.resourceName** 属性値。

3. オプション: 仮想マシンレイテンシーチェックアップを実行して、ネットワークが適切に設定されていることを確認します。
4. オプション: DPDK チェックアップを実行して、namespace が DPDK ワークロード用に準備できているか確認します。

関連情報

- プロジェクトの使用
- 仮想マシンのレイテンシーチェックアップ
- DPDK チェックアップ

8.7.3. DPDK ワークロード用の仮想マシン設定

仮想マシン (VM) 上で Data Packet Development Kit (DPDK) ワークロードを実行すると、レイテンシーの短縮とスループットが向上し、ユーザー空間でのパケット処理を高速化できます。DPDK は、ハードウェアベースの I/O 共有に SR-IOV ネットワークを使用します。

前提条件

- DPDK ワークロードを実行するようにクラスターが設定されている。
- 仮想マシンを実行するプロジェクトを作成し、設定している。

手順

1. **VirtualMachine** マニフェストを編集して、SR-IOV ネットワークインターフェイス、CPU トポロジー、CRI-O アノテーション、Huge Page に関する情報を格納します。

VirtualMachine マニフェストの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: rhel-dpdk-vm
spec:
  running: true
  template:
    metadata:
      annotations:
        cpu-load-balancing.crio.io: disable (1)
        cpu-quota.crio.io: disable (2)
        irq-load-balancing.crio.io: disable (3)
    spec:
      domain:
        cpu:
          sockets: 1 (4)
          cores: 5 (5)
          threads: 2
```

```
        dedicatedCpuPlacement: true
        isolateEmulatorThread: true
      interfaces:
        - masquerade: {}
          name: default
        - model: virtio
          name: nic-east
          pciAddress: '0000:07:00.0'
          sriov: {}
      networkInterfaceMultiqueue: true
      rng: {}
    memory:
      hugepages:
        pageSize: 1Gi (6)
        guest: 8Gi
    networks:
      - name: default
        pod: {}
      - multus:
          networkName: dpdk-net (7)
        name: nic-east
# ...
```

1. このアノテーションは、コンテナーが使用する CPU に対するロードバランシングが無効であることを示します。
2. このアノテーションは、コンテナーが使用する CPU に対する CPU クォータが無効であることを示します。
3. このアノテーションは、コンテナーが使用する CPU に対する Interrupt Request (IRQ) のロードバランシングが無効であることを示します。
4. 仮想マシン内のソケットの数。同じ Non-Uniform Memory Access (NUMA) ノードから CPU をスケジュールするには、このフィールドを **1** に設定する必要があります。
5. 仮想マシン内のコアの数。値は **1** 以上とします。この例では、仮想マシンは 5 個のハイパースレッドか 10 個の CPU でスケジュールされています。
6. Huge Page のサイズ。x86-64 アーキテクチャーの有効な値は 1Gi と 2Mi です。この例は、サイズが 1Gi の 8 個の Huge Page に対する要求です。
7. SR-IOV **NetworkAttachmentDefinition** オブジェクトの名前。

2. エディターを保存し、終了します。
3. **VirtualMachine** マニフェストを適用します。

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

4. ゲストオペレーティングシステムを設定します。次の例は、RHEL 8 OS の設定手順を示しています。
   a. GRUB ブートローダーコマンドラインインターフェイスを使用して、Huge Page を設定します。次の例では、1G の Huge Page を 8 個指定しています。

```
$ grubby --update-kernel=ALL --args="default_hugepagesz=1GB hugepagesz=1G
hugepages=8"
```

b. TuneD アプリケーションで **cpu-partitioning** プロファイルを使用して低レイテンシーチューニングを実現するには、次のコマンドを実行します。

```
$ dnf install -y tuned-profiles-cpu-partitioning
```

```
$ echo isolated_cores=2-9 > /etc/tuned/cpu-partitioning-variables.conf
```

最初の 2 つの CPU (0 と 1) はハウスキーピングタスク用に確保され、残りは DPDK アプリケーション用に分離されます。

```
$ tuned-adm profile cpu-partitioning
```

c. **driverctl** デバイスドライバー制御ユーティリティーを使用して、SR-IOV NIC ドライバーをオーバーライドします。

```
$ dnf install -y driverctl
```

```
$ driverctl set-override 0000:07:00.0 vfio-pci
```

5. 仮想マシンを再起動して変更を適用します。

8.8. OVN-KUBERNETES セカンダリーネットワークへの仮想マシンの接続

仮想マシンを Open Virtual Network (OVN)-Kubernetes セカンダリーネットワークに接続できます。OpenShift Virtualization は、OVN-Kubernetes のレイヤー 2 およびローカルネットトポロジーをサポートします。

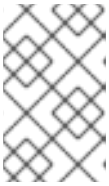
- レイヤー 2 トポロジーは、クラスター全体の論理スイッチでワークロードを接続します。OVN-Kubernetes Container Network Interface (CNI) プラグインは、Geneve (Generic Network Virtualization Encapsulation) プロトコルを使用してノード間にオーバーレイネットワークを作成します。このオーバーレイネットワークを使用すると、追加の物理ネットワークインフラストラクチャーを設定することなく、さまざまなノード上の仮想マシンを接続できます。
- ローカルネットトポロジーは、セカンダリーネットワークを物理アンダーレイに接続します。これにより、east-west クラスタートラフィックとクラスター外で実行されているサービスへのアクセスの両方が可能になります。ただし、クラスターノード上の基盤となる Open vSwitch (OVS) システムの追加設定が必要です。

注記

OVN-Kubernetes セカンダリーネットワークは、マルチネットワークポリシー API と互換性があります。これにより、仮想マシンとの間のトラフィックフローの制御に **MultiNetworkPolicy** カスタムリソース定義 (CRD) を使用できます。**ipBlock** 属性を使用して、特定の CIDR ブロックに対するネットワークポリシーの受信および送信ルールを定義できます。

OVN-Kubernetes セカンダリーネットワークを設定し、そのネットワークに仮想マシンを接続するには、次の手順を実行します。

1. ネットワークアタッチメント定義 (NAD) を作成して、OVN-Kubernetes セカンダリーネットワークの設定 を行います。



注記

ローカルネットトポロジーの場合は、NAD を作成する前に **NodeNetworkConfigurationPolicy** オブジェクトを作成して OVS ブリッジを設定する 必要があります。

2. ネットワークの詳細を仮想マシン仕様に追加して、仮想マシンを OVN-Kubernetes セカンダリーネットワークに接続 します。

8.8.1. OVN-Kubernetes NAD の作成

OpenShift Container Platform Web コンソールまたは CLI を使用して、OVN-Kubernetes レイヤー 2 またはローカルネットネットワーク接続定義 (NAD) を作成できます。

注記

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。

8.8.1.1. CLI を使用してレイヤー 2 トポロジーの NAD を作成する

Pod をレイヤー 2 オーバーレイネットワークに接続する方法を説明するネットワーク接続定義 (NAD) を作成できます。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. **NetworkAttachmentDefinition** オブジェクトを作成します。

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: l2-network
  namespace: my-namespace
spec:
  config: |2
    {
            "cniVersion": "0.3.1", (1)
            "name": "my-namespace-l2-network", (2)
            "type": "ovn-k8s-cni-overlay", (3)
            "topology":"layer2", (4)
            "mtu": 1300, (5)
            "netAttachDefName": "my-namespace/l2-network" (6)
    }
```

1. CNI 仕様のバージョン。必要な値は **0.3.1** です。
2. ネットワークの名前。この属性には namespace がありません。たとえば、**l2-network** という名前のネットワークを、2 つの異なる namespace に存在する 2 つの異なる **NetworkAttachmentDefinition** オブジェクトから参照させることができます。この機能は、異なる namespace の仮想マシンを接続する場合に役立ちます。
3. 設定する CNI プラグインの名前。必要な値は **ovn-k8s-cni-overlay** です。
4. ネットワークのトポロジー設定。必要な値は **layer2** です。
5. オプション: 最大伝送単位 (MTU) の値。デフォルト値はカーネルによって自動的に設定されます。
6. **NetworkAttachmentDefinition** オブジェクトの **metadata** スタンザ内の **namespace** および **name** フィールドの値。

注記

上記の例では、サブネットを定義せずにクラスター全体のオーバーレイを設定します。これは、ネットワークを実装する論理スイッチがレイヤー 2 通信のみを提供することを意味します。仮想マシンの作成時に、静的 IP アドレスを設定するか、動的 IP アドレス用にネットワーク上に DHCP サーバーをデプロイすることによって、IP アドレスを設定する必要があります。

2. マニフェストを適用します。

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

8.8.1.2. CLI を使用してローカルネットトポロジーの NAD を作成する

Pod を基盤となる物理ネットワークに接続する方法を説明するネットワーク接続定義 (NAD) を作成できます。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- Kubernetes NMState Operator がインストールされている。
- OVN-Kubernetes セカンダリーネットワークを Open vSwitch (OVS) ブリッジにマップするための **NodeNetworkConfigurationPolicy** オブジェクトを作成している。

手順

1. **NetworkAttachmentDefinition** オブジェクトを作成します。

```
apiVersion: k8s.cni.cncf.io/v1
kind: NetworkAttachmentDefinition
metadata:
  name: localnet-network
  namespace: default
```

```
spec:
  config: |2
    {
            "cniVersion": "0.3.1", (1)
            "name": "localnet-network", (2)
            "type": "ovn-k8s-cni-overlay", (3)
            "topology": "localnet", (4)
            "netAttachDefName": "default/localnet-network" (5)
    }
```

1. CNI 仕様のバージョン。必要な値は **0.3.1** です。
2. ネットワークの名前。この属性は、OVS ブリッジマッピングを定義する **NodeNetworkConfigurationPolicy** オブジェクトの **spec.desiredState.ovn.bridge-mappings.localnet** フィールドの値と一致する必要があります。
3. 設定する CNI プラグインの名前。必要な値は **ovn-k8s-cni-overlay** です。
4. ネットワークのトポロジー設定。必要な値は **localnet** です。
5. **NetworkAttachmentDefinition** オブジェクトの **metadata** スタンザ内の **namespace** および **name** フィールドの値。

2. マニフェストを適用します。

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

8.8.1.3. Web コンソールを使用してレイヤー 2 トポロジーの NAD を作成する

Pod をレイヤー 2 オーバーレイネットワークに接続する方法を記述したネットワーク接続定義 (NAD) を作成できます。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにアクセスできる。

手順

1. Web コンソールで、**Networking → NetworkAttachmentDefinitions** に移動します。
2. **Create Network Attachment Definition**をクリックします。ネットワーク接続定義は、それを使用する Pod または仮想マシンと同じ namespace にある必要があります。
3. 一意の **Name** およびオプションの **Description** を入力します。
4. **Network Type** リストで **OVN Kubernetes L2 overlay network**を選択します。
5. **Create** をクリックします。

8.8.1.4. Web コンソールを使用してローカルネットトポロジーの NAD を作成する

OpenShift Container Platform Web コンソールを使用してネットワーク接続定義 (NAD) を作成し、ワークロードを物理ネットワークに接続できます。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにアクセスできる。
- **nmstate** を使用して、ローカルネットから OVS ブリッジへのマッピングを設定します。

手順

1. Web コンソールで、**Networking → NetworkAttachmentDefinitions** に移動します。
2. **Create Network Attachment Definition**をクリックします。ネットワーク接続定義は、それを使用する Pod または仮想マシンと同じ namespace にある必要があります。
3. 一意の **Name** およびオプションの **Description** を入力します。
4. **Network Type** リストで **OVN Kubernetes secondary localnet network**を選択します。
5. **Bridge mapping** フィールドに、事前に設定されたローカルネット識別子の名前を入力します。
6. オプション: MTU を指定した値に明示的に設定できます。デフォルト値はカーネルにより選択されます。
7. オプション: VLAN 内のトラフィックをカプセル化します。デフォルト値は none です。
8. **Create** をクリックします。

8.8.2. OVN-Kubernetes セカンダリーネットワークへの仮想マシンの接続

OpenShift Container Platform Web コンソールまたは CLI を使用して、仮想マシンを OVN-Kubernetes セカンダリーネットワークインターフェイスに接続できます。

8.8.2.1. CLI を使用した OVN-Kubernetes セカンダリーネットワークへの仮想マシンの接続

仮想マシン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 OVN-Kubernetes セカンダリーネットワークに接続できます。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 次の例のように、**VirtualMachine** マニフェストを編集して OVN-Kubernetes セカンダリーネットワークインターフェイスの詳細を追加します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-server
spec:
  running: true
  template:
    spec:
      domain:
```

```
      devices:
        interfaces:
        - name: secondary 1
          bridge: {}
      resources:
        requests:
          memory: 1024Mi
    networks:
    - name: secondary 2
      multus:
        networkName: <nad_name> 3
# ...
```

1 OVN-Kubernetes セカンダリーインターフェイスの名前。

2 ネットワークの名前。これは、**spec.template.spec.domain.devices.interfaces.name** フィールドの値と一致する必要があります。

3 **NetworkAttachmentDefinition** オブジェクトの名前。

2. **VirtualMachine** マニフェストを適用します。

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

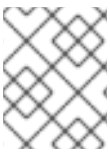
3. オプション: 実行中の仮想マシンを編集している場合は、変更を有効にするためにこれを再起動する必要があります。

8.8.3. 関連情報

- OVN-Kubernetes 追加ネットワークの設定
- Kubernetes NMState Operator について
- OVN-Kubernetes 追加ネットワークマッピングの設定
- ネットワーク追加割り当ての設定

8.9. ホットプラグ対応のセカンダリーネットワークインターフェイス

仮想マシンを停止せずに、セカンダリーネットワークインターフェイスを追加または削除できます。OpenShift Virtualization は、VirtIO デバイスドライバーを使用するセカンダリーインターフェイスのホットプラグをサポートしています。



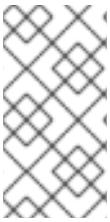
注記

ホットアンプラグは、Single Root I/O Virtualization (SR-IOV) インターフェイスではサポートされていません。

8.9.1. VirtIO の制限事項

各 VirtIO インターフェイスは、仮想マシン内の限られたペリフェラル接続インターフェイス (PCI) スロットの 1 つを使用します。合計 32 個のスロットが利用可能です。PCI スロットは他のデバイスでも使用され、事前に予約する必要があるため、オンデマンドでスロットを利用できない場合があります。

OpenShift Virtualization は、ホットプラグインターフェイス用に最大 4 つのスロットを予約します。これには、接続されている既存のネットワークインターフェイスが含まれます。たとえば、仮想マシンにプラグインされた既存のインターフェイスが 2 つある場合は、さらに 2 つのネットワークインターフェイスをホットプラグできます。



注記

ホットプラグに使用できる実際のスロット数もマシンのタイプによって異なります。たとえば、q35 マシンタイプのデフォルトの PCI トポロジーは、1 台の追加 PCIe デバイスのホットプラグをサポートしています。PCI トポロジーとホットプラグのサポートの詳細は、libvirt のドキュメント を参照してください。

インターフェイスをホットプラグした後に仮想マシンを再起動すると、そのインターフェイスは標準ネットワークインターフェイスの一部になります。

8.9.2. CLI を使用したセカンダリーネットワークインターフェイスのホットプラグ

仮想マシンの実行中に、セカンダリーネットワークインターフェイスを仮想マシンにホットプラグします。

前提条件

- ネットワークアタッチメント定義は、仮想マシンと同じ namespace で設定されます。
- **virtctl** ツールがインストールされている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. ネットワークインターフェイスをホットプラグする仮想マシンが実行していない場合は、次のコマンドを使用して仮想マシンを起動します。

```
$ virtctl start <vm_name> -n <namespace>
```

2. 次のコマンドを使用して、実行中の仮想マシンに新しいネットワークインターフェイスを追加します。仮想マシン仕様を編集すると、新しいネットワークインターフェイスが仮想マシンおよび仮想マシンインスタンス (VMI) 設定に追加されますが、実行中の仮想マシンには接続されません。

```
$ oc edit vm <vm_name>
```

仮想マシン設定の例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: vm-fedora
template:
  spec:
    domain:
      devices:
        interfaces:
```

```
      - name: defaultnetwork
        masquerade: {}
      # new interface
      - name: <secondary_nic> 1
        bridge: {}
  networks:
  - name: defaultnetwork
    pod: {}
  # new network
  - name: <secondary_nic> 2
    multus:
      networkName: <nad_name> 3
# ...
```

1. 新しいネットワークインターフェイスの名前を指定します。
2. ネットワークの名前を指定します。これは、**template.spec.domain.devices.interfaces** リストで定義した新しいネットワークインターフェイスの **name** と同じである必要があります。
3. **NetworkAttachmentDefinition** オブジェクトの名前を指定します。

3. 実行中の仮想マシンにネットワークインターフェイスを接続するには、次のコマンドを実行して仮想マシンのライブマイグレーションを行います。

```
$ virtctl migrate <vm_name>
```

検証

1. 次のコマンドを使用して、仮想マシンのライブマイグレーションが成功したことを確認します。

```
$ oc get VirtualMachineInstanceMigration -w
```

出力例

```
NAME                        PHASE            VMI
kubevirt-migrate-vm-lj62q   Scheduling       vm-fedora
kubevirt-migrate-vm-lj62q   Scheduled        vm-fedora
kubevirt-migrate-vm-lj62q   PreparingTarget   vm-fedora
kubevirt-migrate-vm-lj62q   TargetReady      vm-fedora
kubevirt-migrate-vm-lj62q   Running          vm-fedora
kubevirt-migrate-vm-lj62q   Succeeded        vm-fedora
```

2. VMI ステータスをチェックして、新しいインターフェイスが仮想マシンに追加されていることを確認します。

```
$ oc get vmi vm-fedora -ojsonpath="{ @.status.interfaces }"
```

出力例

```
[
```

```
  {
    "infoSource": "domain, guest-agent",
    "interfaceName": "eth0",
    "ipAddress": "10.130.0.195",
    "ipAddresses": [
      "10.130.0.195",
      "fd02:0:0:3::43c"
    ],
    "mac": "52:54:00:0e:ab:25",
    "name": "default",
    "queueCount": 1
  },
  {
    "infoSource": "domain, guest-agent, multus-status",
    "interfaceName": "eth1",
    "mac": "02:d8:b8:00:00:2a",
    "name": "bridge-interface", 1
    "queueCount": 1
  }
]
```


1 ホットプラグされたインターフェイスが VMI ステータスに表示されます。

8.9.3. CLI を使用したセカンダリーネットワークインターフェイスのホットアンプラグ

実行中の仮想マシンから、セカンダリーネットワークインターフェイスを削除できます。

注記

ホットアンプラグは、Single Root I/O Virtualization (SR-IOV) インターフェイスではサポートされていません。

前提条件

- 仮想マシンが実行している必要があります。
- 仮想マシンは、OpenShift Virtualization 4.14 以降を実行しているクラスター上に作成する必要があります。
- 仮想マシンにはブリッジネットワークインターフェイスが接続されている必要があります。

手順

1. 仮想マシン仕様を編集して、セカンダリーネットワークインターフェイスをホットアンプラグします。インターフェイスの状態を **absent** に設定すると、ネットワークインターフェイスがゲストから切り離されますが、そのインターフェイスは Pod 内にまだ存在しています。

```
$ oc edit vm <vm_name>
```

仮想マシン設定の例

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

```
metadata:
  name: vm-fedora
template:
  spec:
    domain:
      devices:
        interfaces:
          - name: defaultnetwork
            masquerade: {}
          # set the interface state to absent
          - name: <secondary_nic>
            state: absent (1)
            bridge: {}
    networks:
      - name: defaultnetwork
        pod: {}
      - name: <secondary_nic>
        multus:
          networkName: <nad_name>
# ...
```

(1) インターフェイスの状態を **absent** に設定して、実行中の仮想マシンから切り離します。仮想マシン仕様からインターフェイスの詳細を削除しても、セカンダリーネットワークインターフェイスはホットアンプラグされません。

2. 仮想マシンを移行して、Pod からインターフェイスを削除します。

```
$ virtctl migrate <vm_name>
```

8.9.4. 関連情報

- virtctl のインストール
- Linux ブリッジネットワーク接続定義の作成
- Linux ブリッジネットワークへの仮想マシンの割り当て
- SR-IOV ネットワーク接続定義の作成
- 仮想マシンの SR-IOV ネットワークへの接続

8.10. 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。

8.10.1. サービスメッシュへの仮想マシンの追加

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。

重要

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。

前提条件

- Service Mesh Operators がインストールされました。
- Service Mesh コントロールプレーンを作成しました。
- 仮想マシンプロジェクトを Service Mesh メンバーロールに追加しました。

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。

8.10.2. 関連情報

- サービスメッシュオペレーターのインストール
- Service Mesh コントロールプレーンの作成
- Service Mesh メンバーロールへのプロジェクトの追加

8.11. ライブマイグレーション用の専用ネットワークの設定

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。

8.11.1. 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。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。
- 各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) があります。
- ライブマイグレーション用の NIC は同じ VLAN に接続されます。

手順

1. 次の例に従って、**NetworkAttachmentDefinition** マニフェストを作成します。

設定ファイルのサンプル

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: my-secondary-network 1
  namespace: openshift-cnv 2
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "migration-bridge",
    "type": "macvlan",
    "master": "eth1", 3
    "mode": "bridge",
    "ipam": {
      "type": "whereabouts", 4
      "range": "10.200.5.0/24" 5
    }
  }'
```

1. **NetworkAttachmentDefinition** オブジェクトの名前を指定します。
2. 3. ライブマイグレーションに使用する NIC の名前を指定します。
4. NAD にネットワークを提供する CNI プラグインの名前を指定します。
5.

セカンダリーネットワークの IP アドレス範囲を指定します。この範囲は、メインネットワークの IP アドレスと重複してはなりません。

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。

HyperConverged マニフェストの例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  liveMigrationConfig:
    completionTimeoutPerGiB: 800
    network: <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を指定します。

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}'
```

8.11.2. Web コンソールを使用して専用ネットワークを選択する

OpenShift Container Platform Web コンソールを使用して、ライブマイグレーション用の専用ネットワークを選択できます。

前提条件

- ライブマイグレーション用に Multus ネットワークが設定されている。

手順

1. OpenShift Container Platform Web コンソールで **Virtualization** > **Overview** に移動します。

2. **Settings** タブをクリックし、**Live migration** をクリックします。
3. **Live migration network** リストからネットワークを選択します。

8.11.3. 関連情報

- ライブマイグレーションの制限およびタイムアウトの設定

8.12. IP アドレスの設定と表示

仮想マシンを作成するときに IP アドレスを設定できます。IP アドレスは、cloud-init でプロビジョニングされます。

仮想マシンの IP アドレスは、OpenShift Container Platform Web コンソールまたはコマンドラインを使用して表示できます。ネットワーク情報は QEMU ゲストエージェントによって収集されます。

8.12.1. 仮想マシンの IP アドレスの設定

Web コンソールまたはコマンドラインを使用して仮想マシンを作成するときに、静的 IP アドレスを設定できます。

コマンドラインを使用して仮想マシンを作成するときに、動的 IP アドレスを設定できます。

IP アドレスは、cloud-init でプロビジョニングされます。

8.12.1.1. コマンドラインを使用して仮想マシンを作成するときに IP アドレスを設定する

仮想マシンを作成するときに、静的または動的 IP アドレスを設定できます。IP アドレスは、cloud-init でプロビジョニングされます。

注記

VM が Pod ネットワークに接続されている場合、更新しない限り、Pod ネットワークインターフェイスがデフォルトルートになります。

前提条件

- 仮想マシンはセカンダリーネットワークに接続されている。
- 仮想マシンの動的 IP を設定するために、セカンダリーネットワーク上で使用できる DHCP サーバーがある。

手順

- 仮想マシン設定の **spec.template.spec.volumes.cloudInitNoCloud.networkData** スタンザを編集します。
  - 動的 IP アドレスを設定するには、インターフェイス名を指定し、DHCP を有効にします。

```
kind: VirtualMachine
spec:
# ...
  template:
  # ...
```

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

1 インターフェイス名を指定します。

- 静的 IP を設定するには、インターフェイス名と IP アドレスを指定します。

```
kind: VirtualMachine
spec:
# ...
  template:
  # ...
    spec:
      volumes:
      - cloudInitNoCloud:
          networkData: |
            version: 2
            ethernets:
              eth1: 1
                addresses:
                - 10.10.10.14/24 2
```

1 インターフェイス名を指定します。

2 静的 IP アドレスを指定します。

8.12.2. 仮想マシンの IP アドレスの表示

仮想マシンの IP アドレスは、OpenShift Container Platform Web コンソールまたはコマンドラインを使用して表示できます。

ネットワーク情報は QEMU ゲストエージェントによって収集されます。

8.12.2.1. Web コンソールを使用した仮想マシンの IP アドレスの表示

OpenShift Container Platform Web コンソールを使用して、仮想マシンの IP アドレスを表示できます。

注記

セカンダリーネットワークインターフェイスの IP アドレスを表示するには、仮想マシンに QEMU ゲストエージェントをインストールする必要があります。Pod ネットワークインターフェイスには QEMU ゲストエージェントは必要ありません。

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. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Details** タブをクリックして IP アドレスを表示します。

8.12.2.2. コマンドラインを使用した仮想マシンの IP アドレスの表示

コマンドラインを使用して、仮想マシンの IP アドレスを表示できます。

注記

セカンダリーネットワークインターフェイスの IP アドレスを表示するには、仮想マシンに QEMU ゲストエージェントをインストールする必要があります。Pod ネットワークインターフェイスには QEMU ゲストエージェントは必要ありません。

手順

- 次のコマンドを実行して、仮想マシンインスタンスの設定を取得します。

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

出力例

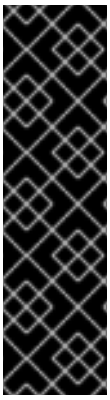
```
# ...
Interfaces:
  Interface Name:  eth0
  Ip Address:      10.244.0.37/24
  Ip Addresses:
    10.244.0.37/24
    fe80::858:aff:fef4:25/64
  Mac:             0a:58:0a:f4:00:25
  Name:            default
  Interface Name:  v2
  Ip Address:      1.1.1.7/24
  Ip Addresses:
    1.1.1.7/24
    fe80::f4d9:70ff:fe13:9089/64
  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
```

8.12.3. 関連情報

- QEMU ゲストエージェントのインストール

8.13. 外部 FQDN を使用した仮想マシンへのアクセス

完全修飾ドメイン名 (FQDN) を使用して、クラスターの外部からセカンダリーネットワークインターフェイスに接続されている仮想マシン (VM) にアクセスできます。



重要

FQDN を使用してクラスター外から VM へのアクセスは、テクノロジープレビューのみの機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

8.13.1. セカンダリーネットワーク用の DNS サーバーの設定

Cluster Network Addons Operator (CNAO) は、**HyperConverged** カスタムリソース (CR) で **deployKubeSecondaryDNS** 機能ゲートを有効にすると、ドメインネームサーバー (DNS) サーバーと監視コンポーネントをデプロイします。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスターのロードバランサーを設定しました。
- **cluster-admin** パーミッションを使用してクラスターにログインしました。

手順

1. 次の例に従って **oc expose** コマンドを実行して、クラスターの外部に DNS サーバーを公開するロードバランサーサービスを作成します。

```
$ oc expose -n openshift-cnv deployment/secondary-dns --name=dns-lb \
  --type=LoadBalancer --port=53 --target-port=5353 --protocol='UDP'
```

2. 次のコマンドを実行して、外部 IP アドレスを取得します。

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

出力例

```
NAME      TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)         AGE
dns-lb    LoadBalancer   172.30.27.5    10.46.41.94     53:31829/TCP    5s
```

3. 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
```

4. 次の例に従って、DNS サーバーと監視コンポーネントを有効にします。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  featureGates:
    deployKubeSecondaryDNS: true
  kubeSecondaryDNSNameServerIP: "10.46.41.94" (1)
# ...
```

(1) ロードバランサーサービスによって公開される外部 IP アドレスを指定します。

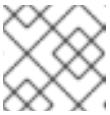
5. ファイルを保存して、エディターを終了します。
6. 次のコマンドを実行して、クラスターの FQDN を取得します。

```
$ oc get dnses.config.openshift.io cluster -o jsonpath='{.spec.baseDomain}'
```

出力例

```
openshift.example.com
```

7. 次のいずれかの方法を使用して、DNS サーバーを指定します。
   - ローカルマシンの **resolv.conf** ファイルに **kubeSecondaryDNSNameServerIP** 値を追加します。

     注記

     resolv.conf ファイルを編集すると、既存の DNS 設定が上書きされます。

   - **kubeSecondaryDNSNameServerIP** 値とクラスター FQDN をエンタープライズ DNS サーバーレコードに追加します。以下に例を示します。

     ```
     vm.<FQDN>. IN NS ns.vm.<FQDN>.
     ```

     ```
     ns.vm.<FQDN>. IN A 10.46.41.94
     ```

8.13.2. クラスター FQDN を使用したセカンダリーネットワーク上の仮想マシンへの接続

クラスターの完全修飾ドメイン名 (FQDN) を使用して、セカンダリーネットワークインターフェイスに接続された実行中の仮想マシンにアクセスできます。

前提条件

- QEMU ゲストエージェントを仮想マシンにインストールしました。
- 仮想マシンの IP アドレスはパブリックです。
- セカンダリーネットワーク用の DNS サーバーを設定しました。

- クラスターの完全修飾ドメイン名 (FQDN) を取得しました。

手順

1. 次のコマンドを実行して、仮想マシン設定からネットワークインターフェイス名を取得します。

```
$ oc get vm -n <namespace> <vm_name> -o yaml
```

出力例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  running: true
  template:
    spec:
      domain:
        devices:
          interfaces:
            - bridge: {}
              name: example-nic
# ...
      networks:
      - multus:
          networkName: bridge-conf
        name: example-nic (1)
```

(1) ネットワークインターフェイスの名前を書き留めます。

2. **ssh** コマンドを使用して仮想マシンに接続します。

```
$ ssh <user_name>@<interface_name>.<vm_name>.<namespace>.vm.<cluster_fqdn>
```

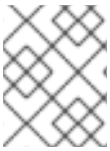
8.13.3. 関連情報

- ロードバランサーを使用した Ingress クラスターの設定
- MetalLB を使用した負荷分散
- 仮想マシンの IP アドレスの設定

8.14. ネットワークインターフェイスの MAC アドレスプールの管理

KubeMacPool コンポーネントは、共有 MAC アドレスプールから仮想マシンネットワークインターフェイスの MAC アドレスを割り当てます。これにより、各ネットワークインターフェイスに一意の MAC アドレスが確実に割り当てられます。

その仮想マシンから作成された仮想マシンインスタンスは、再起動後も割り当てられた 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。

8.14.1. コマンドラインを使用した KubeMacPool の管理

コマンドラインを使用して、KubeMacPool を無効にしたり、再度有効にしたりできます。

KubeMacPool はデフォルトで有効になっています。

手順

- 2 つの namespace で KubeMacPool を無効にするには、次のコマンドを実行します。

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io=ignore
```

- 2 つの namespace で KubeMacPool を再度有効にするには、次のコマンドを実行します。

```
$ oc label namespace <namespace1> <namespace2>
mutatevirtualmachines.kubemacpool.io-
```

第9章 ストレージ

9.1. ストレージ設定の概要

デフォルトのストレージクラス、ストレージプロファイル、Containerized Data Importer (CDI)、データボリューム、および自動ブートソース更新を設定できます。

9.1.1. ストレージ

次のストレージ設定タスクは必須です。

デフォルトのストレージクラスを設定する

クラスターの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。そうしないと、クラスターは自動ブートソース更新を受信できません。

ストレージプロファイルを設定する

ストレージプロバイダーが CDI によって認識されない場合は、ストレージプロファイルを設定する必要があります。ストレージプロファイルは、関連付けられたストレージクラスに基づいて推奨されるストレージ設定を提供します。

次のストレージ設定タスクはオプションです。

ファイルシステムのオーバーヘッドのために追加の PVC スペースを予約する

デフォルトでは、ファイルシステム PVC の 5.5% がオーバーヘッド用に予約されており、その分仮想マシンディスクに使用できるスペースが減少します。別のオーバーヘッド値を設定できます。

ホストパスプロビジョナーを使用してローカルストレージを設定する

ホストパスプロビジョナー (HPP) を使用して、仮想マシンのローカルストレージを設定できます。OpenShift Virtualization Operator をインストールすると、HPP Operator が自動的にインストールされます。

namespace 間でデータボリュームのクローンを作成するためのユーザー権限を設定する

RBAC ロールを設定して、ユーザーが namespace 間でデータボリュームのクローンを作成できるようにすることができます。

9.1.2. コンテナー化されたデータインポーター

次の Containerized Data Importer (CDI) 設定タスクを実行できます。

namespace のリソース要求制限をオーバーライドする

CPU およびメモリーリソースの制限を受ける namespace に仮想マシンディスクをインポート、アップロード、およびクローン作成するように CDI を設定できます。

CDI スクラッチスペースを設定する

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完了するためにスクラッチスペース (一時ストレージ) が必要です。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。

9.1.3. データボリューム

次のデータボリューム設定タスクを実行できます。

データボリュームの事前割り当てを有効にする

CDI は、データボリュームの作成時の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。

データボリュームのアノテーションを管理する

データボリューム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。

9.1.4. ブートソースの更新

次のブートソース更新設定タスクを実行できます。

ブートソースの自動更新を管理する

ブートソースにより、ユーザーは仮想マシン (VM) をよりアクセスしやすく効率的に作成できるようになります。ブートソースの自動更新が有効になっている場合、CDI はイメージをインポート、ポーリング、更新して、新しい仮想マシン用にクローンを作成できるようにします。デフォルトでは、CDI は Red Hat ブートソースを自動的に更新します。次に、カスタムブートソースの自動更新を有効にできます。

9.2. ストレージプロファイルの設定

ストレージプロファイルは、関連付けられたストレージクラスに基づいて推奨される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。

Containerized Data Importer (CDI) は、ストレージプロバイダーの機能を識別し、対話するように設定されている場合はストレージプロバイダーを認識します。

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。ストレージプロファイルをカスタマイズして、ストレージクラスの自動設定を行うこともできます。CDI がストレージプロバイダーを認識しない場合は、ユーザーがストレージプロファイルを設定する必要があります。

重要

Red Hat OpenShift Data Foundation で OpenShift Virtualization を使用する場合は、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。RBD ブロックモードボリュームは、Ceph FS または RBD ファイルシステムモード PVC よりも効率が高く、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。

9.2.1. ストレージプロファイルのカスタマイズ

プロビジョナーのストレージクラスの **StorageProfile** オブジェクトを編集してデフォルトパラメーターを指定できます。これらのデフォルトパラメーターは、**DataVolume** オブジェクトで設定されていない場合にのみ永続ボリューム要求 (PVC) に適用されます。

ストレージクラスのパラメーターは変更できません。変更する必要がある場合は、ストレージクラスを削除して再作成します。その後、ストレージプロファイルに適用していたカスタマイズを再適用する必要があります。

ストレージプロファイルの空の **status** セクションは、ストレージプロビジョナーが Containerized Data Interface (CDI) によって認識されないことを示します。CDI で認識されないストレージプロビジョナーがある場合、ストレージプロファイルをカスタマイズする必要があります。この場合、管理者はストレージプロファイルに適切な値を設定し、割り当てが正常に実行されるようにします。

警告

データボリュームを作成し、YAML 属性を省略し、これらの属性がストレージプロファイルで定義されていない場合は、要求されたストレージは割り当てられず、基礎となる永続ボリューム要求 (PVC) は作成されません。

前提条件

- 計画した設定がストレージクラスとそのプロバイダーでサポートされていることを確認してください。ストレージプロファイルに互換性のない設定を指定すると、ボリュームのプロビジョニングに失敗します。

手順

1. ストレージプロファイルを編集します。この例では、プロビジョナーは CDI によって認識されません。

```
$ oc edit 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**。

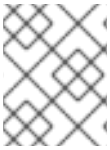
変更を保存した後、選択した値がストレージプロファイルの **status** 要素に表示されます。

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

ストレージプロファイルを使用してストレージクラスのデフォルトクローンメソッドを設定し、**クローンストラテジー** を作成できます。ストレージベンダーが特定のクローン作成方法のみをサポートする場合などに、クローンストラテジーを設定すると便利です。また、リソースの使用の制限やパフォーマンスの最大化を実現する手法を選択することもできます。

クローン作成ストラテジーは、ストレージプロファイルの **cloneStrategy** 属性を以下の値のいずれかに設定して指定できます。

- **snapshot** が設定されている場合、デフォルトでスナップショットが使用されます。CDI がストレージプロバイダーを認識し、プロバイダーが Container Storage Interface (CSI) スナップショットをサポートする場合、CDI はスナップショットメソッドを使用します。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。
- **copy** は、ソース Pod とターゲット Pod を使用して、ソースボリュームからターゲットボリュームにデータをコピーします。ホスト支援型でのクローン作成は、最も効率的な方法です。
- **csi-clone** は、CSI クローン API を使用して、中間ボリュームスナップショットを使用せずに、既存のボリュームのクローンを効率的に作成します。ストレージプロファイルが定義されていない場合にデフォルトで使用される **snapshot** または **copy** とは異なり、CSI ボリュームのクローンは、プロビジョナーのストレージクラスの **StorageProfile** オブジェクトに指定した場合にだけ使用されます。



注記

YAML **spec** セクションのデフォルトの **claimPropertySets** を変更せずに、CLI でクローンストラテジーを設定することもできます。

ストレージプロファイルの例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <provisioner_class>
# ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce 1
    volumeMode:
      Filesystem 2
  cloneStrategy: csi-clone 3
status:
  provisioner: <provisioner>
  storageClass: <provisioner_class>
```

1. アクセスモードを指定します。
2. ボリュームモードを指定します。
3. デフォルトのクローン戦略を指定します。

表9.1 ストレージプロバイダーとデフォルトの動作

ストレージプロバイダー	デフォルト動作
rook-ceph.rbd.csi.ceph.com	スナップショット
openshift-storage.rbd.csi.ceph.com	スナップショット
csi-vxflexos.dellemc.com	CSI Clone
csi-isilon.dellemc.com	CSI Clone
csi-powermax.dellemc.com	CSI Clone
csi-powerstore.dellemc.com	CSI Clone
hspc.csi.hitachi.com	CSI Clone
csi.hpe.com	CSI Clone
spectrumscale.csi.ibm.com	CSI Clone
rook-ceph.rbd.csi.ceph.com	CSI Clone
openshift-storage.rbd.csi.ceph.com	CSI Clone
cephfs.csi.ceph.com	CSI Clone
openshift-storage.cephfs.csi.ceph.com	CSI Clone

9.3. ブートソースの自動更新の管理

次のブートソースの自動更新を管理できます。

- すべての Red Hat ブートソース
- すべてのカスタムブートソース
- 個々の Red Hat またはカスタムブートソース

ブートソースにより、ユーザーは仮想マシン (VM) をよりアクセスしやすく効率的に作成できるようになります。ブートソースの自動更新が有効になっている場合、コンテナー化データインポーター (CDI) はイメージをインポート、ポーリング、更新して、新しい仮想マシン用にクローンを作成できるようにします。デフォルトでは、CDI は Red Hat ブートソースを自動的に更新します。

9.3.1. Red Hat ブートソースの更新の管理

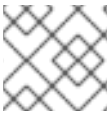
enableCommonBootImageImport 機能ゲートを無効にすることで、システム定義のすべてのブートソースの自動更新をオプトアウトできます。この機能ゲートを無効にすると、すべての **DataImportCron** オブジェクトが削除されます。この場合、オペレーティングシステムイメージを保存する以前にインポートされたブートソースオブジェクトは削除されませんが、管理者はこれらのオブジェクトを手動で削除できます。

enableCommonBootImageImport 機能ゲートが無効になると、**DataSource** オブジェクトがリセットされ、元のブートソースを指さなくなります。管理者は、**DataSource** オブジェクトの永続ボリューム要求 (PVC) またはボリュームスナップショットを新規作成し、それにオペレーティングシステムイメージを追加することで、ブートソースを手動で提供できます。

9.3.1.1. すべてのシステム定義のブートソースの自動更新の管理

ブートソースの自動インポートと更新を無効にすると、リソースの使用量が削減される可能性があります。切断された環境では、ブートソースの自動更新を無効にすると、**CDIDataImportCronOutdated** アラートがログをいっぱいにするのを防ぎます。

すべてのシステム定義のブートソースの自動更新を無効にするには、値を **false** に設定して、**enableCommonBootImageImport** 機能ゲートをオフにします。この値を **true** に設定すると、機能ゲートが再度有効になり、自動更新が再びオンになります。



注記

カスタムブートソースは、この設定の影響を受けません。

手順

- **HyperConverged** カスタムリソース (CR) を編集して、ブートソースの自動更新の機能ゲートを切り替えます。
  - ブートソースの自動更新を無効にするには、**HyperConverged** CR の **spec.featureGates.enableCommonBootImageImport** フィールドを **false** に設定します。以下に例を示します。

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

  - ブートソースの自動更新を再び有効にするには、**HyperConverged** CR の **spec.featureGates.enableCommonBootImageImport** フィールドを **true** に設定します。以下に例を示します。

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

9.3.2. カスタムブートソースの更新の管理

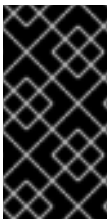
OpenShift Virtualization によって提供されていない **カスタム** ブートソースは、機能ゲートによって制御されません。**HyperConverged** カスタムリソース (CR) を編集して、それらを個別に管理する必要があります。

重要

ストレージクラスを設定する必要があります。そうしないと、クラスターはカスタムブートソースの自動更新を受信できません。詳細は、ストレージクラスの定義 を参照してください。

9.3.2.1. カスタムブートソース更新用のストレージクラスの設定

HyperConverged カスタムリソース (CR) を編集することで、デフォルトのストレージクラスをオーバーライドできます。



重要

ブートソースは、デフォルトのストレージクラスを使用してストレージから作成されます。クラスターにデフォルトのストレージクラスがない場合は、カスタムブートソースの自動更新を設定する前に、デフォルトのストレージクラスを定義する必要があります。

手順

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

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

2. **storageClassName** フィールドに値を入力して、新しいストレージクラスを定義します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: rhel8-image-cron
    spec:
      template:
        spec:
          storageClassName: <new_storage_class> 1
      schedule: "0 */12 * * *" 2
      managedDataSource: <data_source> 3
# ...
```

1. ストレージクラスを定義します。
2. 必須: cron 形式で指定したジョブのスケジュール。
3. 必須: 使用するデータソース。

> For the custom image to be detected as an available boot source, the value of the `spec.dataVolumeTemplates.spec.sourceRef.name` parameter in the VM template must match this value.

3. 現在のデフォルトのストレージクラスから **storageclass.kubernetes.io/is-default-class** アノテーションを削除します。

   a. 次のコマンドを実行して、現在のデフォルトのストレージクラスの名前を取得します。

   ```
   $ oc get storageclass
   ```

   出力例

   ```
   NAME                         PROVISIONER                    RECLAIMPOLICY
   VOLUMEBINDINGMODE    ALLOWVOLUMEEXPANSION  AGE
   csi-manila-ceph              manila.csi.openstack.org       Delete         Immediate
   false                11d
   hostpath-csi-basic (default)  kubevirt.io.hostpath-provisioner Delete
   WaitForFirstConsumer false                11d 1
   ```

   1 この例では、現在のデフォルトのストレージクラスの名前は **hostpath-csi-basic** です。

   b. 次のコマンドを実行して、現在のデフォルトのストレージクラスからアノテーションを削除します。

   ```
   $ oc patch storageclass <current_default_storage_class> -p '{"metadata": {"annotations":
   {"storageclass.kubernetes.io/is-default-class":"false"}}}' 1
   ```

   1 **<current_default_storage_class>** をデフォルトのストレージクラスの **storageClassName** 値に置き換えます。

4. 次のコマンドを実行して、新しいストレージクラスをデフォルトとして設定します。

   ```
   $ oc patch storageclass <new_storage_class> -p '{"metadata":{"annotations":
   {"storageclass.kubernetes.io/is-default-class":"true"}}}' 1
   ```

   1 **<new_storage_class>** を **HyperConverged** CR に追加した **storageClassName** 値に置き換えます。

9.3.2.2. カスタムブートソースの自動更新を有効にする

OpenShift Virtualization は、デフォルトでシステム定義のブートソースを自動的に更新しますが、カスタムブートソースは自動的に更新しません。**HyperConverged** カスタムリソース (CR) を編集して、自動更新を手動で有効にする必要があります。

前提条件

- クラスターにはデフォルトのストレージクラスがあります。

手順

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

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

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

カスタムリソースの例

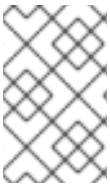
```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      name: centos7-image-cron
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true" 1
      labels:
        instancetype.kubevirt.io/default-preference: centos.7
        instancetype.kubevirt.io/default-instancetype: u1.medium
    spec:
      schedule: "0 */12 * * *" 2
      template:
        spec:
          source:
            registry: 3
              url: docker://quay.io/containerdisks/centos:7-2009
          storage:
            resources:
              requests:
                storage: 30Gi
      garbageCollect: Outdated
      managedDataSource: centos7 4
```

1. このアノテーションは、**volumeBindingMode** が **WaitForFirstConsumer** に設定されたストレージクラスに必要です。
2. cron 形式で指定されるジョブのスケジュール。
3. レジストリーソースからデータボリュームを作成するのに使用します。**node** docker キャッシュに基づくデフォルトの **node pullMethod** ではなく、デフォルトの **pod pullMethod** を使用します。**node** docker キャッシュはレジストリーイメージが**Container.Image** で利用可能な場合に便利ですが、CDI インポーターはこれにアクセスすることは許可されていません。
4. 利用可能なブートソースとして検出するカスタムイメージの場合、イメージの **managedDataSource** の名前が、仮想マシンテンプレート YAML ファイルの **spec.dataVolumeTemplates.spec.sourceRef.name** にあるテンプレートの **DataSource** の名前に一致する必要があります。

3. ファイルを保存します。

9.3.2.3. ボリュームスナップショットのブートソースを有効にする

オペレーティングシステムのベースイメージを保存するストレージクラスに関連付けられた **StorageProfile** のパラメーターを設定して、ボリュームスナップショットのブートソースを有効にします。**DataImportCron** は、元々 PVC ソースのみを維持するように設計されていましたが、特定のストレージタイプでは **VolumeSnapshot** ソースの方が PVC ソースよりも拡張性に優れています。



注記

ストレージプロファイルでは、単一のスナップショットからクローンを作成する場合により適切に拡張できることが証明されているボリュームスナップショットを使用してください。

前提条件

- オペレーティングシステムイメージを含むボリュームスナップショットにアクセスできる。
- ストレージはスナップショットをサポートしている。

手順

1. 次のコマンドを実行して、ブートソースのプロビジョニングに使用されるストレージクラスに対応するストレージプロファイルオブジェクトを開きます。

```
$ oc edit storageprofile <storage_class>
```

2. **StorageProfile** の **dataImportCronSourceFormat** 仕様を確認して、仮想マシンがデフォルトで PVC またはボリュームスナップショットを使用しているか確認します。
3. 必要に応じて、**dataImportCronSourceFormat** 仕様を **snapshot** に更新して、ストレージプロファイルを編集します。

ストレージプロファイルの例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
# ...
spec:
  dataImportCronSourceFormat: snapshot
```

検証

1. ブートソースのプロビジョニングに使用されるストレージクラスに対応するストレージプロファイルオブジェクトを開きます。

```
$ oc get storageprofile <storage_class>  -oyaml
```

2. **StorageProfile** の **dataImportCronSourceFormat** 仕様が 'snapshot' に設定されていること、および **DataImportCron** が指す **DataSource** オブジェクトがボリュームスナップショットを参照していることを確認します。

これで、これらのブートソースを使用して仮想マシンを作成できるようになりました。

9.3.3. 単一ブートソースの自動更新を無効にする

HyperConverged カスタムリソース (CR) を編集することで、カスタムブートソースかシステム定義ブートソースかに関係なく、個々のブートソースの自動更新を無効にできます。

手順

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

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

2. **spec.dataImportCronTemplates** フィールドを編集して、個々のブートソースの自動更新を無効にします。

   カスタムブートソース

   - **spec.dataImportCronTemplates** フィールドからブートソースを削除します。カスタムブートソースの自動更新はデフォルトで無効になっています。

   システム定義のブートソース

   a. ブートソースを **spec.dataImportCronTemplates** に追加します。

   注記

   システム定義のブートソースの自動更新はデフォルトで有効になっていますが、これらのブートソースは追加しない限り CR にリストされません。

   b. **dataimportcrontemplate.kubevirt.io/enable** アノテーションの値を **'false'** に設定します。
   以下に例を示します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  dataImportCronTemplates:
  - metadata:
      annotations:
        dataimportcrontemplate.kubevirt.io/enable: 'false'
      name: rhel8-image-cron
# ...
```

3. ファイルを保存します。

9.3.4. ブートソースのステータスの確認

HyperConverged カスタムリソース (CR) を表示することで、ブートソースがシステム定義であるかカスタムであるかを判断できます。

手順

1. 次のコマンドを実行して、**HyperConverged** CR の内容を表示します。

```
$ oc get hyperconverged kubevirt-hyperconverged -n openshift-cnv -o yaml
```

出力例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
# ...
status:
# ...
  dataImportCronTemplates:
  - metadata:
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true"
      name: centos-7-image-cron
    spec:
      garbageCollect: Outdated
      managedDataSource: centos7
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              url: docker://quay.io/containerdisks/centos:7-2009
          storage:
            resources:
              requests:
                storage: 30Gi
        status: {}
    status:
      commonTemplate: true 1
# ...
  - metadata:
      annotations:
        cdi.kubevirt.io/storage.bind.immediate.requested: "true"
      name: user-defined-dic
    spec:
      garbageCollect: Outdated
      managedDataSource: user-defined-centos-stream8
      schedule: 55 8/12 * * *
      template:
        metadata: {}
        spec:
          source:
            registry:
              pullMethod: node
              url: docker://quay.io/containerdisks/centos-stream:8
          storage:
```

```
          resources:
            requests:
              storage: 30Gi
        status: {}
      status: {} 2
# ...
```

1 システム定義のブートソースを示します。

2 カスタムブートソースを示します。

2. **status.dataImportCronTemplates.status** フィールドを確認して、ブートソースのステータスを確認します。

   - フィールドに **commonTemplate: true** が含まれている場合、それはシステム定義のブートソースです。
   - **status.dataImportCronTemplates.status** フィールドの値が **{}** の場合、それはカスタムブートソースです。

9.4. ファイルシステムオーバーヘッドの PVC 領域の確保

Filesystem ボリュームモードを使用する永続ボリューム要求 (PVC) に仮想マシンディスクを追加する場合は、仮想マシンディスクおよびファイルシステムのオーバーヘッド (メタデータなど) 用に十分なスペースが PVC 上にあることを確認する必要があります。

デフォルトでは、OpenShift Virtualization は PVC 領域の 5.5% をオーバーヘッド用に予約し、その分、仮想マシンディスクに使用できる領域を縮小します。

HCO オブジェクトを編集して、別のオーバーヘッド値を設定できます。値はグローバルに変更でき、特定のストレージクラスの値を指定できます。

9.4.1. デフォルトのファイルシステムオーバーヘッド値の上書き

HCO オブジェクトの **spec.filesystemOverhead** 属性を編集することで、OpenShift Virtualization がファイルシステムオーバーヘッド用に予約する永続ボリューム要求 (PVC) 領域の量を変更します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. 次のコマンドを実行して、**HCO** オブジェクトを編集用に開きます。

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

2. **spec.filesystemOverhead** フィールドを編集して、選択した値でデータを設定します。

```
# ...
spec:
  filesystemOverhead:
```

```
global: "<new_global_value>" (1)
storageClass:
  <storage_class_name>: "<new_value_for_this_storage_class>" (2)
```

1. まだ値が設定されていないストレージクラスに使用されるデフォルトのファイルシステムオーバーヘッドの割合。たとえば、**global: "0.07"** は、ファイルシステムのオーバーヘッド用に PVC の 7% を確保します。
2. 指定されたストレージクラスのファイルシステムのオーバーヘッドの割合 (パーセンテージ)。たとえば、**mystorageclass: "0.04"** は、**mystorageclass** ストレージクラスの PVC のデフォルトオーバーヘッド値を 4% に変更します。

3. エディターを保存して終了し、**HCO** オブジェクトを更新します。

検証

- 次のいずれかのコマンドを実行して、**CDIConfig** ステータスを表示し、変更を確認します。一般的に **CDIConfig** への変更を確認するには以下を実行します。

```
$ oc get cdiconfig -o yaml
```

CDIConfig に対する 特定の変更を表示するには以下を実行します。

```
$ oc get cdiconfig -o jsonpath='{.items..status.filesystemOverhead}'
```

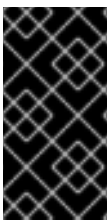
9.5. ホストパスプロビジョナーを使用したローカルストレージの設定

ホストパスプロビジョナー (HPP) を使用して、仮想マシンのローカルストレージを設定できます。

OpenShift Virtualization Operator のインストール時に、Hostpath Provisioner 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、基本ストレージプールを使用して HPP カスタムリソース (CR) を作成します。

9.5.1. 基本ストレージプールを使用したホストパスプロビジョナーの作成

storagePools スタンザを使用して HPP カスタムリソース (CR) を作成することにより、基本ストレージプールを使用してホストパスプロビジョナー (HPP) を設定します。ストレージプールは、CSI ドライバーが使用する名前とパスを指定します。



重要

オペレーティングシステムと同じパーティションにストレージプールを作成しないでください。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。

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。

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。

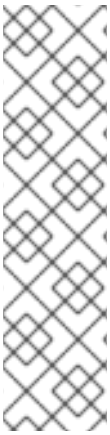
2. ファイルを保存して終了します。
3. 次のコマンドを実行して HPP を作成します。

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

9.5.1.1. ストレージクラスの作成について

ストレージクラスの作成時に、ストレージクラスに属する永続ボリューム (PV) の動的プロビジョニングに影響するパラメーターを設定します。**StorageClass** オブジェクトの作成後には、このオブジェクトのパラメーターを更新できません。

ホストパスプロビジョナー (HPP) を使用するには、**storagePools** スタンザで CSI ドライバーの関連付けられたストレージクラスを作成する必要があります。



注記

仮想マシンは、ローカル PV に基づくデータボリュームを使用します。ローカル PV は特定のノードにバインドされます。ディスクイメージは仮想マシンで使用するために準備されますが、ローカルストレージ PV がすでに固定されたノードに仮想マシンをスケジュールすることができない可能性があります。

この問題を解決するには、Kubernetes Pod スケジューラーを使用して、永続ボリューム要求 (PVC) を正しいノードの PV にバインドします。**volumeBindingMode** パラメーターが **WaitForFirstConsumer** に設定された **StorageClass** 値を使用することにより、PV のバインディングおよびプロビジョニングは、Pod が PVC を使用して作成されるまで遅延します。

9.5.1.2. storagePools スタンザを使用した CSI ドライバーのストレージクラスの作成

ホストパスプロビジョナー (HPP) を使用するには、コンテナーストレージインターフェイス (CSI) ドライバーに関連するストレージクラスを作成する必要があります。

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。

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。

手順

1. **storageclass_csi.yaml** ファイルを作成して、ストレージクラスを定義します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hostpath-csi
provisioner: kubevirt.io.hostpath-provisioner
reclaimPolicy: Delete (1)
volumeBindingMode: WaitForFirstConsumer (2)
parameters:
  storagePool: my-storage-pool (3)
```

(1) **reclaimPolicy** には、**Delete** および **Retain** の 2 つの値があります。値を指定しない場合、デフォルト値は **Delete** です。

(2) **volumeBindingMode** パラメーターは、動的プロビジョニングとボリュームのバインディングが実行されるタイミングを決定します。**WaitForFirstConsumer** を指定して、永続ボリューム要求 (PVC) を使用する Pod が作成されるまで PV のバインディングおよびプロビジョニングを遅延させます。これにより、PV が Pod のスケジュール要件を満たすようになります。

(3) HPP CR で定義されているストレージプールの名前を指定します。

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
```

9.5.2. PVC テンプレートで作成されたストレージプールについて

単一の大きな永続ボリューム (PV) がある場合は、ホストパスプロビジョナー (HPP) カスタムリソース (CR) で PVC テンプレートを定義することにより、ストレージプールを作成できます。

PVC テンプレートで作成されたストレージプールには、複数の HPP ボリュームを含めることができます。PV を小さなボリュームに分割すると、データ割り当ての柔軟性が向上します。

PVC テンプレートは、**PersistentVolumeClaim** オブジェクトの **spec** スタンザに基づいています。

PersistentVolumeClaim オブジェクトの例

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: iso-pvc
spec:
  volumeMode: Block (1)
  storageClassName: my-storage-class
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

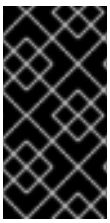
(1) この値は、ブロックボリュームモードの PV にのみ必要です。

HPP CR の **pvcTemplate** 仕様を使用してストレージプールを定義します。Operator は、HPP CSI ドライバーを含む各ノードの **pvcTemplate** 仕様から PVC を作成します。PVC テンプレートから作成される PVC は単一の大きな PV を消費するため、HPP は小規模な動的ボリュームを作成できます。

基本的なストレージプールを、PVC テンプレートから作成されたストレージプールと組み合わせることができます。

9.5.2.1. PVC テンプレートを使用したストレージプールの作成

HPP カスタムリソース (CR) で PVC テンプレートを指定することにより、複数のホストパスプロビジョナー (HPP) ボリューム用のストレージプールを作成できます。



重要

オペレーティングシステムと同じパーティションにストレージプールを作成しないでください。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。

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。

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
```

9.6. 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。

9.6.1. 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。

前提条件

- クラスター管理者権限がある。

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。

9.7. CPU およびメモリークォータをオーバーライドするための CDI の設定

Containerized Data Importer (CDI) を設定して、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。

9.7.1. 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。

AutoResourceLimits フィーチャーゲートを有効にすると、OpenShift Virtualization で自動的に CPU とメモリーの制限が管理されます。namespace に CPU とメモリーの両方のクォータがある場合、メモリー制限は基本割り当ての 2 倍に設定され、CPU 制限は vCPU ごとに 1 つになります。

9.7.2. 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 hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

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。

9.7.3. 関連情報

- プロジェクトごとのリソースクォータ

9.8. CDI のスクラッチ領域の用意

9.8.1. 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。

9.8.2. スクラッチ領域を必要とする CDI 操作

タイプ	理由

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

9.8.3. 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 hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

2. **spec.scratchSpaceStorageClass** フィールドを CR に追加し、値をクラスターに存在するストレージクラスの名前に設定します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
```

```
  name: kubevirt-hyperconverged
spec:
  scratchSpaceStorageClass: "<storage_class>" (1)
```

(1) ストレージクラスを指定しない場合、CDI は設定されている永続ボリューム要求のストレージクラスを使用します。

3. デフォルトのエディターを保存して終了し、**HyperConverged** CR を更新します。

9.8.4. 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

9.8.5. 関連情報

- 動的プロビジョニング

9.9. 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。

9.9.1. 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。

9.9.2. 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
    registry:
      url: <image_url> 2
  storage:
    resources:
      requests:
        storage: 1Gi
  preallocation: true
# ...
```

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ています。ただし、クローン作成操作では事前割り当ては無視されます。
2. レジストリー内のデータソースの URL を指定します。

9.10. 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。

9.10.1. 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: datavolume-example
  annotations:
    v1.multus-cni.io/default-network: bridge-network (1)
# ...
```

(1) Multus ネットワークアノテーション

第10章 ライブマイグレーション

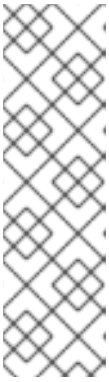
10.1. ライブマイグレーションについて

ライブマイグレーションは、仮想ワークロードに支障を与えることなく、実行中の仮想マシンをクラスター内の別のノードに移行するプロセスです。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。

10.1.1. ライブマイグレーションの要件

ライブマイグレーションには次の要件があります。

- クラスターには、**ReadWriteMany** (RWX) アクセスモードの共有ストレージが必要です。
- クラスターには十分な RAM とネットワーク帯域幅が必要です。



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。

- 仮想マシンがホストモデル CPU を使用する場合、ノードはその CPU をサポートする必要があります。
- ライブマイグレーション用に 専用の Multus ネットワークを設定すること を強く推奨します。専用ネットワークは、移行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。

10.1.2. 一般的なライブマイグレーションタスク

次のライブマイグレーションタスクを実行できます。

- ライブマイグレーションの設定
- ライブマイグレーションの開始とキャンセル
- OpenShift Virtualization Web コンソールの **Migration** タブで、すべてのライブマイグレーションの進行状況を監視する。
- Web コンソールの **Metrics** タブで仮想マシン移行メトリクスを表示する。

10.1.3. 関連情報

- ライブマイグレーション用の Prometheus クエリー
- 仮想マシン移行のチューニング

- VM 実行戦略
- 仮想マシンとクラスターのエビクション戦略

10.2. ライブマイグレーションの設定

ライブマイグレーション設定を行い、移行プロセスがクラスターに負荷を与えないようにすることができます。

ライブマイグレーションポリシーを設定して、さまざまな移行設定を仮想マシンのグループに適用できます。

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 hyperconverged kubevirt-hyperconverged -n openshift-cnv
```

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:
    bandwidthPerMigration: 64Mi (1)
    completionTimeoutPerGiB: 800 (2)
    parallelMigrationsPerCluster: 5 (3)
    parallelOutboundMigrationsPerNode: 2 (4)
    progressTimeout: 150 (5)
```

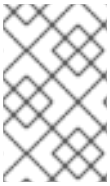
(1) 各マイグレーションの帯域幅制限。値は 1 秒あたりのバイト数です。たとえば、値 **2048Mi** は 2048 MiB/s を意味します。デフォルト: **0** (無制限)。

(2) 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、6 GiB メモリーを搭載した仮想マシンは、4800 秒以内に移行が完了しないとタイムアウトになります。**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。

(3) クラスターで並行して実行される移行の数。デフォルトは **5** です。

(4) ノードごとのアウトバウンドの移行の最大数。デフォルトは **2** です。

(5) 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。デフォルトは **150** です。



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. ライブマイグレーションポリシー

ライブマイグレーションポリシーを作成して、仮想マシンまたはプロジェクトラベルによって定義された仮想マシンのグループにさまざまな移行設定を適用できます。

ヒント

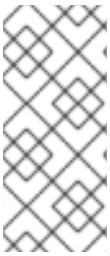
ライブマイグレーションポリシーは、OpenShift Virtualization Web コンソールを使用して作成できます。

10.2.2.1. コマンドラインを使用したライブマイグレーションポリシーの作成

コマンドラインを使用してライブマイグレーションポリシーを作成できます。KubeVirt は、任意のラベルの組み合わせを使用して、選択した仮想マシン (VM) にライブマイグレーションポリシーを適用します。

- **size**、**os**、**gpu** などの仮想マシンラベル
- **priority**、**bandwidth**、または **hpc-workload** などのプロジェクトラベル

ポリシーを特定の仮想マシングループに適用するには、仮想マシングループのすべてのラベルがポリシーのラベルと一致する必要があります。



注記

複数のライブマイグレーションポリシーが仮想マシンに適用される場合は、一致するラベルの数が最も多いポリシーが優先されます。

複数のポリシーがこの基準を満たす場合、ポリシーは一致するラベルキーのアルファベット順に並べ替えられ、その順序の最初のポリシーが優先されます。

手順

1. ライブマイグレーションポリシーを適用する仮想マシンオブジェクトを編集し、対応する仮想マシンラベルを追加します。
   a. リソースの YAML 設定を開きます。

   ```
   $ oc edit vm <vm_name>
   ```

   b. 設定の **.spec.template.metadata.labels** セクションで必要なラベル値を調整します。たとえば、移行ポリシーの目的で仮想マシンを **production** 仮想マシンとしてマークするには、**kubevirt.io/environment: production** 行を追加します。

   ```
   apiVersion: migrations.kubevirt.io/v1alpha1
   kind: VirtualMachine
   metadata:
     name: <vm_name>
     namespace: default
   ```

```
    labels:
      app: my-app
      environment: production
  spec:
    template:
      metadata:
        labels:
          kubevirt.io/domain: <vm_name>
          kubevirt.io/size: large
          kubevirt.io/environment: production
  # ...
```

c. 設定を保存して終了します。

2. 対応するラベルを使用して **MigrationPolicy** オブジェクトを設定します。次の例では、**production** というラベルが付けられたすべての仮想マシンに適用されるポリシーを設定します。

```
apiVersion: migrations.kubevirt.io/v1alpha1
kind: MigrationPolicy
metadata:
  name: <migration_policy>
spec:
  selectors:
    namespaceSelector: 1
      hpc-workloads: "True"
      xyz-workloads-type: ""
    virtualMachineInstanceSelector: 2
      kubevirt.io/environment: "production"
```

1 プロジェクトラベルを指定します。

2 仮想マシンラベルを指定します。

3. 次のコマンドを実行して、移行ポリシーを作成します。

```
$ oc create migrationpolicy -f <migration_policy>.yaml
```

10.2.3. 関連情報

- ライブマイグレーション用の専用 Multus ネットワークの設定

10.3. ライブマイグレーションの開始とキャンセル

OpenShift Container Platform Web コンソール または コマンドライン を使用して、仮想マシンの別のノードへのライブマイグレーションを開始できます。

ライブマイグレーションは、Web コンソール または コマンドライン を使用してキャンセルできます。仮想マシンは元のノードに残ります。

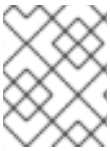
ヒント

virtctl migrate <vm_name> コマンドおよび **virtctl migrate-cancel <vm_name>** コマンドを使用して、ライブマイグレーションを開始およびキャンセルすることもできます。

10.3.1. ライブマイグレーションの開始

10.3.1.1. Web コンソールを使用したライブマイグレーションの開始

OpenShift Container Platform Web コンソールを使用して、実行中の仮想マシンをクラスター内の別のノードにライブマイグレーションできます。



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、ライブマイグレーションを開始できるのはクラスター管理者のみです。

前提条件

- 仮想マシンは移行可能である必要があります。
- 仮想マシンがホストモデル CPU で設定されている場合、クラスターにはその CPU モデルをサポートする利用可能なノードが必要です。

手順

1. Web コンソールで **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンの横にあるオプションメニュー  から **移行** を選択します。
3. **Migrate** をクリックします。

10.3.1.2. コマンドラインを使用してライブマイグレーションを開始する

コマンドラインを使用して仮想マシンの **VirtualMachineInstanceMigration** オブジェクトを作成することで、実行中の仮想マシンのライブマイグレーションを開始できます。

手順

1. 移行する仮想マシンの **VirtualMachineInstanceMigration** マニフェストを作成します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: <migration_name>
spec:
  vmiName: <vm_name>
```

2. 以下のコマンドを実行してオブジェクトを作成します。

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

VirtualMachineInstanceMigration オブジェクトは、仮想マシン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。

検証

- 次のコマンドを実行して、仮想マシンのステータスを取得します。

```
$ oc describe vmi <vm_name> -n <namespace>
```

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
```

10.3.2. ライブマイグレーションのキャンセル

10.3.2.1. Web コンソールを使用したライブマイグレーションのキャンセル

OpenShift Container Platform Web コンソールを使用して、仮想マシン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. Web コンソールで **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンの横にあるオプションメニュー  で **Cancel Migration** を選択します。

10.3.2.2. コマンドラインを使用したライブマイグレーションのキャンセル

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
```

第11章 ノード

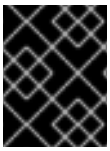
11.1. ノードのメンテナンス

oc adm ユーティリティーまたは **NodeMaintenance** カスタムリソース (CR) を使用してノードをメンテナンスモードにすることができます。

注記

node-maintenance-operator (NMO) は OpenShift Virtualization に同梱されなくなりました。これは、OpenShift Container Platform Web コンソールの **OperatorHub** から、または OpenShift CLI (**oc**) を使用して、スタンドアロン Operator としてデプロイされます。

ノードの修復、フェンシング、メンテナンスの詳細は、Red Hat OpenShift のワークロードの可用性 を参照してください。



重要

仮想マシンでは、共有 **ReadWriteMany** (RWX) アクセスモードを持つ永続ボリューム要求 (PVC) のライブマイグレーションが必要です。

Node Maintenance Operator は、新規または削除された **NodeMaintenance** CR をモニタリングします。新規の **NodeMaintenance** CR が検出されると、新規ワークロードはスケジュールされず、ノードは残りのクラスターから遮断されます。エビクトできるすべての Pod はノードからエビクトされます。**NodeMaintenance** CR が削除されると、CR で参照されるノードは新規ワークロードで利用可能になります。

注記

ノードのメンテナンスタスクに **NodeMaintenance** CR を使用すると、標準の OpenShift Container Platform カスタムリソース処理を使用して **oc adm cordon** および **oc adm drain** コマンドの場合と同じ結果が得られます。

11.1.1. エビクションストラテジー

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 がドレインされます。

仮想マシン (VM) またはクラスターのエビクションストラテジーを設定できます。

仮想マシンエビクションストラテジー

仮想マシンの **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 VMI は、別のノードにライブマイグレーションされます。

OpenShift Virtualization Web コンソールまたは コマンドライン を使用して、仮想マシン (VM) のエビクションストラテジーを設定できます。

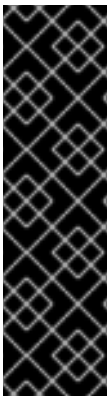
重要

デフォルトのエビクションストラテジーは **LiveMigrate** です。移行不可能な仮想マシンが **LiveMigrate** エビクションストラテジーを使用していると、ノードのドレインが妨げられたり、インフラストラクチャーのアップグレードがブロックされたりする可能性があります。これは、その仮想マシンがノードからエビクトされないためです。この状況では、仮想マシンを手動でシャットダウンしない限り、移行は **Pending** または **Scheduling** 中の状態のままになります。

移行不可能な仮想マシンのエビクションストラテジーを、アップグレードをブロックしない **LiveMigrateIfPossible** に設定する必要があります。移行すべきでない仮想マシンの場合は、**None** に設定する必要があります。

クラスターエビクションストラテジー

ワークロードの継続性またはインフラストラクチャーのアップグレードを優先するために、クラスターのエビクションストラテジーを設定できます。



重要

クラスターエビクションストラテジーの設定は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

表11.1 クラスターエビクションストラテジー

エビクションストラテジー	説明	ワークフローを中断する	アップグレードをブロックする
LiveMigrate[1]	アップグレードよりもワークロードの継続性を優先します。	いいえ	はい[2]
LiveMigrateIfPossible	ワークロードの継続性よりもアップグレードを優先して、環境が確実に更新されるようにします。	はい	いいえ
None[3]	エビクション戦略を使用せずに仮想マシンをシャットダウンします。	はい	いいえ

1. マルチノードクラスターのデフォルトのエビクション戦略。
2. 仮想マシンがアップグレードをブロックする場合は、仮想マシンを手動でシャットダウンする必要があります。
3. シングルノード OpenShift のデフォルトのエビクション戦略。

11.1.1.1. コマンドラインを使用した仮想マシンエビクション戦略の設定

コマンドラインを使用して、仮想マシンのエビクション戦略を設定できます。

重要

デフォルトのエビクションストラテジーは **LiveMigrate** です。移行不可能な仮想マシンが **LiveMigrate** エビクションストラテジーを使用していると、ノードのドレインが妨げられたり、インフラストラクチャーのアップグレードがブロックされたりする可能性があります。これは、その仮想マシンがノードからエビクトされないためです。この状況では、仮想マシンを手動でシャットダウンしない限り、移行は **Pending** または **Scheduling** 中の状態のままになります。

移行不可能な仮想マシンのエビクションストラテジーを、アップグレードをブロックしない **LiveMigrateIfPossible** に設定する必要があります。移行すべきでない仮想マシンの場合は、**None** に設定する必要があります。

手順

1. 次のコマンドを実行して、**VirtualMachine** リソースを編集します。

```
$ oc edit vm <vm_name> -n <namespace>
```

エビクション戦略の例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: <vm_name>
spec:
  template:
    spec:
      evictionStrategy: LiveMigrateIfPossible (1)
# ...
```

(1) エビクション戦略を指定します。デフォルト値は **LiveMigrate** です。

2. 仮想マシンを再起動して変更を適用します。

```
$ virtctl restart <vm_name> -n <namespace>
```

11.1.1.2. コマンドラインを使用したクラスターエビクション戦略の設定

コマンドラインを使用して、クラスターのエビクション戦略を設定できます。

重要

クラスターエビクションストラテジーの設定は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

手順

1. 次のコマンドを実行して、**hyperconverged** リソースを編集します。

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

2. 次の例に示すように、クラスターのエビクション戦略を設定します。

クラスターエビクション戦略の例

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  evictionStrategy: LiveMigrate
# ...
```

11.1.2. 戦略を実行する

spec.running: true で設定された仮想マシンはすぐに再起動されます。**spec.runStrategy** キーを使用すると、特定の条件下で仮想マシンがどのように動作するかを決定するための柔軟性が高まります。

重要

spec.runStrategy キーと **spec.running** キーは相互に排他的です。いずれか 1 つだけを使用できます。

両方のキーを含む仮想マシン設定は無効です。

11.1.2.1. 戦略を実行する

spec.runStrategy キーには 4 つの値があります。

Always

仮想マシンインスタンス (VMI) は、仮想マシンが別のノードに作成されるときに常に存在します。元の VMI が何らかの理由で停止した場合、新しい VMI が作成されます。これは、**running: true** と同じ動作です。

RerunOnFailure

以前のインスタンスに障害が発生した場合、VMI は別のノードで再作成されます。仮想マシンがシャットダウンされた場合など、仮想マシンが正常に停止した場合、インスタンスは再作成されません。

Manual

VMI 状態は、virtctl クライアントコマンドの **start**、**stop**、および **restart** を使用して手動で制御します。仮想マシンは自動的には再起動されません。

Halted

仮想マシンの作成時には VMI は存在しません。これは、**running: false** と同じ動作です。

virtctl start、**stop**、および **restart** コマンドのさまざまな組み合わせは、実行戦略に影響します。

次の表では、仮想マシンの状態間の遷移を説明します。最初の列は、仮想マシンの初期実行戦略を示します。残りの列には、virtctl コマンドと、そのコマンドの実行後の新しい実行戦略が表示されます。

表11.2 virtctl コマンドの前後でストラテジーを実行する

初期実行戦略	Start	Stop	Restart
Always	-	Halted	Always
RerunOnFailure	-	Halted	RerunOnFailure
Manual	Manual	Manual	Manual
Halted	Always	-	-

注記

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を使用してインストールされたクラスター内のノードがマシンの健全性チェックに失敗して使用できない場合は、**runStrategy: Always** または **runStrategy: RerunOnFailure** を持つ仮想マシンが新しいノードで再スケジュールされます。

11.1.2.2. コマンドラインを使用した仮想マシン実行戦略の設定

コマンドラインを使用して、仮想マシンの実行戦略を設定できます。

重要

spec.runStrategy キーと **spec.running** キーは相互に排他的です。両方のキーの値を含む仮想マシン設定は無効です。

手順

- 次のコマンドを実行して、**VirtualMachine** リソースを編集します。

```
$ oc edit vm <vm_name> -n <namespace>
```

実行戦略の例

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

```
kind: VirtualMachine
spec:
  runStrategy: Always
# ...
```

11.1.3. ベアメタルノードのメンテナンス

OpenShift Container Platform をベアメタルインフラストラクチャーにデプロイする場合、クラウドインフラストラクチャーにデプロイする場合と比較すると、追加で考慮する必要のある点があります。クラスターノードが一時的とみなされるクラウド環境とは異なり、ベアメタルノードを再プロビジョニングするには、メンテナンスタスクにより多くの時間と作業が必要になります。

致命的なカーネルエラーが発生したり、NIC カードのハードウェア障害が発生する場合などにベアメタルノードに障害が発生した場合には、障害のあるノードが修復または置き換えられている間に、障害が発生したノード上のワークロードをクラスターの別の場所で再起動する必要があります。ノードのメンテナンスモードにより、クラスター管理者はノードの電源を正常に停止し、ワークロードをクラスターの他の部分に移動させ、ワークロードが中断されないようにします。詳細な進捗とノードのステータスの詳細は、メンテナンス時に提供されます。

11.1.4. 関連情報

- ライブマイグレーションについて

11.2. 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。

11.2.1. 古い CPU モデルのノードラベリングについて

OpenShift Virtualization Operator は、古い CPU モデルの事前定義されたリストを使用して、ノードがスケジュールされた仮想マシンの有効な CPU モデルのみをサポートするようにします。

デフォルトでは、以下の CPU モデルはノード用に生成されたラベルのリストから削除されます。

例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。このリストから CPU モデルを **削除** できませんが、**HyperConverged** CR の **spec.obsoleteCPUs.cpuModels** フィールドを編集してリストに追加することはできます。

11.2.2. CPU 機能のノードのラベル付けについて

反復処理により、最小 CPU モデルのベース CPU 機能は、ノード用に生成されたラベルのリストから削除されます。

以下に例を示します。

- 環境には、**Penryn** と **Haswell** という 2 つのサポートされる CPU モデルが含まれる可能性があります。
- **Penryn** が **minCPU** の CPU モデルとして指定される場合、**Penryn** のベース CPU の各機能は **Haswell** によってサポートされる CPU 機能のリストと比較されます。

例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
```

11.2.3. 古い CPU モデルの設定

HyperConverged カスタムリソース (CR) を編集して、古い CPU モデルのリストを設定できます。

手順

- 古い CPU モデルを **obsoleteCPUs** 配列で指定して、**HyperConverged** カスタムリソースを編集します。以下に例を示します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: openshift-cnv
spec:
  obsoleteCPUs:
    cpuModels: 1
      - "<obsolete_cpu_1>"
      - "<obsolete_cpu_2>"
    minCPUModel: "<minimum_cpu_model>" 2
```

1. **cpuModels** 配列のサンプル値を、古くなった CPU モデルに置き換えます。指定した値はすべて、古くなった CPU モデルの事前定義リストに追加されます。事前定義されたリストは CR に表示されません。
2. この値を、基本的な CPU 機能に使用する最小 CPU モデルに置き換えます。値を指定しない場合は、デフォルトで **Penryn** が使用されます。

11.3. 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。

11.3.1. 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。

11.3.2. 関連情報

- 古い CPU モデルのノードラベルの管理

11.4. 障害が発生したノードを削除して仮想マシンのフェイルオーバーをトリガーする

ノードに障害が発生し、ノードヘルスチェック 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。

11.4.1. 前提条件

- 仮想マシンが実行されていたノードに **NotReady** 状態 が設定されている。
- 障害のあるノードで実行されていた仮想マシンでは、**runStrategy** が **Always** に設定されている。
- OpenShift CLI (**oc**) がインストールされている。

11.4.2. 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。

11.4.3. 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。

11.4.3.1. CLI を使用した仮想マシンインスタンスのリスト表示

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

手順

- 以下のコマンドを実行して、すべての VMI のリストを表示します。

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

第12章 モニタリング

12.1. モニタリングの概要

次のツールを使用して、クラスターと仮想マシン (VM) の正常性を監視できます。

OpenShift Virtualization 仮想マシンの健全性ステータスのモニタリング

OpenShift Container Platform Web コンソールの **Home → Overview** ページに移動して、Web コンソールで OpenShift Virtualization 環境の全体的な健全性を表示します。**Status** カードには、アラートと条件に基づいて OpenShift Virtualization の全体的な健全性が表示されます。

OpenShift Container Platform クラスター診断フレームワーク

OpenShift Container Platform クラスター診断フレームワークを使用してクラスターに対する自動テストを実行し、以下の状態を確認します。

- セカンダリーネットワークインターフェイスに接続された 2 つの仮想マシン間のネットワーク接続とレイテンシー
- パケット損失ゼロで Data Plane Development Kit (DPDK) ワークロードを実行する仮想マシン
- クラスターストレージは OpenShift Virtualization に最適に設定されています

仮想リソースの Prometheus クエリー

vCPU、ネットワーク、ストレージ、およびゲストメモリースワッピングの使用状況とライブマイグレーションの進行状況をクエリーします。

VM カスタムメトリクス

内部 VM メトリクスとプロセスを公開するように、**node-exporter** サービスを設定します。

VM ヘルスチェック

レディネス、ライブネス、ゲストエージェントの ping プローブ、および VM のウォッチドッグを設定します。

runbook

OpenShift Container Platform Web コンソールで OpenShift Virtualization の アラート を引き起こしている問題を診断して解決します。

12.2. OPENSHIFT VIRTUALIZATION クラスター検査フレームワーク

OpenShift Virtualization には、クラスターのメンテナンスとトラブルシューティングに使用できる次の定義済み検査が組み込まれています。

- レイテンシー検査。ネットワーク接続を検証し、セカンダリーネットワークインターフェイスに接続された 2 つの仮想マシン (VM) 間のレイテンシーを測定します。

重要

レイテンシー検査を実行する前に、まずクラスターノードに ブリッジインターフェイスを作成 して、仮想マシンのセカンダリーインターフェイスをノード上の任意のインターフェイスに接続する必要があります。ブリッジインターフェイスを作成しないと、仮想マシンが起動せず、ジョブが失敗します。

- ストレージ検査。クラスターストレージが OpenShift Virtualization に合わせて最適に設定されているかどうかを確認します。
- DPDK 検査。ノードが Data Plane Development Kit (DPDK) ワークロードを使用して仮想マシンをパケット損失なしで実行できることを確認します。

重要

OpenShift Virtualization クラスター検査フレームワークはテクノロジープレビューのみの機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

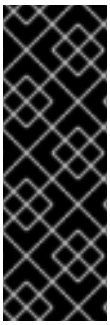
Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

12.2.1. OpenShift Virtualization クラスター検査フレームワークについて

チェックアップは、特定のクラスター機能が期待どおりに機能するかどうかを確認できる自動テストワークロードです。クラスター検査フレームワークは、ネイティブの Kubernetes リソースを使用して、チェックアップを設定および実行します。

事前定義されたチェックアップを使用することで、クラスター管理者はクラスターの保守性を向上させ、予期しない動作をトラブルシューティングし、エラーを最小限に抑え、時間を節約できます。また、チェックアップの結果を確認し、専門家と共有してさらに分析することもできます。ベンダーは、提供する機能やサービスのチェックアップを作成して公開し、顧客環境が正しく設定されていることを確認できます。

既存の namespace で事前定義されたチェックアップを実行するには、チェックアップ用のサービスアカウントの設定、サービスアカウント用の **Role** および **RoleBinding** オブジェクトの作成、チェックアップのパーミッションの有効化、入力 config map とチェックアップジョブの作成が含まれます。チェックアップは複数回実行できます。



重要

以下が常に必要になります。

- チェックアップイメージを適用する前に、信頼できるソースからのものであることを確認します。
- **Role** および **RoleBinding** オブジェクトを作成する前に、チェックアップパーミッションを確認してください。

12.2.2. Web コンソールを使用した検査の実行

Web コンソールを使用して初めて検査を実行する場合は、次の手順に従います。追加の検査については、いずれかの検査タブで **Run checkup** をクリックし、ドロップダウンメニューから適切な検査を選択します。

12.2.2.1. Web コンソールを使用したレイテンシー検査の実行

レイテンシー検査を実行して、ネットワーク接続を確認し、セカンダリーネットワークインターフェイスに接続された 2 つの仮想マシン間のレイテンシーを測定します。

前提条件

- namespace に **NetworkAttachmentDefinition** を追加する必要があります。

手順

1. Web コンソールで **Virtualization → Checkups** に移動します。
2. **Network latency** タブをクリックします。
3. **Install permissions** をクリックします。
4. **Run checkup** をクリックします。
5. **Name** フィールドに検査の名前を入力します。
6. ドロップダウンメニューから **NetworkAttachmentDefinition** を選択します。
7. オプション: **Sample duration (seconds)** フィールドでレイテンシーサンプルの期間を設定します。
8. オプション: **Set maximum desired latency (milliseconds)** を有効にして時間間隔を定義し、最大遅延時間間隔を定義します。
9. オプション: **Select nodes** を有効にし、**Source node** と **Target node** を指定して、特定のノードをターゲットにします。
10. **Run** クリックします。

レイテンシー検査のステータスは、**Latency checkup** タブの **Checkups** リストで確認できます。詳細は、検査名をクリックしてください。

12.2.2.2. Web コンソールを使用したストレージ検査の実行

ストレージ検査を実行して、仮想マシンのストレージが正しく動作していることを確認します。

手順

1. Web コンソールで **Virtualization → Checkups** に移動します。
2. **Storage** タブをクリックします。
3. **Install permissions** をクリックします。
4. **Run checkup** をクリックします。
5. **Name** フィールドに検査の名前を入力します。
6. **Timeout (minutes)** フィールドに検査のタイムアウト値を入力します。
7. **Run** クリックします。

ストレージチェックアップのステータスは、**Storage** タブの **Checkups** リストで確認できます。詳細は、検査名をクリックしてください。

12.2.3. コマンドラインを使用した検査の実行

コマンドラインを使用して初めて検査を実行する場合は、次の手順に従います。

12.2.3.1. コマンドラインを使用したレイテンシー検査の実行

定義済みのチェックアップを使用して、ネットワーク接続を確認し、セカンダリーネットワークインターフェイスに接続されている 2 つの仮想マシン (VM) 間の待機時間を測定します。レイテンシーチェックアップでは ping ユーティリティーを使用します。

次の手順でレイテンシーチェックアップを実行します。

1. サービスアカウント、ロール、ロールバインディングを作成して、レイテンシーチェックアップへのクラスターアクセス権を提供します。
2. config map を作成し、チェックアップを実行して結果を保存するための入力を行います。
3. チェックアップを実行するジョブを作成します。
4. config map で結果を確認します。
5. オプション: チェックアップを再実行するには、既存の config map とジョブを削除し、新しい config map とジョブを作成します。
6. 完了したら、レイテンシーチェックアップリソースを削除します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスターには少なくとも 2 つのワーカーノードがあります。
- namespace のネットワーク接続定義を設定しました。

手順

1. レイテンシーチェックアップ用の **ServiceAccount**、**Role**、**RoleBinding** マニフェストを作成します。

例12.1 ロールマニフェストファイルの例

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: vm-latency-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kubevirt-vm-latency-checker
rules:
- apiGroups: ["kubevirt.io"]
  resources: ["virtualmachineinstances"]
  verbs: ["get", "create", "delete"]
- apiGroups: ["subresources.kubevirt.io"]
  resources: ["virtualmachineinstances/console"]
  verbs: ["get"]
```

```
  - apiGroups: ["k8s.cni.cncf.io"]
    resources: ["network-attachment-definitions"]
    verbs: ["get"]
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: kubevirt-vm-latency-checker
  subjects:
  - kind: ServiceAccount
    name: vm-latency-checkup-sa
  roleRef:
    kind: Role
    name: kubevirt-vm-latency-checker
    apiGroup: rbac.authorization.k8s.io
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: Role
  metadata:
    name: kiagnose-configmap-access
  rules:
  - apiGroups: [ "" ]
    resources: [ "configmaps" ]
    verbs: ["get", "update"]
  ---
  apiVersion: rbac.authorization.k8s.io/v1
  kind: RoleBinding
  metadata:
    name: kiagnose-configmap-access
  subjects:
  - kind: ServiceAccount
    name: vm-latency-checkup-sa
  roleRef:
    kind: Role
    name: kiagnose-configmap-access
    apiGroup: rbac.authorization.k8s.io
```

2. **ServiceAccount**、**Role**、**RoleBinding** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <latency_sa_roles_rolebinding>.yaml 1
```

1 **<target_namespace>** は、チェックアップを実行する namespace です。これは、**NetworkAttachmentDefinition** オブジェクトが存在する既存の namespace である必要があります。

3. チェックアップの入力パラメーターを含む **ConfigMap** マニフェストを作成します。

入力 config map の例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-vm-latency-checkup-config
```

```
  labels:
    kiagnose/checkup-type: kubevirt-vm-latency
data:
  spec.timeout: 5m
  spec.param.networkAttachmentDefinitionNamespace: <target_namespace>
  spec.param.networkAttachmentDefinitionName: "blue-network" 1
  spec.param.maxDesiredLatencyMilliseconds: "10" 2
  spec.param.sampleDurationSeconds: "5" 3
  spec.param.sourceNode: "worker1" 4
  spec.param.targetNode: "worker2" 5
```

1. **NetworkAttachmentDefinition** オブジェクトの名前。
2. オプション: 仮想マシン間の必要な最大待機時間 (ミリ秒単位)。測定されたレイテンシーがこの値を超えると、チェックアップは失敗します。
3. オプション: レイテンシーチェックの継続時間 (秒単位)。
4. オプション: 指定すると、このノードからターゲットノードまでの待ち時間が測定されます。ソースノードが指定されている場合、**spec.param.targetNode** フィールドは空にできません。
5. オプション: 指定すると、ソースノードからこのノードまでの待ち時間が測定されます。

4. ターゲット namespace に config map マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <latency_config_map>.yaml
```

5. チェックアップを実行する **Job** マニフェストを作成します。

ジョブマニフェストの例

```
apiVersion: batch/v1
kind: Job
metadata:
  name: kubevirt-vm-latency-checkup
  labels:
    kiagnose/checkup-type: kubevirt-vm-latency
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccountName: vm-latency-checkup-sa
      restartPolicy: Never
      containers:
        - name: vm-latency-checkup
          image: registry.redhat.io/container-native-virtualization/vm-network-latency-checkup-
rhel9:v4.16.0
          securityContext:
            allowPrivilegeEscalation: false
            capabilities:
              drop: ["ALL"]
            runAsNonRoot: true
            seccompProfile:
```

```
      type: "RuntimeDefault"
  env:
    - name: CONFIGMAP_NAMESPACE
      value: <target_namespace>
    - name: CONFIGMAP_NAME
      value: kubevirt-vm-latency-checkup-config
    - name: POD_UID
      valueFrom:
        fieldRef:
          fieldPath: metadata.uid
```

6. **Job** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <latency_job>.yaml
```

7. ジョブが完了するまで待ちます。

```
$ oc wait job kubevirt-vm-latency-checkup -n <target_namespace> --for condition=complete --timeout 6m
```

8. 以下のコマンドを実行して、レイテンシーチェックアップの結果を確認します。測定された最大レイテンシーが **spec.param.maxDesiredLatencyMilliseconds** 属性の値よりも大きい場合、チェックアップは失敗し、エラーが返されます。

```
$ oc get configmap kubevirt-vm-latency-checkup-config -n <target_namespace> -o yaml
```

出力 config map の例 (成功)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-vm-latency-checkup-config
  namespace: <target_namespace>
  labels:
    kiagnose/checkup-type: kubevirt-vm-latency
data:
  spec.timeout: 5m
  spec.param.networkAttachmentDefinitionNamespace: <target_namespace>
  spec.param.networkAttachmentDefinitionName: "blue-network"
  spec.param.maxDesiredLatencyMilliseconds: "10"
  spec.param.sampleDurationSeconds: "5"
  spec.param.sourceNode: "worker1"
  spec.param.targetNode: "worker2"
  status.succeeded: "true"
  status.failureReason: ""
  status.completionTimestamp: "2022-01-01T09:00:00Z"
  status.startTimestamp: "2022-01-01T09:00:07Z"
  status.result.avgLatencyNanoSec: "177000"
  status.result.maxLatencyNanoSec: "244000" (1)
  status.result.measurementDurationSec: "5"
  status.result.minLatencyNanoSec: "135000"
  status.result.sourceNode: "worker1"
  status.result.targetNode: "worker2"
```

1 測定された最大レイテンシー (ナノ秒)。

9. オプション: チェックアップが失敗した場合に詳細なジョブログを表示するには、次のコマンドを使用します。

```
$ oc logs job.batch/kubevirt-vm-latency-checkup -n <target_namespace>
```

10. 以下のコマンドを実行して、以前に作成したジョブおよび config map を削除します。

```
$ oc delete job -n <target_namespace> kubevirt-vm-latency-checkup
```

```
$ oc delete config-map -n <target_namespace> kubevirt-vm-latency-checkup-config
```

11. オプション: 別のチェックアップを実行する予定がない場合は、ロールマニフェストを削除します。

```
$ oc delete -f <latency_sa_roles_rolebinding>.yaml
```

12.2.3.2. コマンドラインを使用したストレージ検査の実行

事前定義された検査を使用して、OpenShift Container Platform クラスターストレージが OpenShift Virtualization ワークロードを実行するために最適に設定されていることを確認します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスター管理者が、次の例のように、ストレージ検査のサービスアカウントと namespace に必要な **cluster-reader** 権限を作成した。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: kubevirt-storage-checkup-clustereader
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-reader
subjects:
- kind: ServiceAccount
  name: storage-checkup-sa
  namespace: <target_namespace> 1
```

1 検査の実行対象の namespace。

手順

1. ストレージ検査用の **ServiceAccount**、**Role**、および **RoleBinding** マニフェストファイルを作成します。

例12.2 サービスアカウント、ロール、ロールバインディングマニフェストファイルの例

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: storage-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: storage-checkup-role
rules:
  - apiGroups: [ "" ]
    resources: [ "configmaps" ]
    verbs: ["get", "update"]
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachines" ]
    verbs: [ "create", "delete" ]
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachineinstances" ]
    verbs: [ "get" ]
  - apiGroups: [ "subresources.kubevirt.io" ]
    resources: [ "virtualmachineinstances/addvolume",
"virtualmachineinstances/removevolume" ]
    verbs: [ "update" ]
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachineinstancemigrations" ]
    verbs: [ "create" ]
  - apiGroups: [ "cdi.kubevirt.io" ]
    resources: [ "datavolumes" ]
    verbs: [ "create", "delete" ]
  - apiGroups: [ "" ]
    resources: [ "persistentvolumeclaims" ]
    verbs: [ "delete" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: storage-checkup-role
subjects:
  - kind: ServiceAccount
    name: storage-checkup-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: storage-checkup-role
```

2. ターゲット namespace に **ServiceAccount**、**Role**、**RoleBinding** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <storage_sa_roles_rolebinding>.yaml
```

3. **ConfigMap** と **Job** マニフェストファイルを作成します。この config map に、検査ジョブの入力パラメーターを含めます。

入力 config map とジョブマニフェストの例

入力 config map とジョブマニフェストの例

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: storage-checkup-config
  namespace: $CHECKUP_NAMESPACE
data:
  spec.timeout: 10m
  spec.param.storageClass: ocs-storagecluster-ceph-rbd-virtualization
  spec.param.vmiTimeout: 3m
---
apiVersion: batch/v1
kind: Job
metadata:
  name: storage-checkup
  namespace: $CHECKUP_NAMESPACE
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccount: storage-checkup-sa
      restartPolicy: Never
      containers:
        - name: storage-checkup
          image: quay.io/kiagnose/kubevirt-storage-checkup:main
          imagePullPolicy: Always
          env:
            - name: CONFIGMAP_NAMESPACE
              value: $CHECKUP_NAMESPACE
            - name: CONFIGMAP_NAME
              value: storage-checkup-config
```

4. ターゲット namespace に **ConfigMap** と **Job** マニフェストファイルを適用し、検査を実行します。

```
$ oc apply -n <target_namespace> -f <storage_configmap_job>.yaml
```

5. ジョブが完了するまで待ちます。

```
$ oc wait job storage-checkup -n <target_namespace> --for condition=complete --timeout
10m
```

6. 次のコマンドを実行して、検査の結果を確認します。

```
$ oc get configmap storage-checkup-config -n <target_namespace> -o yaml
```

出力 config map の例 (成功)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: storage-checkup-config
```

```
  labels:
    kiagnose/checkup-type: kubevirt-storage
data:
  spec.timeout: 10m
  status.succeeded: "true" (1)
  status.failureReason: "" (2)
  status.startTimestamp: "2023-07-31T13:14:38Z" (3)
  status.completionTimestamp: "2023-07-31T13:19:41Z" (4)
  status.result.cnvVersion: 4.16.2 (5)
  status.result.defaultStorageClass: trident-nfs (6)
  status.result.goldenImagesNoDataSource: <data_import_cron_list> (7)
  status.result.goldenImagesNotUpToDate: <data_import_cron_list> (8)
  status.result.ocpVersion: 4.16.0 (9)
  status.result.pvcBound: "true" (10)
  status.result.storageProfileMissingVolumeSnapshotClass: <storage_class_list> (11)
  status.result.storageProfilesWithEmptyClaimPropertySets: <storage_profile_list> (12)
  status.result.storageProfilesWithSmartClone: <storage_profile_list> (13)
  status.result.storageProfilesWithSpecClaimPropertySets: <storage_profile_list> (14)
  status.result.storageProfilesWithRWX: |-
    ocs-storagecluster-ceph-rbd
    ocs-storagecluster-ceph-rbd-virtualization
    ocs-storagecluster-cephfs
    trident-iscsi
    trident-minio
    trident-nfs
    windows-vms
  status.result.vmBootFromGoldenImage: VMI "vmi-under-test-dhkb8" successfully booted
  status.result.vmHotplugVolume: |-
    VMI "vmi-under-test-dhkb8" hotplug volume ready
    VMI "vmi-under-test-dhkb8" hotplug volume removed
  status.result.vmLiveMigration: VMI "vmi-under-test-dhkb8" migration completed
  status.result.vmVolumeClone: 'DV cloneType: "csi-clone"'
  status.result.vmsWithNonVirtRbdStorageClass: <vm_list> (15)
  status.result.vmsWithUnsetEfsStorageClass: <vm_list> (16)
```

(1) 検査が成功したか (**true**)、失敗したか (**false**) を示します。

(2) 検査が失敗した場合の失敗の理由。

(3) 検査が開始した時刻 (RFC 3339 時刻形式)。

(4) 検査が完了した時刻 (RFC 3339 時刻形式)。

(5) OpenShift Virtualization のバージョン。

(6) デフォルトのストレージクラスがあるかどうかを指定します。

(7) データソースの準備ができていないゴールデンイメージのリスト。

(8) データインポート cron が最新ではないゴールデンイメージのリスト。

(9) OpenShift Container Platform のバージョン。

(10)

10 Mi の PVC が作成され、プロビジョナーによってバインドされているかどうかを指定します。

11 スナップショットベースのクローンを使用しているが、VolumeSnapshotClass が欠落しているストレージプロファイルのリスト。

12 不明なプロビジョナーを持つストレージプロファイルのリスト。

13 スマートクローン (CSI/スナップショット) をサポートするストレージプロファイルのリスト。

14 ストレージプロファイル仕様でオーバーライドされる claimPropertySets のリスト。

15 仮想化ストレージクラスが存在する場合に Ceph RBD ストレージクラスを使用する仮想マシンのリスト。

16 GID と UID がストレージクラスに設定されていない Elastic File Store (EFS) ストレージクラスを使用する仮想マシンのリスト。

7. 以下のコマンドを実行して、以前に作成したジョブおよび config map を削除します。

```
$ oc delete job -n <target_namespace> storage-checkup
```

```
$ oc delete config-map -n <target_namespace> storage-checkup-config
```

8. オプション: 別のチェックアップを実行する予定がない場合は、**ServiceAccount**、**Role**、**RoleBinding** マニフェストを削除します。

```
$ oc delete -f <storage_sa_roles_rolebinding>.yaml
```

12.2.3.3. コマンドラインを使用した DPDK 検査の実行

事前定義されたチェックアップを使用して、OpenShift Container Platform クラスターノードが Data Plane Development Kit (DPDK) ワークロードがある仮想マシン (VM) をパケット損失ゼロで実行できるか確認します。DPDK チェックアップは、トラフィックジェネレーターと、テスト DPDK アプリケーションを実行している仮想マシン間でトラフィックを実行します。

次の手順で DPDK チェックアップを実行します。

1. DPDK チェック用のサービスアカウント、ロール、およびロールバインディングを作成します。
2. config map を作成し、チェックアップを実行して結果を保存するための入力を行います。
3. チェックアップを実行するジョブを作成します。
4. config map で結果を確認します。
5. オプション: チェックアップを再実行するには、既存の config map とジョブを削除し、新しい config map とジョブを作成します。
6. 完了したら、DPDK チェックリソースを削除します。

前提条件

- OpenShift CLI (**oc**) がインストールされている。
- クラスターは DPDK アプリケーションを実行するように設定されています。
- プロジェクトは DPDK アプリケーションを実行するように設定されています。

手順

1. DPDK チェック用の **ServiceAccount**、**Role**、および **RoleBinding** マニフェストを作成します。

例12.3 サービスアカウント、ロール、ロールバインディングマニフェストファイルの例

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: dpdk-checkup-sa
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kiagnose-configmap-access
rules:
  - apiGroups: [ "" ]
    resources: [ "configmaps" ]
    verbs: [ "get", "update" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: kiagnose-configmap-access
subjects:
  - kind: ServiceAccount
    name: dpdk-checkup-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kiagnose-configmap-access
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: kubevirt-dpdk-checker
rules:
  - apiGroups: [ "kubevirt.io" ]
    resources: [ "virtualmachineinstances" ]
    verbs: [ "create", "get", "delete" ]
  - apiGroups: [ "subresources.kubevirt.io" ]
    resources: [ "virtualmachineinstances/console" ]
    verbs: [ "get" ]
  - apiGroups: [ "" ]
    resources: [ "configmaps" ]
    verbs: [ "create", "delete" ]
---
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: RoleBinding
metadata:
  name: kubevirt-dpdk-checker
subjects:
  - kind: ServiceAccount
    name: dpdk-checkup-sa
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: kubevirt-dpdk-checker
```

2. **ServiceAccount**、**Role**、**RoleBinding** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <dpdk_sa_roles_rolebinding>.yaml
```

3. チェックアップの入力パラメーターを含む **ConfigMap** マニフェストを作成します。

入力 config map の例

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dpdk-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-dpdk
data:
  spec.timeout: 10m
  spec.param.networkAttachmentDefinitionName: <network_name> (1)
  spec.param.trafficGenContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-traffic-gen:v0.4.0 (2)
  spec.param.vmUnderTestContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-vm:v0.4.0" (3)
```

(1) **NetworkAttachmentDefinition** オブジェクトの名前。

(2) トラフィックジェネレーターのコンテナーディスクイメージ。この例では、アップストリームの Project Quay Container Registry からイメージをプルしています。

(3) テスト対象の仮想マシンのコンテナーディスクイメージ。この例では、アップストリームの Project Quay Container Registry からイメージをプルしています。

4. ターゲット namespace に **ConfigMap** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <dpdk_config_map>.yaml
```

5. チェックアップを実行する **Job** マニフェストを作成します。

ジョブマニフェストの例

```
apiVersion: batch/v1
kind: Job
metadata:
```

```
  name: dpdk-checkup
  labels:
    kiagnose/checkup-type: kubevirt-dpdk
spec:
  backoffLimit: 0
  template:
    spec:
      serviceAccountName: dpdk-checkup-sa
      restartPolicy: Never
      containers:
        - name: dpdk-checkup
          image: registry.redhat.io/container-native-virtualization/kubevirt-dpdk-checkup-
rhel9:v4.16.0
          imagePullPolicy: Always
          securityContext:
            allowPrivilegeEscalation: false
            capabilities:
              drop: ["ALL"]
            runAsNonRoot: true
            seccompProfile:
              type: "RuntimeDefault"
          env:
            - name: CONFIGMAP_NAMESPACE
              value: <target-namespace>
            - name: CONFIGMAP_NAME
              value: dpdk-checkup-config
            - name: POD_UID
              valueFrom:
                fieldRef:
                  fieldPath: metadata.uid
```

6. **Job** マニフェストを適用します。

```
$ oc apply -n <target_namespace> -f <dpdk_job>.yaml
```

7. ジョブが完了するまで待ちます。

```
$ oc wait job dpdk-checkup -n <target_namespace> --for condition=complete --timeout 10m
```

8. 次のコマンドを実行して、検査の結果を確認します。

```
$ oc get configmap dpdk-checkup-config -n <target_namespace> -o yaml
```

出力 config map の例 (成功)

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: dpdk-checkup-config
  labels:
    kiagnose/checkup-type: kubevirt-dpdk
data:
  spec.timeout: 10m
  spec.param.NetworkAttachmentDefinitionName: "dpdk-network-1"
```

```
  spec.param.trafficGenContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-traffic-gen:v0.4.0"
  spec.param.vmUnderTestContainerDiskImage: "quay.io/kiagnose/kubevirt-dpdk-checkup-vm:v0.4.0"
  status.succeeded: "true" (1)
  status.failureReason: "" (2)
  status.startTimestamp: "2023-07-31T13:14:38Z" (3)
  status.completionTimestamp: "2023-07-31T13:19:41Z" (4)
  status.result.trafficGenSentPackets: "480000000" (5)
  status.result.trafficGenOutputErrorPackets: "0" (6)
  status.result.trafficGenInputErrorPackets: "0" (7)
  status.result.trafficGenActualNodeName: worker-dpdk1 (8)
  status.result.vmUnderTestActualNodeName: worker-dpdk2 (9)
  status.result.vmUnderTestReceivedPackets: "480000000" (10)
  status.result.vmUnderTestRxDroppedPackets: "0" (11)
  status.result.vmUnderTestTxDroppedPackets: "0" (12)
```

1. 検査が成功したか (**true**)、失敗したか (**false**) を示します。
2. 検査が失敗した場合の失敗の理由。
3. 検査が開始した時刻 (RFC 3339 時刻形式)。
4. 検査が完了した時刻 (RFC 3339 時刻形式)。
5. トラフィックジェネレーターから送信されたパケットの数。
6. トラフィックジェネレーターから送信されたエラーパケットの数。
7. トラフィックジェネレーターが受信したエラーパケットの数。
8. トラフィックジェネレーター仮想マシンがスケジュールされたノード。
9. テスト対象の仮想マシンがスケジュールされたノード。
10. テスト対象の仮想マシンで受信したパケットの数。
11. DPDK アプリケーションによって破棄された入力トラフィックパケット。
12. DPDK アプリケーションから破棄された出力トラフィックパケット。

9. 以下のコマンドを実行して、以前に作成したジョブおよび config map を削除します。

```
$ oc delete job -n <target_namespace> dpdk-checkup
```

```
$ oc delete config-map -n <target_namespace> dpdk-checkup-config
```

10. オプション: 別のチェックアップを実行する予定がない場合は、**ServiceAccount**、**Role**、**RoleBinding** マニフェストを削除します。

```
$ oc delete -f <dpdk_sa_roles_rolebinding>.yaml
```

12.2.3.3.1. DPDK チェックアップ config map パラメーター

次の表は、クラスター DPDK 準備状況チェックを実行するときに、入力 **ConfigMap** マニフェストの **data** スタンザに設定できる必須パラメーターとオプションパラメーターを示しています。

表12.1 DPDK チェックアップ config map の入力パラメーター

パラメーター	説明	必須かどうか
spec.timeout	チェックアップが失敗するまでの時間 (分単位)。	True
spec.param.networkAttachmentDefinitionName	接続されている SR-IOV NIC の **NetworkAttachmentDefinition** オブジェクトの名前。	True
spec.param.trafficGenContainerDiskImage	トラフィックジェネレーターのコンテナーディスクイメージ。	True
spec.param.trafficGenTargetNodeName	トラフィックジェネレーター仮想マシンがスケジュールされるノード。DPDK トラフィックを許可するようにノードを設定する必要があります。	False
spec.param.trafficGenPacketsPerSecond	1 秒あたりのパケット数 (キロ (k) または 100 万 (m) 単位)。デフォルト値は 8m です。	False
spec.param.vmUnderTestContainerDiskImage	テスト対象の仮想マシンのコンテナーディスクイメージ。	True
spec.param.vmUnderTestTargetNodeName	テスト対象の仮想マシンがスケジュールされるノード。DPDK トラフィックを許可するようにノードを設定する必要があります。	False
spec.param.testDuration	トラフィックジェネレーターが実行される期間 (分単位)。デフォルト値は 5 分です。	False
spec.param.portBandwidthGbps	SR-IOV NIC の最大帯域幅。デフォルト値は 10Gbps です。	False
spec.param.verbose	**true** に設定すると、検査ログの詳細度が上がります。デフォルト値は **false** です。	False

12.2.3.3.2. RHEL 仮想マシン用コンテナーディスクイメージのビルド

カスタムの Red Hat Enterprise Linux (RHEL) 8 OS イメージを **qcow2** 形式でビルドし、それを使用してコンテナーディスクイメージを作成できます。クラスターからアクセス可能なレジストリーにコンテ

ナーディスクイメージを保存し、DPDK チェック config map の **spec.param.vmContainerDiskImage** 属性でイメージの場所を指定できます。

コンテナーディスクイメージをビルドするには、Image Builder 仮想マシン (VM) を作成する必要があります。**Image Builder 仮想マシン** は、カスタム RHEL イメージのビルドに使用できる RHEL 8 仮想マシンです。

前提条件

- Image Builder 仮想マシンは RHEL 8.7 を実行でき、**/var** ディレクトリーに少なくとも 2 つの CPU コア、4 GiB RAM、20 GB の空き領域がある。
- Image Builder ツールとその CLI (**composer-cli**) が仮想マシンにインストールされている。
- **virt-customize** ツールがインストールされている。

```
# dnf install libguestfs-tools
```

- Podman CLI ツール (**podman**) がインストールされている。

手順

1. RHEL 8.7 イメージをビルドできることを確認します。

```
# composer-cli distros list
```

注記

非 root として **composer-cli** コマンドを実行するには、ユーザーを **weldr** または **root** グループに追加します。

```
# usermod -a -G weldr user
```

```
$ newgrp weldr
```

2. 次のコマンドを入力して、インストールするパッケージ、カーネルのカスタマイズ、起動時に無効化するサービスを含むイメージブループリントファイルを TOML 形式で作成します。

```
$ cat << EOF > dpdk-vm.toml
name = "dpdk_image"
description = "Image to use with the DPDK checkup"
version = "0.0.1"
distro = "rhel-87"

[[customizations.user]]
name = "root"
password = "redhat"

[[packages]]
name = "dpdk"

[[packages]]
name = "dpdk-tools"
```

```
[[packages]]
name = "driverctl"

[[packages]]
name = "tuned-profiles-cpu-partitioning"

[customizations.kernel]
append = "default_hugepagesz=1GB hugepagesz=1G hugepages=1"

[customizations.services]
disabled = ["NetworkManager-wait-online", "sshd"]
EOF
```

3. 次のコマンドを実行して、ブループリントファイルを Image Builder ツールにプッシュします。

```
# composer-cli blueprints push dpdk-vm.toml
```

4. ブループリント名と出力ファイル形式を指定して、システムイメージを生成します。作成プロセスを開始すると、イメージのユニバーサル一意識別子 (UUID) が表示されます。

```
# composer-cli compose start dpdk_image qcow2
```

5. 作成プロセスが完了するまで待ちます。作成ステータスが **FINISHED** になると次のステップに進めます。

```
# composer-cli compose status
```

6. 次のコマンドを入力し、UUID を指定して **qcow2** イメージファイルをダウンロードします。

```
# composer-cli compose image <UUID>
```

7. 次のコマンドを実行して、カスタマイズスクリプトを作成します。

```
$ cat <<EOF >customize-vm
#!/bin/bash

# Setup hugepages mount
mkdir -p /mnt/huge
echo "hugetlbfs /mnt/huge hugetlbfs defaults,pagesize=1GB 0 0" >> /etc/fstab

# Create vfio-noiommu.conf
echo "options vfio enable_unsafe_noiommu_mode=1" > /etc/modprobe.d/vfio-noiommu.conf

# Enable guest-exec,guest-exec-status on the qemu-guest-agent configuration
sed -i '/^BLACKLIST_RPC=/ { s/guest-exec-status//; s/guest-exec//g }' /etc/sysconfig/qemu-ga
sed -i '/^BLACKLIST_RPC=/ { s/,\+/,/g; s/^,\|,$//g }' /etc/sysconfig/qemu-ga
EOF
```

8. **virt-customize** ツールを使用して、Image Builder ツールによって生成されたイメージをカスタマイズします。

```
$ virt-customize -a <UUID>-disk.qcow2 --run=customize-vm --selinux-relabel
```

9. コンテナーディスクイメージのビルドに必要なすべてのコマンドを含む Dockerfile を作成するには、次のコマンドを入力します。

```
$ cat << EOF > Dockerfile
FROM scratch
COPY --chown=107:107 <UUID>-disk.qcow2 /disk/
EOF
```

ここでは、以下のようになります。

<UUID>-disk.qcow2
カスタムイメージの名前を **qcow2** 形式で指定します。

10. 次のコマンドを実行し、コンテナーをビルドしてタグを追加します。

```
$ podman build . -t dpdk-rhel:latest
```

11. 次のコマンドを実行して、クラスターからアクセスできるレジストリーにコンテナーディスクイメージをプッシュします。

```
$ podman push dpdk-rhel:latest
```

12. DPDK チェックアップ config map の **spec.param.vmUnderTestContainerDiskImage** 属性で、コンテナーディスクイメージへのリンクを指定します。

12.2.4. 関連情報

- 仮想マシンの複数ネットワークへの割り当て
- Intel NIC を使用した DPDK モードでの Virtual Function の使用
- SR-IOV と Node Tuning Operator を使用した DPDK ラインレートの実現
- Image Builder のインストール
- How to register and subscribe a RHEL system to the Red Hat Customer Portal using Red Hat Subscription Manager

12.3. 仮想リソースの PROMETHEUS クエリー

OpenShift Virtualization は、vCPU、ネットワーク、ストレージ、ゲストメモリースワッピングなどのクラスターインフラストラクチャーリソースの消費を監視するために使用できるメトリックを提供します。メトリクスを使用して、ライブマイグレーションのステータスを照会することもできます。

12.3.1. 前提条件

- vCPU メトリックを使用するには、**schedstats=enable** カーネル引数を **MachineConfig** オブジェクトに適用する必要があります。このカーネル引数を使用すると、デバッグとパフォーマンスチューニングに使用されるスケジューラーの統計が有効になり、スケジューラーに小規模な負荷を追加できます。詳細は、ノードへのカーネル引数の追加 を参照してください。
- ゲストメモリースワップクエリーがデータを返すには、仮想ゲストでメモリースワップを有効にする必要があります。

12.3.2. メトリクスのクエリー

OpenShift Container Platform モニタリングダッシュボードでは、Prometheus のクエリー言語 (PromQL) クエリーを実行し、プロットに可視化されるメトリクスを検査できます。この機能により、クラスターの状態と、モニターしているユーザー定義のワークロードに関する情報が提供されます。

クラスター管理者は、すべての OpenShift Container Platform のコアプロジェクトおよびユーザー定義プロジェクトのメトリクスをクエリーできます。

開発者として、メトリクスのクエリー時にプロジェクト名を指定する必要があります。選択したプロジェクトのメトリクスを表示するには、必要な権限が必要です。

12.3.2.1. クラスター管理者としてのすべてのプロジェクトのメトリクスのクエリー

クラスター管理者またはすべてのプロジェクトの表示権限を持つユーザーとして、メトリクス UI ですべてのデフォルト OpenShift Container Platform およびユーザー定義プロジェクトのメトリクスにアクセスできます。

前提条件

- **cluster-admin** クラスターロールまたはすべてのプロジェクトの表示権限を持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブから、**Observe → Metrics** を選択します。
2. 1 つ以上のクエリーを追加するには、次のいずれかを実行します。

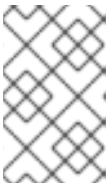
オプション	説明
カスタムクエリーを作成します。	Prometheus Query Language (PromQL) クエリーを **Expression** フィールドに追加します。 PromQL 式を入力すると、オートコンプリートの提案がドロップダウンリストに表示されます。これらの提案には、関数、メトリクス、ラベル、および時間トークンが含まれます。キーボードの矢印を使用して提案された項目のいずれかを選択し、Enter を押して項目を式に追加できます。また、マウスポインターを推奨項目の上に移動して、その項目の簡単な説明を表示することもできます。
複数のクエリーを追加します。	**クエリーの追加** を選択します。
既存のクエリーを複製します。	オプションメニューを選択します  クエリーの横にある **Duplicate query** を選択します。

オプション	説明
クエリーの実行を無効にします。	オプションメニューを選択します クエリーの横にある **Disable query** を選択します。

3. 作成したクエリーを実行するには、**Run queries** を選択します。クエリーからのメトリクスはプロットで可視化されます。クエリーが無効な場合は、UI にエラーメッセージが表示されます。

注記

大量のデータで動作するクエリーは、時系列グラフの描画時にタイムアウトするか、ブラウザーをオーバーロードする可能性があります。これを回避するには、**Hide graph** を選択し、メトリクステーブルのみを使用してクエリーを調整します。次に、使用できるクエリーを確認した後に、グラフを描画できるようにプロットを有効にします。



注記

デフォルトでは、クエリーテーブルに、すべてのメトリクスとその現在の値をリスト表示する拡張ビューが表示されます。˅ を選択すると、クエリーの拡張ビューを最小にすることができます。

4. オプション: ページ URL には、実行したクエリーが含まれます。このクエリーのセットを再度使用できるようにするには、この URL を保存します。

5. 視覚化されたメトリクスを調べます。最初に、有効な全クエリーの全メトリクスがプロットに表示されます。次のいずれかを実行して、表示するメトリクスを選択できます。

オプション	説明
クエリーからすべてのメトリクスを非表示にします。	オプションメニューをクリックします クエリーを選択し、**Hide all series** をクリックします。
特定のメトリクスを非表示にします。	クエリーテーブルに移動し、メトリクス名の近くにある色付きの四角形をクリックします。
プロットを拡大し、時間範囲を変更します。	次のいずれかになります。 ● プロットを水平にクリックし、ドラッグして、時間範囲を視覚的に選択します。 ● 左上隅のメニューを使用して、時間範囲を選択します。

オプション	説明
時間範囲をリセットします。	**Reset zoom** を選択します。
特定の時点でのすべてのクエリーの出力を表示します。	その時点でプロット上にマウスカーソルを置きます。クエリーの出力はポップアップに表示されます。
プロットを非表示にします。	**Hide graph** を選択します。

12.3.2.2. 開発者が行うユーザー定義プロジェクトのメトリクスのクエリー

ユーザー定義のプロジェクトのメトリクスには、開発者またはプロジェクトの表示権限を持つユーザーとしてアクセスできます。

Developer パースペクティブには、選択したプロジェクトの事前に定義された CPU、メモリー、帯域幅、およびネットワークパケットのクエリーが含まれます。また、プロジェクトの CPU、メモリー、帯域幅、ネットワークパケット、およびアプリケーションメトリクスについてカスタム Prometheus Query Language (PromQL) クエリーを実行することもできます。

注記

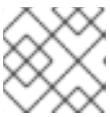
開発者は **Developer** パースペクティブのみを使用でき、**Administrator** パースペクティブは使用できません。開発者は、1 度に 1 つのプロジェクトのメトリクスのみをクエリーできます。

前提条件

- 開発者として、またはメトリクスで表示しているプロジェクトの表示権限を持つユーザーとしてクラスターへのアクセスがある。
- ユーザー定義プロジェクトのモニタリングが有効化されている。
- ユーザー定義プロジェクトにサービスをデプロイしている。
- サービスのモニター方法を定義するために、サービスの **ServiceMonitor** カスタムリソース定義 (CRD) を作成している。

手順

1. OpenShift Container Platform Web コンソールの **Developer** パースペクティブから、**Observe → Metrics** を選択します。
2. **Project:** 一覧でメトリクスで表示するプロジェクトを選択します。
3. **Select query** 一覧からクエリーを選択するか、**Show PromQL** を選択して、選択したクエリーに基づいてカスタム PromQL クエリーを作成します。クエリーからのメトリクスはプロットで可視化されます。



注記

Developer パースペクティブでは、1 度に 1 つのクエリーのみを実行できます。

4. 次のいずれかを実行して、視覚化されたメトリクスを調べます。

オプション	説明
プロットを拡大し、時間範囲を変更します。	次のいずれかになります。 ● プロットを水平にクリックし、ドラッグして、時間範囲を視覚的に選択します。 ● 左上隅のメニューを使用して、時間範囲を選択します。
時間範囲をリセットします。	**Reset zoom** を選択します。
特定の時点でのすべてのクエリーの出力を表示します。	その時点でプロット上にマウスカーソルを置きます。クエリーの出力はポップアップに表示されます。

12.3.3. 仮想化メトリクス

以下のメトリクスの記述には、Prometheus Query Language (PromQL) クエリーのサンプルが含まれます。これらのメトリクスは API ではなく、バージョン間で変更される可能性があります。仮想化メトリクスの完全なリストは、KubeVirt コンポーネントメトリクス を参照してください。

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。

12.3.3.1. vCPU メトリック

以下のクエリーは、入出力 I/O) を待機している仮想マシンを特定します。

kubevirt_vmi_vcpu_wait_seconds_total

仮想マシンの vCPU の待機時間 (秒単位) を返します。タイプ: カウンター。

'0' より大きい値は、仮想 CPU は実行する用意ができているが、ホストスケジューラーがこれをまだ実行できないことを意味します。実行できない場合には I/O に問題があることを示しています。

注記

vCPU メトリックをクエリーするには、最初に **schedstats=enable** カーネル引数を **MachineConfig** オブジェクトに適用する必要があります。このカーネル引数を使用すると、デバッグとパフォーマンスチューニングに使用されるスケジューラーの統計が有効になり、スケジューラーに小規模な負荷を追加できます。

vCPU 待機時間クエリーの例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_vcpu_wait_seconds_total[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。

12.3.3.2. ネットワークメトリック

以下のクエリーは、ネットワークを飽和状態にしている仮想マシンを特定できます。

kubevirt_vmi_network_receive_bytes_total

仮想マシンのネットワークで受信したトラフィックの合計量 (バイト単位) を返します。タイプ: カウンター。

kubevirt_vmi_network_transmit_bytes_total

仮想マシンのネットワーク上で送信されるトラフィックの合計量 (バイト単位) を返します。タイプ: カウンター。

ネットワークトラフィッククエリーの例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_network_receive_bytes_total[6m])) + sum by (name, namespace) (rate(kubevirt_vmi_network_transmit_bytes_total[6m]))) > 0 1
```

1 このクエリーは、6 分間の任意のタイミングで最大のネットワークトラフィックを送信する上位 3 の仮想マシンを返します。

12.3.3.3. ストレージメトリクス

12.3.3.3.1. ストレージ関連のトラフィック

以下のクエリーは、大量のデータを書き込んでいる仮想マシンを特定できます。

kubevirt_vmi_storage_read_traffic_bytes_total

仮想マシンのストレージ関連トラフィックの合計量 (バイト単位) を返します。タイプ: カウンター。

kubevirt_vmi_storage_write_traffic_bytes_total

仮想マシンのストレージ関連トラフィックのストレージ書き込みの合計量 (バイト単位) を返します。タイプ: カウンター。

ストレージ関連のトラフィッククエリーの例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_storage_read_traffic_bytes_total[6m])) + sum by (name, namespace) (rate(kubevirt_vmi_storage_write_traffic_bytes_total[6m]))) > 0 1
```

1 上記のクエリーは、6 分間の任意のタイミングで最も大きなストレージトラフィックを送信する上位 3 の仮想マシンを返します。

12.3.3.3.2. ストレージスナップショットデータ

kubevirt_vmsnapshot_disks_restored_from_source

ソース仮想マシンから復元された仮想マシンディスクの総数を返します。タイプ: ゲージ。

kubevirt_vmsnapshot_disks_restored_from_source_bytes

ソース仮想マシンから復元された容量をバイト単位で返します。タイプ: ゲージ。

ストレージスナップショットデータクエリーの例

```
kubevirt_vmsnapshot_disks_restored_from_source{vm_name="simple-vm",
vm_namespace="default"} 1
```

1 このクエリーは、ソース仮想マシンから復元された仮想マシンディスクの総数を返します。

```
kubevirt_vmsnapshot_disks_restored_from_source_bytes{vm_name="simple-vm",
vm_namespace="default"} 1
```

1 このクエリーは、ソース仮想マシンから復元された容量をバイト単位で返します。

12.3.3.3.3. I/O パフォーマンス

以下のクエリーで、ストレージデバイスの I/O パフォーマンスを判別できます。

kubevirt_vmi_storage_iops_read_total
仮想マシンが実行している 1 秒あたりの書き込み I/O 操作の量を返します。タイプ: カウンター。

kubevirt_vmi_storage_iops_write_total
仮想マシンが実行している 1 秒あたりの読み取り I/O 操作の量を返します。タイプ: カウンター。

I/O パフォーマンスクエリーの例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_storage_iops_read_total[6m])) + sum by
(name, namespace) (rate(kubevirt_vmi_storage_iops_write_total[6m]))) > 0 1
```

1 上記のクエリーは、6 分間の任意のタイミングで最も大きな I/O 操作を実行している上位 3 の仮想マシンを返します。

12.3.3.4. ゲストメモリーのスワップメトリクス

以下のクエリーにより、メモリースワップを最も多く実行しているスワップ対応ゲストを特定できます。

kubevirt_vmi_memory_swap_in_traffic_bytes
仮想ゲストがスワップされているメモリーの合計量 (バイト単位) を返します。タイプ: ゲージ。

kubevirt_vmi_memory_swap_out_traffic_bytes
仮想ゲストがスワップアウトされているメモリーの合計量 (バイト単位) を返します。タイプ: ゲージ。

メモリースワップクエリーの例

```
topk(3, sum by (name, namespace) (rate(kubevirt_vmi_memory_swap_in_traffic_bytes[6m])) + sum
by (name, namespace) (rate(kubevirt_vmi_memory_swap_out_traffic_bytes[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。

12.3.3.5. ライブマイグレーションのメトリクス

次のメトリクスをクエリーして、ライブマイグレーションのステータスを表示できます。

kubevirt_vmi_migration_data_processed_bytes

新しい仮想マシン (VM) に移行されたゲストオペレーティングシステムデータの量。タイプ: ゲージ。

kubevirt_vmi_migration_data_remaining_bytes

移行されていないゲストオペレーティングシステムデータの量。タイプ: ゲージ。

kubevirt_vmi_migration_memory_transfer_rate_bytes

ゲスト OS でメモリーがダーティーになる速度。ダーティメモリーとは、変更されたがまだディスクに書き込まれていないデータです。タイプ: ゲージ。

kubevirt_vmi_migrations_in_pending_phase

保留中の移行の数。タイプ: ゲージ。

kubevirt_vmi_migrations_in_scheduling_phase

スケジュール移行の数。タイプ: ゲージ。

kubevirt_vmi_migrations_in_running_phase

実行中の移行の数。タイプ: ゲージ。

kubevirt_vmi_migration_succeeded

正常に完了した移行の数。タイプ: ゲージ。

kubevirt_vmi_migration_failed

失敗した移行の数。タイプ: ゲージ。

12.3.4. 関連情報

- モニタリングの概要
- Querying Prometheus
- Prometheus クエリーの例

12.4. 仮想マシンのカスタムメトリクスの公開

OpenShift Container Platform には、コアプラットフォームコンポーネントのモニタリングを提供する事前に設定され、事前にインストールされた自己更新型のモニタリングスタックが含まれます。このモニタリングスタックは、Prometheus モニタリングシステムをベースにしています。Prometheus は Time Series を使用するデータベースであり、メトリクスのルール評価エンジンです。

OpenShift Container Platform モニタリングスタックの使用のほかに、CLI を使用してユーザー定義プロジェクトのモニタリングを有効にし、**node-exporter** サービスで仮想マシン用に公開されるカスタムメトリックをクエリーできます。

12.4.1. ノードエクスポーターサービスの設定

node-exporter エージェントは、メトリクスを収集するクラスター内のすべての仮想マシンにデプロイされます。node-exporter エージェントをサービスとして設定し、仮想マシンに関連付けられた内部メトリクスおよびプロセスを公開します。

前提条件

- OpenShift Container Platform CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。
- **cluster-monitoring-config ConfigMap** オブジェクトを **openshift-monitoring** プロジェクトに作成する。
- **enableUserWorkload** を **true** に設定して、**user-workload-monitoring-config ConfigMap** オブジェクトを **openshift-user-workload-monitoring** プロジェクトに設定する。

手順

1. **Service** YAML ファイルを作成します。以下の例では、このファイルは **node-exporter-service.yaml** という名前です。

```
kind: Service
apiVersion: v1
metadata:
  name: node-exporter-service 1
  namespace: dynamation 2
  labels:
    servicetype: metrics 3
spec:
  ports:
    - name: exmet 4
      protocol: TCP
      port: 9100 5
      targetPort: 9100 6
  type: ClusterIP
  selector:
    monitor: metrics 7
```

1. 仮想マシンからメトリクスを公開する node-exporter サービス。
2. サービスが作成される namespace。
3. サービスのラベル。**ServiceMonitor** はこのラベルを使用してこのサービスを照会します。
4. **ClusterIP** サービスのポート 9100 でメトリクスを公開するポートに指定された名前。
5. リクエストをリッスンするために **node-exporter-service** によって使用されるターゲットポート。
6. **monitor** ラベルが設定された仮想マシンの TCP ポート番号。
7. 仮想マシンの Pod を照会するために使用されるラベル。この例では、ラベル **monitor** のある仮想マシンの Pod と、**metrics** の値がマッチします。

2. node-exporter サービスを作成します。

```
$ oc create -f node-exporter-service.yaml
```

12.4.2. ノードエクスポーターサービスが設定された仮想マシンの設定

node-exporter ファイルを仮想マシンにダウンロードします。次に、仮想マシンの起動時に node-exporter サービスを実行する **systemd** サービスを作成します。

前提条件

- コンポーネントの Pod は **openshift-user-workload-monitoring** プロジェクトで実行されます。
- このユーザー定義プロジェクトをモニターする必要のあるユーザーに **monitoring-edit** ロールを付与します。

手順

1. 仮想マシンにログインします。
2. **node-exporter** ファイルのバージョンに適用されるディレクトリーパスを使用して、**node-exporter** ファイルを仮想マシンにダウンロードします。

```
$ wget
https://github.com/prometheus/node_exporter/releases/download/v1.3.1/node_exporter-
1.3.1.linux-amd64.tar.gz
```

3. 実行ファイルを展開して、**/usr/bin** ディレクトリーに配置します。

```
$ sudo tar xvf node_exporter-1.3.1.linux-amd64.tar.gz \
    --directory /usr/bin --strip 1 "*/node_exporter"
```

4. ディレクトリーのパス**/etc/systemd/system**に **node_exporter.service** ファイルを作成します。この **systemd** サービスファイルは、仮想マシンの再起動時に node-exporter サービスを実行します。

```
[Unit]
Description=Prometheus Metrics Exporter
After=network.target
StartLimitIntervalSec=0

[Service]
Type=simple
Restart=always
RestartSec=1
User=root
ExecStart=/usr/bin/node_exporter

[Install]
WantedBy=multi-user.target
```

5. **systemd** サービスを有効にし、起動します。

```
$ sudo systemctl enable node_exporter.service
$ sudo systemctl start node_exporter.service
```

検証

- node-exporter エージェントが仮想マシンからのメトリクスを報告していることを確認します。

```
$ curl http://localhost:9100/metrics
```

出力例

```
go_gc_duration_seconds{quantile="0"} 1.5244e-05
go_gc_duration_seconds{quantile="0.25"} 3.0449e-05
go_gc_duration_seconds{quantile="0.5"} 3.7913e-05
```

12.4.3. 仮想マシンのカスタムモニタリングラベルの作成

単一サービスから複数の仮想マシンに対するクエリーを有効にするには、仮想マシンの YAML ファイルにカスタムラベルを追加します。

前提条件

- OpenShift Container Platform CLI (**oc**) がインストールされている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- 仮想マシンを停止および再起動するための Web コンソールへのアクセス権限がある。

手順

1. 仮想マシン設定ファイルの **template** spec を編集します。この例では、ラベル **monitor** の値が **metrics** になります。

```
spec:
  template:
    metadata:
      labels:
        monitor: metrics
```

2. 仮想マシンを停止して再起動し、**monitor** ラベルに指定されたラベル名を持つ新しい Pod を作成します。

12.4.3.1. メトリクスを取得するための node-exporter サービスのクエリー

仮想マシンのメトリクスは、**/metrics** の正規名の下に HTTP サービスエンドポイント経由で公開されます。メトリクスのクエリー時に、Prometheus は仮想マシンによって公開されるメトリクスエンドポイントからメトリクスを直接収集し、これらのメトリクスを確認用に表示します。

前提条件

- **cluster-admin** 権限を持つユーザーまたは **monitoring-edit** ロールを持つユーザーとしてクラスターにアクセスできる。

- node-exporter サービスを設定して、ユーザー定義プロジェクトのモニタリングを有効にしている。

手順

1. サービスの namespace を指定して、HTTP サービスエンドポイントを取得します。

```
$ oc get service -n <namespace> <node-exporter-service>
```

2. node-exporter サービスの利用可能なすべてのメトリクスを一覧表示するには、**metrics** リソースをクエリーします。

```
$ curl http://<172.30.226.162:9100>/metrics | grep -vE "^#|^$"
```

出力例

```
node_arp_entries{device="eth0"} 1
node_boot_time_seconds 1.643153218e+09
node_context_switches_total 4.4938158e+07
node_cooling_device_cur_state{name="0",type="Processor"} 0
node_cooling_device_max_state{name="0",type="Processor"} 0
node_cpu_guest_seconds_total{cpu="0",mode="nice"} 0
node_cpu_guest_seconds_total{cpu="0",mode="user"} 0
node_cpu_seconds_total{cpu="0",mode="idle"} 1.10586485e+06
node_cpu_seconds_total{cpu="0",mode="iowait"} 37.61
node_cpu_seconds_total{cpu="0",mode="irq"} 233.91
node_cpu_seconds_total{cpu="0",mode="nice"} 551.47
node_cpu_seconds_total{cpu="0",mode="softirq"} 87.3
node_cpu_seconds_total{cpu="0",mode="steal"} 86.12
node_cpu_seconds_total{cpu="0",mode="system"} 464.15
node_cpu_seconds_total{cpu="0",mode="user"} 1075.2
node_disk_discard_time_seconds_total{device="vda"} 0
node_disk_discard_time_seconds_total{device="vdb"} 0
node_disk_discarded_sectors_total{device="vda"} 0
node_disk_discarded_sectors_total{device="vdb"} 0
node_disk_discards_completed_total{device="vda"} 0
node_disk_discards_completed_total{device="vdb"} 0
node_disk_discards_merged_total{device="vda"} 0
node_disk_discards_merged_total{device="vdb"} 0
node_disk_info{device="vda",major="252",minor="0"} 1
node_disk_info{device="vdb",major="252",minor="16"} 1
node_disk_io_now{device="vda"} 0
node_disk_io_now{device="vdb"} 0
node_disk_io_time_seconds_total{device="vda"} 174
node_disk_io_time_seconds_total{device="vdb"} 0.054
node_disk_io_time_weighted_seconds_total{device="vda"} 259.79200000000003
node_disk_io_time_weighted_seconds_total{device="vdb"} 0.039
node_disk_read_bytes_total{device="vda"} 3.71867136e+08
node_disk_read_bytes_total{device="vdb"} 366592
node_disk_read_time_seconds_total{device="vda"} 19.128
node_disk_read_time_seconds_total{device="vdb"} 0.039
node_disk_reads_completed_total{device="vda"} 5619
node_disk_reads_completed_total{device="vdb"} 96
node_disk_reads_merged_total{device="vda"} 5
node_disk_reads_merged_total{device="vdb"} 0
```

```
node_disk_write_time_seconds_total{device="vda"} 240.66400000000002
node_disk_write_time_seconds_total{device="vdb"} 0
node_disk_writes_completed_total{device="vda"} 71584
node_disk_writes_completed_total{device="vdb"} 0
node_disk_writes_merged_total{device="vda"} 19761
node_disk_writes_merged_total{device="vdb"} 0
node_disk_written_bytes_total{device="vda"} 2.007924224e+09
node_disk_written_bytes_total{device="vdb"} 0
```

12.4.4. ノードエクスポーターサービスの ServiceMonitor リソースの作成

Prometheus クライアントライブラリーを使用し、**/metrics** エンドポイントからメトリクスを収集して、node-exporter サービスが公開するメトリクスにアクセスし、表示できます。**ServiceMonitor** カスタムリソース定義 (CRD) を使用して、ノードエクスポーターサービスをモニターします。

前提条件

- **cluster-admin** 権限を持つユーザーまたは **monitoring-edit** ロールを持つユーザーとしてクラスターにアクセスできる。
- node-exporter サービスを設定して、ユーザー定義プロジェクトのモニタリングを有効にしている。

手順

1. **ServiceMonitor** リソース設定の YAML ファイルを作成します。この例では、サービスモニターはラベル **metrics** が指定されたサービスとマッチし、30 秒ごとに **exmet** ポートをクエリーします。

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: node-exporter-metrics-monitor
  name: node-exporter-metrics-monitor (1)
  namespace: dynamation (2)
spec:
  endpoints:
  - interval: 30s (3)
    port: exmet (4)
    scheme: http
  selector:
    matchLabels:
      servicetype: metrics
```

(1) **ServiceMonitor** の名前。

(2) **ServiceMonitor** が作成される namespace。

(3) ポートをクエリーする間隔。

(4) 30 秒ごとにクエリーされるポートの名前

2. node-exporter サービスの **ServiceMonitor** 設定を作成します。

```
$ oc create -f node-exporter-metrics-monitor.yaml
```

12.4.4.1. クラスター外のノードエクスポーターサービスへのアクセス

クラスター外の node-exporter サービスにアクセスし、公開されるメトリクスを表示できます。

前提条件

- **cluster-admin** 権限を持つユーザーまたは **monitoring-edit** ロールを持つユーザーとしてクラスターにアクセスできる。
- node-exporter サービスを設定して、ユーザー定義プロジェクトのモニタリングを有効にしている。

手順

1. node-exporter サービスを公開します。

```
$ oc expose service -n <namespace> <node_exporter_service_name>
```

2. ルートの FQDN(完全修飾ドメイン名) を取得します。

```
$ oc get route -o=custom-columns=NAME:.metadata.name,DNS:.spec.host
```

出力例

```
NAME                    DNS
node-exporter-service   node-exporter-service-dynamation.apps.cluster.example.org
```

3. **curl** コマンドを使用して、node-exporter サービスのメトリクスを表示します。

```
$ curl -s http://node-exporter-service-dynamation.apps.cluster.example.org/metrics
```

出力例

```
go_gc_duration_seconds{quantile="0"} 1.5382e-05
go_gc_duration_seconds{quantile="0.25"} 3.1163e-05
go_gc_duration_seconds{quantile="0.5"} 3.8546e-05
go_gc_duration_seconds{quantile="0.75"} 4.9139e-05
go_gc_duration_seconds{quantile="1"} 0.000189423
```

12.4.5. 関連情報

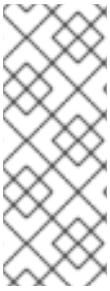
- モニタリングスタックの設定
- ユーザー定義プロジェクトのモニタリングの有効化
- メトリックの管理
- モニタリングダッシュボードの確認
- ヘルスチェックの使用によるアプリケーションの正常性の監視

- 設定マップの作成および使用
- 仮想マシンの状態の制御

12.5. 仮想マシンの下向きメトリクスの公開

管理者は、まず **downwardMetrics** フィーチャーゲートを有効にし、次に **downwardMetrics** デバイスを設定することで、ホストおよび仮想マシン (VM) メトリクスの限定セットをゲスト仮想マシンに公開できます。

ユーザーは、コマンドラインまたは **仮想マシン -dump- メトリクスツール** を使用してメトリクスの結果を表示できます。



注記

Red Hat Enterprise Linux (RHEL) 9 では、コマンドラインを使用して下向きメトリクスを表示します。コマンドラインを使用した下向きメトリクスの表示 を参照してください。

vm-dump-metrics ツールは、Red Hat Enterprise Linux (RHEL) 9 プラットフォームではサポートされていません。

12.5.1. downwardMetrics フィーチャーゲートの有効化または無効化

次のいずれかのアクションを実行することにより、**downwardMetrics** フィーチャーゲートを有効または無効にできます。

- デフォルトのエディターで HyperConverged カスタムリソース (CR) を編集する
- コマンドラインを使用する

12.5.1.1. YAML ファイルでの下向きメトリクスフィーチャーゲートの有効化または無効化

ホスト仮想マシンの下向きメトリクスを公開するには、YAML ファイルを編集して **downwardMetrics** フィーチャーゲートを有効にします。

前提条件

- 機能ゲートを有効にするには、管理者権限が必要です。

手順

1. 次のコマンドを実行して、デフォルトのエディターで HyperConverged カスタムリソース (CR) を開きます。

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

2. 次のように、downwardMetrics フィーチャーゲートを有効または無効にすることを選択します。
   - **downwardMetrics** フィーチャーゲートを有効にするには、**spec.featureGates.downwardMetrics** を追加してから **true** に設定します。以下に例を示します。

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

- **downwardMetrics** フィーチャーゲートを無効にするには、**spec.featureGates.downwardMetrics** を **false** に設定します。以下に例を示します。

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

12.5.1.2. コマンドラインからの下向きメトリクスフィーチャーゲートの有効化または無効化

ホスト仮想マシンの下向きメトリクスを公開するには、コマンドラインを使用して、**downwardMetrics** フィーチャーゲートを有効にします。

前提条件

- 機能ゲートを有効にするには、管理者権限が必要です。

手順

- 以下のように **downwardMetrics** フィーチャーゲートを有効または無効にするか選択します。
  - 次の例に示すコマンドを実行して、**downwardMetrics** フィーチャーゲートを有効にします。

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

  - 次の例に示すコマンドを実行して、**downwardMetrics** フィーチャーゲートを無効にします。

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

12.5.2. 下向きメトリクスデバイスの設定

ホスト仮想マシンの下向きメトリクスのキャプチャーを有効にするには、**downwardMetrics** デバイスを含む設定ファイルを作成します。このデバイスを追加すると、メトリクスが **virtio-serial** ポートを通じて公開されるようになります。

前提条件

- 最初に **downwardMetrics** フィーチャーゲートを有効にする必要があります。

手順

- 次の例に示すように、**downwardMetrics** デバイスを含む YAML ファイルを編集または作成します。

  downwardMetrics 設定ファイルの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: fedora
  namespace: default
spec:
  dataVolumeTemplates:
    - metadata:
        name: fedora-volume
      spec:
        sourceRef:
          kind: DataSource
          name: fedora
          namespace: openshift-virtualization-os-images
        storage:
          resources: {}
          storageClassName: hostpath-csi-basic
  instancetype:
    name: u1.medium
  preference:
    name: fedora
  running: true
  template:
    metadata:
      labels:
        app.kubernetes.io/name: headless
    spec:
      domain:
        devices:
          downwardMetrics: {} (1)
      subdomain: headless
      volumes:
        - dataVolume:
            name: fedora-volume
          name: rootdisk
        - cloudInitNoCloud:
            userData: |
              #cloud-config
              chpasswd:
                expire: false
```

```
      password: '<password>' 2
      user: fedora
    name: cloudinitdisk
```

1. **downwardMetrics** デバイス。
2. **fedora** ユーザーのパスワード。

12.5.3. 下向きメトリクスの表示

以下のオプションのいずれかを使用して、下向きメトリクスを表示できます。

- コマンドラインインターフェイス (CLI)
- **vm-dump-metrics** ツール

注記

Red Hat Enterprise Linux (RHEL) 9 では、コマンドラインを使用して下向きメトリクスを表示します。vm-dump-metrics ツールは、Red Hat Enterprise Linux (RHEL) 9 プラットフォームではサポートされていません。

12.5.3.1. コマンドラインを使用した下向きメトリクスの表示

ゲスト仮想マシン (VM) 内からコマンドを入力すると、下向きメトリクスを表示できます。

手順

- 以下のコマンドを実行します。

```
$ sudo sh -c 'printf "GET /metrics/XML\n\n" > /dev/virtio-ports/org.github.vhostmd.1'
```

```
$ sudo cat /dev/virtio-ports/org.github.vhostmd.1
```

12.5.3.2. vm-dump-metrics ツールを使用した下向きメトリクスの表示

下向きメトリクスを表示するには、**vm-dump-metrics** ツールをインストールし、そのツールを使用してメトリクスの結果を公開します。

注記

Red Hat Enterprise Linux (RHEL) 9 では、コマンドラインを使用して下向きメトリクスを表示します。vm-dump-metrics ツールは、Red Hat Enterprise Linux (RHEL) 9 プラットフォームではサポートされていません。

手順

1. 以下のコマンドを実行して **vm-dump-metrics** ツールをインストールします。

```
$ sudo dnf install -y vm-dump-metrics
```

2. 次のコマンドを実行して、メトリクス結果を取得します。

```
$ sudo vm-dump-metrics
```

出力例

```
<metrics>
  <metric type="string" context="host">
    <name>HostName</name>
    <value>node01</value>
[...]
  <metric type="int64" context="host" unit="s">
    <name>Time</name>
    <value>1619008605</value>
  </metric>
  <metric type="string" context="host">
    <name>VirtualizationVendor</name>
    <value>kubevirt.io</value>
  </metric>
</metrics>
```

12.6. 仮想マシンのヘルスチェック

VirtualMachine リソースで readiness プローブと liveness プローブを定義することにより、仮想マシン (VM) のヘルスチェックを設定できます。

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

readiness プローブと liveness プローブを使用して、異常な仮想マシン (VM) を検出および処理します。VM の仕様に 1 つ以上のプローブを含めて、準備ができていない VM にトラフィックが到達しないようにし、VM が応答しなくなったときに新しい VM が作成されるようにすることができます。

readiness プローブ は、VM がサービス要求を受け入れることができるかどうかを判断します。プローブが失敗すると、VM は、準備状態になるまで、利用可能なエンドポイントのリストから削除されます。

liveness プローブ は、VM が応答しているかどうかを判断します。プローブが失敗すると、VM は削除され、応答性を復元するために、新しい VM が作成されます。

VirtualMachine オブジェクトの **spec.readinessProbe** フィールドと **spec.livenessProbe** フィールドを設定することで、readiness プローブと liveness プローブを設定できます。これらのフィールドは、以下のテストをサポートします。

HTTP GET

プローブは、Web フックを使用して VM の正常性を判断します。このテストは、HTTP の応答コードが 200 から 399 までの値の場合に正常と見なされます。完全に初期化されている場合に、HTTP ステータスコードを返すアプリケーションで HTTP GET テストを使用できます。

TCP ソケット

プローブは、VM へのソケットを開こうとします。プローブが接続を確立できる場合のみ、VM は正常であると見なされます。TCP ソケットテストは、初期化が完了するまでリスニングを開始しないアプリケーションで使用できます。

ゲストエージェントの ping

プローブは、**guest-ping** コマンドを使用して、QEMU ゲストエージェントが仮想マシンで実行されているかどうかを判断します。

12.6.1.1. HTTP readiness プローブの定義

仮想マシン (VM) 設定の **spec.readinessProbe.httpGet** フィールドを設定して、HTTP readiness プローブを定義します。

手順

1. VM 設定ファイルに readiness プローブの詳細を含めます。

HTTP GET テストを使用した readiness プローブの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        httpGet: 1
          port: 1500 2
          path: /healthz 3
          httpHeaders:
          - name: Custom-Header
            value: Awesome
        initialDelaySeconds: 120 4
        periodSeconds: 20 5
        timeoutSeconds: 10 6
        failureThreshold: 3 7
        successThreshold: 3 8
# ...
```

1. VM に接続するために実行する HTTP GET 要求。
2. プローブがクエリーする VM のポート。上記の例では、プローブはポート 1500 をクエリーします。
3. HTTP サーバーでアクセスするパス。上記の例では、サーバーの /healthz パスのハンドラーが成功コードを返した場合、VM は正常であると見なされます。ハンドラーが失敗コードを返した場合、VM は使用可能なエンドポイントのリストから削除されます。
4. VM が起動してから準備プローブが開始されるまでの時間 (秒単位)。
5. プローブの実行間の遅延 (秒単位)。デフォルトの遅延は 10 秒です。この値は **timeoutSeconds** よりも大きくなければなりません。
6. プローブがタイムアウトになり、VM が失敗したと見なされるまでの非アクティブな秒数。デフォルト値は 1 です。この値は **periodSeconds** 未満である必要があります。
7. プローブが失敗できる回数。デフォルトは 3 です。指定された試行回数になると、Pod には **Unready** というマークが付けられます。

8 成功とみなされるまでにプローブが失敗後に成功を報告する必要のある回数。デフォルトでは 1 回です。

2. 次のコマンドを実行して VM を作成します。

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

12.6.1.2. TCP readiness プローブの定義

仮想マシン (VM) 設定の **spec.readinessProbe.tcpSocket** フィールドを設定して、TCP readiness プローブを定義します。

手順

1. TCP readiness プローブの詳細を VM 設定ファイルに追加します。

TCP ソケットテストを含む readiness プローブの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        initialDelaySeconds: 120 1
        periodSeconds: 20 2
        tcpSocket: 3
          port: 1500 4
        timeoutSeconds: 10 5
# ...
```

1 VM が起動してから準備プローブが開始されるまでの時間 (秒単位)。

2 プローブの実行間の遅延 (秒単位)。デフォルトの遅延は 10 秒です。この値は **timeoutSeconds** よりも大きくなければなりません。

3 実行する TCP アクション。

4 プローブがクエリーする VM のポート。

5 プローブがタイムアウトになり、VM が失敗したと見なされるまでの非アクティブな秒数。デフォルト値は 1 です。この値は **periodSeconds** 未満である必要があります。

2. 次のコマンドを実行して VM を作成します。

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

12.6.1.3. HTTP liveness プローブの定義

仮想マシン (VM) 設定の **spec.livenessProbe.httpGet** フィールドを設定して、HTTP liveness プローブを定義します。readiness プローブと同様に、liveness プローブの HTTP および TCP テストの両方を定義できます。この手順では、HTTP GET テストを使用して liveness プローブのサンプルを設定します。

手順

1. VM 設定ファイルに HTTP liveness プローブの詳細を含めます。

HTTP GET テストを使用した liveness プローブの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      livenessProbe:
        initialDelaySeconds: 120 (1)
        periodSeconds: 20 (2)
        httpGet: (3)
          port: 1500 (4)
          path: /healthz (5)
          httpHeaders:
          - name: Custom-Header
            value: Awesome
        timeoutSeconds: 10 (6)
# ...
```

(1) VM が起動してから liveness プローブが開始されるまでの時間 (秒単位)。

(2) プローブの実行間の遅延 (秒単位)。デフォルトの遅延は 10 秒です。この値は **timeoutSeconds** よりも大きくなければなりません。

(3) VM に接続するために実行する HTTP GET 要求。

(4) プローブがクエリーする VM のポート。上記の例では、プローブはポート 1500 をクエリーします。VM は、cloud-init を介してポート 1500 に最小限の HTTP サーバーをインストールして実行します。

(5) HTTP サーバーでアクセスするパス。上記の例では、サーバーの **/healthz** パスのハンドラーが成功コードを返した場合、VM は正常であると見なされます。ハンドラーが失敗コードを返した場合、VM は削除され、新しい VM が作成されます。

(6) プローブがタイムアウトになり、VM が失敗したと見なされるまでの非アクティブな秒数。デフォルト値は 1 です。この値は **periodSeconds** 未満である必要があります。

2. 次のコマンドを実行して VM を作成します。

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

12.6.2. ウォッチドッグの定義

次の手順を実行して、ゲスト OS の正常性を監視するウォッチドッグを定義できます。

1. 仮想マシン (VM) のウォッチドッグデバイスを設定します。
2. ゲストにウォッチドッグエージェントをインストールします。

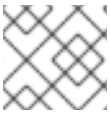
ウォッチドッグデバイスはエージェントを監視し、ゲストオペレーティングシステムが応答しない場合、次のいずれかのアクションを実行します。

- **poweroff**: VM の電源がすぐにオフになります。**spec.running** が **true** に設定されているか、**spec.runStrategy** が **manual** に設定されていない場合、VM は再起動します。
- **reset**: VM はその場で再起動し、ゲストオペレーティングシステムは反応できません。

注記

再起動時間が原因で liveness プローブがタイムアウトする場合があります。クラスターレベルの保護が liveness プローブの失敗を検出すると、VM が強制的に再スケジュールされ、再起動時間が長くなる可能性があります。

- **shutdown**: すべてのサービスを停止することで、VM の電源を正常にオフにします。



注記

ウォッチドッグは、Windows VM では使用できません。

12.6.2.1. 仮想マシンのウォッチドッグデバイスの設定

仮想マシン (VM) のウォッチドッグデバイスを設定するとします。

前提条件

- VM には、**i6300esb** ウォッチドッグデバイスのカーネルサポートが必要です。Red Hat Enterprise Linux(RHEL) イメージが、**i6300esb** をサポートしている。

手順

1. 次の内容で **YAML** ファイルを作成します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm2-rhel84-watchdog
  name: <vm-name>
spec:
  running: false
  template:
    metadata:
```

```
    labels:
      kubevirt.io/vm: vm2-rhel84-watchdog
  spec:
    domain:
      devices:
        watchdog:
          name: <watchdog>
          i6300esb:
            action: "poweroff" (1)
# ...
```

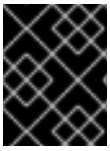
(1) **poweroff**、**reset**、または **shutdown** を指定します。

上記の例では、電源オフアクションを使用して、RHEL8 VM で **i6300esb** ウォッチドッグデバイスを設定し、デバイスを **/dev/watchdog** として公開します。

このデバイスは、ウォッチドッグバイナリーで使用できるようになりました。

2. 以下のコマンドを実行して、YAML ファイルをクラスターに適用します。

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



重要

この手順は、ウォッチドッグ機能をテストするためにのみ提供されており、実稼働マシンでは実行しないでください。

1. 以下のコマンドを実行して、VM がウォッチドッグデバイスに接続されていることを確認します。

```
$ lspci | grep watchdog -i
```

2. 以下のコマンドのいずれかを実行して、ウォッチドッグがアクティブであることを確認します。

- カーネルパニックをトリガーします。

```
# echo c > /proc/sysrq-trigger
```

- ウォッチドッグサービスを停止します。

```
# pkill -9 watchdog
```

12.6.2.2. ゲストへのウォッチドッグエージェントのインストール

ゲストにウォッチドッグエージェントをインストールし、**watchdog** サービスを開始します。

手順

1. root ユーザーとして仮想マシンにログインします。
2. **watchdog** ドッグパッケージとその依存関係をインストールします。

```
# yum install watchdog
```

3. **/etc/watchdog.conf** ファイルの次の行のコメントを外し、変更を保存します。

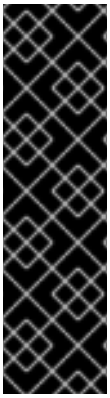
```
#watchdog-device = /dev/watchdog
```

4. 起動時に **watchdog** サービスを開始できるようにします。

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

12.6.3. ゲストエージェントの ping プローブの定義

仮想マシン (VM) 設定の **spec.readinessProbe.guestAgentPing** フィールドを設定して、ゲストエージェント ping プローブを定義します。



重要

ゲストエージェント ping プローブは、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

前提条件

- 仮想マシンに QEMU ゲストエージェントをインストールして有効にする必要があります。

手順

1. VM 設定ファイルにゲストエージェント ping プローブの詳細を含めます。以下に例を示します。

ゲストエージェント ping プローブの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  annotations:
  name: fedora-vm
  namespace: example-namespace
# ...
spec:
  template:
    spec:
      readinessProbe:
        guestAgentPing: {} 1
        initialDelaySeconds: 120 2
        periodSeconds: 20 3
        timeoutSeconds: 10 4
```

```
      failureThreshold: 3 5
      successThreshold: 3 6
# ...
```

1. VM に接続するためのゲストエージェント ping プローブ。
2. オプション: VM が起動してからゲストエージェントプローブが開始されるまでの時間 (秒単位)。
3. オプション: プローブを実行する間の遅延 (秒単位)。デフォルトの遅延は 10 秒です。この値は **timeoutSeconds** よりも大きくなければなりません。
4. オプション: プローブがタイムアウトになり、VM が失敗したと見なされるまでの非アクティブの秒数。デフォルト値は 1 です。この値は **periodSeconds** 未満である必要があります。
5. オプション: プローブが失敗できる回数。デフォルトは 3 です。指定された試行回数になると、Pod には **Unready** というマークが付けられます。
6. オプション: 成功とみなされるまでにプローブが失敗後に成功を報告する必要のある回数。デフォルトでは 1 回です。

2. 次のコマンドを実行して VM を作成します。

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

12.6.4. 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

12.7. OPENSHIFT VIRTUALIZATION の RUNBOOK

OpenShift Virtualization Operator の runbook は、openshift/runbooks Git リポジトリーで管理されており、GitHub で表示できます。OpenShift Virtualization アラート をトリガーする問題を診断して解決するには、runbook の手順に従います。

OpenShift Virtualization のアラートは、Web コンソールの **Virtualization → Overview** タブに表示されます。

12.7.1. CDIDataImportCronOutdated

- **CDIDataImportCronOutdated** アラートの runbook を表示 します。

12.7.2. CDIDataVolumeUnusualRestartCount

- **CDIDataVolumeUnusualRestartCount** アラートの runbook を表示 します。

12.7.3. CDIDefaultStorageClassDegraded

- **CDIDefaultStorageClassDegraded** アラートの runbook を表示 します。

12.7.4. CDIMultipleDefaultVirtStorageClasses

- **CDIMultipleDefaultVirtStorageClasses** アラートの runbook を表示 します。

12.7.5. CDINoDefaultStorageClass

- **CDINoDefaultStorageClass** アラートの runbook を表示 します。

12.7.6. CDINotReady

- **CDINotReady** アラートの runbook を表示 します。

12.7.7. CDIOperatorDown

- **CDIOperatorDown** アラートの runbook を表示 します。

12.7.8. CDIStorageProfilesIncomplete

- **CDIStorageProfilesIncomplete** アラートの runbook を表示 します。

12.7.9. CnaoDown

- **CnaoDown** アラートの runbook を表示 します。

12.7.10. CnaoNMstateMigration

- **CnaoNMstateMigration** アラートの runbook を表示 します。

12.7.11. HCOInstallationIncomplete

- **HCOInstallationIncomplete** アラートの runbook を表示 します。

12.7.12. HPPNotReady

- **HPPNotReady** アラートの runbook を表示します。

12.7.13. HPPOperatorDown

- **HPPOperatorDown** アラートの runbook を表示 します。

12.7.14. HPPSharingPoolPathWithOS

- **HPPSharingPoolPathWithOS** アラートの runbook を表示 します。

12.7.15. KubemacpoolDown

- **KubemacpoolDown** アラートの runbook を表示 します。

12.7.16. KubeMacPoolDuplicateMacsFound

- **KubeMacPoolDuplicateMacsFound** アラートの runbook を表示 します。

12.7.17. KubeVirtComponentExceedsRequestedCPU

- **KubeVirtComponentExceedsRequestedCPU** アラートが 非推奨 になりました。

12.7.18. KubeVirtComponentExceedsRequestedMemory

- **KubeVirtComponentExceedsRequestedMemory** アラートは 非推奨 になりました。

12.7.19. KubeVirtCRModified

- **KubeVirtCRModified** アラートの runbook を表示 します。

12.7.20. KubeVirtDeprecatedAPIRequested

- **KubeVirtDeprecatedAPIRequested** アラートの runbook を表示 します。

12.7.21. KubeVirtNoAvailableNodesToRunVMs

- **KubeVirtNoAvailableNodesToRunVMs** アラートの runbook を表示 します。

12.7.22. KubevirtVmHighMemoryUsage

- **KubevirtVmHighMemoryUsage** アラートの runbook を表示 します。

12.7.23. KubeVirtVMIExcessiveMigrations

- **KubeVirtVMIExcessiveMigrations** アラートの runbook を表示 します。

12.7.24. LowKVMNodesCount

- **LowKVMNodesCount** アラートの runbook を表示 します。

12.7.25. LowReadyVirtControllersCount

- **LowReadyVirtControllersCount** アラートの runbook を表示 します。

12.7.26. LowReadyVirtOperatorsCount

- **LowReadyVirtOperatorsCount** アラートの runbook を表示 します。

12.7.27. LowVirtAPICount

- **LowVirtAPICount** アラートの runbook を表示 します。

12.7.28. LowVirtControllersCount

- **LowVirtControllersCount** アラートの runbook を表示 します。

12.7.29. LowVirtOperatorCount

- **LowVirtOperatorCount** アラートの runbook を表示 します。

12.7.30. NetworkAddonsConfigNotReady

- **NetworkAddonsConfigNotReady** アラートの runbook を表示 します。

12.7.31. NoLeadingVirtOperator

- **NoLeadingVirtOperator** アラートの runbook を表示 します。

12.7.32. NoReadyVirtController

- **NoReadyVirtController** アラートの runbook を表示 します。

12.7.33. NoReadyVirtOperator

- **NoReadyVirtOperator** アラートの runbook を表示 します。

12.7.34. OrphanedVirtualMachineInstances

- **OrphanedVirtualMachineInstances** アラートの runbook を表示 します。

12.7.35. OutdatedVirtualMachineInstanceWorkloads

- **OutdatedVirtualMachineInstanceWorkloads** アラートの runbook を表示 します。

12.7.36. SingleStackIPv6Unsupported

- **SingleStackIPv6Unsupported** アラートの runbook を表示 します。

12.7.37. SSPCommonTemplatesModificationReverted

- **SSPCommonTemplatesModificationReverted** アラートの runbook を表示 します。

12.7.38. SSPDown

- **SSPDown** アラートの runbook を表示 します。

12.7.39. SSPFailingToReconcile

- **SSPFailingToReconcile** アラートの runbook を表示します。

12.7.40. SSPHighRateRejectedVms

- **SSPHighRateRejectedVms** アラートの runbook を表示 します。

12.7.41. SSPTemplateValidatorDown

- **SSPTemplateValidatorDown** アラートの runbook を表示 します。

12.7.42. UnsupportedHCOModification

- **UnsupportedHCOModification** アラートの runbook を表示 します。

12.7.43. VirtAPIDown

- **VirtAPIDown** アラートの runbook を表示 します。

12.7.44. VirtApiRESTErrorsBurst

- **VirtApiRESTErrorsBurst** アラートの runbook を表示 します。

12.7.45. VirtApiRESTErrorsHigh

- **VirtApiRESTErrorsHigh** アラートの runbook を表示 します。

12.7.46. VirtControllerDown

- **VirtControllerDown** アラートの runbook を表示 します。

12.7.47. VirtControllerRESTErrorsBurst

- **VirtControllerRESTErrorsBurst** アラートの runbook を表示 します。

12.7.48. VirtControllerRESTErrorsHigh

- **VirtControllerRESTErrorsHigh** アラートの runbook を表示 します。

12.7.49. VirtHandlerDaemonSetRolloutFailing

- **VirtHandlerDaemonSetRolloutFailing** アラートの runbook を表示 します。

12.7.50. VirtHandlerRESTErrorsBurst

- **VirtHandlerRESTErrorsBurst** アラートの runbook を表示 します。

12.7.51. VirtHandlerRESTErrorsHigh

- **VirtHandlerRESTErrorsHigh** アラートの runbook を表示 します。

12.7.52. VirtOperatorDown

- **VirtOperatorDown** アラートの runbook を表示 します。

12.7.53. VirtOperatorRESTErrorsBurst

- **VirtOperatorRESTErrorsBurst** アラートの runbook を表示 します。

12.7.54. VirtOperatorRESTErrorsHigh

- **VirtOperatorRESTErrorsHigh** アラートの runbook を表示 します。

12.7.55. VirtualMachineCRCErrors

- **VirtualMachineCRCErrors** アラートの runbook は、アラートの名前が **VMStorageClassWarning** に変更されたため非推奨になりました。
    - **VMStorageClassWarning** アラートの runbook を表示 します。

12.7.56. VMCannotBeEvicted

- **VMCannotBeEvicted** アラートの runbook を表示 します。

12.7.57. VMStorageClassWarning

- **VMStorageClassWarning** アラートの runbook を表示 します。

第13章 SUPPORT

13.1. サポートの概要

以下のツールを使用して、環境に関するデータを収集し、クラスターと仮想マシン (VM) の状態を監視し、OpenShift 仮想化リソースのトラブルシューティングを行うことができます。

13.1.1. Web コンソール

OpenShift Container Platform Web コンソールには、クラスターと OpenShift Virtualization コンポーネントおよびリソースのリソース使用量、アラート、イベント、および傾向が表示されます。

表13.1 監視とトラブルシューティングのための Web コンソールページ

ページ	説明
Overview ページ	クラスターの詳細、ステータス、アラート、インベントリー、およびリソースの使用状況
Virtualization → **Overview** タブ	OpenShift 仮想化のリソース、使用状況、アラート、およびステータス
Virtualization → **Top consumers** タブ	CPU、メモリー、およびストレージの上位コンシューマー
Virtualization → **Migrations** タブ	ライブマイグレーションの進捗
VirtualMachines → **VirtualMachine** → **VirtualMachine details** → **Metrics** タブ	VM リソースの使用状況、ストレージ、ネットワーク、および移行
VirtualMachines → **VirtualMachine** → **VirtualMachine details** → **Events** タブ	VM イベントリスト
VirtualMachines → **VirtualMachine** → **VirtualMachine details** → **Diagnostics** タブ	仮想マシンのステータス条件とボリュームスナップショットのステータス

13.1.2. Red Hat サポート用のデータ収集

Red Hat サポートに サポートケース を送信する場合、デバッグ情報を提供すると役立ちます。次の手順を実行して、デバッグ情報を収集できます。

環境に関するデータの収集

Prometheus および Alertmanager を設定し、OpenShift Container Platform および OpenShift Virtualization の **must-gather** データを収集します。

VM に関するデータの収集

must-gather データとメモリーダンプを VM から収集します。

OpenShift 仮想化のための must-gather ツール

must-gather ツールを設定および使用します。

13.1.3. トラブルシューティング

OpenShift Virtualization コンポーネントと VM のトラブルシューティングを行い、Web コンソールでアラートをトリガーする問題を解決します。

イベント

VM、namespace、およびリソースの重要なライフサイクル情報を表示します。

ログ

OpenShift Virtualization コンポーネントおよび VM のログを表示および設定します。

データボリュームのトラブルシューティング

条件とイベントを分析して、データボリュームのトラブルシューティングを行います。

13.2. RED HAT サポート用のデータ収集

Red Hat サポートに サポートケース を送信する際に、以下のツールを使用して OpenShift Container Platform と OpenShift Virtualization のデバッグ情報を提供すると役立ちます。

must-gather ツール

must-gather ツールは、リソース定義やサービスログなどの診断情報を収集します。

Prometheus

Prometheus は Time Series を使用するデータベースであり、メトリクスのルール評価エンジンです。Prometheus は処理のためにアラートを Alertmanager に送信します。

Alertmanager

Alertmanager サービスは、Prometheus から送信されるアラートを処理します。また、Alertmanager は外部の通知システムにアラートを送信します。

OpenShift Container Platform モニタリングスタックの詳細は、OpenShift Container Platform モニタリングについて を参照してください。

13.2.1. 環境に関するデータの収集

環境に関するデータを収集すると、根本原因の分析および特定に必要な時間が最小限に抑えられます。

前提条件

- Prometheus メトリクスデータの保持期間を最低 7 日間に設定する。
- クラスターの外部で表示および永続化できるように、Alertmanager を設定して、関連するアラートをキャプチャーし、アラート通知を専用のメールボックスに送信 する。
- 影響を受けるノードおよび仮想マシンの正確な数を記録する。

手順

1. クラスターの must-gather データを収集 します。
2. 必要に応じて、Red Hat OpenShift Data Foundation の must-gather データを収集 します。
3. OpenShift Virtualization の必須データを収集 します。
4. クラスターの Prometheus メトリクスを収集 します。

13.2.2. 仮想マシンに関するデータの収集

仮想マシン (VM) の誤動作に関するデータを収集することで、根本原因の分析および特定に必要な時間を最小限に抑えることができます。

前提条件

- Linux VM: 最新の QEMU ゲストエージェントをインストール します。
- Windows 仮想マシン:
  - Windows パッチ更新の詳細を記録します。
  - 最新の VirtIO ドライバーをインストール します。
  - 最新の QEMU ゲストエージェントをインストール します。
  - Remote Desktop Protocol (RDP) が有効になっている場合は、デスクトップビューアー を使用して接続し、接続ソフトウェアに問題があるかどうかを確認します。

手順

1. **/usr/bin/gather** スクリプトを使用して、VM の必須収集データを収集 します。
2. VM を再起動する **前** に、クラッシュした VM のスクリーンショットを収集します。
3. 修復を試みる **前** に、VM からメモリーダンプを収集 します。
4. 誤動作している仮想マシンに共通する要因を記録します。たとえば、仮想マシンには同じホストまたはネットワークがあります。

13.2.3. OpenShift Virtualization の must-gather ツールの使用

OpenShift Virtualization イメージで **must-gather** コマンドを実行することにより、OpenShift Virtualization リソースに関するデータを収集できます。

デフォルトのデータ収集には、次のリソースに関する情報が含まれています。

- 子オブジェクトを含む OpenShift Virtualization Operator namespace
- すべての OpenShift Virtualization カスタムリソース定義 (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

現在、インスタンスタイプの情報はデフォルトでは収集されません。ただし、オプションでコマンドを実行して収集することもできます。

手順

- 以下のコマンドを実行して、OpenShift Virtualization に関するデータを収集します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.2 \
  -- /usr/bin/gather
```

13.2.3.1. must-gather ツールオプション

oc adm must-gather コマンドを実行すると、必要なイメージを明示的に指定しなくても、クラスターにデプロイされているすべての Operators および製品の **must-gather** イメージを収集できます。あるいは、次のオプションに対してスクリプトと環境変数の組み合わせを指定することもできます。

- namespace から詳細な仮想マシン (VM) 情報の収集する
- 特定の仮想マシンに関する詳細情報の収集
- image、image-stream、および image-stream-tags 情報の収集
- **must-gather** ツールが使用する並列プロセスの最大数の制限

13.2.3.1.1. パラメーター

環境変数

互換性のあるスクリプトの環境変数を指定できます。

NS=<namespace_name>

指定した namespace から **virt-launcher** Pod の詳細を含む仮想マシン情報を収集します。**VirtualMachine** および **VirtualMachineInstance** CR データはすべての namespace で収集されます。

VM=<vm_name>

特定の仮想マシンに関する詳細を収集します。このオプションを使用するには、**NS** 環境変数を使用して namespace も指定する必要があります。

PROS=<number_of_processes>

must-gather ツールが使用する並列処理の最大数を変更します。デフォルト値は **5** です。

重要

並列処理が多すぎると、パフォーマンスの問題が発生する可能性があります。並列処理の最大数を増やすことは推奨されません。

スクリプト

各スクリプトは、特定の環境変数の組み合わせとのみ互換性があります。

/usr/bin/gather

デフォルトの **must-gather** スクリプトを使用します。すべての namespace からクラスターデータが収集され、基本的な仮想マシン情報のみが含まれます。このスクリプトは、**PROS** 変数とのみ互換性があります。

/usr/bin/gather --vms_details

OpenShift Virtualization リソースに属する VM ログファイル、VM 定義、コントロールプレーンログ、および namespace を収集します。namespace の指定には、その子オブジェクトが含まれます。namespace または仮想マシンを指定せずにこのパラメーターを使用する場合、**must-gather** ツールはクラスター内のすべての仮想マシンについてこのデータを収集します。このスクリプトはすべての環境変数と互換性がありますが、**VM** 変数を使用する場合は namespace を指定する必要があります。

/usr/bin/gather --images

image、image-stream、および image-stream-tags カスタムリソース情報を収集します。このスクリプトは、**PROS** 変数とのみ互換性があります。

/usr/bin/gather --instancetypes

インスタンスタイプの情報を収集します。この情報は現在、デフォルトでは収集されません。ただし、オプションで収集することもできます。

13.2.3.1.2. 使用方法および例

環境変数はオプションです。スクリプトは、単独で実行することも、1 つ以上の互換性のある環境変数を使用して実行することもできます。

表13.2 互換性のあるパラメーター

スクリプト	互換性のある環境変数
/usr/bin/gather	* **PROS=<number_of_processes>**
/usr/bin/gather --vms_details	* namespace の場合: **NS=<namespace_name>** * 仮想マシンの場合: **VM=<vm_name> NS=<namespace_name>** * **PROS=<number_of_processes>**
/usr/bin/gather --images	* **PROS=<number_of_processes>**

構文

クラスター上のすべての Operator と製品の **must-gather** ログを 1 回のパスで収集するには、次のコマンドを実行します。

```
$ oc adm must-gather --all-images
```

個々の **must-gather** イメージに追加のパラメーターを渡す必要がある場合は、次のコマンドを使用します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.2 \
  -- <environment_variable_1> <environment_variable_2> <script_name>
```

デフォルトのデータ収集の並列プロセス

デフォルトでは、5 つのプロセスを並行して実行します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.2 \
  -- PROS=5 /usr/bin/gather 1
```

1 デフォルトを変更することで、並列プロセスの数を変更できます。

詳細な仮想マシン情報

次のコマンドは、**mynamespace** namespace にある **my-vm** 仮想マシンの詳細な仮想マシン情報を収集します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.2 \
  -- NS=mynamespace VM=my-vm /usr/bin/gather --vms_details 1
```

1 **VM** 環境変数を使用する場合、**NS** 環境変数は必須です。

image、image-stream、および image-stream-tags 情報

以下のコマンドは、クラスターからイメージ、image-stream、および image-stream-tags 情報を収集します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.2 \
  /usr/bin/gather --images
```

インスタンスタイプの情報

次のコマンドは、クラスターからインスタンスタイプ情報を収集します。

```
$ oc adm must-gather \
  --image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel9:v4.16.2 \
  /usr/bin/gather --instancetypes
```

13.3. トラブルシューティング

OpenShift Virtualization は、仮想マシンと仮想化コンポーネントのトラブルシューティングに使用するツールとログを提供します。

OpenShift Virtualization コンポーネントのトラブルシューティングは、Web コンソールで提供されるツールまたは **oc** CLI ツールを使用して実行できます。

13.3.1. Events

OpenShift Container Platform イベント は重要なライフサイクル情報の記録であり、仮想マシン、namespace、リソース問題のモニタリングおよびトラブルシューティングに役立ちます。

- 仮想マシンイベント: Web コンソールで **VirtualMachine details** ページの **Events** タブに移動します。

  namespace イベント

  namespace イベントを表示するには、次のコマンドを実行します。

  ```
  $ oc get events -n <namespace>
  ```

  特定のイベントの詳細は、イベントのリスト を参照してください。

  リソースイベント

  リソースイベントを表示するには、次のコマンドを実行します。

```
$ oc describe <resource> <resource_name>
```

13.3.2. Pod ログ

Web コンソールまたは CLI を使用して、OpenShift Virtualization Pod のログを表示できます。Web コンソールで LokiStack を使用して、集約ログ を表示することもできます。

13.3.2.1. OpenShift Virtualization Pod ログの詳細レベルの設定

HyperConverged カスタムリソース (CR) を編集することで、OpenShift Virtualization Pod ログの詳細レベルを設定できます。

手順

1. 特定のコンポーネントのログの詳細度を設定するには、次のコマンドを実行して、デフォルトのテキストエディターで **HyperConverged** CR を開きます。

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

2. **spec.logVerbosityConfig** スタンザを編集して、1 つ以上のコンポーネントのログレベルを設定します。以下に例を示します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  logVerbosityConfig:
    kubevirt:
      virtAPI: 5 (1)
      virtController: 4
      virtHandler: 3
      virtLauncher: 2
      virtOperator: 6
```

(1) ログの詳細度の値は **1** ~ **9** の範囲の整数である必要があり、数値が大きいほど詳細なログであることを示します。この例では、**virtAPI** コンポーネントのログは、優先度が **5** 以上の場合に公開されます。

3. エディターを保存して終了し、変更を適用します。

13.3.2.2. Web コンソールを使用して virt-launcher Pod のログを表示する

OpenShift Container Platform Web コンソールを使用して、仮想マシンの **virt-launcher** Pod ログを表示できます。

手順

1. **Virtualization** → **VirtualMachines** に移動します。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。

3. **General** タイルで、Pod 名をクリックして **Pod details** ページを開きます。
4. **Logs** タブをクリックして、ログを表示します。

13.3.2.3. CLI を使用した OpenShift Virtualization Pod ログの表示

oc CLI ツールを使用して、OpenShift Virtualization Pod のログを表示できます。

手順

1. 以下のコマンドを実行して、OpenShift Virtualization の namespace 内の Pod のリストを表示します。

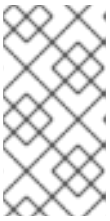
```
$ oc get pods -n openshift-cnv
```

例13.1 出力例

```
NAME                               READY   STATUS    RESTARTS   AGE
disks-images-provider-7gqbc        1/1     Running   0          32m
disks-images-provider-vg4kx        1/1     Running   0          32m
virt-api-57fcc4497b-7qfmc          1/1     Running   0          31m
virt-api-57fcc4497b-tx9nc          1/1     Running   0          31m
virt-controller-76c784655f-7fp6m   1/1     Running   0          30m
virt-controller-76c784655f-f4pbd   1/1     Running   0          30m
virt-handler-2m86x                 1/1     Running   0          30m
virt-handler-9qs6z                 1/1     Running   0          30m
virt-operator-7ccfdbf65f-q5snk     1/1     Running   0          32m
virt-operator-7ccfdbf65f-vllz8     1/1     Running   0          32m
```

2. Pod ログを表示するには、次のコマンドを実行します。

```
$ oc logs -n openshift-cnv <pod_name>
```



注記

Pod の起動に失敗した場合は、**--previous** オプションを使用して、最後の試行からのログを表示できます。

ログ出力をリアルタイムで監視するには、**-f** オプションを使用します。

例13.2 出力例

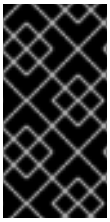
```
{"component":"virt-handler","level":"info","msg":"set verbosity to 2","pos":"virt-handler.go:453","timestamp":"2022-04-17T08:58:37.373695Z"}
{"component":"virt-handler","level":"info","msg":"set verbosity to 2","pos":"virt-handler.go:453","timestamp":"2022-04-17T08:58:37.373726Z"}
{"component":"virt-handler","level":"info","msg":"setting rate limiter to 5 QPS and 10 Burst","pos":"virt-handler.go:462","timestamp":"2022-04-17T08:58:37.373782Z"}
{"component":"virt-handler","level":"info","msg":"CPU features of a minimum baseline CPU model: map[apic:true clflush:true cmov:true cx16:true cx8:true de:true fpu:true fxsr:true lahf_lm:true lm:true mca:true mce:true mmx:true msr:true mtrr:true nx:true pae:true pat:true pge:true pni:true pse:true pse36:true sep:true sse:true sse2:true sse4.1:true
```

```
ssse3:true syscall:true tsc:true]","pos":"cpu_plugin.go:96","timestamp":"2022-04-
17T08:58:37.390221Z"}
{"component":"virt-handler","level":"warning","msg":"host model mode is expected to
contain only one model","pos":"cpu_plugin.go:103","timestamp":"2022-04-
17T08:58:37.390263Z"}
{"component":"virt-handler","level":"info","msg":"node-labeller is
running","pos":"node_labeller.go:94","timestamp":"2022-04-17T08:58:37.391011Z"}
```

13.3.3. ゲストシステムログ

仮想マシンゲストのブートログを表示すると、問題の診断に役立ちます。OpenShift Container Platform Web コンソールまたは **oc** CLI を使用して、ゲストのログへのアクセスを設定し、ログを表示できます。

デフォルトでは無効になっています。仮想マシンでこの設定が明示的に有効または無効になっていない場合、クラスター全体のデフォルト設定が継承されます。



重要

認証情報やその他の個人を特定できる情報 (PII) などの機密情報がシリアルコンソールに書き込まれている場合、他の表示されるすべてのテキストと一緒にそれらもログに記録されます。Red Hat では、機密データの送信にはシリアルコンソールではなく SSH を使用することを推奨しています。

13.3.3.1. Web コンソールを使用して仮想マシンゲストシステムログへのデフォルトアクセスを有効にする

Web コンソールを使用して、仮想マシンゲストシステムログへのデフォルトアクセスを有効にできます。

手順

1. サイドメニューから、**Virtualization → Overview** をクリックします。
2. **Settings** タブをクリックします。
3. **Cluster → Guest management** をクリックします。
4. **Enable guest system log access**をオンに設定します。

13.3.3.2. CLI を使用して仮想マシンゲストシステムログへのデフォルトアクセスを有効にする

HyperConverged カスタムリソース (CR) を編集して、仮想マシンゲストシステムログへのデフォルトアクセスを有効にできます。

手順

1. 以下のコマンドを実行して、デフォルトのエディターで **HyperConverged** CR を開きます。

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

2. **disableSerialConsoleLog** の値を更新します。以下に例を示します。

```
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
spec:
  virtualMachineOptions:
    disableSerialConsoleLog: true (1)
#...
```

(1) 仮想マシン上でシリアルコンソールアクセスをデフォルトで有効にする場合は、**disableSerialConsoleLog** の値を **false** に設定します。

13.3.3.3. Web コンソールを使用して単一仮想マシンのゲストシステムログアクセスを設定する

Web コンソールを使用して、単一仮想マシンの仮想マシンゲストシステムログへのアクセスを設定できます。この設定は、クラスター全体のデフォルト設定よりも優先されます。

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て、**VirtualMachine details** ページを開きます。
3. **Configuration** タブをクリックします。
4. **Guest system log access**をオンまたはオフに設定します。

13.3.3.4. CLI を使用して単一仮想マシンのゲストシステムログアクセスを設定する

VirtualMachine CR を編集することで、単一仮想マシンの仮想マシンゲストシステムログへのアクセスを設定できます。この設定は、クラスター全体のデフォルト設定よりも優先されます。

手順

1. 次のコマンドを実行して、仮想マシンのマニフェストを編集します。

```
$ oc edit vm <vm_name>
```

2. **logSerialConsole** フィールドの値を更新します。以下に例を示します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  template:
    spec:
      domain:
        devices:
          logSerialConsole: true (1)
#...
```

(1) ゲストのシリアルコンソールログへのアクセスを有効にするには、**logSerialConsole** の値を **true** に設定します。

3. 次のコマンドを実行して、新しい設定を仮想マシンに適用します。

```
$ oc apply vm <vm_name>
```

4. オプション: 実行中の仮想マシンを編集した場合は、仮想マシンを再起動して新しい設定を適用します。以下に例を示します。

```
$ virtctl restart <vm_name> -n <namespace>
```

13.3.3.5. Web コンソールを使用してゲストシステムログを表示する

Web コンソールを使用して、仮想マシンゲストのシリアルコンソールログを表示できます。

前提条件

- ゲストシステムログアクセスが有効になっている。

手順

1. サイドメニューから **Virtualization → VirtualMachines** をクリックします。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Diagnostics** タブをクリックします。
4. **Guest system logs** をクリックしてシリアルコンソールをロードします。

13.3.3.6. CLI を使用してゲストシステムログを表示する

oc logs コマンドを実行して、仮想マシンゲストのシリアルコンソールログを表示できます。

前提条件

- ゲストシステムログアクセスが有効になっている。

手順

- 次のコマンドを実行してログを表示します。その場合、**<namespace>** と **<vm_name>** を独自の値に置き換えます。

```
$ oc logs -n <namespace> -l kubevirt.io/domain=<vm_name> --tail=-1 -c guest-console-log
```

13.3.4. ログアグリゲーション

ログを集約してフィルタリングすることで、容易にトラブルシューティングを行えます。

13.3.4.1. LokiStack を使用した OpenShift Virtualization 集約ログの表示

Web コンソールで LokiStack を使用すると、OpenShift Virtualization Pod およびコンテナーの集約ログを表示できます。

前提条件

- LokiStack をデプロイしている。

手順

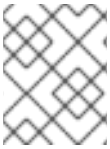
1. Web コンソールで **Observe → Logs** に移動します。
2. ログタイプのリストから、**virt-launcher** Pod ログの場合は **application** を選択し、OpenShift Virtualization コントロールプレーン Pod およびコンテナーの場合は **infrastructure** を選択します。
3. **Show Query** をクリックしてクエリーフィールドを表示します。
4. クエリーフィールドに LogQL クエリーを入力し、**Run Query** をクリックしてフィルタリングされたログを表示します。

13.3.4.2. OpenShift Virtualization LogQL クエリー

Web コンソールの **Observe → Logs** ページで Loki Query Language (LogQL) クエリーを実行することで、OpenShift Virtualization コンポーネントの集約ログを表示およびフィルタリングできます。

デフォルトのログタイプは **infrastructure** です。**virt-launcher** のログタイプは **application** です。

オプション: 行フィルター式を使用して、文字列または正規表現の追加や除外を行えます。



注記

クエリーが多数のログに一致する場合、クエリーがタイムアウトになる可能性があります。

表13.3 OpenShift Virtualization LogQL クエリーの例

コンポーネント	LogQL クエリー
すべて	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"`
cdi-apiserver **cdi-deployment** **cdi-operator**	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="storage"`
hco-operator	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="deployment"`

コンポーネント	LogQL クエリー
kubemacpool	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="network"
virt-api **virt-controller** **virt-handler** **virt-operator**	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="compute"
ssp-operator	{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="schedule"
Container	{log_type=~".+",kubernetes_container_name=~"<container>|<container>"} (1) |json|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" (1) 1つ以上のコンテナーを縦線記号 (|) で区切って指定します。
virt-launcher	このクエリーを実行する前に、ログタイプのリストから **application** を選択する必要があります。 {log_type=~".+", kubernetes_container_name="compute"}|json |!= "custom-ga-command" (1) (1) **|!= "custom-ga-command"** は、**custom-ga-command** の文字列を含む libvirt ログを除外します。(BZ#2177684)

行フィルター式を使用して、文字列や正規表現を追加または除外するようにログ行をフィルタリングできます。

表13.4 行フィルター式

行フィルター式	説明
|= "<string>"	ログ行に文字列が含まれています
!= "<string>"	ログ行に文字列は含まれていません
|~ "<regex>"	ログ行に正規表現が含まれています
!~ "<regex>"	ログ行に正規表現は含まれていません

行フィルター式の例

```
{log_type=~".+"}|json
|kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster"
|= "error" != "timeout"
```

LokiStack および LogQL の関連情報

- ログストレージについて
- LokiStack のデプロイ
- Grafana ドキュメントの LogQL log queries

13.3.5. 一般的なエラーメッセージ

以下のエラーメッセージが OpenShift Virtualization ログに表示される場合があります。

ErrImagePull または ImagePullBackOff

デプロイメント設定が正しくないか、参照されているイメージに問題があることを示します。

13.3.6. データボリュームのトラブルシューティング

DataVolume オブジェクトの **Conditions** および **Events** セクションを確認して、問題を分析および解決できます。

13.3.6.1. データボリュームの条件とイベントについて

コマンドによって生成された **Conditions** および **Events** セクションの出力を調べることで、データボリュームの問題を診断できます。

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

Conditions セクションには、次の **Types** が表示されます。

- **Bound**
- **running**
- **Ready**

Events セクションでは、以下の追加情報を提供します。

- イベントの **Type**
- ロギングの **Reason**
- イベントの **Source**
- 追加の診断情報が含まれる **Message**

oc describe からの出力には常に **Events** が含まれるとは限りません。

Status、**Reason**、または **Message** が変化すると、イベントが生成されます。状態およびイベントはどちらもデータボリュームの状態の変更に対応します。

たとえば、インポート操作中に URL のスペルを誤ると、インポートにより 404 メッセージが生成されます。メッセージの変更により、理由と共にイベントが生成されます。**Conditions** セクションの出力も更新されます。

13.3.6.2. データボリュームの状態とイベントの分析

describe コマンドで生成される **Conditions** セクションおよび **Events** セクションを検査することにより、永続ボリューム要求 (PVC) に関連してデータボリュームの状態を判別します。また、操作がアクティブに実行されているか、または完了しているかどうかを判断します。また、データボリュームのステータスに関する特定の詳細、およびどのように現在の状態になったかに関する情報を提供するメッセージを受信する可能性があります。

状態の組み合わせは多数あります。それぞれは一意のコンテキストで評価される必要があります。

各種の組み合わせの例を以下に示します。

- **Bound** - この例では、正常にバインドされた PVC が表示されます。
  Type は **Bound** であるため、**Status** は **True** になります。PVC がバインドされていない場合、**Status** は **False** になります。

  PVC がバインドされると、PVC がバインドされていることを示すイベントが生成されます。この場合、**Reason** は **Bound** で、**Status** は **True** です。**Message** はデータボリュームを所有する PVC を示します。

  Events セクションの **Message** では、PVC がバインドされている期間 (**Age**) およびどのリソース (**From**) によってバインドされているか、**datavolume-controller** に関する詳細が提供されます。

  出力例

```
Status:
  Conditions:
    Last Heart Beat Time:  2020-07-15T03:58:24Z
    Last Transition Time:  2020-07-15T03:58:24Z
    Message:               PVC win10-rootdisk Bound
    Reason:                Bound
    Status:                True
    Type:                  Bound
...
  Events:
```

```
  Type    Reason     Age    From                Message
  ----    ------     ----   ----                -------
  Normal  Bound      24s    datavolume-controller  PVC example-dv Bound
```

- **Running** - この場合、**Type** が **Running** で、**Status** が **False** であることに注意してください。これは、試行された操作が失敗する原因となったイベントが発生し、Status が **True** から **False** に変化したことを示しています。
  ただし、**Reason** が **Completed** であり、**Message** フィールドには **Import Complete** が表示されることに注意してください。

  Events セクションには、**Reason** および **Message** に失敗した操作に関する追加のトラブルシューティング情報が含まれます。この例では、**Events** セクションの最初の **Warning** に一覧表示される **Message** に、**404** によって接続できないことが示唆されます。

  この情報から、インポート操作が実行されており、データボリュームにアクセスしようとしている他の操作に対して競合が生じることを想定できます。

  出力例

```
Status:
  Conditions:
    Last Heart Beat Time:  2020-07-15T04:31:39Z
    Last Transition Time:  2020-07-15T04:31:39Z
    Message:               Import Complete
    Reason:                Completed
    Status:                False
    Type:                  Running
...
  Events:
    Type     Reason      Age              From                Message
    ----     ------      ----             ----                -------
    Warning  Error       12s (x2 over 14s)  datavolume-controller  Unable to connect
    to http data source: expected status code 200, got 404. Status: 404 Not Found
```

- **Ready**: **Type** が **Ready** であり、**Status** が **True** の場合、以下の例のようにデータボリュームは使用可能な状態になります。データボリュームが使用可能な状態にない場合、**Status** は **False** になります。

  出力例

```
Status:
  Conditions:
    Last Heart Beat Time: 2020-07-15T04:31:39Z
    Last Transition Time:  2020-07-15T04:31:39Z
    Status:                True
    Type:                  Ready
```

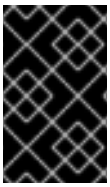
第14章 バックアップおよび復元

14.1. 仮想マシンスナップショットを使用したバックアップと復元

スナップショットを使用して、仮想マシンをバックアップおよび復元できます。スナップショットは、次のストレージプロバイダーによってサポートされています。

- 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。

最も整合性の高いオンライン (実行状態) 仮想マシンのスナップショットを作成する際、QEMU ゲストエージェントがオペレーティングシステムに含まれていない場合はインストールします。QEMU ゲストエージェントは、デフォルトの Red Hat テンプレートに含まれています。

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。

14.1.1. スナップショット

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。

次のスナップショットアクションを実行できます。

- 新規スナップショットの作成
- スナップショットから仮想マシンのコピーを作成する

- 特定の仮想マシンに割り当てられているすべてのスナップショットのリスト表示
- スナップショットからの仮想マシンの復元
- 既存の仮想マシンスナップショットの削除

仮想マシンスナップショットコントローラーとカスタムリソース

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カスタムリソース定義 (CRD) として定義される 3 つの新しい API オブジェクトが導入されています。

- **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。

14.1.2. スナップショットの作成

OpenShift Container Platform Web コンソールまたはコマンドラインを使用して、仮想マシンのスナップショットを作成できます。

14.1.2.1. Web コンソールを使用したスナップショットを作成する

OpenShift Container Platform Web コンソールを使用して、仮想マシンのスナップショットを作成できます。

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が含まれます。

- データボリュームまたは永続ボリュームの要求のいずれか
- 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. Web コンソールで **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. 仮想マシンが実行中の場合は、オプションメニュー  をクリックし、**Stop** を選択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. スナップショット名を入力します。

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含めることができないディスクがあり、続行する場合は、**I am aware of this warning and wish to proceed** を選択します。

8. **Save** をクリックします。

14.1.2.2. コマンドラインを使用した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。

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. 次の例のように、YAML ファイルを作成して、新しい **VirtualMachineSnapshot** の名前と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。

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineSnapshot
metadata:
  name: <snapshot_name>
spec:
  source:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
```

2. **VirtualMachineSnapshot** オブジェクトを作成します。

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

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。

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 <vm_name> <snapshot_name> --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、**VirtualMachineSnapshotContent** にバインドされていることと、**readyToUse** フラグが **true** に設定されていることを確認します。

```
$ oc describe vmsnapshot <snapshot_name>
```

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。

14.1.3. 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の検証

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

- オンライン仮想マシンスナップショットを作成しようとしている必要があります。

手順

1. 次のいずれかの操作を実行して、スナップショット指示からの出力を表示します。
   - コマンドラインを使用して、**VirtualMachineSnapshot** オブジェクト YAML の **status** スタンザのインジケーター出力を表示します。
   - Web コンソールの **Snapshot details** 画面で、**VirtualMachineSnapshot → Status** をクリックします。
2. **status.indications** パラメーターの値を表示して、オンラインの仮想マシンスナップショットのステータスを確認します。
   - **Online** は、オンラインスナップショットの作成中に仮想マシンが実行されていたことを示します。
   - **GuestAgent** は、オンラインスナップショットの作成中に QEMU ゲストエージェントが実行されていたことを示します。

- **NoGuestAgent** は、オンラインスナップショットの作成中に QEMU ゲストエージェントが実行されていなかったことを示します。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。

14.1.4. スナップショットからの仮想マシンの復元

OpenShift Container Platform Web コンソールまたはコマンドラインを使用して、スナップショットから仮想マシンを復元できます。

14.1.4.1. Web コンソールを使用したスナップショットからの仮想マシンの復元

OpenShift Container Platform 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仮想マシンを復元できます。

手順

1. Web コンソールで **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. 仮想マシンが実行中の場合は、オプションメニュー  をクリックし、**Stop** を選択して電源を切ります。
4. **Snapshots** タブをクリックして、仮想マシンに関連付けられたスナップショットのリストを表示します。
5. スナップショットを選択して、**Snapshot Details** 画面を開きます。
6. オプションメニュー  をクリックし、**Restore VirtualMachine from snapshot** を選択します。
7. **Restore** をクリックします。

14.1.4.2. コマンドラインを使用したスナップショットからの仮想マシンの復元

コマンドラインを使用して、既存の仮想マシン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

- 復元する仮想マシンの電源を切ります。

手順

1. 次の例のように、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。

```
apiVersion: snapshot.kubevirt.io/v1alpha1
kind: VirtualMachineRestore
```

```
metadata:
  name: <vm_restore>
spec:
  target:
    apiGroup: kubevirt.io
    kind: VirtualMachine
    name: <vm_name>
  virtualMachineSnapshotName: <snapshot_name>
```

2. **VirtualMachineRestore** オブジェクトを作成します。

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

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。

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、および **complete** フラグが **true** に設定されていることを確認します。

```
$ oc get vmrestore <vm_restore>
```

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。

14.1.5. スナップショットの削除

OpenShift Container Platform Web コンソールまたはコマンドラインを使用して、仮想マシンのスナップショットを削除できます。

14.1.5.1. Web コンソールを使用してスナップショットを削除する

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. Web コンソールで **Virtualization → VirtualMachines** に移動します。
2. 仮想マシンを選択して、**VirtualMachine details** ページを開きます。
3. **Snapshots** タブをクリックして、仮想マシンに関連付けられたスナップショットのリストを表示します。
4. スナップショットの横にあるオプションメニュー  をクリックし、**Delete snapshot** を選択します。
5. **Delete** をクリックします。

14.1.5.2. 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。

```
$ oc delete vmsnapshot <snapshot_name>
```

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。

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
```

14.1.6. 関連情報

- CSI ボリュームスナップショット

14.2. 仮想マシンのバックアップと復元

重要

Red Hat は、OADP 1.3.x 以降で OpenShift Virtualization 4.14 以降を使用することをサポートしています。

OADP 1.3.0 より前のバージョンでは、OpenShift Virtualization のバックアップと復元ではサポートされていません。

OpenShift API for Data Protection 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。

OADP Operator をインストールし、バックアップの場所を設定することで、OpenShift Virtualization を使用した OpenShift API for Data Protection (OADP) をインストールできます。その後、Data Protection Application をインストールできます。

注記

OpenShift Virtualization を使用した OpenShift API for Data Protection は、バックアップおよび復元のストレージオプションとして次のものをサポートしています。

- Container Storage Interface (CSI) バックアップ
- DataMover による Container Storage Interface (CSI) バックアップ

次のストレージオプションは対象外です。

- ファイルシステムのバックアップと復元
- ボリュームスナップショットのバックアップと復元

詳細は、ファイルシステムバックアップを使用してアプリケーションをバックアップする: Kopia または Restic を参照してください。

制限されたネットワーク環境に OADP Operator をインストールするには、最初にデフォルトの OperatorHub ソースを無効にして、Operator カタログをミラーリングする必要があります。

詳細は、ネットワークが制限された環境での Operator Lifecycle Manager の使用 を参照してください。

14.2.1. OpenShift Virtualization を使用した OADP のインストールと設定

クラスター管理者は、OADP Operator をインストールして OADP をインストールします。

最新バージョンの OADP Operator は、Velero 1.14 をインストールします。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. ストレージプロバイダーの指示に従って、OADP Operator をインストールします。
2. **kubevirt** および **openshift** OADP プラグインを使用して Data Protection Application (DPA) 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。

警告

Red Hat のサポート対象は、次のオプションに限られています。

- CSI バックアップ
- DataMover による CSI バックアップ

Restore CR を作成して **Backup** CR を復元します。

関連情報

- OADP プラグイン
- **Backup** カスタムリソース (CR)
- **Restore** CR
- ネットワークが制限された環境での Operator Lifecycle Manager の使用

14.2.2. Data Protection Application 1.3 のインストール

DataProtectionApplication API のインスタンスを作成して、Data Protection Application (DPA) をインストールします。

前提条件

- OADP Operator をインストールする。
- オブジェクトストレージをバックアップロケーションとして設定する必要がある。
- スナップショットを使用して PV をバックアップする場合、クラウドプロバイダーはネイティブスナップショット API または Container Storage Interface (CSI) スナップショットのいずれかをサポートする必要がある。
- バックアップとスナップショットの場所で同じ認証情報を使用する場合は、デフォルトの名前である **cloud-credentials** を使用して **Secret** を作成する必要がある。
- バックアップとスナップショットの場所で異なる認証情報を使用する場合は、以下のように 2 つの **Secrets** を作成する必要がある。
  - バックアップの場所用のカスタム名を持つ **Secret**。この **Secret** を **DataProtectionApplication** CR に追加します。
  - スナップショットの場所用の別のカスタム名を持つ **Secret**。この **Secret** を **DataProtectionApplication** CR に追加します。

注記

インストール中にバックアップまたはスナップショットの場所を指定したくない場合は、空の **credentials-velero** ファイルを使用してデフォルトの **Secret** を作成できます。デフォルトの **Secret** がない場合、インストールは失敗します。

手順

1. **Operators → Installed Operators** をクリックして、OADP Operator を選択します。
2. **Provided APIs** で、**DataProtectionApplication** ボックスの **Create instance** をクリックします。
3. **YAML View** をクリックして、**DataProtectionApplication** マニフェストのパラメーターを更新します。

```
apiVersion: oadp.openshift.io/v1alpha1
```

```
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
  namespace: openshift-adp (1)
spec:
  configuration:
    velero:
      defaultPlugins:
        - kubevirt (2)
        - gcp (3)
        - csi (4)
        - openshift (5)
      resourceTimeout: 10m (6)
    nodeAgent: (7)
      enable: true (8)
      uploaderType: kopia (9)
      podConfig:
        nodeSelector: <node_selector> (10)
  backupLocations:
    - velero:
        provider: gcp (11)
        default: true
        credential:
          key: cloud
          name: <default_secret> (12)
        objectStorage:
          bucket: <bucket_name> (13)
          prefix: <prefix> (14)
```

(1) OADP のデフォルトの namespace は **openshift-adp** です。namespace は変数であり、設定可能です。

(2) **kubevirt** プラグインは OpenShift Virtualization に必須です。

(3) バックアッププロバイダーのプラグインがある場合には、それを指定します (例: **gcp**)。

(4) CSI スナップショットを使用して PV をバックアップするには、**csi** プラグインが必須です。**csi** プラグインは、Velero CSI ベータスナップショット API を使用します。スナップショットの場所を設定する必要はありません。

(5) **openshift** プラグインは必須です。

(6) Velero CRD の可用性、volumeSnapshot の削除、バックアップリポジトリーの可用性など、タイムアウトが発生するまでに複数の Velero リソースを待機する時間を分単位で指定します。デフォルトは 10m です。

(7) 管理要求をサーバーにルーティングする管理エージェント。

(8) **nodeAgent** を有効にしてファイルシステムバックアップを実行する場合は、この値を **true** に設定します。

(9) 組み込み DataMover を使用するには、アップローダーとして **kopia** と入力します。**nodeAgent** はデーモンセットをデプロイします。これは、**nodeAgent** Pod が各ワーキングノード上で実行されることを意味します。ファイルシステムバックアップを設定するには、**spec.defaultVolumesToFsBackup: true** を **Backup** CR に追加します。

10 Kopia が利用可能なノードを指定します。デフォルトでは、Kopia はすべてのノードで実行されます。

11 バックアッププロバイダーを指定します。

12 バックアッププロバイダーにデフォルトのプラグインを使用する場合は、**Secret** の正しいデフォルト名を指定します (例: **cloud-credentials-gcp**)。カスタム名を指定すると、そのカスタム名がバックアップの場所に使用されます。**Secret** 名を指定しない場合は、デフォルトの名前が使用されます。

13 バックアップの保存場所としてバケットを指定します。バケットが Velero バックアップ専用のバケットでない場合は、接頭辞を指定する必要があります。

14 バケットが複数の目的で使用される場合は、Velero バックアップの接頭辞を指定します (例: **velero**)。

4. **Create** をクリックします。

検証

1. 次のコマンドを実行して OpenShift API for Data Protection (OADP) リソースを表示し、インストールを検証します。

```
$ oc get all -n openshift-adp
```

出力例

```
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/oadp-operator-controller-manager-67d9494d47-6l8z8   2/2     Running   0          2m8s
pod/node-agent-9cq4q                                 1/1     Running   0          94s
pod/node-agent-m4lts                                 1/1     Running   0          94s
pod/node-agent-pv4kr                                 1/1     Running   0          95s
pod/velero-588db7f655-n842v                          1/1     Running   0          95s

NAME                                                     TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)    AGE
service/oadp-operator-controller-manager-metrics-service   ClusterIP   172.30.70.140
<none>        8443/TCP   2m8s
service/openshift-adp-velero-metrics-svc                 ClusterIP   172.30.10.0      <none>
8085/TCP   8h

NAME                        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
daemonset.apps/node-agent   3         3         3       3            3           <none>          96s

NAME                                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/oadp-operator-controller-manager   1/1     1            1           2m9s
deployment.apps/velero                             1/1     1            1           96s

NAME                                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/oadp-operator-controller-manager-67d9494d47   1         1         1       2m9s
replicaset.apps/velero-588db7f655                          1         1         1       96s
```

2. 次のコマンドを実行して、**DataProtectionApplication** (DPA) が調整されていることを確認します。

```
$ oc get dpa dpa-sample -n openshift-adp -o jsonpath='{.status}'
```

出力例

```
{"conditions":[{"lastTransitionTime":"2023-10-27T01:23:57Z","message":"Reconcile
complete","reason":"Complete","status":"True","type":"Reconciled"}]}
```

3. **type** が **Reconciled** に設定されていることを確認します。

4. 次のコマンドを実行して、バックアップの保存場所を確認し、**PHASE** が **Available** であることを確認します。

```
$ oc get backupStorageLocation -n openshift-adp
```

出力例

```
NAME          PHASE       LAST VALIDATED   AGE     DEFAULT
dpa-sample-1  Available   1s               3d16h   true
```

14.3. 障害復旧

OpenShift Virtualization は、サイト停止後に環境を確実に復元するために、障害復旧 (DR) ソリューションの使用をサポートします。これらのメソッドを使用するには、OpenShift Virtualization のデプロイメントを事前に計画する必要があります。

14.3.1. 障害復旧方法について

障害復旧 (DR) の概念、アーキテクチャー、計画上の考慮事項の概要については、Red Hat ナレッジベースの Red Hat OpenShift Virtualization disaster recovery guide を参照してください。

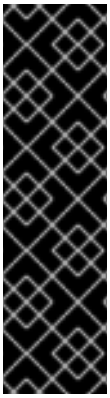
OpenShift Virtualization には、Metropolitan Disaster Recovery (Metro-DR) および Regional-DR という 2 つの主要な DR メソッドがあります。

14.3.1.1. Metro-DR

Metro-DR は同期レプリケーションを使用します。プライマリーサイトとセカンダリーサイトの両方のストレージに書き込むため、サイト間でデータが常に同期されます。ストレージプロバイダーは正常に同期されることを確認する責任があるため、その環境はストレージプロバイダーのスループットと遅延の要件を満たしている必要があります。

14.3.1.2. Regional-DR

Regional-DR は非同期レプリケーションを使用します。プライマリーサイトのデータは、定期的にセカンダリーサイトと同期されます。このタイプのレプリケーションでは、プライマリーサイトとセカンダリーサイト間の接続遅延が大きくなる場合があります。



重要

Regional-DR はテクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

14.3.2. 障害復旧のためのアプリケーションの定義

Red Hat Advanced Cluster Management (RHACM) が管理または検出する仮想マシンを使用して、障害復旧用のアプリケーションを定義します。

14.3.2.1. RHACM が管理する仮想マシンを定義する際のベストプラクティス

仮想マシンを含む RHACM 管理対象アプリケーションは、GitOps ワークフローを使用し、RHACM アプリケーションまたは **ApplicationSet** を作成して作成する必要があります。

RHACM が管理する仮想マシンを定義するときに、顧客体験と成功の可能性を高めるために実行できるアクションがいくつかあります。

PVC とポピュレーターを使用して仮想マシンのストレージを定義する
データボリュームは暗黙的に永続ボリューム要求 (PVC) を作成するため、データボリュームとデータボリュームテンプレートを使用した仮想マシンは GitOps モデルに調和しません。

仮想マシンディスクのポリューション元を選択するときは、インポート方法を使用します。
複製された PVC を使用する仮想マシンを保護できない Regional-DR の制限を回避するには、インポートメソッドを使用します。

インポート方法を使用するには、ソフトウェアカタログから RHEL イメージを選択します。Red Hat では、一貫した結果を得るために、フローティングタグではなく特定のバージョンのイメージを使用することを推奨しています。KubeVirt コミュニティーは、Quay リポジトリーで他のオペレーティングシステム用のコンテナーディスクを管理します。

pullMethod: node を使用する
レジストリーソースからデータボリュームを作成するときに、Pod **pullMethod: node** を使用して、Red Hat レジストリーからコンテナーイメージをプルするために必要な OpenShift Container Platform プルシークレットを活用します。

14.3.2.2. RHACM で検出された仮想マシンを定義する際のベストプラクティス

RHACM 管理対象アプリケーションではないクラスター内の任意の仮想マシンを、RHACM 検出アプリケーションとして設定できます。これには、Migration Toolkit for Virtualization (MTV) を使用してインポートされた仮想マシン、OpenShift Virtualization Web コンソールを使用して作成された仮想マシン、または CLI などの他の手段で作成された仮想マシンが含まれます。

RHACM で検出された仮想マシンを定義するときに、顧客体験と成功の可能性を高めるために実行できるアクションがいくつかあります。

MTV、OpenShift Virtualization Web コンソール、またはカスタム仮想マシンを使用するときに仮想マシンを保護する

現在、自動ラベル付けは利用できないため、MTV、OpenShift Virtualization Web コンソール、またはカスタム仮想マシンを使用する場合、アプリケーション所有者は仮想マシンアプリケーションのコンポーネントに手動でラベルを付ける必要があります。

仮想マシンを作成したら、仮想マシンに関連付けられている次のリソースに共通ラベル (**VirtualMachine**、**DataVolume**、**PersistentVolumeClaim**、**Service**、**Route**、**Secret**、および **ConfigMap**) を適用します。OpenShift Virtualization は仮想マシンインスタンス (VMI) または Pod を自動的に作成および管理するため、これらにラベルを付けないでください。

仮想マシンに VirtualMachine オブジェクト以外のものを含める
動作中の仮想マシンには通常、データボリューム、永続ボリューム要求 (PVC)、サービス、ルート、シークレット、**ConfigMap** オブジェクト、**VirtualMachineSnapshot** オブジェクトも含まれます。

仮想マシンをより大きな論理アプリケーションの一部として含める
これには、他の Pod ベースのワークロードと仮想マシンが含まれます。

14.3.3. 障害復旧シナリオ中の仮想マシンの動作

仮想マシンは通常、再配置とフェイルオーバーの両方の障害復旧フロー中に Pod ベースのワークロードと同様に動作します。

再配置
プライマリー環境にまだアクセス可能な場合は、再配置を使用してアプリケーションをプライマリー環境からセカンダリー環境に移動します。再配置中、仮想マシンは正常に終了し、複製されていないデータはセカンダリー環境に同期され、仮想マシンはセカンダリー環境で起動します。

正常に終了するため、このシナリオではデータが失われることはありません。したがって、仮想マシンオペレーティングシステムはクラッシュ回復を実行する必要がありません。

Failover
プライマリー環境で重大な障害が発生し、再配置を使用してワークロードをセカンダリー環境に移動することが非現実的または不可能な場合は、フェイルオーバーを使用します。フェイルオーバーを実行すると、ストレージはプライマリー環境からフェンスされ、仮想マシンディスクへの I/O が突然停止になり、レプリケートされたデータを使用してセカンダリー環境で仮想マシンが再起動になります。

フェイルオーバーによるデータ損失が予想されます。損失の程度は、同期レプリケーションを使用する Metro-DR を使用するか、非同期レプリケーションを使用する Regional-DR を使用するかによって異なります。Regional-DR はスナップショットベースのレプリケーション間隔を使用するため、データ損失のウィンドウはレプリケーション間隔の長さに比例します。仮想マシンが再起動すると、オペレーティングシステムがクラッシュ回復を実行する可能性があります。

14.3.4. Red Hat OpenShift Data Foundation の Metro-DR

OpenShift Virtualization は、OpenShift Data Foundation の Metro-DR ソリューション をサポートしています。これにより、プライマリーサイトとセカンダリーサイトにインストールされているマネージド OpenShift Virtualization クラスター間で双方向の同期データレプリケーションが可能になります。このソリューションは、Red Hat Advanced Cluster Management (RHACM)、Red Hat Ceph Storage、および OpenShift Data Foundation のコンポーネントを組み合わせたものです。

サイトで障害が発生した場合、このソリューションを使用してアプリケーションをプライマリーサイトからセカンダリーサイトにフェイルオーバーし、障害が発生したサイトを復元した後にプライマリーサイトにアプリケーションを再配置します。

この同期ソリューションは、遅延が 10 ミリ秒以下の大都市圏の遠隔データセンターでのみ利用できます。

OpenShift Virtualization を使用した OpenShift Data Foundation の Metro-DR ソリューションの使用に関する詳細は、Red Hat ナレッジベース または IBM の OpenShift Data Foundation Metro-DR ドキュメントを参照してください。

関連情報

- OpenShift ワークロード用の OpenShift Data Foundation Disaster Recovery の設定

関連情報

- Red Hat Kubernetes 2.10 向け高度なクラスター管理