OpenShift Dedicated 4

仮想化

OpenShift Virtualization のインストールと使用方法。

Last Updated: 2024-04-25

OpenShift Dedicated 4 仮想化

OpenShift Virtualization のインストールと使用方法。

法律上の通知

概要

このドキュメントでは、AWS の OpenShift Service で OpenShift Virtualization を使用する方法について説明します。

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 OVN-Kubernetes または OpenShift SDN とともに使用できます。

Compliance Operator をインストールし、**ocp4-moderate** および **ocp4-moderate-node** を使用してスキャンを実行することにより、OpenShift Virtualization クラスターのコンプライアンスの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 4.15 は OpenShift Dedicated 4 クラスターで使用するためにサポートされます。Open Shift Virtualization の最新の z-stream リリースを使用するには、最初に Open Shift Container Platform の最新バージョンにアップグレードする必要があります。

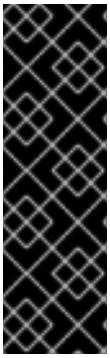
1.1.2. 仮想マシンディスクのボリュームとアクセスモードについて

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。



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

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

1.1.3. 関連情報

- OpenShift Dedicated ストレージの一般用語集
- 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
- チューニングおよびスケーリングガイド

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について説明します。

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

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	Resources	動詞
cdi.kubevirt.io:admin	**datavolumes、uploadtokenrequests**	* (すべて)
	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	Resources	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
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	Resources	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	*	*
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	Resources	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	Resources	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**

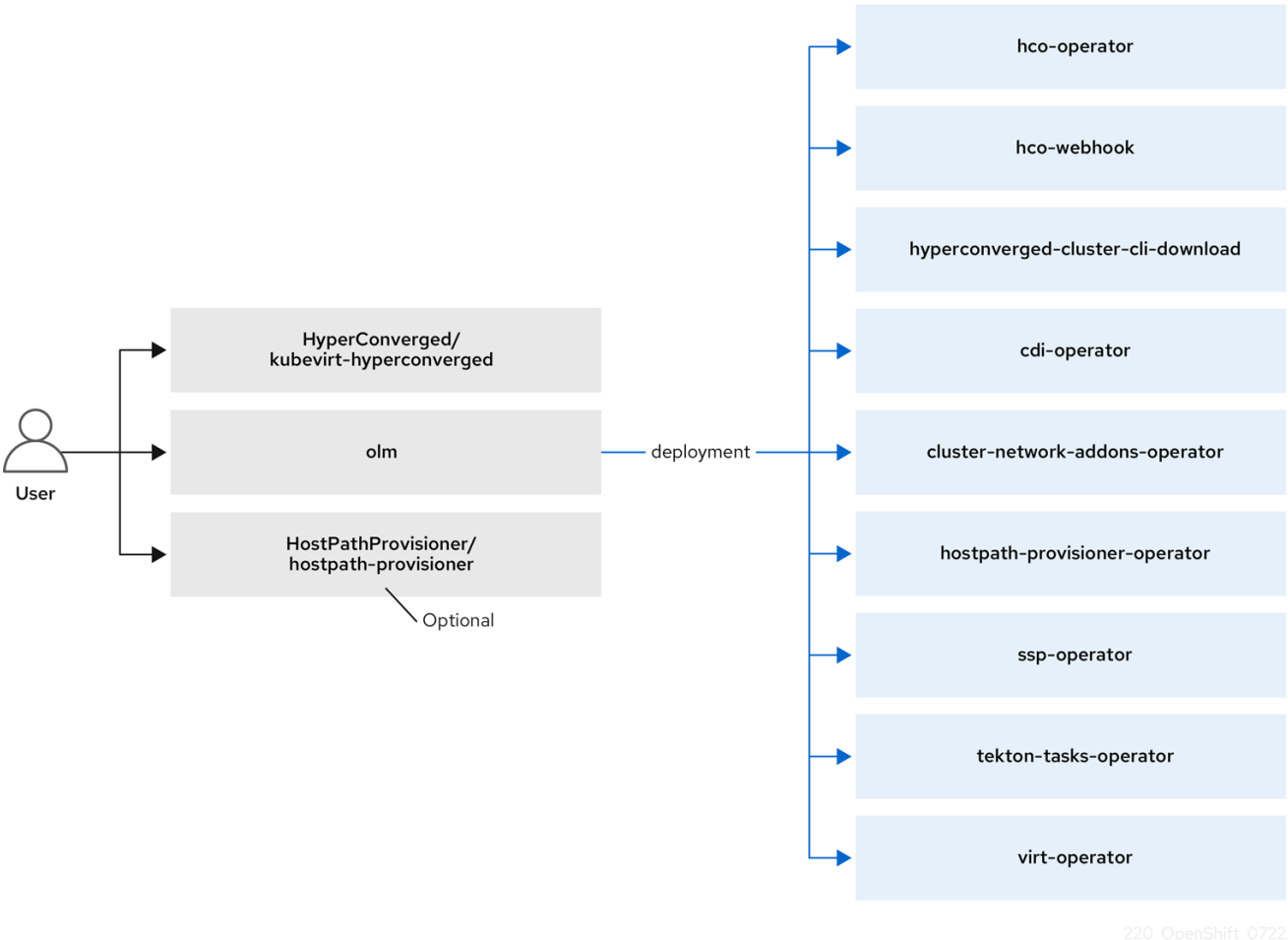
- スケーリング: **ssp-operator**
- テンプレート作成: **tekton-tasks-operator**

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

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

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

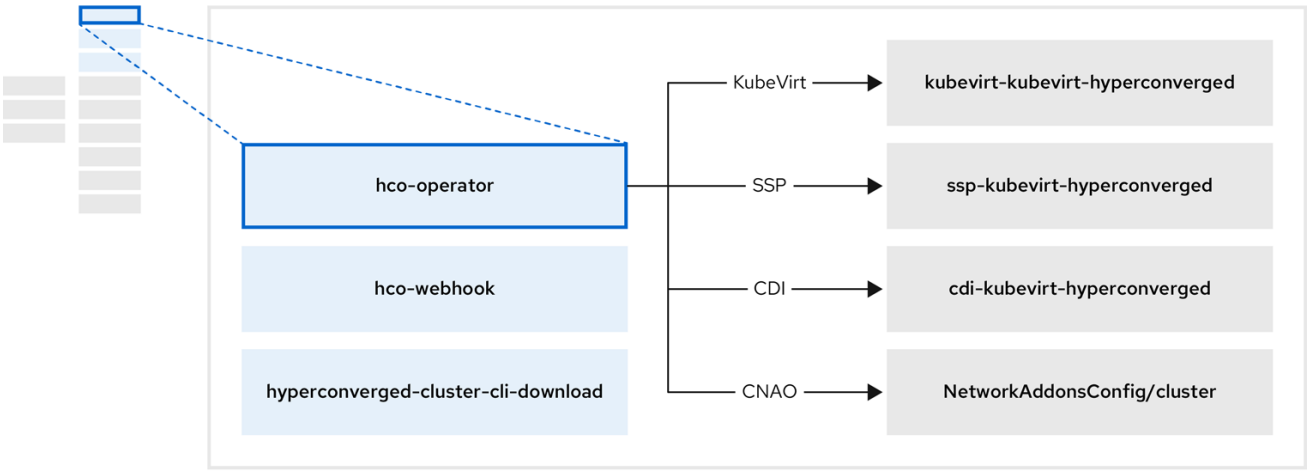


表1.7 HyperConverged Operator コンポーネント

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。

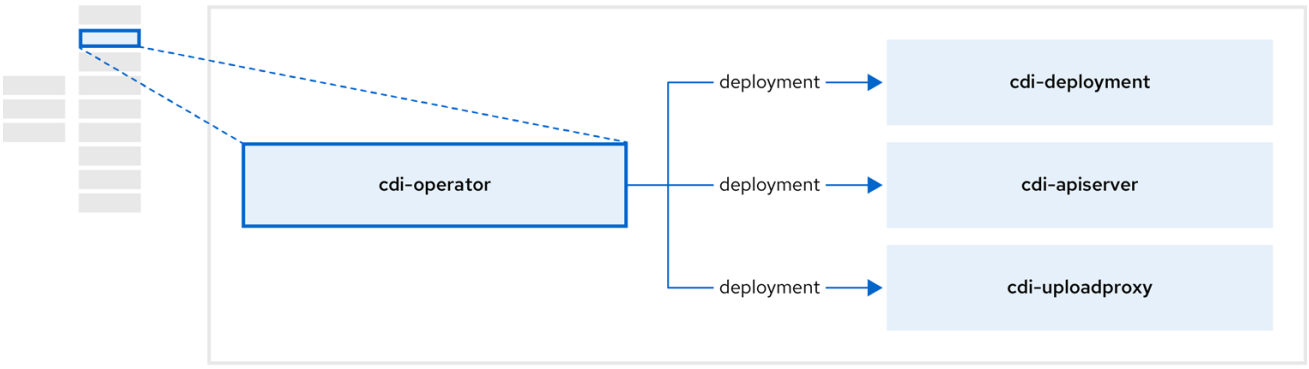


表1.8 CDI Operator コンポーネント

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。
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。

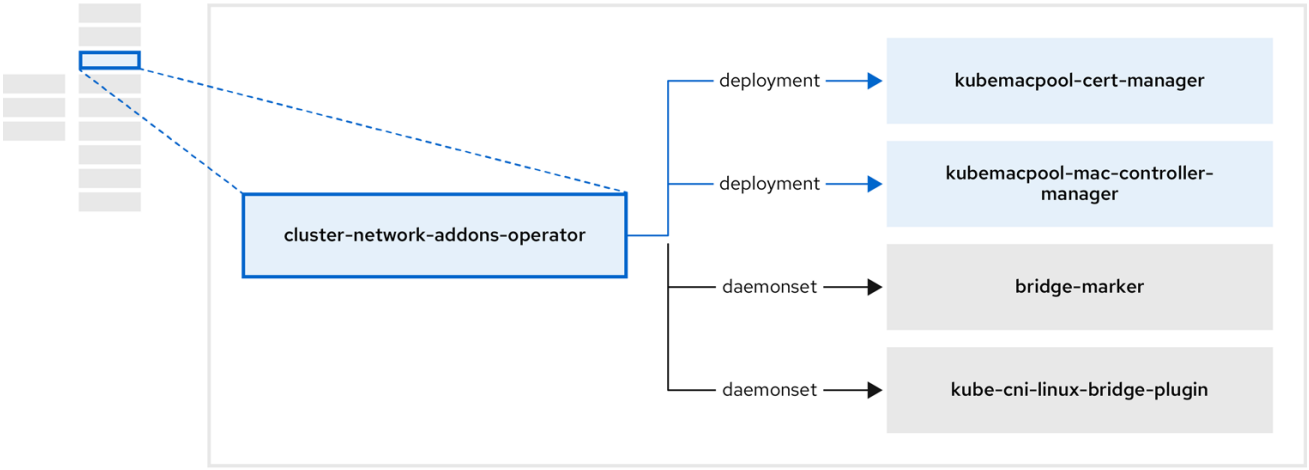


表1.9 Cluster Network Addons Operator コンポーネント

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。

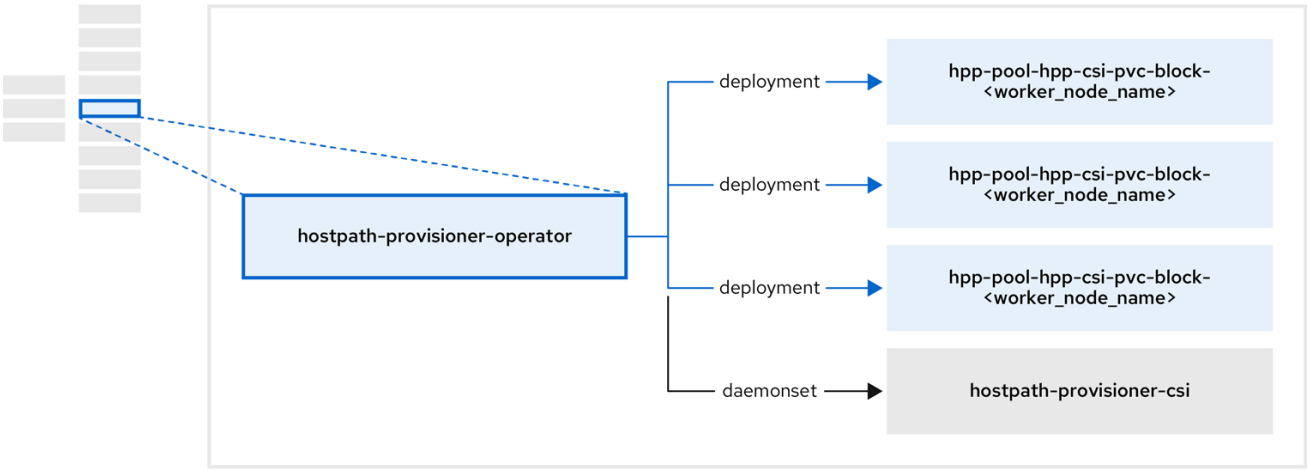


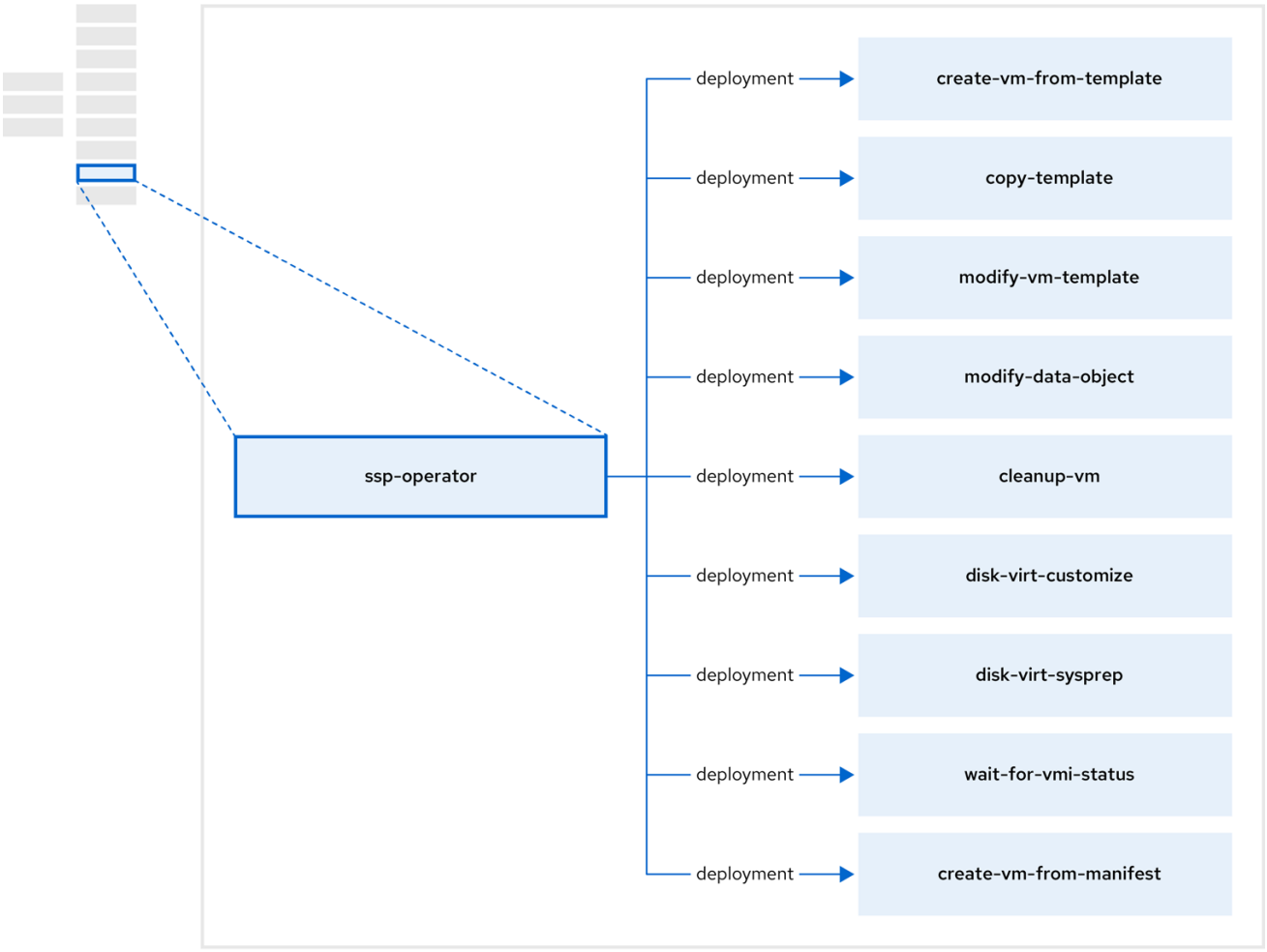
表1.10 HPP Operator コンポーネント

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。

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

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。



467_OpenShift_1023

表1.11 SSP Operator コンポーネント

コンポーネント	説明
deployment/create-vm-from-template	テンプレートから仮想マシンを作成します。
deployment/copy-template	仮想マシンテンプレートをコピーします。
deployment/modify-vm-template	仮想マシンテンプレートを作成または削除します。

コンポーネント	説明
deployment/modify-data-object	データボリュームまたはデータソースを作成または削除します。
deployment/cleanup-vm	仮想マシンでスクリプトまたはコマンドを実行し、後で仮想マシンを停止または削除します。
deployment/disk-virt-customize	**virt-customize** を使用して、ターゲット永続ボリューム要求 (PVC) で **customize** スクリプトを実行します。
deployment/disk-virt-sysprep	**virt-sysprep** を使用して、ターゲット PVC で **sysprep** スクリプトを実行します。
deployment/wait-for-vmi-status	特定の仮想マシンインスタンス (VMI) ステータスを待機し、そのステータスに応じて失敗または成功します。
deployment/create-vm-from-manifest	マニフェストから仮想マシンを作成します。

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。

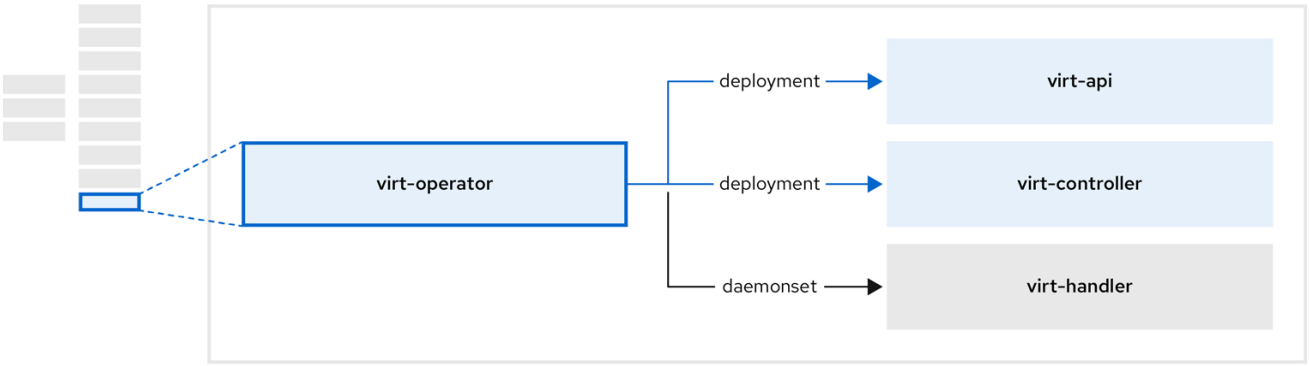


表1.12 virt-operator コンポーネント

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。

コンポーネント	説明
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 スタートガイド

2.1. 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。

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

OpenShift Dedicated 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 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
- 仮想マシンのノードの指定
- **Virtctl** コマンド

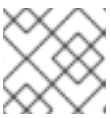
2.1.2. 仮想マシンの作成と管理

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。

- 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。



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。

2.1.3. 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。
- 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

2.2. VIRTCTL および LIBGUESTFS CLI ツールの使用

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

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。

2.2.1. 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。

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

OpenShift Dedicated 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てから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解凍します。

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解凍します。

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
```

2.2.1.2. 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 Dedicated サブスクリプションを持っている必要があります。

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-4.15-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
```

2.2.2. 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。

2.2.2.1. 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2.1 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。

2.2.2.2. 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2.2 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。

2.2.2.3. 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2.3 仮想マシン管理コマンド

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

2.2.2.4. 仮想マシン接続コマンド

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

表2.4 仮想マシン接続コマンド

コマンド	説明
virtctl console <vm_name>	仮想マシンのシリアルコンソールに接続します。
virtctl expose vm <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。
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。

2.2.2.5. 仮想マシンエクスポートコマンド

virtctl vmexport コマンドを使用して、仮想マシン、仮想マシンスナップショット、または永続ボリューム要求 (PVC) からエクスポートされたボリュームを作成、ダウンロード、または削除できます。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2.5 仮想マシンエクスポートコマンド

コマンド	説明

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

コマンド	説明
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。

2.2.2.6. 仮想マシンメモリーダンプコマンド

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

前提条件

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

```
$ oc patch 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2.6 仮想マシンメモリーダンプコマンド

コマンド	説明

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

2.2.2.7. ホットプラグおよびホットアンプラグコマンド

virtctl を使用して、実行中の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2.7 ホットプラグおよびホットアンプラグコマンド

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

コマンド	説明
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。

2.2.2.8. イメージアップロードコマンド

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

表2.8 イメージアップロードコマンド

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

2.2.3. 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。

2.2.3.1. 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。
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。

フラグオプション	説明
--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。

2.3. WEB コンソールの概要

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

表2.9 Virtualization ページ

ページ	説明
Overview ページ	OpenShift Virtualization 環境を管理および監視します。
Catalog ページ	テンプレートのカタログから仮想マシンを作成します。

ページ	説明
VirtualMachines ページ	仮想マシンを作成および管理します。
Templates ページ	テンプレートを作成および管理します。
InstanceTypes ページ	仮想マシンのインスタンスタイプを作成および管理します。
Preferences ページ	仮想マシンの設定を作成および管理します。
Bootable volumes ページ	ブート可能ボリュームのデータソースを作成および管理します。
MigrationPolicies ページ	ワークロードの移行ポリシーを作成および管理します。
Checkups ページ	仮想マシンのネットワーク遅延とストレージのチェックを実行します。

表2.10 キー

アイコン	説明
	Edit アイコン
	Link アイコン
	Start VM アイコン
	Stop VM アイコン
	Restart VM アイコン
	Pause VM アイコン
	Unpause VM アイコン

2.3.1. Overview ページ

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

例2.1 Overview ページ

要素	説明
virtctl のダウンロード	リソースを管理するには、**virtctl** コマンドラインツールをダウンロードします。
Overview タブ	リソース、使用率、アラート、およびステータス。
Top consumers タブ	CPU、メモリー、およびストレージリソースのトップコンシューマー。
Migrations タブ	ライブマイグレーションのステータス。
Settings タブ	**Settings** タブには、**Cluster** タブ、**User** タブ、**Preview features** タブがあります。
Settings → Cluster タブ	OpenShift Virtualization のバージョン、更新ステータス、ライブマイグレーション、テンプレートプロジェクト、ロードバランサーサービス、ゲスト管理、リソース管理、SCSI 永続予約の設定。
Settings → User タブ	公開 SSH キー、ユーザー権限、ウェルカム情報の設定。
Settings → **Preview features**	Web コンソールで プレビュー機能 を有効にできます。このタブの機能は頻繁に変更されます。 プレビュー機能はデフォルトで無効になっているため、運用環境では有効にしないでください。

2.3.1.1. Overview タブ

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

例2.2 Overview タブ

要素	説明
Getting started resources カード	• **Quick Starts** タイル: 仮想マシンの作成、インポート、および実行方法の段階的な手順とタスクを確認できます。 • **Feature highlights** タイル: 主要な仮想化機能に関する最新情報が表示されます。 • **Related operators** タイル: Kubernetes NMState Operator などの Operator をインストールします。
Memory タイル	過去 1 日間の傾向を示すグラフを含むメモリー使用率。
Storage タイル	過去 1 日間の傾向を示すグラフを含むストレージ使用率。

要素	説明
vCPU usage タイル	過去 1 日間の傾向を示すグラフを含む vCPU 使用率。
VirtualMachines タイル	過去 1 日間の傾向を示すグラフを含む仮想マシンの数。
Alerts タイル	重大度別にグループ化された OpenShift Virtualization アラート。
VirtualMachine statuses タイル	ステータスごとにグループ化された仮想マシンの数。
VirtualMachines per resource グラフ	テンプレートとインスタンスタイプから作成された仮想マシンの数。

2.3.1.2. Top consumers タブ

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

例2.3 Top consumers タブ

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

2.3.1.3. Migrations タブ

Migrations タブには、仮想マシンの移行のステータスが表示されます。

例2.4 Migrations タブ

要素	説明
Time period リスト	仮想マシンの移行をフィルタリングする期間を選択します。
VirtualMachineInstanceMigrations 情報 テーブル	仮想マシンの移行のリスト。

2.3.1.4. Settings タブ

Settings タブには、クラスター全体の設定が表示されます。

例2.5 Settings タブのタブ

タブ	説明
Cluster タブ	OpenShift Virtualization のバージョン、更新ステータス、ライブマイグレーション、テンプレートプロジェクト、ロードバランサーサービス、ゲスト管理、リソース管理、SCSI 永続予約の設定。
User タブ	公開 SSH キー管理、ユーザー権限、ウェルカム情報設定。
Preview features タブ	Web コンソールで プレビュー機能 を有効にできます。これらの機能は頻繁に変更されます。

2.3.1.4.1. クラスタータブ

Cluster タブには、OpenShift Virtualization のバージョンと更新ステータスが表示されます。 Cluster タブでは、ライブマイグレーションやその他の設定を設定します。

例2.6 Cluster タブ

要素	説明
インストール済みバージョン	OpenShift Virtualization バージョン
更新ステータス	OpenShift 仮想化の更新ステータス。

要素	説明
Channel	OpenShift 仮想化の更新チャネル。
General Settings セクション	このセクションを展開して、**Live migration** 設定、**SSH configuration** 設定、および **Template project** 設定を設定します。
General Settings → **Live Migration** セクション	このセクションを展開して、ライブマイグレーション設定を設定します。
General Settings → **Live Migration** → **Max. migrations per cluster** フィールド	クラスターごとのライブマイグレーションの最大数を選択します。
General Settings → **Live Migration** → **Max. migrations per node** フィールド	ノードごとのライブマイグレーションの最大数を選択します。
General Settings → **Live Migration** → **Live migration network** リスト	ライブマイグレーション専用のセカンダリーネットワークを選択します。
General Settings → **SSH Configuration** → **SSH over LoadBalancer service** スイッチ	仮想マシンに対する SSH 接続に使用する LoadBalancer サービスの作成を有効にします。 ロードバランサーを設定する必要があります。
General Settings → **SSH Configuration** → **SSH over NodePort service** スイッチ	仮想マシンへの SSH 接続に使用するノードポートサービスの作成を許可します。
General Settings → **Template project** セクション	このセクションを展開して、Red Hat テンプレートのプロジェクトを選択します。デフォルトのプロジェクトは **openshift** です。 Red Hat テンプレートを複数のプロジェクトに保存するには、テンプレートを複製 し、複製したテンプレートのプロジェクトを選択します。
Guest Management	このセクションを展開して、**Automatic subscription of new RHEL VirtualMachines** 設定と **Enable guest system log access** スイッチを設定します。

要素	説明
Guest Management → Automatic subscription of new RHEL VirtualMachines	このセクションを展開して、Red Hat Enterprise Linux (RHEL) 仮想マシンとゲストシステムログアクセスの自動サブスクリプションを有効にします。 この機能を有効にするには、クラスター管理者のパーミッション、組織 ID、およびアクティベーションキーが必要です。
Guest Management → Automatic subscription of new RHEL VirtualMachines → Activation Key フィールド	アクティベーションキーを入力します。
Guest Management → Automatic subscription of new RHEL VirtualMachines → Organization ID フィールド	組織 ID を入力します。
Guest Management → Automatic subscription of new RHEL VirtualMachines → Enable auto updates for RHEL VirtualMachines スイッチ	RHEL リポジトリーからの更新の自動取得を有効にします。 この機能を有効にするには、アクティベーションキーと組織 ID が必要です。
Guest Management → Enable guest system log access スイッチ	仮想マシンのゲストシステムログへのアクセスを有効にします。
リソースの管理	このセクションを展開して、**Auto-compute CPU limits** 設定と **Kernel Samepage Merging (KSM)** スイッチを設定します。
Resource Management → Auto-compute CPU limits	ラベルを含むプロジェクトで、自動計算 CPU 制限を有効にします。
Resource Management → Kernel Samepage Merging (KSM)	クラスター内のすべてのノードで KSM を有効にします。
SCSI Persistent Reservation	このセクションを展開して、**Enable persistent reservation** スイッチを設定します。

要素	説明
SCSI Persistent Reservation → Enable persistent reservation	ディスクの SCSI 予約を有効にします。このオプションは、クラスター対応アプリケーションにのみ使用する必要があります。

2.3.1.4.2. ユーザータブ

User タブでは、ユーザー権限を表示し、公開 SSH キーとウェルカム情報を管理します。

例2.7 User タブ

要素	説明
Manage SSH keys セクション	このセクションを展開して、公開 SSH キーをプロジェクトに追加します。 キーは、選択したプロジェクトでその後作成するすべての仮想マシンに自動的に追加されます。
Permissions セクション	このセクションをデプロイメントすると、クラスター全体のユーザー権限が表示されます。
ようこそ情報 セクション	このセクションを展開すると、**Welcome information** ダイアログが表示または非表示になります。

2.3.1.4.3. プレビュー機能タブ

Web コンソールで プレビュー機能 を有効にできます。このタブの機能は頻繁に変更されます。

2.3.2. Catalog ページ

Catalog ページの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します。

例2.8 Catalog ページ

要素	説明
InstanceTypes タブ	仮想マシンを作成するための起動可能なボリュームとインスタンスのタイプを表示します。
Template catalog タブ	仮想マシンを作成するためのテンプレートのカタログを表示します。

2.3.2.1. InstanceTypes タブ

InstanceTypes タブのインスタンスタイプから仮想マシンを作成します。

要素	説明
Add volume ボタン	クリックして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、ボリュームスナップショット、またはデータソースを使用します。
Volumes project フィールド	ブート可能なボリュームが保存されているプロジェクト。デフォルトは **openshift-virtualization-os-images** です。
Filter フィールド	オペレーティングシステムまたはリソースごとにブートソースをフィルタリングします。
Search フィールド	ブートソースを名前で検索します。
Manage columns アイコン	テーブルに表示する列を最大 9 つ選択します。
ボリュームテーブル	仮想マシンのブート可能ボリュームを選択します。
Red Hat provided タブ	Red Hat が提供するインスタンスタイプを選択します。
User provided タブ	**InstanceType** ページで作成したインスタンスタイプを選択します。
VirtualMachine details ペイン	仮想マシンの設定を表示します。
Name フィールド	オプション: 仮想マシン名を入力します。
Storage class フィールド	ストレージクラスを選択します。
公開 SSH キー	編集アイコンをクリックして、新規または既存の公開 SSH キーを追加します。
Dynamic SSH key injection スイッチ	動的 SSH キーの注入を有効にします。 RHEL のみ、動的 SSH キーの注入をサポートしています。
作成後にこの VirtualMachine を開始する チェックボックス	仮想マシンが自動的に起動しないようにするには、このチェックボックスをオフにします。
Create VirtualMachine ボタン	仮想マシンを作成します。
View YAML & CLI ボタン	YAML 設定ファイルと、コマンドラインから仮想マシンを作成するための **virtctl create** コマンドを表示します。

2.3.2.2. Template catalog タブ

Template catalog タブでテンプレートを選択して、仮想マシンを作成します。

例2.9 Template catalog タブ

要素	説明
Template project リスト	Red Hat テンプレートが存在するプロジェクトを選択します。 デフォルトでは、Red Hat テンプレートは **openshift** プロジェクトに保存されます。テンプレートプロジェクトは、Overview ページ →Settings タブ → Cluster タブ で編集できます。
All items | Default templates | User templates	使用可能なすべてのテンプレートを表示する場合は All items、デフォルトテンプレートを表示する場合は Default templates、そのユーザーが作成したテンプレートを表示する場合は User templates をクリックします。
Boot source available チェックボックス	チェックボックスをオンにして、使用可能なブートソースを含むテンプレートを表示します。
Operating system チェックボックス	チェックボックスをオンにして、選択したオペレーティングシステムのテンプレートを表示します。
Workload チェックボックス	チェックボックスをオンにして、選択したワークロードを含むテンプレートを表示します。
Search フィールド	テンプレートをキーワードで検索します。
テンプレートタイル	テンプレートタイルをクリックして、テンプレートの詳細を表示し、仮想マシンを作成します。

2.3.3. VirtualMachines ページ

VirtualMachines ページで仮想マシンを作成および管理します。

例2.10 VirtualMachines ページ

要素	説明
Create ボタン	テンプレート、ボリューム、または YAML 設定ファイルから仮想マシンを作成します。
Filter フィールド	ステータス、テンプレート、オペレーティングシステム、またはノードで仮想マシンをフィルタリングします。
Search フィールド	名前、ラベル、または IP アドレスで仮想マシンを検索します。
Manage columns アイコン	テーブルに表示する列を最大 9 つ選択します。Namespace 列は、Projects リストから All Projects が選択されている場合にのみ表示されます。

要素	説明
仮想マシンテーブル	仮想マシンのリスト。 仮想マシンの横にあるアクションメニュー  をクリックして、Stop、Restart、Pause、Clone、Migrate、Copy SSH command、Edit labels、Edit annotations, or Delete を選択します。Stop を選択すると、、アクションメニューの Force stop が Stop に置き換わります。オペレーティングシステムが応答しなくなった場合は、Force stop を使用して即時シャットダウンを開始します。 仮想マシンをクリックして、VirtualMachine details ページに移動します。

2.3.3.1. VirtualMachine details ページ

仮想マシンの設定は、VirtualMachine details ページの Configuration タブで行います。

例2.11 VirtualMachine details ページ

要素	説明
Actions メニュー	Actions メニューをクリックして、Stop、Restart、Pause、Clone、Migrate、Copy SSH command、Edit labels、Edit annotations、または Delete を選択します。Stop を選択すると、、アクションメニューの Force stop が Stop に置き換わります。オペレーティングシステムが応答しなくなった場合は、Force stop を使用して即時シャットダウンを開始します。
Overview タブ	リソースの使用率、アラート、ディスク、およびデバイス。
Metrics タブ	メモリー、CPU、ストレージ、ネットワーク、移行のメトリクス。
YAML タブ	仮想マシンの YAML 設定ファイル。
Configuration タブ	Details、Storage、Network、Scheduling、SSH、Initial run、および Metadata タブが含まれます。
Configuration → Details タブ	仮想マシンの VirtualMachine details を設定します。
Configuration → Storage タブ	仮想マシンのストレージを設定します。

要素	説明
Configuration → Network タブ	仮想マシンのネットワークを設定します。
Configuration → Scheduling タブ	特定のノードで実行される仮想マシンのスケジュールを設定します。
Configuration → SSH タブ	仮想マシンの SSH 設定を設定します。
Configuration → Initial run タブ	仮想マシンの cloud-init を設定するか、仮想マシンが Windows の場合は Sysprep を設定します。
Configuration → Metadata タブ	仮想マシンのラベルとアノテーションメタデータを設定します。
Events タブ	仮想マシンのイベントのリストを表示します。
Console タブ	仮想マシンへのコンソールセッションを開きます。
Snapshots タブ	スナップショットを作成し、そのスナップショットから仮想マシンを復元します。
Diagnostics タブ	ステータス条件とボリュームスナップショットのステータスを表示します。

2.3.3.1.1. Overview タブ

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

例2.12 Overview タブ

要素	説明
Details タイル	一般的な仮想マシン情報。
Utilization タイル	CPU、Memory、Storage、および Network transfer グラフ。デフォルトでは、**ネットワーク転送** は、すべてのネットワークの合計を表示します。特定のネットワークの内訳を表示するには、Breakdown by network をクリックします。
Hardware devices タイル	GPU とホストデバイス。
File systems タイル	ファイルシステム情報。 この情報は、ゲストエージェントにより提供されます。

要素	説明
Services タイル	サービスのリスト。
Active users タイル	アクティブなユーザーのリスト。
Alerts タイル	重大度別にグループ化された OpenShift Virtualization アラート。
General タイル	Namespace、Node、VirtualMachineInstance、Pod、Owner の情報。
Snapshots タイル	Take snapshot および snapshots テーブル。
Network interfaces タイル	Network interfaces テーブル。
Disks タイル	Disks テーブル。

2.3.3.1.2. Metrics タブ

Metrics タブには、メモリー、CPU、ネットワーク、ストレージ、移行の使用状況グラフと、ライブマイグレーションの進行状況が表示されます。

例2.13 Metrics タブ

要素	説明
Time range リスト	結果をフィルタリングする期間を選択します。
Virtualization ダッシュボード	現在のプロジェクトの Workloads タブにリンクします。
Utilization	Memory および CPU グラフ。
Storage	Storage total read/write および Storage IOPS total read/write グラフ。
Network	Network in、Network out、Network bandwidth、および Network interface グラフ。Network interface リストから All networks または特定のネットワークを選択します。
Migration	Migration および KV data transfer rate グラフ。
LiveMigration progress	LiveMigration の完了ステータス。

2.3.3.1.3. YAML タブ

YAML タブで YAML ファイルを編集して、仮想マシンを設定します。

例2.14 YAML タブ

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

2.3.3.1.4. Configuration タブ

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

例2.15 Configuration タブ上のタブ

要素	説明
Search フィールド	キーワードで設定を検索します。
Details タブ	仮想マシンの詳細。
Storage タブ	仮想マシンのストレージを設定します。
Network タブ	仮想マシンのネットワークを設定します。
Scheduling タブ	特定のノードで実行される仮想マシンのスケジュールを設定します。
SSH タブ	仮想マシンの SSH 設定を設定します。
Initial run タブ	仮想マシンの cloud-init を設定するか、仮想マシンが Windows の場合は Sysprep を設定します。
Metadata タブ	仮想マシンのラベルとアノテーションメタデータを設定します。

2.3.3.1.4.1. Details タブ

Details タブでは、仮想マシンの詳細を管理します。

-

例2.16 Details タブ

設定	説明
説明	編集アイコンをクリックして、説明を入力します。
Workload profile	編集アイコンをクリックして、ワークロードプロファイルを編集します。
CPU | Memory	編集アイコンをクリックして、**CPU | Memory** 要求を編集します。仮想マシンを再起動して、変更を適用します。
Hostname	仮想マシンのホスト名。仮想マシンを再起動して、変更を適用します。
Headless mode	ヘッドレスモードを有効にする。仮想マシンを再起動して、変更を適用します。
Guest system log access	ゲストシステムのログアクセスを有効にします。
Hardware devices	GPU とホストデバイスを管理します。
Boot management	ブートモードと順序を変更し、**Start in pause mode** を有効にします。

2.3.3.1.4.2. ストレージタブ

Storage タブでは、仮想マシンのディスクと環境を管理します。

例2.17 Storage タブ

設定	説明
Add disk ボタン	仮想マシンにディスクを追加します。
Filter フィールド	ディスクの種類でフィルタリングします。
Search フィールド	ディスクを名前で検索します。
Mount Windows drivers disk チェックボックス	VirtIO ドライバーをインストールするために、**virtio-win** コンテナーディスクを CD-ROM としてマウントする場合に選択します。
Disks テーブル	仮想マシンのディスクのリスト。 ディスクの横にある actions メニュー をクリックして、**Edit** または **Detach** を選択します。

設定	説明
Add Config Map, Secret or Service Account	リンクをクリックして、リソースリストから設定マップ、シークレット、またはサービスアカウントを選択します。

2.3.3.1.4.3. Network タブ

Network タブでは、ネットワークインターフェイスを管理します。

例2.18 Network interfaces テーブル。

設定	説明
Add network interface ボタン	仮想マシンにネットワークインターフェイスを追加します。
Filter フィールド	インターフェイスタイプでフィルタリングします。
Search フィールド	名前またはラベルでネットワークインターフェイスを検索します。
Network interface テーブル	ネットワークインターフェイスのリスト。 ネットワークインターフェイスの横にある actions メニュー  をクリックして、**Edit** または **Delete** を選択します。

2.3.3.1.4.4. Scheduling タブ

Scheduling タブで、特定のノードで実行されるように仮想マシンを設定します。

仮想マシンを再起動して変更を適用します。

例2.19 Scheduling タブ

設定	説明
Node selector	編集アイコンをクリックして、ラベルを追加し、適格なノードを指定します。
容認	編集アイコンをクリックして、許容範囲を追加し、適格なノードを指定します。

設定	説明
アフィニティールール	編集アイコンをクリックして、アフィニティールールを追加します。
Descheduler スイッチ	Descheduler を有効または無効にします。Descheduler は、実行中の Pod をエビクトして、Pod をより適切なノードに再スケジュールできるようにします。 仮想マシンをライブマイグレーションできない場合、このフィールドは無効になります。
専用リソース	編集アイコンをクリックして、**Schedule this workload with dedicated resources (guaranteed policy)** を選択します。
エビクションストラテジー	編集アイコンをクリックして、仮想マシンのエビクション戦略として **LiveMigrate** を選択します。

2.3.3.1.4.5. SSH タブ

SSH タブでは、SSH の詳細を設定します。

例2.20 SSH タブ

設定	説明
SSH access セクション	このセクションを展開して、**SSH using virtctl** と **SSH service type** を設定します。
Public SSH key セクション	このセクションを展開して、公開 SSH キーと動的 SSH 公開キーの注入を設定します。

2.3.3.1.4.6. Initial run

Initial run タブでは、cloud-init 設定の管理や、Windows 仮想マシン用の Sysprep の設定を行います。

仮想マシンを再起動して変更を適用します。

例2.21 Initial run タブ

要素	説明
Cloud-init	編集アイコンをクリックして、cloud-init 設定を編集します。
Sysprep	編集アイコンをクリックして **Autounattend.xml** または **Unattend.xml** 応答ファイルをアップロードし、Windows 仮想マシンのセットアップを自動化します。

2.3.3.1.4.7. Metadata タブ

Metadata タブでは、ラベルとアノテーションを設定します。

例2.22 Metadata タブ

要素	説明
Labels	ラベルを管理するには、編集アイコンをクリックします。
Annotations	アノテーションを管理するには、編集アイコンをクリックします。

2.3.3.1.5. Events タブ

Events タブには、仮想マシンイベントのリストが表示されます。

2.3.3.1.6. Console タブ

Console タブで仮想マシンへのコンソールセッションを開くことができます。

例2.23 Console タブ

要素	説明
Guest login credentials セクション	**Guest login credentials** を展開して、**cloud-init** で作成された認証情報を表示します。コピーアイコンをクリックして、認証情報をクリップボードにコピーします。
Console リスト	**VNC コンソール** または **Serial コンソール** を選択します。 Windows 仮想マシンの場合は、**Desktop viewer** オプションが表示されます。同じネットワーク上のマシンに RDP クライアントをインストールする必要があります。
Send key リスト	コンソールに送信するキーストロークの組み合わせを選択します。
Paste ボタン	文字列をクリップボードから VNC コンソールに貼り付けます。
Disconnect ボタン	コンソール接続を切断します。 新しいコンソールセッションを開く場合は、コンソール接続を手動で切断する必要があります。それ以外の場合、最初のコンソールセッションは引き続きバックグラウンドで実行されます。

2.3.3.1.7. Snapshots タブ

Snapshots タブでは、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の作成、スナップショットの復元、ラベルまたはアノテーションの編集、ボリュームスナップショットの編集または削除が可能です。

例2.24 Snapshots タブ

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

2.3.3.1.8. Diagnostics タブ

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

例2.25 Diagnostics タブ

要素	説明
Status conditions テーブル	仮想マシンに関して報告された状態のリストを表示します。
Filter フィールド	ステータス条件をカテゴリーと条件でフィルタリングします。
Search フィールド	ステータス条件を理由で検索します。
Manage columns アイコン	テーブルに表示する列を最大 9 つ選択します。
Volume snapshot status テーブル	ボリューム、スナップショットの有効化ステータス、および理由のリスト
DataVolume status テーブル	データボリュームとその **Phase** および **Progress** の値のリスト。

2.3.4. Templates ページ

VirtualMachine Templates ページで仮想マシンテンプレートを作成、編集、およびクローン作成します。

注記

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

例2.26 VirtualMachine Templates ページ

要素	説明
Create Template ボタン	YAML 設定ファイルを編集してテンプレートを作成します。
Filter フィールド	テンプレートをタイプ、ブートソース、テンプレートプロバイダー、またはオペレーティングシステムでフィルタリングします。
Search フィールド	名前またはラベルでテンプレートを検索します。
Manage columns アイコン	テーブルに表示する列を最大 9 つ選択します。**Namespace** 列は、**Projects** リストから **All Projects** が選択されている場合にのみ表示されます。
仮想マシンテンプレートテーブル	仮想マシンテンプレートのリスト。 テンプレートの横にある actions メニュー ⋮ をクリックして、**Edit**、**Clone**、**Edit boot source**、**Edit boot source reference**、**Edit labels**、**Edit annotations**、または **Delete** を選択します。Red Hat が提供するテンプレートは編集できません。Red Hat テンプレートのクローンを作成してから、カスタムテンプレートを編集できます。

2.3.4.1. テンプレートの詳細ページ

Template details ページで、テンプレートの設定を表示し、カスタムテンプレートを編集します。

例2.27 Template details タブ

要素	説明
YAML スイッチ	**ON** を設定して、YAML 設定ファイルでライブの変更を表示します。
Actions メニュー	**Actions** メニューをクリックして、**Edit**、**Clone**、**Edit boot source**、**Edit boot source reference** の **Edit labels** の **Edit annotations**、または **Delete** を選択します。

要素	説明
Details タブ	テンプレートの設定と設定。
YAML タブ	YAML 設定ファイル。
Scheduling タブ	スケジューリング設定。
Network interfaces タブ	ネットワークインターフェイス管理。
Disks タブ	ディスク管理。
Scripts タブ	Cloud-init、SSH キー、および Sysprep の管理。
Parameters タブ	名前とクラウドユーザーのパスワード管理。

2.3.4.1.1. Details タブ

Details タブでカスタムテンプレートを設定します。

例2.28 Details タブ

要素	説明
Name	テンプレート名
Namespace	テンプレートの namespace。
Labels	編集アイコンをクリックして、ラベルを編集します。
Annotations	編集アイコンをクリックして、注釈を編集します。
Display name	編集アイコンをクリックして、表示名を編集します。
Description	編集アイコンをクリックして、説明を入力します。
Operating system	オペレーティングシステム名。
CPU|Memory	編集アイコンをクリックして、CPU|Memory 要求を編集します。 CPU の数は、**sockets * threads * cores** の式を使用して計算されます。
Machine type	テンプレートマシンタイプ。

要素	説明
Boot mode	編集アイコンをクリックして、起動モードを編集します。
Base template	このテンプレートの作成に使用されたベーステンプレートの名前。
Created at	テンプレートの作成日。
Owner	テンプレート所有者。
Boot order	テンプレートの起動順序。
Boot source	ブートソースの可用性。
Provider	テンプレートプロバイダー
Support	テンプレートのサポートレベル。
GPU devices	編集アイコンをクリックして、GPU デバイスを追加します。
Host devices	編集アイコンをクリックして、ホストデバイスを追加します。
Headless mode	編集アイコンをクリックしてヘッドレスモードを **ON** に設定し、VNC コンソールを無効にします。

2.3.4.1.2. YAML タブ

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

例2.29 YAML タブ

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

2.3.4.1.3. Scheduling タブ

Scheduling タブでスケジュールを設定します。

例2.30 Scheduling タブ

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

2.3.4.1.4. Network interfaces タブ

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

例2.31 Network interfaces タブ

設定	説明
Add network interface ボタン	テンプレートにネットワークインターフェイスを追加します。
Filter フィールド	インターフェイスタイプでフィルタリングします。
Search フィールド	名前またはラベルでネットワークインターフェイスを検索します。
Network interface テーブル	ネットワークインターフェイスのリスト。 ネットワークインターフェイスの横にある actions メニュー ⋮ をクリックして、**Edit** または **Delete** を選択します。

2.3.4.1.5. Disks タブ

ディスクは、**Disks** タブで管理します。

例2.32 Disks タブ

設定	説明
Add disk ボタン	テンプレートにディスクを追加します。
Filter フィールド	ディスクの種類でフィルタリングします。
Search フィールド	ディスクを名前で検索します。
Disks テーブル	テンプレートディスクのリスト。 ディスクの横にある actions メニュー  をクリックして、**Edit** または **Detach** を選択します。

2.3.4.1.6. Scripts タブ

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

例2.33 Scripts タブ

要素	説明
Cloud-init	編集アイコンをクリックして、cloud-init 設定を編集します。
公開 SSH キー	編集アイコンをクリックして、新しいシークレットを作成するか、既存のシークレットを Linux 仮想マシンに割り当てます。
Sysprep	編集アイコンをクリックして **Autounattend.xml** または **Unattend.xml** 応答ファイルをアップロードし、Windows 仮想マシンのセットアップを自動化します。

2.3.4.1.7. パラメータータブ

パラメーター タブで、選択したテンプレート設定を編集します。

例2.34 Parameters タブ

要素	説明
名前	このテンプレートから作成された仮想マシンの名前パラメーターを設定します。

要素	説明
CLOUD_USER_PASSWORD	このテンプレートから作成された仮想マシンのクラウドユーザーパスワードパラメーターを設定します。

2.3.5. InstanceTypes ページ

InstanceTypes ページで仮想マシンインスタンスタイプを表示および管理します。

例2.35 VirtualMachineClusterInstancetypes ページ

要素	説明
Create ボタン	YAML 設定ファイルを編集して、インスタンスタイプを作成します。
Search フィールド	インスタンスタイプを名前またはラベルで検索します。
Manage columns アイコン	テーブルに表示する列を最大 9 つ選択します。**Namespace** 列は、**Projects** リストから **All Projects** が選択されている場合にのみ表示されます。
Instance types テーブル	インスタンスのリスト。 インスタンスタイプの横にあるアクションメニュー ⋮ をクリックして、**Clone** または **Delete** を選択します。

インスタンスタイプをクリックして、**VirtualMachineClusterInstancetypes details** ページを表示します。

2.3.5.1. VirtualMachineClusterInstancetypes details ページ

VirtualMachineClusterInstancetypes details ページでインスタンスタイプを設定します。

例2.36 VirtualMachineClusterInstancetypes details ページ

要素	説明
Details タブ	フォームを編集してインスタンスタイプを設定します。
YAML タブ	YAML 設定ファイルを編集して、インスタンスタイプを設定します。
Actions メニュー	**Edit labels**、**Edit annotations**、**Edit VirtualMachineClusterInstancetype**、または **Delete VirtualMachineClusterInstancetype** を選択します。

2.3.5.1.1. Details タブ

インスタンスタイプを設定するには、**Details** タブでフォームを編集します。

例2.37 Details タブ

要素	説明
Name	VirtualMachineClusterInstancetype の名前。
Labels	編集アイコンをクリックして、ラベルを編集します。
Annotations	編集アイコンをクリックして、注釈を編集します。
Created at	インスタンスタイプの作成日。
Owner	インスタンスタイプの所有者。

2.3.5.1.2. YAML タブ

インスタンスタイプを設定するには、**YAML** タブで YAML ファイルを編集します。

例2.38 YAML タブ

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

2.3.6. Preferences ページ

Preferences ページで仮想マシンの設定を表示および管理します。

例2.39 VirtualMachineClusterPreferences ページ

要素	説明
Create ボタン	YAML 設定ファイルを編集して設定を作成します。
Search フィールド	名前またはラベルで設定を検索します。
Manage columns アイコン	テーブルに表示する列を最大 9 つ選択します。**Namespace** 列は、**Projects** リストから **All Projects** が選択されている場合にのみ表示されます。
Preferences テーブル	設定のリスト。 設定の横にあるアクションメニュー  をクリックして、**Clone** または **Delete** を選択します。

設定をクリックして、**VirtualMachineClusterPreference details** ページを表示します。

2.3.6.1. VirtualMachineClusterPreference details ページ

VirtualMachineClusterPreference details ページで設定を指定します。

例2.40 VirtualMachineClusterPreference details ページ

要素	説明
Details タブ	フォームを編集して設定を指定します。
YAML タブ	YAML 設定ファイルを編集して設定を指定します。
Actions メニュー	**Edit labels**、**Edit annotations**、**Edit VirtualMachineClusterPreference**、または **Delete VirtualMachineClusterPreference** を選択します。

2.3.6.1.1. Details タブ

Details タブでフォームを編集して設定を指定します。

例2.41 Details タブ

要素	説明
Name	VirtualMachineClusterPreference の名前。
Labels	編集アイコンをクリックして、ラベルを編集します。

要素	説明
Annotations	編集アイコンをクリックして、注釈を編集します。
Created at	設定の作成日。
Owner	設定の所有者。

2.3.6.1.2. YAML タブ

設定タイプを設定するには、**YAML** タブで YAML ファイルを編集します。

例2.42 YAML タブ

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

2.3.7. Bootable volumes ページ

Bootable volumes ページで、使用可能なブート可能ボリュームを表示および管理します。

例2.43 Bootable volumes ページ

要素	説明
Add volume ボタン	フォームに入力するか、YAML 設定ファイルを編集して、ブート可能ボリュームを追加します。
Filter フィールド	ブート可能ボリュームをオペレーティングシステムとリソースタイプでフィルタリングします。
Search フィールド	ブート可能ボリュームを名前またはラベルで検索します。
Manage columns アイコン	テーブルに表示する列を最大 9 つ選択します。**Namespace** 列は、**Projects** リストから **All Projects** が選択されている場合にのみ表示されます。

要素	説明
ブート可能ボリュームテーブル	ブート可能なボリュームのリスト。 ブート可能ボリュームの横にあるアクションメニュー  をクリックして、**Edit**、**Remove from list**、または **Delete** を選択します。

ブート可能ボリュームをクリックして、**DataSource details** ページを表示します。

2.3.7.1. DataSource details ページ

DataSource details ページでは、ブート可能ボリュームの永続ボリューム要求 (PVC) を設定します。

例2.44 DataSource details ページ

要素	説明
Details タブ	フォームを編集して PVC を設定します。
YAML タブ	YAML 設定ファイルを編集して PVC を設定します。

2.3.7.1.1. Details タブ

Details タブのフォームを編集して、ブート可能ボリュームの永続ボリューム要求 (PVC) を設定します。

例2.45 Details タブ

要素	説明
Name	データソース名。
Namespace	データソースの namespace。
Labels	編集アイコンをクリックして、ラベルを編集します。
Annotations	編集アイコンをクリックして、注釈を編集します。
Created at	データソースの作成日。
Owner	データソースの所有者。
DataImportCron	データソースの **DataImportCron** オブジェクト。

要素	説明
Default Instance Type	このデータソースのデフォルトのインスタンス型。
Preference	特定のワークロードを実行するために必要な、優先される **VirtualMachine** 属性値。
条件 表	データソースの型、ステータス、最終更新、理由、メッセージが表示されます。

2.3.7.1.2. YAML タブ

ブート可能ボリュームの永続ボリューム要求を設定するには、**YAML** タブで YAML ファイルを編集します。

例2.46 YAML タブ

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

2.3.8. MigrationPolicies ページ

MigrationPolicies ページでワークロードの移行ポリシーを管理します。

例2.47 MigrationPolicies ページ

要素	説明
移行ポリシーの作成	フォームに設定とラベルを入力するか、YAML ファイルを編集して、移行ポリシーを作成します。
Search フィールド	名前またはラベルで移行ポリシーを検索します。
Manage columns アイコン	テーブルに表示する列を最大 9 つ選択します。**Namespace** 列は、**Projects** リストから **All Projects** が選択されている場合にのみ表示されます。

要素	説明
MigrationPolicies テーブル	移行ポリシーのリスト。 移行ポリシーの横にあるアクションメニュー  をクリックして、Edit または Delete を選択します。

移行ポリシーをクリックして、MigrationPolicy details ページを表示します。

2.3.8.1. MigrationPolicy details ページ

移行ポリシーは、MigrationPolicy details ページで設定します。

例2.48 MigrationPolicy details ページ

要素	説明
Details タブ	フォームを編集して移行ポリシーを設定します。
YAML タブ	YAML 設定ファイルを編集して、移行ポリシーを設定します。
Actions メニュー	Edit または Delete を選択します。

2.3.8.1.1. Details タブ

Details タブでカスタムテンプレートを設定します。

例2.49 Details タブ

要素	説明
Name	移行ポリシー名。
Description	移行ポリシーの説明。
設定	編集アイコンをクリックして、移行ポリシー設定を更新します。
移行ごとの帯域幅	移行ごとの帯域幅要求。帯域幅を無制限にするには、値を **0** に設定します。
自動収束	自動収束が有効になっている場合は、移行を確実に成功させるために、仮想マシンのパフォーマンスと可用性が低下する可能性があります。
ポストコピー	ポストコピーポリシー。

要素	説明
完了タイムアウト	秒単位の完了タイムアウト値。
プロジェクトラベル	**Edit** をクリックして、プロジェクトラベルを編集します。
VirtualMachine のラベル	**Edit** をクリックして仮想マシンのラベルを編集します。

2.3.8.1.2. YAML タブ

移行ポリシーを設定するには、**YAML** タブで YAML ファイルを編集します。

例2.50 YAML タブ

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

2.3.9. Checkups ページ

Checkups ページでは、仮想マシンのネットワーク遅延とストレージのチェックを実行します。

例2.51 Checkups ページ

要素	説明
Network latency タブ	ネットワーク遅延チェックを実行します。
Storage タブ	ストレージチェックを実行します。

第3章 インストール

3.1. 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。

3.1.1. 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。

- Amazon Web Services ベアメタルインスタンス。

3.1.1.1. OpenShift Dedicated 上の OpenShift Virtualization

OpenShift Virtualization は、Red Hat OpenShift Dedicated クラスターで実行できます。

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

- installer-provisioned infrastructure を使用してクラスターをインストールできます。その際に、**install-config.yaml** ファイルを編集してワーカーノードのベアメタルインスタンスタイプを確実に指定してください。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。
  詳細は、AWS へのインストール に関する OpenShift Dedicated 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 Dedicated 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 Dedicated 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 Dedicated はロードバランサーとその関連リソースを削除します。

ネットワーク

- 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。

- Amazon Elastic File System (EFS) および Amazon Elastic Block Store (EBS) は、パフォーマンスと機能が限定されるため、OpenShift Virtualization での使用は推奨されません。代わりに共有ストレージを使用してください。

関連情報

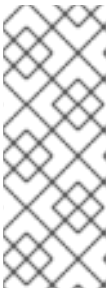
- OVN-Kubernetes セカンダリーネットワークへの仮想マシンの接続
- サービスを使用して仮想マシンを公開する

3.1.2. 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。

3.1.2.1. 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、必要なノードアフィニティールールの設定 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。

3.1.2.2. 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)。

3.1.2.3. ストレージ要件

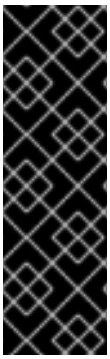
- OpenShift Dedicated によってサポートされます。
- ストレージプロビジョナーがスナップショットをサポートしている場合は、**VolumeSnapshotClass** オブジェクトをデフォルトのストレージクラスに関連付ける必要があります。

3.1.2.3.1. 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。



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

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

3.1.3. 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。

3.1.4. 物理リソースのオーバーヘッド要件

OpenShift Virtualization は OpenShift Dedicated 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 Dedicated の要件に

加えて、以下のオーバーヘッドの要件を満たす必要があります。クラスター内の物理リソースを過剰にサブスクライブすると、パフォーマンスに影響する可能性があります。

重要

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

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ついてのディスク上のストレージの予想される影響に相当します。

仮想マシンのストレージオーバーヘッド

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

例

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

関連情報

- OpenShift Dedicated ストレージの一般用語集

3.2. 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 Dedicated クラスターに追加します。

3.2.1. OpenShift Virtualization Operator のインストール

OpenShift Dedicated 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。

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

OpenShift Dedicated 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 Dedicated 4 をクラスターにインストールします。
- **cluster-admin** パーミッションを持つユーザーとして OpenShift Dedicated Web コンソールにログインします。
- ベアメタルコンピュートノードインスタンスタイプに基づいてマシンプールを作成する。詳細は、このセクションの関連情報のマシンプールの作成を参照してください。

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 についての情報を確認してから、**Install** をクリックします。
5. **Install Operator** ページで以下を行います。
   a. 選択可能な **Update Channel** オプションの一覧から **stable** を選択します。これにより、OpenShift Dedicated 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。

関連情報

- マシンセットの作成

3.2.1.2. 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。

3.2.1.2.1. 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 Dedicated 4 をクラスターにインストールします。
- 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。

3.2.1.2.2. 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。
- ベアメタルコンピュートノードインスタンスタイプに基づいてマシンプールを作成する。

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.v4.15.1   OpenShift Virtualization   4.15.1
Succeeded
```

関連情報

- マシンセットの作成

3.2.2. 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。

3.3. 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。

3.3.1. 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。

3.3.1.1. 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 Dedicated 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。

3.3.1.2. 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

- **dedicated-admin** パーミッションを持つアカウントを使用して OpenShift Dedicated 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。

3.3.1.3. Web コンソールを使用した namespace の削除

OpenShift Dedicated 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 Dedicated 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。

3.3.1.4. 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 Dedicated 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。

3.3.2. 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 Dedicated 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4章 インストール後の設定

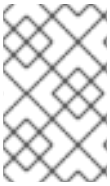
4.1. インストール後の設定

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:
  - OpenShift Dedicated 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

4.2. 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。

4.2.1. 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。

tolerations

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

4.2.2. ノード配置ルールの適用

コマンドラインを使用して **HyperConverged** または **HostPathProvisioner** オブジェクトを編集して、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。

4.2.3. ノード配置ルールの例

HyperConverged または **HostPathProvisioner** オブジェクトを編集して、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。

4.2.3.1. 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。

4.2.3.2. 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。

4.2.4. 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
- ノードテイントを使用した Pod 配置の制御

4.3. 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。

4.3.1. ネットワーキングオペレーターのインストール

4.3.2. 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後、仮想マシンへのライブマイグレーションまたは外部アクセス用に Linux ブリッジネットワークを設定できます。

4.3.2.1. 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。

4.3.2.2. 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。

4.3.3. 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。

4.3.3.1. 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}'
```

4.3.3.2. 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 Dedicated 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 ネットワークを設定しました。

手順

1. OpenShift Dedicated 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。

4.3.4. 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 Dedicated Web コンソールを使用して、仮想マシンのロードバランサーサービスの作成を有効にできます。

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。

4.4. 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。

- 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。

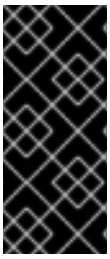
オプション: ホストパスプロビジョナー (HPP) を使用して、ローカルストレージを設定できます。

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。

4.4.1. 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。



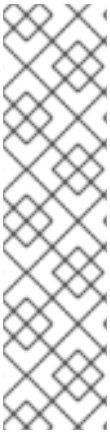
重要

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。

4.4.1.1. 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。

前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **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
```

第5章 更新

5.1. 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。

5.1.1. RHEL 9 上の OpenShift Virtualization

OpenShift Virtualization 4.15 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 4.15 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 4.15 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。

5.1.1.1. 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。

5.1.2. 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 Dedicated 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 Dedicated 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 Dedicated 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 Dedicated バージョンとの互換性が確保されます。

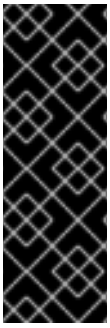
- サブスクリプションの承認ストラテジーが **Automatic** に設定されている場合に、更新プロセスは、Operator の新規バージョンが **stable** チャネルで利用可能になるとすぐに開始します。サポート可能な環境を確保するために、**自動** 承認ストラテジーを使用することを強く推奨します。OpenShift Virtualization の各マイナーバージョンは、対応する OpenShift Dedicated バージョンを実行する場合にのみサポートされます。たとえば、OpenShift Virtualization 4.15 は OpenShift Dedicated 4.15 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 Dedicated および OpenShift Virtualization の更新が同期されていない場合、クラスターはサポートされなくなります。

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

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

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



重要

AWS Elastic Block Store (EBS)ストレージを使用する仮想マシンが実行されている場合、それらをライブマイグレーションすることはできず、OpenShift Dedicated クラスターの更新をブロックする可能性があります。

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

5.1.2.1. 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。

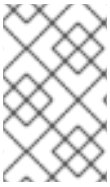
5 分間

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。

5.1.3. 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
# ...
```

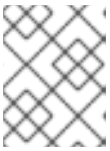
1. ワークロードの自動更新を実行するのに使用できるメソッド。設定可能な値は **LiveMigrate** および **Evict** です。上記の例のように両方のオプションを有効にした場合に、ライブマイグレーションをサポートする VMI には **LiveMigrate** を、ライブマイグレーションをサポートしない VMI には **Evict** を、更新に使用します。ワークロードの自動更新を無効にするには、**workloadUpdateStrategy**スタンザを削除するか、**workloadUpdateMethods: []** を設定して配列を空のままにします。

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。

5.1.4. 保留中の Operator 更新の承認

5.1.4.1. 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 Dedicated 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。

5.1.5. 更新ステータスの監視

5.1.5.1. 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
```

5.1.5.2. 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。

5.1.6. 関連情報

- Operator について
- Operator Lifecycle Manager の概念およびリソース
- Cluster Service Version (CSV)
- 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6章 仮想マシン

6.1. RED HAT イメージからの仮想マシンの作成

6.1.1. 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。

6.1.1.1. 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。

6.1.1.1.1. 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。

6.1.1.1.2. 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。

6.1.1.2. 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。

6.1.2. テンプレートからの仮想マシンの作成

OpenShift Dedicated 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。

6.1.2.1. 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。
- 仮想マシン設定の詳細は、**Overview**、**YAML**、および **Configuration** タブのドキュメント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。

6.1.2.2. テンプレートから仮想マシンを作成する

OpenShift Dedicated Web コンソールを使用して、利用可能なブートソースを含むテンプレートから仮想マシン(VM)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。

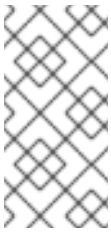
手順

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. 仮想マシンの **Quick create VirtualMachine** をクリックして、テンプレートから VM を作成します。
   オプション: 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。

6.1.2.2.1. ストレージボリュームタイプ

表6.1 ストレージボリュームタイプ

タイプ	説明

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。
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。

タイプ	説明
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。

6.1.2.2.2. 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。
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。
名前	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 URLURL**、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)	ボリュームは単一ノードで読み取り/書き込みとしてマウントできます。
	ReadWriteMany (RWX)	ボリュームは、一度に多くのノードで読み取り/書き込みとしてマウントできます。  **注記** このモードはライブマイグレーションに必要です。

6.1.2.2.3. 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。

6.1.3. インスタンスタイプからの仮想マシンの作成

OpenShift Dedicated 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(VM)を作成できます。

注記

OpenShift Virtualization 4.15 以降のインスタンスタイプからの仮想マシンの作成は、OpenShift Dedicated クラスターで使用するためにサポートされます。OpenShift Virtualization 4.14 では、インスタンスタイプから仮想マシンを作成することはテクノロジープレビュー機能であり、OpenShift Dedicated クラスターでの使用はサポートされていません。

6.1.3.1. インスタンスタイプからの仮想マシンの作成

OpenShift Dedicated 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(VM)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。

手順

1. Web コンソールで、**Virtualization → Catalog** に移動し、**InstanceTypes** タブをクリックします。
2. 次のオプションのいずれかを選択します。
   - 起動可能なボリュームを選択します。

     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。
   - **Add volume** をクリックして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、またはデータソースを使用します。次に、**Save** をクリックします。
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. 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。
5. 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**:

a. 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。

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。

d. **Save** をクリックします。

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。

6.1.4. 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。

6.1.4.1. 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。

例6.1 RHEL 仮想マシンのマニフェストの例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  labels:
    app: <vm_name> 1
  name: <vm_name>
spec:
  dataVolumeTemplates:
  - apiVersion: cdi.kubevirt.io/v1beta1
    kind: DataVolume
    metadata:
      name: <vm_name>
    spec:
      sourceRef:
        kind: DataSource
        name: rhel9 2
        namespace: openshift-virtualization-os-images
      storage:
        resources:
          requests:
            storage: 30Gi
  running: false
  template:
```

```
metadata:
  labels:
    kubevirt.io/domain: <vm_name>
spec:
  domain:
    cpu:
      cores: 1
      sockets: 2
      threads: 1
    devices:
      disks:
      - disk:
          bus: virtio
        name: rootdisk
      - disk:
          bus: virtio
        name: cloudinitdisk
      interfaces:
      - masquerade: {}
        name: default
      rng: {}
    features:
      smm:
        enabled: true
    firmware:
      bootloader:
        efi: {}
    resources:
      requests:
        memory: 8Gi
  evictionStrategy: LiveMigrate
  networks:
  - name: default
    pod: {}
  volumes:
  - dataVolume:
      name: <vm_name>
    name: rootdisk
  - cloudInitNoCloud:
      userData: |-
        #cloud-config
        user: cloud-user
        password: '<password>' 3
        chpasswd: { expire: False }
    name: cloudinitdisk
```

1. 仮想マシンの名前を指定します。
2. **Hyperconvered** CR の **spec.dataImportCronTemplate.spec.managedDataSource** フィールドに名前を指定します。
3. cloud-user のパスワードを指定します。

2. マニフェストファイルを使用して仮想マシンを作成します。

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

3. オプション: 仮想マシンを起動します。

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

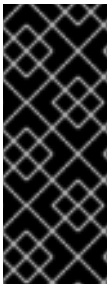
6.2. カスタムイメージからの仮想マシンの作成

6.2.1. 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 Persistent Volume Claim (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 Dedicated 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。

6.2.2. 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。

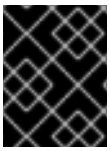
コンテナーディスクの自動更新を有効にすることができます。詳細は、ブートソースの自動更新の管理 を参照してください。

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、**DeploymentConfig** オブジェクトを削除します。

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。

6.2.2.1. 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
```

6.2.2.2. 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。

6.2.2.3. 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 Dedicated 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して、仮想マシン(VM)を作成できます。

手順

1. Web コンソールで **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。

6.2.2.4. 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. PV 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
```

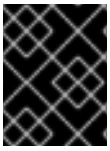
1. **VirtualMachine** マニフェストで定義したデータボリューム名を指定します。

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
```

6.2.3. 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。

6.2.3.1. 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 Dedicated 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して、仮想マシン(VM)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。

6.2.3.2. 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. PV 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。

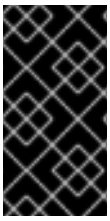
3. プロビジョニングが完了し、シリアルコンソールにアクセスして仮想マシンが起動したことを確認します。

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

6.2.4. 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。

6.2.4.1. 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 Dedicated Web コンソールを使用して、アップロードしたオペレーティングシステムイメージから仮想マシン(VM)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。

6.2.4.2. 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 を作成しました。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。

6.2.4.2.1. 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 Dedicated 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。

6.2.4.2.2. 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 Dedicated 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

6.2.4.3. 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必要です。
- 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ます。

- **virtctl** がインストールされている。
- クライアントマシンは、OpenShift Dedicated ルーターの証明書を信頼するように設定されている必要がある。

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
```

6.2.5. 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 Persistent Volume Claim (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。

6.2.5.1. 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。

6.2.5.1.1. 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。

6.2.5.1.2. 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。

6.2.5.1.3. 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
```

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

OpenShift Dedicated 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して、仮想マシン(VM)を作成できます。OpenShift Dedicated Web コンソールを使用して永続ボリューム要求(PVC)のクローンを作成して仮想マシン(VM)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。

6.2.5.3. 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 Persistent Volume Claim (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。

6.2.5.3.1. 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 Persistent Volume Claim (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。

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。

6.2.5.3.2. 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 Persistent Volume Claim (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) PV 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
```

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

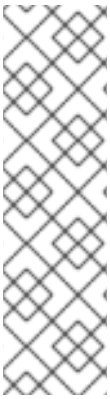
QEMU ゲストエージェントは、仮想マシンで実行され、仮想マシン、ユーザー、ファイルシステム、およびセカンダリーネットワークに関する情報をホストに渡すデーモンです。

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。

6.2.6.1. QEMU ゲストエージェントのインストール

6.2.6.1.1. 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>
```

6.2.6.1.2. 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。

6.2.6.2. 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6.2 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。

ドライバー名	ハードウェア ID	説明
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。

6.2.6.2.1. 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。

6.2.6.2.2. 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

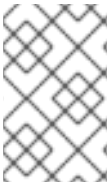
必要な Windows ドライバーをインストールするには、VirtIO コンテナーディスクを Windows 仮想マシンにアタッチする必要があります。これは既存の仮想マシンに対して実行できます。

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。

6.2.6.2.3. 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 のバージョンについての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。

6.2.6.2.4. 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については、お使いの Windows バージョンについてのインストールドキュメントを参照してください。

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。

6.2.6.2.5. 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。

6.2.6.3. VirtIO ドライバーの更新

6.2.6.3.1. 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。

6.3. 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。

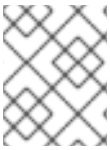
- VNC コンソール
- Serial Console
- Windows 仮想マシン用のデスクトップビューアー

6.3.1. VNC コンソールへの接続

OpenShift Dedicated 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。

6.3.1.1. Web コンソールを使用した VNC コンソールへの接続

OpenShift Dedicated 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。

6.3.1.2. 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
```

6.3.1.3. 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 以降の仮想マシンを実行している。

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. 次のコマンドを実行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>" 1
```

1 期間は時間と分で指定でき、最小期間は 10 分です。例: **5h30m**。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. 次のコマンドを実行してクラスターにログインします。

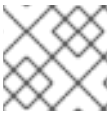
```
$ oc login --token ${VNC_TOKEN}
```

2. 次のコマンドを実行し、**virtctl** 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>
```

6.3.2. シリアルコンソールへの接続

OpenShift Dedicated 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。

6.3.2.1. 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 Dedicated 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。

6.3.2.2. 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。

6.3.3. 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。

6.3.3.1. 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 Dedicated Web コンソールを使用して、Windows 仮想マシン(VM)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を Windows仮想マシンにインストールしました。
- 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。

6.4. 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。

6.4.1. 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。

6.4.2. 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。

6.4.2.1. 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。

6.4.2.2. 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。

ヒント

OpenShift Dedicated Web コンソールを使用して、キーをプロジェクトに追加することもできます。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。

6.4.2.2.1. 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 Dedicated Web コンソールを使用して仮想マシンを作成する際に、静的で管理されるパブリック 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

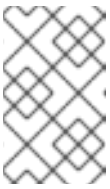
- **Configuration** タブの **Scripts** タブをクリックします。
  シークレット名は **Authorized SSH key** セクションに表示されます。

6.4.2.2.2. インスタンスタイプから仮想マシンを作成する場合のキーの追加

OpenShift Dedicated 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(VM)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。OpenShift Dedicated 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際に、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 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、**InstanceTypes** タブをクリックします。
2. 次のオプションのいずれかを選択します。
   - 起動可能なボリュームを選択します。

     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。
   - **Add volume** をクリックして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、またはデータソースを使用します。次に、**Save** をクリックします。

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. 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。

5. 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**:
     a. 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。
     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。
     d. **Save** をクリックします。

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。

6.4.2.2.3. 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。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  dataVolumeTemplates:
  - apiVersion: cdi.kubevirt.io/v1beta1
    kind: DataVolume
    metadata:
      name: example-vm-disk
```

```
  spec:
    sourceRef:
      kind: DataSource
      name: rhel9
      namespace: openshift-virtualization-os-images
    storage:
      resources:
        requests:
          storage: 30Gi
running: false
template:
  metadata:
    labels:
      kubevirt.io/domain: example-vm
  spec:
    domain:
      cpu:
        cores: 1
        sockets: 2
        threads: 1
      devices:
        disks:
        - disk:
            bus: virtio
          name: rootdisk
        - disk:
            bus: virtio
          name: cloudinitdisk
        interfaces:
        - masquerade: {}
          name: default
        rng: {}
      features:
        smm:
          enabled: true
      firmware:
        bootloader:
          efi: {}
      resources:
        requests:
          memory: 8Gi
    evictionStrategy: LiveMigrate
    networks:
    - name: default
      pod: {}
    volumes:
    - dataVolume:
        name: example-volume
      name: example-vm-disk
      - cloudInitNoCloud: <.>
          userData: |-
            #cloud-config
            user: cloud-user
            password: <password>
            chpasswd: { expire: False }
        name: cloudinitdisk
```

```
    accessCredentials:
     - sshPublicKey:
         propagationMethod:
           noCloud: {}
         source:
           secret:
             secretName: authorized-keys <.>
---
apiVersion: v1
kind: Secret
metadata:
  name: authorized-keys
data:
  key: |
    MIIEpQIBAAKCAQEAulqb/Y... <.>
```

<.> **cloudInitNoCloud** データソースを指定します。<.> **Secret** オブジェクト名を指定します。
<.> 公開 SSH キーを貼り付けます。

2. **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. 仮想マシンを起動します。

```
$ 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
# ...
```

6.4.2.3. 動的なキー管理

OpenShift Dedicated 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挿入を有効にできます。その後、実行時にキーを更新できます。

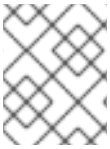
注記

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。

6.4.2.3.1. 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 Dedicated 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際に、動的パブリック SSH キーの挿入を有効にできます。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。

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。

6.4.2.3.2. 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 Dedicated 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(VM)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。OpenShift Dedicated 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際に、動的な SSH キー挿入を有効にできます。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、**InstanceTypes** タブをクリックします。

2. 次のオプションのいずれかを選択します。

   - 起動可能なボリュームを選択します。

     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。

   - **Add volume** をクリックして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、またはデータソースを使用します。次に、**Save** をクリックします。

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. 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。
6. 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**:
     a. 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。
     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。
     d. **Save** をクリックします。
7. **VirtualMachine details** セクションで **Dynamic SSH key injection** をオンに設定します。
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。

6.4.2.3.3. 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 Dedicated Web コンソールを使用して、仮想マシン(VM)の動的キー挿入を有効にできます。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。

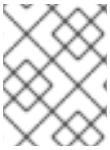
c. オプション: **Automatically apply this key to any new VirtualMachine you create in this project** を選択します。

5. **Dynamic SSH key injection** をオンに設定します。

6. **Save** をクリックします。

6.4.2.3.4. 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。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: example-namespace
spec:
  dataVolumeTemplates:
  - apiVersion: cdi.kubevirt.io/v1beta1
    kind: DataVolume
    metadata:
      name: example-vm-disk
    spec:
      sourceRef:
        kind: DataSource
        name: rhel9
        namespace: openshift-virtualization-os-images
      storage:
        resources:
          requests:
            storage: 30Gi
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/domain: example-vm
```

```
    spec:
      domain:
        cpu:
          cores: 1
          sockets: 2
          threads: 1
        devices:
          disks:
          - disk:
              bus: virtio
            name: rootdisk
          - disk:
              bus: virtio
            name: cloudinitdisk
          interfaces:
          - masquerade: {}
            name: default
          rng: {}
        features:
          smm:
            enabled: true
        firmware:
          bootloader:
            efi: {}
        resources:
          requests:
            memory: 8Gi
      evictionStrategy: LiveMigrate
      networks:
      - name: default
        pod: {}
      volumes:
      - dataVolume:
          name: example-volume
        name: example-vm-disk
        - cloudInitNoCloud: <.>
            userData: |-
              #cloud-config
              user: cloud-user
              password: <password>
              chpasswd: { expire: False }
              runcmd:
                - [ setsebool, -P, virt_qemu_ga_manage_ssh, on ]
          name: cloudinitdisk
      accessCredentials:
        - sshPublicKey:
            propagationMethod:
              qemuGuestAgent:
                users: ["user1","user2","fedora"] <.>
            source:
              secret:
                secretName: authorized-keys <.>
---
apiVersion: v1
kind: Secret
metadata:
```

```
  name: authorized-keys
data:
  key: |
    MIIEpQIBAAKCAQEAulqb/Y... <.>
```

<.> **cloudInitNoCloud** データソースを指定します。<.> ユーザー名を指定します。<.> **Secret** オブジェクト名を指定します。<.> 公開 SSH キーを貼り付けます。

2. **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. 仮想マシンを起動します。

```
$ 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: ["user1","user2","fedora"]
            source:
              secret:
                secretName: authorized-keys
# ...
```

6.4.2.4. 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。

6.4.3. 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>
```

6.4.4. 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。

6.4.4.1. 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。

6.4.4.2. サービスの作成

OpenShift Dedicated 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(VM)を公開するサービスを作成できます。

6.4.4.2.1. 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 Dedicated Web コンソールを使用して、仮想マシンのロードバランサーサービスの作成を有効にできます。

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。

6.4.4.2.2. Web コンソールを使用したサービスの作成

OpenShift Dedicated 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

- 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を設定しました。
- ロードバランサーサービスを作成するには、ロードバランサーサービスの作成を有効にしました。

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。

6.4.4.2.3. 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。

6.4.4.2.4. 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
```

6.4.4.3. 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。

6.4.5. 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

- セカンダリーネットワーク を設定した。
- ネットワークアタッチメント定義 を作成した。

6.4.5.1. 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 Dedicated 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。

6.4.5.2. 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
```

6.5. 仮想マシンの編集

OpenShift Dedicated Web コンソールを使用して仮想マシン(VM)設定を更新できます。YAML ファイル または **VirtualMachine の詳細** 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。

6.5.1. 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 をインストールした。

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>
```

6.5.2. 仮想マシンへのディスクの追加

OpenShift Dedicated 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。

6.5.2.1. 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。

フィールド	説明
URL を使用したインポート (PVC の作成)	URL (HTTP または HTTPS エンドポイント) を介してコンテンツをインポートします。
既存 PVC の使用	クラスターですでに利用可能な PVC を使用します。
既存の PVC のクローン作成 (PVC の作成)	クラスターで利用可能な既存の PVC を選択し、このクローンを作成します。
レジストリーを使用したインポート (PVC の作成)	コンテナーレジストリーを使用してコンテンツをインポートします。
名前	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 URLURL**、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)	ボリュームは単一ノードで読み取り/書き込みとしてマウントできます。

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。

6.5.3. 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 Dedicated 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

6.6. 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。

6.6.1. 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。

6.6.2. 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。

6.6.3. 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。

6.6.4. 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。

6.7. 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。

6.7.1. 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 Dedicated 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。

6.7.2. 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。

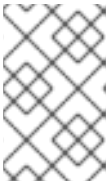
前提条件

- 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

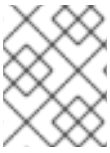
このコマンドは、現在のプロジェクト内の VM のみを削除します。削除する仮想マシンが別のプロジェクトまたは namespace にある場合は、**-n <project_name>** オプションを指定します。

6.8. 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。

6.8.1. 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。

6.8.2. 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/example-export/external/manifests/all (1)
      - type: auth-header-secret
        url: https://vmexport-proxy.test.net/api/export.kubevirt.io/v1alpha1/namespaces/example/virtualmachineexports/example-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 **Data Volume** マニフェスト、外部 URL のイングレス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 **Data Volume** 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。

6.9. 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。

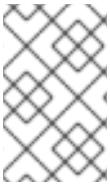
6.9.1. 仮想マシンインスタンスについて

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。

6.9.2. 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。

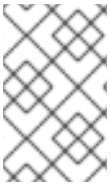
手順

- 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
```

6.9.3. 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。

6.9.4. 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 Dedicated 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。

6.9.5. 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>
```

6.9.6. 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. Open Shift Dedicated 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。

6.10. 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。

6.10.1. 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。

6.10.2. 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

- 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。

6.10.3. 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。

6.10.4. 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。

6.10.5. 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。

6.11. 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。

6.11.1. 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 Persistent Volume Claims (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。

6.11.2. 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 Persistent Volume Claim (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。

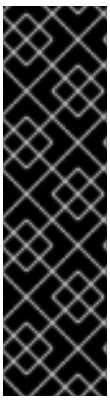
3. 変更を適用するには、エディターを保存し、終了します。
4. オプション: 実行中の仮想マシンを編集している場合は、変更を有効にするためにこれを再起動する必要があります。

6.12. 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



重要

Red Hat OpenShift Pipelines を使用した仮想マシンの管理は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

6.12.1. 前提条件

- **cluster-admin** パーミッションを持つ OpenShift Dedicated クラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift Pipelines CLI がインストールされている。

6.12.2. Scheduling, Scale, and Performance (SSP) リソースのデプロイ

SSP Operator のサンプル Tekton Tasks と Pipelines は、OpenShift Virtualization をインストールするときにデフォルトではデプロイされません。SSP Operator の Tekton リソースをデプロイするには、**HyperConverged** カスタムリソース (CR) で **deployTektonTaskResources** 機能ゲートを有効にします。

手順

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

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

2. **spec.featureGates.deployTektonTaskResources** フィールドを **true** に設定します。

```
apiVersion: hco.kubevirt.io/v1beta1
kind: HyperConverged
metadata:
  name: kubevirt-hyperconverged
  namespace: kubevirt-hyperconverged
spec:
  tektonPipelinesNamespace: <user_namespace> (1)
  featureGates:
    deployTektonTaskResources: true (2)
# ...
```

(1) パイプラインが実行される namespace。

(2) SSP オペレーターによって Tekton リソースをデプロイするために有効になる機能ゲート。

注記

後でフィーチャーゲートを無効にしても、タスクとサンプルパイプラインは引き続き使用できます。

3. 変更を保存し、エディターを終了します。

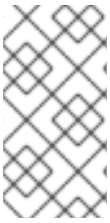
6.12.3. SSP Operator によってサポートされる仮想マシンのタスク

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6.3 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。

タスク	説明
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 **edit-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。

6.12.4. パイプラインの例

SSP Operator には、次の **Pipeline** マニフェストの例が含まれています。サンプルパイプラインは、Web コンソールまたは CLI を使用して実行できます。

複数バージョンの Windows が必要な場合は、複数のインストーラーパイプラインを実行する必要がある可能性があります。複数のインストーラーパイプラインを実行する場合、それぞれに **autounattend** config map やベースイメージ名などの一意のパラメーターが必要です。たとえば、Windows 10 および Windows 11 または Windows Server 2022 イメージが必要な場合は、Windows efi インストーラーパイプラインと Windows bios インストーラーパイプラインの両方を実行する必要があります。一方で、Windows 11 および Windows Server 2022 イメージが必要な場合、実行する必要があるのは Windows efi インストーラーパイプラインのみです。

Windows EFI インストーラーパイプライン

このパイプラインは、Windows 11 または Windows Server 2022 を Windows インストールイメージ (ISO ファイル) から新しいデータボリュームにインストールします。インストールプロセスの実行には、カスタムアンサーファイルが使用されます。

Windows BIOS インストーラーパイプライン

このパイプラインは、Windows 10 を Windows インストールイメージ (ISO ファイル) から新しいデータボリュームにインストールします。インストールプロセスの実行には、カスタムアンサーファイルが使用されます。

Windows カスタマイズパイプライン

このパイプラインは、基本的な Windows 10、11、または Windows Server 2022 インストールのデータボリュームを複製し、Microsoft SQL Server Express または Microsoft Visual Studio Code をインストールすることでカスタマイズして、新しいイメージとテンプレートを作成します。

注記

サンプルパイプラインは、OpenShift Dedicated によって事前定義された **sysprep** で、Microsoft ISO ファイルに適した設定マップ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。

6.12.4.1. Web コンソールを使用してサンプルパイプラインを実行する

サンプルパイプラインは、Web コンソールの **Pipelines** メニューから実行できます。

手順

1. サイドメニューの **Pipelines** → **Pipelines** をクリックします。
2. パイプラインを選択して、**Pipeline details** ページを開きます。
3. **Actions** リストから、**Start** を選択します。**Start Pipeline** ダイアログが表示されます。
4. パラメーターのデフォルト値を保持し、**Start** をクリックしてパイプラインを実行します。**Details** タブでは、各タスクの進行状況が追跡され、パイプラインのステータスが表示されます。

6.12.4.2. CLI を使用してサンプルパイプラインを実行する

PipelineRun リソースを使用して、サンプルパイプラインを実行します。**PipelineRun** オブジェクトは、パイプラインの実行中のインスタンスです。これは、クラスター上の特定の入力、出力、および実行パラメーターで実行されるパイプラインをインスタンス化します。また、パイプライン内のタスクごとに **TaskRun** オブジェクトを作成します。

手順

1. Windows 10 インストーラーパイプラインを実行するには、次の **PipelineRun** マニフェストを作成します。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: windows10-installer-run-
  labels:
    pipelinerun: windows10-installer-run
spec:
  params:
  - name: winImageDownloadURL
    value: <link_to_windows_10_iso> (1)
  pipelineRef:
    name: windows10-installer
  taskRunSpecs:
    - pipelineTaskName: copy-template
      taskServiceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template
```

```
      taskServiceAccountName: modify-vm-template-task
    - pipelineTaskName: create-vm-from-template
      taskServiceAccountName: create-vm-from-template-task
    - pipelineTaskName: wait-for-vmi-status
      taskServiceAccountName: wait-for-vmi-status-task
    - pipelineTaskName: create-base-dv
      taskServiceAccountName: modify-data-object-task
    - pipelineTaskName: cleanup-vm
      taskServiceAccountName: cleanup-vm-task
  status: {}
```

1 Windows 10 64 ビット ISO ファイルの URL を指定します。製品の言語は英語 (米国) でなければなりません。

2. **PipelineRun** マニフェストを適用します。

```
$ oc apply -f windows10-installer-run.yaml
```

3. Windows 10 カスタマイズパイプラインを実行するには、次の **PipelineRun** マニフェストを作成します。

```
apiVersion: tekton.dev/v1beta1
kind: PipelineRun
metadata:
  generateName: windows10-customize-run-
  labels:
    pipelinerun: windows10-customize-run
spec:
  params:
    - name: allowReplaceGoldenTemplate
      value: true
    - name: allowReplaceCustomizationTemplate
      value: true
  pipelineRef:
    name: windows10-customize
  taskRunSpecs:
    - pipelineTaskName: copy-template-customize
      taskServiceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template-customize
      taskServiceAccountName: modify-vm-template-task
    - pipelineTaskName: create-vm-from-template
      taskServiceAccountName: create-vm-from-template-task
    - pipelineTaskName: wait-for-vmi-status
      taskServiceAccountName: wait-for-vmi-status-task
    - pipelineTaskName: create-base-dv
      taskServiceAccountName: modify-data-object-task
    - pipelineTaskName: cleanup-vm
      taskServiceAccountName: cleanup-vm-task
    - pipelineTaskName: copy-template-golden
      taskServiceAccountName: copy-template-task
    - pipelineTaskName: modify-vm-template-golden
      taskServiceAccountName: modify-vm-template-task
status: {}
```

4. **PipelineRun** マニフェストを適用します。

```
$ oc apply -f windows10-customize-run.yaml
```

6.12.5. 関連情報

- Red Hat OpenShift Pipelines を使用してアプリケーションの CI/CD ソリューションを作成する
- Windows 仮想マシンの作成

6.13. 高度な仮想マシン管理

6.13.1. 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

6.13.1.1. 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。

6.13.1.2. 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

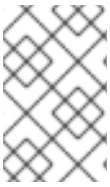
6.13.2. 仮想マシンのノードの指定

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。

6.13.2.1. 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

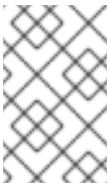
仮想マシンは、キーと値のペアまたはこのフィールドで指定したペアを使用してラベルが付けられたノードに Pod をスケジュールできます。ノードには、リスト表示されたすべてのペアに一致するラベルがなければなりません。

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。

tolerations

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



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 Dedicated は、制約を満たさない場合、実行中のワークロードを再スケジューリングしません。

6.13.2.2. 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。

6.13.2.2.1. 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
# ...
```

6.13.2.2.2. 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。

6.13.2.2.3. 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。

6.13.2.2.4. 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"
# ...
```

6.13.2.3. 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
- ノードテイントを使用した Pod 配置の制御

6.13.3. 証明書ローテーションの設定

証明書ローテーションパラメーターを設定して、既存の証明書を置き換えます。

6.13.3.1. 証明書ローテーションの設定

これは、Web コンソールでの OpenShift Virtualization のインストール時に、または **HyperConverged** カスタムリソース (CR) でインストール後に実行することができます。

手順

1. 以下のコマンドを実行して **HyperConverged** CR を開きます。

```
$ oc edit 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。

6.13.3.2. 証明書ローテーションパラメーターのトラブルシューティング

1 つ以上の **certConfig** 値を削除すると、デフォルト値が以下のいずれかの条件と競合する場合を除き、デフォルト値に戻ります。

- **ca.renewBefore** の値は **ca.duration** の値以下である必要があります。
- **server.duration** の値は **ca.duration** の値以下である必要があります。
- **server.renewBefore** の値は **server.duration** の値以下である必要があります。

デフォルト値がこれらの条件と競合すると、エラーが発生します。

以下の例で **server.duration** 値を削除すると、デフォルト値の **24h0m0s** は **ca.duration** の値よりも大きくなり、指定された条件と競合します。

例

```
certConfig:
  ca:
    duration: 4h0m0s
    renewBefore: 1h0m0s
  server:
    duration: 4h0m0s
    renewBefore: 4h0m0s
```

これにより、以下のエラーメッセージが表示されます。

```
error: hyperconvergeds.hco.kubevirt.io "kubevirt-hyperconverged" could not be patched: admission webhook "validate-hco.kubevirt.io" denied the request: spec.certConfig: ca.duration is smaller than server.duration
```

エラーメッセージには、最初の競合のみが記載されます。続行する前に、すべての certConfig の値を確認します。

6.13.4. デフォルトの CPU モデルの設定

HyperConverged カスタムリソース (CR) の **defaultCPUModel** 設定を使用して、クラスター全体のデフォルト CPU モデルを定義します。

仮想マシン (VM) の CPU モデルは、仮想マシンおよびクラスター内の CPU モデルの可用性によって異なります。

- 仮想マシンに定義された CPU モデルがない場合:
  - **defaultCPUModel** は、クラスター全体のレベルで定義された CPU モデルを使用して自動的に設定されます。
- 仮想マシンとクラスターの両方に CPU モデルが定義されている場合:
  - 仮想マシンの CPU モデルが優先されます。
- 仮想マシンにもクラスターにも CPU モデルが定義されていない場合:
  - ホストモデルは、ホストレベルで定義された CPU モデルを使用して自動的に設定されます。

6.13.4.1. デフォルトの CPU モデルの設定

HyperConverged カスタムリソース (CR) を更新して、**defaultCPUModel** を設定します。OpenShift Virtualization の実行中に、**defaultCPUModel** を変更できます。

注記

defaultCPUModel では、大文字と小文字が区別されます。

前提条件

- OpenShift CLI (oc) のインストール。

手順

1. 以下のコマンドを実行して **HyperConverged** CR を開きます。

```
$ oc edit 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。

6.13.5. 仮想マシンに UEFI モードを使用する

Unified Extensible Firmware Interface (UEFI) モードで仮想マシン (VM) を起動できます。

6.13.5.1. 仮想マシンの UEFI モードについて

レガシー BIOS などの Unified Extensible Firmware Interface (UEFI) は、コンピューターの起動時にハードウェアコンポーネントやオペレーティングシステムのイメージファイルを初期化します。UEFI は BIOS よりも最新の機能とカスタマイズオプションをサポートするため、起動時間を短縮できます。

これは、**.efi** 拡張子を持つファイルに初期化と起動に関する情報をすべて保存します。このファイルは、EFI System Partition (ESP) と呼ばれる特別なパーティションに保管されます。ESP には、コンピューターにインストールされるオペレーティングシステムのブートローダープログラムも含まれます。

6.13.5.2. UEFI モードでの仮想マシンの起動

VirtualMachine マニフェストを編集して、UEFI モードで起動するように仮想マシンを設定できます。

前提条件

- OpenShift CLI (**oc**) がインストールされている。

手順

1. **VirtualMachine** マニフェストファイルを編集または作成します。**spec.firmware.bootloader** スタンザを使用して、UEFI モードを設定します。

セキュアブートがアクティブな状態の UEFI モードでのブート

```
apiversion: kubevirt.io/v1
kind: VirtualMachine
```

```
metadata:
  labels:
    special: vm-secureboot
  name: vm-secureboot
spec:
  template:
    metadata:
      labels:
        special: vm-secureboot
    spec:
      domain:
        devices:
          disks:
          - disk:
              bus: virtio
            name: containerdisk
        features:
          acpi: {}
          smm:
            enabled: true (1)
        firmware:
          bootloader:
            efi:
              secureBoot: true (2)
# ...
```

1. OpenShift Virtualization では、UEFI モードでセキュアブートを実行するために **SMM** (System Management Mode) を有効にする必要があります。

2. OpenShift Virtualization は、UEFI モードを使用する場合に、セキュアブートの有無に関わらず、仮想マシンをサポートします。セキュアブートが有効な場合には、UEFI モードが必要です。ただし、セキュアブートを使用せずに UEFI モードを有効にできます。

2. 以下のコマンドを実行して、マニフェストをクラスターに適用します。

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

6.13.6. 仮想マシンの PXE ブートの設定

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

6.13.6.1. 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

- 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 Dedicated 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
```

6.13.6.2. 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、Custom Resource Definition)

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

ネットワーク接続定義 (NAD)

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。

6.13.7. 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。

6.13.7.1. ポリシー属性

仮想マシン (VM) をスケジュールするには、ポリシー属性と、仮想マシンがノードでスケジュールされる際の互換性について一致する CPU 機能を指定します。仮想マシンに指定されるポリシー属性は、その仮想マシンをノードにスケジュールする方法を決定します。

ポリシー属性	説明
force	仮想マシンは強制的にノードでスケジュールされます。これは、ホストの CPU が仮想マシンの CPU に対応していない場合でも該当します。
require	仮想マシンが特定の CPU モデルおよび機能仕様で設定されていない場合に仮想マシンに適用されるデフォルトのポリシー。このデフォルトポリシー属性または他のポリシー属性のいずれかを持つ CPU ノードの検出をサポートするようにノードが設定されていない場合、仮想マシンはそのノードでスケジュールされません。ホストの CPU が仮想マシンの CPU をサポートしているか、ハイパーバイザーが対応している CPU モデルをエミュレートできる必要があります。
optional	仮想マシンがホストの物理マシンの CPU でサポートされている場合は、仮想マシンがノードに追加されます。
disable	仮想マシンは CPU ノードの検出機能と共にスケジュールすることはできません。
forbid	この機能がホストの CPU でサポートされ、CPU ノード検出が有効になっている場合でも、仮想マシンはスケジュールされません。

6.13.7.2. 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。

6.13.7.3. 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。

6.13.7.4. ホストモデルでの仮想マシンのスケジューリング

仮想マシン (VM) の CPU モデルが **host-model** に設定されている場合、仮想マシンはスケジュールされているノードの CPU モデルを継承します。

手順

- 仮想マシン設定ファイルの **domain** 仕様を編集します。以下の例は、仮想マシン (VM) に指定される **host-model** を示しています。

```
apiVersion: kubevirt/v1alpha3
kind: VirtualMachine
metadata:
  name: myvm
spec:
  template:
    spec:
      domain:
        cpu:
          model: host-model 1
```

1 スケジュールされるノードの CPU モデルを継承する仮想マシン。

6.13.7.5. 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
[...]
```

6.13.8. 仮想マシンの高可用性について

修復ノードを設定することで、仮想マシン (VM) の高可用性を有効化できます。

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、メンテナンスについて、詳しくは Red Hat OpenShift のワークロードの可用性 を参照してください。

6.14. 仮想マシンディスク

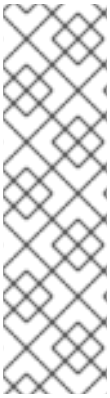
6.14.1. 仮想マシンディスクのホットプラグ

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。

ホットプラグされたディスクは、再起動後も仮想マシンに残ります。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。

6.14.1.1. 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 Dedicated 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(VM)にアタッチすることで、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、アンプラグする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。

6.14.1.2. 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>
```

6.14.2. 仮想マシンディスクの拡張

ディスクの Persistent Volume Claim (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。

6.14.2.1. 仮想マシンディスク PVC の拡張

ディスクの Persistent Volume Claim (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

6.14.2.2. 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
```

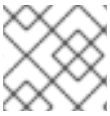
データボリューム用の関連情報

- 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

第7章 ネットワーク

7.1. 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 Dedicated ネットワークとその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。

7.1.1. 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、Custom Resource Definition)

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

ネットワーク接続定義 (NAD)

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。

7.1.2. 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。

7.1.3. 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

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。

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。

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。OpenShift Dedicated 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 IP アドレスを表示できます。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。

7.1.4. 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。

7.1.5. 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。

7.1.6. 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。

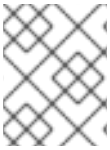
- **virtctl port-forward コマンド**
  virtctl port-foward コマンドを **.ssh/config** ファイルに追加し、OpenSSH を使用して仮想マシンに接続します。

- **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。

7.2. 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。

7.2.1. 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。

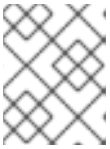
手順

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
```

7.2.2. 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 Dedicated 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}"
```

7.2.3. 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。

7.3. 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。

7.3.1. 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。

7.3.2. 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]**

7.3.3. 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
```

7.4. 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。

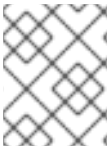
- レイヤー 2 トポロジーは、クラスター全体の論理スイッチでワークロードを接続します。OVN-Kubernetes Container Network Interface (CNI) プラグインは、 Geneve (Generic Network Virtualization Encapsulation) プロトコルを使用してノード間にオーバーレイネットワークを作成します。このオーバーレイネットワークを使用すると、追加の物理ネットワークインフラストラクチャーを設定することなく、さまざまなノード上の仮想マシンを接続できます。
- ローカルネットトポロジーは、セカンダリーネットワークを物理アンダ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。

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。
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。

7.4.1. OVN-Kubernetes NAD の作成

OpenShift Dedicated 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 localnet ネットワークアタッチメント定義(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。

7.4.1.1. 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
```

7.4.1.2. 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
```

7.4.2. 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 Dedicated Web コンソールまたは CLI を使用して、仮想マシン(VM)を OVN-Kubernetes セカンダリーネットワークインターフェイスに割り当てることができます。

7.4.2.1. 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: default
          masquerade: {}
        - name: secondary 1
          bridge: {}
      resources:
        requests:
          memory: 1024Mi
    networks:
    - name: default
      pod: {}
    - 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。

7.4.2.2. 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。

7.4.2.3. 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 Dedicated 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

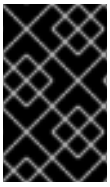
1. Web コンソールで、**Networking → NetworkAttachmentDefinitions** に移動します。
2. **Create Network Attachment Definition**をクリックします。ネットワーク接続定義は、それを使用する Pod または仮想マシンと同じ namespace にある必要があります。
3. 一意の **Name** およびオプションの **Description** を入力します。
4. **Network Type** リストで **OVN Kubernetes secondary localnet network**を選択します。
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。

7.5. 仮想マシンのサービスメッシュへの接続

OpenShift Virtualization が OpenShift Service Mesh に統合されるようになりました。IPv4 を使用してデフォルトの Pod ネットワークで仮想マシンのワークロードを実行する Pod 間のトラフィックのモニター、可視化、制御が可能です。

7.5.1. 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。

7.6. 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用に専用の セカンダリーネットワーク 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。

7.6.1. 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}'
```

7.6.2. 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 Dedicated 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 ネットワークを設定しました。

手順

1. OpenShift Dedicated 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。

7.6.3. 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

7.7. 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。

OpenShift Dedicated 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 IP アドレスを表示できます。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。

7.7.1. 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。

7.7.1.1. 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ます。
- 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 サーバーがあります。

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。

7.7.2. 仮想マシンの IP アドレスの表示

OpenShift Dedicated 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 IP アドレスを表示できます。

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。

7.7.2.1. 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 Dedicated Web コンソールを使用して、仮想マシン(VM)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 Dedicated 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。

7.7.2.2. 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
```

7.7.3. 関連情報

- QEMU ゲストエージェントのインストール

7.8. 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。

7.8.1. 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8章 ストレージ

8.1. 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。

8.1.1. ストレージ

次のストレージ設定タスクは必須です。

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。

8.1.2. 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。

8.1.3. 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。

8.1.4. 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。

8.2. 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) がストレージプロバイダーを認識しない場合は、ストレージプロファイルを設定する必要があります。

認識されたストレージタイプの場合、CDI は PVC の作成を最適化する値を提供します。ただし、ストレージプロファイルをカスタマイズする場合は、ストレージクラスの自動設定を行うことができます。

8.2.1. 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。

8.2.1.1. 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。このクローン作成ストラテジーは、一時的なボリュームスナップショットを使用してボリュームのクローンを作成します。ストレージプロビジョナーは、Container Storage Interface (CSI) スナップショットをサポートする必要があります。
- **copy** は、ソース Pod とターゲット Pod を使用して、ソースボリュームからターゲットボリュームにデータをコピーします。ホスト支援型でのクローン作成は、最も効率的な方法です。
- **csi-clone** は、CSI クローン API を使用して、中間ボリュームスナップショットを使用せずに、既存のボリュームのクローンを効率的に作成します。ストレージプロファイルが定義されていない場合にデフォルトで使用される **snapshot** または **copy** とは異なり、CSI ボリュームのクローンは、プロビジョナーのストレージクラスの **StorageProfile** オブジェクトに指定した場合にだけ使用されます。

注記

YAML **spec** セクションのデフォルトの **claimPropertySets** を変更せずに、CLI でクローンストラテジーを設定することもできます。

ストレージプロファイルの例

```
apiVersion: cdi.kubevirt.io/v1beta1
kind: StorageProfile
metadata:
  name: <provisioner_class>
# ...
spec:
  claimPropertySets:
  - accessModes:
    - ReadWriteOnce (1)
    volumeMode:
      Filesystem (2)
  cloneStrategy: csi-clone (3)
status:
  provisioner: <provisioner>
  storageClass: <provisioner_class>
```

1. アクセスモードを指定します。
2. ボリュームモードを指定します。
3. デフォルトのクローン戦略を指定します。

8.3. 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。

8.3.1. 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。

8.3.1.1. 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。

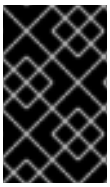
```
$ oc patch hyperconverged kubevirt-hyperconverged -n openshift-cnv \
  --type json -p '[{"op": "replace", "path": \
  "/spec/featureGates/enableCommonBootImageImport", \
  "value": false}]'
```

  - 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}]'
```

8.3.2. 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

ストレージプロファイル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、ストレージプロファイルの設定 を参照してください。

8.3.2.1. 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。

8.3.2.2. 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
    spec:
      schedule: "0 */12 * * *" 2
      template:
        spec:
          source:
            registry: 3
              url: docker://quay.io/containerdisks/centos:7-2009
          storage:
            resources:
              requests:
                storage: 10Gi
      managedDataSource: centos7 4
      retentionPolicy: "None" 5
```

1. このアノテーションは、**volumeBindingMode** が **WaitForFirstConsumer** に設定されたストレージクラスに必要です。
2. cron 形式で指定されるジョブのスケジュール。
3. レジストリーソースからデータボリュームを作成するのに使用します。**node** docker キャッシュに基づくデフォルトの **node pullMethod** ではなく、デフォルトの **pod pullMethod** を使用します。**node** docker キャッシュはレジストリーイメージが**Container.Image** で利用可能な場合に便利ですが、CDI インポーターはこれにアクセスすることは許可されていません。

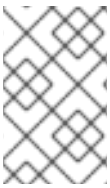
4 利用可能なブートソースとして検出するカスタムイメージの場合、イメージの **managedDataSource** の名前が、仮想マシンテンプレート YAML ファイルの

5 cron ジョブが削除されたときにデータボリュームおよびデータソースを保持するには、**All** を使用します。cron ジョブが削除されたときにデータボリュームおよびデータソースを削除するには、**None** を使用します。

3. ファイルを保存します。

8.3.2.3. 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。

8.3.3. 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。

8.3.4. 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。

8.4. 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。

8.4.1. 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}'
```

8.5. 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。

8.5.1. 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
```

8.5.1.1. 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

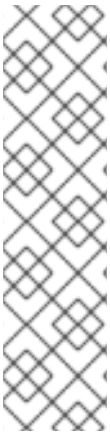
仮想マシンは、ローカル PV に基づくデータボリュームを使用します。ローカル PV は特定のノードにバインドされます。ディスクイメージは仮想マシンで使用するために準備されますが、ローカルストレージ PV がすでに固定されたノードに仮想マシンをスケジュールすることができない可能性があります。

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。

8.5.1.2. 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。

前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. **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

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
```

8.5.2. 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
```

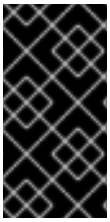
1 この値は、ブロックボリュームモードの PV にのみ必要です。

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。

8.5.2.1. 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
```

8.6. 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。

8.6.1. 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。

8.7. 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。

8.7.1. 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。

8.7.2. 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。

8.7.3. 関連情報

- プロジェクトごとのリソースクォータ

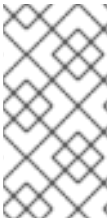
8.8. CDI のスクラッチ領域の用意

8.8.1. 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。

8.8.2. スクラッチ領域を必要とする CDI 操作

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

8.8.3. 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 (PVC) のストレージクラスを使用します。

3. デフォルトのエディターを保存して終了し、**HyperConverged** CR を更新します。

8.8.4. 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

8.8.5. 関連情報

- 動的プロビジョニング

8.9. 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。

8.9.1. 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。

8.9.2. 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
# ...
```

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。

8.10. 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。

8.10.1. 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9章 ライブマイグレーション

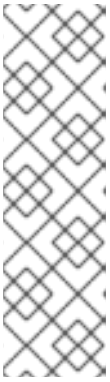
9.1. ライブマイグレーションについて

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。

9.1.1. 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。

9.1.2. 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設定を設定します。
  - 制限とタイムアウト
  - ノードまたはクラスターごとの最大移行数
  - 既存のネットワークから専用のライブマイグレーションネットワークを選択する
- ライブマイグレーションの開始とキャンセル
- 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
- 仮想マシン移行メトリクスの表示

9.1.3. 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

9.2. 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。

9.2.1. ライブマイグレーション設定

次のライブマイグレーション設定を設定できます。

- 制限とタイムアウト
- ノードまたはクラスターごとの最大移行数

9.2.1.1. 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。

9.2.2. 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

Web コンソール を使用してライブマイグレーションポリシーを作成できます。

9.2.2.1. 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。ライブマイグレーションポリシーは、ラベルの任意の組み合わせを使用して、選択した仮想マシンに適用されます。

- **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. 次の例のように **MigrationPolicy** オブジェクトを作成します。

```
apiVersion: migrations.kubevirt.io/v1alpha1
kind: MigrationPolicy
metadata:
```

```
  name: <migration_policy>
spec:
  selectors:
    namespaceSelector: 1
      hpc-workloads: "True"
      xyz-workloads-type: ""
    virtualMachineInstanceSelector: 2
      workload-type: "db"
      operating-system: ""
```

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。

2. 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
```

9.2.3. 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

9.3. ライブマイグレーションの開始とキャンセル

OpenShift Dedicated 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。

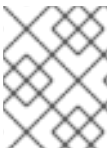
ヒント

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。

9.3.1. ライブマイグレーションの開始

9.3.1.1. 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 Dedicated Web コンソールを使用して、実行中の仮想マシン(VM)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。

9.3.1.2. 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
```

9.3.2. ライブマイグレーションのキャンセル

9.3.2.1. 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 Dedicated Web コンソールを使用して、仮想マシン(VM)のライブマイグレーションを取り消すことができます。

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。

9.3.2.2. 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
```

9.3.3. 関連情報

- Web コンソールを使用したすべてのライブマイグレーションの進行状況の監視
- Web コンソールを使用した仮想マシン移行メトリクスの表示

第10章 ノード

10.1. 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。OpenShift Dedicated 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、メンテナンスについて、詳しくは 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。

10.1.1. エビクションストラテジー

ノードをメンテナンス状態にすると、そのノードはスケジュール不能としてマークされ、そこからすべての仮想マシンと Pod が排出されます。

仮想マシンまたはクラスターに対してエビクション戦略を設定できます。

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。このエビクション戦略を持つ VMI は、別のノードにライブマイグレーションされます。
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戦略は **LiveMigrate** です。**LiveMigrate** エビクション戦略を使用する移行不可能な仮想マシンは、仮想マシンがノードからエビクションされないため、ノードのドレインを妨げたり、インフラストラクチャーのアップグレードをブロックしたりする可能性があります。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戦略を、アップグレードをブロックしない **LiveMigrateIfPossible** に設定するか、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。

10.1.1.1. 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戦略は **LiveMigrate** です。**LiveMigrate** エビクション戦略を使用する移行不可能な仮想マシンは、仮想マシンがノードからエビクションされないため、ノードのドレインを妨げたり、インフラストラクチャーのアップグレードをブロックしたりする可能性があります。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戦略を、アップグレードをブロックしない **LiveMigrateIfPossible** に設定するか、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>
```

10.1.2. 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。

10.1.2.1. 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について説明します。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10.1 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。

10.1.2.2. 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
# ...
```

10.1.3. 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についての詳細は、メンテナンス時に提供されます。

10.1.4. 関連情報

- ライブマイグレーションについて

10.2. 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。

10.2.1. 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10.1 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。

10.2.2. 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10.2 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10.3 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10.4 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
```

10.2.3. 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。

10.3. 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。

10.3.1. 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。

10.3.2. 関連情報

- 古い CPU モデルのノードラベルの管理

第11章 MONITORING

11.1. 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。

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。

ランブック

OpenShift Dedicated Web コンソールで OpenShift Virtualization アラート をトリガーする問題を診断し、解決します。

11.2. 仮想リソースの PROMETHEUS クエリー

OpenShift Dedicated モニタリングダッシュボードを使用して仮想化メトリクスをクエリーします。OpenShift Virtualization は、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クスを提供します。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。

11.2.1. 前提条件

- 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。

11.2.2. メトリクスのクエリー

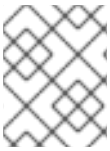
OpenShift Dedicated モニタリングダッシュボードを使用すると、Prometheus Query Language (PromQL) クエリーを実行して、プロット上に視覚化されたメトリクスを調べることができます。この機能により、クラスターの状態と、モニターしているユーザー定義のワークロードに関する情報が提供されます。

dedicated-admin として、ユーザー定義プロジェクトに関するメトリクスに対して、一度に 1 つ以上の namespace 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。

11.2.2.1. 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

dedicated-admin またはすべてのプロジェクトの表示パーミッションを持つユーザーとして、メトリクス UI ですべてのデフォルト OpenShift Dedicated およびユーザー定義プロジェクトのメトリクスにアクセスできます。



注記

専任の管理者のみが、OpenShift Dedicated モニタリングで提供されるサードパーティーの UI にアクセスできます。

前提条件

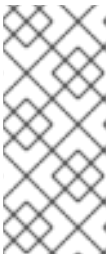
- **dedicated-admin** ロールまたはすべてのプロジェクトの表示パーミッションを持つユーザーとしてクラスターにアクセスできる。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. OpenShift Dedicated Web コンソールの **Administrator** パースペクティブで、**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。
クエリーの実行を無効にします。	オプションメニュー を選択します クエリーの横にある **Disable query** を選択します。

3. 作成したクエリーを実行するには、**Run queries** を選択します。クエリーからのメトリクスはプロットで可視化されます。クエリーが無効な場合は、UI にエラーメッセージが表示されます。



注記

大量のデータで動作するクエリーは、時系列グラフの描画時にタイムアウトするか、ブラウザーをオーバーロードする可能性があります。これを回避するには、**Hide graph** を選択し、メトリクステーブルのみを使用してクエリーを調整します。次に、使用できるクエリーを確認した後に、グラフを描画できるようにプロットを有効にします。

注記

デフォルトでは、クエリーテーブルに、すべてのメトリクスとその現在の値をリスト表示する拡張ビューが表示されます。˅ を選択すると、クエリーの拡張ビューを最小にすることができます。

4. オプション: ページ URL には、実行したクエリーが含まれます。このクエリーのセットを再度使用できるようにするには、この URL を保存します。

5. 視覚化されたメトリクスを調べます。最初に、有効な全クエリーの全メトリクスがプロットに表示されます。次のいずれかを実行して、表示するメトリクスを選択できます。

オプション	説明
クエリーからすべてのメトリクスを非表示にします。	オプションメニューをクリックします  クエリーを選択し、**Hide all series** をクリックします。
特定のメトリクスを非表示にします。	クエリーテーブルに移動し、メトリクス名の近くにある色付きの四角形をクリックします。
プロットを拡大し、時間範囲を変更します。	次のいずれかになります。 ● プロットを水平にクリックし、ドラッグして、時間範囲を視覚的に選択します。 ● 左上隅のメニューを使用して、時間範囲を選択します。
時間範囲をリセットします。	**Reset zoom** を選択します。
特定の時点でのすべてのクエリーの出力を表示します。	その時点でプロット上にマウスカーソルを置きます。クエリーの出力はポップアップに表示されます。
プロットを非表示にします。	**Hide graph** を選択します。

11.2.2.2. 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

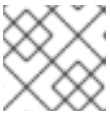
開発者は **Developer** パースペクティブのみを使用でき、**Administrator** パースペクティブは使用できません。開発者は、1度に1つのプロジェクトのメトリクスのみをクエリーできます。開発者は OpenShift Dedicated モニタリングで提供されるサードパーティーの UI にアクセスできません。

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を有効にしている。
- ユーザー定義プロジェクトにサービスをデプロイしている。
- サービスのモニター方法を定義するために、サービスの **ServiceMonitor** カスタムリソース定義 (CRD) を作成している。

手順

1. OpenShift Dedicated 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。

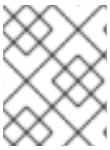
4. 次のいずれかを実行して、視覚化されたメトリクスを調べます。

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。

オプション	説明
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。

11.2.3. 仮想化メトリクス

以下のメトリクスの記述には、Prometheus Query Language (PromQL) クエリーのサンプルが含まれます。これらのメトリクスは API ではなく、バージョン間で変更される可能性があります。



注記

以下の例では、期間を指定する **topk** クエリーを使用します。その期間中に仮想マシンが削除された場合でも、クエリーの出力に依然として表示されます。

11.2.3.1. ネットワークメトリクス

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。

11.2.3.2. ストレージメトリクス

11.2.3.2.1. 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。

11.2.3.2.2. 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。

11.2.3.2.3. 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。

11.2.3.3. 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。

11.2.3.4. 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。

11.2.4. 関連情報

- モニタリングの概要
- Querying Prometheus
- Prometheus クエリーの例

11.3. 仮想マシンのカスタムメトリクスの公開

OpenShift Dedicated には、コアプラットフォームコンポーネントの監視を提供する、事前設定、事前インストールが行われ、自動更新される監視スタックが含まれています。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。

11.3.1. 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 Dedicated CLI (**oc**)をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしている。
- **cluster-monitoring-config ConfigMap** オブジェクトを **openshift-monitoring** プロジェクトに作成する。
- **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
```

11.3.2. 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
```

11.3.3. 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 Dedicated CLI (**oc**)をインストールしている。
- **cluster-admin** 権限を持つユーザーとしてログインしている。
- 仮想マシンを停止および再起動するための Web コンソールへのアクセス権限がある。

手順

1. 仮想マシン設定ファイルの **template** spec を編集します。この例では、ラベル **monitor** の値が **metrics** になります。

```
spec:
  template:
    metadata:
      labels:
        monitor: metrics
```

2. 仮想マシンを停止して再起動し、**monitor** ラベルに指定されたラベル名を持つ新しい Pod を作成します。

11.3.3.1. 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
```

11.3.4. 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
```

11.3.4.1. 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
```

11.3.5. 関連情報

- 設定マップの作成および使用
- 仮想マシンの状態の制御

11.4. 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。

11.4.1. 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。

11.4.1.1. 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
```

11.4.1.2. 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
```

11.4.1.3. 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
```

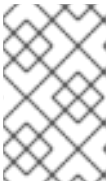
11.4.2. ウォッチドッグの定義

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。

11.4.2.1. 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
```

11.4.2.2. 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
```

11.5. OPENSHIFT VIRTUALIZATION ランブック

これらのランブックの手順を使用して、OpenShift Virtualization の アラート をトリガーする問題を診断および解決できます。

OpenShift Virtualization アラートは、Web コンソールの **Virtualization → Overview →** Overview タブに表示されます。

11.5.1. CDIDataImportCronOutdated

意味
このアラートは、**DataImportCron** が最新のディスクイメージバージョンをポーリングまたはインポートできない場合に発生します。

DataImportCron は、ディスクイメージをポーリングして最新バージョンをチェックし、イメージを永続ボリュームクレーム (PVC) としてインポートします。このプロセスにより、PVC が確実に最新バージョンに更新され、仮想マシン (VM) の信頼できるクローンソースまたはゴールデンイメージとして使用できるようになります。

ゴールデンイメージの場合、**latest** はディストリビューションの最新のオペレーティングシステムを参照します。他のディスクイメージの場合、**latest** は利用可能なイメージの最新のハッシュを参照します。

影響
古いディスクイメージから仮想マシンが作成される場合があります。

クローン作成に使用できるソース PVC がないため、仮想マシンの起動に失敗する場合があります。

診断

1. クラスターのデフォルトストレージクラスを確認します。

```
$ oc get sc
```

出力には、デフォルトのストレージクラスの名前の横に **(default)** が付いたストレージクラスが表示されます。**DataImportCron** がゴールデンイメージをポーリングしてインポートするには、クラスターまたは **DataImportCron** 仕様で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。ストレージクラスが定義されていない場合、DataVolume コントローラーは PVC の作成に失敗し、次のイベントが表示されます: **DataVolume.storage spec is missing accessMode and no storageClass to choose profile**。

2. **DataImportCron** namespace と名前を取得します。

```
$ oc get dataimportcron -A -o json | jq -r '.items[] | \
  select(.status.conditions[] | select(.type == "UpToDate" and \
  .status == "False")) | .metadata.namespace + "/" + .metadata.name'
```

3. クラスターでデフォルトのストレージクラスが定義されていない場合は、**DataImportCron** 仕様でデフォルトのストレージクラスをチェックします。

```
$ oc get dataimportcron <dataimportcron> -o yaml | \
  grep -B 5 storageClassName
```

出力例

```
    url: docker://.../cdi-func-test-tinycore
  storage:
    resources:
      requests:
        storage: 5Gi
  storageClassName: rook-ceph-block
```

4. **DataImportCron** オブジェクトに関連付けられた **DataVolume** の名前を取得します。

```
$ oc -n <namespace> get dataimportcron <dataimportcron> -o json | \
  jq .status.lastImportedPVC.name
```

5. **DataVolume** ログでエラーメッセージを確認します。

```
$ oc -n <namespace> get dv <datavolume> -o yaml
```

6. **CDI_NAMESPACE** 環境変数を設定します。

```
$ export CDI_NAMESPACE="$(oc get deployment -A | \
  grep cdi-operator | awk '{print $1}')"
```

7. **cdi-deployment** ログでエラーメッセージを確認します。

```
$ oc logs -n $CDI_NAMESPACE deployment/cdi-deployment
```

軽減策

1. クラスターまたは **DataImportCron** 仕様でデフォルトのストレージクラスを設定し、ゴールデンイメージをポーリングしてインポートします。更新された Containerized Data Importer (CDI) は、数秒以内に問題を解決します。

2. 問題が解決しない場合は、影響を受ける **DataImportCron** オブジェクトに関連付けられているデータボリュームを削除します。CDI は、デフォルトのストレージクラスでデータボリュームを再作成します。

3. クラスターが制限されたネットワーク環境にインストールされている場合は、自動更新をオプトアウトするために **enableCommonBootImageImport** フィーチャーゲートを無効にします。

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

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.2. CDIDataVolumeUnusualRestartCount

意味
このアラートは、**DataVolume** オブジェクトが 3 回以上再起動した場合に発生します。

影響
データボリュームは、永続ボリューム要求での仮想マシンディスクのインポートと作成を担当します。データボリュームの再起動が 3 回を超えると、これらの操作が成功する可能性は低くなります。問題を診断して解決する必要があります。

診断

1. 3 回以上再起動した Containerized Data Importer (CDI) Pod を検索します。

```
$ oc get pods --all-namespaces -l app=containerized-data-importer -o=jsonpath='{range
.items[?(@.status.containerStatuses[0].restartCount>3)]}{.metadata.name}{"/"}
{.metadata.namespace}{"\n"}'
```

2. Pod の詳細を取得します。

```
$ oc -n <namespace> describe pods <pod>
```

3. Pod のログでエラーメッセージをチェックします。

```
$ oc -n <namespace> logs <pod>
```

軽減策
データボリュームを削除し、問題を解決して、新しいデータボリュームを作成します。

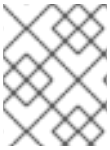
問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.3. CDIDefaultStorageClassDegraded

意味
このアラートは、スマートクローン作成 (CSI かスナップショットベース) または ReadWriteMany アクセスモードをサポートするデフォルトのストレージクラスがない場合に発生します。

影響
デフォルトのストレージクラスがスマートクローン作成をサポートしていない場合、デフォルトのクローン作成方法はホスト支援によるクローン作成となり、効率が大幅に低下します。

デフォルトのストレージクラスが ReadWriteMany をサポートしていない場合、仮想マシン (VM) はライブマイグレーションできません。



注記

デフォルトの OpenShift Virtualization ストレージクラスは、VM ディスクの作成時にデフォルトの OpenShift Dedicated ストレージクラスよりも優先されます。

診断

1. 次のコマンドを実行して、デフォルトの OpenShift Virtualization ストレージクラスを取得します。

```
$ oc get sc -o jsonpath='{.items[?(@.metadata.annotations.storageclass\.kubevirt\.io/is-default-virt-class=="true")].metadata.name}'
```

2. デフォルトの OpenShift Virtualization ストレージクラスが存在する場合は、次のコマンドを実行して、それが ReadWriteMany をサポートしていることを確認します。

```
$ oc get storageprofile <storage_class> -o json | jq '.status.claimPropertySets'| grep ReadWriteMany
```

3. OpenShift Virtualization ストレージクラスがない場合は、以下のコマンドを実行してデフォルトの OpenShift Dedicated ストレージクラスを取得します。

```
$ oc get sc -o jsonpath='{.items[?(@.metadata.annotations.storageclass\.kubevirt\.io/is-default-class=="true")].metadata.name}'
```

4. デフォルトの OpenShift Dedicated ストレージクラスが存在する場合は、次のコマンドを実行して ReadWriteMany をサポートしていることを確認します。

```
$ oc get storageprofile <storage_class> -o json | jq '.status.claimPropertySets'| grep ReadWriteMany
```

軽減策
デフォルトのストレージクラス (OpenShift Container Platform または OpenShift Virtualization) があり、そのデフォルトストレージクラスがスマートクローン作成と ReadWriteMany をサポートしていることを確認します。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.4. CDIMultipleDefaultVirtStorageClasses

意味
このアラートは、複数のストレージクラスにアノテーション **storageclass.kubevirt.io/is-default-virt-class: "true"** がある場合に発生します。

影響
storageclass.kubevirt.io/is-default-virt-class: "true" アノテーションは、デフォルトの OpenShift Virtualization ストレージクラスを定義します。

複数のデフォルトの OpenShift Virtualization ストレージクラスが定義されている場合、ストレージクラスが指定されていないデータボリュームは、最後に作成されたデフォルトのストレージクラスを受け取ります。

診断
次のコマンドを実行して、デフォルトの OpenShift Virtualization ストレージクラスのリストを取得します。

```
$ oc get sc -o jsonpath='{.items[?(@.metadata.annotations.storageclass\.kubevirt\.io/is-default-virt-class=="true")].metadata.name}'
```

軽減策

他のストレージクラスからアノテーションを削除して、デフォルトの OpenShift Virtualization ストレージクラスが 1 つだけ定義されていることを確認します。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.5. CDINoDefaultStorageClass

意味
このアラートは、デフォルトの OpenShift Dedicated または OpenShift Virtualization ストレージクラスが定義されていない場合に発生します。

影響
OpenShift Container Platform または OpenShift Virtualization のデフォルトストレージクラスが定義されていない場合、デフォルトストレージクラスを要求するデータボリューム (ストレージクラスが指定されていない) は保留状態のままになります。

診断

1. 以下のコマンドを実行して、デフォルトの OpenShift Dedicated ストレージクラスを確認します。

```
$ oc get sc -o jsonpath='{.items[?(@.metadata.annotations.storageclass\.kubevirt\.io/is-default-class=="true")].metadata.name}'
```

2. 次のコマンドを実行して、デフォルトの OpenShift Virtualization ストレージクラスを確認します。

```
$ oc get sc -o jsonpath='{.items[?(@.metadata.annotations.storageclass\.kubevirt\.io/is-default-virt-class=="true")].metadata.name}'
```

軽減策
OpenShift Dedicated または OpenShift Virtualization のいずれか、または両方用のデフォルトストレージクラスを作成します。

仮想マシンディスクイメージの作成において、デフォルトの OpenShift Virtualization ストレージクラスは、デフォルトの OpenShift Container Platform ストレージクラスよりも優先されます。

- 次のコマンドを実行して、デフォルトの OpenShift Dedicated ストレージクラスを作成します。

```
$ oc patch storageclass <storage-class-name> -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

- 次のコマンドを実行して、デフォルトの OpenShift Virtualization ストレージクラスを作成します。

```
$ oc patch storageclass <storage-class-name> -p '{"metadata": {"annotations": {"storageclass.kubevirt.io/is-default-virt-class":"true"}}}'
```

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.6. CDINotReady

意味
このアラートは、Containerized Data Importer (CDI) がデグレード状態にある場合に発生します。

- 進行中でない
- 使用不可

影響
CDI は使用できないため、ユーザーは CDI のデータボリュームを使用して永続ボリュームクレーム (PVC) に仮想マシンディスクをビルドできません。CDI コンポーネントの準備ができておらず、準備完了状態への進行が停止しました。

診断

1. **CDI_NAMESPACE** 環境変数を設定します。

```
$ export CDI_NAMESPACE="$(oc get deployment -A | \
  grep cdi-operator | awk '{print $1}')"
```

2. 準備ができていないコンポーネントの CDI デプロイメントをチェックします。

```
$ oc -n $CDI_NAMESPACE get deploy -l cdi.kubevirt.io
```

3. 失敗した Pod の詳細をチェックします。

```
$ oc -n $CDI_NAMESPACE describe pods <pod>
```

4. 失敗した Pod のログをチェックします。

```
$ oc -n $CDI_NAMESPACE logs <pod>
```

軽減策
根本原因の特定と問題の解決を試みてください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.7. CDIOperatorDown

意味
このアラートは、Containerized Data Importer (CDI) Operator がダウンしたときに発生します。CDI Operator は、データボリュームや永続ボリューム要求 (PVC) コントローラーなどの CDI インフラストラクチャーコンポーネントをデプロイおよび管理します。これらのコントローラーは、ユーザーが PVC 上に仮想マシンディスクをビルドするのに役立ちます。

影響
CDI コンポーネントがデプロイに失敗するか、必要な状態を維持できない可能性があります。CDI のインストールが正しく機能しない可能性があります。

診断

1. **CDI_NAMESPACE** 環境変数を設定します。

```
$ export CDI_NAMESPACE="$(oc get deployment -A | grep cdi-operator | \
  awk '{print $1}')"
```

2. **cdi-operator** Pod が現在実行されているかどうかを確認します。

```
$ oc -n $CDI_NAMESPACE get pods -l name=cdi-operator
```

3. **cdi-operator** Pod の詳細を取得します。

```
$ oc -n $CDI_NAMESPACE describe pods -l name=cdi-operator
```

4. **cdi-operator** Pod のログでエラーをチェックします。

```
$ oc -n $CDI_NAMESPACE logs -l name=cdi-operator
```

軽減策
問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.8. CDIStorageProfilesIncomplete

意味
このアラートは、Containerized Data Importer (CDI) ストレージプロファイルが不完全な場合に発生します。

ストレージプロファイルが不完全な場合、CDI は、仮想マシン (VM) ディスクの作成に必要な **volumeMode** や **accessModes** などの永続ボリューム要求 (PVC) フィールドを推測できません。

影響
CDI は PVC 上に仮想マシンディスクを作成できません。

診断

- 不完全なストレージプロファイルを特定します。

```
$ oc get storageprofile <storage_class>
```

軽減策

- 次の例のように、不足しているストレージプロファイル情報を追加します。

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

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.9. CnaoDown

意味
このアラートは、Cluster Network Addons Operator (CNAO) がダウンしたときに発生します。CNAO は、追加のネットワークコンポーネントをクラスターの上にデプロイします。

影響
CNAO が実行されていない場合、クラスターは仮想マシンコンポーネントへの変更を調整できません。その結果、変更が有効にならない場合があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get deployment -A | \
  grep cluster-network-addons-operator | awk '{print $1}')"
```

2. **cluster-network-addons-operator** Pod のステータスをチェックします。

```
$ oc -n $NAMESPACE get pods -l name=cluster-network-addons-operator
```

3. **cluster-network-addons-operator** ログでエラーメッセージをチェックします。

```
$ oc -n $NAMESPACE logs -l name=cluster-network-addons-operator
```

4. **cluster-network-addons-operator** Pod の詳細を取得します。

```
$ oc -n $NAMESPACE describe pods -l name=cluster-network-addons-operator
```

軽減策
問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.10. HCOInstallationIncomplete

意味
このアラートは、HyperConverged Cluster Operator (HCO) が **HyperConverged** カスタムリソース (CR) なしで 1 時間以上実行された場合に発生します。

このアラートには次の原因があります。

- インストールプロセス中に HCO をインストールしましたが、**HyperConverged** CR を作成していません。
- アンインストールプロセス中に、HCO をアンインストールする前に **HyperConverged** CR を削除しましたが、HCO はまだ実行されています。

軽減策
軽減策は、HCO をインストールするかアンインストールするかによって異なります。

- **HyperConverged** CR をデフォルト値で作成して、インストールを完了します。

```
$ cat <<EOF | oc apply -f -
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: hco-operatorgroup
  namespace: kubevirt-hyperconverged
spec: {}
EOF
```

- HCO をアンインストールします。アンインストールプロセスが引き続き実行される場合は、アラートをキャンセルするためにその問題を解決する必要があります。

11.5.11. HPPNotReady

意味
このアラートは、ホストパスプロビジョナー (HPP) のインストールが劣化状態にある場合に発生します。

HPP は、ホストパスボリュームを動的にプロビジョニングして、永続ボリューム要求 (PVC) 用のストレージを提供します。

影響
HPP は使用できません。そのコンポーネントは準備ができておらず、準備完了状態に向かって進んでいません。

診断

1. **HPP_NAMESPACE** 環境変数を設定します。

```
$ export HPP_NAMESPACE="$(oc get deployment -A | \
  grep hostpath-provisioner-operator | awk '{print $1}')"
```

2. 現在準備が整っていない HPP コンポーネントを確認します。

```
$ oc -n $HPP_NAMESPACE get all -l k8s-app=hostpath-provisioner
```

3. 失敗した Pod の詳細を取得します。

```
$ oc -n $HPP_NAMESPACE describe pods <pod>
```

4. 失敗した Pod のログをチェックします。

```
$ oc -n $HPP_NAMESPACE logs <pod>
```

軽減策
診断手順で得られた情報に基づいて、根本原因の特定と問題の解決を試みます。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.12. HPPOperatorDown

意味
このアラートは、ホストパスプロビジョナー (HPP) オペレーターがダウンしたときに発生します。

HPP Operator は、ホストパスボリュームをプロビジョニングするデーモンセットなど、HPP インフラストラクチャーコンポーネントをデプロイおよび管理します。

影響
HPP コンポーネントがデプロイに失敗するか、必要な状態のままになる可能性があります。その結果、HPP のインストールがクラスターで正しく機能しない可能性があります。

診断

1. **HPP_NAMESPACE** 環境変数を設定します。

```
$ HPP_NAMESPACE="$(oc get deployment -A | grep \
  hostpath-provisioner-operator | awk '{print $1}')"
```

2. **hostpath-provisioner-operator** Pod が現在実行中かどうかをチェックします。

```
$ oc -n $HPP_NAMESPACE get pods -l name=hostpath-provisioner-operator
```

3. **hostpath-provisioner-operator** Pod の詳細を取得します。

```
$ oc -n $HPP_NAMESPACE describe pods -l name=hostpath-provisioner-operator
```

4. **hostpath-provisioner-operator** Pod のログでエラーをチェックします。

```
$ oc -n $HPP_NAMESPACE logs -l name=hostpath-provisioner-operator
```

軽減策
診断手順で得られた情報に基づいて、根本原因の特定と問題の解決を試みます。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.13. HPPSharingPoolPathWithOS

意味
このアラートは、ホストパスプロビジョナー (HPP) がファイルシステムを **kubelet** やオペレーティングシステム (OS) などの他の重要なコンポーネントと共有している場合に発生します。

HPP は、ホストパスボリュームを動的にプロビジョニングして、永続ボリューム要求 (PVC) 用のストレージを提供します。

影響
共有ホストパスプールは、ノードのディスクに圧力をかけます。ノードのパフォーマンスと安定性が低下している可能性があります。

診断

1. **HPP_NAMESPACE** 環境変数を設定します。

```
$ export HPP_NAMESPACE="$(oc get deployment -A | \
  grep hostpath-provisioner-operator | awk '{print $1}')"
```

2. **hostpath-provisioner-csi** デーモンセット Pod のステータスを取得します。

```
$ oc -n $HPP_NAMESPACE get pods | grep hostpath-provisioner-csi
```

3. **hostpath-provisioner-csi** ログを確認して、共有プールとパスを特定します。

```
$ oc -n $HPP_NAMESPACE logs <csi_daemonset> -c hostpath-provisioner
```

出力例

```
I0208 15:21:03.769731       1 utils.go:221] pool (<legacy, csi-data-dir>/csi),
shares path with OS which can lead to node disk pressure
```

-

軽減策
診断セクションで取得したデータを使用して、プールパスが OS と共有されないようにします。具体的な手順は、ノードやその他の状況によって異なります。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.14. KubemacpoolDown

意味
KubeMacPool がダウンしています。**KubeMacPool** は、MAC アドレスの割り当てと MAC アドレスの競合の防止を担当します。

影響
KubeMacPool がダウンしている場合、**VirtualMachine** オブジェクトを作成できません。

診断

1. **KMP_NAMESPACE** 環境変数を設定します。

```
$ export KMP_NAMESPACE="$(oc get pod -A --no-headers -l \
  control-plane=mac-controller-manager | awk '{print $1}')"
```

2. **KMP_NAME** 環境変数を設定します。

```
$ export KMP_NAME="$(oc get pod -A --no-headers -l \
  control-plane=mac-controller-manager | awk '{print $2}')"
```

3. **KubeMacPool-manager** Pod の詳細を取得します。

```
$ oc describe pod -n $KMP_NAMESPACE $KMP_NAME
```

4. **KubeMacPool-manager** ログでエラーメッセージを確認します。

```
$ oc logs -n $KMP_NAMESPACE $KMP_NAME
```

軽減策
問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.15. KubeMacPoolDuplicateMacsFound

意味
このアラートは、**KubeMacPool** が重複した MAC アドレスを検出したときに発生します。

KubeMacPool は、MAC アドレスの割り当てと MAC アドレスの競合の防止を担当します。**KubeMacPool** が起動すると、クラスターをスキャンして、管理された namespace 内の仮想マシン (VM) の MAC アドレスを探します。

影響
同じ LAN で MAC アドレスが重複していると、ネットワークの問題が発生する可能性があります。

診断

1. namespace と **kubemacpool-mac-controller** Pod の名前を取得します。

```
$ oc get pod -A -l control-plane=mac-controller-manager --no-headers \
  -o custom-columns=":metadata.namespace,:metadata.name"
```

2. **kubemacpool-mac-controller** ログから重複する MAC アドレスを取得します。

```
$ oc logs -n <namespace> <kubemacpool_mac_controller> | \
  grep "already allocated"
```

出力例

```
mac address 02:00:ff:ff:ff:ff already allocated to
vm/kubemacpool-test/testvm, br1,
conflict with: vm/kubemacpool-test/testvm2, br1
```

軽減策

1. 仮想マシンを更新して、重複する MAC アドレスを削除します。
2. **kubemacpool-mac-controller** Pod を再起動します。

```
$ oc delete pod -n <namespace> <kubemacpool_mac_controller>
```

11.5.16. KubeVirtComponentExceedsRequestedCPU

意味
このアラートは、コンポーネントの CPU 使用率が要求された制限を超えたときに発生します。

影響
CPU リソースの使用率が最適ではなく、ノードが過負荷になっている可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. コンポーネントの CPU リクエスト制限を確認します。

```
$ oc -n $NAMESPACE get deployment <component> -o yaml | grep requests: -A 2
```

3. PromQL クエリーを使用して、実際の CPU 使用率を確認します。

```
node_namespace_pod_container:container_cpu_usage_seconds_total:sum_rate
{namespace="$NAMESPACE",container="<component>"}
```

詳細については、Prometheus のドキュメント を参照してください。

軽減策
HCO カスタムリソースの CPU リクエスト制限を更新します。

11.5.17. KubeVirtComponentExceedsRequestedMemory

意味
このアラートは、コンポーネントのメモリー使用率が要求された制限を超えたときに発生します。

影響
メモリーリソースの使用が最適化されておらず、ノードが過負荷になっている可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. コンポーネントのメモリー要求制限を確認します。

```
$ oc -n $NAMESPACE get deployment <component> -o yaml | \
  grep requests: -A 2
```

3. PromQL クエリーを使用して、実際のメモリー使用率を確認します。

```
container_memory_usage_bytes{namespace="$NAMESPACE",container="<component>"}
```

詳細については、Prometheus のドキュメント を参照してください。

軽減策
HCO カスタムリソースのメモリー要求制限を更新します。

11.5.18. KubeVirtCRModified

意味
このアラートは、HyperConverged Cluster Operator (HCO) のオペランドが HCO 以外の誰かまたは何かによって変更されたときに発生します。

HCO は、OpenShift Virtualization とそのサポートオペレーターを独自の方法で設定し、予期しない変更があった場合にそのオペランドを上書きします。ユーザーは、オペランドを直接変更してはなりません。**HyperConverged** カスタムリソースは、設定の信頼できる情報源です。

影響
オペランドを手動で変更すると、クラスター設定が変動し、不安定になる可能性があります。

診断

- アラートの詳細で **component_name** の値を確認して、変更されているオペランドの種類 (**kubevirt**) とオペランド名 (**kubevirt-kubevirt-hyperconverged**) を特定します。

```
Labels
  alertname=KubevirtHyperconvergedClusterOperatorCRModification
  component_name=kubevirt/kubevirt-kubevirt-hyperconverged
  severity=warning
```

軽減策
HCO オペランドを直接変更しないでください。**HyperConverged** オブジェクトを使用してクラスターを設定します。

オペランドが手動で変更されていない場合、アラートは 10 分後に自動的に解決されます。

11.5.19. KubeVirtDeprecatedAPIRequested

意味
このアラートは、非推奨の **KubeVirt** API が使用されている場合に発生します。

影響
将来のリリースで API が削除されるとリクエストが失敗するため、非推奨の API の使用は推奨されません。

診断

- 次の例のように、アラートの Description セクションと Summary セクションを確認して、非推奨の API を特定します。
  Description

  非推奨の virtualmachines.kubevirt.io/v1alpha3 API へのリクエストが検出されました。

  Summary

  過去 10 分間に 2 つのリクエストが検出されました。

軽減策
完全にサポートされている API を使用します。非推奨の API が使用されていない場合、アラートは 10 分後に自動的に解決されます。

11.5.20. KubeVirtNoAvailableNodesToRunVMs

意味
このアラートは、クラスター内のノード CPU が仮想化をサポートしていない場合、または仮想化拡張機能が有効になっていない場合に発生します。

影響
仮想マシンを実行するには、ノードが仮想化をサポートし、BIOS で仮想化機能が有効になっている必要があります。

診断

- ノードでハードウェア仮想化がサポートされているかどうかを確認します。

```
$ oc get nodes -o json|jq '.items[]|{"name": .metadata.name, "kvm":
.status.allocatable["devices.kubevirt.io/kvm"]}'
```

出力例

```
{
  "name": "shift-vwpsz-master-0",
  "kvm": null
}
{
  "name": "shift-vwpsz-master-1",
  "kvm": null
}
{
  "name": "shift-vwpsz-master-2",
```

```
  "kvm": null
}
{
  "name": "shift-vwpsz-worker-8bxkp",
  "kvm": "1k"
}
{
  "name": "shift-vwpsz-worker-ctgmc",
  "kvm": "1k"
}
{
  "name": "shift-vwpsz-worker-gl5zl",
  "kvm": "1k"
}
```

"kvm": null または **"kvm": 0** のノードは仮想化拡張機能をサポートしません。

"kvm": "1k" のノードは仮想化拡張機能をサポートします。

軽減策
ハードウェアおよび CPU 仮想化拡張機能がすべてのノードで有効になっていること、およびノードに正しくラベルが付けられていることを確認してください。

詳細は、OpenShift Virtualization reports no nodes are available, cannot start VMs を参照してください。

問題を解決できない場合は、カスタマーポータル にログインし、サポートケースをオープンしてください。

11.5.21. KubevirtVmHighMemoryUsage

意味
このアラートは、仮想マシン (VM) をホストするコンテナーの空きメモリーが 20 MB 未満の場合に発生します。

影響
コンテナーのメモリー制限を超えると、コンテナー内で実行されている仮想マシンはランタイムによって終了されます。

診断

1. **virt-launcher** Pod の詳細を取得します。

```
$ oc get pod <virt-launcher> -o yaml
```

2. **virt-launcher** Pod でメモリー使用率が多い **compute** コンテナープロセスを特定します。

```
$ oc exec -it <virt-launcher> -c compute -- top
```

軽減策

- 次の例のように、**VirtualMachine** 仕様のメモリー制限を増やします。

```
spec:
  running: false
```

```
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-name
    spec:
      domain:
        resources:
          limits:
            memory: 200Mi
          requests:
            memory: 128Mi
```

11.5.22. KubeVirtVMIExcessiveMigrations

意味
このアラートは、仮想マシンインスタンス (VMI) のライブマイグレーションが 24 時間に 12 回を超えた場合に発生します。

この移行率は、アップグレード中でも異常に高くなっています。このアラートは、ネットワークの中断やリソース不足など、クラスターインフラストラクチャーの問題を示している可能性があります。

影響
頻繁に移行する仮想マシン (VM) では、移行中にメモリーページフォールトが発生するため、パフォーマンスが低下する可能性があります。

診断

1. ワーカーノードに十分なリソースがあることを確認します。

```
$ oc get nodes -l node-role.kubernetes.io/worker= -o json | \
  jq .items[].status.allocatable
```

出力例

```
{
  "cpu": "3500m",
  "devices.kubevirt.io/kvm": "1k",
  "devices.kubevirt.io/sev": "0",
  "devices.kubevirt.io/tun": "1k",
  "devices.kubevirt.io/vhost-net": "1k",
  "ephemeral-storage": "38161122446",
  "hugepages-1Gi": "0",
  "hugepages-2Mi": "0",
  "memory": "7000128Ki",
  "pods": "250"
}
```

2. ワーカーノードのステータスを確認します。

```
$ oc get nodes -l node-role.kubernetes.io/worker= -o json | \
  jq .items[].status.conditions
```

出力例

```
{
  "lastHeartbeatTime": "2022-05-26T07:36:01Z",
  "lastTransitionTime": "2022-05-23T08:12:02Z",
  "message": "kubelet has sufficient memory available",
  "reason": "KubeletHasSufficientMemory",
  "status": "False",
  "type": "MemoryPressure"
},
{
  "lastHeartbeatTime": "2022-05-26T07:36:01Z",
  "lastTransitionTime": "2022-05-23T08:12:02Z",
  "message": "kubelet has no disk pressure",
  "reason": "KubeletHasNoDiskPressure",
  "status": "False",
  "type": "DiskPressure"
},
{
  "lastHeartbeatTime": "2022-05-26T07:36:01Z",
  "lastTransitionTime": "2022-05-23T08:12:02Z",
  "message": "kubelet has sufficient PID available",
  "reason": "KubeletHasSufficientPID",
  "status": "False",
  "type": "PIDPressure"
},
{
  "lastHeartbeatTime": "2022-05-26T07:36:01Z",
  "lastTransitionTime": "2022-05-23T08:24:15Z",
  "message": "kubelet is posting ready status",
  "reason": "KubeletReady",
  "status": "True",
  "type": "Ready"
}
```


3. ワーカーノードにログインし、**kubelet** サービスが実行されていることを確認します。

```
$ systemctl status kubelet
```

4. **kubelet** ジャーナルログでエラーメッセージを確認します。

```
$ journalctl -r -u kubelet
```

軽減策
VM ワークロードを中断することなく実行するのに十分なリソース (CPU、メモリー、ディスク) がワーカーノードにあることを確認します。

問題が解決しない場合は、根本原因を特定して問題を解決してください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.23. LowKVMNodesCount

意味
このアラートは、クラスター内の 2 つ未満のノードに KVM リソースがある場合に発生します。

影響
クラスターには、ライブマイグレーション用の KVM リソースを備えた少なくとも 2 つのノードが必要です。

どのノードにも KVM リソースがない場合、仮想マシンをスケジュールまたは実行することはできません。

診断

- KVM リソースを持つノードを特定します。

```
$ oc get nodes -o jsonpath='{.items[*].status.allocatable}' | \
  grep devices.kubevirt.io/kvm
```

軽減策
KVM リソースのないノードに KVM をインストールします。

11.5.24. LowReadyVirtControllersCount

意味
このアラートは、1 つ以上の **virt-controller** Pod が実行されているが、これらの Pod が過去 5 分間 **Ready** 状態になかった場合に発生します。

virt-controller デバイスは、仮想マシンインスタンス (VMI) のカスタムリソース定義 (CRD) をモニターし、関連する Pod を管理します。デバイスは VMI の Pod を作成し、そのライフサイクルを管理します。このデバイスは、クラスター全体の仮想化機能にとって重要です。

影響
このアラートは、クラスターレベルの障害が発生する可能性があることを示します。新しい VMI の起動や既存の VMI のシャットダウンなど、VM のライフサイクル管理に関連するアクションは失敗します。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-controller** デバイスが利用可能であることを確認します。

```
$ oc get deployment -n $NAMESPACE virt-controller \
  -o jsonpath='{.status.readyReplicas}'
```

3. **virt-controller** デプロイメントのステータスを確認します。

```
$ oc -n $NAMESPACE get deploy virt-controller -o yaml
```

4. **virt-controller** デプロイメントの詳細を取得して、Pod のクラッシュやイメージのプルの失敗などのステータス条件を確認します。

```
$ oc -n $NAMESPACE describe deploy virt-controller
```

5. ノードに問題が発生していないか確認してください。たとえば、**NotReady** 状態にある可能性があります。

```
$ oc get nodes
```

軽減策
このアラートには、次のような複数の原因が考えられます。

- クラスターのメモリーが不足しています。
- ノードがダウンしています。
- API サーバーが過負荷になっています。たとえば、スケジューラーの負荷が高く、完全には使用できない場合があります。
- ネットワークの問題があります。

根本原因の特定と問題の解決を試みてください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.25. LowReadyVirtOperatorsCount

意味
このアラートは、1 つ以上の **virt-operator** Pod が実行されているときに発生しますが、これらの Pod はいずれも過去 10 分間 **Ready** 状態ではありません。

virt-operator は、クラスターで開始する最初の Operator です。**virt-operator** デプロイメントには、2 つの **virt-operator** Pod のデフォルトのレプリカがあります。

その主な責任には、次のものが含まれます。

- クラスターのインストール、ライブ更新、およびライブアップグレード
- **virt-controller**、**virt-handler**、**virt-launcher** などの最上位コントローラーのライフサイクルをモニターし、それらの調整を管理する
- 証明書のローテーションやインフラストラクチャー管理など、特定のクラスター全体のタスク

影響
クラスターレベルの障害が発生する可能性があります。証明書のローテーション、アップグレード、およびコントローラーの調整などの重要なクラスター全体の管理機能は利用できなくなる可能性があります。このような状態でも、**NoReadyVirtOperator** アラートがトリガーされます。

virt-operator には、クラスター内の仮想マシンに対する直接の責任はありません。したがって、一時的に利用できなくても仮想マシンのワークロードに大きな影響はありません。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-operator** デプロイメントの名前を取得します。

```
$ oc -n $NAMESPACE get deploy virt-operator -o yaml
```

3. **virt-operator** デプロイメントの詳細を取得します。

```
$ oc -n $NAMESPACE describe deploy virt-operator
```

4. **NotReady** 状態などのノードの問題をチェックします。

```
$ oc get nodes
```

軽減策
診断手順で得られた情報に基づいて、根本原因の特定と問題の解決を試みます。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.26. LowVirtAPICount

意味
このアラートは、スケジューリングに少なくとも 2 つのノードが使用可能であるにもかかわらず、60 分間に使用可能な **virt-api** Pod が 1 つしか検出されない場合に発生します。

影響
virt-api Pod が単一障害点になるため、ノードの削除中に API 呼び出しの停止が発生する可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. 使用可能な **virt-api** Pod の数を確認します。

```
$ oc get deployment -n $NAMESPACE virt-api \
  -o jsonpath='{.status.readyReplicas}'
```

3. エラー条件について、**virt-api** デプロイメントのステータスを確認します。

```
$ oc -n $NAMESPACE get deploy virt-api -o yaml
```

4. **NotReady** 状態のノードなどの問題がないかノードを確認します。

```
$ oc get nodes
```

軽減策
根本原因の特定と問題の解決を試みてください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.27. LowVirtControllersCount

意味

このアラートは、検出された **virt-controller** Pod の数が少ない場合に発生します。高可用性を確保するには、少なくとも 1 つの **virt-controller** Pod が利用可能である必要があります。レプリカのデフォルト数は、2 です。

virt-controller デバイスは、仮想マシンインスタンス (VMI) のカスタムリソース定義 (CRD) をモニターし、関連する Pod を管理します。デバイスは VMI の Pod を作成し、Pod のライフサイクルを管理します。このデバイスは、クラスター全体の仮想化機能にとって重要です。

影響
OpenShift Virtualization の応答性に悪影響が及ぶ可能性があります。たとえば、特定のリクエストが見逃される場合があります。

さらに、別の **virt-launcher** インスタンスが予期せず終了した場合、OpenShift Virtualization が完全に応答しなくなる可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. 実行 **中の virt-controller** Pod が利用可能であることを確認します。

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-controller
```

3. **virt-launcher** ログでエラーメッセージを確認します。

```
$ oc -n $NAMESPACE logs <virt-launcher>
```

4. **virt-launcher** Pod の詳細を取得して、予期しない終了や **NotReady** 状態などのステータス条件を確認します。

```
$ oc -n $NAMESPACE describe pod/<virt-launcher>
```

軽減策
このアラートには、次のようなさまざまな原因が考えられます。

- クラスターに十分なメモリーがない
- ノードがダウンしている
- API サーバーが過負荷になっています。たとえば、スケジューラーの負荷が高く、完全には使用できない場合があります。
- ネットワークの問題

根本原因を特定し、可能であれば修正します。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.28. LowVirtOperatorCount

意味

このアラートは、**Ready** 状態の **virt-operator** Pod が 1 つだけ過去 60 分間実行されている場合に発生します。

virt-operator は、クラスターで開始する最初の Operator です。その主な責任には、次のものが含まれます。

- クラスターのインストール、ライブ更新、およびライブアップグレード
- **virt-controller**、**virt-handler**、**virt-launcher** などの最上位コントローラーのライフサイクルをモニターし、それらの調整を管理する
- 証明書のローテーションやインフラストラクチャー管理など、特定のクラスター全体のタスク

影響
virt-operator は、デプロイメントに高可用性 (HA) を提供できません。HA には、**Ready** 状態の **virt-operator Pod** が 2 つ以上必要です。デフォルトのデプロイは 2 つの Pod です。

virt-operator には、クラスター内の仮想マシンに対する直接の責任はありません。したがって、可用性の低下が VM のワークロードに大きな影響を与えることはありません。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-operator Pod** の状態を確認します。

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-operator
```

3. 影響を受ける **virt-operator** Pod のログを確認します。

```
$ oc -n $NAMESPACE logs <virt-operator>
```

4. 影響を受ける **virt-operator** Pod の詳細を取得します。

```
$ oc -n $NAMESPACE describe pod <virt-operator>
```

軽減策
診断手順で得られた情報に基づいて、根本原因の特定と問題の解決を試みます。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.29. NetworkAddonsConfigNotReady

意味
このアラートは、Cluster Network Addons Operator (CNAO) の **NetworkAddonsConfig** カスタムリソース (CR) の準備ができていない場合に発生します。

CNAO は、追加のネットワークコンポーネントをクラスターにデプロイします。このアラートは、デプロイメントされたコンポーネントのいずれかの準備ができていないことを示します。

影響

ネットワーク機能が影響を受けます。

診断

1. **NetworkAddonsConfig** CR のステータス条件をチェックして、準備ができていないデプロイメントまたはデーモンセットを特定します。

```
$ oc get networkaddonsconfig \
  -o custom-columns="":.status.conditions[*].message
```

出力例

```
DaemonSet "cluster-network-addons/macvtap-cni" update is being processed...
```

2. コンポーネントの Pod でエラーをチェックします。

```
$ oc -n cluster-network-addons get daemonset <pod> -o yaml
```

3. コンポーネントのログをチェックします。

```
$ oc -n cluster-network-addons logs <pod>
```

4. コンポーネントの詳細でエラー状態をチェックします。

```
$ oc -n cluster-network-addons describe <pod>
```

軽減策
根本原因の特定と問題の解決を試みてください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.30. NoLeadingVirtOperator

意味
このアラートは、**virt-operator** Pod が **Ready** 状態であるにも関わらず、リーダーリースを持つ **virt-operator** Pod が 10 分間検出されない場合に発生します。このアラートは、使用できるリーダー Pod がないことを示しています。

virt-operator は、クラスターで開始する最初の Operator です。その主な責任には、次のものが含まれます。

- クラスターのインストール、ライブ 更新、およびライブアップグレード
- **virt-controller**、**virt-handler**、**virt-launcher** などの最上位コントローラーのライフサイクルをモニターし、それらの調整を管理する
- 証明書のローテーションやインフラストラクチャー管理など、特定のクラスター全体のタスク

virt-operator デプロイメントには、2 つの Pod のデフォルトのレプリカがあり、1 つの Pod がリーダーリースを保持しています。

影響

このアラートは、クラスターレベルでの障害を示します。その結果、証明書のローテーション、アップグレード、コントローラーの調整など、重要なクラスター全体の管理機能が利用できなくなる可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A -o \
  custom-columns="":.metadata.namespace)"
```

2. **virt-operator Pod** のステータスを取得します。

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-operator
```

3. **virt-operator** Pod のログを確認して、リーダーのステータスをチェックします。

```
$ oc -n $NAMESPACE logs | grep lead
```

リーダー Pod の例:

```
{"component":"virt-operator","level":"info","msg":"Attempting to acquire
leader status","pos":"application.go:400","timestamp":"2021-11-30T12:15:18.635387Z"}
I1130 12:15:18.635452       1 leaderelection.go:243] attempting to acquire
leader lease <namespace>/virt-operator...
I1130 12:15:19.216582       1 leaderelection.go:253] successfully acquired
lease <namespace>/virt-operator
{"component":"virt-operator","level":"info","msg":"Started leading",
"pos":"application.go:385","timestamp":"2021-11-30T12:15:19.216836Z"}
```

リーダー以外の Pod の例:

```
{"component":"virt-operator","level":"info","msg":"Attempting to acquire
leader status","pos":"application.go:400","timestamp":"2021-11-30T12:15:20.533696Z"}
I1130 12:15:20.533792       1 leaderelection.go:243] attempting to acquire
leader lease <namespace>/virt-operator...
```

4. 影響を受ける **virt-operator** Pod の詳細を取得します。

```
$ oc -n $NAMESPACE describe pod <virt-operator>
```

軽減策
診断手順で得られた情報に基づいて、根本原因を見つけて問題を解決してください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.31. NoReadyVirtController

意味
このアラートは、使用可能な **virt-controller** デバイスが 5 分間検出されなかった場合に発生します。

virt-controller デバイスは、仮想マシンインスタンス (VMI) のカスタムリソース定義を監視し、関連する Pod を管理します。デバイスは VMI の Pod を作成し、Pod のライフサイクルを管理します。

したがって、**virt-controller** デバイスは、すべてのクラスター全体の仮想化機能にとって重要です。

影響
仮想マシンライフサイクル管理に関連するアクションはすべて失敗します。これには特に、新しい VMI の起動または既存の VMI のシャットダウンが含まれます。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-controller** デバイスの数を検証します。

```
$ oc get deployment -n $NAMESPACE virt-controller \
  -o jsonpath='{.status.readyReplicas}'
```

3. **virt-controller** デプロイメントのステータスを確認します。

```
$ oc -n $NAMESPACE get deploy virt-controller -o yaml
```

4. **virt-controller** デプロイメントの詳細を取得して、Pod のクラッシュやイメージのプルの失敗などのステータス条件をチェックします。

```
$ oc -n $NAMESPACE describe deploy virt-controller
```

5. **virt-controller Pod** の詳細を取得します。

```
$ get pods -n $NAMESPACE | grep virt-controller
```

6. **virt-controller** Pod のログでエラーメッセージをチェックします。

```
$ oc logs -n $NAMESPACE <virt-controller>
```

7. **NotReady** 状態などの問題がないかノードをチェックします。

```
$ oc get nodes
```

軽減策
診断手順で得られた情報に基づいて、根本原因を見つけて問題を解決してください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.32. NoReadyVirtOperator

意味
このアラートは、**Ready** 状態の **virt-operator** Pod が 10 分間検出されなかった場合に発生します。

virt-operator は、クラスターで開始する最初の Operator です。その主な責任には、次のものが含まれます。

- クラスターのインストール、ライブ更新、およびライブアップグレード
- **virt-controller**、**virt-handler**、**virt-launcher** などの最上位コントローラーのライフサイクルを監視し、それらの調整を管理する
- 証明書のローテーションやインフラストラクチャー管理など、特定のクラスター全体のタスク

デフォルトのデプロイメントは 2 つの **virt-operator** Pod です。

影響
このアラートは、クラスターレベルの障害を示します。証明書のローテーション、アップグレード、コントローラーの調整などの重要なクラスター管理機能が利用できない場合があります。

virt-operator には、クラスター内の仮想マシンに対する直接の責任はありません。したがって、一時的に利用できなくてもワークロードに大きな影響はありません。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-operator** デプロイメントの名前を取得します。

```
$ oc -n $NAMESPACE get deploy virt-operator -o yaml
```

3. **virt-operator** デプロイメントの説明を生成します。

```
$ oc -n $NAMESPACE describe deploy virt-operator
```

4. **NotReady** 状態などのノードの問題をチェックします。

```
$ oc get nodes
```

軽減策
診断手順で得られた情報に基づいて、根本原因の特定と問題の解決を試みます。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.33. OrphanedVirtualMachineInstances

意味
このアラートは、仮想マシンインスタンス (VMI) または **virt-launcher** Pod が、実行中の **virt-handler** Pod を持たないノードで実行されている場合に発生します。このような VMI は **orphaned** と呼ばれます。

影響
孤立した VMI は管理できません。

診断

1. **virt-handler Pod** のステータスを確認して、それらが実行されているノードを表示します。

```
$ oc get pods --all-namespaces -o wide -l kubevirt.io=virt-handler
```

2. VMI のステータスを確認して、実行中の **virt-handler** Pod を持たないノードで実行されている VMI を特定します。

```
$ oc get vmis --all-namespaces
```

3. **virt-handler** デーモンのステータスを確認します。

```
$ oc get daemonset virt-handler --all-namespaces
```

出力例

```
NAME         DESIRED  CURRENT  READY  UP-TO-DATE  AVAILABLE ...
virt-handler 2        2        2      2           2         ...
```

Desired、**Ready**、および **Available** 列に同じ値が含まれている場合、デーモンセットは正常であると見なされます。

4. **virt-handler** デーモンセットが正常でない場合は、**virt-handler** デーモンセットで Pod のデプロイの問題を確認します。

```
$ oc get daemonset virt-handler --all-namespaces -o yaml | jq .status
```

5. ノードの **NotReady** ステータスなどの問題を確認します。

```
$ oc get nodes
```

6. ワークロード配置ポリシーについては、**KubeVirt** カスタムリソース (CR) の **spec.workloads** スタンザを確認してください。

```
$ oc get kubevirt kubevirt --all-namespaces -o yaml
```

軽減策
ワークロード配置ポリシーが設定されている場合は、VMI を持つノードをポリシーに追加します。

ノードからの **virt-handler** Pod の削除の考えられる原因には、ノードのテイントと容認、または Pod のスケジューリングルールの変更が含まれます。

根本原因の特定と問題の解決を試みてください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.34. OutdatedVirtualMachineInstanceWorkloads

意味
このアラートは、OpenShift Virtualization コントロールプレーンが更新されてから 24 時間後に、古い **virt-launcher** Pod で実行中の仮想マシンインスタンス (VMI) が検出されたときに発生します。

影響
古い VMI は、新しい OpenShift Virtualization 機能にアクセスできない可能性があります。

古い VMI は、**virt-launcher** Pod の更新に関連するセキュリティー修正を受け取りません。

診断

1. 古い VMI を特定します。

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

2. **KubeVirt** カスタムリソース (CR) をチェックして、**workloadUpdateMethods** が **workloadUpdateStrategy** スタンザで設定されているかどうかを確認します。

```
$ oc get kubevirt --all-namespaces -o yaml
```

3. 古い VMI をそれぞれチェックして、ライブマイグレーション可能かどうかを判断します。

```
$ oc get vmi <vmi> -o yaml
```

出力例

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstance
# ...
  status:
    conditions:
    - lastProbeTime: null
      lastTransitionTime: null
      message: cannot migrate VMI which does not use masquerade
      to connect to the pod network
      reason: InterfaceNotLiveMigratable
      status: "False"
      type: LiveMigratable
```

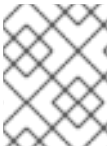
軽減策
ワークロードの自動更新の設定
HyperConverged CR を更新して、ワークロードの自動更新を有効にします。

ライブマイグレーション不可能な VMI に関連付けられた VM の停止

- VMI がライブマイグレーション可能でなく、対応する **VirtualMachine** オブジェクトに **runStrategy: always** が設定されている場合、仮想マシン (VM) を手動で停止することにより VMI を更新することができます。

```
$ virctl stop --namespace <namespace> <vm>
```

新しい VMI は、更新された **virt-launcher** Pod ですぐにスピンアップし、停止した VMI を置き換えます。これは、再起動アクションに相当します。



注記

ライブマイグレーション可能 な VM を手動で停止することは破壊的であり、ワークロードが中断されるため推奨できません。

ライブ移行可能な VMI の移行

VMI がライブマイグレーション可能な場合は、実行中の特定の VMI を対象とする **VirtualMachineInstanceMigration** オブジェクトを作成することで更新できます。VMI は、更新された **virt-launcher** Pod に移行されます。

1. **VirtualMachineInstanceMigration** マニフェストを作成し、**migration.yaml** として保存します。

```
apiVersion: kubevirt.io/v1
kind: VirtualMachineInstanceMigration
metadata:
  name: <migration_name>
  namespace: <namespace>
spec:
  vmiName: <vmi_name>
```

2. 移行をトリガーする **VirtualMachineInstanceMigration** オブジェクトを作成します。

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

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.35. SingleStackIPv6Unsupported

意味
このアラートは、シングルスタック IPv6 クラスターに OpenShift Virtualization をインストールすると発生します。

影響
仮想マシンを作成することはできません。

診断

- 次のコマンドを実行して、クラスターのネットワーク設定を確認します。

```
$ oc get network.config cluster -o yaml
```

出力には、クラスターネットワークの IPv6 CIDR のみが表示されます。

出力例

```
apiVersion: config.openshift.io/v1
kind: Network
metadata:
  name: cluster
spec:
  clusterNetwork:
  - cidr: fd02::/48
    hostPrefix: 64
```

軽減策
OpenShift Virtualization をシングルスタック IPv4 クラスターまたはデュアルスタック IPv4/IPv6 クラスターにインストールします。

11.5.36. SSPCommonTemplatesModificationReverted

意味
このアラートは、Scheduling、Scale、and Performance (SSP) Operator が調整手順の一環として共通テンプレートへの変更を元に戻すときに発生します。

SSP Operator は、共通テンプレートおよびテンプレートバリデーターをデプロイし、調整します。ユーザーまたはスクリプトが共通テンプレートを変更すると、その変更は SSP オペレーターによって元に戻されます。

影響
共通テンプレートへの変更は上書きされます。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get deployment -A | grep ssp-operator | \
  awk '{print $1}')"
```

2. 変更が元に戻されたテンプレートの **ssp-operator** ログを確認します。

```
$ oc -n $NAMESPACE logs --tail=-1 -l control-plane=ssp-operator | \
  grep 'common template' -C 3
```

軽減策
変更の原因を特定して解決するようにしてください。

テンプレート自体ではなく、テンプレートのコピーのみを変更するようにしてください。

11.5.37. SSPDown

意味
このアラートは、すべての Scheduling、Scale and Performance (SSP) Operator Pod がダウンしたときに発生します。

SSP オペレーターは、共通テンプレートとテンプレートバリデーターのデプロイと調整を担当します。

影響
依存コンポーネントがデプロイされていない可能性があります。コンポーネントの変更は調整されない場合があります。その結果、共通テンプレートおよび/またはテンプレートバリデーターが失敗した場合、更新またはリセットされない可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get deployment -A | grep ssp-operator | \
  awk '{print $1}')"
```

2. **ssp-operator Pod** のステータスをチェックします。

```
$ oc -n $NAMESPACE get pods -l control-plane=ssp-operator
```

3. **ssp-operator Pod** の詳細を取得します。

```
$ oc -n $NAMESPACE describe pods -l control-plane=ssp-operator
```

4. **ssp-operator** ログでエラーメッセージをチェックします。

```
$ oc -n $NAMESPACE logs --tail=-1 -l control-plane=ssp-operator
```

軽減策
根本原因の特定と問題の解決を試みてください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.38. SSPFailingToReconcile

意味
このアラートは、SSP Operator が実行されているにもかかわらず、Scheduling、Scale and Performance (SSP) Operator の調整サイクルが繰り返し失敗した場合に発生します。

SSP オペレーターは、共通テンプレートとテンプレートバリデーターのデプロイと調整を担当します。

影響
依存コンポーネントがデプロイされていない可能性があります。コンポーネントの変更は調整されない場合があります。その結果、共通テンプレートまたはテンプレートバリデーターが失敗した場合、更新またはリセットされない可能性があります。

診断

1. **NAMESPACE** 環境変数をエクスポートします。

```
$ export NAMESPACE="$(oc get deployment -A | grep ssp-operator | \
  awk '{print $1}')"
```

2. **ssp-operator Pod** の詳細を取得します。

```
$ oc -n $NAMESPACE describe pods -l control-plane=ssp-operator
```

3. **ssp-operator** ログでエラーをチェックします。

```
$ oc -n $NAMESPACE logs --tail=-1 -l control-plane=ssp-operator
```

4. **virt-template-validator** Pod のステータスを取得します。

```
$ oc -n $NAMESPACE get pods -l name=virt-template-validator
```

5. **virt-template-validator** Pod の詳細を取得します。

```
$ oc -n $NAMESPACE describe pods -l name=virt-template-validator
```

6. **virt-template-validator** ログでエラーをチェックします。

```
$ oc -n $NAMESPACE logs --tail=-1 -l name=virt-template-validator
```

軽減策

根本原因の特定と問題の解決を試みてください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.39. SSPHighRateRejectedVms

意味
このアラートは、ユーザーまたはスクリプトが無効な設定を使用して多数の仮想マシン (VM) を作成または変更しようとすると発生します。

影響
VM は作成または変更されません。その結果、環境が期待どおりに動作しない可能性があります。

診断

1. **NAMESPACE** 環境変数をエクスポートします。

```
$ export NAMESPACE="$(oc get deployment -A | grep ssp-operator | \
  awk '{print $1}')"
```

2. **virt-template-validator** ログで、原因を示す可能性のあるエラーをチェックします。

```
$ oc -n $NAMESPACE logs --tail=-1 -l name=virt-template-validator
```

出力例

```
{"component":"kubevirt-template-validator","level":"info","msg":"evalution
summary for ubuntu-3166wmdbbfkroku0:\nminimal-required-memory applied: FAIL,
value 1073741824 is lower than minimum [2147483648]\n\nsucceeded=false",
"pos":"admission.go:25","timestamp":"2021-09-28T17:59:10.934470Z"}
```

軽減策
根本原因の特定と問題の解決を試みてください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.40. SSPTemplateValidatorDown

意味
このアラートは、すべての Template Validator Pod がダウンしたときに発生します。

Template Validator は、仮想マシン (VM) をチェックして、テンプレートに違反していないことを確認します。

影響
VM はテンプレートに対して検証されません。その結果、それぞれのワークロードに一致しない仕様で仮想マシンが作成される可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get deployment -A | grep ssp-operator | \
  awk '{print $1}')"
```

2. **virt-template-validator** Pod のステータスを取得します。

```
$ oc -n $NAMESPACE get pods -l name=virt-template-validator
```

3. **virt-template-validator** Pod の詳細を取得します。

```
$ oc -n $NAMESPACE describe pods -l name=virt-template-validator
```

4. **virt-template-validator** ログでエラーメッセージをチェックします。

```
$ oc -n $NAMESPACE logs --tail=-1 -l name=virt-template-validator
```

軽減策
根本原因の特定と問題の解決を試みてください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.41. UnsupportedHCOModification

意味
このアラートは、JSON パッチアノテーションを使用して HyperConverged Cluster Operator (HCO) のオペランドを変更したときに発生します。

HCO は、OpenShift Virtualization とそのサポートオペレーターを独自の方法で設定し、予期しない変更があった場合にそのオペランドを上書きします。ユーザーは、オペランドを直接変更してはなりません。

ただし、変更が必要であり、HCO API でサポートされていない場合は、JSON パッチアノテーションを使用して HCO に強制的にオペレーターに変更を設定させることができます。これらの変更は、調整プロセス中に HCO によって元に戻されません。

影響
JSON パッチアノテーションを誤って使用すると、予期しない結果が生じたり、環境が不安定になったりする可能性があります。

コンポーネントのカスタムリソースの構造が変更される可能性があるため、JSON パッチアノテーションを使用してシステムをアップグレードすることは危険です。

診断

- JSON パッチアノテーションを特定するには、アラートの詳細で **annotation_name** を確認します。

```
Labels
  alertname=KubevirtHyperconvergedClusterOperatorUSModification
  annotation_name=kubevirt.kubevirt.io/jsonpatch
  severity=info
```

軽減策

オペランドを変更するには、HCO API を使用することを推奨します。ただし、JSON パッチアノテーションを使用しないと変更できない場合は、注意して進めてください。

潜在的な問題を回避するために、アップグレード前に JSON パッチアノテーションを削除します。

11.5.42. VirtAPIDown

意味
このアラートは、すべての API サーバー Pod がダウンしたときに発生します。

影響
OpenShift Virtualization オブジェクトは API 呼び出しを送信できません。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-api** Pod のステータスを確認します。

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-api
```

3. **virt-api** デプロイメントのステータスを確認します。

```
$ oc -n $NAMESPACE get deploy virt-api -o yaml
```

4. Pod のクラッシュやイメージのプルの失敗などの問題について、**virt-api** デプロイメントの詳細を確認します。

```
$ oc -n $NAMESPACE describe deploy virt-api
```

5. **NotReady** 状態のノードなどの問題をチェックします。

```
$ oc get nodes
```

軽減策
根本原因の特定と問題の解決を試みてください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.43. VirtApiRESTErrorsBurst

意味
過去 5 分間に **virt-api** Pod で REST 呼び出しの 80% 以上が失敗しました。

影響
virt-api への REST 呼び出しの失敗率が非常に高いと、API 呼び出しの応答と実行が遅くなり、API 呼び出しが完全に無視される可能性があります。

ただし、現在実行中の仮想マシンのワークロードが影響を受ける可能性はほとんどありません。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. デプロイメントの **virt-api** Pod のリストを取得します。

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-api
```

3. **virt-api** ログでエラーメッセージをチェックします。

```
$ oc logs -n $NAMESPACE <virt-api>
```

4. **virt-api** Pod の詳細を取得します。

```
$ oc describe -n $NAMESPACE <virt-api>
```

5. ノードに問題が発生していないか確認してください。たとえば、**NotReady** 状態にある可能性があります。

```
$ oc get nodes
```

6. **virt-api** デプロイメントのステータスを確認します。

```
$ oc -n $NAMESPACE get deploy virt-api -o yaml
```

7. **virt-api** デプロイメントの詳細を取得します。

```
$ oc -n $NAMESPACE describe deploy virt-api
```

軽減策
診断手順で得られた情報に基づいて、根本原因の特定と問題の解決を試みます。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.44. VirtApiRESTErrorsHigh

意味
過去 60 分間に **virt-api** Pod で 5% 以上の REST 呼び出しが失敗しました。

影響
virt-api への REST 呼び出しの失敗率が高いと、API 呼び出しの応答と実行が遅くなる可能性があります。

ただし、現在実行中の仮想マシンのワークロードが影響を受ける可能性はほとんどありません。

診断

1. **NAMESPACE** 環境変数を次のように設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-api** Pod のステータスを確認します。

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-api
```

3. **virt-api** ログをチェックします。

```
$ oc logs -n  $NAMESPACE <virt-api>
```

4. **virt-api** Pod の詳細を取得します。

```
$ oc describe -n $NAMESPACE <virt-api>
```

5. ノードに問題が発生していないか確認してください。たとえば、**NotReady** 状態にある可能性があります。

```
$ oc get nodes
```

6. **virt-api** デプロイメントのステータスを確認します。

```
$ oc -n $NAMESPACE get deploy virt-api -o yaml
```

7. **virt-api** デプロイメントの詳細を取得します。

```
$ oc -n $NAMESPACE describe deploy virt-api
```

軽減策
診断手順で得られた情報に基づいて、根本原因の特定と問題の解決を試みます。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.45. VirtControllerDown

意味
実行 **中の virt-controller** Pod が 5 分間検出されませんでした。

影響
仮想マシン (VM) のライフサイクル管理に関連するアクションはすべて失敗します。これには特に、新しい仮想マシンインスタンス (VMI) の起動や既存の VMI のシャットダウンが含まれます。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-controller** デプロイメントのステータスを確認します。

```
$ oc get deployment -n $NAMESPACE virt-controller -o yaml
```

3. **virt-controller** Pod のログを確認します。

```
$ oc get logs <virt-controller>
```

軽減策
このアラートには、次のようなさまざまな原因が考えられます。

- ノードリソースの枯渇
- クラスターに十分なメモリーがない
- ノードがダウンしている
- API サーバーが過負荷になっています。たとえば、スケジューラーの負荷が高く、完全には使用できない場合があります。
- ネットワークの問題

根本原因を特定し、可能であれば修正します。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.46. VirtControllerRESTErrorsBurst

意味
過去 5 分間で、**virt-controller** Pod の REST 呼び出しの 80% 以上が失敗しました。

virt-controller が API サーバーへの接続を完全に失った可能性があります。

このエラーは、多くの場合、次のいずれかの問題が原因で発生します。

- API サーバーが過負荷になっているため、タイムアウトが発生します。これに該当するかどうかを確認するには、API サーバーのメトリクスを確認し、その応答時間と全体的な呼び出しを表示します。
- **virt-controller** Pod が API サーバーに到達できない。これは通常、ノードの DNS の問題とネットワーク接続の問題が原因で発生します。

影響
ステータスの更新は反映されず、移行などのアクションは実行できません。ただし、実行中のワークロードは影響を受けません。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. 使用可能な **virt-controller** Pod を一覧表示します。

```
$ oc get pods -n $NAMESPACE -l=kubevirt.io=virt-controller
```

3. API サーバーに接続するときのエラーメッセージについては、**virt-controller** ログを確認します。

```
$ oc logs -n  $NAMESPACE <virt-controller>
```

軽減策

- **virt-controller** Pod が API サーバーに接続できない場合は、Pod を削除して強制的に再起動します。

```
$ oc delete -n $NAMESPACE <virt-controller>
```

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.47. VirtControllerRESTErrorsHigh

意味
過去 60 分間に **virt-controller** で REST 呼び出しの 5% 以上が失敗しました。

これはおそらく、**virt-controller** が API サーバーへの接続を部分的に失ったためです。

このエラーは、多くの場合、次のいずれかの問題が原因で発生します。

- API サーバーが過負荷になっているため、タイムアウトが発生します。これに該当するかどうかを確認するには、API サーバーのメトリクスを確認し、その応答時間と全体的な呼び出しを表示します。
- **virt-controller** Pod が API サーバーに到達できない。これは通常、ノードの DNS の問題とネットワーク接続の問題が原因で発生します。

影響
仮想マシンの起動や移行、スケジューリングなどのノード関連のアクションが遅れます。実行中のワークロードは影響を受けませんが、現在のステータスのレポートが遅れる可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. 使用可能な **virt-controller** Pod を一覧表示します。

```
$ oc get pods -n $NAMESPACE -l=kubevirt.io=virt-controller
```

3. API サーバーに接続するときのエラーメッセージについては、**virt-controller** ログを確認します。

```
$ oc logs -n  $NAMESPACE <virt-controller>
```

軽減策

- **virt-controller** Pod が API サーバーに接続できない場合は、Pod を削除して強制的に再起動します。

```
$ oc delete -n $NAMESPACE <virt-controller>
```

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.48. VirtHandlerDaemonSetRolloutFailing

意味
virt-handler デーモンセットは、15 分後に 1 つ以上のワーカーノードにデプロイできませんでした。

影響
このアラートは警告です。すべての **virt-handler** デーモンセットがデプロイに失敗したことを示すわけではありません。したがって、クラスターが過負荷にならない限り、仮想マシンの通常のライフサイクルは影響を受けません。

診断
実行中の **virt-handler** Pod を持たないワーカーノードを特定します。

1. **NAMESPACE** 環境変数をエクスポートします。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-handler** Pod のステータスを確認して、デプロイされていない Pod を特定します。

```
$ oc get pods -n $NAMESPACE -l=kubevirt.io=virt-handler
```

3. **virt-handler** Pod のワーカーノードの名前を取得します。

```
$ oc -n $NAMESPACE get pod <virt-handler> -o jsonpath='{.spec.nodeName}'
```

軽減策
リソースが不足しているために **virt-handler** Pod のデプロイに失敗した場合は、影響を受けるワーカーノード上の他の Pod を削除できます。

11.5.49. VirtHandlerRESTErrorsBurst

意味
過去 5 分間に **virt-handler** で REST 呼び出しの 80% 以上が失敗しました。このアラートは通常、 **virt-handler** Pod が API サーバーに接続できないことを示します。

このエラーは、多くの場合、次のいずれかの問題が原因で発生します。

- API サーバーが過負荷になっているため、タイムアウトが発生します。これに該当するかどうかを確認するには、API サーバーのメトリクスを確認し、その応答時間と全体的な呼び出しを表示します。
- **virt-handler** Pod が API サーバーに到達できません。これは通常、ノードの DNS の問題とネットワーク接続の問題が原因で発生します。

影響
ステータスの更新は反映されず、移行などのノード関連のアクションは失敗します。ただし、影響を受けるノードで実行中のワークロードは影響を受けません。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-handler** Pod のステータスをチェックします。

```
$ oc get pods -n $NAMESPACE -l=kubevirt.io=virt-handler
```

3. API サーバーに接続するときのエラーメッセージについては、**virt-handler** ログを確認します。

```
$ oc logs -n  $NAMESPACE <virt-handler>
```

軽減策

- **virt-handler** が API サーバーに接続できない場合、Pod を削除して強制的に再起動します。

```
$ oc delete -n $NAMESPACE <virt-handler>
```

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.50. VirtHandlerRESTErrorsHigh

意味
過去 60 分間に、REST 呼び出しの 5% 以上が **virt-handler** で失敗しました。このアラートは通常、**virt-handler** Pod が API サーバーへの接続を部分的に失ったことを示します。

このエラーは、多くの場合、次のいずれかの問題が原因で発生します。

- API サーバーが過負荷になっているため、タイムアウトが発生します。これに該当するかどうかを確認するには、API サーバーのメトリクスを確認し、その応答時間と全体的な呼び出しを表示します。
- **virt-handler** Pod が API サーバーに到達できません。これは通常、ノードの DNS の問題とネットワーク接続の問題が原因で発生します。

影響
ワークロードの開始や移行などのノード関連のアクションは、**virt-handler** が実行されているノードで遅延します。実行中のワークロードは影響を受けませんが、現在のステータスのレポートが遅れる可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. 障害が発生している **virt-handler** Pod を特定するために、利用可能な **virt-handler** Pod をリストします。

```
$ oc get pods -n $NAMESPACE -l=kubevirt.io=virt-handler
```

3. 障害が発生している **virt-handler** Pod のログで、API サーバー接続エラーを確認します。

```
$ oc logs -n $NAMESPACE <virt-handler>
```

エラーメッセージの例:

```
{"component":"virt-handler","level":"error","msg":"Can't patch node my-
node","pos":"heartbeat.go:96","reason":"the server has received too many API requests and
has asked us to try again later","timestamp":"2023-11-
06T11:11:41.099883Z","uid":"132c50c2-8d82-4e49-8857-dc737adcd6cc"}
```

軽減策
Pod を削除して再起動を強制します。

```
$ oc delete -n $NAMESPACE <virt-handler>
```

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.51. VirtOperatorDown

意味
このアラートは、**Running** 状態の **virt-operator** Pod が 10 分間検出されなかった場合に発生します。

virt-operator は、クラスターで開始する最初の Operator です。その主な責任には、次のものが含まれます。

- クラスターのインストール、ライブ更新、およびライブアップグレード
- **virt-controller**、**virt-handler**、**virt-launcher** などの最上位コントローラーのライフサイクルを監視し、それらの調整を管理する
- 証明書のローテーションやインフラストラクチャー管理など、特定のクラスター全体のタスク

virt-operator デプロイメントには、2 つの Pod のデフォルトレプリカが設定されます。

影響
このアラートは、クラスターレベルでの障害を示します。証明書のローテーション、アップグレード、コントローラーの調整など、重要なクラスター全体の管理機能が利用できない場合があります。

virt-operator には、クラスター内の仮想マシンに対する直接の責任はありません。したがって、一時的に利用できなくても仮想マシンのワークロードに大きな影響はありません。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-operator** デプロイメントのステータスを確認します。

```
$ oc -n $NAMESPACE get deploy virt-operator -o yaml
```

3. **virt-operator** デプロイメントの詳細を取得します。

```
$ oc -n $NAMESPACE describe deploy virt-operator
```

4. **virt-operator Pod** のステータスをチェックします。

```
$ oc get pods -n $NAMESPACE -l=kubevirt.io=virt-operator
```

5. **NotReady** 状態などのノードの問題をチェックします。

```
$ oc get nodes
```

軽減策
診断手順で得られた情報に基づいて、根本原因を見つけて問題を解決してください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.52. VirtOperatorRESTErrorsBurst

意味
このアラートは、**virt-operator** Pod の REST 呼び出しの 80% 以上が過去 5 分間に失敗した場合に発生します。これは通常、**virt-operator** Pod が API サーバーに接続できないことを示しています。

このエラーは、多くの場合、次のいずれかの問題が原因で発生します。

- API サーバーが過負荷になっているため、タイムアウトが発生します。これに該当するかどうかを確認するには、API サーバーのメトリクスを確認し、その応答時間と全体的な呼び出しを表示します。
- **virt-operator** Pod が API サーバーに到達できない。これは通常、ノードの DNS の問題とネットワーク接続の問題が原因で発生します。

影響
アップグレードやコントローラーの調整などのクラスターレベルのアクションは利用できない場合があります。

ただし、仮想マシン (VM) や VM インスタンス (VMI) などのワークロードは影響を受けない可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-operator Pod** のステータスをチェックします。

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-operator
```

3. API サーバーに接続するときのエラーメッセージについては、**virt-operator** ログを確認します。

```
$ oc -n $NAMESPACE logs <virt-operator>
```

4. **virt-operator** Pod の詳細を取得します。

```
$ oc -n $NAMESPACE describe pod <virt-operator>
```

軽減策

- **virt-operator** が API サーバーに接続できない場合、Pod を削除して強制的に再起動します。

```
$ oc delete -n $NAMESPACE <virt-operator>
```

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.53. VirtOperatorRESTErrorsHigh

意味
このアラートは、過去 60 分間に **virt-operator** Pod で 5% を超える REST 呼び出しが失敗した場合に発生します。これは通常、**virt-operator** Pod が API サーバーに接続できないことを示しています。

このエラーは、多くの場合、次のいずれかの問題が原因で発生します。

- API サーバーが過負荷になっているため、タイムアウトが発生します。これに該当するかどうかを確認するには、API サーバーのメトリクスを確認し、その応答時間と全体的な呼び出しを表示します。
- **virt-operator** Pod が API サーバーに到達できない。これは通常、ノードの DNS の問題とネットワーク接続の問題が原因で発生します。

影響
アップグレードやコントローラーの調整などのクラスターレベルのアクションが遅れる場合があります。

ただし、仮想マシン (VM) や VM インスタンス (VMI) などのワークロードは影響を受けない可能性があります。

診断

1. **NAMESPACE** 環境変数を設定します。

```
$ export NAMESPACE="$(oc get kubevirt -A \
  -o custom-columns="":.metadata.namespace)"
```

2. **virt-operator Pod** のステータスをチェックします。

```
$ oc -n $NAMESPACE get pods -l kubevirt.io=virt-operator
```

3. API サーバーに接続するときのエラーメッセージについては、**virt-operator** ログを確認します。

```
$ oc -n $NAMESPACE logs <virt-operator>
```

4. **virt-operator** Pod の詳細を取得します。

```
$ oc -n $NAMESPACE describe pod <virt-operator>
```

軽減策

- **virt-operator** が API サーバーに接続できない場合、Pod を削除して強制的に再起動します。

```
$ oc delete -n $NAMESPACE <virt-operator>
```

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

11.5.54. VMCannotBeEvicted

意味
このアラートは、仮想マシン (VM) のエビクションストラテジーが **LiveMigration** に設定されているが、仮想マシンが移行可能でない場合に発生します。

影響
移行不可能な仮想マシンは、ノードの削除を妨げます。この状態は、ノードのドレインや更新などの操作に影響します。

診断

1. VMI 設定をチェックして、**evictionStrategy** の値が **LiveMigrate**であるかどうかを判断します。

```
$ oc get vmis -o yaml
```

2. **LIVE-MIGRATABLE** 列の **False** ステータスを確認して、移行できない VMI を特定します。

```
$ oc get vmis -o wide
```

3. VMI の詳細を取得し、**spec.conditions** をチェックして問題を特定します。

```
$ oc get vmi <vmi> -o yaml
```

出力例

```
status:
  conditions:
  - lastProbeTime: null
    lastTransitionTime: null
    message: cannot migrate VMI which does not use masquerade to connect
    to the pod network
    reason: InterfaceNotLiveMigratable
    status: "False"
    type: LiveMigratable
```

軽減策

VMI の **evictionStrategy** を **シャットダウン** するように設定するか、VMI の移行を妨げている問題を解決します。

11.5.55. VMStorageClassWarning

意味
このアラートは、ストレージクラスが正しく設定されていない場合に発生します。システム全体で共有されるダミーページは、異なるプロセスまたはスレッド間でデータの書き込みと読み取りが行われるときに CRC エラーを引き起こします。

影響
CRC エラーが多数発生すると、クラスターのパフォーマンスが大幅に低下する可能性があります。

診断

1. Web コンソールで **Observe → Metrics** に移動します。
2. 次の PromQL クエリーを実行して、ストレージクラスが正しく設定されていない仮想マシンのリストを取得します。

   ```
   kubevirt_ssp_vm_rbd_volume{rxbounce_enabled="false", volume_mode="Block"} == 1
   ```

   出力には、**rxbounce_enabled** のないストレージクラスを使用する仮想マシンのリストが表示されます。

   出力例

   ```
   kubevirt_ssp_vm_rbd_volume{name="testvmi-gwgdqp22k7", namespace="test_ns",
   pv_name="testvmi-gwgdqp22k7", rxbounce_enabled="false", volume_mode="Block"} 1
   ```

3. 次のコマンドを実行して、ストレージクラス名を取得します。

   ```
   $ oc get pv <pv_name> -o=jsonpath='{.spec.storageClassName}'
   ```

軽減策
krbd:rxbounce マップオプションを使用して、デフォルトの OpenShift Virtualization ストレージクラスを作成します。詳細は 、Windows VM の ODF PersistentVolumes の最適化 を参照してください。

問題を解決できない場合は、カスタマーポータル にログインしてサポートケースを開き、診断手順で収集したアーティファクトを添付してください。

第12章 サポート

12.1. 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。

12.1.1. 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12.1 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

12.1.2. 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 Dedicated 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。

12.1.3. 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。

12.2. 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。

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 Dedicated モニタリングスタックの詳細は、OpenShift Dedicated モニタリング について を参照してください。

12.2.1. 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

- クラスターの Prometheus メトリクスを収集します。

12.2.2. 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. 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。
2. 修復を試みる **前** に、VM からメモリーダンプを収集 します。
3. 誤動作している仮想マシンに共通する要因を記録します。たとえば、仮想マシンには同じホストまたはネットワークがあります。

12.3. 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。

12.3.1. イベント

OpenShift Container Platform イベント は重要なライフサイクル情報の記録であり、仮想マシン、namespace、リソース問題のモニタリングおよびトラブルシューティングに役立ちます。

- 仮想マシンイベント: Web コンソールで **VirtualMachine details** ページの **Events** タブ に移動します。

  namespace イベント

  namespaceイベントを表示するには、次のコマンドを実行します。

  ```
  $ oc get events -n <namespace>
  ```

  特定のイベントの詳細は、イベントのリスト を参照してください。

  リソースイベント

  リソースイベントを表示するには、次のコマンドを実行します。

  ```
  $ oc describe <resource> <resource_name>
  ```

12.3.2. 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。

12.3.2.1. 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。

12.3.2.2. Web コンソールを使用して virt-launcher Pod のログを表示する

OpenShift Dedicated 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。

12.3.2.3. 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。

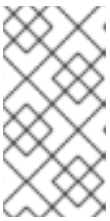
```
$ oc get pods -n openshift-cnv
```

例12.1 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12.2 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"}
```

12.3.3. 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。ゲストのログへのアクセスを設定し、OpenShift Dedicated Web コンソールまたは **oc** CLI のいずれかを使用してそれらを表示できます。

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。

12.3.3.1. 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。

12.3.3.2. 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。

12.3.3.3. 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。

12.3.3.4. 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>
```

12.3.3.5. 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。

12.3.3.6. 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
```

12.3.4. 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。

12.3.4.1. 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

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。

12.3.4.2. 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12.2 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"`
kubemacpool	`{log_type=~".+"}|json |kubernetes_labels_app_kubernetes_io_part_of="hyperconverged-cluster" |kubernetes_labels_app_kubernetes_io_component="network"`

コンポーネント	LogQL クエリー
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12.3 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

12.3.5. 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。

12.3.6. 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。

12.3.6.1. 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。

12.3.6.2. 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ついての特定の詳細、およびどのように現在の状態になったかについての情報を提供するメッセージを受信する可能性があります。

状態の組み合わせは多数あります。それぞれは一意のコンテキストで評価される必要があります。

各種の組み合わせの例を以下に示します。

- **Bound** - この例では、正常にバインドされた PVC が表示されます。
  Type は **Bound** であるため、**Status** は **True** になります。PVC がバインドされていない場合、**Status** は **False** になります。

  PVC がバインドされると、PVC がバインドされていることを示すイベントが生成されます。この場合、**Reason** は **Bound** で、**Status** は **True** です。**Message** はデータボリュームを所有する PVC を示します。

  Events セクションの **Message** では、PVC がバインドされている期間 (**Age**) およびどのリソース (**From**) によってバインドされているか、**datavolume-controller** に関する詳細が提供されます。

  出力例

```
Status:
  Conditions:
    Last Heart Beat Time:  2020-07-15T03:58:24Z
    Last Transition Time:  2020-07-15T03:58:24Z
    Message:               PVC win10-rootdisk Bound
    Reason:                Bound
    Status:                True
    Type:                  Bound
...
  Events:
    Type     Reason     Age    From                Message
    ----     ------     ----   ----                -------
    Normal   Bound      24s    datavolume-controller  PVC example-dv Bound
```

- **Running** - この場合、**Type** が **Running** で、**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
```

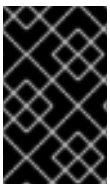
第13章 バックアップおよび復元

13.1. 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。

- Kubernetes Volume Snapshot API をサポートする Container Storage Interface (CSI) ドライバーを備えた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。

13.1.1. 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 SCC 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。

13.1.2. スナップショットの作成

OpenShift Dedicated 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。

13.1.2.1. 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 Dedicated 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。

13.1.2.2. 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。

13.1.3. 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。

13.1.4. スナップショットからの仮想マシンの復元

OpenShift Dedicated 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(VM)を復元できます。

13.1.4.1. 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

仮想マシン(VM)は、OpenShift Dedicated Web コンソールのスナップショットで表される以前の設定に復元できます。

手順

1. 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 VirtualMachineSnapshot** を選択します。
7. **Restore** をクリックします。

13.1.4.2. 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/v1beta1
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。

13.1.5. スナップショットの削除

OpenShift Dedicated 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。

13.1.5.1. 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 VirtualMachineSnapshot** を選択します。
5. **Delete** をクリックします。

13.1.5.2. 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
```

13.2. OADP のインストールおよび設定

クラスター管理者は、OADP Operator をインストールして、OpenShift API for Data Protection (OADP) をインストールします。Operator は Velero 1.12 をインストールします。

バックアップストレージプロバイダーのデフォルトの **Secret** を作成し、Data Protection Application をインストールします。

13.2.1. OADP Operator のインストール

Operator Lifecycle Manager (OLM)を使用して、OpenShift Dedicated 4 に OpenShift API for Data Protection (OADP)オペレーターをインストールします。

OADP Operator は Velero 1.12 をインストールします。

前提条件

- **cluster-admin** 権限を持つユーザーとしてログインしている。

手順

1. OpenShift Dedicated Web コンソールで、**Operators → OperatorHub** をクリックします。
2. **Filter by keyword** フィールドを使用して、**OADP Operator** を検索します。
3. **OADP Operator** を選択し、**Install** をクリックします。
4. **Install** をクリックして、**openshift-adp** プロジェクトに Operator をインストールします。
5. **Operators → Installed Operators** をクリックして、インストールを確認します。

13.2.2. バックアップおよびスナップショットの場所、ならびにそのシークレットについて

DataProtectionApplication カスタムリソース (CR) で、バックアップおよびスナップショットの場所、ならびにそのシークレットを指定します。

バックアップの場所
Multicloud Object Gateway または MinIO などの AWS S3 互換オブジェクトストレージを、バックアップの場所として指定します。

Velero は、オブジェクトストレージのアーカイブファイルとして、OpenShift Dedicated リソース、Kubernetes オブジェクト、および内部イメージをバックアップします。

スナップショットの場所
クラウドプロバイダーのネイティブスナップショット API を使用して永続ボリュームをバックアップする場合、クラウドプロバイダーをスナップショットの場所として指定する必要があります。

Container Storage Interface (CSI) スナップショットを使用する場合、CSI ドライバーを登録するために **VolumeSnapshotClass** CR を作成するため、スナップショットの場所を指定する必要はありません。

ファイルシステムバックアップ (FSB) を使用する場合、FSB がオブジェクトストレージ上にファイルシステムをバックアップするため、スナップショットの場所を指定する必要はありません。

シークレット
バックアップとスナップショットの場所が同じ認証情報を使用する場合、またはスナップショットの場所が必要ない場合は、デフォルトの **Secret** を作成します。

バックアップとスナップショットの場所で異なる認証情報を使用する場合は、次の 2 つの secret オブジェクトを作成します。

- **DataProtectionApplication** CR で指定する、バックアップの場所用のカスタム **Secret**。
- **DataProtectionApplication** CR で参照されない、スナップショットの場所用のデフォルト **Secret**。

重要

Data Protection Application には、デフォルトの **Secret** が必要です。作成しないと、インストールは失敗します。

インストール中にバックアップまたはスナップショットの場所を指定したくない場合は、空の **credentials-velero** ファイルを使用してデフォルトの **Secret** を作成できます。

13.2.2.1. デフォルト Secret の作成

バックアップとスナップショットの場所が同じ認証情報を使用する場合、またはスナップショットの場所が必要ない場合は、デフォルトの **Secret** を作成します。

注記

DataProtectionApplication カスタムリソース (CR) にはデフォルトの **Secret** が必要です。作成しないと、インストールは失敗します。バックアップの場所の **Secret** の名前が指定されていない場合は、デフォルトの名前が使用されます。

インストール時にバックアップの場所の認証情報を使用しない場合は、空の **credentials-velero** ファイルを使用してデフォルト名前で **Secret** を作成できます。

前提条件

- オブジェクトストレージとクラウドストレージがある場合は、同じ認証情報を使用する必要があります。
- Velero のオブジェクトストレージを設定する必要があります。
- オブジェクトストレージ用の **credentials-velero** ファイルを適切な形式で作成する必要があります。

手順

- デフォルト名で **Secret** を作成します。

```
$ oc create secret generic cloud-credentials -n openshift-adp --from-file cloud=credentials-velero
```

Secret は、Data Protection Application をインストールするときに、**DataProtectionApplication** CR の **spec.backupLocations.credential** ブロックで参照されます。

13.2.3. Data Protection Application の設定

Velero リソースの割り当てを設定するか、自己署名 CA 証明書を有効にして、Data Protection Application を設定できます。

13.2.3.1. Velero の CPU とメモリーのリソース割り当てを設定

DataProtectionApplication カスタムリソース (CR) マニフェストを編集して、**Velero** Pod の CPU およびメモリーリソースの割り当てを設定します。

前提条件

- OpenShift API for Data Protection (OADP) Operator がインストールされている必要があります。

手順

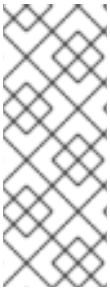
- 次の例のように、**DataProtectionApplication** CR マニフェストの **spec.configuration.velero.podConfig.ResourceAllocations** ブロックの値を編集します。

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
spec:
```

```
# ...
  configuration:
    velero:
      podConfig:
        nodeSelector: <node selector> 1
        resourceAllocations: 2
          limits:
            cpu: "1"
            memory: 1024Mi
          requests:
            cpu: 200m
            memory: 256Mi
```

1 Velero podSpec に提供されるノードセレクターを指定します。

2 リストされている **resourceAllocations** は、平均使用量です。



注記

Kopia は OADP 1.3 以降のリリースで選択できます。Kopia はファイルシステムのバックアップに使用できます。組み込みの Data Mover を使用する Data Mover の場合は、Kopia が唯一の選択肢になります。

Kopia は Restic よりも多くのリソースを消費するため、それに応じて CPU とメモリーの要件を調整しなければならない場合があります。

13.2.3.2. 自己署名 CA 証明書の有効化

certificate signed by unknown authority エラーを防ぐために、**DataProtectionApplication** カスタムリソース (CR) マニフェストを編集して、オブジェクトストレージの自己署名 CA 証明書を有効にする必要があります。

前提条件

- OpenShift API for Data Protection (OADP) Operator がインストールされている必要があります。

手順

- **DataProtectionApplication** CR マニフェストの **spec.backupLocations.velero.objectStorage.caCert** パラメーターと **spec.backupLocations.velero.config** パラメーターを編集します。

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
spec:
# ...
  backupLocations:
    - name: default
      velero:
        provider: aws
        default: true
```

```
      objectStorage:
        bucket: <bucket>
        prefix: <prefix>
        caCert: <base64_encoded_cert_string> 1
      config:
        insecureSkipTLSVerify: "false" 2
# ...
```

1. Base64 でエンコードされた CA 証明書文字列を指定します。
2. **insecureSkipTLSVerify** 設定は、**"true"** または **"false"** のいずれかに設定できます。**"true"** に設定すると、SSL/TLS セキュリティーが無効になります。**"false"** に設定すると、SSL/TLS セキュリティーが有効になります。

13.2.3.2.1. Velero デプロイメント用のエイリアス化した velero コマンドで CA 証明書を使用する

Velero CLI のエイリアスを作成することで、システムにローカルにインストールせずに Velero CLI を使用できます。

前提条件

- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform クラスターにログインしている。
- OpenShift CLI (**oc**) がインストールされている。

1. エイリアス化した Velero コマンドを使用するには、次のコマンドを実行します。

```
$ alias velero='oc -n openshift-adp exec deployment/velero -c velero -it -- ./velero'
```

2. 次のコマンドを実行して、エイリアスが機能していることを確認します。

例

```
$ velero version
Client:
 Version: v1.12.1-OADP
 Git commit: -
Server:
 Version: v1.12.1-OADP
```

3. このコマンドで CA 証明書を使用するには、次のコマンドを実行して証明書を Velero デプロイメントに追加できます。

```
$ CA_CERT=$(oc -n openshift-adp get dataprotectionapplications.oadp.openshift.io
<dpa-name> -o jsonpath='{.spec.backupLocations[0].velero.objectStorage.caCert}')

$ [[ -n $CA_CERT ]] && echo "$CA_CERT" | base64 -d | oc exec -n openshift-adp -i
deploy/velero -c velero -- bash -c "cat > /tmp/your-cacert.txt" || echo "DPA BSL has no
caCert"
```

```
$ velero describe backup <backup_name> --details --cacert /tmp/<your_cacert>.txt
```

4. バックアップログを取得するために、次のコマンドを実行します。

```
$ velero backup logs  <backup_name>  --cacert /tmp/<your_cacert.txt>
```

このログを使用して、バックアップできないリソースの障害と警告を表示できます。

5. Velero Pod が再起動すると、**/tmp/your-cacert.txt** ファイルが消去されます。そのため、前の手順のコマンドを再実行して **/tmp/your-cacert.txt** ファイルを再作成する必要があります。

6. 次のコマンドを実行すると、**/tmp/your-cacert.txt** ファイルを保存した場所にファイルがまだ存在するかどうかを確認できます。

```
$ oc exec -n openshift-adp -i deploy/velero -c velero -- bash -c "ls /tmp/your-cacert.txt"
/tmp/your-cacert.txt
```

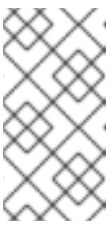
OpenShift API for Data Protection (OADP) の今後のリリースでは、この手順が不要になるように証明書を Velero Pod にマウントする予定です。

13.2.4. Data Protection Application 1.2 以前のインストール

DataProtectionApplication API のインスタンスを作成して、Data Protection Application (DPA) をインストールします。

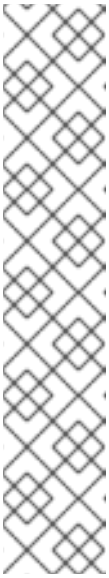
前提条件

- OADP Operator をインストールする必要がある。
- オブジェクトストレージをバックアップ場所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。



注記

Velero は、OADP namespace に **velero-repo-credentials** という名前のシークレットを作成します。これには、デフォルトのバックアップリポジトリーパスワードが含まれます。バックアップリポジトリーを対象とした最初のバックアップを実行する **前** に、base64 としてエンコードされた独自のパスワードを使用してシークレットを更新できます。更新するキーの値は **Data[repository-password]** です。

DPA を作成した後、バックアップリポジトリーを対象としたバックアップを初めて実行するときに、Velero はシークレットが **velero-repo-credentials** のバックアップリポジトリーを作成します。これには、デフォルトのパスワードまたは置き換えたパスワードが含まれます。最初のバックアップの **後** にシークレットパスワードを更新すると、新しいパスワードが **velero-repo-credentials** のパスワードと一致しなくなり、Velero は古いバックアップに接続できなくなります。

手順

1. **Operators → Installed Operators** をクリックして、OADP Operator を選択します。
2. **Provided APIs** で、**DataProtectionApplication** ボックスの **Create instance** をクリックします。
3. **YAML View** をクリックして、**DataProtectionApplication** マニフェストのパラメーターを更新します。

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
metadata:
  name: <dpa_sample>
  namespace: openshift-adp
spec:
  configuration:
    velero:
      defaultPlugins:
        - kubevirt (1)
        - gcp (2)
        - csi (3)
        - openshift (4)
      resourceTimeout: 10m (5)
    restic:
      enable: true (6)
      podConfig:
        nodeSelector: <node_selector> (7)
  backupLocations:
    - velero:
        provider: gcp (8)
        default: true
        credential:
          key: cloud
          name: <default_secret> (9)
        objectStorage:
          bucket: <bucket_name> (10)
          prefix: <prefix> (11)
```

-

1. **kubevirt** プラグインは OpenShift Virtualization に必須です。
2. 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**)。
3. 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。
4. **openshift** プラグインは必須です。
5. 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。
6. Restic インストールを無効にする場合は、この値を **false** に設定します。Restic はデーモンセットをデプロイします。これは、Restic Pod が各動作ノードで実行していることを意味します。OADP バージョン 1.2 以降では、**spec.defaultVolumesToFsBackup: true** を **Backup** CR に追加することで、バックアップ用に Restic を設定できます。OADP バージョン 1.1 では、**spec.defaultVolumesToRestic: true** を **Backup** CR に追加します。
7. Restic を使用できるノードを指定します。デフォルトでは、Restic はすべてのノードで実行されます。
8. バックアッププロバイダーを指定します。
9. 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。
10. 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。
11. 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/restic-9cq4q                                     1/1     Running   0          94s
pod/restic-m4lts                                     1/1     Running   0          94s
pod/restic-pv4kr                                     1/1     Running   0          95s
pod/velero-588db7f655-n842v                          1/1     Running   0          95s
```

```
NAME                                                     TYPE        CLUSTER-IP       EXTERNAL-IP
PORT(S)    AGE
service/oadp-operator-controller-manager-metrics-service   ClusterIP   172.30.70.140
<none>        8443/TCP   2m8s

NAME                 DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR   AGE
daemonset.apps/restic   3         3         3       3            3           <none>          96s

NAME                                        READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/oadp-operator-controller-manager    1/1     1            1           2m9s
deployment.apps/velero                              1/1     1            1           96s

NAME                                                   DESIRED   CURRENT   READY   AGE
replicaset.apps/oadp-operator-controller-manager-67d9494d47    1         1         1       2m9s
replicaset.apps/velero-588db7f655                              1         1         1       96s
```

2. 次のコマンドを実行して、**DataProtectionApplication** (DPA) が調整されていることを確認します。

```
$ oc get dpa dpa-sample -n openshift-adp -o jsonpath='{.status}'
```

出力例

```
{"conditions":[{"lastTransitionTime":"2023-10-27T01:23:57Z","message":"Reconcile
complete","reason":"Complete","status":"True","type":"Reconciled"}]}
```

3. **type** が **Reconciled** に設定されていることを確認します。

4. 次のコマンドを実行して、バックアップ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
```

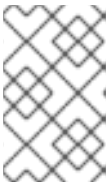
13.2.5. Data Protection Application 1.3 のインストール

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必要がある。
- オブジェクトストレージをバックアップ場所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. Built-in DataMover を使用するアップロード者として **kopia** を入力します。**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
```

13.2.5.1. DataProtectionApplication CR で CSI を有効にする

CSI スナップショットを使用して永続ボリュームをバックアップするには、**DataProtectionApplication** カスタムリソース (CR) で Container Storage Interface (CSI) を有効にします。

前提条件

- クラウドプロバイダーは、CSI スナップショットをサポートする必要があります。

手順

- 次の例のように、**DataProtectionApplication** CR を編集します。

```
apiVersion: oadp.openshift.io/v1alpha1
kind: DataProtectionApplication
...
spec:
  configuration:
    velero:
      defaultPlugins:
      - openshift
      - csi (1)
```

(1) **csi** デフォルトプラグインを追加します。

13.2.6. OADP のアンインストール

OpenShift API for Data Protection (OADP) をアンインストールするには、OADP Operator を削除します。詳細は、クラスターからの Operators の削除 を参照してください。

13.3. 仮想マシンのバックアップと復元

OpenShift API for Data Protection (OADP) を使用して、仮想マシンをバックアップおよび復元します。

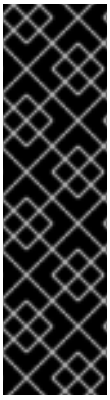
前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. ストレージプロバイダーの指示に従って、OADP Operator をインストールします。
2. **kubevirt** および **openshift** プラグインを使用してデータ保護アプリケーションをインストールします。
3. **Backup** カスタムリソース (CR) を作成して、仮想マシンをバックアップします。
4. **Restore** CR を作成して、**Backup** CR を復元します。

13.4. 仮想マシンのバックアップ



重要

OpenShift Virtualization の OADP は、テクノロジープレビュー機能としてのみご利用いただけます。テクノロジープレビュー機能は、Red Hat 製品サポートのサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではない場合があります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

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

OpenShift API for Data Protection (OADP) の **Backup** カスタムリソース (CR) を作成して、仮想マシン (VM) をバックアップします。

Backup CR は以下のアクションを実行します。

- Noobaa や Minio などの S3 互換オブジェクトストレージにアーカイブファイルを作成して、OpenShift Virtualization リソースをバックアップします。
- 以下のオプションのいずれかを使用して、仮想マシンディスクをバックアップします。
  - Ceph RBD または Ceph FS などの CSI 対応クラウドストレージ上の Container Storage Interface (CSI) スナップショット。
  - ファイルシステムバックアップを使用してアプリケーションをバックアップします。オブジェクトストレージ上の Kopia または Restic。

注記

OADP はバックアップフックを提供し、バックアップ操作の前に仮想マシンのファイルシステムをフリーズし、バックアップの完了時にフリーズを解除します。

kubevirt-controller は、バックアップ操作の前後に Velero が **virt-freezer** バイナリーを実行できるようにするアノテーションで **virt-launcher** Pod を作成します。

freeze および **unfreeze** API は、仮想マシンスナップショット API のサブリソースです。詳細は、仮想マシンのスナップショットについてを参照してください。

フックを **Backup** CR に追加して、バックアップ操作の前後に特定の仮想マシンでコマンドを実行できます。

Backup CR の代わりに **Schedule** CR を作成することにより、バックアップをスケジュールします。

13.4.1. バックアップ CR の作成

Backup カスタムリソース (CR) を作成して、Kubernetes イメージ、内部イメージ、および永続ボリューム (PV) をバックアップします。

前提条件

- OpenShift API for Data Protection (OADP) Operator をインストールしている。
- **DataProtectionApplication** CR が **Ready** 状態である。

- バックアップ場所の前提条件:
  - Velero 用に S3 オブジェクトストレージを設定する必要があります。
  - **DataProtectionApplication** CR でバックアップの場所を設定する必要があります。
- スナップショットの場所の前提条件:
  - クラウドプロバイダーには、ネイティブスナップショット API が必要であるか、Container Storage Interface (CSI) スナップショットをサポートしている必要があります。
  - CSI スナップショットの場合、CSI ドライバーを登録するために **VolumeSnapshotClass** CR を作成する必要があります。
  - **DataProtectionApplication** CR でボリュームの場所を設定する必要があります。

手順

1. 次のコマンドを入力して、**backupStorageLocations** CR を取得します。

```
$ oc get backupStorageLocations -n openshift-adp
```

出力例

```
NAMESPACE       NAME              PHASE       LAST VALIDATED   AGE   DEFAULT
openshift-adp   velero-sample-1   Available   11s              31m
```

2. 次の例のように、**Backup** CR を作成します。

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup>
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  hooks: {}
  includedNamespaces:
  - <namespace> 1
  includedResources: [] 2
  excludedResources: [] 3
  storageLocation: <velero-sample-1> 4
  ttl: 720h0m0s
  labelSelector: 5
    matchLabels:
      app=<label_1>
      app=<label_2>
      app=<label_3>
  orLabelSelectors: 6
  - matchLabels:
      app=<label_1>
      app=<label_2>
      app=<label_3>
```

1. バックアップする namespace の配列を指定します。
2. オプション: バックアップに含めるリソースの配列を指定します。リソースは、ショートカット (Pods は po など) または完全修飾の場合があります。指定しない場合、すべてのリソースが含まれます。
3. オプション: バックアップから除外するリソースの配列を指定します。リソースは、ショートカット (Pods は po など) または完全修飾の場合があります。
4. **backupStorageLocations** CR の名前を指定します。
5. 指定したラベルを **すべて** 持つバックアップリソースの {key,value} ペアのマップ。
6. 指定したラベルを **1 つ以上** 持つバックアップリソースの {key,value} ペアのマップ。

3. **Backup** CR のステータスが **Completed** したことを確認します。

```
$ oc get backup -n openshift-adp <backup> -o jsonpath='{.status.phase}'
```

13.4.1.1. CSI スナップショットを使用した永続ボリュームのバックアップ

Backup CR を作成する前に、クラウドストレージの **VolumeSnapshotClass** カスタムリソース (CR) を編集して、Container Storage Interface (CSI) スナップショットを使用して永続ボリュームをバックアップします。

前提条件

- クラウドプロバイダーは、CSI スナップショットをサポートする必要があります。
- **DataProtectionApplication** CR で CSI を有効にする必要があります。

手順

- **metadata.labels.velero.io/csi-volumesnapshot-class: "true"** のキー: 値ペアを **VolumeSnapshotClass** CR に追加します。

  設定ファイルのサンプル

```
apiVersion: snapshot.storage.k8s.io/v1
kind: VolumeSnapshotClass
metadata:
  name: <volume_snapshot_class_name>
  labels:
    velero.io/csi-volumesnapshot-class: "true" (1)
  annotations:
    snapshot.storage.kubernetes.io/is-default-class: true (2)
driver: <csi_driver>
deletionPolicy: <deletion_policy_type> (3)
```

1. **true** に設定する必要があります。
2. **true** に設定する必要があります。

3 OADP は、CSI および Data Mover のバックアップと復元に対して、**Retain** および **Delete** 削除ポリシータイプをサポートしています。OADP 1.2 Data Mover の場合、削除ポリシー

次のステップ

- これで、**Backup** CR を作成できます。

13.4.1.2. Restic を使用したアプリケーションのバックアップ

Backup カスタムリソース (CR) を編集して、Restic を使用して Kubernetes リソース、内部イメージ、および永続ボリュームをバックアップします。

DataProtectionApplication CR でスナップショットの場所を指定する必要はありません。

重要

Restic は、**hostPath** ボリュームのバックアップをサポートしません。詳細は、追加の Restic 制限事項 を参照してください。

前提条件

- OpenShift API for Data Protection (OADP) Operator をインストールしている。
- **DataProtectionApplication** CR で **spec.configuration.restic.enable** を **false** に設定して、デフォルトの Restic インストールを無効にしていない。
- **DataProtectionApplication** CR が **Ready** 状態である。

手順

- 次の例のように、**Backup** CR を編集します。

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup>
  labels:
    velero.io/storage-location: default
  namespace: openshift-adp
spec:
  defaultVolumesToFsBackup: true 1
...
```

1 OADP バージョン 1.2 以降では、**defaultVolumesToFsBackup: true** 設定を **spec** ブロック内に追加します。OADP バージョン 1.1 では、**defaultVolumesToRestic: true** を追加します。

13.4.1.3. バックアップフックの作成

Backup カスタムリソース (CR) を編集して、Pod 内のコンテナーでコマンドを実行するためのバックアップフックを作成します。

プレ フックは、Pod のバックアップが作成される前に実行します。**ポスト** フックはバックアップ後に実行します。

手順

- 次の例のように、**Backup** CR の **spec.hooks** ブロックにフックを追加します。

```
apiVersion: velero.io/v1
kind: Backup
metadata:
  name: <backup>
  namespace: openshift-adp
spec:
  hooks:
    resources:
      - name: <hook_name>
        includedNamespaces:
        - <namespace> 1
        excludedNamespaces: 2
        - <namespace>
        includedResources:
        - pods 3
        excludedResources: [] 4
        labelSelector: 5
          matchLabels:
            app: velero
            component: server
        pre: 6
          - exec:
              container: <container> 7
              command:
              - /bin/uname 8
              - -a
              onError: Fail 9
              timeout: 30s 10
        post: 11
...
```

1. オプション: フックが適用される namespace を指定できます。この値が指定されていない場合、フックはすべてのネームスペースに適用されます。
2. オプション: フックが適用されない namespace を指定できます。
3. 現在、Pod は、フックを適用できる唯一のサポート対象リソースです。
4. オプション: フックが適用されないリソースを指定できます。
5. オプション: このフックは、ラベルに一致するオブジェクトにのみ適用されます。この値が指定されていない場合、フックはすべてのネームスペースに適用されます。
6. バックアップの前に実行するフックの配列。
7. オプション: コンテナーが指定されていない場合、コマンドは Pod の最初のコンテナーで実行されます。

8 これは、追加される init コンテナーのエントリーポイントです。

9 エラー処理に許可される値は、**Fail** と **Continue** です。デフォルトは **Fail** です。

10 オプション: コマンドの実行を待機する時間。デフォルトは **30s** です。

11 このブロックでは、バックアップ後に実行するフックの配列を、バックアップ前のフックと同じパラメーターで定義します。

13.5. 仮想マシンの復元

Restore CR を作成して、OpenShift API for Data Protection (OADP) の **Backup** カスタムリソース (CR) を復元します。

フック を **Restore** CR に追加して、アプリケーションコンテナーの起動前に init コンテナーでコマンドを実行するか、アプリケーションコンテナー自体でコマンドを実行することができます。

13.5.1. 復元 CR の作成

Restore CR を作成して、 **Backup** カスタムリソース (CR) を復元します。

前提条件

- OpenShift API for Data Protection (OADP) Operator をインストールしている。
- **DataProtectionApplication** CR が **Ready** 状態である。
- Velero **Backup** CR がある。
- 永続ボリューム (PV) の容量は、バックアップ時に要求されたサイズと一致する必要があります。必要に応じて、要求されたサイズを調整します。

手順

1. 次の例のように、**Restore** CR を作成します。

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: <restore>
  namespace: openshift-adp
spec:
  backupName: <backup> 1
  includedResources: [] 2
  excludedResources:
  - nodes
  - events
  - events.events.k8s.io
  - backups.velero.io
  - restores.velero.io
  - resticrepositories.velero.io
  restorePVs: true 3
```

1. **Backup** CR の名前
2. オプション: 復元プロセスに含めるリソースの配列を指定します。リソースは、ショートカット (**Pods** は **po** など) または完全修飾の場合があります。指定しない場合、すべてのリソースが含まれます。
3. オプション: **RestorePVs** パラメーターを **false** に設定すると、コンテナーストレージインターフェイス (CSI) スナップショットの **VolumeSnapshot** から、または **VolumeSnaphshotLocation** が設定されている場合はネイティブスナップショットからの **PersistentVolumes** の復元をオフにすることができます。

2. 次のコマンドを入力して、**Restore** CR のステータスが **Completed** であることを確認します。

```
$ oc get restore -n openshift-adp <restore> -o jsonpath='{.status.phase}'
```

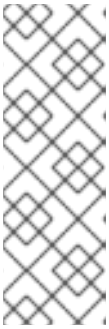
3. 次のコマンドを入力して、バックアップリソースが復元されたことを確認します。

```
$ oc get all -n <namespace> 1
```

1. バックアップした namespace。

4. ボリュームを使用して **DeploymentConfig** を復元する場合、または復元後のフックを使用する場合は、次のコマンドを入力して **dc-post-restore.sh** クリーンアップスクリプトを実行します。

```
$ bash dc-restic-post-restore.sh -> dc-post-restore.sh
```



注記

復元プロセス中に、OADP Velero プラグインは **DeploymentConfig** オブジェクトをスケールダウンし、Pod をスタンドアロン Pod として復元します。これは、クラスターが復元された **DeploymentConfig** Pod を復元時にすぐに削除することを防ぎ、復元フックと復元後のフックが復元された Pod 上でアクションを完了できるようにするために行われます。以下に示すクリーンアップスクリプトは、これらの切断された Pod を削除し、**DeploymentConfig** オブジェクトを適切な数のレプリカにスケールアップします。

例13.1 dc-restic-post-restore.sh → dc-post-restore.sh クリーンアップスクリプト

```
#!/bin/bash
set -e

# if sha256sum exists, use it to check the integrity of the file
if command -v sha256sum >/dev/null 2>&1; then
  CHECKSUM_CMD="sha256sum"
else
  CHECKSUM_CMD="shasum -a 256"
fi

label_name () {
    if [ "${#1}" -le "63" ]; then
 echo $1
```

```
 return
    fi
    sha=$(echo -n $1|$CHECKSUM_CMD)
    echo "${1:0:57}${sha:0:6}"
}

OADP_NAMESPACE=${OADP_NAMESPACE:=openshift-adp}

if [[ $# -ne 1 ]]; then
    echo "usage: ${BASH_SOURCE} restore-name"
    exit 1
fi

echo using OADP Namespace $OADP_NAMESPACE
echo restore: $1

label=$(label_name $1)
echo label: $label

echo Deleting disconnected restore pods
oc delete pods -l oadp.openshift.io/disconnected-from-dc=$label

for dc in $(oc get dc --all-namespaces -l oadp.openshift.io/replicas-modified=$label -o
jsonpath='{range .items[*]}{.metadata.namespace}{","}{.metadata.name}{","}
{.metadata.annotations.oadp\.openshift\.io/original-replicas}{","}
{.metadata.annotations.oadp\.openshift\.io/original-paused}{"\n"}')
do
    IFS=',' read -ra dc_arr <<< "$dc"
    if [ ${#dc_arr[0]} -gt 0 ]; then
 echo Found deployment ${dc_arr[0]}/${dc_arr[1]}, setting replicas: ${dc_arr[2]}, paused:
${dc_arr[3]}
 cat <<EOF | oc patch dc  -n ${dc_arr[0]} ${dc_arr[1]} --patch-file /dev/stdin
spec:
  replicas: ${dc_arr[2]}
  paused: ${dc_arr[3]}
EOF
    fi
done
```

13.5.1.1. 復元フックの作成

Restore カスタムリソース (CR) を編集して、Pod 内のコンテナーでコマンドを実行する復元フックを作成します。

2 種類の復元フックを作成できます。

- **init** フックは、init コンテナーを Pod に追加して、アプリケーションコンテナーが起動する前にセットアップタスクを実行します。
  Restic バックアップを復元する場合は、復元フック init コンテナーの前に **restic-wait** init コンテナーが追加されます。

- **exec** フックは、復元された Pod のコンテナーでコマンドまたはスクリプトを実行します。

手順

- 次の例のように、**Restore CR** の **spec.hooks** ブロックにフックを追加します。

```
apiVersion: velero.io/v1
kind: Restore
metadata:
  name: <restore>
  namespace: openshift-adp
spec:
  hooks:
    resources:
      - name: <hook_name>
        includedNamespaces:
        - <namespace> 1
        excludedNamespaces:
        - <namespace>
        includedResources:
        - pods 2
        excludedResources: []
        labelSelector: 3
          matchLabels:
            app: velero
            component: server
        postHooks:
        - init:
            initContainers:
            - name: restore-hook-init
              image: alpine:latest
              volumeMounts:
              - mountPath: /restores/pvc1-vm
                name: pvc1-vm
              command:
              - /bin/ash
              - -c
            timeout: 4
        - exec:
            container: <container> 5
            command:
            - /bin/bash 6
            - -c
            - "psql < /backup/backup.sql"
            waitTimeout: 5m 7
            execTimeout: 1m 8
            onError: Continue 9
```

1. オプション: フックが適用される namespace の配列。この値が指定されていない場合、フックはすべてのネームスペースに適用されます。
2. 現在、Pod は、フックを適用できる唯一のサポート対象リソースです。
3. オプション: このフックは、ラベルセレクターに一致するオブジェクトにのみ適用されます。
4. オプション: Timeout は、**initContainers** が完了するまで Velero が待機する最大時間を指定します。

5 オプション: コンテナーが指定されていない場合、コマンドは Pod の最初のコンテナーで実行されます。

6 これは、追加される init コンテナーのエントリーポイントです。

7 オプション: コンテナーの準備が整うまでの待機時間。これは、コンテナーが起動して同じコンテナー内の先行するフックが完了するのに十分な長さである必要があります。設定されていない場合、復元プロセスの待機時間は無期限になります。

8 オプション: コマンドの実行を待機する時間。デフォルトは **30s** です。

9 エラー処理に許可される値は、**Fail** および **Continue** です。

- **Continue**: コマンドの失敗のみがログに記録されます。
- **Fail**: Pod 内のコンテナーで復元フックが実行されなくなりました。**Restore** CR のステータスは **PartiallyFailed** になります。