



OpenShift Container Platform 4.4

Container-native Virtualization

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

OpenShift Container Platform 4.4 Container-native Virtualization

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

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Container-native_virtualization.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本書では、OpenShift Container Platform で Container-native Virtualization を使用する方法についての情報を提供します。

目次

第1章 CONTAINER-NATIVE VIRTUALIZATION について	9
1.1. CONTAINER-NATIVE VIRTUALIZATION の機能	9
1.2. CONTAINER-NATIVE VIRTUALIZATION のサポート	9
第2章 CONTAINER-NATIVE VIRTUALIZATION リリースノート	10
2.1. CONTAINER-NATIVE VIRTUALIZATION リリースノート	10
2.1.1. Container-native Virtualization 2.3 について	10
2.1.1.1. Container-native Virtualization の機能	10
2.1.1.2. Container-native Virtualization のサポート	10
2.1.2. 新機能および変更された機能	10
2.1.2.1. ネットワーク	11
2.1.2.2. ストレージ	11
2.1.2.3. Web コンソール	11
2.1.3. 主な技術上の変更点	12
2.1.4. 既知の問題	12
第3章 CONTAINER-NATIVE VIRTUALIZATION インストール	16
3.1. CONTAINER-NATIVE VIRTUALIZATION のクラスターの設定	16
3.2. CONTAINER-NATIVE VIRTUALIZATION のインストール	16
3.2.1. 前提条件	16
3.2.2. Container-native Virtualization カタログにサブスクライブします。	16
3.2.3. Container-native Virtualization のデプロイ	17
3.3. VIRTCTL クライアントのインストール	18
3.3.1. Container-native Virtualization リポジトリの有効化	18
3.3.2. virtctl クライアントのインストール	18
3.4. CONTAINER-NATIVE VIRTUALIZATION のアンインストール	19
3.4.1. 前提条件	19
3.4.2. CNV Operator Deployment カスタムリソースの削除	19
3.4.3. カタログサブスクリプションの削除	20
3.4.4. Web コンソールを使用した namespace の削除	20
第4章 CONTAINER-NATIVE VIRTUALIZATION のアップグレード	21
4.1. CONTAINER-NATIVE VIRTUALIZATION のアップグレードについて	21
4.1.1. Container-native Virtualization のアップグレードについて	21
4.1.2. Container-native Virtualization のクラスターへの作用について	21
4.2. CONTAINER-NATIVE VIRTUALIZATION の次のマイナーバージョンへのアップグレード	22
4.3. アップグレードステータスの監視	22
第5章 CLI ツールの使用	24
5.1. 前提条件	24
5.2. VIRTCTL クライアントコマンド	24
5.3. OPENSIFT CONTAINER PLATFORM クライアントコマンド	25
第6章 仮想マシン	26
6.1. 仮想マシンの作成	26
6.1.1. 仮想マシンウィザードの実行による仮想マシンの作成	26
6.1.1.1. 仮想マシンウィザードのフィールド	27
6.1.1.2. Cloud-init フィールド	29
6.1.1.3. CD-ROM フィールド	29
6.1.1.4. ネットワークフィールド	29
6.1.1.5. ストレージフィールド	30
6.1.2. 仮想マシンウィザードの作成用の事前に設定された YAML ファイルの貼り付け	30

6.1.3. CLI の使用による仮想マシンの作成	31
6.1.4. 仮想マシンのストレージボリュームタイプ	32
6.1.5. 追加リソース	33
6.2. 仮想マシンの編集	33
6.2.1. Web コンソールでの仮想マシンの編集	33
6.2.2. Web コンソールを使用した仮想マシンの YAML 設定の編集	34
6.2.3. CLI を使用した仮想マシン YAML 設定の編集	34
6.2.4. 仮想マシンへの仮想ディスクの追加	35
6.2.5. 仮想マシンへのネットワークインターフェイスの追加	35
6.2.6. 仮想マシンの CD-ROM の編集	36
6.3. ブート順序の編集	36
6.3.1. Web コンソールでのブート順序一覧への項目の追加	36
6.3.2. Web コンソールでのブート順序一覧の編集	37
6.3.3. YAML 設定ファイルでのブート順序一覧の編集	37
6.3.4. Web コンソールでのブート順序一覧からの項目の削除	38
6.4. 仮想マシンの削除	38
6.4.1. Web コンソールの使用による仮想マシンの削除	38
6.4.2. CLI の使用による仮想マシンの削除	39
6.5. 仮想マシンインスタンスの削除	39
6.5.1. 仮想マシンインスタンスの一覧表示	39
6.5.2. 仮想マシンインスタンスの削除	39
6.6. 仮想マシンの状態の制御	40
6.6.1. 仮想マシンの起動	40
6.6.2. 仮想マシンの再起動	41
6.6.3. 仮想マシンの停止	41
6.6.4. 仮想マシンの一時停止の解除	42
6.7. 仮想マシンコンソールへのアクセス	42
6.7.1. 仮想マシンコンソールのセッション	42
6.7.2. Web コンソールの使用による仮想マシンへの接続	43
6.7.2.1. ターミナルへの接続	43
6.7.2.2. シリアルコンソールへの接続	43
6.7.2.3. VNC コンソールへの接続	44
6.7.2.4. RDP コンソールへの接続	44
6.7.3. CLI コマンドの使用による仮想マシンコンソールへのアクセス	45
6.7.3.1. SSH 経由での仮想マシンインスタンスへのアクセス	45
6.7.3.2. 仮想マシンインスタンスのシリアルコンソールへのアクセス	45
6.7.3.3. VNC を使用した仮想マシンインスタンスのグラフィカルコンソールへのアクセス	46
6.7.3.4. RDP コンソールの使用による Windows 仮想マシンへの接続	46
6.8. VIRTIO ドライバーの既存の WINDOWS 仮想マシンへのインストール	47
6.8.1. VirtIO ドライバーについて	47
6.8.2. Microsoft Windows 仮想マシンのサポートされる VirtIO ドライバー	48
6.8.3. VirtIO ドライバーコンテナードiskの仮想マシンへの追加	48
6.8.4. VirtIO ドライバーの既存 Windows 仮想マシンへのインストール	49
6.8.5. 仮想マシンからの VirtIO コンテナードiskの削除	50
6.9. VIRTIO ドライバーの新規 WINDOWS 仮想マシンへのインストール	50
6.9.1. 前提条件	50
6.9.2. VirtIO ドライバーについて	50
6.9.3. Microsoft Windows 仮想マシンのサポートされる VirtIO ドライバー	51
6.9.4. VirtIO ドライバーコンテナードiskの仮想マシンへの追加	51
6.9.5. Windows インストール時の VirtIO ドライバーのインストール	52
6.9.6. 仮想マシンからの VirtIO コンテナードiskの削除	53
6.10. 高度な仮想マシン管理	53
6.10.1. 管理タスクの自動化	53

6.10.1.1. Red Hat Ansible Automation について	53
6.10.1.2. 仮想マシン作成の自動化	53
6.10.1.3. 例: 仮想マシンを作成するための Ansible Playbook	55
6.10.2. 仮想マシンの PXE ブートの設定	56
6.10.2.1. 前提条件	56
6.10.2.2. Container-native Virtualization ネットワークの用語集	56
6.10.2.3. MAC アドレスを指定した PXE ブート	56
6.10.2.4. テンプレート: PXE ブートの仮想マシンインスタンス設定ファイル	59
6.10.3. ゲストメモリーの管理	60
6.10.3.1. ゲストメモリーのオーバーコミットの設定	60
6.10.3.2. ゲストメモリーオーバーヘッドアカウンティングの有効化	60
6.10.4. 仮想マシン用の専用リソースの有効化	61
6.10.4.1. 専用リソースについて	61
6.10.4.2. 前提条件	61
6.10.4.3. 仮想マシンの専用リソースの有効化	62
6.11. 仮想マシンのインポート	62
6.11.1. DataVolume インポートの TLS 証明書	62
6.11.1.1. DataVolume インポートの認証に使用する TLS 証明書の追加	62
6.11.1.2. 例: TLS 証明書から作成される ConfigMap	62
6.11.2. DataVolume の使用による仮想マシンイメージのインポート	63
6.11.2.1. 前提条件	63
6.11.2.2. CDI がサポートする操作マトリックス	63
6.11.2.3. DataVolume について	64
6.11.2.4. DataVolume のあるオブジェクトへの仮想マシンイメージのインポート	64
6.11.2.5. テンプレート: DataVolume 仮想マシン設定ファイル	66
6.11.2.6. テンプレート: DataVolume インポート設定ファイル	67
6.11.3. DataVolume の使用による仮想マシンイメージのブロックストレージへのインポート	68
6.11.3.1. 前提条件	68
6.11.3.2. DataVolume について	68
6.11.3.3. ブロック PersistentVolume について	68
6.11.3.4. ローカルブロック PersistentVolume の作成	69
6.11.3.5. DataVolume を使用した仮想マシンイメージのブロック PersistentVolume へのインポート	70
6.11.3.6. CDI がサポートする操作マトリックス	71
6.11.4. VMware 仮想マシンまたはテンプレートのインポート	72
6.11.4.1. VDDK イメージのイメージレジストリーの設定	73
6.11.4.1.1. 内部イメージレジストリーの設定	73
6.11.4.1.2. 内部イメージレジストリーへのアクセスの設定	74
6.11.4.1.3. 外部イメージレジストリーへのアクセスの設定	77
6.11.4.2. VDDK イメージの作成および使用	78
6.11.4.3. 仮想マシンウィザードによる VMware 仮想マシンまたはテンプレートのインポート	80
6.11.4.4. インポートされた VMware 仮想マシンの NIC 名の更新	82
6.11.4.5. VMware 仮想マシンのインポートのトラブルシューティング	82
6.11.4.5.1. エラーメッセージ	83
6.11.4.5.2. 既知の問題	83
6.11.4.6. 仮想マシンウィザードのフィールド	83
6.11.4.6.1. 仮想マシンウィザードのフィールド	83
6.11.4.6.2. Cloud-init フィールド	85
6.11.4.6.3. ネットワークフィールド	85
6.11.4.6.4. ストレージフィールド	85
6.12. 仮想マシンのクローン作成	86
6.12.1. 複数の namespace 間で DataVolume 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	86
6.12.1.1. 前提条件	86
6.12.1.2. DataVolume について	86

6.12.1.3. DataVolume のクローン作成のための RBAC リソースの作成	86
6.12.2. 新規 DataVolume への仮想マシンディスクのクローン作成	88
6.12.2.1. 前提条件	88
6.12.2.2. DataVolume について	88
6.12.2.3. 新規 DataVolume への仮想マシンディスクの PersistentVolumeClaim のクローン作成	88
6.12.2.4. テンプレート: DataVolume クローン設定ファイル	89
6.12.2.5. CDI がサポートする操作マトリックス	90
6.12.3. DataVolumeTemplate の使用による仮想マシンのクローン作成	90
6.12.3.1. 前提条件	90
6.12.3.2. DataVolume について	90
6.12.3.3. DataVolumeTemplate の使用による、クローン作成された PersistentVolumeClaim からの仮想マシンの新規作成	90
6.12.3.4. テンプレート: DataVolume 仮想マシン設定ファイル	92
6.12.3.5. CDI がサポートする操作マトリックス	93
6.12.4. 新規ブロックストレージ DataVolume への仮想マシンディスクのクローン作成	93
6.12.4.1. 前提条件	93
6.12.4.2. DataVolume について	94
6.12.4.3. ブロック PersistentVolume について	94
6.12.4.4. ローカルブロック PersistentVolume の作成	94
6.12.4.5. 新規 DataVolume への仮想マシンディスクの PersistentVolumeClaim のクローン作成	95
6.12.4.6. CDI がサポートする操作マトリックス	96
6.13. 仮想マシンのネットワーク	97
6.13.1. 仮想マシンのデフォルト Pod ネットワークの使用	97
6.13.1.1. コマンドラインでのマスカレードモードの設定	97
6.13.1.2. バインディング方法の選択	98
6.13.1.2.1. ネットワークフィールド	98
6.13.1.3. デフォルトネットワーク用の仮想マシン設定の例	99
6.13.1.3.1. テンプレート: 仮想マシンの設定ファイル	99
6.13.1.3.2. テンプレート: Windows 仮想マシンインスタンスの設定ファイル	99
6.13.2. 仮想マシンの複数ネットワークへの割り当て	100
6.13.2.1. Container-native Virtualization ネットワークの用語集	101
6.13.2.2. NetworkAttachmentDefinition の作成	101
6.13.2.3. 前提条件	101
6.13.2.3.1. Web コンソールでの Linux ブリッジ NetworkAttachmentDefinition の作成	101
6.13.2.3.2. CLI での Linux ブリッジ NetworkAttachmentDefinition の作成	102
6.13.2.4. 仮想マシンの NIC の作成	104
6.13.2.5. ネットワークフィールド	104
6.13.3. QEMU ゲストエージェントの仮想マシンへのインストール	105
6.13.3.1. 前提条件	105
6.13.3.2. QEMU ゲストエージェントの Linux 仮想マシンへのインストール	105
6.13.3.3. QEMU ゲストエージェントの Windows 仮想マシンへのインストール	105
6.13.3.3.1. VirtIO ドライバーの既存 Windows 仮想マシンへのインストール	105
6.13.3.3.2. Windows インストール時の VirtIO ドライバーのインストール	106
6.13.4. vNIC の IP アドレスの仮想マシンへの表示	107
6.13.4.1. 前提条件	107
6.13.4.2. CLI での仮想マシンインターフェイスの IP アドレスの表示	107
6.13.4.3. Web コンソールでの仮想マシンインターフェイスの IP アドレスの表示	108
6.14. 仮想マシンディスク	108
6.14.1. 仮想マシンのローカルストレージの設定	108
6.14.1.1. ホストパスプロビジョナーについて	108
6.14.1.2. Red Hat Enterprise Linux CoreOS 8 でのホストパスプロビジョナー用の SELinux の設定	109
6.14.1.3. 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有効化	110
6.14.1.4. StorageClass オブジェクトの作成	111

6.14.2. virtctl ツールの使用によるローカルディスクイメージのアップロード	111
6.14.2.1. 前提条件	112
6.14.2.2. DataVolume について	112
6.14.2.3. アップロード DataVolume の作成	112
6.14.2.4. ローカルディスクイメージの DataVolume へのアップロード	112
6.14.2.5. CDI がサポートする操作マトリックス	114
6.14.3. ブロックストレージ DataVolume へのローカルディスクイメージのアップロード	114
6.14.3.1. 前提条件	114
6.14.3.2. DataVolume について	114
6.14.3.3. ブロック PersistentVolume について	115
6.14.3.4. ローカルブロック PersistentVolume の作成	115
6.14.3.5. アップロード DataVolume の作成	116
6.14.3.6. ローカルディスクイメージの DataVolume へのアップロード	117
6.14.3.7. CDI がサポートする操作マトリックス	118
6.14.4. ローカル仮想マシンディスクの別のノードへの移動	118
6.14.4.1. ローカルボリュームの別のノードへのクローン作成	119
6.14.5. 空のディスクイメージを追加して仮想ストレージを拡張する	121
6.14.5.1. DataVolume について	121
6.14.5.2. DataVolume を使用した空のディスクイメージの作成	121
6.14.5.3. テンプレート: 空のディスクイメージの DataVolume 設定ファイル	122
6.14.6. DataVolume のストレージのデフォルト	122
6.14.6.1. DataVolume のストレージ設定について	123
6.14.6.1.1. アクセスモード	123
6.14.6.1.2. ボリュームモード	123
6.14.6.2. Web コンソールでの kubevirt-storage-class-defaults ConfigMap の編集	123
6.14.6.3. CLI での kubevirt-storage-class-defaults ConfigMap の編集	124
6.14.6.4. 複数のストレージクラスのデフォルト例	125
6.14.7. CDI のスクラッチ領域の用意	125
6.14.7.1. DataVolume について	125
6.14.7.2. スクラッチ領域について	125
手動プロビジョニング	126
6.14.7.3. スクラッチ領域を必要とする CDI 操作	126
6.14.7.4. CDI 設定での StorageClass の定義	126
6.14.7.5. CDI がサポートする操作マトリックス	127
6.14.8. DataVolume の削除	127
6.14.8.1. DataVolume について	128
6.14.8.2. すべての DataVolume の一覧表示	128
6.14.8.3. DataVolume の削除	128
第7章 仮想マシンテンプレート	129
7.1. 仮想マシンテンプレートの作成	129
7.1.1. Web コンソールでのインタラクティブなウィザードを使用した仮想マシンテンプレートの作成	129
7.1.2. 仮想マシンテンプレートのインタラクティブなウィザードのフィールド	130
7.1.2.1. 仮想マシンテンプレートウィザードのフィールド	130
7.1.2.2. Cloud-init フィールド	131
7.1.2.3. ネットワークフィールド	131
7.1.2.4. ストレージフィールド	132
7.2. 仮想マシンテンプレートの編集	133
7.2.1. Web コンソールでの仮想マシンテンプレートの編集	133
7.2.2. Web コンソールでの仮想マシンテンプレート YAML 設定の編集	133
7.2.3. 仮想マシンテンプレートへの仮想ディスクの追加	133
7.2.4. ネットワークインターフェイスの仮想マシンテンプレートへの追加	134
7.2.5. 仮想マシンテンプレートの CD-ROM の編集	134

7.3. 仮想マシンテンプレートの専用リソースの有効化	135
7.3.1. 専用リソースについて	135
7.3.2. 前提条件	135
7.3.3. 仮想マシンテンプレートの専用リソースの有効化	135
7.4. 仮想マシンテンプレートの削除	136
7.4.1. Web コンソールでの仮想マシンテンプレートの削除	136
第8章 ライブマイグレーション	137
8.1. 仮想マシンのライブマイグレーション	137
8.1.1. 前提条件	137
8.1.2. ライブマイグレーションについて	137
8.1.3. LiveMigration のアクセスモードの更新	137
8.2. ライブマイグレーションの制限およびタイムアウト	137
8.2.1. ライブマイグレーションの制限およびタイムアウトの設定	137
8.2.2. クラスター全体のライブマイグレーションの制限およびタイムアウト	138
8.3. 仮想マシンインスタンスの別のノードへの移行	139
8.3.1. Web コンソールでの仮想マシンインスタンスのライブマイグレーションの開始	139
8.3.2. CLI での仮想マシンインスタンスのライブマイグレーションの開始	140
8.4. 仮想マシンインスタンスのライブマイグレーションのモニター	140
8.4.1. Web コンソールでの仮想マシンインスタンスのライブマイグレーションのモニター	140
8.4.2. CLI での仮想マシンインスタンスのライブマイグレーションのモニター	140
8.5. 仮想マシンインスタンスのライブマイグレーションの取り消し	141
8.5.1. Web コンソールでの仮想マシンインスタンスのライブマイグレーションの取り消し	141
8.5.2. CLI での仮想マシンインスタンスのライブマイグレーションの取り消し	142
8.6. 仮想マシンのエビクションストラテジーの設定	142
8.6.1. LiveMigration エビクションストラテジーでのカスタム仮想マシンの設定	142
第9章 ノードのメンテナンス	143
9.1. TLS 証明書の手動更新	143
9.1.1. TLS 証明書の更新	143
9.2. ノードのメンテナンスモード	143
9.2.1. ノードのメンテナンスモードについて	143
9.3. ノードのメンテナンスモードへの設定	144
9.3.1. ノードのメンテナンスモードについて	144
9.3.2. Web コンソールでのノードのメンテナンスモードへの設定	144
9.3.3. CLI でのノードのメンテナンスモードへの設定	145
9.4. メンテナンスモードからのノードの再開	145
9.4.1. Web コンソールでのメンテナンスモードからのノードの再開	145
9.4.2. CLI でのメンテナンスモードからのノードの再開	146
第10章 ノードのネットワーク	148
10.1. ノードのネットワーク状態の確認	148
10.1.1. nmstate について	148
10.1.2. ノードのネットワーク状態の表示	148
10.2. ノードのネットワーク設定の更新	149
10.2.1. nmstate について	149
10.2.2. ノード上でのインターフェイスの作成	149
10.2.3. ノード上でのポリシー更新の確認	150
10.2.4. ノードからインターフェイスの削除	151
10.2.5. インターフェイスの削除後のノードネットワーク設定の復元	152
10.2.6. 例: Linux ブリッジインターフェイス NodeNetworkConfigurationPolicy	152
10.2.7. 例: VLAN インターフェイス NodeNetworkConfigurationPolicy	153
10.2.8. 例: ボンドインターフェイス NodeNetworkConfigurationPolicy	154
10.3. ノードのネットワーク設定のトラブルシューティング	156

10.3.1. 正確でない NodeNetworkConfigurationPolicy 設定のトラブルシューティング	156
第11章 ログイン、イベント、およびモニタリング	160
11.1. 仮想マシンログの表示	160
11.1.1. 仮想マシンのログについて	160
11.1.2. CLI での仮想マシンログの表示	160
11.1.3. Web コンソールでの仮想マシンログの表示	160
11.2. イベントの表示	160
11.2.1. 仮想マシンイベントについて	161
11.2.2. Web コンソールでの仮想マシンのイベントの表示	161
11.2.3. CLI での namespace イベントの表示	161
11.2.4. CLI でのリソースイベントの表示	161
11.3. 仮想マシンのワークロードに関する情報の表示	161
11.3.1. Virtual Machines ダッシュボードについて	162
11.4. 仮想マシンの正常性のモニタリング	162
11.4.1. liveness および readiness プローブ	163
11.4.2. HTTP liveness プローブの定義	163
11.4.3. TCP liveness プローブの定義	164
11.4.4. readiness プローブの定義	165
11.5. OPENSIFT CONTAINER PLATFORM DASHBOARD を使用したクラスター情報の取得	166
11.5.1. OpenShift Container Platform ダッシュボードページについて	166
11.6. OPENSIFT CONTAINER PLATFORM クラスターモニタリング、ログイン、および TELEMETRY	167
11.6.1. OpenShift Container Platform クラスターモニタリングについて	167
11.6.2. クラスターログインコンポーネント	168
11.6.3. Telemetry について	168
11.6.3.1. Telemetry で収集される情報	169
11.6.4. CLI のトラブルシューティングおよびデバッグコマンド	169
11.7. RED HAT サポート向けの CONTAINER-NATIVE VIRTUALIZATION データの収集	169
11.7.1. must-gather ツールについて	170
11.7.2. Container-native Virtualization データの収集について	170
11.7.3. 特定の機能に関するデータ収集	170

第1章 CONTAINER-NATIVE VIRTUALIZATION について

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

1.1. CONTAINER-NATIVE VIRTUALIZATION の機能

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

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

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

機能強化された Web コンソールは、これらの仮想化されたリソースを OpenShift Container Platform クラスターコンテナ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。

Container-native Virtualization は [OVN-Kubernetes](#) または [OpenShiftSDN](#) ネットワークプロバイダーのいずれかを使って使用できます。

1.2. CONTAINER-NATIVE VIRTUALIZATION のサポート



重要

Container-native Virtualization はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。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ついての詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

第2章 CONTAINER-NATIVE VIRTUALIZATION リリースノート

2.1. CONTAINER-NATIVE VIRTUALIZATION リリースノート

2.1.1. Container-native Virtualization 2.3 について

2.1.1.1. Container-native Virtualization の機能

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

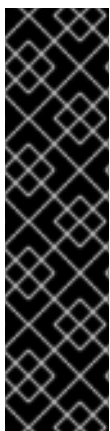
Container-native Virtualization は、Kubernetes カスタムリソースを使って新規オブジェクトを OpenShift Container Platform クラスタに追加し、仮想化タスクを有効にします。これらのタスクには、以下が含まれます。

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

機能強化された Web コンソールは、これらの仮想化されたリソースを OpenShift Container Platform クラスタコンテナ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。

Container-native Virtualization は [OVN-Kubernetes](#) または [OpenShiftSDN](#) ネットワークプロバイダーのいずれかを使って使用できます。

2.1.1.2. Container-native Virtualization のサポート



重要

Container-native Virtualization はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。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ついての詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

2.1.2. 新機能および変更された機能

- Container-native Virtualization は、Windows Server のワークロードを実行する Microsoft の Windows Server Virtualization Validation Program (SVVP) で認定されています。

- SVVP の認定は、Intel および AMD の CPU に適用されます。
- [SVVP 認定](#) Red Hat Enterprise Linux CoreOS 8 ワーカーの名前は、 **Red Hat OpenShift Container Platform 4 on RHEL CoreOS 8** です。
- Microsoft Windows 10 と互換性がある新しいテンプレートを利用できます。

2.1.2.1. ネットワーク

- Container-native Virtualization は、[OVN-Kubernetes](#) または [OpenShiftSDN](#) ネットワークプロバイダーのいずれかを使って使用できます。
- Container-native Virtualization は [nmstate](#) を使用してノードネットワークの状態を報告し、設定します。単一の設定マニフェストをクラスターに適用して、すべてのノードに Linux ブリッジ、ボンドおよび VLAN デバイスを作成するなどして、ネットワークポリシーの設定を変更することができます。
- 詳細は、[ノードのネットワーク](#) の章を参照してください。

2.1.2.2. ストレージ

- これで、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。
- **virtctl** ツールは、サーバー側のアップロードの後処理を非同期的に監視し、[仮想マシンのディスクのアップロード](#) のステータスをより正確に報告できるようになりました。

2.1.2.3. Web コンソール

- Web コンソールで仮想マシンの **Paused** ステータスを表示できるようになりました。Web コンソールでは、**Virtual Machines** ダッシュボードと **Virtual Machine Details** ページにこのステータスが表示されます。



注記

oc CLI を使用して、仮想マシンの **Paused** ステータスを表示することもできます。

```
$ oc get vmi testvm -o=jsonpath='{.status.conditions[?(@.type=="Paused")].message}'
```

- 仮想マシンのステータスが **Paused** の場合、[Web コンソールからその一時停止を解除](#) できます。
- 仮想マシンの一覧内の仮想マシンから切り離された仮想マシンインスタンスを表示し、管理できます。
- Container-native Virtualization を使用すると、[仮想マシンウィザード](#) で CD-ROM を設定できます。ドロップダウンリストから CD-ROM 設定のタイプ (**Container**、**URL**、または **Attach Disk**) を選択できます。また、**Virtual Machine Details** ページで [CD-ROM 設定を編集](#) することもできます。
- [ブート順序の一覧](#) が **Virtual Machine Details** で利用可能になりました。項目を追加したり、削除したり、またブート順序の一覧を変更したりすることができます。
- 仮想マシンおよび仮想マシンテンプレートウィザードでは、さまざまなオペレーティングシステムの CPU およびメモリーサイズおよびディスクバスの要件を検証できるようになりました。

特定のオペレーティングシステムの要件を満たさない仮想マシンまたは仮想マシンテンプレートを作成しようとする、ウィザードによりリソースの警告が表示されます。

- **Virtual Machine Details** ページで、[仮想マシンの専用リソース](#) を表示し、設定できるようになりました。専用リソースを有効にしたら、仮想マシンのワークロードが他のプロセスで使用されていない CPU で実行されることを確認します。これにより、仮想マシンのパフォーマンスとレイテンシー予測の精度が向上します。

2.1.3. 主な技術上の変更点

- Container-native Virtualization を、**openshift-cnv** という名前の単一 namespace にデプロイする必要があります。**openshift-cnv** namespace は、存在していない場合、デプロイメント時に自動的に作成されるようになりました。
- エラーを防ぐために、Container-native Virtualization デプロイメント時に作成する **CNV Operator Deployment** カスタムリソースの名前を変更できなくなりました。デフォルト名の **kubevirt-hyperconverged** という名前のカスタムリソースを作成しようとする場合、作成が失敗し、エラーメッセージが Web コンソールに表示されます。
- 意図しないデータ損失を防ぐために、クラスターに仮想マシンまたは DataVolume が定義されている場合は、Container-native Virtualization をアンインストールできなくなりました。
 - Container-native Virtualization をアンインストールする前に、すべての [仮想マシン](#) (VM)、[仮想マシンインスタンス](#) (VMI)、および [DataVolume](#) (DV) を手動で削除する必要があります。
 - Container-native Virtualization のアンインストールを試行する際に VM、VMI、または DV オブジェクトが存在する場合は、残りのオブジェクトが削除されるまでアンインストールプロセスは完了しません。



注記

保留中のオブジェクトが原因でアンインストールが一時停止されていることを確認するには、[Events](#) タブを表示します。

2.1.4. 既知の問題

- KubeMacPool は Container-native Virtualization 2.3 で無効にされています。つまり、Pod または仮想マシンのセカンダリーインターフェイスは、プールから一意のアドレスではなく、無作為に生成された MAC アドレスを取得します。稀なケースで、無作為に割り当てられた MAC アドレスが競合する可能性があります。(BZ#1816971)
- Web コンソールで **Container-native Virtualization Operator** のサブスクリプションチャンネルを編集しようとする際に、**Subscription Overview** の **Channel** ボタンをクリックすると JavaScript エラーが発生することがあります。(BZ#1796410)
 - 回避策として、以下の **oc** patch コマンドを実行して、CLI から Container-native Virtualization 2.3 へのアップグレードプロセスをトリガーします。

```
$ export TARGET_NAMESPACE=openshift-cnv CNV_CHANNEL=2.3 && oc patch -n
"${TARGET_NAMESPACE}" $(oc get subscription -n ${TARGET_NAMESPACE} --no-
headers -o name) --type='json' -p='[{"op": "replace", "path": "/spec/channel",
"value": "${CNV_CHANNEL}"}, {"op": "replace", "path": "/spec/installPlanApproval",
"value": "Automatic"}]'
```


このコマンドは、アップグレードチャネル **2.3** にサブスクリプションをポイントし、自動更新を有効にします。

- 仮想マシンおよび仮想マシンテンプレートウィザードでは、**virtIO** が CD-ROM を割り当てる際にデフォルトのインターフェイスになります。ただし、**virtIO** CD-ROM は仮想マシンの検証をパスせず、これを作成できません。(BZ#1817394)
 - 回避策として、仮想マシンおよび仮想マシンテンプレートを作成する際に、**SATA** を CD-ROM インターフェイスとして選択します。
- Containerized Data Importer (CDI) は、インポートおよびアップロード操作に CDIConfig オブジェクトの **scratchSpaceStorageClass** 設定を常に使用する訳ではありません。代わりに、CDI はデフォルトのストレージクラスを使用してスクラッチ領域を割り当てます。(BZ#1828198)
 - 回避策として、クラスターのデフォルトストレージクラスが定義されていることを確認します。以下のコマンドを使用して、必要なアノテーションを適用できます。

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

- 以前のバージョンの Container-native Virtualization のデプロイ時に Operator デプロイメントのカスタムリソースの名前を変更した場合、Container-native Virtualization 2.3 に直接アップグレードすることはできません。カスタムリソースは、デフォルト名の **kubevirt-hyperconverged** という名前にする必要があります。(BZ#1822266)
 - 回避策として、以下のいずれかを実行できます。
 - 既存のカスタムリソースの名前を **kubevirt-hyperconverged** に変更します。
 - デフォルトの **kubevirt-hyperconverged** という名前の新規のカスタムリソースを作成します。次に、**kubevirt-hyperconverged** という名前のカスタムリソースを削除します。
- OpenShift Container Platform 4.4 Web コンソールには、NIC を仮想マシンに追加する際に **slirp** がオプションとして含まれますが、**slirp** は有効な NIC タイプではありません。NIC を仮想マシンに追加する場合、**slirp** は選択しないでください。(BZ#1828744)
- 移行後、仮想マシンには新規の IP アドレスが割り当てられます。ただし、コマンドの **oc get vmi** および **oc describe vmi** は古くなった IP アドレスを含む出力を依然として出力します。(BZ#1686208)
 - 回避策として、以下のコマンドを実行して正しい IP アドレスを表示します。

```
$ oc get pod -o wide
```

- 管理者権限を持たないユーザーは、仮想マシンのウィザードを使用して、L2 ネットワークのプロジェクトにネットワークインターフェイスを追加できません。この問題は、ユーザーがネットワーク割り当て定義を読み込む権限がないために発生します。(BZ#1743985)
 - 回避策として、ネットワークの割り当て定義を読み込むパーミッションを持つユーザーを指定します。
 1. 以下の例を使用して **ClusterRole** および **ClusterRoleBinding** オブジェクトを YAML 設定ファイルに定義します。

```
apiVersion: rbac.authorization.k8s.io/v1
```

```
kind: ClusterRole
metadata:
  name: cni-resources
rules:
- apiGroups: ["k8s.cni.cncf.io"]
  resources: ["*"]
  verbs: ["*"]
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: <role-binding-name>
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cni-resources
subjects:
- kind: User
  name: <user to grant the role to>
  namespace: <namespace of the user>
```

2. **cluster-admin** ユーザーとして以下のコマンドを実行し、定義した **ClusterRole** および **ClusterRoleBinding** オブジェクトを作成します。

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

- ノードの CPU モデルが異なると、ライブマイグレーションに失敗します。ノードに同じ物理 CPU モデルがある場合でも、マイクロコードの更新によって導入される違いにより同じ状況が生じます。これは、デフォルト設定が、ライブマイグレーションと互換性のないホスト CPU パススルー動作をトリガーするためです。(BZ#1760028)
- 回避策として、以下の例のように **kubevirt-config** ConfigMap にデフォルトの CPU モデルを設定します。



注記

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

1. 以下のコマンドを実行して、編集するために **kubevirt-config** ConfigMap を開きます。

```
$ oc edit configmap kubevirt-config -n openshift-cnv
```

2. ConfigMap を編集します。

```
kind: ConfigMap
metadata:
  name: kubevirt-config
data:
  default-cpu-model: "<cpu-model>" 1
```

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

- Haswell CPU を使用して仮想マシンの作成および起動を試行する場合、仮想マシンの起動が誤ってラベルが付けられたノードによって失敗する可能性があります。これは、以前のバージョンの Container-native Virtualization からの動作の変更点になります。この場合、仮想マシンは Haswell ホストで正常に起動されました。(BZ#1781497)
回避策として、可能であれば別の CPU モデルを選択します。
- Container-native Virtualization のアップグレードプロセスは、Operator Lifecycle Manager (OLM) の中断により失敗する可能性があります。この問題は、Container-native Virtualization Operator の状態を追跡するために宣言型 API を使用することに関連する制限によって生じます。インストール時に自動更新を有効にすることにより、この問題が発生するリスクが低くなります。(BZ#1759612)
- Container-native Virtualization は、**oc adm drain** または **kubectl drain** のいずれかを実行してトリガーされるノードのドレイン (解放) を確実に特定することができません。これらのコマンドは、Container-native Virtualization がデプロイされているクラスタのノードでは実行しないでください。ノードは、それらの上部で実行されている仮想マシンがある場合にドレイン (解放) を実行しない可能性があります。現時点の解決策として、ノードをメンテナンス状態にする方法があります(BZ#1707427)
- **Operators → Installed Operators** ページで **Subscription** タブに移動し、現在のアップグレードチャンネルをクリックしてこれを編集する場合、結果が表示されない可能性があります。これが発生すると、エラーは表示されません。(BZ#1796410)
 - 回避策として、以下の **oc patch** コマンドを実行して、CLI から Container-native Virtualization 2.3 へのアップグレードプロセスをトリガーします。

```
$ export TARGET_NAMESPACE=openshift-cnv CNV_CHANNEL=2.3 && oc patch -n  
"${TARGET_NAMESPACE}" $(oc get subscription -n ${TARGET_NAMESPACE} --no-  
headers -o name) --type='json' -p='[{"op": "replace", "path": "/spec/channel",  
"value": "${CNV_CHANNEL}"}, {"op": "replace", "path": "/spec/installPlanApproval",  
"value": "Automatic"}]'
```

このコマンドは、アップグレードチャンネル **2.3** にサブスクリプションをポイントし、自動更新を有効にします。

第3章 CONTAINER-NATIVE VIRTUALIZATION インストール

3.1. CONTAINER-NATIVE VIRTUALIZATION のクラスターの設定

OpenShift Container Platform の評価版バージョンを取得するには、OpenShift Container Platform ホームページから評価版をダウンロードしてください。

Container-native Virtualization はデフォルトで OpenShift Container Platform と連携しますが、以下のインストール設定が推奨されます。

- OpenShift Container Platform クラスターを [ベアメタル](#) にインストールします。コンピューターノードを、クラスター内でホストする仮想マシンの数およびサイズに応じて管理します。
- [モニタリング](#) をクラスターに設定します。

3.2. CONTAINER-NATIVE VIRTUALIZATION のインストール

Container-native Virtualization をインストールし、仮想化機能を OpenShift Container Platform クラスターに追加します。

OpenShift Container Platform 4.4 [Web コンソール](#) を使用して、Container-native Virtualization Operator にサブスクライブし、これをデプロイすることができます。

3.2.1. 前提条件

- OpenShift Container Platform 4.4



重要

Container-native Virtualization はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。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ついての詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

3.2.2. Container-native Virtualization カタログにサブスクライブします。

Container-native Virtualization をインストールする前に、OpenShift Container Platform Web コンソールから Container-native virtualization カタログにサブスクライブします。サブスクライブにより、**openshift-cnv** namespace に Container-native Virtualization Operator へのアクセスが付与されます。

手順

1. ブラウザーウィンドウを開き、OpenShift Container Platform Web コンソールにログインします。
2. **Operators** → **OperatorHub** ページに移動します。

3. **OpenShift Virtualization** を検索し、これを選択します。
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. **Create Operator Subscription** ページで以下を実行します。
 - a. 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**

Container-native 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。

- b. 選択可能な **Update Channel** オプションから **2.3** を選択します。
 - c. **Approval Strategy** では、デフォルト値である **Automatic** が選択されていることを確認します。Container-native Virtualization は、z-stream の新規リリースが利用可能になると自動的に更新されます。
6. **Subscribe** をクリックし、Operator を **openshift-cnv** namespace で利用可能にします。**Installed Operators** 画面では、Container-native Virtualization がインストールを終了すると、**Status** に **Succeeded** が表示されます。

3.2.3. Container-native Virtualization のデプロイ

CNV Operator Deployment カスタムリソースを作成し、container-native virtualization をデプロイします。

前提条件

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

手順

1. **Operators** → **Installed Operators** ページに移動します。
2. **OpenShift Virtualization** をクリックします。
3. **CNV Operator Deployment** タブをクリックし、**Create HyperConverged Cluster** をクリックします。



警告

デプロイメントのエラーを回避するには、カスタムリソースの名前を変更しないでください。次のステップに進む前に、カスタムリソースの名前がデフォルトの **kubevirt-hyperconverged** であることを確認します。

4. YAML を表示して、**BareMetalPlatform: true** であることを確認します。必要な場合は、値を **false** から **true** に変更してから続行します。
5. **Create** をクリックして Container-native Virtualization を起動します。
6. **Workloads → Pods** ページに移動して、Container-native 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、Container-native Virtualization にアクセスできます。

3.3. VIRTCTL クライアントのインストール

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

Container-native Virtualization リポジトリを有効にし、**kubevirt-virtctl** パッケージをインストールしてクライアントをシステムにインストールします。

3.3.1. Container-native Virtualization リポジトリの有効化

Red Hat は、Red Hat Enterprise Linux 8 および Red Hat Enterprise Linux 7 向けの Container-native Virtualization リポジトリを提供します。

- Red Hat Enterprise Linux 8 リポジトリ: **cnv-2.3-for-rhel-8-x86_64-rpms**
- Red Hat Enterprise Linux 7 リポジトリ: **rhel-7-server-cnv-2.3-rpms**

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

手順

- **subscription manager** を使用して、お使いのシステムに適した Container-native Virtualization リポジトリを有効にします。

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

3.3.2. virtctl クライアントのインストール

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

手順

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

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

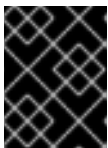
Container-native Virtualization の [CLI ツールの使用](#) も参照してください。

3.4. CONTAINER-NATIVE VIRTUALIZATION のアンインストール

OpenShift Container Platform [Web コンソール](#) を使用して Container-native Virtualization をアンインストールできます。

3.4.1. 前提条件

- Container-native Virtualization 2.3 がインストールされている必要があります。
- すべての [仮想マシン](#)、[仮想マシンインスタンス](#)、および [DataVolume](#) を削除する必要があります。



重要

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


3.4.2. CNV Operator Deployment カスタムリソースの削除

Container-native Virtualization をアンインストールするには、まず **CNV Operator Deployment** カスタムリソースを削除する必要があります。

前提条件

- アクティブな **CNV Operator Deployment** カスタムリソース

手順

1. OpenShift Container Platform Web コンソールから、**Projects** 一覧より **openshift-cnv** を選択します。
2. **Operators** → **Installed Operators** ページに移動します。
3. **OpenShift Virtualization** をクリックします。
4. **CNV Operator Deployment** タブをクリックします。
5. ディスクの Options メニュー  を **kubevirt-hyperconverged** カスタムリソースを含む行でクリックします。拡張されたメニューで、**Delete HyperConverged Cluster** をクリックします。
6. 確認ウィンドウで **Delete** をクリックします。
7. **Workloads** → **Pods** ページに移動し、Operator Pod のみが実行中であることを確認します。
8. ターミナルウィンドウを開き、以下のコマンドを実行して残りのリソースをクリーンアップします。

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

3.4.3. カタログサブスクリプションの削除

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

前提条件

- Container-native virtualization カタログへのアクティブなサブスクリプション

手順

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



注記

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

3.4.4. Web コンソールを使用した namespace の削除


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



注記

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

手順

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

第4章 CONTAINER-NATIVE VIRTUALIZATION のアップグレード

次のマイナーバージョンの Container-native Virtualization に手動でアップグレードし、Web コンソールを使用して更新のステータスをモニターできます。



重要

Container-native Virtualization はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。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ついての詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

4.1. CONTAINER-NATIVE VIRTUALIZATION のアップグレードについて

4.1.1. Container-native Virtualization のアップグレードについて

- OpenShift Container Platform Web コンソールを使用して Operator サブスクリプションのチャネルを変更することで、Container-native Virtualization の次のマイナーバージョンにアップグレードできます。
- Container-native Virtualization のインストール時に **z-stream** の自動更新を有効にできます。
- 更新は、OpenShift Container Platform のインストール時にデプロイされる **Marketplace Operator** 経由で送信されます。Marketplace Operator は外部 Operator をクラスターに対して利用可能にします。
- 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。

4.1.2. Container-native Virtualization のクラスターへの作用について

- アップグレードを実行しても仮想マシンのワークロードは中断しません。
 - 仮想マシン Pod は、アップグレード時に再起動したり、移行したりしません。**virt-launcher** Pod を更新する必要がある場合は、仮想マシンの再起動またはライブマイグレーションが必要になります。



注記

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

- アップグレードによってネットワーク接続が中断されることはありません。
- DataVolume およびその関連付けられた PersistentVolumeClaim はアップグレード時に保持されます。

4.2. CONTAINER-NATIVE VIRTUALIZATION の次のマイナーバージョンへのアップグレード

OpenShift Container Platform Web コンソールを使用して Operator サブスクリプションのチャンネルを変更することで、Container-native Virtualization を次のマイナーバージョンに手動でアップグレードできます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてのクラスターへのアクセスがあること。

手順

1. OpenShift Container Platform Web コンソールにアクセスし、**Operators → Installed Operators** に移動します。
2. **OpenShift Virtualization** をクリックし、**Operator Details** ページを開きます。
3. **Subscription** タブをクリックし、**Subscription Overview** ページを開きます。
4. **Channel** ペインで、バージョン番号の右側にある鉛筆アイコンをクリックし、**Change Subscription Update Channel** ウィンドウを開きます。
5. 次のマイナーバージョンを選択します。たとえば、Container-native Virtualization 2.3 にアップグレードする場合には、2.3 を選択します。
6. **Save** をクリックします。
7. **Operators → Installed Operators** に移動してアップグレードのステータスを確認します。以下の **oc** コマンドを実行してステータスを確認することもできます。

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

4.3. アップグレードステータスの監視

Container-native Virtualization アップグレードステータスをモニターする最適な方法として、ClusterServiceVersion (CSV) (CSV) **PHASE** を監視できます。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アクセスできる。
- OpenShift CLI (**oc**) をインストールしている。

手順

1. 以下のコマンドを実行します。

```
$ oc get csv
```

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。

VERSION	REPLACES	PHASE
2.2.1	kubevirt-hyperconverged-operator.v2.2.0	Installing
2.2.0		Replacing

3. オプション: 以下のコマンドを実行して、すべての Container-native Virtualization コンポーネントの状態の集約されたステータスをモニターします。

```
$ oc get hco -n openshift-cnv kubevirt-hyperconverged \
-o=jsonpath='{range .status.conditions[*]}.{type}{"\t"}{.status}{"\t"}{.message}{"\n"}{end}'
```

アップグレードが成功すると、以下の出力が得られます。

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

関連情報

- [ClusterServiceVersion \(CSV\)](#)

第5章 CLI ツールの使用

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

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

5.1. 前提条件

- **virtctl** クライアントをインストール する必要があります。

5.2. VIRTCTL クライアントコマンド

virtctl クライアントは、Container-native Virtualization リソースを管理するためのコマンドラインユーティリティです。以下の表には、Container-native Virtualization のドキュメント全体で使用されている **virtctl** コマンドが記載されています。



注記

コマンドで使用できるオプションの一覧を表示するには、これを **-h** または **--help** フラグを指定して実行します。以下は例になります。

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

表5.1 virtctl クライアントコマンド

コマンド	説明
virtctl start <vm_name>	仮想マシンを起動します。
virtctl stop <vm_name>	仮想マシンを停止します。
virtctl pause vm vmi <object_name>	仮想マシンまたは仮想マシンインスタンスを一時停止します。マシンの状態がメモリーに保持されます。
virtctl unpause vm vmi <object_name>	仮想マシンまたは仮想マシンインスタンスの一時停止を解除します。
virtctl restart <vm_name>	仮想マシンを再起動します。
virtctl expose <vm_name>	仮想マシンまたは仮想マシンインスタンスの指定されたポートを転送するサービスを作成し、このサービスをノードの指定されたポートで公開します。
virtctl console <vmi_name>	仮想マシンインスタンスのシリアルコンソールに接続します。
virtctl vnc <vmi_name>	仮想マシンインスタンスへの VNC 接続を開きます。

コマンド	説明
virtctl image-upload dv <datavolume_name> --image-path= </path/to/image> --no-create	仮想マシンイメージをすでに存在する DataVolume にアップロードします。
virtctl image-upload dv <datavolume_name> --size= <datavolume_size> --image-path= </path/to/image>	仮想マシンイメージを新規 DataVolume にアップロードします。

5.3. OPENSIFT CONTAINER PLATFORM クライアントコマンド

OpenShift Container Platform **oc** クライアントは、仮想マシン (**vm**) および仮想マシンインスタンス (**vmi**) オブジェクトタイプを含む、OpenShift Container Platform リソースを管理するためのコマンドラインユーティリティです。



注記

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

表5.2 **oc** コマンド

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

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

第6章 仮想マシン

6.1. 仮想マシンの作成

以下のいずれかの手順を使用して、仮想マシンを作成します。

- 仮想マシンウィザードの実行
- 仮想マシンウィザードによる事前に設定された YAML ファイルの貼り付け
- CLI の使用
- 仮想マシンウィザードによる VMware 仮想マシンまたはテンプレートのインポート



警告

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

6.1.1. 仮想マシンウィザードの実行による仮想マシンの作成

Web コンソールは、**General**、**Networking**、**Storage**、**Advanced**、および **Review** ステップに移動し、仮想マシンの作成プロセスを単純化するインタラクティブなウィザードを特長としています。すべての必須フィールドには * のマークが付けられます。必要なフィールドが完了したら、仮想マシンを確認し、これを作成することができます。

Network Interface Card (NIC) およびストレージディスクを作成し、それらの作成後に仮想マシンに割り当てることができます。

起動可能なディスク

URL または **Container** のいずれかが **General** ステップで **Source** として選択される場合、**rootdisk** ディスクが作成され、**Bootable Disk** として仮想マシンに割り当てられます。**rootdisk** を変更できますが、これを削除することはできません。

Bootable Disk は、仮想マシンにディスクが割り当てられていない場合、**PXE** ソースからプロビジョニングされる仮想マシンには不要です。1つ以上のディスクが仮想マシンに割り当てられている場合、**Bootable Disk** を1つを選択する必要があります。


前提条件

- ウィザードを使用して仮想マシンを作成する場合、仮想マシンのストレージメディアは Read-Write-Many(RWX)PVC をサポートする必要があります。


手順


1. サイドメニューから **Workloads** → **Virtual Machines** をクリックします。
2. **Create Virtual Machine** をクリックし、**New with Wizard** を選択します。

3. **General** ステップで必要なすべてのフィールドに入力します。**Template** を選択すると、これらのフィールドへの入力が自動的に行われます。
4. **Next** をクリックして **Networking** ステップに進みます。デフォルトで **nic0** NIC が割り当てられます。
 - a. (オプション) **Add Network Interface** をクリックして追加の NIC を作成します。

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

5. **Next** をクリックして **Storage** 画面に進みます。
 - a. (オプション) **Add Disk** をクリックして追加のディスクを作成します。これらのディスクの

削除は、Options メニュー  をクリックし、**Delete** を選択して実行できます。

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

6. **Review and Create** をクリックします。**Results** 画面には、仮想マシンの JSON 設定ファイルが表示されます。

仮想マシンは **Workloads** → **Virtual Machines** に一覧表示されます。

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

6.1.1.1. 仮想マシンウィザードのフィールド

名前	パラメーター	説明
Template		仮想マシンの作成に使用するテンプレート。テンプレートを選択すると、他のフィールドが自動的に入力されます。
ソース	PXE	PXE メニューから仮想マシンをプロビジョニングします。クラスターに PXE 対応の NIC が必要になります。
	URL	HTTP または S3 エンドポイントで利用できるイメージから仮想マシンをプロビジョニングします。

名前	パラメーター	説明
	コンテナ	クラスターからアクセスできるレジストリーの起動可能なオペレーティングシステムコンテナから仮想マシンをプロビジョニングします。例: kubevirt/cirros-registry-disk-demo
	Disk	ディスクから仮想マシンをプロビジョニングします。
Operating System		仮想マシン用に選択される主なオペレーティングシステム。
Flavor	small、medium、large、tiny、Custom	仮想マシンに割り当てられる CPU およびメモリーの量を決定するプリセット。 Flavor に設定される Preset はオペレーティングシステムによって決まります。
Memory		仮想マシンに割り当てられるメモリーのサイズ (GiB 単位)。
CPU		仮想マシンに割り当てられる CPU の量。
Workload Profile	High Performance	高パフォーマンスのワークロードに対して最適化された仮想マシン設定。
	Server	サーバーワークロードの実行に最適化されたプロファイル。
	Desktop	デスクトップで使用するための仮想マシン設定。
名前		この名前には、小文字 (a-z)、数字 (0-9)、およびハイフン (-) を含めることができ、最大 253 文字を使用できます。最初と最後の文字は英数字にする必要があります。この名前には、大文字、スペース、ピリオド (.), または特殊文字を使用できません。
説明		オプションの説明フィールド。

名前	パラメーター	説明
Start virtual machine on creation		これを選択すると、作成時に仮想マシンが自動的に起動します。

6.1.1.2. Cloud-init フィールド

名前	説明
Hostname	仮想マシンの特定のホスト名を設定します。
Authenticated SSH Keys	仮想マシンの <code>~/.ssh/authorized_keys</code> にコピーされるユーザーの公開鍵。
Use custom script	他のオプションを、カスタム cloud-init スクリプトを貼り付けるフィールドに置き換えます。

6.1.1.3. CD-ROM フィールド

ソース	説明
Container	コンテナのパスを指定します。例: kubevirt/fedora-registry-disk: latest
URL	URL パスおよびサイズ (GiB 単位) を指定します。次に、ドロップダウンリストからこの URL のストレージクラスを選択します。
Attach Disk	割り当てる仮想マシンディスクを選択します。

6.1.1.4. ネットワークフィールド

名前	説明
名前	ネットワークインターフェイスの名前。
Model	ネットワークインターフェイスカードのモデルを示します。サポートされる値は、 e1000 、 e1000e 、 ne2k_pci 、 pcnet 、 rtl8139 、および virtIO です。
Network	利用可能な NetworkAttachmentDefinition オブジェクトの一覧。

名前	説明
Type	利用可能なバインディングメソッドの一覧。デフォルトの Pod ネットワークについては、 masquerade が唯一の推奨されるバインディングメソッドになります。セカンダリーネットワークの場合は、 bridge バインディングメソッドを使用します。 masquerade メソッドは、デフォルト以外のネットワークではサポートされません。
MAC Address	ネットワークインターフェイスの MAC アドレス。MAC アドレスが指定されていない場合、セッションの一時アドレスが生成されます。

6.1.1.5. ストレージフィールド

名前	説明
ソース	仮想マシンの空のディスクを選択するか、または URL、Container、Attach Cloned Disk、または Attach Disk などの選択可能なオプションから選択します。既存ディスクを選択し、これを仮想マシンに割り当てるには、利用可能な PersistentVolumeClaim (PVC) の一覧から Attach Cloned Disk または Attach Disk を選択します。
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 (GiB)	ディスクのサイズ (GiB)。
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 StorageClass 。

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

Web コンソールの **Workloads** → **Virtual Machines** 画面で YAML 設定ファイルを作成するか、またはこれを貼り付けて仮想マシンを作成します。YAML 編集画面を開くと、常に有効な **example** 仮想マシン設定がデフォルトで提供されます。

Create をクリックする際に YAML 設定が無効な場合、エラーメッセージでエラーが発生したパラメーターが示唆されます。エラーは一度に 1 つのみ表示されます。

**注記**

編集集中に YAML 画面から離れると、設定に対して加えた変更が取り消されます。

手順

1. サイドメニューから **Workloads** → **Virtual Machines** をクリックします。
2. **Create Virtual Machine** をクリックし、**New from YAML** を選択します。
3. 編集可能なウィンドウで仮想マシンの設定を作成するか、またはこれを貼り付けます。
 - a. または、YAML 画面にデフォルトで提供される **example** 仮想マシンを使用します。
4. (オプション) **Download** をクリックして YAML 設定ファイルをその現在の状態でダウンロードします。
5. **Create** をクリックして仮想マシンを作成します。

仮想マシンは **Workloads** → **Virtual Machines** に一覧表示されます。

6.1.3. CLI の使用による仮想マシンの作成**手順**

VirtualMachine 設定ファイルの **spec** オブジェクトは、コア数やメモリーの量、ディスクタイプおよび使用するボリュームなどの仮想マシン設定を参照します。

1. 関連する PVC **claimName** をボリュームとして参照し、仮想マシンディスクを仮想マシンに割り当てます。
2. OpenShift Container Platform クライアントで仮想マシンを作成するには、以下のコマンドを実行します。

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

3. 仮想マシンは **Stopped** 状態で作成されるため、これを起動して仮想マシンインスタンスを実行します。

**注記**

ReplicaSet は、一定数の同一 Pod の可用性を保証することを目的としています。現時点で、ReplicaSet は Container-native Virtualization でサポートされていません。

表6.1 ドメイン設定

設定	説明
Cores	仮想マシン内のコア数。1以上の値である必要があります。
Memory	ノードによって仮想マシンに割り当てられる RAM の量。 M (メガバイト) または Gi (ギガバイト) で値を指定します。

設定	説明
ディスク	参照されるボリュームの名前。ボリュームの名前に一致する必要があります。

表6.2 ボリューム設定

設定	説明
名前	ボリュームの名前。DNS ラベルであり、仮想マシン内で一意である必要があります。
PersistentVolumeClaim	仮想マシンに割り当てる PVC。PVC の claimName は仮想マシンと同じプロジェクトになければなりません。

6.1.4. 仮想マシンのストレージボリュームタイプ

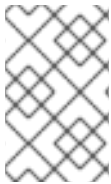
仮想マシンのストレージボリュームタイプがドメインおよびボリューム設定と共に一覧表示されます。

ephemeral	ネットワークボリュームを読み取り専用のバック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。バックングボリュームは 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。バックングボリューム (PVC) はいずれの方法でも変更されません。
persistentVolumeClaim	<p>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。</p> <p>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。</p>
dataVolume	DataVolume は、インポート、クローンまたはアップロード操作で仮想マシンディスクの準備プロセスを管理することによって persistentVolumeClaim ディスクタイプにビルドされます。このボリュームタイプを使用する仮想マシンは、ボリュームが準備できるまで起動しないことが保証されます。
cloudInitNoCloud	参照される cloud-init NoCloud データソースが含まれるディスクを割り当て、ユーザーデータおよびメタデータを仮想マシンに提供します。cloud-init インストールは仮想マシンディスク内で必要になります。

containerDisk	<p>コンテナイメージレジストリーに保存される、仮想マシンディスクなどのイメージを参照します。イメージはレジストリーからプルされ、仮想マシンの作成時にボリュームに組み込まれます。containerDisk ボリュームは一時的なボリュームです。これは、仮想マシンが停止されるか、再起動するか、または削除される際に破棄されます。</p> <p>コンテナディスクは単一の仮想マシンに制限されず、永続ストレージを必要としない多数の仮想マシンのクローンを作成するのに役立ちます。</p> <p>RAW および QCOW2 形式のみがコンテナ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。</p>
emptyDisk	<p>仮想マシンインターフェースのライフサイクルに関連付けられるスパースの QCOW2 ディスクを追加で作成します。データは仮想マシンのゲストによって実行される再起動後も存続しますが、仮想マシンが Web コンソールから停止または再起動する場合には破棄されます。空のディスクは、アプリケーションの依存関係および一時ディスクの一時ファイルシステムの制限を上回るデータを保存するために使用されます。</p> <p>ディスク 容量 サイズも指定する必要があります。</p>

6.1.5. 追加リソース

[KubeVirt v.0.26.5 API リファレンス](#) の **VirtualMachineSpec** 定義は、仮想マシン仕様のパラメーターおよび階層のより範囲の広いコンテキストを提供します。



注記

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

6.2. 仮想マシンの編集

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

6.2.1. Web コンソールでの仮想マシンの編集

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

手順

1. サイドメニューから **Workloads** → **Virtual Machines** をクリックします。
2. 仮想マシンを選択して、**Virtual Machine Overview** 画面を開きます。
3. 鉛筆アイコンをクリックして、該当するフィールドを編集可能にします。
4. 関連する変更を加え、**Save** をクリックします。

仮想マシンが実行中の場合、変更内容は仮想マシンが再起動されるまで反映されません。

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

Web コンソールを使用して、仮想マシンの YAML 設定を編集します。

すべてのパラメーターを更新できる訳ではありません。変更不可の値を編集し、**Save** をクリックすると、更新できなかったパラメーターを示すエラーメッセージが出されます。

YAML 設定は、仮想マシンが **Running** の場合に編集できますが、変更は仮想マシンが停止され、再度起動された後にのみ有効になります。



注記

編集中に YAML 画面から離れると、設定に対して加えた変更が取り消されます。

手順

1. サイドメニューから **Workloads** → **Virtual Machine** をクリックします。
2. 仮想マシンを選択します。
3. **YAML** タブをクリックして編集可能な設定を表示します。
4. オプション: **Download** をクリックして YAML ファイルをその現在の状態でローカルにダウンロードできます。
5. ファイルを編集し、**Save** をクリックします。

オブジェクトの更新されたバージョン番号を含む、変更が正常に行われたことを示す確認メッセージが表示されます。

6.2.3. CLI を使用した仮想マシン YAML 設定の編集

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

前提条件

- YAML オブジェクト設定ファイルを使って仮想マシンを設定していること。
- **oc** CLI をインストールしていること。

手順

1. 以下のコマンドを実行して、仮想マシン設定を更新します。

```
$ oc edit <object_type> <object_ID>
```
2. オブジェクト設定を開きます。
3. YAML を編集します。
4. 実行中の仮想マシンを編集する場合は、以下のいずれかを実行する必要があります。
 - 仮想マシンの再起動
 - 新規の設定を有効にするために、以下のコマンドを実行します。

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

6.2.4. 仮想マシンへの仮想ディスクの追加

以下の手順を使用して仮想ディスクを仮想マシンに追加します。

手順

1. **Virtual Machines** タブで、仮想マシンを選択します。
2. **Disks** タブを選択します。
3. **Add Disks** をクリックして、**Add Disk** ウィンドウを開きます。
4. **Add Disk** ウィンドウで、**Source**、**Name**、**Size**、**Interface**、および **Storage Class** を指定します。
5. ドロップダウンリストおよびチェックボックスを使用して、ディスク設定を編集します。
6. **OK** をクリックします。

6.2.5. 仮想マシンへのネットワークインターフェ이스の追加

以下の手順を使用してネットワークインターフェ이스を仮想マシンに追加します。

手順

1. **Virtual Machines** タブで、仮想マシンを選択します。
2. **Network Interfaces** タブを選択します。
3. **Add Network Interface** をクリックします。
4. **Add Network Interface** ウィンドウで、ネットワークインターフェ이스の **Name**、**Model**、**Network**、**Type**、および **MAC Address** を指定します。
5. **Add** をクリックしてネットワークインターフェ이스を追加します。
6. 仮想マシンを再起動して、アクセスを有効にします。
7. ドロップダウンリストとチェックボックスを編集して、ネットワークインターフェ이스を設定します。
8. **Save Changes** をクリックします。
9. **OK** をクリックします。

新規のネットワークインターフェ이스は、ユーザーが再起動するまで **Create Network Interface** 一覧の上部に表示されます。

新規のネットワークインターフェ이스には、仮想マシンを再起動するまで **Pending VM restart** のリンク状態が表示されます。Link State (リンク状態) にカーソルを合わせて詳細情報を表示します。

ネットワークインターフェイスクードが仮想マシンで定義され、ネットワークに接続されている場合は、**Link State** はデフォルトで **Up** に設定されます。

6.2.6. 仮想マシンの CD-ROM の編集

以下の手順を使用して、仮想マシンの CD-ROM を設定します。

手順

1. **Virtual Machines** タブで、仮想マシンを選択します。
2. **Overview** タブを選択します。
3. CD-ROM 設定を追加または編集するには、**CD-ROMs** ラベルの右側にある鉛筆アイコンをクリックします。**Edit CD-ROM** ウィンドウが開きます。
 - CD-ROM が編集できない場合、以下のメッセージが表示されます: **The virtual machine doesn't have any CD-ROMs attached.**
 - 利用可能な CD-ROM がある場合は、**-** をクリックして CD-ROM を削除できます。
4. **Edit CD-ROM** ウィンドウで、以下を実行します。
 - a. **Media Type** のドロップダウンリストから CD-ROM 設定のタイプを選択します。CD-ROM 設定タイプは **Container**、**URL**、および **Persistent Volume Claim** です。
 - b. それぞれの **Type** に必要な情報を入力します。
 - c. すべての CD-ROM が追加されたら、**Save** をクリックします。

6.3. ブート順序の編集

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

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

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

6.3.1. Web コンソールでのブート順序一覧への項目の追加

Web コンソールを使用して、ブート順序一覧に項目を追加します。

手順

1. サイドメニューから **Workloads** → **Virtual Machines** をクリックします。
2. 仮想マシンを選択して、**Virtual Machine Overview** 画面を開きます。
3. **Boot Order** の右側にある鉛筆アイコンをクリックします。YAML 設定が存在しない場合や、これがブート順序一覧の初回作成時の場合、以下のメッセージが表示されます。 **No resource selected.VM will attempt to boot disks from YAML by order of appearance in YAML file.Please select a boot source.**

4. **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カード (NIC) を選択します。
5. 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。
6. **Save** をクリックします。

6.3.2. Web コンソールでのブート順序一覧の編集

Web コンソールで起動順序一覧を編集します。

手順

1. サイドメニューから **Workloads** → **Virtual Machines** をクリックします。
2. 仮想マシンを選択して、**Virtual Machine Overview** 画面を開きます。
3. **Boot Order** の右側にある鉛筆アイコンをクリックします。
4. ブート順序一覧で項目を移動するのに適した方法を選択します。
 - スクリーンリーダーを使用しない場合、移動する項目の横にある矢印アイコンにカーソルを合わせ、項目を上下にドラッグし、選択した場所にドロップします。
 - スクリーンリーダーを使用する場合は、上矢印キーまたは下矢印を押して、ブート順序一覧で項目を移動します。次に **Tab** キーを押して、選択した場所に項目をドロップします。
5. **Save** をクリックします。

6.3.3. YAML 設定ファイルでのブート順序一覧の編集

CLI を使用して、YAML 設定ファイルのブート順序の一覧を編集します。

手順

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

```
$ oc edit vm example
```

2. YAML ファイルを編集し、ディスクまたはネットワークインターフェイスカード (NIC) に関連付けられたブート順序の値を変更します。以下は例になります。

```
disks:
- bootOrder: 1 ❶
  disk:
    bus: virtio
    name: containerdisk
- disk:
  bus: virtio
  name: cloudinitdisk
- cdrom:
  bus: virtio
  name: cd-drive-1
interfaces:
- boot Order: 2 ❷
```

```

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カードに指定されたブート順序の値。
3. YAML ファイルを保存します。
 4. **reload the content** をクリックして、Web コンソールで YAML ファイルの更新されたブート順序の値をブート順序一覧に適用します。

6.3.4. Web コンソールでのブート順序一覧からの項目の削除

Web コンソールを使用して、ブート順序の一覧から項目を削除します。

手順

1. サイドメニューから **Workloads** → **Virtual Machines** をクリックします。
2. 仮想マシンを選択して、**Virtual Machine Overview** 画面を開きます。
3. **Boot Order** の右側にある鉛筆アイコンをクリックします。
4. 項目の横にある **Remove** アイコンをクリックします。この項目はブート順序の一覧から削除され、利用可能なブートソースの一覧に保存されます。ブート順序一覧からすべての項目を削除する場合、以下のメッセージが表示されます。**No resource selected.VM will attempt to boot disks from YAML by order of appearance in YAML file.Please select a boot source.**

6.4. 仮想マシンの削除

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

6.4.1. Web コンソールの使用による仮想マシンの削除

仮想マシンを削除すると、仮想マシンはクラスターから永続的に削除されます。



注記

仮想マシンを削除する際に、これが使用する DataVolume は自動的に削除されます。

手順

1. Container-native Virtualization コンソールのサイドメニューから **Workloads** → **Virtual Machines** をクリックします。
2. 削除する仮想マシンの **⋮** ボタンをクリックして **Delete Virtual Machine** を選択します。
 - または、仮想マシン名をクリックして **Virtual Machine Details** 画面を開き、**Actions** → **Delete Virtual Machine** をクリックします。
3. 確認のポップアップウィンドウで、**Delete** をクリックし、仮想マシンを永続的に削除します。

6.4.2. CLI の使用による仮想マシンの削除

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



注記

仮想マシンを削除する際に、これが使用する DataVolume は自動的に削除されます。

前提条件

- 削除する仮想マシンの名前を特定すること。

手順

- 以下のコマンドを実行し、仮想マシンを削除します。

```
$ oc delete vm <vm_name>
```

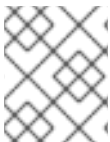


注記

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

6.5. 仮想マシンインスタンスの削除

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。仮想マシンインスタンス (VMI) を手動で削除するには、**oc** コマンドラインインターフェイス (CLI) を使用します。



注記

以下の手順を使用して、Container-native Virtualization をアンインストールする前に、未処理の VMI を確認して削除します。

6.5.1. 仮想マシンインスタンスの一覧表示

oc コマンドラインインターフェイス (CLI) を使用して、クラスターの仮想マシンインスタンスを一覧表示できます。

手順

- 以下のコマンドを実行して、仮想マシンインスタンスの一覧を表示します。

```
$ oc get vmis
```

6.5.2. 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) を使用して仮想マシンインスタンスを削除できます。

前提条件

- 削除する必要のある仮想マシンインスタンスの名前を特定します。

手順

- 以下のコマンドを実行し、仮想マシンインスタンスを削除します。

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



注記

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

6.6. 仮想マシンの状態の制御

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




注記

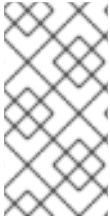
コマンドラインインターフェイス (CLI) から仮想マシンを制御するには、[virtctl クライアント](#) を使用します。

6.6.1. 仮想マシンの起動

Web コンソールから仮想マシンを起動できます。

手順

1. **Workloads** → **Virtual Machines** をクリックします。
2. 起動する仮想マシンが含まれる行を見つけます。
3. ユースケースに適したメニューに移動します。
 - 複数の仮想マシンでのアクションの実行が可能なこのページに留まるには、以下を実行します。
 - a. 行の右端にある Options メニュー  をクリックします。
 - 選択した仮想マシンを起動する前に、その仮想マシンの総合的な情報を表示するには、以下を実行します。
 - a. 仮想マシンの名前をクリックして、**Virtual Machine Details** ページにアクセスします。
 - b. **Actions** をクリックします。
4. **Start Virtual Machine** を選択します。
5. 確認ウィンドウで **Start** をクリックし、仮想マシンを起動します。



注記

URL ソースからプロビジョニングされる仮想マシンの初回起動時に、Container-native Virtualization が URL エンドポイントからコンテナ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。

6.6.2. 仮想マシンの再起動


Web コンソールから実行中の仮想マシンを再起動できます。



重要

エラーを回避するには、ステータスが **Importing** の仮想マシンは再起動しないでください。

手順


1. **Workloads** → **Virtual Machines** をクリックします。
2. 再起動する仮想マシンが含まれる行を見つけます。
3. ユースケースに適したメニューに移動します。
 - 複数の仮想マシンでのアクションの実行が可能なこのページに留まるには、以下を実行します。
 - a. 行の右端にある Options メニュー  をクリックします。
 - 選択した仮想マシンを再起動する前に、その仮想マシンの総合的な情報を表示するには、以下を実行します。
 - a. 仮想マシンの名前をクリックして、**Virtual Machine Details** ページにアクセスします。
 - b. **Actions** をクリックします。
4. **Restart Virtual Machine** を選択します。
5. 確認ウィンドウで **Restart** をクリックし、仮想マシンを再起動します。

6.6.3. 仮想マシンの停止

Web コンソールから仮想マシンを停止できます。

手順

1. **Workloads** → **Virtual Machines** をクリックします。
2. 停止する仮想マシンが含まれる行を見つけます。
3. ユースケースに適したメニューに移動します。
 - 複数の仮想マシンでのアクションの実行が可能なこのページに留まるには、以下を実行します。

- a. 行の右端にある Options メニュー  をクリックします。
- 選択した仮想マシンを停止する前に、その仮想マシンの総合的な情報を表示するには、以下を実行します。
 - a. 仮想マシンの名前をクリックして、**Virtual Machine Details** ページにアクセスします。
 - b. **Actions** をクリックします。
4. **Stop Virtual Machine** を選択します。
5. 確認ウィンドウで **Stop** をクリックし、仮想マシンを停止します。

6.6.4. 仮想マシンの一時停止の解除

Web コンソールから仮想マシンの一時停止を解除できます。

前提条件

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



注記

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

手順

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

6.7. 仮想マシンコンソールへのアクセス

Container-native Virtualization は、異なる製品タスクを実現するために使用できる異なる仮想マシンコンソールを提供します。Web コンソールおよび CLI コマンドを使用してこれらのコンソールにアクセスできます。

6.7.1. 仮想マシンコンソールのセッション

Web コンソールの **Virtual Machine Details** 画面の **Consoles** タブから、実行中の仮想マシンの VNC およびシリアルコンソールに接続することができます。

グラフィカルな **VNC コンソール** と **シリアルコンソール** の2つのコンソールを使用できます。**VNC コンソール** は、**Consoles** タブに移動する際には常にデフォルトで開きます。**VNC ConsoleSerial Console** リストを使用してコンソールを切り換えることができます。

コンソールのセッションは切断しない限り、バックグラウンドでアクティブな状態のままになります。**Disconnect before switching** チェックボックスがアクティブな場合にコンソールを切り替えると、現在のコンソールセッションは切断され、選択したコンソールの新規セッションが仮想マシンに接続されます。これにより、一度に1つのコンソールセッションのみが開かれます。

VNC コンソールのオプション

Send Key ボタンでは、仮想マシンに送信するキーの組み合わせを一覧表示します。

シリアルコンソールのオプション

Disconnect ボタンを使用して、仮想マシンから **Serial Console** セッションを手動で切断します。**Reconnect** ボタンを使用して **Serial Console** セッションを仮想マシンに対して手動で開きます。

6.7.2. Web コンソールの使用による仮想マシンへの接続

6.7.2.1. ターミナルへの接続

Web コンソールを使用して仮想マシンに接続することができます。

手順

1. 正しいプロジェクトを指定していることを確認します。そうでない場合は、**Project** 一覧をクリックして適切なプロジェクトを選択します。
2. **Workloads** → **Virtual Machines** をクリックして、プロジェクトに仮想マシンを表示します。
3. 仮想マシンを選択します。
4. **Overview** タブで、**virt-launcher-`<vm-name>`** Pod をクリックします。
5. **Terminal** タブをクリックします。ターミナルが空白の場合、ターミナルをクリックし、任意のキーを押して接続を開始します。

6.7.2.2. シリアルコンソールへの接続

Web コンソールの **Virtual Machine Details** 画面の **Consoles** タブから、実行中の仮想マシンの **Serial Console** に接続します。

手順

1. Container-native Virtualization コンソールで **Workloads** → **Virtual Machines** をクリックします。
2. 仮想マシンを選択します。
3. **Consoles** をクリックします。VNC コンソールがデフォルトで開きます。
4. **VNC Console** ドロップダウンリストをクリックし、**Serial Console** を選択します。

6.7.2.3. VNC コンソールへの接続

Web コンソールの **Virtual Machine Details** 画面の **Consoles** タブから、実行中の仮想マシンの VNC コンソールに接続します。

手順

1. Container-native Virtualization コンソールで **Workloads** → **Virtual Machines** をクリックします。
2. 仮想マシンを選択します。
3. **Consoles** をクリックします。VNC コンソールがデフォルトで開きます。

6.7.2.4. RDP コンソールへの接続

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

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

前提条件

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

手順

1. Container-native Virtualization コンソールで **Workloads** → **Virtual Machines** をクリックします。
2. Windows 仮想マシンを選択します。
3. **Consoles** タブをクリックします。
4. **Consoles** 一覧をクリックし、**Desktop Viewer** を選択します。
5. **Network Interface** 一覧で、layer-2 NIC を選択します。
6. **Launch Remote Desktop** をクリックし、**console.rdp** ファイルをダウンロードします。
7. RDP クライアントを開き、**console.rdp** ファイルを参照します。たとえば、**remmina** を使用します。

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

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

6.7.3. CLI コマンドの使用による仮想マシンコンソールへのアクセス

6.7.3.1. SSH 経由での仮想マシンインスタンスへのアクセス

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

virtctl expose コマンドは、仮想マシンインスタンスのポートをノードポートに転送し、有効にされたアクセスのサービスを作成します。以下の例では、**fedora-vm-ssh** サービスを作成します。このサービスは、**<fedora-vm>** 仮想マシンのポート 22 をノード上のポートに転送します。

前提条件

- 仮想マシンインスタンスと同じプロジェクトにいる必要があります。
- アクセスする仮想マシンインスタンスは、**masquerade** バインディングメソッド方法を使用してデフォルトの Pod ネットワークに接続されている。
- アクセスする仮想マシンインスタンスが実行中であること。
- OpenShift CLI (**oc**) をインストールします。

手順

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

```
$ virtctl expose vm <fedora-vm> --port=20022 --target-port=22 --name=fedora-vm-ssh --type=NodePort 1
```

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

2. サービスをチェックし、サービスが取得したポートを見つけます。

```
$ oc get svc
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP  PORT(S)          AGE
fedora-vm-ssh NodePort    127.0.0.1    <none>       20022:32551/TCP  6s
```

この例では、サービスは **32551** ポートを取得しています。

3. SSH 経由で仮想マシンインスタンスにログインします。ノードの **ipAddress** および直前の手順で確認したポートを使用します。

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

6.7.3.2. 仮想マシンインスタンスのシリアルコンソールへのアクセス

virtctl console コマンドは、指定された仮想マシンインスタンスへのシリアルコンソールを開きます。

前提条件

- **virt-viewer** パッケージがインストールされていること。
- アクセスする仮想マシンインスタンスが実行中であること。

手順

- **virtctl** でシリアルコンソールに接続します。

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

6.7.3.3. VNC を使用した仮想マシンインスタンスのグラフィカルコンソールへのアクセス

virtctl クライアントユーティリティーは **remote-viewer** 機能を使用し、実行中の仮想マシンインスタンスに対してグラフィカルコンソールを開くことができます。この機能は **virt-viewer** パッケージに組み込まれています。

前提条件

- **virt-viewer** パッケージがインストールされていること。
- アクセスする仮想マシンインスタンスが実行中であること。



注記

リモートマシンで SSH 経由で **virtctl** を使用する場合、X セッションをマシンに転送する必要があります。

手順

1. **virtctl** ユーティリティーを使用してグラフィカルインターフェイスに接続します。

```
$ virtctl vnc <VMI>
```

2. コマンドが失敗した場合には、トラブルシューティング情報を収集するために **-v** フラグの使用を試行します。

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

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

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

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

前提条件

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

手順

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

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

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

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

```
...
spec:
  networks:
    - name: default
      pod: {}
    - multus:
        networkName: cnv-bridge
        name: bridge-net
  ...
status:
  interfaces:
    - interfaceName: eth0
      ipAddress: 198.51.100.0/24
      ipAddresses:
        198.51.100.0/24
      mac: a0:36:9f:0f:b1:70
      name: default
    - interfaceName: eth1
      ipAddress: 192.0.2.0/24
      ipAddresses:
        192.0.2.0/24
        2001:db8::/32
      mac: 00:17:a4:77:77:25
      name: bridge-net
  ...
```

3. レイヤー 2 ネットワークインターフェイスの IP アドレスを特定し、これをコピーします。これは直前の例では **192.0.2.0** であり、IPv6 を選択する場合は **2001:db8::** になります。
4. RDP クライアントを開き、接続用に直前の手順でコピーした IP アドレスを使用します。
5. **Administrator** ユーザー名およびパスワードを入力して、Windows 仮想マシンに接続します。

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

6.8.1. VirtIO ドライバーについて

VirtIO ドライバーは、Microsoft Windows 仮想マシンが Container-native Virtualization で実行されるために必要な準仮想化デバイスドライバーです。サポートされるドライバーは、[Red Hat Container Catalog](#) の **container-native-virtualization/virtio-win** コンテナディスクで利用できます。

container-native-virtualization/virtio-win コンテナディスクは、ドライバーのインストールを有効にするために SATA CD ドライブとして仮想マシンに割り当てられる必要があります。仮想マシン上での Windows のインストール時に VirtIO ドライバーをインストールすることも、既存の Windows インストールに追加することもできます。

ドライバーのインストール後に、**container-native-virtualization/virtio-win** コンテナディスクは仮想マシンから削除できます。

[Installing Virtio drivers on a new Windows virtual machine](#) も参照してください。

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

表6.3 サポートされるドライバー

ドライバー名	ハードウェア ID	説明
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	ブロックドライバー。 Other devices グループの SCSI Controller として表示される場合があります。
viorng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	エントロピーソースドライバー。 Other devices グループの PCI Device として表示される場合があります。
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	ネットワークドライバー。 Other devices グループの Ethernet Controller として表示される場合があります。VirtIO NIC が設定されている場合にのみ利用できます。

6.8.3. VirtIO ドライバーコンテナディスクの仮想マシンへの追加

Container-native Virtualization は、[Red Hat Container Catalog](#) で利用できる Microsoft Windows の VirtIO ドライバーをコンテナディスクとして配布します。これらのドライバーを Windows 仮想マシンにインストールするには、仮想マシン設定ファイルで **container-native-virtualization/virtio-win** コンテナディスクを SATA CD ドライブとして仮想マシンに割り当てます。

前提条件

- **container-native-virtualization/virtio-win** コンテナディスクを [Red Hat Container Catalog](#) からダウンロードすること。コンテナディスクがクラスターにない場合は Red Hat レジストリーからダウンロードされるため、これは必須ではありません。

手順

1. **container-native-virtualization/virtio-win** コンテナディスクを **cdrom** ディスクとして Windows 仮想マシン設定ファイルに追加します。コンテナディスクは、クラスターにない場合はレジストリーからダウンロードされます。

```
spec:
  domain:
    devices:
      disks:
        - name: virtiocontainerdisk
          bootOrder: 2 1
```

```

cdrom:
  bus: sata
volumes:
- containerDisk:
    image: container-native-virtualization/virtio-win
    name: virtiocontainerdisk

```

- 1 Container-native Virtualization は、**VirtualMachine** 設定ファイルに定義される順序で仮想マシンディスクを起動します。**container-native-virtualization/virtio-win** コンテナードiskの前に仮想マシンの他のディスクを定義するか、またはオプションの **bootOrder** パラメーターを使用して仮想マシンが正しいディスクから起動するようにできます。ディスクに **bootOrder** を指定する場合、これは設定のすべてのディスクに指定される必要があります。

2. ディスクは、仮想マシンが起動すると利用可能になります。

- コンテナードiskを実行中の仮想マシンに追加する場合、変更を有効にするために CLI で **oc apply -f <vm.yaml>** を使用するか、または仮想マシンを再起動します。
- 仮想マシンが実行されていない場合、**virtctl start <vm>** を使用します。

仮想マシンが起動したら、VirtIO ドライバーを割り当てられた SATA CD ドライブからインストールできます。

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

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



注記

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

手順

1. 仮想マシンを起動し、グラフィカルコンソールに接続します。
2. Windows ユーザーセッションにログインします。
3. **Device Manager** を開き、**Other devices** を拡張して、**Unknown device** を一覧表示します。
 - a. **Device Properties** を開いて、不明なデバイスを特定します。デバイスを右クリックし、**Properties** を選択します。
 - b. **Details** タブをクリックし、**Property** リストで **Hardware Ids** を選択します。
 - c. **Hardware Ids** の **Value** をサポートされる VirtIO ドライバーと比較します。
4. デバイスを右クリックし、**Update Driver Software** を選択します。

5. **Browse my computer for driver software**をクリックし、VirtIO ドライバーが置かれている割り当て済みの SATA CD ドライブの場所に移動します。ドライバーは、ドライバーのタイプ、オペレーティングシステム、および CPU アーキテクチャー別に階層的に編成されます。
6. **Next** をクリックしてドライバーをインストールします。
7. 必要なすべての VirtIO ドライバーに対してこのプロセスを繰り返します。
8. ドライバーのインストール後に、**Close** をクリックしてウィンドウを閉じます。
9. 仮想マシンを再起動してドライバーのインストールを完了します。

6.8.5. 仮想マシンからの VirtIO コンテナディスクの削除

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

手順

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

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

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

2. 変更を有効にするために仮想マシンを再起動します。

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

6.9.1. 前提条件

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

6.9.2. VirtIO ドライバーについて

VirtIO ドライバーは、Microsoft Windows 仮想マシンが Container-native Virtualization で実行されるために必要な準仮想化デバイスドライバーです。サポートされるドライバーは、[Red Hat Container Catalog](#) の **container-native-virtualization/virtio-win** コンテナディスクで利用できます。

container-native-virtualization/virtio-win コンテナディスクは、ドライバーのインストールを有効に

するために SATA CD ドライブとして仮想マシンに割り当てられる必要があります。仮想マシン上での Windows のインストール時に VirtIO ドライバーをインストールすることも、既存の Windows インストールに追加することもできます。

ドライバーのインストール後に、**container-native-virtualization/virtio-win** コンテナディスクは仮想マシンから削除できます。

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

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

表6.4 サポートされるドライバー

ドライバー名	ハードウェア ID	説明
viostor	VEN_1AF4&DEV_1001 VEN_1AF4&DEV_1042	ブロックドライバー。Other devices グループの SCSI Controller として表示される場合があります。
viorng	VEN_1AF4&DEV_1005 VEN_1AF4&DEV_1044	エントロピーソースドライバー。Other devices グループの PCI Device として表示される場合があります。
NetKVM	VEN_1AF4&DEV_1000 VEN_1AF4&DEV_1041	ネットワークドライバー。Other devices グループの Ethernet Controller として表示される場合があります。VirtIO NIC が設定されている場合にのみ利用できます。

6.9.4. VirtIO ドライバーコンテナディスクの仮想マシンへの追加

Container-native Virtualization は、[Red Hat Container Catalog](#) で利用できる Microsoft Windows の VirtIO ドライバーをコンテナディスクとして配布します。これらのドライバーを Windows 仮想マシンにインストールするには、仮想マシン設定ファイルで **container-native-virtualization/virtio-win** コンテナディスクを SATA CD ドライブとして仮想マシンに割り当てます。

前提条件

- **container-native-virtualization/virtio-win** コンテナディスクを [Red Hat Container Catalog](#) からダウンロードすること。コンテナディスクがクラスターにない場合は Red Hat レジストリーからダウンロードされるため、これは必須ではありません。

手順

1. **container-native-virtualization/virtio-win** コンテナディスクを **cdrom** ディスクとして Windows 仮想マシン設定ファイルに追加します。コンテナディスクは、クラスターにない場合はレジストリーからダウンロードされます。

```
spec:
  domain:
```

```

devices:
  disks:
    - name: virtiocontainerdisk
      bootOrder: 2 1
      cdrom:
        bus: sata
  volumes:
    - containerDisk:
        image: container-native-virtualization/virtio-win
        name: virtiocontainerdisk

```

- 1** Container-native Virtualization は、**VirtualMachine** 設定ファイルに定義される順序で仮想マシンディスクを起動します。**container-native-virtualization/virtio-win** コンテナディスクの前に仮想マシンの他のディスクを定義するか、またはオプションの **bootOrder** パラメーターを使用して仮想マシンが正しいディスクから起動するようにできます。ディスクに **bootOrder** を指定する場合、これは設定のすべてのディスクに指定される必要があります。

2. ディスクは、仮想マシンが起動すると利用可能になります。

- コンテナディスクを実行中の仮想マシンに追加する場合、変更を有効にするために CLI で **oc apply -f <vm.yaml>** を使用するか、または仮想マシンを再起動します。
- 仮想マシンが実行されていない場合、**virtctl start <vm>** を使用します。

仮想マシンが起動したら、VirtIO ドライバーを割り当てられた SATA CD ドライブからインストールできます。

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

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



注記

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

手順

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

7. Windows インストールを完了します。

6.9.6. 仮想マシンからの VirtIO コンテナードディスクの削除

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

手順

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

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

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

2. 変更を有効にするために仮想マシンを再起動します。

6.10. 高度な仮想マシン管理

6.10.1. 管理タスクの自動化

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

6.10.1.1. Red Hat Ansible Automation について

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

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

6.10.1.2. 仮想マシン作成の自動化

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

前提条件

- [Red Hat Ansible Engine](#) バージョン 2.8 以降。

手順

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

```
kubevirt_vm:
  namespace:
  name:
  cpu_cores:
  memory:
  disks:
    - name:
      volume:
        containerDisk:
          image:
        disk:
          bus:
```



注記

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

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

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

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

```
kubevirt_vm:
  namespace: default
  name: vm1
  state: running ❶
  cpu_cores: 1
```



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

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

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

5. 出力を確認し、プレイが正常に実行されたかどうかを確認します。

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

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

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

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

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

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

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

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

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

追加情報

- [Playbook の概要](#)
- [Playbook の検証ツール](#)

6.10.2. 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 Container-native 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。

6.10.2.1. 前提条件

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

6.10.2.2. Container-native Virtualization ネットワークの用語集

Container-native Virtualization は、カスタムリソースおよびプラグインを使用して高度なネットワーク機能を提供します。

以下の用語は、Container-native 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。Container-native 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、Customer 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。

NetworkAttachmentDefinition

Pod、仮想マシン、および仮想マシンインスタンスを1つ以上のネットワークに割り当てることを可能にする Multus プロジェクトによって導入される CRD。

PXE (Preboot eXecution Environment)

管理者がネットワーク経由でサーバーからクライアントマシンを起動できるようにするインターフェイス。ネットワークのブートにより、オペレーティングシステムおよび他のソフトウェアをクライアントにリモートでロードできます。

6.10.2.3. 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 NetworkAttachmentDefinition を参照して仮想マシンインスタンスを起動します。また PXE サーバーで必要な場合には、仮想マシンインスタンスの設定ファイルで MAC アドレスを指定することもできます。

前提条件

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

手順

1. クラスターに PXE ネットワークを設定します。
 - a. PXE ネットワーク **pxe-net-conf** の NetworkAttachmentDefinition 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"
      },
      {
        "type": "cnv-tuning" ❶
      }
    ]
  }'
```

❶ **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 NetworkAttachmentDefinition オブジェクトを作成します。

```
$ 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 サーバーが必要な場合には、ネットワークおよび MAC アドレスを指定します。MAC アドレスが指定されていない場合、値は自動的に割り当てられます。ただし、この時点で自動的に割り当てられる MAC アドレスは永続しないことに注意してください。

bootOrder が **1** に設定されており、インターフェ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 NetworkAttachmentDefinition 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 NetworkAttachmentDefinition 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
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
```

9. 仮想マシンのインターフェイスおよび MAC アドレスを確認し、ブリッジに接続されたインターフェイスに MAC アドレスが指定されていることを確認します。この場合、PXE ブートには IP アドレスなしに **eth1** を使用しています。他のインターフェイス **eth0** は OpenShift Container Platform から IP アドレスを取得しています。

```
$ ip addr
```

```
...
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
```

6.10.2.4. テンプレート: PXE ブートの仮想マシンインスタンス設定ファイル

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:
  creationTimestamp: null
  labels:
    special: vmi-pxe-boot
  name: vmi-pxe-boot
spec:
  domain:
    devices:
      disks:
        - disk:
            bus: virtio
            name: containerdisk
            bootOrder: 2
        - disk:
            bus: virtio
            name: cloudinitdisk
      interfaces:
        - masquerade: {}
          name: default
        - bridge: {}
          name: pxe-net
          macAddress: de:00:00:00:00:de
          bootOrder: 1
      machine:
        type: ""
      resources:
        requests:
          memory: 1024M
      networks:
        - name: default
          pod: {}
        - multus:
            networkName: pxe-net-conf
            name: pxe-net
      terminationGracePeriodSeconds: 0
    volumes:
      - name: containerdisk
        containerDisk:
          image: kubevirt/fedora-cloud-container-disk-demo
      - cloudInitNoCloud:
          userData: |
            #!/bin/bash
            echo "fedora" | passwd fedora --stdin
          name: cloudinitdisk
      status: {}
```

6.10.3. ゲストメモリーの管理

ゲストメモリー設定を特定のユースケースに合わせて調整する必要がある場合、ゲストの YAML 設定ファイルを編集してこれを実行できます。Container-native Virtualization は、ゲストメモリーのオーバーコミットの設定と、ゲストメモリーのオーバーコミットアカウンティングの無効化を許可します。

この手順にはどちらの場合もリスクが伴います。そのため、経験のある管理者のみが対応するようにしてください。

6.10.3.1. ゲストメモリーのオーバーコミットの設定

仮想ワークロードに利用可能な量を上回るメモリーが必要な場合、メモリーのオーバーコミットを使用してホストのメモリーのすべてまたはそのほとんどを仮想マシンインスタンスに割り当てることができます。メモリーのオーバーコミットを有効にすることは、通常ホストに予約されるリソースを最大化できることを意味します。

たとえば、ホストに 32 GB RAM がある場合、メモリーのオーバーコミットを使用してそれぞれ 4 GB RAM を持つ 8 つの仮想マシンに対応できます。これは、仮想マシンがそれらのメモリーのすべてを同時に使用しないという前提で機能します。

手順

1. 仮想マシンインスタンスに対し、クラスターから要求された以上のメモリーが利用可能であることを明示的に示すために、仮想マシン設定ファイルを編集し、**spec.domain.memory.guest** を **spec.domain.resources.requests.memory** よりも高い値に設定します。このプロセスはメモリーのオーバーコミットと呼ばれています。

以下の例では、**1024M** がクラスターから要求されますが、仮想マシンインスタンスには **2048M** が利用可能であると通知されます。ノードに利用可能な空のメモリーが十分にある限り、仮想マシンインスタンスは最大 2048M を消費します。

```
kind: VirtualMachine
spec:
  template:
    domain:
      resources:
        requests:
          memory: 1024M
      memory:
        guest: 2048M
```



注記

ノードがメモリー不足の状態になると、Pod のエビクシヨンルールと同じルールが仮想マシンインスタンスに適用されます。

2. 仮想マシンを作成します。

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

6.10.3.2. ゲストメモリーオーバーヘッドアカウンティングの無効化

**警告**

この手順は、特定のユースケースでのみ有効であり、上級ユーザーのみが試行するようにしてください。

要求する量に加えて、少量のメモリーが各仮想マシンインスタンスによって要求されます。追加のメモリーは、それぞれの **VirtualMachineInstance** プロセスをラップするインフラストラクチャーに使用されます。

通常は推奨される方法ではありませんが、ゲストメモリーオーバーヘッドアカウンティングを無効にすることでノード上の仮想マシンインスタンスの密度を増やすことは可能です。

手順

1. ゲストメモリーオーバーヘッドアカウンティングを無効にするには、YAML 設定ファイルを編集し、**overcommitGuestOverhead** の値を **true** に設定します。このパラメーターはデフォルトで無効にされています。

```
kind: VirtualMachine
spec:
  template:
    domain:
      resources:
        overcommitGuestOverhead: true
        requests:
          memory: 1024M
```

**注記**

overcommitGuestOverhead が有効にされている場合、これはゲストのオーバーヘッドをメモリー制限 (ある場合) に追加します。

2. 仮想マシンを作成します。

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

6.10.4. 仮想マシン用の専用リソースの有効化

パフォーマンスを向上させるために、仮想マシンには CPU などのノードの専用リソースを持たせることができます。

6.10.4.1. 専用リソースについて

仮想マシンの専用リソースを有効にする場合、仮想マシンのワークロードは他のプロセスで使用されない CPU でスケジュールされます。専用リソースを使用することで、仮想マシンのパフォーマンスとレイテンシーの予測の精度を向上させることができます。

6.10.4.2. 前提条件

- **CPU マネージャー** はノードに設定される必要があります。仮想マシンのワークロードをスケジューリングする前に、ノードに **cpumanager = true** ラベルが設定されていることを確認します。

6.10.4.3. 仮想マシンの専用リソースの有効化

Web コンソールの **Virtual Machine Overview** ページで仮想マシンの専用リソースを有効にすることができます。

手順

1. サイドメニューから **Workloads** → **Virtual Machines** をクリックします。
2. 仮想マシンを選択して、**Virtual Machine Overview** ページを開きます。
3. **Details** タブをクリックします。
4. **Dedicated Resources** フィールドの右側にある鉛筆アイコンをクリックして、**Dedicated Resources** ウィンドウを開きます。
5. **Schedule this workload with dedicated resources (guaranteed policy)** を選択します。
6. **Save** をクリックします。

6.11. 仮想マシンのインポート

6.11.1. DataVolume インポートの TLS 証明書

6.11.1.1. DataVolume インポートの認証に使用する TLS 証明書の追加

ソースからデータをインポートするには、レジストリーまたは HTTPS エンドポイントの TLS 証明書を ConfigMap に追加する必要があります。この ConfigMap は、宛先 DataVolume の namespace に存在する必要があります。

TLS 証明書の相対パスを参照して ConfigMap を作成します。

手順

1. 正しい namespace にあることを確認します。ConfigMap は、同じ namespace にある場合に DataVolume によってのみ参照されます。

```
$ oc get ns
```

2. ConfigMap を作成します。

```
$ oc create configmap <configmap-name> --from-file=</path/to/file/ca.pem>
```

6.11.1.2. 例: TLS 証明書から作成される ConfigMap

以下は、**ca.pem** TLS 証明書で作成される ConfigMap の例です。

```
apiVersion: v1
kind: ConfigMap
metadata:
```

```
name: tls-certs
data:
  ca.pem: |
    -----BEGIN CERTIFICATE-----
    ... <base64 encoded cert> ...
    -----END CERTIFICATE-----
```

6.11.2. DataVolume の使用による仮想マシンイメージのインポート

既存の仮想マシンイメージは OpenShift Container Platform クラスタにインポートできます。Container-native Virtualization は DataVolume を使用してデータのインポートおよび基礎となる PersistentVolumeClaim (PVC) の作成を自動化します。



重要

ディスクイメージを PVC にインポートする際に、ディスクイメージは PVC で要求されるストレージの全容量を使用するように拡張されます。この領域を使用するには、仮想マシンのディスクパーティションおよびファイルシステムの拡張が必要になる場合があります。

サイズ変更の手順は、仮想マシンにインストールされるオペレーティングシステムによって異なります。詳細は、該当するオペレーティングシステムのドキュメントを参照してください。

6.11.2.1. 前提条件

- エンドポイントに TLS 証明書が必要な場合、証明書は DataVolume と同じ namespace の [ConfigMap に組み込む](#) 必要があり、これは DataVolume 設定で参照されます。
- この操作を正常に実行するためには、[StorageClass を定義するか、CDI のスクラッチ領域を用意](#) する必要がある場合があります。

6.11.2.2. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubvirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	<input type="checkbox"/> TAR	<input type="checkbox"/> TAR

✓ 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

+ アーカイブはブロックモード DV をサポートしません。

6.11.2.3. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.11.2.4. DataVolume のあるオブジェクトへの仮想マシンイメージのインポート

インポートされたイメージから仮想マシンを作成するには、仮想マシンを作成する前にイメージの場所を **VirtualMachine** 設定ファイルに指定します。

前提条件

- OpenShift CLI (**oc**) のインストール。
- RAW、ISO、または QCOW2 形式の仮想マシンディスクイメージ (オプションで **xz** または **gz** を使用して圧縮される)
- イメージがデータソースにアクセスするために必要な認証情報と共にホストされる **HTTP** エンドポイント
- 1つ以上の利用可能な PersistentVolume

手順

1. インポートする必要がある仮想ディスクイメージをホストする **HTTP** ファイルサーバーを特定します。正しい形式での完全な URL が必要になります。
 - <http://www.example.com/path/to/data>
2. データソースに認証情報が必要な場合、**endpoint-secret.yaml** ファイルを編集し、更新された設定をクラスターに適用します。

```
apiVersion: v1
kind: Secret
metadata:
  name: <endpoint-secret>
  labels:
    app: containerized-data-importer
type: Opaque
data:
  accessKeyId: "" ❶
  secretKey: "" ❷
```

- ❶ オプション: キーまたはユーザー名 (base64 エンコード)

2 オプション: シークレットまたはパスワード、base64 エンコード

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

3. 仮想マシン設定ファイルを編集し、インポートする必要があるイメージのデータソースを指定します。この例では、Fedora イメージがインポートされます。

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  creationTimestamp: null
  labels:
    kubevirt.io/vm: vm-fedora-datavolume
  name: vm-fedora-datavolume
spec:
  dataVolumeTemplates:
    - metadata:
        creationTimestamp: null
        name: fedora-dv
      spec:
        pvc:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 2Gi
          storageClassName: local
        source:
          http:
            url:
              https://download.fedoraproject.org/pub/fedora/linux/releases/28/Cloud/x86_64/images/Fedora
              -Cloud-Base-28-1.1.x86_64.qcow2 1
            secretRef: "" 2
            certConfigMap: "" 3
        status: {}
      running: false
    template:
      metadata:
        creationTimestamp: null
        labels:
          kubevirt.io/vm: vm-fedora-datavolume
      spec:
        domain:
          devices:
            disks:
              - disk:
                  bus: virtio
                  name: datavolumedisk1
        machine:
          type: ""
        resources:
          requests:
            memory: 64M
        terminationGracePeriodSeconds: 0
```

```
volumes:
- dataVolume:
  name: fedora-dv
  name: datavolumedisk1
status: {}
```

- 1 インポートする必要があるイメージの **HTTP** ソース。
- 2 **secretRef** パラメーターはオプションです。
- 3 **certConfigMap** は、自己署名証明書またはシステム CA バンドルで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ために必要です。参照される ConfigMap は DataVolume と同じ namespace にある必要があります。

4. 仮想マシンを作成します。

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



注記

oc create コマンドは、DataVolume および仮想マシンを作成します。CDI コントローラーは適切なアノテーションを使って基礎となる 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、DataVolume のステータスは **Succeeded** に変更され、仮想マシンの起動が可能になります。

DataVolume のプロビジョニングはバックグラウンドで実行されるため、これをモニターする必要はありません。仮想マシンは起動できますが、これはインポートが完了するまで実行されません。

オプションの検証手順

1. **oc get pods** を実行し、インポーター Pod を見つけます。この Pod は指定された URL からイメージをダウンロードし、これをプロビジョニングされた PV に保存します。
2. **Succeeded** が表示されるまで DataVolume のステータスをモニターします。

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

- 1 仮想マシン設定ファイルに指定された DataVolume のデータラベル。

3. プロビジョニングが完了し、VMI が起動したことを検証するには、そのシリアルコンソールへのアクセスを試行します。

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

6.11.2.5. テンプレート: DataVolume 仮想マシン設定ファイル

example-dv-vm.yaml

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
```

```

labels:
  kubevirt.io/vm: example-vm
name: example-vm
spec:
  dataVolumeTemplates:
  - metadata:
      name: example-dv
    spec:
      pvc:
        accessModes:
        - ReadWriteOnce
        resources:
          requests:
            storage: 1G
        source:
          http:
            url: "" ❶
    running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: example-vm
    spec:
      domain:
        cpu:
          cores: 1
        devices:
          disks:
            - disk:
                bus: virtio
                name: example-dv-disk
        machine:
          type: q35
        resources:
          requests:
            memory: 1G
        terminationGracePeriodSeconds: 0
      volumes:
      - dataVolume:
          name: example-dv
          name: example-dv-disk

```

❶ インポートする必要があるイメージの **HTTP** ソース (該当する場合)。

6.11.2.6. テンプレート: DataVolume インポート設定ファイル

example-import-dv.yaml

```

apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: "example-import-dv"
spec:
  source:

```

```

http:
  url: "" ❶
  secretRef: "" ❷
pvc:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: "1G"

```

- ❶ インポートする必要があるイメージの **HTTP** ソース。
- ❷ **secretRef** パラメーターはオプションです。

6.11.3. DataVolume の使用による仮想マシンイメージのブロックストレージへのインポート

既存の仮想マシンイメージは OpenShift Container Platform クラスターにインポートできます。Container-native Virtualization は DataVolume を使用してデータのインポートおよび基礎となる PersistentVolumeClaim (PVC) の作成を自動化します。



重要

ディスクイメージを PVC にインポートする際に、ディスクイメージは PVC で要求されるストレージの全容量を使用するように拡張されます。この領域を使用するには、仮想マシンのディスクパーティションおよびファイルシステムの拡張が必要になる場合があります。

サイズ変更の手順は、仮想マシンにインストールされるオペレーティングシステムによって異なります。詳細は、該当するオペレーティングシステムのドキュメントを参照してください。

6.11.3.1. 前提条件

- [CDI でサポートされる操作マトリックス](#) に応じてスクラッチ領域が必要な場合、まずは、この操作が正常に実行されるように [StorageClass](#) を定義するか、または [CDI スクラッチ領域を用意](#) します。

6.11.3.2. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.11.3.3. ブロック PersistentVolume について

ブロック PersistentVolume (PV) は、raw ブロックデバイスによってサポートされる PV です。これらのボリュームにはファイルシステムがなく、ディスクに直接書き込む仮想マシンや、独自のストレージサービスを実装する仮想マシンにはパフォーマンス上の利点があります。

raw ブロックボリュームは、PV および PersistentVolumeClaim (PVC) 仕様で **volumeMode: Block** を指定してプロビジョニングされます。

6.11.3.4. ローカルブロック PersistentVolume の作成

ファイルにデータを設定し、これをループデバイスとしてマウントすることにより、ノードでローカルブロック PersistentVolume (PV) を作成します。次に、このループデバイスを PV 設定で **Block** ボリュームとして参照し、これを仮想マシンイメージのブロックデバイスとして使用できます。

手順

1. ローカル PV を作成するノードに **root** としてログインします。この手順では、**node01** を例に使用します。
2. ファイルを作成して、これを null 文字で設定し、ブロックデバイスとして使用できるようにします。以下の例では、2Gb (20 100Mb ブロック) のサイズのファイル **loop10** を作成します。

```
$ dd if=/dev/zero of=<loop10> bs=100M count=20
```

3. **loop10** ファイルをループデバイスとしてマウントします。

```
$ losetup </dev/loop10>d3 <loop10> ❶ ❷
```

- ❶ ループデバイスがマウントされているファイルパスです。
- ❷ 前の手順で作成したファイルはループデバイスとしてマウントされます。

4. マウントされたループデバイスを参照する **PersistentVolume** 設定を作成します。

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <local-block-pv10>
  annotations:
spec:
  local:
    path: </dev/loop10> ❶
  capacity:
    storage: <2Gi>
  volumeMode: Block ❷
  storageClassName: local ❸
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
              operator: In
              values:
                - <node01> ❹
```

- ❶ ノード上のループデバイスのパス。
- ❷ ブロック PV であることを指定します。

- 3 オプション: PV に StorageClass を設定します。これを省略する場合、クラスターのデフォルトが使用されます。
- 4 ブロックデバイスがマウントされたノード。

5. ブロック PV を作成します。

```
# oc create -f <local-block-pv10.yaml> 1
```

- 1 直前の手順で作成された PersistentVolume のファイル名。

6.11.3.5. DataVolume を使用した仮想マシンイメージのブロック PersistentVolume へのインポート

既存の仮想マシンイメージは OpenShift Container Platform クラスターにインポートできます。Container-native Virtualization は DataVolume を使用してデータのインポートおよび基礎となる PersistentVolumeClaim (PVC) の作成を自動化します。その後、仮想マシン設定で DataVolume を参照できます。

前提条件

- RAW、ISO、または QCOW2 形式の仮想マシンディスクイメージ (オプションで **xz** または **gz** を使用して圧縮される)
- データソースにアクセスするために必要な認証情報と共にイメージがホストされる **HTTP** または **s3** エンドポイント
- 少なくとも1つ以上の利用可能なブロック PV。

手順

- データソースに認証情報が必要な場合、**endpoint-secret.yaml** ファイルを編集し、更新された設定をクラスターに適用します。
 - 選択するテキストエディターで **endpoint-secret.yaml** ファイルを編集します。

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

- 1 オプション: キーまたはユーザー名 (base64 エンコード)
- 2 オプション: シークレットまたはパスワード、base64 エンコード

- シークレットを更新します。

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

2. インポートするイメージのデータソースを指定する **DataVolume** および **volumeMode: Block** を作成して、利用可能なブロック PV が使用されるようにします。

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <import-pv-datavolume> ❶
spec:
  storageClassName: local ❷
  source:
    http:
      url:
        <http://download.fedoraproject.org/pub/fedora/linux/releases/28/Cloud/x86_64/images/Fedora-Cloud-Base-28-1.1.x86_64.qcow2> ❸
        secretRef: <endpoint-secret> ❹
  pvc:
    volumeMode: Block ❺
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi>
```

- ❶ DataVolume の名前。
- ❷ オプション: ストレージクラスを設定するか、またはこれを省略してクラスターのデフォルトを受け入れます。
- ❸ インポートするイメージの **HTTP** ソース。
- ❹ データソースに認証が必要である場合にのみ必要です。
- ❺ ブロック PV にインポートするために必要です。

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

```
$ oc create -f <import-pv-datavolume.yaml> ❶
```

- ❶ 直前の手順で作成されたファイル名 DataVolume。

6.11.3.6. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	<input type="checkbox"/> TAR	<input type="checkbox"/> TAR

✓ 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

+ アーカイブはブロックモード DV をサポートしません。

6.11.4. VMware 仮想マシンまたはテンプレートのインポート

単一の VMware 仮想マシンまたはテンプレートを OpenShift Container Platform クラスターにインポートできます。

VMware テンプレートをインポートする場合、ウィザードはテンプレートに基づいて仮想マシンを作成します。



重要

VMware 仮想マシンまたはテンプレートのインポートはテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。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ついての詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

インポートプロセスでは、VMware Virtual Disk Development Kit (VDDK) を使用して VMware 仮想ディスクをコピーします。VDDK SDK をダウンロードし、VDDK イメージをビルドし、イメージレジストリーにイメージをアップロードしてからこれを **v2v-vmware** ConfigMap に追加できます。

仮想マシンウィザードで VMware 仮想マシンをインポートしてから、仮想マシンのネットワーク名を更新することができます。

6.11.4.1. VDDK イメージのイメージレジストリーの設定

内部 OpenShift Container Platform イメージレジストリーまたは VDDK イメージのセキュアな外部イメージレジストリーのいずれかを設定できます。



注記

VDDK イメージをパブリックリポジトリに保存すると、VMware ライセンスの条件に違反する可能性があります。

6.11.4.1.1. 内部イメージレジストリーの設定

イメージレジストリー Operator 設定を更新して、ベアメタルに内部 OpenShift Container Platform イメージレジストリーを設定できます。

6.11.4.1.1.1. イメージレジストリーの管理状態の変更

イメージレジストリーを起動するには、イメージレジストリー Operator 設定の **managementState** を **Removed** から **Managed** に変更する必要があります。

手順

- **ManagementState** イメージレジストリー Operator 設定を **Removed** から **Managed** に変更します。以下は例になります。

```
$ oc patch configs.imageregistry.operator.openshift.io cluster --type merge --patch '{"spec": {"managementState": "Managed"}}'
```

6.11.4.1.1.2. ベアメタルの場合のレジストリーストレージの設定

クラスター管理者は、インストール後にレジストリーをストレージを使用できるように設定する必要があります。

前提条件

- クラスター管理者のパーミッション。
- ベアメタル上のクラスター。
- Red Hat OpenShift Container Storage などのクラスターのプロビジョニングされた永続ストレージ。



重要

OpenShift Container Platform は、1つのレプリカのみが存在する場合にイメージレジストリーストレージの **ReadWriteOnce** アクセスをサポートします。2つ以上のレプリカで高可用性をサポートするイメージレジストリーをデプロイするには、**ReadWriteMany** アクセスが必要です。

- 100Gi の容量が必要です。

手順

1. レジストリーをストレージを使用できるように設定するには、**configs.imageregistry/cluster** リソースの **spec.storage.pvc** を変更します。



注記

共有ストレージを使用する場合は、外部からアクセスを防ぐためにセキュリティ設定を確認します。

2. レジストリー Pod がないことを確認します。

```
$ oc get pod -n openshift-image-registry
```



注記

ストレージタイプが **emptyDIR** の場合、レプリカ数が **1** を超えることはありません。

3. レジストリー設定を確認します。

```
$ oc edit configs.imageregistry.operator.openshift.io
```

出力例

```
storage:
  pvc:
    claim:
```

claim フィールドを空のままにし、**image-registry-storage** PVC の自動作成を可能にします。

4. **clusteroperator** ステータスを確認します。

```
$ oc get clusteroperator image-registry
```

6.11.4.1.2. 内部イメージレジストリーへのアクセスの設定

レジストリーをルートで公開して、クラスター内から OpenShift Container Platform 内部レジストリーに直接アクセスできます。

6.11.4.1.2.1. クラスターからレジストリーへの直接アクセス

クラスター内からレジストリーにアクセスすることができます。

手順

内部ルートを使用して、クラスターからレジストリーにアクセスします。

1. ノードのアドレスを取得することにより、ノードにアクセスします。

```
$ oc get nodes
$ oc debug nodes/<node_address>
```

2. ノード上で **oc** や **podman** などのツールにアクセスするには、以下のコマンドを実行します。

-

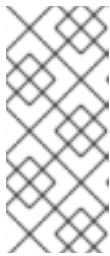
```
sh-4.2# chroot /host
```

- アクセストークンを使用してコンテナイメージレジストリーにログインします。

```
sh-4.2# oc login -u kubeadmin -p <password_from_install_log> https://api-int.
<cluster_name>.<base_domain>:6443
sh-4.2# podman login -u kubeadmin -p $(oc whoami -t) image-registry.openshift-image-
registry.svc:5000
```

以下のようなログインを確認するメッセージが表示されるはずです。

```
Login Succeeded!
```



注記

ユーザー名には任意の値を指定でき、トークンには必要な情報がすべて含まれます。コロンが含まれるユーザー名を指定すると、ログインに失敗します。

イメージレジストリー Operator はルートを作成するため、**default-route-openshift-image-registry.<cluster_name>** のようになります。

- レジストリーに対して **podman pull** および **podman push** 操作を実行します。



重要

任意のイメージをプルできますが、**system:registry** ロールを追加している場合は、各自のプロジェクトにあるレジストリーにのみイメージをプッシュすることができます。

次の例では、以下を使用します。

コンポーネント	値
<registry_ip>	172.30.124.220
<port>	5000
<project>	openshift
<image>	image
<tag>	省略 (デフォルトは latest)

- 任意のイメージをプルします。

```
$ podman pull name.io/image
```

- D. 新規イメージに `<registry_ip>:<port>/<project>/<image>` 形式でタグ付けします。プロジェクト名は、イメージを正しくレジストリーに配置し、これに後でアクセスできるようにするために OpenShift Container Platform のプル仕様に表示される必要があります。

```
$ podman tag name.io/image image-registry.openshift-image-registry.svc:5000/openshift/image
```



注記

指定されたプロジェクトについて **system:image-builder** ロールを持っている必要があります。このロールにより、ユーザーはイメージの書き出しやプッシュを実行できます。そうでない場合は、次の手順の **podman push** は失敗します。テストするために、新規プロジェクトを作成してイメージをプッシュできます。

- c. 新しくタグ付けされたイメージをレジストリーにプッシュします。

```
$ podman push image-registry.openshift-image-registry.svc:5000/openshift/image
```

6.11.4.1.2.2. セキュアなレジストリーの手動による公開

クラスター内から OpenShift Container Platform レジストリーにログインするのではなく、外部からレジストリーにアクセスできるように、このレジストリーをルートに公開します。この方法を使うと、ルートアドレスを使ってクラスターの外部からレジストリーにログインし、ルートのホストを使ってイメージにタグ付けしたり、イメージをプッシュしたりできます。

前提条件:

- 以下の前提条件は自動的に実行されます。
 - レジストリー Operator のデプロイ。
 - Ingress Operator のデプロイ。

手順

configs.imageregistry.operator.openshift.io リソースで **DefaultRoute** パラメーターを使用するか、またはカスタムルートを使用してルートを公開することができます。

DefaultRoute を使用してレジストリーを公開するには、以下を実行します。

1. **DefaultRoute** を **True** に設定します。

```
$ oc patch configs.imageregistry.operator.openshift.io/cluster --patch '{"spec": {"defaultRoute": true}}' --type=merge
```

2. **podman** でログインします。

```
$ HOST=$(oc get route default-route -n openshift-image-registry --template='{{ .spec.host }}')
$ podman login -u $(oc whoami) -p $(oc whoami -t) --tls-verify=false $HOST 1
```

- 1** **--tls-verify=false** は、ルートのクラスターのデフォルト証明書が信頼されない場合に必要になります。Ingress Operator で、信頼されるカスタム証明書をデフォルト証明書として設定できます。

カスタムルートを使用してレジストリーを公開するには、以下を実行します。

1. ルートの TLS キーでシークレットを作成します。

```
$ oc create secret tls public-route-tls \
  -n openshift-image-registry \
  --cert=</path/to/tls.crt> \
  --key=</path/to/tls.key>
```

この手順はオプションです。シークレットを作成しない場合、ルートは Ingress Operator からデフォルトの TLS 設定を使用します。

2. レジストリー Operator では、以下のようになります。

```
spec:
  routes:
    - name: public-routes
      hostname: myregistry.mycorp.organization
      secretName: public-route-tls
  ...
```



注記

レジストリーのルートのカスタム TLS 設定を指定している場合は **secretName** のみを設定します。

6.11.4.1.3. 外部イメージレジストリーへのアクセスの設定

VDDK イメージの外部イメージレジストリーを使用する場合、外部イメージレジストリーの認証局を OpenShift Container Platform クラスターに追加できます。

オプションで、Docker 認証情報からプルシークレットを作成し、これをサービスアカウントに追加できます。

6.11.4.1.3.1. クラスターへの認証局の追加

以下の手順でイメージのプッシュおよびプル時に使用する認証局 (CA) をクラスターに追加することができます。

前提条件

- クラスター管理者の権限があること。
- レジストリーの公開証明書 (通常は、**/etc/docker/certs.d/** ディレクトリーにある **hostname/ca.crt** ファイル)。

手順

1. 自己署名証明書を使用するレジストリーの信頼される証明書が含まれる ConfigMap を **openshift-config** namespace に作成します。それぞれの CA ファイルについて、ConfigMap のキーが **hostname[..port]** 形式のレジストリーのホスト名であることを確認します。

```
$ oc create configmap registry-cas -n openshift-config \
--from-file=myregistry.corp.com..5000=/etc/docker/certs.d/myregistry.corp.com:5000/ca.crt \
--from-file=otherregistry.com=/etc/docker/certs.d/otherregistry.com/ca.crt
```

2. クラスタイメージの設定を更新します。

```
$ oc patch image.config.openshift.io/cluster --patch '{"spec":{"additionalTrustedCA":
{"name":"registry-cas"}}}' --type=merge
```

6.11.4.1.3.2. Pod が他のセキュリティ保護されたレジストリーからイメージを参照できるようにする設定

Docker クライアントの **.dockercfg** **\$HOME/.docker/config.json** ファイルは、セキュア/非セキュアなレジストリーに事前にログインしている場合に認証情報を保存する Docker 認証情報ファイルです。

OpenShift Container Platform の内部レジストリーにないセキュリティ保護されたコンテナイメージをプルするには、Docker 認証情報でプルシークレットを作成し、これをサービスアカウントに追加する必要があります。

手順

- セキュリティ保護されたレジストリーの **.dockercfg** ファイルがすでにある場合は、以下を実行してそのファイルからシークレットを作成できます。

```
$ oc create secret generic <pull_secret_name> \
--from-file=.dockercfg=<path/to/.dockercfg> \
--type=kubernetes.io/dockercfg
```

- または、**\$HOME/.docker/config.json** ファイルがある場合は以下を実行します。

```
$ oc create secret generic <pull_secret_name> \
--from-file=.dockerconfigjson=<path/to/.docker/config.json> \
--type=kubernetes.io/dockerconfigjson
```

- セキュアなレジストリーについての Docker 認証情報ファイルがまだない場合には、以下のコマンドを実行してシークレットを作成することができます。

```
$ oc create secret docker-registry <pull_secret_name> \
--docker-server=<registry_server> \
--docker-username=<user_name> \
--docker-password=<password> \
--docker-email=<email>
```

- Pod のイメージをプルするためのシークレットを使用するには、そのシークレットをサービスアカウントに追加する必要があります。この例では、サービスアカウントの名前は、Pod が使用するサービスアカウントの名前に一致する必要があります。**default** はデフォルトのサービスアカウントです。

```
$ oc secrets link default <pull_secret_name> --for=pull
```

6.11.4.2. VDDK イメージの作成および使用

VMware Virtual Disk Development Kit (VDDK) をダウンロードして、VDDK イメージをビルドし、VDDK イメージをイメージレジストリーにプッシュできます。次に、VDDK イメージを **v2v-vmware** ConfigMap に追加します。

前提条件

- OpenShift Container Platform 内部イメージレジストリーまたはセキュアな外部レジストリーにアクセスできる必要がある。

手順

1. 一時ディレクトリーを作成し、これに移動します。

```
$ mkdir /tmp/<dir_name> && cd /tmp/<dir_name>
```

2. ブラウザーで [VMware code](#) に移動し、**SDKs** をクリックします。
3. **Compute Virtualization** で **Virtual Disk Development Kit(VDDK)** をクリックします。
4. 最新の VDDK リリースを選択し、**Download** をクリックしてから VDDK アーカイブを一時ディレクトリーに保存します。
5. VDDK アーカイブを展開します。

```
$ tar -xzf VMware-vix-disklib-<version>.x86_64.tar.gz
```

6. **Dockerfile** を作成します。

```
$ cat > Dockerfile <<EOF
FROM busybox:latest
COPY vmware-vix-disklib-distrib /vmware-vix-disklib-distrib
RUN mkdir -p /opt
ENTRYPOINT ["cp", "-r", "/vmware-vix-disklib-distrib", "/opt"]
EOF
```

7. イメージをビルドします。

```
$ podman build . -t <registry_route_or_server_path>/vddk:<tag> ❶
```

- ❶ イメージレジストリーを指定します。

- 内部 OpenShift Container Platform レジストリーの場合は、内部レジストリールート (例: **image-registry.openshift-image-registry.svc:5000/openshift/vddk:<tag>**) を使用します。
- 外部レジストリーの場合は、サーバー名、パスおよびタグを指定します (例: **server.example.com:5000/vddk:<tag>**)。

8. イメージをレジストリーにプッシュします。

```
$ podman push <registry_route_or_server_path>/vddk:<tag>
```

9. イメージが OpenShift Container Platform 環境からアクセスできることを確認します。

10. **openshift-cnv** プロジェクトで **v2v-vmware** ConfigMap を編集します。

```
$ oc edit configmap v2v-vmware -n openshift-cnv
```

11. **vddk-init-image** パラメーターを **data** スタンザに追加します。

```
...
data:
  vddk-init-image: <registry_route_or_server_path>/vddk:<tag>
```

6.11.4.3. 仮想マシンウィザードによる VMware 仮想マシンまたはテンプレートのインポート

仮想マシンウィザードを使用して VMware 仮想マシンまたはテンプレートをインポートできます。

前提条件

- VDDK イメージを作成して、これをイメージレジストリーにプッシュし、**v2v-vmware** ConfigMap に追加する必要があります。
- インポートしたディスクに十分なストレージ容量が必要です。



警告

ディスクサイズが利用可能なストレージ領域よりも大きい仮想マシンをインポートしようとする、この操作は完了しません。オブジェクトの削除をサポートするためのリソースが十分でないため、別の仮想マシンをインポートしたり、ストレージをクリーンアップしたりすることはできません。この状況を解決するには、ストレージバックエンドにオブジェクトストレージデバイスを追加する必要があります。


- VMware 仮想マシンの電源がオフになっていること。

手順


1. Container-native Virtualization Web コンソールで **Workloads** → **Virtual Machines** をクリックします。
2. **Create Virtual Machine** をクリックし、**Import with Wizard** を選択します。
3. **General** 画面で、以下の手順を実行します。
 - a. **Provider** 一覧から、**VMware** を選択します。
 - b. **vCenter instance** 一覧から **Connect to New Instance** または保存された vCenter インスタンスを選択します。
 - **Connect to New Instance** を選択する場合、**vCenter hostname**、**Username**、**Password** を入力します。

仮想マシンウィザードを使用して VMware 仮想マシンまたはテンプレートをインポートできます。

- 保存された vCenter インスタンスを選択する場合、ウィザードは保存された認証情報を使用して vCenter に接続します。
 - c. **VM or Template to Import** 一覧からインポートする仮想マシンまたはテンプレートを選択します。
 - d. オペレーティングシステムを選択します。
 - e. **Flavor** 一覧から既存フレーバーまたは **Custom** を選択します。
Custom を選択した場合は、**Memory (GB)** および **CPUs** を指定します。
 - f. **Workload Profile** を選択します。
 - g. 仮想マシン名が namespace の別の仮想マシンで使用されている場合、名前を更新します。
 - h. **Next** をクリックします。
4. **Networking** 画面で以下の手順を実行します。

- a. ネットワークインターフェイスの Options メニュー  をクリックし、**Edit** を選択します。
- b. 有効なネットワークインターフェイス名を入力します。
この名前には、小文字 (**a-z**)、数字 (**0-9**)、およびハイフン (-) を含めることができ、最大 253 文字を使用できます。最初と最後の文字は英数字にする必要があります。この名前には、大文字、スペース、ピリオド (.)、または特殊文字を使用できません。
- c. ネットワークインターフェイスモデルを選択します。
- d. ネットワーク定義を選択します。
- e. ネットワークインターフェイスの種類を選択します。
- f. MAC アドレスを入力します。
- g. **Save** をクリックした後に、**Next** をクリックします。

5. **Storage** 画面で以下の手順を実行します。

- a. ディスクの Options メニュー  をクリックし、**Edit** を選択します。
- b. 有効な名前を入力します。
この名前には、小文字 (**a-z**)、数字 (**0-9**)、およびハイフン (-) を含めることができ、最大 253 文字を使用できます。最初と最後の文字は英数字にする必要があります。この名前には、大文字、スペース、ピリオド (.)、または特殊文字を使用できません。
- c. インターフェイスタイプを選択します。
- d. ストレージクラスを作成します。
ストレージクラスを選択しない場合、Container-native Virtualization はデフォルトストレージクラスを使用して仮想マシンを作成します。
- e. **Save** をクリックした後に、**Next** をクリックします。

6. **Advanced** 画面で、**cloud-init** を使用している場合は **Hostname** および **Authorized SSH Keys** を入力します。
7. **Next** をクリックします。
8. 設定を確認し、**Create Virtual Machine** をクリックします。
Successfully created virtual machine というメッセージが表示され、仮想マシンに作成されたリソースの一覧が表示されます。電源がオフになった仮想マシンが **Workloads → Virtual Machines** に表示されます。
9. **See virtual machine details** をクリックして、インポートされた仮想マシンのダッシュボードを表示します。
エラーが発生した場合は、以下の手順を実行します。
 - a. **Workloads → Pods** をクリックします。
 - b. 変換 Pod (例: **kubevirt-v2v-conversion-rhel7-mini-1-27b9h**) をクリックします。
 - c. **Logs** をクリックし、エラーメッセージの有無を確認します。

ウィザードのフィールドについての詳細は、[virtual machine wizard fields](#) を参照してください。

6.11.4.4. インポートされた VMware 仮想マシンの NIC 名の更新

VMware からインポートされた仮想マシンの NIC 名を、Container-native Virtualization の命名規則に適合するように更新します。

手順

1. 仮想マシンにログインします。
2. **/etc/sysconfig/network-scripts** ディレクトリーに移動します。
3. ネットワーク設定ファイル名を **ifcfg-eth0** に変更します。

```
$ mv vmnic0 ifcfg-eth0 1
```

- 1 追加のネットワーク設定ファイルには、**ifcfg-eth1**、**ifcfg-eth2** などの番号が順番に付けられます。

4. ネットワーク設定ファイルで **NAME** および **DEVICE** パラメーターを更新します。

```
NAME=eth0
DEVICE=eth0
```

5. ネットワークを再起動します。

```
$ systemctl restart network
```

6.11.4.5. VMware 仮想マシンのインポートのトラブルシューティング

インポートされた仮想マシンのステータスが **Import error: (VMware)** の場合は、Conversion Pod ログでエラーの有無を確認できます。

1. Conversion Pod 名を取得します。

```
$ oc get pods -n <project> | grep v2v ①
kubevirt-v2v-conversion-f66f7d-zqkz7      1/1   Running   0      4h49m
```

- ① インポートされた仮想マシンのプロジェクトを指定します。

2. Conversion Pod ログを取得します。

```
$ oc logs kubevirt-v2v-conversion-f66f7d-zqkz7 -f -n <project>
```

6.11.4.5.1. エラーメッセージ

- インポートされた仮想マシンイベントでエラーメッセージ **Readiness probe failed** が表示される場合、以下のエラーメッセージが Conversion Pod ログに表示されます。

```
INFO - have error: ('virt-v2v error: internal error: invalid argument: libvirt domain
'v2v_migration_vm_1' is running or paused. It must be shut down in order to perform virt-v2v
conversion',)"
```

インポートする前に、仮想マシンがシャットダウンしていることを確認する必要があります。

6.11.4.5.2. 既知の問題

- OpenShift Container Platform 環境には、インポートされたディスク用のストレージ容量が十分にある必要があります。
ディスクサイズが利用可能なストレージ領域よりも大きい仮想マシンをインポートしようとすると、この操作は完了しません。オブジェクトの削除をサポートするためのリソースが十分でないため、別の仮想マシンをインポートしたり、ストレージをクリーンアップしたりすることはできません。この状況を解決するには、ストレージバックエンドにオブジェクトストレージデバイスをさらに追加する必要があります。(BZ#1721504)
- NFS を使用したストレージを、変換 Pod に割り当てられる 2 GB ディスクに使用する場合は、[hostPath ボリュームを設定](#) する必要があります。(BZ#1814611)

6.11.4.6. 仮想マシンウィザードのフィールド

6.11.4.6.1. 仮想マシンウィザードのフィールド

名前	パラメーター	説明
Template		仮想マシンの作成に使用するテンプレート。テンプレートを選択すると、他のフィールドが自動的に入力されます。
ソース	PXE	PXE メニューから仮想マシンをプロビジョニングします。クラスターに PXE 対応の NIC が必要になります。

名前	パラメーター	説明
	URL	HTTP または S3 エンドポイントで利用できるイメージから仮想マシンをプロビジョニングします。
	コンテナ	クラスターからアクセスできるレジストリーの起動可能なオペレーティングシステムコンテナから仮想マシンをプロビジョニングします。例: kubevirt/cirros-registry-disk-demo
	Disk	ディスクから仮想マシンをプロビジョニングします。
Operating System		仮想マシン用に選択される主なオペレーティングシステム。
Flavor	small、medium、large、tiny、Custom	仮想マシンに割り当てられる CPU およびメモリーの量を決定するプリセット。 Flavor に設定される Preset はオペレーティングシステムによって決まります。
Memory		仮想マシンに割り当てられるメモリーのサイズ (GiB 単位)。
CPU		仮想マシンに割り当てられる CPU の量。
Workload Profile	High Performance	高パフォーマンスのワークロードに対して最適化された仮想マシン設定。
	Server	サーバーワークロードの実行に最適化されたプロファイル。
	Desktop	デスクトップで使用するための仮想マシン設定。
名前		この名前には、小文字 (a-z)、数字 (0-9)、およびハイフン (-) を含めることができ、最大 253 文字を使用できます。最初と最後の文字は英数字にする必要があります。この名前には、大文字、スペース、ピリオド (.), または特殊文字を使用できません。

名前	パラメーター	説明
説明		オプションの説明フィールド。
Start virtual machine on creation		これを選択すると、作成時に仮想マシンが自動的に起動します。

6.11.4.6.2. Cloud-init フィールド

名前	説明
Hostname	仮想マシンの特定のホスト名を設定します。
Authenticated SSH Keys	仮想マシンの <code>~/.ssh/authorized_keys</code> にコピーされるユーザーの公開鍵。
Use custom script	他のオプションを、カスタム cloud-init スクリプトを貼り付けるフィールドに置き換えます。

6.11.4.6.3. ネットワークフィールド

名前	説明
名前	ネットワークインターフェイスの名前。
Model	ネットワークインターフェイスカードのモデルを示します。サポートされる値は、 e1000 、 e1000e 、 ne2k_pci 、 pcnet 、 rtl8139 、および virtIO です。
Network	利用可能な NetworkAttachmentDefinition オブジェクトの一覧。
Type	利用可能なバインディングメソッドの一覧。デフォルトの Pod ネットワークについては、 masquerade が唯一の推奨されるバインディングメソッドになります。セカンダリーネットワークの場合は、 bridge バインディングメソッドを使用します。 masquerade メソッドは、デフォルト以外のネットワークではサポートされません。
MAC Address	ネットワークインターフェイスの MAC アドレス。MAC アドレスが指定されていない場合、セッションの一時アドレスが生成されます。

6.11.4.6.4. ストレージフィールド

名前	説明
ソース	仮想マシンの空のディスクを選択するか、または URL 、 Container 、 Attach Cloned Disk 、または Attach Disk などの選択可能なオプションから選択します。既存ディスクを選択し、これを仮想マシンに割り当てるには、利用可能な PersistentVolumeClaim (PVC) の一覧から Attach Cloned Disk または Attach Disk を選択します。
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 (GiB)	ディスクのサイズ (GiB)。
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 StorageClass 。

6.12. 仮想マシンのクローン作成

6.12.1. 複数の namespace 間で DataVolume 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 ClusterRole を作成する必要があります。この ClusterRole 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。

6.12.1.1. 前提条件

- **cluster-admin** ロールを持つユーザーのみが ClusterRole を作成できます。

6.12.1.2. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.12.1.3. DataVolume 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 ClusterRole を作成します。

手順

1. ClusterRole マニフェストを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: <datavolume-cloner> ❶
rules:
- apiGroups: ["cdi.kubevirt.io"]
  resources: ["datavolumes/source"]
  verbs: ["*"]
```

- ❶ ClusterRole の一意の名前。

2. クラスターに ClusterRole を作成します。

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

- ❶ 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 ClusterRole を参照します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: <allow-clone-to-user> ❶
  namespace: <Source namespace> ❷
subjects:
- kind: ServiceAccount
  name: default
  namespace: <Destination namespace> ❸
roleRef:
  kind: ClusterRole
  name: datavolume-cloner ❹
  apiGroup: rbac.authorization.k8s.io
```

- ❶ RoleBinding の一意の名前。
- ❷ ソース DataVolume の namespace。
- ❸ DataVolume のクローンが作成される namespace。
- ❹ 直前の手順で作成した ClusterRole の名前。

4. クラスターに RoleBinding を作成します。

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

-

- 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。

6.12.2. 新規 DataVolume への仮想マシンディスクのクローン作成

DataVolume 設定ファイルでソース PVC を参照し、新規 DataVolume に仮想マシンディスクの PersistentVolumeClaim (PVC) のクローンを作成できます。

6.12.2.1. 前提条件

- この操作を正常に実行するためには、[StorageClass を定義するか、CDI のスクラッチ領域を用意](#) する必要がある場合があります。[CDI がサポートする操作マトリックス](#) は、スクラッチ領域を必要とする条件を示しています。
- ユーザーは、仮想マシンディスクの PVC のクローンを別の namespace に作成するために [追加のパーミッション](#) が必要です。

6.12.2.2. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.12.2.3. 新規 DataVolume への仮想マシンディスクの PersistentVolumeClaim のクローン作成

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



注記

Volume が仮想マシンとは別に作成される場合、DataVolume のライフサイクルは仮想マシンから切り離されます。仮想マシンが削除されても、DataVolume もその関連付けられた PVC も削除されません。

前提条件

- 使用する既存の仮想マシンディスクの PVC を判別すること。クローン作成の前に、PVC に関連付けられた仮想マシンの電源を切る必要があります。
- OpenShift CLI (**oc**) をインストールしている。

手順

- 関連付けられた PVC の名前および namespace を特定するために、クローン作成に必要な仮想マシンディスクを確認します。
- 新規 DataVolume の名前、ソース PVC の名前および namespace、および新規 DataVolume のサイズを指定する DataVolume オブジェクトの YAML ファイルを作成します。
以下は例になります。

```
apiVersion: cdi.kubevirt.io/v1alpha1
```

```

kind: DataVolume
metadata:
  name: <cloner-datavolume> ❶
spec:
  source:
    pvc:
      namespace: "<source-namespace>" ❷
      name: "<my-favorite-vm-disk>" ❸
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> ❹

```

- ❶ 新規 DataVolume の名前。
- ❷ ソース PVC が存在する namespace。
- ❸ ソース PVC の名前。
- ❹ 新規 DataVolume のサイズ。十分な領域を割り当てる必要があります。そうでない場合には、クローン操作は失敗します。サイズはソース PVC と同じか、またはそれよりも大きくなければなりません。

3. DataVolume を作成して PVC のクローン作成を開始します。

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



注記

DataVolume は仮想マシンが PVC の作成前に起動することを防ぐため、PVC のクローン作成中に新規 DataVolume を参照する仮想マシンを作成できます。

6.12.2.4. テンプレート: DataVolume クローン設定ファイル

example-clone-dv.yaml

```

apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: "example-clone-dv"
spec:
  source:
    pvc:
      name: source-pvc
      namespace: example-ns
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: "1G"

```

6.12.2.5. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

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

✓ 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

+ アーカイブはブロックモード DV をサポートしません。

6.12.3. DataVolumeTemplate の使用による仮想マシンのクローン作成

既存の仮想マシンの PersistentVolumeClaim (PVC) のクローン作成により、新規の仮想マシンを作成できます。**dataVolumeTemplate** を仮想マシン設定ファイルに含めることにより、元の PVC から新規の DataVolume を作成します。

6.12.3.1. 前提条件

- この操作を正常に実行するためには、[StorageClass](#) を定義するか、CDI のスクラッチ領域を用意する必要がある場合があります。[CDI がサポートする操作マトリックス](#) は、スクラッチ領域を必要とする条件を表示します。
- ユーザーは、仮想マシンディスクの PVC のクローンを別の namespace に作成するために [追加のパーミッション](#) が必要です。

6.12.3.2. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.12.3.3. DataVolumeTemplate の使用による、クローン作成された PersistentVolumeClaim からの仮想マシンの新規作成

既存の仮想マシンの PersistentVolumeClaim (PVC) のクローンを DataVolume に作成する仮想マシンを作成できます。仮想マシン **spec** の **dataVolumeTemplate** を参照することにより、**source** PVC のクローンを DataVolume に作成され、これは次に仮想マシンを作成するために自動的に使用されます。



注記

DataVolume が仮想マシンの DataVolumeTemplate の一部として作成されると、DataVolume のライフサイクルは仮想マシンに依存します。つまり、仮想マシンが削除されると、DataVolume および関連付けられた PVC も削除されます。

前提条件

- 使用する既存の仮想マシンディスクの PVC を判別すること。クローン作成の前に、PVC に関連付けられた仮想マシンの電源を切る必要があります。
- OpenShift CLI (**oc**) をインストールしている。

手順

1. 関連付けられた PVC の名前および namespace を特定するために、クローン作成に必要な仮想マシンを確認します。
2. **VirtualMachine** オブジェクトの YAML ファイルを作成します。以下の仮想マシンのサンプルでは、**source-namespace** namespace にある **my-favorite-vm-disk** のクローンを作成します。**favorite-clone** という **2Gi** DataVolume が **my-favorite-vm-disk** から作成されます。以下は例になります。

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: vm-dv-clone
  name: vm-dv-clone 1
spec:
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: vm-dv-clone
    spec:
      domain:
        devices:
          disks:
            - disk:
                bus: virtio
                name: root-disk
          resources:
            requests:
              memory: 64M
          volumes:
            - dataVolume:
                name: favorite-clone
                name: root-disk
      dataVolumeTemplates:
        - metadata:
```

```

    name: favorite-clone
  spec:
    pvc:
      accessModes:
        - ReadWriteOnce
      resources:
        requests:
          storage: 2Gi
    source:
      pvc:
        namespace: "source-namespace"
        name: "my-favorite-vm-disk"

```

① 作成する仮想マシン。

3. PVC のクローンが作成された DataVolume で仮想マシンを作成します。

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

6.12.3.4. テンプレート: DataVolume 仮想マシン設定ファイル

example-dv-vm.yaml

```

apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  labels:
    kubevirt.io/vm: example-vm
  name: example-vm
spec:
  dataVolumeTemplates:
    - metadata:
        name: example-dv
      spec:
        pvc:
          accessModes:
            - ReadWriteOnce
          resources:
            requests:
              storage: 1G
        source:
          http:
            url: "" ①
  running: false
  template:
    metadata:
      labels:
        kubevirt.io/vm: example-vm
    spec:
      domain:
        cpu:
          cores: 1
      devices:
        disks:

```



```

- disk:
  bus: virtio
  name: example-dv-disk
machine:
  type: q35
resources:
  requests:
    memory: 1G
terminationGracePeriodSeconds: 0
volumes:
- dataVolume:
  name: example-dv
  name: example-dv-disk

```

- 1 インポートする必要があるイメージの **HTTP** ソース (該当する場合)。

6.12.3.5. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	<input type="checkbox"/> TAR	<input type="checkbox"/> TAR

✓ 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

+ アーカイブはブロックモード DV をサポートしません。

6.12.4. 新規ブロックストレージ **DataVolume** への仮想マシンディスクのクローン作成

DataVolume 設定ファイルでソース PVC を参照し、新規ブロック DataVolume に仮想マシンディスクの PersistentVolumeClaim (PVC) のクローンを作成できます。

6.12.4.1. 前提条件

- [CDI でサポートされる操作マトリックス](#) に応じてスクラッチ領域が必要な場合、まずは、この操作が正常に実行されるように [StorageClass](#) を定義するか、または [CDI スクラッチ領域](#) を用意します。
- ユーザーは、仮想マシンディスクの PVC のクローンを別の namespace に作成するために [追加のパーミッション](#) が必要です。

6.12.4.2. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.12.4.3. ブロック PersistentVolume について

ブロック PersistentVolume (PV) は、raw ブロックデバイスによってサポートされる PV です。これらのボリュームにはファイルシステムがなく、ディスクに直接書き込む仮想マシンや、独自のストレージサービスを実装する仮想マシンにはパフォーマンス上の利点があります。

raw ブロックボリュームは、PV および PersistentVolumeClaim (PVC) 仕様で **volumeMode: Block** を指定してプロビジョニングされます。

6.12.4.4. ローカルブロック PersistentVolume の作成

ファイルにデータを設定し、これをループデバイスとしてマウントすることにより、ノードでローカルブロック PersistentVolume (PV) を作成します。次に、このループデバイスを PV 設定で **Block** ボリュームとして参照し、これを仮想マシンイメージのブロックデバイスとして使用できます。

手順

1. ローカル PV を作成するノードに **root** としてログインします。この手順では、**node01** を例に使用します。
2. ファイルを作成して、これを null 文字で設定し、ブロックデバイスとして使用できるようにします。以下の例では、2Gb (20 100Mb ブロック) のサイズのファイル **loop10** を作成します。

```
$ dd if=/dev/zero of=<loop10> bs=100M count=20
```

3. **loop10** ファイルをループデバイスとしてマウントします。

```
$ losetup </dev/loop10>d3 <loop10> 1 2
```

- 1 ループデバイスがマウントされているファイルパスです。
- 2 前の手順で作成したファイルはループデバイスとしてマウントされます。

4. マウントされたループデバイスを参照する **PersistentVolume** 設定を作成します。

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <local-block-pv10>
  annotations:
```

```
spec:
  local:
    path: </dev/loop10> ❶
    capacity:
      storage: <2Gi>
    volumeMode: Block ❷
    storageClassName: local ❸
    accessModes:
      - ReadWriteOnce
    persistentVolumeReclaimPolicy: Delete
    nodeAffinity:
      required:
        nodeSelectorTerms:
          - matchExpressions:
              - key: kubernetes.io/hostname
                operator: In
                values:
                  - <node01> ❹
```

- ❶ ノード上のループデバイスのパス。
- ❷ ブロック PVであることを指定します。
- ❸ オプション: PVに StorageClassを設定します。これを省略する場合、クラスターのデフォルトが使用されます。
- ❹ ブロックデバイスがマウントされたノード。

5. ブロック PV を作成します。

```
# oc create -f <local-block-pv10.yaml> ❶
```

- ❶ 直前の手順で作成された PersistentVolume のファイル名。

6.12.4.5. 新規 DataVolume への仮想マシンディスクの PersistentVolumeClaim のクローン作成

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



注記

Volume が仮想マシンとは別に作成される場合、DataVolume のライフサイクルは仮想マシンから切り離されます。仮想マシンが削除されても、DataVolume もその関連付けられた PVC も削除されません。

前提条件

- 使用する既存の仮想マシンディスクの PVC を判別すること。クローン作成の前に、PVCに関連付けられた仮想マシンの電源を切る必要があります。
- OpenShift CLI (**oc**) のインストール。

- ソース PVC と同じか、またはこれよりも大きい1つ以上の利用可能なブロック PersistentVolume (PV)。

手順

1. 関連付けられた PVC の名前および namespace を特定するために、クローン作成に必要な仮想マシンディスクを確認します。
2. 新規 DataVolume の名前、ソース PVC の名前および namespace、利用可能なブロック PV を使用できるようにするために **volumeMode: Block**、および新規 DataVolume のサイズを指定する DataVolume オブジェクトの YAML ファイルを作成します。
以下は例になります。

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <cloner-datavolume> ❶
spec:
  source:
    pvc:
      namespace: "<source-namespace>" ❷
      name: "<my-favorite-vm-disk>" ❸
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> ❹
    volumeMode: Block ❺
```

- ❶ 新規 DataVolume の名前。
- ❷ ソース PVC が存在する namespace。
- ❸ ソース PVC の名前。
- ❹ 新規 DataVolume のサイズ。十分な領域を割り当てる必要があります。そうでない場合には、クローン操作は失敗します。サイズはソース PVC と同じか、またはそれよりも大きくなければなりません。
- ❺ 宛先がブロック PVであることを指定します。

3. DataVolume を作成して PVC のクローン作成を開始します。

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



注記

DataVolume は仮想マシンが PVC の作成前に起動することを防ぐため、PVC のクローン作成中に新規 DataVolume を参照する仮想マシンを作成できます。

6.12.4.6. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

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

✓ 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

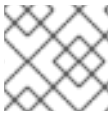
**カスタム認証局が必要な場合にスクラッチ領域が必要

+ アーカイブはブロックモード DV をサポートしません。

6.13. 仮想マシンのネットワーク

6.13.1. 仮想マシンのデフォルト Pod ネットワークの使用

Container-native Virtualization でデフォルトの Pod ネットワークを使用できます。これを実行するには、**masquerade** バインディングメソッドを使用する必要があります。これは、デフォルトの Pod ネットワークを使用する場合にのみ推奨されるバインディングメソッドです。デフォルト以外のネットワークには、**masquerade** モードを使用しないでください。



注記

セカンダリーネットワークの場合は、**bridge** バインディングメソッドを使用します。

6.13.1.1. コマンドラインでのマスカレードモードの設定

マスカレードモードを使用し、仮想マシンの送信トラフィックを Pod IP アドレスの背後で非表示にすることができます。マスカレードモードは、ネットワークアドレス変換 (NAT) を使用して仮想マシンを Linux ブリッジ経由で Pod ネットワークバックエンドに接続します。

仮想マシンの設定ファイルを編集して、マスカレードモードを有効にし、トラフィックが仮想マシンに到達できるようにします。

前提条件

- 仮想マシンは、IPv4 アドレスを取得するために DHCP を使用できるように設定される必要があります。以下の例では、DHCP を使用するように設定されます。

手順

1. 仮想マシン設定ファイルの **interfaces** 仕様を編集します。

```
kind: VirtualMachine
spec:
  domain:
    devices:
      interfaces:
        - name: red
          masquerade: {} ❶
        ports:
          - port: 80 ❷
  networks:
    - name: red
      pod: {}
```

- ❶ マスカレードモードを使用した接続
- ❷ ポート 80 での受信トラフィックの許可

2. 仮想マシンを作成します。

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

6.13.1.2. バインディング方法の選択

Container-native Virtualization [Web コンソールウィザード](#) から仮想マシンを作成する場合、**Networking** 画面で必要なバインディングメソッドを選択します。

6.13.1.2.1. ネットワークフィールド

名前	説明
名前	ネットワークインターフェイスの名前。
Model	ネットワークインターフェイスカードのモデルを示します。サポートされる値は、 e1000 、 e1000e 、 ne2k_pci 、 pcnet 、 rtl8139 、および virtIO です。
Network	利用可能な NetworkAttachmentDefinition オブジェクトの一覧。
Type	利用可能なバインディングメソッドの一覧。デフォルトの Pod ネットワークについては、 masquerade が唯一の推奨されるバインディングメソッドになります。セカンダリーネットワークの場合は、 bridge バインディングメソッドを使用します。 masquerade メソッドは、デフォルト以外のネットワークではサポートされません。

名前	説明
MAC Address	ネットワークインターフェースの MAC アドレス。 MAC アドレスが指定されていない場合、セッションの一時アドレスが生成されます。

6.13.1.3. デフォルトネットワーク用の仮想マシン設定の例

6.13.1.3.1. テンプレート: 仮想マシンの設定ファイル

```

apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  name: example-vm
  namespace: default
spec:
  running: false
  template:
    spec:
      domain:
        devices:
          disks:
            - name: containerdisk
              disk:
                bus: virtio
            - name: cloudinitdisk
              disk:
                bus: virtio
          interfaces:
            - masquerade: {}
              name: default
      resources:
        requests:
          memory: 1024M
      networks:
        - name: default
          pod: {}
      volumes:
        - name: containerdisk
          containerDisk:
            image: kubevirt/fedora-cloud-container-disk-demo
        - name: cloudinitdisk
          cloudInitNoCloud:
            userData: |
              #!/bin/bash
              echo "fedora" | passwd fedora --stdin

```

6.13.1.3.2. テンプレート: Windows 仮想マシンインスタンスの設定ファイル

```

apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:

```

```

labels:
  special: vmi-windows
name: vmi-windows
spec:
  domain:
    clock:
      timer:
        hpet:
          present: false
        hyperv: {}
        pit:
          tickPolicy: delay
        rtc:
          tickPolicy: catchup
      utc: {}
  cpu:
    cores: 2
  devices:
    disks:
      - disk:
          bus: sata
          name: pvcdisk
    interfaces:
      - masquerade: {}
        model: e1000
        name: default
  features:
    acpi: {}
    apic: {}
    hyperv:
      relaxed: {}
      spinlocks:
        spinlocks: 8191
      vpic: {}
  firmware:
    uuid: 5d307ca9-b3ef-428c-8861-06e72d69f223
  machine:
    type: q35
  resources:
    requests:
      memory: 2Gi
  networks:
    - name: default
      pod: {}
  terminationGracePeriodSeconds: 0
  volumes:
    - name: pvcdisk
      persistentVolumeClaim:
        claimName: disk-windows

```

6.13.2. 仮想マシンの複数ネットワークへの割り当て

Container-native Virtualization は、仮想マシンの複数ネットワークへの接続を可能にする layer-2 vNIC ネットワーク機能を提供します。複数インターフェイスへのアクセスによって異なる既存のワークロードを持つ仮想マシンをインポートできます。また、仮想マシンをネットワーク経由で起動できるように

PXE ネットワークを設定することもできます。

まず、ネットワーク管理者は Web コンソールまたは CLI で namespace のブリッジ NetworkAttachmentDefinition を設定します。次に、ユーザーは NIC を作成し、その namespace 内の Pod および仮想マシンをブリッジネットワークに割り当てることができます。

6.13.2.1. Container-native Virtualization ネットワークの用語集

Container-native Virtualization は、カスタムリソースおよびプラグインを使用して高度なネットワーク機能を提供します。

以下の用語は、Container-native 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。Container-native 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、Customer 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。

NetworkAttachmentDefinition

Pod、仮想マシン、および仮想マシンインスタンスを1つ以上のネットワークに割り当てておくことを可能にする Multus プロジェクトによって導入される CRD。

PXE (Preboot eXecution Environment)

管理者がネットワーク経由でサーバーからクライアントマシンを起動できるようにするインターフェイス。ネットワークのブートにより、オペレーティングシステムおよび他のソフトウェアをクライアントにリモートでロードできます。

6.13.2.2. NetworkAttachmentDefinition の作成

6.13.2.3. 前提条件

- Linux ブリッジは、すべてのノードに設定して割り当てておく必要がある。詳細は、[ノードのネットワーク](#) セクションを参照してください。

6.13.2.3.1. Web コンソールでの Linux ブリッジ NetworkAttachmentDefinition の作成

NetworkAttachmentDefinition は、layer-2 デバイスを Container-native Virtualization クラスターの特定の namespace に公開するカスタムリソースです。

ネットワーク管理者は、NetworkAttachmentDefinition を作成して既存の layer-2 ネットワークを Pod および仮想マシンに提供できます。

手順

- Web コンソールで、**Networking** → **Network Attachment Definitions** をクリックします。
- Create Network Attachment Definition** をクリックします。

3. 一意の **Name** およびオプションの **Description** を入力します。
4. **Network Type** 一覧をクリックし、**CNV Linux bridge** を選択します。
5. **Bridge Name** フィールドにブリッジの名前を入力します。
6. (オプション) リソースに VLAN ID が設定されている場合、**VLAN Tag Number** フィールドに ID 番号を入力します。
7. **Create** をクリックします。

6.13.2.3.2. CLI での Linux ブリッジ NetworkAttachmentDefinition の作成

ネットワーク管理者は、タイプ **cnv-bridge** の NetworkAttachmentDefinition を、レイヤー 2 ネットワークを Pod および仮想マシンに提供するように設定できます。



注記

NetworkAttachmentDefinition は Pod または仮想マシンと同じ namespace にある必要があります。

手順

1. 任意のローカルディレクトリーで NetworkAttachmentDefinition の新規ファイルを作成します。このファイルには、お使いの設定に合わせて変更された以下の内容が含まれる必要があります。

```
apiVersion: "k8s.cni.cncf.io/v1"
kind: NetworkAttachmentDefinition
metadata:
  name: a-bridge-network
  annotations:
    k8s.v1.cni.cncf.io/resourceName: bridge.network.kubevirt.io/br0 ❶
spec:
  config: '{
    "cniVersion": "0.3.1",
    "name": "a-bridge-network", ❷
    "plugins": [
      {
        "type": "cnv-bridge", ❸
        "bridge": "br0" ❹
      },
      {
        "type": "cnv-tuning" ❺
      }
    ]
  }'
```

- ❶ このアノテーションを NetworkAttachmentDefinition に追加する場合、仮想マシンインスタンスは **br0** ブリッジが接続されているノードでのみ実行されます。
- ❷ 必須。ConfigMap の名前を入力します。設定名を NetworkAttachmentDefinition の **name** 値に一致させることが推奨されます。
- ❸ この NetworkAttachmentDefinition のネットワークを提供する Container Network

Interface (CNI) フラグインの実際の名前。異なる CNI を使用するのでない限り、このフィールドは変更しないでください。

- ④ ブリッジの名前が **br0** でない場合、ブリッジの実際の名前に置き換える必要があります。
- ⑤ 必須。これにより、MAC プールマネージャーが接続に固有の MAC アドレスを割り当てます。

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

2. ブリッジネットワークに接続する必要のある仮想マシンまたは仮想マシンインスタンスの設定を編集します。

```
apiVersion: v1
kind: VirtualMachine
metadata:
  name: example-vm
spec:
  domain:
    devices:
      interfaces:
        - masquerade: {}
          name: default
        - bridge: {}
          name: bridge-net ①
    ...

  networks:
    - name: default
      pod: {}
    - name: bridge-net ②
    multus:
      networkName: a-bridge-network ③
    ...
```

- ① ② ブリッジインターフェイスおよびネットワークの **name** の値は同じである必要があります。

- ③ NetworkAttachmentDefinition の実際の **name** 値に置き換える必要があります。



注記

仮想マシンインスタンスは、**default** Pod ネットワーク、および **a-bridge-network** という名前の NetworkAttachmentDefinition によって定義される **bridge-net** の両方に接続されます。

3. 設定ファイルをリソースに適用します。

```
$ oc create -f <local/path/to/network-attachment-definition.yaml>
```



注記

次のセクションで vNIC を定義する際に、**NETWORK** の値が直前のセクションで作成した NetworkAttachmentDefinition のブリッジネットワーク名であることを確認します。

6.13.2.4. 仮想マシンの NIC の作成

Web コンソールから追加の NIC を作成し、これを仮想マシンに割り当てます。

手順

1. Container-native Virtualization コンソールの適切なプロジェクトで、**Workloads → Virtual Machines** をクリックします。
2. 仮想マシンを選択します。
3. **Network Interfaces** をクリックし、仮想マシンにすでに割り当てられている NIC を表示します。
4. **Create Network Interface** をクリックし、一覧に新規スロットを作成します。
5. 新規 NIC の **Name**、**Model**、**Network**、**Type**、および **MAC Address** を入力します。
6. ✓ ボタンをクリックして NIC を保存し、これを仮想マシンに割り当てます。

6.13.2.5. ネットワークフィールド

名前	説明
名前	ネットワークインターフェイスの名前。
Model	ネットワークインターフェイスカードのモデルを示します。サポートされる値は、 e1000 、 e1000e 、 ne2k_pci 、 pcnet 、 rtl8139 、および virtIO です。
Network	利用可能な NetworkAttachmentDefinition オブジェクトの一覧。
Type	利用可能なバインディングメソッドの一覧。デフォルトの Pod ネットワークについては、 masquerade が唯一の推奨されるバインディングメソッドになります。セカンダリーネットワークの場合は、 bridge バインディングメソッドを使用します。 masquerade メソッドは、デフォルト以外のネットワークではサポートされません。
MAC Address	ネットワークインターフェイスの MAC アドレス。MAC アドレスが指定されていない場合、セッションの一時アドレスが生成されます。

仮想マシンにオプションの **QEMU ゲストエージェント** をインストールし、ホストが追加のネットワークについての関連情報を表示できるようにします。

6.13.3. QEMU ゲストエージェントの仮想マシンへのインストール

QEMU ゲストエージェントは仮想マシンで実行されるデーモンです。エージェントは仮想マシン上で、追加ネットワークの IP アドレスなどのネットワーク情報をホストに渡します。

6.13.3.1. 前提条件

- 以下のコマンドを実行して、ゲストエージェントがインストールされており、実行中であることを確認します。

```
$ systemctl status qemu-guest-agent
```

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

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

手順

- コンソールのいずれか、または SSH を使用して仮想マシンのコマンドラインにアクセスします。
- QEMU ゲストエージェントを仮想マシンにインストールします。

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

- QEMU ゲストエージェントサービスを起動します。

```
$ systemctl start qemu-guest-agent
```

- サービスに永続性があることを確認します。

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

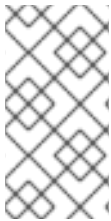
Web コンソールで仮想マシンまたは仮想マシンテンプレートのいずれかを作成する際に、ウィザードの **cloud-init** セクションの **custom script** フィールドを使用して QEMU ゲストエージェントをインストールし、起動することもできます。

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

Windows 仮想マシンの場合、QEMU ゲストエージェントは、以下の手順のいずれかを使用してインストールできる VirtIO ドライバーに含まれています。

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

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



注記

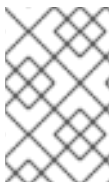
この手順では、ドライバーを Windows に追加するための汎用的なアプローチを使用しています。このプロセスは Windows のバージョンごとに若干異なる可能性があります。特定のインストール手順については、お使いの Windows バージョンについてのインストールドキュメントを参照してください。

手順

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

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

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



注記

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

手順

1. 仮想マシンを起動し、グラフィカルコンソールに接続します。
2. Windows インストールプロセスを開始します。
3. **Advanced** インストールを選択します。

4. ストレージの宛先は、ドライバーがロードされるまで認識されません。**Load driver** をクリックします。
5. ドライバーは SATA CD ドライブとして割り当てられます。**OK** をクリックし、CD ドライバーでロードするストレージドライバーを参照します。ドライバーは、ドライバーのタイプ、オペレーティングシステム、および CPU アーキテクチャー別に階層的に編成されます。
6. 必要なすべてのドライバーについて直前の 2 つの手順を繰り返します。
7. Windows インストールを完了します。

6.13.4. vNIC の IP アドレスの仮想マシンへの表示

QEMU ゲストエージェントは仮想マシンで実行され、割り当てられた NIC の IP アドレスをホストに渡します。これにより、Web コンソールおよび **oc** クライアントの両方から IP アドレスを表示できます。

6.13.4.1. 前提条件

1. 以下のコマンドを実行して、ゲストエージェントがインストールされており、実行中であることを確認します。

```
$ systemctl status qemu-guest-agent
```

2. ゲストエージェントがインストールされておらず、実行されていない場合は、[仮想マシン上でゲストエージェントをインストールし、実行します](#)。

6.13.4.2. CLI での仮想マシンインターフェイスの IP アドレスの表示

ネットワークインターフェイス設定は **oc describe vmi <vmi_name>** コマンドに含まれます。

IP アドレス情報は、仮想マシン上で **ip addr** を実行するか、または **oc get vmi <vmi_name> -o yaml** を実行して表示することもできます。

手順

- **oc describe** コマンドを使用して、仮想マシンインターフェイス設定を表示します。

```
$ oc describe vmi <vmi_name>

...
Interfaces:
  Interface Name: eth0
  Ip Address:    10.244.0.37/24
  Ip Addresses:
    10.244.0.37/24
    fe80::858:aff:fe4:25/64
  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

```

6.13.4.3. Web コンソールでの仮想マシンインターフェイスの IP アドレスの表示

IP 情報は、仮想マシンの **Virtual Machine Overview** 画面に表示されます。

手順

1. Container-native Virtualization コンソールで **Workloads** → **Virtual Machines** をクリックします。
2. 仮想マシンの名前をクリックして、**Virtual Machine Overview** 画面を開きます。

それぞれの割り当てられた NIC の情報は **IP ADDRESSES** の下に表示されます。

6.14. 仮想マシンディスク

6.14.1. 仮想マシンのローカルストレージの設定

ホストパスプロビジョナー機能を使用して、仮想マシンのローカルストレージを設定できます。

6.14.1.1. ホストパスプロビジョナーについて

ホストパスプロビジョナーは、Container-native 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。

Container-native Virtualization Operator のインストール時に、ホストパスプロビジョナー Operator は自動的にインストールされます。これを使用するには、以下を実行する必要があります。

- SELinux を設定します。
 - Red Hat Enterprise Linux CoreOS 8 ワーカーを使用する場合は、各ノードに MachineConfig オブジェクトを作成する必要があります。
 - それ以外の場合には、SELinux ラベル **container_file_t** を各ノードの PersistentVolume (PV) バッキングディレクトリーに適用します。
- HostPathProvisioner カスタムリソースを作成します。
- ホストパスプロビジョナーの **StorageClass** オブジェクトを作成します。

ホストパスプロビジョナー Operator は、カスタムリソースの作成時にプロビジョナーを各ノードに **DaemonSet** としてデプロイします。カスタムリソースファイルでは、ホストパスプロビジョナーが作成する PersistentVolume のバッキングディレクトリーを指定します。

6.14.1.2. Red Hat Enterprise Linux CoreOS 8 でのホストパスプロビジョナー用の SELinux の設定

HostPathProvisioner カスタムリソースを作成する前に、SELinux を設定する必要があります。Red Hat Enterprise Linux CoreOS 8 ワーカーで SELinux を設定するには、各ノードに **MachineConfig** オブジェクトを作成する必要があります。



注記

Red Hat Enterprise Linux CoreOS ワーカーを使用しない場合は、この手順を省略します。

前提条件

- ホストパスプロビジョナーが作成する PersistentVolume (PV) 用に、各ノードにバックギングディレクトリーを作成します。

手順

1. MachineConfig ファイルを作成します。以下は例になります。

```
$ touch machineconfig.yaml
```

2. ファイルを編集し、ホストパスプロビジョナーが PV を作成するディレクトリーを組み込みます。以下に例を示します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  name: 50-set-selinux-for-hostpath-provisioner
  labels:
    machineconfiguration.openshift.io/role: worker
spec:
  config:
    ignition:
      version: 2.2.0
    systemd:
      units:
        - contents: |
            [Unit]
            Description=Set SELinux chcon for hostpath provisioner
            Before=kubelet.service

            [Service]
            ExecStart=/usr/bin/chcon -Rt container_file_t <path/to/backing/directory> 1

            [Install]
            WantedBy=multi-user.target
          enabled: true
          name: hostpath-provisioner.service
```

- 1 プロビジョナーが PV を作成するバックギングディレクトリーを指定します。

3. **MachineConfig** オブジェクトを作成します。

```
$ oc create -f machineconfig.yaml -n <namespace>
```

6.14.1.3. 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有効化

ホストパスプロビジョナーをデプロイし、仮想マシンがローカルストレージを使用できるようにするには、最初に HostPathProvisioner カスタムリソースを作成します。

前提条件

- ホストパスプロビジョナーが作成する PersistentVolume (PV) 用に、各ノードにバックギングディレクトリーを作成します。
- SELinux コンテキスト **container_file_t** を各ノードの PV バックギングディレクトリーに適用します。以下は例になります。

```
$ sudo chcon -t container_file_t -R </path/to/backing/directory>
```



注記

Red Hat Enterprise Linux CoreOS 8 ワーカーを使用する場合は、代わりに MachineConfig マニフェストを使用して SELinux を設定する必要があります。

手順

- HostPathProvisioner カスタムリソースファイルを作成します。以下は例になります。

```
$ touch hostpathprovisioner_cr.yaml
```

- ファイルを編集し、**spec.pathConfig.path** の値がホストパスプロビジョナーが PV を作成するディレクトリーであることを確認します。以下に例を示します。

```
apiVersion: hostpathprovisioner.kubevirt.io/v1alpha1
kind: HostPathProvisioner
metadata:
  name: hostpath-provisioner
spec:
  imagePullPolicy: IfNotPresent
  pathConfig:
    path: "</path/to/backing/directory>" ❶
    useNamingPrefix: "false" ❷
```

- ❶ プロビジョナーが PV を作成するバックギングディレクトリーを指定します。
- ❷ 作成された PV にバインドされる PersistentVolumeClaim (PVC) の名前をディレクトリー名の接頭辞として使用する場合には、この値を **true** に変更します。



注記

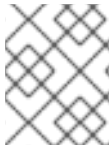
バックギングディレクトリーを作成していない場合、プロビジョナーはこの作成を試行します。**container_file_t** SELinux コンテキストを適用していない場合、これにより **Permission denied** エラーが生じる可能性があります。

3. **openshift-cnv** namespace にカスタムリソースを作成します。

```
$ oc create -f hostpathprovisioner_cr.yaml -n openshift-cnv
```

6.14.1.4. StorageClass オブジェクトの作成

StorageClass オブジェクトの作成時に、ストレージクラスに属する PersistentVolume (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。



注記

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。

手順

1. ストレージクラスを定義する YAML ファイルを作成します。以下に例を示します。

```
$ touch storageclass.yaml
```

2. ファイルを編集します。以下に例を示します。

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: hostpath-provisioner ❶
provisioner: kubevirt.io/hostpath-provisioner
reclaimPolicy: Delete ❷
volumeBindingMode: WaitForFirstConsumer ❸
```

- ❶ この値を変更することで、オプションでストレージクラスの名前を変更できます。
- ❷ **reclaimPolicy** には、**Delete** および **Retain** の2つの値があります。値を指定しない場合、ストレージクラスはデフォルトで **Delete** に設定されます。
- ❸ **volumeBindingMode** 値は、動的プロビジョニングおよびボリュームバインディングが実行されるタイミングを決定します。**WaitForFirstConsumer** を指定して、PersistentVolumeClaim (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. **StorageClass** オブジェクトを作成します。

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

追加情報

- [ストレージクラス](#)

6.14.2. virtctl ツールの使用によるローカルディスクイメージのアップロード

virtctl コマンドラインユーティリティーを使用して、ローカルに保存されたディスクイメージを新規の **DataVolume** にアップロードできます。

6.14.2.1. 前提条件

- **kubevirt-virtctl** パッケージの [インストール](#)
- [CDI でサポートされる操作マトリックス](#) に応じてスクラッチ領域が必要な場合、まずは、この操作が正常に実行されるように [StorageClass](#) を定義するか、または [CDI スクラッチ領域を用意](#) します。

6.14.2.2. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.14.2.3. アップロード DataVolume の作成

ローカルディスクイメージのアップロードに使用する **upload** データソースで DataVolume を手動で作成できます。

手順

1. **spec: source: upload{}** を指定する DataVolume 設定を作成します。

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <upload-datavolume> ❶
spec:
  source:
    upload: {}
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> ❷
```

- ❶ DataVolume の名前。
- ❷ DataVolume のサイズ。この値がアップロードするディスクのサイズ以上であることを確認します。

2. 以下のコマンドを実行して DataVolume を作成します。

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

6.14.2.4. ローカルディスクイメージの DataVolume へのアップロード

virtctl CLI ユーティリティーを使用して、ローカルディスクイメージをクライアントマシンからクラスター内の DataVolume (DV) にアップロードできます。この手順の実行時に、すでにクラスターに存在する DV を使用するか、または新規の DV を作成することができます。



注記

ローカルディスクイメージのアップロード後に、これを仮想マシンに追加できます。

前提条件

- RAW、ISO、または QCOW2 形式の仮想マシンディスクイメージ (オプションで **xz** または **gz** を使用して圧縮される)
- **kubevirt-virtctl** パッケージがクライアントマシンにインストールされていること。
- クライアントマシンが OpenShift Container Platform ルーターの証明書を信頼するように設定されていること。

手順

1. 以下を特定します。
 - 使用するアップロード DataVolume の名前。この DataVolume が存在しない場合、これは自動的に作成されます。
 - DataVolume のサイズ (アップロード手順の実行時に作成する必要がある場合)。サイズはディスクイメージのサイズ以上である必要があります。
 - アップロードする必要がある仮想マシンディスクイメージのファイルの場所。
2. **virtctl image-upload** コマンドを実行してディスクイメージをアップロードします。直前の手順で特定したパラメーターを指定します。以下は例になります。

```
$ virtctl image-upload dv <datavolume_name> \ ❶
--size=<datavolume_size> \ ❷
--image-path=</path/to/image> \ ❸
```

- ❶ DataVolume の名前。
- ❷ DataVolume のサイズ。例: **--size=500Mi**、**--size=1G**
- ❸ 仮想マシンディスクイメージのファイルパス。



注記

- 新規 DataVolume 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。
- HTTPS を使用したセキュアでないサーバー接続を許可するには、**--insecure** パラメーターを使用します。**--insecure** フラグを使用する際に、アップロードエンドポイントの信頼性は検証 **されない** 点に注意してください。

3. オノノヨン。DataVolume 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 DataVolume オブジェクトを表示します。

```
$ oc get dvs
```

6.14.2.5. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubvirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	<input type="checkbox"/> TAR	<input type="checkbox"/> TAR

✓ 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

+ アーカイブはブロックモード DV をサポートしません。

6.14.3. ブロックストレージ DataVolume へのローカルディスクイメージのアップロード

virtctl コマンドラインユーティリティを使用して、ローカルディスクイメージをブロック DataVolume にアップロードできます。

このワークフローでは、ローカルブロックデバイスを使用して PersistentVolume を使用し、このブロックボリュームを **upload** DataVolume に関連付け、**virtctl** を使用してローカルディスクイメージを DataVolume にアップロードできます。

6.14.3.1. 前提条件

- **kubvirt-virtctl** パッケージの [インストール](#)
- [CDI でサポートされる操作マトリックス](#) に応じてスクラッチ領域が必要な場合、まずは、この操作が正常に実行されるように [StorageClass](#) を定義するか、または [CDI スクラッチ領域を用意](#) します。

6.14.3.2. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.14.3.3. ブロック PersistentVolume について

ブロック PersistentVolume (PV) は、raw ブロックデバイスによってサポートされる PV です。これらのボリュームにはファイルシステムがなく、ディスクに直接書き込む仮想マシンや、独自のストレージサービスを実装する仮想マシンにはパフォーマンス上の利点があります。

raw ブロックボリュームは、PV および PersistentVolumeClaim (PVC) 仕様で **volumeMode: Block** を指定してプロビジョニングされます。

6.14.3.4. ローカルブロック PersistentVolume の作成

ファイルにデータを設定し、これをループデバイスとしてマウントすることにより、ノードでローカルブロック PersistentVolume (PV) を作成します。次に、このループデバイスを PV 設定で **Block** ボリュームとして参照し、これを仮想マシンイメージのブロックデバイスとして使用できます。

手順

1. ローカル PV を作成するノードに **root** としてログインします。この手順では、**node01** を例に使用します。
2. ファイルを作成して、これを null 文字で設定し、ブロックデバイスとして使用できるようにします。以下の例では、2Gb (20 100Mb ブロック) のサイズのファイル **loop10** を作成します。

```
$ dd if=/dev/zero of=<loop10> bs=100M count=20
```

3. **loop10** ファイルをループデバイスとしてマウントします。

```
$ losetup </dev/loop10>d3 <loop10> ① ②
```

- ① ループデバイスがマウントされているファイルパスです。
- ② 前の手順で作成したファイルはループデバイスとしてマウントされます。

4. マウントされたループデバイスを参照する **PersistentVolume** 設定を作成します。

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <local-block-pv10>
  annotations:
spec:
  local:
    path: </dev/loop10> ①
  capacity:
    storage: <2Gi>
  volumeMode: Block ②
  storageClassName: local ③
  accessModes:
```

```

- ReadWriteOnce
persistentVolumeReclaimPolicy: Delete
nodeAffinity:
  required:
    nodeSelectorTerms:
      - matchExpressions:
          - key: kubernetes.io/hostname
            operator: In
            values:
              - <node01> ④

```

- ① ノード上のループデバイスのパス。
- ② ブロック PVであることを指定します。
- ③ オプション: PVに StorageClass を設定します。これを省略する場合、クラスターのデフォルトが使用されます。
- ④ ブロックデバイスがマウントされたノード。

5. ブロック PV を作成します。

```
# oc create -f <local-block-pv10.yaml> ①
```

- ① 直前の手順で作成された PersistentVolume のファイル名。

6.14.3.5. アップロード DataVolume の作成

ローカルディスクイメージのアップロードに使用する **upload** データソースで DataVolume を手動で作成できます。

手順

1. **spec: source: upload{}** を指定する DataVolume 設定を作成します。

```

apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <upload-datavolume> ①
spec:
  source:
    upload: {}
  pvc:
    accessModes:
      - ReadWriteOnce
    resources:
      requests:
        storage: <2Gi> ②

```

- ① DataVolume の名前。
- ② DataVolume のサイズ。この値がアップロードするディスクのサイズ以上であることを確認します。

2. 以下のコマンドを実行して DataVolume を作成します。

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

6.14.3.6. ローカルディスクイメージの DataVolume へのアップロード

virtctl CLI ユーティリティを使用して、ローカルディスクイメージをクライアントマシンからクラスター内の DataVolume (DV) にアップロードできます。この手順の実行時に、すでにクラスターに存在する DV を使用するか、または新規の DV を作成することができます。



注記

ローカルディスクイメージのアップロード後に、これを仮想マシンに追加できます。

前提条件

- RAW、ISO、または QCOW2 形式の仮想マシンディスクイメージ (オプションで **xz** または **gz** を使用して圧縮される)
- **kubevirt-virtctl** パッケージがクライアントマシンにインストールされていること。
- クライアントマシンが OpenShift Container Platform ルーターの証明書を信頼するように設定されていること。

手順

1. 以下を特定します。
 - 使用するアップロード DataVolume の名前。この DataVolume が存在しない場合、これは自動的に作成されます。
 - DataVolume のサイズ (アップロード手順の実行時に作成する必要がある場合)。サイズはディスクイメージのサイズ以上である必要があります。
 - アップロードする必要がある仮想マシンディスクイメージのファイルの場所。
2. **virtctl image-upload** コマンドを実行してディスクイメージをアップロードします。直前の手順で特定したパラメーターを指定します。以下は例になります。

```
$ virtctl image-upload dv <datavolume_name> \ ❶
--size=<datavolume_size> \ ❷
--image-path=</path/to/image> \ ❸
```

- ❶ DataVolume の名前。
- ❷ DataVolume のサイズ。例: **--size=500Mi**、**--size=1G**
- ❸ 仮想マシンディスクイメージのファイルパス。



注記

- 新規 DataVolume を作成する必要がある場合は、**--size** パラメーターを省略し、**--no-create** フラグを含めます。
- HTTPS を使用したセキュアでないサーバー接続を許可するには、**--insecure** パラメーターを使用します。**--insecure** フラグを使用する際に、アップロードエンドポイントの信頼性は検証 **されない** 点に注意してください。

3. オプション。DataVolume 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 DataVolume オブジェクトを表示します。

```
$ oc get dvs
```

6.14.3.7. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubvirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	<input type="checkbox"/> TAR	<input type="checkbox"/> TAR

✓ 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

+ アーカイブはブロックモード DV をサポートしません。

6.14.4. ローカル仮想マシンディスクの別のノードへの移動

ローカルボリュームストレージを使用する仮想マシンは、特定のノードで実行されるように移動することができます。

以下の理由により、仮想マシンを特定のノードに移動する場合があります。

- 現在のノードにローカルストレージ設定に関する制限がある。
- 新規ノードがその仮想マシンのワークロードに対して最適化されている。

ローカルストレージを使用する仮想マシンを移行するには、DataVolume を使用して基礎となるボリュームのクローンを作成する必要があります。クローン操作が完了したら、新規 DataVolume を使用できるように [仮想マシン設定を編集](#) するか、または [新規 DataVolume を別の仮想マシンに追加](#) できます。



注記

cluster-admin ロールのないユーザーには、複数の namespace 間でボリュームのクローンを作成できるように [追加のユーザーパーミッション](#) が必要になります。

6.14.4.1. ローカルボリュームの別のノードへのクローン作成

基礎となる PersistentVolumeClaim (PVC) のクローンを作成して、仮想マシンディスクを特定のノードで実行するように移行することができます。

仮想マシンディスクのノードが適切なノードに作成されることを確認するには、新規の PersistentVolume (PV) を作成するか、または該当するノードでそれを特定します。一意のラベルを PV に適用し、これが DataVolume で参照できるようにします。



注記

宛先 PV のサイズはソース PVC と同じか、またはそれよりも大きくなければなりません。宛先 PV がソース PVC よりも小さい場合、クローン作成操作は失敗します。

前提条件

- 仮想マシンが実行されていないこと。仮想マシンディスクのクローンを作成する前に、仮想マシンの電源を切ります。

手順

1. ノードに新規のローカル PV を作成するか、またはノードにすでに存在しているローカル PV を特定します。
 - **nodeAffinity.nodeSelectorTerms** パラメーターを含むローカル PV を作成します。以下のマニフェストは、**node01** に **10Gi** のローカル PV を作成します。

```
kind: PersistentVolume
apiVersion: v1
metadata:
  name: <destination-pv> ❶
  annotations:
spec:
  accessModes:
    - ReadWriteOnce
  capacity:
    storage: 10Gi ❷
  local:
    path: /mnt/local-storage/local/disk1 ❸
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname
```

```

    operator: In
    values:
      - node01 ④
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local
  volumeMode: Filesystem

```

- ① PV の名前。
 - ② PV のサイズ。十分な領域を割り当てる必要があります。そうでない場合には、クローン操作は失敗します。サイズはソース PVC と同じか、またはそれよりも大きくなければなりません。
 - ③ ノードのマウントパス。
 - ④ PV を作成するノードの名前。
- ターゲットノードに存在する PV を特定します。設定の **nodeAffinity** フィールドを確認して、PV がプロビジョニングされるノードを特定することができます。

```
$ oc get pv <destination-pv> -o yaml
```

以下のスニペットは、PV が **node01** にあることを示しています。

```

...
spec:
  nodeAffinity:
    required:
      nodeSelectorTerms:
        - matchExpressions:
            - key: kubernetes.io/hostname ①
              operator: In
              values:
                - node01 ②
...

```

- ① **kubernetes.io/hostname** キーでは、ノードを選択するためにノードホスト名を使用します。
- ② ノードのホスト名です。

2. PV に一意のラベルを追加します。

```
$ oc label pv <destination-pv> node=node01
```

3. 以下を参照する DataVolume マニフェストを作成します。

- 仮想マシンの PVC 名と namespace。
- 直前の手順で PV に適用されたラベル。
- 宛先 PV のサイズ。

```

apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: <clone-datavolume> ❶
spec:
  source:
    pvc:
      name: "<source-vm-disk>" ❷
      namespace: "<source-namespace>" ❸
  pvc:
    accessModes:
      - ReadWriteOnce
    selector:
      matchLabels:
        node: node01 ❹
    resources:
      requests:
        storage: <10Gi> ❺

```

- ❶ 新規 DataVolume の名前。
- ❷ ソース PVC の名前。PVC 名が分からない場合は、仮想マシン設定 **spec.volumes.persistentVolumeClaim.claimName** で確認できます。
- ❸ ソース PVC が存在する namespace。
- ❹ 直前の手順で PV に追加したラベル。
- ❺ 宛先 PV のサイズ。

4. DataVolume マニフェストをクラスターに適用してクローン作成の操作を開始します。

```
$ oc apply -f <clone-datavolume.yaml>
```

DataVolume は、仮想マシンの PVC のクローンを特定のノード上の PV に作成します。

6.14.5. 空のディスクイメージを追加して仮想ストレージを拡張する

空のディスクイメージを Container-native Virtualization に追加することによって、ストレージ容量を拡張したり、新規のデータパーティションを作成したりできます。

6.14.5.1. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.14.5.2. DataVolume を使用した空のディスクイメージの作成

DataVolume 設定ファイルをカスタマイズし、デプロイすることにより、新規の空のディスクイメージを PersistentVolumeClaim に作成することができます。

前提条件

- 1つ以上の利用可能な PersistentVolume
- OpenShift CLI (**oc**) のインストール。

手順

1. DataVolume 設定ファイルを編集します。

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  pvc:
    # Optional: Set the storage class or omit to accept the default
    # storageClassName: "hostpath"
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

2. 以下のコマンドを実行して、空のディスクイメージを作成します。

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

6.14.5.3. テンプレート: 空のディスクイメージの DataVolume 設定ファイル

blank-image-datavolume.yaml

```
apiVersion: cdi.kubevirt.io/v1alpha1
kind: DataVolume
metadata:
  name: blank-image-datavolume
spec:
  source:
    blank: {}
  pvc:
    # Optional: Set the storage class or omit to accept the default
    # storageClassName: "hostpath"
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 500Mi
```

6.14.6. DataVolume のストレージのデフォルト

kubevirt-storage-class-defaults ConfigMap は DataVolume の **アクセスモード** および **ボリュームモード** のデフォルト設定を提供します。Web コンソールで、基礎となるストレージにより適した

DataVolume を作成するために、ConfigMap に対してストレージクラスのデフォルトを編集したり、追加したりできます。

6.14.6.1. DataVolume のストレージ設定について

DataVolume では、定義された **アクセスモード** と **ボリュームモード** を Web コンソールで作成する必要があります。これらのストレージ設定は、デフォルトで **ReadWriteOnce** アクセスモードおよび **Filesystem** ボリュームモードで設定されます。

これらの設定は、**openshift-cnv** namespace で **kubevirt-storage-class-defaults** ConfigMap を編集して変更できます。また、異なるストレージタイプについて Web コンソールで DataVolume を作成するために、他のストレージクラスの設定を追加することもできます。



注記

基礎となるストレージでサポートされるストレージ設定を設定する必要があります。

Web コンソールで作成するすべての DataVolume は、ConfigMap にも定義されるストレージクラスを指定しない限り、デフォルトのストレージ設定を使用します。

6.14.6.1.1. アクセスモード

DataVolume は以下のアクセスモードをサポートします。

- **ReadWriteOnce**: ボリュームは単一ノードで、read-write としてマウントできます。**ReadWriteOnce** にはより多様性があり、これはデフォルトの設定です。
- **ReadWriteMany**: ボリュームは数多くのノードで、read-write としてマウントできます。**ReadWriteMany** は、ノード間の仮想マシンのライブマイグレーションなどの、一部の機能で必要になります。

ReadWriteMany は、基礎となるストレージがこれをサポートしている場合に推奨されます。

6.14.6.1.2. ボリュームモード

ボリュームモードは、ボリュームをフォーマットされたファイルシステムで使用するか、または raw ブロック状態のままにするかを定義します。DataVolume は以下のボリュームモードをサポートします。

- **Filesystem**: DataVolume にファイルシステムを作成します。これはデフォルトの設定です。
- **Block**: ブロック DataVolume を作成します。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。

6.14.6.2. Web コンソールでの kubevirt-storage-class-defaults ConfigMap の編集

DataVolume のストレージ設定を、**openshift-cnv** namespace の **kubevirt-storage-class-defaults** ConfigMap を編集して変更します。また、異なるストレージタイプについて Web コンソールで DataVolume を作成するために、他のストレージクラスの設定を追加することもできます。



注記

基礎となるストレージでサポートされるストレージ設定を設定する必要があります。

手順

1. サイドメニューから、**Workloads** → **Config Maps** をクリックします。
2. **Project** 一覧で、**openshift-cnv** を選択します。
3. **kubevirt-storage-class-defaults** をクリックして **Config Map Overview** を開きます。
4. **YAML** タブをクリックして編集可能な設定を表示します。
5. 基礎となるストレージに適したストレージ設定で **data** の値を更新します。

```
...
data:
  accessMode: ReadWriteOnce ❶
  volumeMode: Filesystem ❷
  <new>.accessMode: ReadWriteMany ❸
  <new>.volumeMode: Block ❹
```

- ❶ デフォルトの accessMode は **ReadWriteOnce** です。
- ❷ デフォルトの volumeMode は **Filesystem** です。
- ❸ ストレージクラスのアクセスモードを追加する場合は、パラメーターの **<new>** 部分をストレージクラス名に置き換えます。
- ❹ ストレージクラスのボリュームモードを追加する場合は、パラメーターの **<new>** 部分をストレージクラス名に置き換えます。

6. **Save** をクリックして ConfigMap を更新します。

6.14.6.3. CLI での kubevirt-storage-class-defaults ConfigMap の編集

DataVolume のストレージ設定を、**openshift-cnv** namespace の **kubevirt-storage-class-defaults** ConfigMap を編集して変更します。また、異なるストレージタイプについて Web コンソールで DataVolume を作成するために、他のストレージクラスの設定を追加することもできます。



注記

基礎となるストレージでサポートされるストレージ設定を設定する必要があります。

手順

1. **oc edit** を使用して ConfigMap を編集します。

```
$ oc edit configmap kubevirt-storage-class-defaults -n openshift-cnv
```

2. ConfigMap の **data** 値を更新します。

```
...
data:
  accessMode: ReadWriteOnce ❶
  volumeMode: Filesystem ❷
  <new>.accessMode: ReadWriteMany ❸
  <new>.volumeMode: Block ❹
```


- ① デフォルトの `accessMode` は **ReadWriteOnce** です。
- ② デフォルトの `volumeMode` は **Filesystem** です。
- ③ ストレージクラスのアクセスモードを追加する場合は、パラメーターの **<new>** 部分をストレージクラス名に置き換えます。
- ④ ストレージクラスのボリュームモードを追加する場合は、パラメーターの **<new>** 部分をストレージクラス名に置き換えます。

3. エディターを保存し、終了して ConfigMap を更新します。

6.14.6.4. 複数のストレージクラスのデフォルト例

以下の YAML ファイルは、**kubevirt-storage-class-defaults** ConfigMap の例です。これには、**migration** および **block** の2つのストレージクラスについてストレージが設定されます。

ConfigMap を更新する前に、すべての設定が基礎となるストレージでサポートされていることを確認してください。

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: kubevirt-storage-class-defaults
  namespace: openshift-cnv
...
data:
  accessMode: ReadWriteOnce
  volumeMode: Filesystem
  nfs-sc.accessMode: ReadWriteMany
  nfs-sc.volumeMode: Filesystem
  block-sc.accessMode: ReadWriteMany
  block-sc.volumeMode: Block
```

6.14.7. CDI のスクラッチ領域の用意

6.14.7.1. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.14.7.2. スクラッチ領域について

Containerized Data Importer (CDI) では、仮想マシンイメージのインポートやアップロードなどの、一部の操作を実行するためにスクラッチ領域 (一時ストレージ) が必要になります。このプロセスで、CDI は、宛先 DataVolume (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。

CDIConfig オブジェクトにより、**scratchSpaceStorageClass** を CDIConfig オブジェクトの **spec**: セクションに指定して、スクラッチ領域 PVC をバインドするために使用する StorageClass を定義することができます。

定義された StorageClass がクラスターの StorageClass に一致しない場合、クラスターに定義されたデフォルト StorageClass が使用されます。クラスターで定義されたデフォルトの StorageClass 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 StorageClass が使用されます。



注記

CDI では、元の DataVolume 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 StorageClass 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。

6.14.7.3. スクラッチ領域を必要とする CDI 操作

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

6.14.7.4. CDI 設定での StorageClass の定義

CDI 設定で StorageClass を定義し、CDI 操作のスクラッチ領域を動的にプロビジョニングします。

手順

- **oc** クライアントを使用して **cdiconfig/config** を編集し、**spec: scratchSpaceStorageClass** を追加または編集してクラスターの StorageClass に一致させます。

```
$ oc edit cdiconfig/config
```

```
API Version: cdi.kubevirt.io/v1alpha1
kind: CDIConfig
metadata:
  name: config
...
spec:
  scratchSpaceStorageClass: "<storage_class>"
...
```

6.14.7.5. CDI がサポートする操作マトリックス

このマトリックスにはエンドポイントに対してコンテンツタイプのサポートされる CDI 操作が表示されます。これらの操作にはスクラッチ領域が必要です。

コンテンツタイプ	HTTP	HTTPS	HTTP Basic 認証	レジストリー	アップロード
kubevirt (QCOW2)	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2** ✓ GZ* ✓ XZ*	✓ QCOW2 ✓ GZ* ✓ XZ*	✓ QCOW2* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ QCOW2* ✓ GZ* ✓ XZ*
KubeVirt (RAW)	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW ✓ GZ ✓ XZ	✓ RAW* <input type="checkbox"/> GZ <input type="checkbox"/> XZ	✓ RAW* ✓ GZ* ✓ XZ*
Archive+	✓ TAR	✓ TAR	✓ TAR	<input type="checkbox"/> TAR	<input type="checkbox"/> TAR

✓ 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

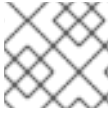
+ アーカイブはブロックモード DV をサポートしません。

追加リソース

- StorageClass およびこれらがクラスターで定義される方法についての詳細は、[Dynamic provisioning](#) セクションを参照してください。

6.14.8. DataVolume の削除

oc コマンドラインインターフェイスを使用して、DataVolume を手動で削除できます。

**注記**

仮想マシンを削除する際に、これが使用する DataVolume は自動的に削除されます。

6.14.8.1. DataVolume について

DataVolume オブジェクトは、Containerized Data Importer (CDI) プロジェクトで提供されるカスタムリソースです。DataVolume は、基礎となる PersistentVolumeClaim (PVC) に関連付けられるインポート、クローン作成、およびアップロード操作のオーケストレーションを行います。DataVolume は KubeVirt に統合され、仮想マシンが PVC の作成前に起動することを防ぎます。

6.14.8.2. すべての DataVolume の一覧表示

oc コマンドラインインターフェイスを使用して、クラスターの DataVolume を一覧表示できます。

手順

- 以下のコマンドを実行して、DataVolume を一覧表示します。

```
$ oc get dvs
```

6.14.8.3. DataVolume 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 DataVolume を削除できます。

前提条件

- 削除する必要がある DataVolume の名前を特定します。

手順

- 以下のコマンドを実行し、DataVolume を削除します。

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

**注記**

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

第7章 仮想マシンテンプレート


7.1. 仮想マシンテンプレートの作成

仮想マシンテンプレートの使用は、同様の設定を持つ複数の仮想マシンを作成するための簡単な方法です。テンプレートの作成後、仮想マシンの作成時にテンプレートを参照できます。

7.1.1. Web コンソールでのインタラクティブなウィザードを使用した仮想マシンテンプレートの作成

Web コンソールは、**General**、**Networking**、**Storage**、**Advanced**、および **Review** ステップに移動し、仮想マシンテンプレートの作成プロセスを単純化するインタラクティブなウィザードを特長としています。すべての必須フィールドには * のマークが付けられます。ウィザードは必須フィールドへの値の入力が完了するまで次のステップに移動することを防ぎます。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machine Templates** をクリックします。
2. **Create Template** をクリックし、**New with Wizard** を選択します。
3. **General** ステップで必要なすべてのフィールドに入力します。
4. **Next** をクリックして **Networking** 画面に進みます。デフォルトで **nic0** という名前の NIC が割り当てられます。
 - a. オプション: **Add Network Interface** をクリックし、追加の NIC を作成します。
 - b. オプション: すべての NIC の削除は、Options メニュー  をクリックし、**Delete** を選択して実行できます。テンプレートから作成される仮想マシンには、割り当てられている NIC は不要です。NIC は仮想マシンの作成後に作成できます。
5. **Next** をクリックして **Storage** 画面に進みます。
 - a. オプション: **Add Disk** をクリックして追加のディスクを作成します。
 - b. オプション: ディスクをクリックして選択可能なフィールドを変更します。✓ ボタンをクリックして変更を保存します。
 - c. オプション: **Disk** をクリックし、**Select Storage** の一覧から利用可能なディスクを選択します。



注記

URL または **Container** のいずれかが **General** ステップで **Source** として選択される場合、**rootdisk** ディスクが作成され、**Bootable Disk** として仮想マシンに割り当てられます。**rootdisk** を変更できますが、これを削除することはできません。

Bootable Disk は、仮想マシンにディスクが割り当てられていない場合、**PXE** ソースからプロビジョニングされる仮想マシンには不要です。1つ以上のディスクが仮想マシンに割り当てられている場合、**Bootable Disk** を1つを選択する必要があります。

6. **Create Virtual Machine Template** をクリックします。**Results** 画面には、仮想マシンテンプレートの JSON 設定ファイルが表示されます。

テンプレートは **Workloads** → **Virtual Machine Templates** に一覧表示されます。

7.1.2. 仮想マシンテンプレートのインタラクティブなウィザードのフィールド

以下の表は、**Create Virtual Machine Template** のインタラクティブなウィザードの **Basic Settings**、**Networking**、および **Storage** ペインのフィールドを説明しています。

7.1.2.1. 仮想マシンテンプレートウィザードのフィールド

名前	パラメーター	説明
ソース	PXE	PXE メニューから仮想マシンをプロビジョニングします。クラスターに PXE 対応の NIC が必要になります。
	URL	HTTP または S3 エンドポイントで利用できるイメージから仮想マシンをプロビジョニングします。
	コンテナ	クラスターからアクセスできるレジストリーの起動可能なオペレーティングシステムコンテナから仮想マシンをプロビジョニングします。例: kubevirt/cirros-registry-disk-demo
	Disk	ディスクから仮想マシンをプロビジョニングします。
Operating System		仮想マシン用に選択される主なオペレーティングシステム。
Flavor	small、medium、large、tiny、Custom	仮想マシンに割り当てられる CPU およびメモリーの量を決定するプリセット。 Flavor に設定される Preset はオペレーティングシステムによって決まります。

名前	パラメーター	説明
Memory		仮想マシンに割り当てられるメモリーのサイズ (GiB 単位)。
CPU		仮想マシンに割り当てられる CPU の量。
Workload Profile	High Performance	高パフォーマンスのワークロードに対して最適化された仮想マシン設定。
	Server	サーバーワークロードの実行に最適化されたプロファイル。
	Desktop	デスクトップで使用するための仮想マシン設定。
名前		この名前には、小文字 (a-z)、数字 (0-9)、およびハイフン (-) を含めることができ、最大 253 文字を使用できます。最初と最後の文字は英数字にする必要があります。この名前には、大文字、スペース、ピリオド (.)、または特殊文字を使用できません。
説明		オプションの説明フィールド。

7.1.2.2. Cloud-init フィールド

名前	説明
Hostname	仮想マシンの特定のホスト名を設定します。
Authenticated SSH Keys	仮想マシンの <code>~/.ssh/authorized_keys</code> にコピーされるユーザーの公開鍵。
Use custom script	他のオプションを、カスタム cloud-init スクリプトを貼り付けるフィールドに置き換えます。

7.1.2.3. ネットワークフィールド

名前	説明
名前	ネットワークインターフェイスの名前。

名前	説明
Model	ネットワークインターフェイスカードのモデルを示します。サポートされる値は、 e1000 、 e1000e 、 ne2k_pci 、 pcnet 、 rtl8139 、および virtIO です。
Network	利用可能な NetworkAttachmentDefinition オブジェクトの一覧。
Type	利用可能なバインディングメソッドの一覧。デフォルトの Pod ネットワークについては、 masquerade が唯一の推奨されるバインディングメソッドになります。セカンダリーネットワークの場合は、 bridge バインディングメソッドを使用します。 masquerade メソッドは、デフォルト以外のネットワークではサポートされません。
MAC Address	ネットワークインターフェイスの MAC アドレス。MAC アドレスが指定されていない場合、セッションの一時アドレスが生成されます。

7.1.2.4. ストレージフィールド

名前	説明
ソース	仮想マシンの空のディスクを選択するか、または URL 、 Container 、 Attach Cloned Disk 、または Attach Disk などの選択可能なオプションから選択します。既存ディスクを選択し、これを仮想マシンに割り当てるには、利用可能な PersistentVolumeClaim (PVC) の一覧から Attach Cloned Disk または Attach Disk を選択します。
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 (GiB)	ディスクのサイズ (GiB)。
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 StorageClass 。

7.2. 仮想マシンテンプレートの編集

仮想マシンテンプレートは、Web コンソールで YAML エディターの詳細設定を編集するか、または **Virtual Machine Template Overview** 画面のパラメーターのサブセットを編集して更新することができます。

7.2.1. Web コンソールでの仮想マシンテンプレートの編集

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

手順

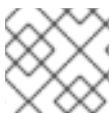
1. サイドメニューから **Workloads** → **Virtual Machine Templates** をクリックします。
2. 仮想マシンテンプレートを選択して、**Virtual Machine Template Overview** 画面を開きます。
3. 鉛筆アイコンをクリックして、該当するフィールドを編集可能にします。
4. 関連する変更を加え、**Save** をクリックします。

仮想マシンテンプレートの編集は、そのテンプレートですでに作成された仮想マシンには影響を与えません。

7.2.2. Web コンソールでの仮想マシンテンプレート YAML 設定の編集

Web コンソールで仮想マシンテンプレートの YAML 設定を編集できます。

すべてのパラメーターが変更可能である訳ではありません。無効な設定で **Save** をクリックすると、エラーメッセージで変更できないパラメーターが示唆されます。



注記

編集中に YAML 画面から離れると、設定に対して加えた変更が取り消されます。

手順

1. Container-native Virtualization コンソールで **Workloads** → **Virtual Machine Templates** をクリックします。
2. テンプレートを選択します。
3. **YAML** タブをクリックして編集可能な設定を表示します。
4. ファイルを編集し、**Save** をクリックします。

オブジェクトの更新されたバージョン番号を含む、変更が正常に行われたことを示す確認メッセージが表示されます。

7.2.3. 仮想マシンテンプレートへの仮想ディスクの追加

以下の手順を使用して、仮想ディスクを仮想マシンテンプレートに追加します。

手順

1. **Virtual Machine Templates** タブで、仮想マシンテンプレートを選択します。
2. **Disks** タブを選択します。
3. **Add Disks** をクリックして、**Add Disk** ウィンドウを開きます。
4. **Add Disk** ウィンドウで、**Source**、**Name**、**Size**、**Interface**、および **Storage Class** を指定します。
5. ドロップダウンリストおよびチェックボックスを使用して、ディスク設定を編集します。
6. **OK** をクリックします。

7.2.4. ネットワークインターフェイスの仮想マシンテンプレートへの追加

以下の手順を使用して、ネットワークインターフェイスを仮想マシンテンプレートに追加します。

手順

1. **Virtual Machine Templates** タブで、仮想マシンテンプレートを選択します。
2. **Network Interfaces** タブを選択します。
3. **Add Network Interface** をクリックします。
4. **Add Network Interface** ウィンドウで、ネットワークインターフェイスの **Name**、**Model**、**Network**、**Type**、および **MAC Address** を指定します。
5. **Add** をクリックしてネットワークインターフェイスを追加します。
6. 仮想マシンを再起動して、アクセスを有効にします。
7. ドロップダウンリストとチェックボックスを編集して、ネットワークインターフェイスを設定します。
8. **Save Changes** をクリックします。
9. **OK** をクリックします。

新規のネットワークインターフェイスは、ユーザーが再起動するまで **Create Network Interface** 一覧の上部に表示されます。

新規のネットワークインターフェイスには、仮想マシンを再起動するまで **Pending VM restart** のリンク状態が表示されます。Link State (リンク状態) にカーソルを合わせて詳細情報を表示します。

ネットワークインターフェイスカードが仮想マシンで定義され、ネットワークに接続されている場合は、**Link State** はデフォルトで **Up** に設定されます。

7.2.5. 仮想マシンテンプレートの CD-ROM の編集

以下の手順を使用して、仮想マシンの CD-ROM を設定します。

手順

1. **Virtual Machine Templates** タブで、仮想マシンテンプレートを選択します。
2. **Overview** タブを選択します。
3. CD-ROM 設定を追加または編集するには、**CD-ROMs** ラベルの右側にある鉛筆アイコンをクリックします。**Edit CD-ROM** ウィンドウが開きます。
 - CD-ROM が編集できない場合、以下のメッセージが表示されます: **The virtual machine doesn't have any CD-ROMs attached.**
 - 利用可能な CD-ROM がある場合は、- をクリックして CD-ROM を削除できます。
4. **Edit CD-ROM** ウィンドウで、以下を実行します。
 - a. **Media Type** のドロップダウンリストから CD-ROM 設定のタイプを選択します。CD-ROM 設定タイプは **Container**、**URL**、および **Persistent Volume Claim** です。
 - b. それぞれの **Type** に必要な情報を入力します。
 - c. すべての CD-ROM が追加されたら、**Save** をクリックします。

7.3. 仮想マシンテンプレートの専用リソースの有効化

パフォーマンスを向上させるために、仮想マシンには CPU などのノードの専用リソースを持たせることができます。

7.3.1. 専用リソースについて

仮想マシンの専用リソースを有効にする場合、仮想マシンのワークロードは他のプロセスで使用されない CPU でスケジュールされます。専用リソースを使用することで、仮想マシンのパフォーマンスとレイテンシーの予測の精度を向上させることができます。

7.3.2. 前提条件

- **CPU マネージャー** はノードに設定される必要があります。仮想マシンのワークロードをスケジュールする前に、ノードに **cpumanager = true** ラベルが設定されていることを確認します。

7.3.3. 仮想マシンテンプレートの専用リソースの有効化

Web コンソールの **Virtual Machine Template Overview** ページで仮想マシンテンプレートの専用リソースを有効にすることができます。

手順

1. サイドメニューから **Workloads** → **Virtual Machine Templates** をクリックします。
2. 仮想マシンテンプレートを選択して、**Virtual Machine Template Overview** ページを開きます。
3. **Details** タブをクリックします。
4. **Dedicated Resources** フィールドの右側にある鉛筆アイコンをクリックして、**Dedicated Resources** ウィンドウを開きます。
5. **Schedule this workload with dedicated resources (guaranteed policy)** を選択します。

6. **Save** をクリックします。


7.4. 仮想マシンテンプレートの削除

Web コンソールで仮想マシンテンプレートを削除できます。

7.4.1. Web コンソールでの仮想マシンテンプレートの削除

仮想マシンテンプレートを削除すると、仮想マシンはクラスターから永続的に削除されます。

手順

1. Container-native Virtualization コンソールで **Workloads** → **Virtual Machine Templates** をクリックします。
2. このペインから仮想マシンテンプレートを削除できます。これにより、1つのペインで複数のマシンに対してアクションを実行することがより容易になります。または、**Virtual Machine Template Details** ペインから仮想マシンテンプレートを削除することもできます。この場合、選択されたテンプレートの総合的な詳細情報を確認できます。
 - 削除するテンプレートの Options メニュー  をクリックし、**Delete Template** を選択します。
 - テンプレート名をクリックして **Virtual Machine Template Details** ペインを開き、**Actions** → **Delete Template** をクリックします。
3. 確認のポップアップウィンドウで、**Delete** をクリックし、テンプレートを永続的に削除します。

第8章 ライブマイグレーション

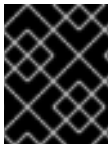
8.1. 仮想マシンのライブマイグレーション

8.1.1. 前提条件

- LiveMigration を使用する前に、仮想マシンで使用されるストレージクラスに、共有 ReadWriteMany (RWX) アクセスモードを持つ PersistentVolumeClaim (PVC) があることを確認します。[DataVolume のストレージのデフォルト](#) について参照し、ストレージ設定が正しいことを確認します。

8.1.2. ライブマイグレーションについて

ライブマイグレーションは、仮想ワークロードまたはアクセスに支障を与えることなく、実行中の仮想マシンインスタンスをクラスター内の別のノードに移行するプロセスです。これは、他のノードに移行する仮想マシンインスタンスを選択する場合は手動プロセスで実行でき、仮想マシンインスタンスに **LiveMigrate** エビクションストラテジーがあり、これが実行されるノードがメンテナース状態の場合には自動プロセスで実行できます。



重要

仮想マシンでは、共有 ReadWriteMany (RWX) アクセスモードを持つ PersistentVolumeClaim (PVC) のライブマイグレーションが必要です。

8.1.3. LiveMigration のアクセスモードの更新

LiveMigration が適切に機能するには、ReadWriteMany (RWX) アクセスモードを使用する必要があります。必要に応じて、以下の手順でアクセスモードを更新します。

手順

- RWX アクセスモードを設定するには、以下の **oc patch** コマンドを実行します。

```
$ oc patch -n openshift-cnv \
  cm kubevirt-storage-class-defaults \
  -p '{"data":{"$<STORAGE_CLASS>'.accessMode':"ReadWriteMany"}}'
```

関連情報:

- [仮想マシンインスタンスの別のノードへの移行](#)
- [ノードのメンテナースモード](#)
- [ライブマイグレーションの制限](#)

8.2. ライブマイグレーションの制限およびタイムアウト

ライブマイグレーションの制限およびタイムアウトは、移行プロセスがクラスターに負担をかけないようにするために適用されます。**kubevirt-config** 設定ファイルを編集してこれらの設定を行います。

8.2.1. ライブマイグレーションの制限およびタイムアウトの設定

更新された key:value フィールドを **openshift-cnv** namespace にある **kubevirt-config** 設定ファイルに追加することによって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

- **kubevirt-config** 設定ファイルを編集し、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。以下の例は、デフォルト値を示しています。

```
$ oc edit configmap kubevirt-config -n openshift-cnv

apiVersion: v1
kind: ConfigMap
metadata:
  name: kubevirt-config
  namespace: kubevirt
  labels:
    kubevirt.io: ""
data:
  feature-gates: "LiveMigration"
  migrations: |-
    parallelMigrationsPerCluster: 5
    parallelOutboundMigrationsPerNode: 2
    bandwidthPerMigration: 64Mi
    completionTimeoutPerGiB: 800
    progressTimeout: 150
```

8.2.2. クラスター全体のライブマイグレーションの制限およびタイムアウト

表8.1 移行パラメーター

パラメーター	説明	デフォルト
parallelMigrationsPerCluster	クラスターで並行して実行される移行の数。	5
parallelOutboundMigrationsPerNode	ノードごとのアウトバウンドの移行の最大数。	2
bandwidthPerMigration	それぞれの移行の帯域幅 (MiB/s)。	64Mi

パラメーター	説明	デフォルト
completionTimeoutPerGiB	マイグレーションが(メモリー GiB あたりの秒単位の) この時間内に終了しなかった場合、マイグレーションは取り消されます。たとえば、6GiB メモリーを持つ仮想マシンインスタンスは、4800 秒内にマイグレーションを完了しない場合にタイムアウトします。 Migration Method が BlockMigration の場合、移行するディスクのサイズは計算に含まれます。	800
progressTimeout	マイグレーションは、メモリーコピーの進展が(秒単位の) この時間内に見られない場合に取り消されます。	150

8.3. 仮想マシンインスタンスの別のノードへの移行

Web コンソールまたは CLI のいずれかで仮想マシンインスタンスのライブマイグレーションを手動で開始します。

8.3.1. Web コンソールでの仮想マシンインスタンスのライブマイグレーションの開始


実行中の仮想マシンインスタンスをクラスター内の別のノードに移行します。



注記

Migrate Virtual Machine アクションはすべてのユーザーに対して表示されますが、仮想マシンの移行を開始できるのは管理者ユーザーのみとなります。

手順

1. Container-native Virtualization コンソールで **Workloads** → **Virtual Machines** をクリックします。
2. この画面からマイグレーションを開始できます。これにより、1つの画面で複数のマシンに対してアクションを実行することがより容易になります。または、**Virtual Machine Details** 画面から仮想マシンを停止することもできます。この場合、選択された仮想マシンの総合的な詳細情報を確認できます。
 - 仮想マシンの末尾の Options メニュー  をクリックし、**Migrate Virtual Machine** を選択します。
 - 仮想マシン名をクリックし、**Virtual Machine Details** 画面を開き、**Actions** → **Migrate Virtual Machine** をクリックします。
3. **Migrate** をクリックして、仮想マシンを別のノードに移行します。

8.3.2. CLI での仮想マシンインスタンスのライブマイグレーションの開始

クラスターに **VirtualMachineInstanceMigration** オブジェクトを作成し、仮想マシンインスタンスの名前を参照して、実行中の仮想マシンインスタンスのライブマイグレーションを開始します。

手順

1. 移行する仮想マシンインスタンスの **VirtualMachineInstanceMigration** 設定ファイルを作成します。 **vmi-migrate.yaml** はこの例になります。

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstanceMigration
metadata:
  name: migration-job
spec:
  vmiName: vmi-fedora
```

2. クラスターにオブジェクトを作成します。

```
$ oc create -f vmi-migrate.yaml
```

VirtualMachineInstanceMigration オブジェクトは、仮想マシンインスタンスのライブマイグレーションをトリガーします。このオブジェクトは、手動で削除されない場合、仮想マシンインスタンスが実行中である限りクラスターに存在します。

関連情報:

- [仮想マシンインスタンスのライブマイグレーションのモニター](#)
- [仮想マシンインスタンスのライブマイグレーションの取り消し](#)

8.4. 仮想マシンインスタンスのライブマイグレーションのモニター

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

8.4.1. Web コンソールでの仮想マシンインスタンスのライブマイグレーションのモニター

移行期間中、仮想マシンのステータスは **Migrating** になります。このステータスは **Virtual Machines** 一覧に表示されるか、または移行中の仮想マシンの **Virtual Machine Details** 画面に表示されます。

手順

- Container-native Virtualization コンソールで **Workloads** → **Virtual Machines** をクリックします。

8.4.2. CLI での仮想マシンインスタンスのライブマイグレーションのモニター

仮想マシンの移行のステータスは、**VirtualMachineInstance** 設定の **Status** コンポーネントに保存されます。

手順

- 移行中の仮想マシンインスタンスで **oc describe** コマンドを使用します。

```
$ oc describe vmi vmi-fedora

...
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
```


8.5. 仮想マシンインスタンスのライブマイグレーションの取り消し

仮想マシンインスタンスを元のノードに残すためにライブマイグレーションを取り消すことができます。

Web コンソールまたは CLI のいずれかでライブマイグレーションを取り消します。


8.5.1. Web コンソールでの仮想マシンインスタンスのライブマイグレーションの取り消し

仮想マシンインスタンスのライブマイグレーションは、**Workloads → Virtual Machines** 画面の各仮想

マシンにある Options メニュー  を使用するか、または **Virtual Machine Details** 画面の **Actions** メニューからキャンセルできます。

手順

1. Container-native Virtualization コンソールで **Workloads → Virtual Machines** をクリックします。
2. この画面から移行を取り消すことができます。これにより、1つの画面で複数のマシンに対してアクションを実行することがより容易になります。または、**Virtual Machine Details** 画面から仮想マシンを停止することもできます。この場合、選択された仮想マシンの総合的な詳細情報を確認できます。

- 仮想マシンの末尾の Options メニュー  をクリックし、**Cancel Virtual Machine Migration** を選択します。
- 仮想マシン名をクリックして **Virtual Machine Details** 画面を開き、**Actions → Cancel Virtual Machine Migration** をクリックします。

3. **Cancel Migration** をクリック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。

8.5.2. CLI での仮想マシンインスタンスのライブマイグレーションの取り消し

移行に関連付けられた **VirtualMachineInstanceMigration** オブジェクトを削除して、仮想マシンインスタンスのライブマイグレーションを取り消します。

手順

- ライブマイグレーションをトリガーした **VirtualMachineInstanceMigration** オブジェクトを削除します。この例では、**migration-job** が使用されています。

```
$ oc delete vmim migration-job
```

8.6. 仮想マシンのエビクションストラテジーの設定

LiveMigrate エビクションストラテジーは、ノードがメンテナース状態になるか、ドレイン (解放) される場合に仮想マシンインスタンスが中断されないようにします。このエビクションストラテジーを持つ仮想マシンインスタンスのライブマイグレーションが別のノードに対して行われます。

8.6.1. LiveMigration エビクションストラテジーでのカスタム仮想マシンの設定

LiveMigration エビクションストラテジーはカスタム仮想マシンでのみ設定する必要があります。共通テンプレートには、デフォルトでこのエビクションストラテジーが設定されています。

手順

1. **evictionStrategy: LiveMigrate** オプションを、仮想マシン設定ファイルの **spec** セクションに追加します。この例では、**oc edit** を使用して **VirtualMachine** 設定ファイルの関連するスニペットを更新します。

```
$ oc edit vm <custom-vm> -n <my-namespace>
```

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachine
metadata:
  name: custom-vm
spec:
  terminationGracePeriodSeconds: 30
  evictionStrategy: LiveMigrate
  domain:
    resources:
      requests:
    ...
```

2. 更新を有効にするために仮想マシンを再起動します。

```
$ virtctl restart <custom-vm> -n <my-namespace>
```

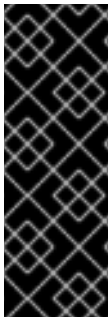
第9章 ノードのメンテナンス

9.1. TLS 証明書の手動更新

Container-native Virtualization コンポーネントの TLS 証明書はインストール時に作成され、1 年間有効になります。これらの証明書は期限切れになる前に手動で更新する必要があります。

9.1.1. TLS 証明書の更新

Container-native Virtualization の TLS 証明書を更新するには、**rotate-certs** スクリプトをダウンロードし、実行します。このスクリプトは、GitHub の **kubevirt/hyperconverged-cluster-operator** リポジトリから入手できます。



重要

証明書を更新すると、以下の操作の影響が確認されます。

- 移行はキャンセルされます
- イメージのアップロードはキャンセルされます
- VNC およびコンソールの接続が閉じられる。

前提条件

- **cluster-admin** 権限を持つユーザーとしてクラスターにログインしていることを確認します。このスクリプトは、アクティブセッションをクラスターに対して使用し、**openshift-cnv** namespace の証明書を更新します。

手順

1. GitHub から **rotate-certs.sh** スクリプトをダウンロードします。

```
$ curl -O https://raw.githubusercontent.com/kubevirt/hyperconverged-cluster-operator/master/tools/rotate-certs.sh
```

2. スクリプトが実行可能であることを確認します。

```
$ chmod +x rotate-certs.sh
```

3. スクリプトを実行します。

```
$ ./rotate-certs.sh -n openshift-cnv
```

TLS 証明書は更新され、1 年間有効になります。

9.2. ノードのメンテナンスモード

9.2.1. ノードのメンテナンスモードについて

ノードがメンテナンス状態になると、ノードにはスケジュール対象外のマークが付けられ、ノードからすべての仮想マシンおよび Pod がドレイン (解放) されます。**LiveMigrate** エビクションストラテ

ジーを持つ仮想マシンインスタンスのライブマイグレーションは、サービスが失われることなく実行されます。このエビクションストラテジーはデフォルトで共通テンプレートから作成される仮想マシンで設定されますが、カスタム仮想マシンの場合には手動で設定される必要があります。

エビクションストラテジーのない仮想マシンインスタンスは、ノードで削除され、別のノードで再作成されます。



重要

仮想マシンでは、共有 ReadWriteMany (RWX) アクセスモードを持つ PersistentVolumeClaim (PVC) のライブマイグレーションが必要です。

追加リソース:

- [仮想マシンのライブマイグレーション](#)
- [仮想マシンのエビクションストラテジーの設定](#)

9.3. ノードのメンテナンスマードへの設定

9.3.1. ノードのメンテナンスマードについて

ノードがメンテナンスマードになると、ノードにはスケジュール対象外のマークが付けられ、ノードからすべての仮想マシンおよび Pod がドレイン (解放) されます。**LiveMigrate** エビクションストラテジーを持つ仮想マシンインスタンスのライブマイグレーションは、サービスが失われることなく実行されます。このエビクションストラテジーはデフォルトで共通テンプレートから作成される仮想マシンで設定されますが、カスタム仮想マシンの場合には手動で設定される必要があります。

エビクションストラテジーのない仮想マシンインスタンスは、ノードで削除され、別のノードで再作成されます。



重要

仮想マシンでは、共有 ReadWriteMany (RWX) アクセスモードを持つ PersistentVolumeClaim (PVC) のライブマイグレーションが必要です。


Web コンソールまたは CLI のいずれかでノードをメンテナンスマードにします。

9.3.2. Web コンソールでのノードのメンテナンスマードへの設定

Compute → Nodes 一覧で各ノードにある Options メニュー  を使用するか、または Node Details 画面の Actions コントロールを使用してノードをメンテナンスマードに設定します。

手順

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

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

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

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

9.3.3. CLI でのノードのメンテナンスモードへの設定

ノード名、およびこれをメンテナンスモードに設定する理由を参照する **NodeMaintenance** カスタムリソース (CR) オブジェクトを作成し、ノードをメンテナンスモードに設定します。

手順

1. ノードメンテナンス CR 設定を作成します。この例では、**node02-maintenance.yaml** という CR を使用します。

```
apiVersion: kubevirt.io/v1alpha1
kind: NodeMaintenance
metadata:
  name: node02-maintenance
spec:
  nodeName: node02
  reason: "Replacing node02"
```

2. **NodeMaintenance** オブジェクトをクラスターに作成します。

```
$ oc apply -f <node02-maintenance.yaml>
```

ノードは **liveMigration** エビクションストラテジーを持つ仮想マシンインスタンスのライブマイグレーションを実行し、これがスケジュール対象外になるようにノードにテイントを設定します。このノードの他のすべての Pod および仮想マシンは削除され、別のノードで再作成されます。

追加リソース:


- [メンテナンスモードからのノードの再開](#)

9.4. メンテナンスモードからのノードの再開

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


Web コンソールまたは CLI のいずれかでノードをメンテナンスモードから再開します。

9.4.1. Web コンソールでのメンテナンスモードからのノードの再開

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

手順

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

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

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

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

9.4.2. CLI でのメンテナンスモードからのノードの再開

ノードの **NodeMaintenance** オブジェクトを削除することによって、メンテナンスモードからノードを再開し、これを再度スケジュール可能な状態にします。

手順

1. **NodeMaintenance** オブジェクトを見つけます。

```
$ oc get nodemaintenance
```

2. オプション: **NodeMaintenance** オブジェクトを検査し、これが正しいノードに関連付けられていることを確認します。

```
$ oc describe nodemaintenance <node02-maintenance>
```

```
Name:      node02-maintenance
Namespace:
Labels:
Annotations:
API Version: kubevirt.io/v1alpha1
Kind:      NodeMaintenance
...
Spec:
  Node Name: node02
  Reason:   Replacing node02
```

3. **NodeMaintenance** オブジェクトを削除します。

| \$ oc delete nodemaintenance <node02-maintenance>

第10章 ノードのネットワーク

10.1. ノードのネットワーク状態の確認

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

10.1.1. nmstate について

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

ノードのネットワークは、以下のオブジェクトによって監視され更新されます。

NodeNetworkState

そのノード上のネットワークの状態を報告します。

NodeNetworkConfigurationPolicy

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

NodeNetworkConfigurationEnactment

各ノードに制定されたネットワークポリシーを報告します。

10.1.2. ノードのネットワーク状態の表示

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

手順

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

```
$ oc get nns
```

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

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

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkState
metadata:
  name: node01 ①
status:
  currentState: ②
  dns-resolver:
  ...
  interfaces:
  ...
  route-rules:
  ...
```



```

routes:
...
lastSuccessfulUpdateTime: "2020-01-31T12:14:00Z" ❸

```

- ❶ **NodeNetworkState** の名前はノードから取られています。
- ❷ **currentState** には、DNS、インターフェイス、およびルートを含む、ノードの完全なネットワーク設定が含まれます。
- ❸ 最後に成功した更新のタイムスタンプ。これは、ノードが到達可能であり、レポートの鮮度の評価に使用できる限り定期的に更新されます。

10.2. ノードのネットワーク設定の更新

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

10.2.1. nmstate について

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

ノードのネットワークは、以下のオブジェクトによって監視され更新されます。

NodeNetworkState

そのノード上のネットワークの状態を報告します。

NodeNetworkConfigurationPolicy

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

NodeNetworkConfigurationEnactment

各ノードに制定されたネットワークポリシーを報告します。

10.2.2. ノード上でのインターフェイスの作成

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

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

手順

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

```

apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:

```

```

name: <br1-eth1-policy> ❶
spec:
  nodeSelector: ❷
    node-role.kubernetes.io/worker: "" ❸
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with eth1 as a port ❹
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
        bridge:
          options:
            stp:
              enabled: false
        port:
          - name: eth1

```

- ❶ Policy の名前。
- ❷ オプション。**nodeSelector** を含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
- ❸ この例では **node-role.kubernetes.io/worker: ""** ノードセレクターを使用し、クラスター内のすべてのワーカーノードを選択します。
- ❹ オプション。インターフェイスの人間が判読できる説明。

2. Policy を作成します。

```
$ oc apply -f <br1-eth1-policy.yaml> ❶
```

- ❶ Policy マニフェストのファイル名。

10.2.3. ノード上でのポリシー更新の確認

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

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

手順

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

```
$ oc get nncp
```

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

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

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

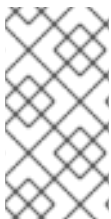
```
$ oc get nnce
```

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

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

10.2.4. ノードからインターフェイスの削除

NodeNetworkConfigurationPolicy オブジェクトを編集してノードからインターフェイスを削除し、インターフェイスの **state** を **absent** に設定します。



注記

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

手順

1. インターフェイスの作成に使用する **NodeNetworkConfigurationPolicy** マニフェストを更新します。以下の例では、Linux ブリッジが削除されます。

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: <br1-eth1-policy> ❶
spec:
  nodeSelector: ❷
    node-role.kubernetes.io/worker: "" ❸
  desiredState:
    interfaces:
      - name: br1
        type: linux-bridge
        state: absent ❹
```

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

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

2. ノード上で Policy を更新し、インターフェイスを削除します。

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

- ① Policy マニフェストのファイル名。

10.2.5. インターフェイスの削除後のノードネットワーク設定の復元

ノードからインターフェイスを削除しても、ノードのネットワーク設定は以前の状態に自動的に復元されません。インターフェイスを削除した後は、クラスター全体で、以前にインターフェイスに割り当てられたか、または以前のインターフェイスの下位に置かれたノード NIC は **down** 状態になります。新規 **NodeNetworkConfigurationPolicy** マニフェストをクラスターに適用した後に NIC を復元します。

手順

1. NIC および必要な状態の **up** を指定する **NodeNetworkConfigurationPolicy** マニフェストを作成します。

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: eth1
spec:
  desiredState:
    interfaces:
    - name: eth1
      type: ethernet
      state: up
      ipv4:
        dhcp: true
        enabled: true
```

2. マニフェストをクラスターに適用します。

```
$ oc apply -f <eth1.yaml> ①
```

- ① Policy マニフェストのファイル名。

10.2.6. 例: Linux ブリッジインターフェイス NodeNetworkConfigurationPolicy

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

以下の YAML ファイルは、Linux ブリッジインターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
```

```

metadata:
  name: br1-eth1-policy ❶
spec:
  nodeSelector: ❷
    kubernetes.io/hostname: <node01> ❸
  desiredState:
    interfaces:
      - name: br1 ❹
        description: Linux bridge with eth1 as a port ❺
        type: linux-bridge ❻
        state: up ❼
        ipv4:
          dhcp: true ❽
          enabled: true ❾
        bridge:
          options:
            stp:
              enabled: false ❿
          port:
            - name: eth1 ⓫

```

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

10.2.7. 例: VLAN インターフェイス NodeNetworkConfigurationPolicy

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

以下の YAML ファイルは、VLAN インターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```
apiVersion: nmstate.io/v1alpha1
```

```

kind: NodeNetworkConfigurationPolicy
metadata:
  name: vlan-eth1-policy ❶
spec:
  nodeSelector: ❷
    kubernetes.io/hostname: <node01> ❸
  desiredState:
    interfaces:
      - name: eth1.102 ❹
        description: VLAN using eth1 ❺
        type: vlan ❻
        state: up ❼
        vlan:
          base-iface: eth1 ❽
          id: 102 ❾

```

- ❶ Policy の名前。
- ❷ オプション。 **nodeSelector** を含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
- ❸ この例では、 **hostname** ノードセクターを使用します。
- ❹ インターフェイスの名前。
- ❺ オプション。人間が判読できるインターフェイスの説明。
- ❻ インターフェイスのタイプ。以下の例では VLAN を作成します。
- ❼ 作成後のインターフェイスの要求された状態。
- ❽ VLAN が接続されているノードの NIC。
- ❾ VLAN タグ。

10.2.8. 例: ボンドインターフェイス NodeNetworkConfigurationPolicy

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



注記

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

- mode=1 active-backup
- mode=5 balance-tlb
- mode=6 balance-alb

以下の YAML ファイルは、ボンドインターフェイスのマニフェストの例です。これには、独自の情報で置き換える必要のあるサンプルの値が含まれます。

```

apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: bond0-eth1-eth2-policy ❶
spec:
  nodeSelector: ❷
    kubernetes.io/hostname: <node01> ❸
  desiredState:
    interfaces:
      - name: bond0 ❹
        description: Bond enslaving eth1 and eth2 ❺
        type: bond ❻
        state: up ❼
        ipv4:
          dhcp: true ❽
          enabled: true ❾
        link-aggregation:
          mode: active-backup ❿
          options:
            miimon: '140' ㉑
          slaves: ㉒
            - eth1
            - eth2
        mtu: 1450 ㉓

```

- ❶ Policy の名前。
- ❷ オプション。**nodeSelector** を含めない場合、ポリシーはクラスター内のすべてのノードに適用されます。
- ❸ この例では、**hostname** ノードセクターを使用します。
- ❹ インターフェイスの名前。
- ❺ オプション。人間が判読できるインターフェイスの説明。
- ❻ インターフェイスのタイプ。この例では、ボンドを作成します。
- ❼ 作成後のインターフェイスの要求された状態。
- ❽ オプション。**dhcp** を使用しない場合は、静的 IP を設定するか、IP アドレスなしでインターフェイスを出ることができます。
- ❾ この例では **ipv4** を有効にします。
- ❿ ボンドのドライバーモード。この例では、アクティブなバックアップモードを使用します。
- ㉑ オプション。この例では、miimon を使用して 140ms ごとにボンドリンクを検査します。
- ㉒ ボンド内の下位ノードの NIC。
- ㉓ オプション。ボンドの Maximum transmission unit (MTU)指定がない場合、この値はデフォルトで **1500** に設定されます。

10.3. ノードのネットワーク設定のトラブルシューティング

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

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

10.3.1. 正確でない NodeNetworkConfigurationPolicy 設定のトラブルシューティング

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

この例では、Linux ブリッジ Policy は 3 つのマスターノードと 3 つのワーカーノードを持つクラスターのサンプルに適用されます。Policy は正しくないインターフェイスを参照するために、適用することができません。エラーを確認するには、利用可能な nmstate リソースを調べます。その後、正しい設定で Policy を更新できます。

手順

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

```
apiVersion: nmstate.io/v1alpha1
kind: NodeNetworkConfigurationPolicy
metadata:
  name: ens01-bridge-testfail
spec:
  desiredState:
    interfaces:
      - name: br1
        description: Linux bridge with the wrong port
        type: linux-bridge
        state: up
        ipv4:
          dhcp: true
          enabled: true
        bridge:
          options:
            stp:
              enabled: false
          port:
            - name: ens01
```

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

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

2. 以下のコマンドを実行して Policy のステータスを確認します。


```
$ oc get nncp
```

この出力は、Policy が失敗したことを示しています。

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

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

3. Enactment を一覧表示し、Policy がいずれかのノードで成功したかどうかを確認します。Policy がサブセットに対してのみ失敗した場合は、問題は特定のノード設定にあることがあることが示唆されます。Policy がすべてのノードで失敗した場合は、問題は Policy に関連したものであることが示唆されます。

```
$ oc get nnce
```

この出力は、Policy がすべてのノードで失敗したことを示しています。

```
NAME                STATUS
output
master-1.ens01-bridge-testfail FailedToConfigure
master-2.ens01-bridge-testfail FailedToConfigure
master-3.ens01-bridge-testfail FailedToConfigure
worker-1.ens01-bridge-testfail FailedToConfigure
worker-2.ens01-bridge-testfail FailedToConfigure
worker-3.ens01-bridge-testfail FailedToConfigure
```

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

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

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

```
error reconciling NodeNetworkConfigurationPolicy at desired state apply: , failed to execute
nmstatectl set --no-commit --timeout 480: 'exit status 1' "
...
libnmstate.error.NmstateVerificationError:
desired
=====
---
name: br1
type: linux-bridge
state: up
bridge:
  options:
    group-forward-mask: 0
    mac-ageing-time: 300
    multicast-snooping: true
  stp:
    enabled: false
    forward-delay: 15
```

```

    hello-time: 2
    max-age: 20
    priority: 32768
  port:
  - name: ens01
description: Linux bridge with the wrong port
ipv4:
  address: []
  auto-dns: true
  auto-gateway: true
  auto-routes: true
  dhcp: true
  enabled: true
ipv6:
  enabled: false
mac-address: 01-23-45-67-89-AB
mtu: 1500

```

```

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

```

```

difference
=====
--- desired
+++ current
@@ -13,8 +13,7 @@
    hello-time: 2
    max-age: 20

```

```

    priority: 32768
  - port:
  - - name: ens01
+ port: []
  description: Linux bridge with the wrong port
  ipv4:
    address: []
    line 651, in _assert_interfaces_equal\n
current_state.interfaces[ifname],\nlibnmstate.error.NmstateVerificationError:

```

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

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

```
$ oc get nns master-1 -o yaml
```

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

```

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

```

6. 既存の Policy を編集してエラーを修正します。

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

```

...
  port:
    - name: ens1

```

Policy を保存して修正を適用します。

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

```
$ oc get nncp
```

```

NAME                STATUS
ens01-bridge-testfail SuccessfullyConfigured

```

更新された Policy は、クラスターのすべてのノードで正常に設定されました。

第11章 ロギング、イベント、およびモニターリング

11.1. 仮想マシンログの表示

11.1.1. 仮想マシンのログについて

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

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

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



警告

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

11.1.2. CLI での仮想マシンログの表示

仮想マシンランチャー Pod から仮想マシンログを取得します。

手順

- 以下のコマンドを実行します。

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

11.1.3. Web コンソールでの仮想マシンログの表示

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

手順

1. Container-native Virtualization コンソールで **Workloads** → **Virtual Machines** をクリックします。
2. 仮想マシンをクリックし、**Virtual Machine Details** パネルを開きます。
3. **Overview** タブで、**POD** セクションの **virt-launcher-*<name>*** Pod をクリックします。
4. **Logs** をクリックします。

11.2. イベントの表示

11.2.1. 仮想マシンイベントについて

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

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

[OpenShift Container Platform クラスタ内のシステムイベント情報の表示](#) も参照してください。

11.2.2. Web コンソールでの仮想マシンのイベントの表示

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

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

手順

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

11.2.3. CLI での namespace イベントの表示

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

手順

- namespace で、**oc get** コマンドを使用します。

```
$ oc get events
```

11.2.4. CLI でのリソースイベントの表示

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

手順

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

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

11.3. 仮想マシンのワークロードに関する情報の表示

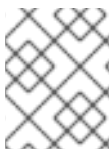
OpenShift Container Platform Web コンソールで **Virtual Machines** ダッシュボードを使用して、仮想マシンについての概要を表示できます。

11.3.1. Virtual Machines ダッシュボードについて

Workloads → **Virtual Machines** ページに移動し、仮想マシンを選択して、OpenShift Container Platform Web コンソールから **Virtual Machines** ダッシュボードにアクセスします。

ダッシュボードには、以下のカードが含まれます。

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



注記

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

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

11.4. 仮想マシンの正常性のモニタリング

以下の手順を使用して liveness および readiness プローブを作成し、仮想マシンの正常性監視します。

11.4.1. liveness および readiness プローブ

VirtualMachineInstance (VMI) が失敗すると、**liveness プローブ**は VMI を停止します。次に、VirtualMachine などのコントローラーが他の VMI を起動し、仮想マシンの応答を復元します。

readiness プローブは、サービスとエンドポイントに対して VirtualMachineInstance がサービスからトラフィックを受信できることを示します。readiness プローブが失敗すると、VirtualMachineInstance はプロブが復旧するまで、該当のエンドポイントから削除されます。

11.4.2. HTTP liveness プローブの定義

以下の手順では、HTTP liveness プローブを定義する設定ファイルの例について説明します。

手順

1. 以下のコードブロックをサンプルとして使用し、YAML 設定ファイルをカスタマイズして HTTP liveness プローブを作成します。この例では、以下のようになります。
 - 初回の **120** 秒の遅延後に VirtualMachineInstance のポート **1500** をクエリーする **spec.livenessProbe.httpGet** を使用してプロブを設定します。
 - VirtualMachineInstance は、**cloud-init** を使用して、ポート **1500** に最小限の HTTP サーバーをインストールし、実行します。

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-fedora
  name: vmi-fedora
spec:
  domain:
    devices:
      disks:
        - disk:
            bus: virtio
            name: containerdisk
        - disk:
            bus: virtio
            name: cloudinitdisk
      resources:
        requests:
          memory: 1024M
  livenessProbe:
    initialDelaySeconds: 120
    periodSeconds: 20
    httpGet:
      port: 1500
      timeoutSeconds: 10
  terminationGracePeriodSeconds: 0
  volumes:
    - name: containerdisk
      registryDisk:
        image: kubevirt/fedora-cloud-registry-disk-demo
    - cloudInitNoCloud:
        userData: |-
```

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

2. 以下のコマンドを実行して VirtualMachineInstance を作成します。

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

11.4.3. TCP liveness プローブの定義

以下の手順では、TCP liveness プローブを定義する設定ファイルの例について説明します。

手順

1. このコードブロックをサンプルとして使用し、YAML 設定ファイルをカスタマイズして TCP liveness プローブを作成します。この例では、以下のようになります。
 - 初回の **120** 秒の遅延後に VirtualMachineInstance のポート **1500** をクエリーする **spec.livenessProbe.tcpSocket** を使用してプローブを設定します。
 - VirtualMachineInstance は、**cloud-init** を使用して、ポート **1500** に最小限の HTTP サーバーをインストールし、実行します。

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-fedora
    name: vmi-fedora
spec:
  domain:
    devices:
      disks:
        - disk:
            bus: virtio
            name: containerdisk
        - disk:
            bus: virtio
            name: cloudinitdisk
      resources:
        requests:
          memory: 1024M
  livenessProbe:
    initialDelaySeconds: 120
    periodSeconds: 20
    tcpSocket:
      port: 1500
    timeoutSeconds: 10
    terminationGracePeriodSeconds: 0
```



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

2. 以下のコマンドを実行して VirtualMachineInstance を作成します。

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

11.4.4. readiness プロープの定義

以下の手順では、readiness プロープを定義する設定ファイルの例について説明します。

手順

1. YAML 設定ファイルをカスタマイズして readiness プロープを作成します。readiness プロープは liveness プロープと同じように設定されます。ただし、この例では以下の違いに注意してください。
 - readiness プロープは、異なる仕様名を使用して保存されます。たとえば、readiness プロープを **spec.livenessProbe.<type-of-probe>** としてではなく、**spec.readinessProbe** として作成します。
 - readiness プロープを作成する場合、プローブが複数回失敗または成功する場合に備えて **ready** と **non-ready** 状態の間に切り換えられるように **failureThreshold** および **successThreshold** をオプションで設定します。

```
apiVersion: kubevirt.io/v1alpha3
kind: VirtualMachineInstance
metadata:
  labels:
    special: vmi-fedora
    name: vmi-fedora
spec:
  domain:
    devices:
      disks:
        - disk:
            bus: virtio
            name: containerdisk
        - disk:
            bus: virtio
            name: cloudinitdisk
  resources:
```

```

      requests:
        memory: 1024M
    readinessProbe:
      httpGet:
        port: 1500
      initialDelaySeconds: 120
      periodSeconds: 20
      timeoutSeconds: 10
      failureThreshold: 3
      successThreshold: 3
    terminationGracePeriodSeconds: 0
    volumes:
    - name: containerdisk
      registryDisk:
        image: kubevirt/fedora-cloud-registry-disk-demo
    - cloudInitNoCloud:
        userData: |-
          #cloud-config
          password: fedora
          chpasswd: { expire: False }
        bootcmd:
        - setenforce 0
        - dnf install -y nmap-ncat
        - systemd-run --unit=httpserver nc -klp 1500 -e '/usr/bin/echo -e HTTP/1.1 200
OK\n\nHello World!'
      name: cloudinitdisk

```

2. 以下のコマンドを実行して VirtualMachineInstance を作成します。

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

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

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

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

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

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

- **Details** は、クラスターの詳細情報の概要を表示します。
ステータスには、**ok**、**error**、**warning**、**in progress**、および **unknown** が含まれます。リソースでは、カスタムのステータス名を追加できます。
 - クラスター
 - プロバイダー
 - バージョン

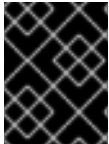
- **Cluster Inventory** は、リソースの数および関連付けられたステータスの詳細を表示します。これは、問題の解決に介入が必要な場合に役立ちます。以下についての情報が含まれます。
 - ノード数
 - Pod 数
 - 永続ストレージボリューム要求 (Persistent Storage Volume Claim)
 - 仮想マシン (Container-native Virtualization がインストールされている場合に利用可能)
 - クラスター内のベアメタルホスト。これらはステータス別に一覧表示されます (**metal3** 環境でのみ利用可能)。
- **Cluster Health** では、関連するアラートおよび説明を含む、クラスターの現在の健全性についてのサマリーを表示します。Container-native Virtualization がインストールされている場合は、Container-native Virtualization の全体的な健全性も診断されます。複数のサブシステムが存在する場合は、**See All** をクリックして、各サブシステムのステータスを表示します。
- **Cluster Capacity** チャートは、管理者が追加リソースがクラスターで必要になるタイミングを把握するのに役立ちます。このチャートには、現在の消費量を表示する内側の円が含まれ、外側の円には、以下の情報を含む、リソースに対して設定されたしきい値が表示されます。
 - CPU 時間
 - メモリー割り当て
 - 消費されるストレージ
 - 消費されるネットワークリソース
- **Cluster Utilization** は指定された期間における各種リソースの容量を表示します。これは、管理者がリソースの高い消費量の規模および頻度を理解するのに役立ちます。
- **Events** は、Pod の作成または別のホストへの仮想マシンの移行などのクラスター内の最近のアクティビティーに関連したメッセージを一覧表示します。
- **Top Consumers** は、管理者がクラスターリソースが消費される状況を把握するのに役立ちます。リソースをクリックし、指定されたクラスターリソース (CPU、メモリー、またはストレージ) の最大量を消費する Pod およびノードを一覧表示する詳細ページに切り替えます。

11.6. OPENSIFT CONTAINER PLATFORM クラスターモニターリング、ログイン、および TELEMETRY

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

11.6.1. OpenShift Container Platform クラスターモニターリングについて

OpenShift Container Platform には、[Prometheus](#) オープンソースプロジェクトおよびその幅広いエコシステムをベースとする事前に設定され、事前にインストールされた自己更新型のモニターリングスタックが同梱されます。これはクラスターのモニターリング機能を提供し、クラスター管理者に問題の発生を即時に通知するアラートのセットと [Grafana](#) ダッシュボードのセットを提供します。クラスターモニターリングスタックは、OpenShift Container Platform クラスターのモニターリング用のみにサポートされています。



重要

今後の OpenShift Container Platform の更新との互換性を確保するために、指定されたモニターリングスタックのオプションのみを設定することがサポートされます。

11.6.2. クラスターロギングコンポーネント

クラスターロギングコンポーネントは Elasticsearch、Fluentd、Kibana (EFK) に基づいています。コレクターの [Fluentd](#) は、OpenShift Container Platform クラスターの各ノードにデプロイされます。これはすべてのノードおよびコンテナのログを収集し、それらを [Elasticsearch](#) (ES) に書き込みます。[Kibana](#) は、ユーザーおよび管理者が集計されたデータを使って高度な視覚化およびダッシュボードを作成できる中央の Web UI です。

現時点で、5 種類のクラスターロギングコンポーネントがあります。

- **logStore:** これはログが保存される場所です。現在の実装は Elasticsearch です。
- **collection:** これは、ノードからログを収集し、それらをフォーマットし、logStore に保存するコンポーネントです。現在の実装は Fluentd です。
- **visualization:** これは、ログ、グラフ、チャートなどを表示するために使用される UI コンポーネントです。現在の実装は Kibana です。
- **curation:** これは期間に基づいてログをトリミングするコンポーネントです。現在の実装は Curator です。

クラスターロギングについての詳細は、[OpenShift Container Platform クラスターロギング](#) のドキュメントを参照してください。

11.6.3. Telemetry について

Telemetry は厳選されたクラスターモニターリングメトリクスのサブセットを Red Hat に送信します。これらのメトリクスは継続的に送信され、以下について記述します。

- OpenShift Container Platform クラスターのサイズ
- OpenShift Container Platform コンポーネントの健全性およびステータス
- 実行されるアップグレードの正常性およびステータス
- OpenShift Container Platform のコンポーネントおよび機能についての使用情報 (一部の制限された情報)
- クラスターモニターリングコンポーネントによってレポートされるアラートについてのサマリー情報

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

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

11.6.3.1. Telemetry で収集される情報

Telemetry によって収集される主な情報には、以下が含まれます。

- クラスターごとに利用可能な更新の数
- 更新に使用されるチャネルおよびイメージリポジトリ
- 更新中に発生するエラーの数
- 実行中の更新の進捗情報
- クラスターごとのマシン数
- CPU コアの数およびマシンの RAM のサイズ
- etcd クラスターのメンバー数、および現在 etcd クラスターに保存されているオブジェクトの数
- マシンタイプ (インフラまたはマスター) ごとに使用される CPU コアおよび RAM の数
- クラスターごとに使用される CPU コアおよび RAM の数
- クラスター内での実行中の仮想マシンインスタンスの数
- クラスターごとの OpenShift Container Platform フレームワークコンポーネントの使用
- OpenShift Container Platform クラスターのバージョン
- クラスターにインストールされている OpenShift Container Platform フレームワークコンポーネントの健全性、状態、およびステータス。たとえば、クラスターバージョン Operator、クラスターモニタリング、イメージレジストリー、およびロギング用の Elasticsearch がこれらのコンポーネントに含まれます。
- インストール時に生成される一意でランダムな識別子
- Amazon Web Services などの OpenShift Container Platform がデプロイされているプラットフォームの名前

Telemetry は、ユーザー名、パスワード、またはユーザーリソースの名前またはアドレスなどの識別情報を収集しません。

11.6.4. CLI のトラブルシューティングおよびデバッグコマンド

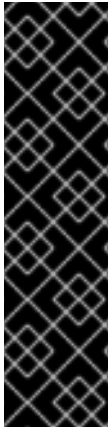
oc クライアントのトラブルシューティングおよびデバッグコマンドの一覧については、[OpenShift Container Platform CLI ツール](#) のドキュメントを参照してください。

11.7. RED HAT サポート向けの CONTAINER-NATIVE VIRTUALIZATION データの収集

サポートケースを作成する際、ご使用のクラスターについてのデバッグ情報を Red Hat サポートに提供していただくと Red Hat のサポートに役立ちます。

must-gather ツールを使用すると、仮想マシンおよび Container-native Virtualization に関連する他のデータを含む、OpenShift Container Platform クラスターについての診断情報を収集できます。

迅速なサポートを得るには、OpenShift Container Platform と container-native virtualization の両方についての診断情報を提供してください。



重要

Container-native Virtualization はテクノロジープレビュー機能です。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。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ついての詳細は、<https://access.redhat.com/ja/support/offerings/techpreview/> を参照してください。

11.7.1. must-gather ツールについて

oc adm must-gather CLI コマンドは、以下のような問題のデバッグに必要となる可能性のあるクラスターからの情報を収集します。

- リソース定義
- 監査ログ
- サービスログ

--image 引数を指定してコマンドを実行する際にイメージを指定できます。イメージを指定する際、ツールはその機能または製品に関連するデータを収集します。

oc adm must-gather を実行すると、新しい Pod がクラスターに作成されます。データは Pod で収集され、**must-gather.local** で始まる新規ディレクトリーに保存されます。このディレクトリーは、現行の作業ディレクトリーに作成されます。

11.7.2. Container-native Virtualization データの収集について

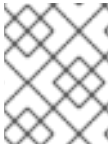
oc adm must-gather CLI コマンドを使用してクラスターについての情報を収集できます。これには、Container-native Virtualization に関連する機能およびオブジェクトが含まれます。

- Hyperconverged Cluster Operator namespace(および子オブジェクト)
- Container-native Virtualization リソースに属するすべての namespace (およびそれらの子オブジェクト)
- すべての Container-native Virtualization カスタムリソース定義 (CRD)
- 仮想マシンを含むすべての namespace
- すべての仮想マシン定義

must-gather を使用して Container-native Virtualization データを収集するには、Container-Native Virtualization イメージ **--image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8** を指定する必要があります。

11.7.3. 特定の機能に関するデータ収集

oc adm must-gather CLI コマンドを **--image** または **--image-stream** 引数と共に使用して、特定に機能についてのデバッグ情報を収集できます。**must-gather** ツールは複数のイメージをサポートするため、単一のコマンドを実行して複数の機能についてのデータを収集できます。



注記

特定の機能データに加えてデフォルトの **must-gather** データを収集するには、**--image-stream=openshift/must-gather** 引数を追加します。

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

手順

1. **must-gather** データを保存するディレクトリーに移動します。
2. **oc adm must-gather** コマンドを1つまたは複数の **--image** または **--image-stream** 引数と共に実行します。たとえば、以下のコマンドは、デフォルトのクラスターデータと {VirtProductName} に固有の情報の両方を収集します。

```
$ oc adm must-gather \
--image-stream=openshift/must-gather \ ❶
--image=registry.redhat.io/container-native-virtualization/cnv-must-gather-rhel8 ❷
```

- ❶ デフォルトの OpenShift Container Platform **must-gather** イメージ
- ❷ {Virt virtualization} の **must-gather** イメージ

must-gather ツールを追加の引数と共に使用し、クラスターロギングおよびクラスター内の Cluster Logging Operator に関連するデータを収集できます。クラスターロギングの場合、以下のコマンドを実行します。

```
$ oc adm must-gather --image=$(oc -n openshift-logging get deployment.apps/cluster-logging-operator \
-o jsonpath='{.spec.template.spec.containers[?(@.name == "cluster-logging-operator")].image}')
```

例11.1 クラスターロギングの **must-gather** の出力例

```
├── cluster-logging
│   ├── clo
│   │   ├── cluster-logging-operator-74dd5994f-6ttgt
│   │   ├── clusterlogforwarder_cr
│   │   ├── cr
│   │   ├── csv
│   │   ├── deployment
│   │   └── logforwarding_cr
│   ├── collector
│   │   └── fluentd-2tr64
│   └── curator
```



```

├── curator-1596028500-zkz4s
├── eo
│   ├── csv
│   ├── deployment
│   └── elasticsearch-operator-7dc7d97b9d-jb4r4
├── es
│   ├── cluster-elasticsearch
│   │   ├── aliases
│   │   ├── health
│   │   ├── indices
│   │   ├── latest_documents.json
│   │   ├── nodes
│   │   ├── nodes_stats.json
│   │   └── thread_pool
│   ├── cr
│   ├── elasticsearch-cdm-lp8l38m0-1-794d6dd989-4jxms
│   ├── logs
│   └── elasticsearch-cdm-lp8l38m0-1-794d6dd989-4jxms
├── install
│   ├── co_logs
│   ├── install_plan
│   ├── olmo_logs
│   └── subscription
├── kibana
│   ├── cr
│   └── kibana-9d69668d4-2rk vz
├── cluster-scoped-resources
├── core
│   ├── nodes
│   │   └── ip-10-0-146-180.eu-west-1.compute.internal.yaml
│   └── persistentvolumes
│       └── pvc-0a8d65d9-54aa-4c44-9ecc-33d9381e41c1.yaml
├── event-filter.html
├── gather-debug.log
├── namespaces
├── openshift-logging
│   ├── apps
│   │   ├── daemonsets.yaml
│   │   ├── deployments.yaml
│   │   ├── replicaset.yaml
│   │   └── statefulsets.yaml
│   ├── batch
│   │   ├── cronjobs.yaml
│   │   └── jobs.yaml
│   ├── core
│   │   ├── configmaps.yaml
│   │   ├── endpoints.yaml
│   │   └── events
│   │       ├── curator-1596021300-wn2ks.162634ebf0055a94.yaml
│   │       ├── curator.162638330681bee2.yaml
│   │       ├── elasticsearch-delete-app-1596020400-gm6nl.1626341a296c16a1.yaml
│   │       ├── elasticsearch-delete-audit-1596020400-9l9n4.1626341a2af81bbd.yaml
│   │       ├── elasticsearch-delete-infra-1596020400-v98tk.1626341a2d821069.yaml
│   │       ├── elasticsearch-rollover-app-1596020400-cc5vc.1626341a3019b238.yaml
│   │       ├── elasticsearch-rollover-audit-1596020400-s8d5s.1626341a31f7b315.yaml
│   │       └── elasticsearch-rollover-infra-1596020400-7mgv8.1626341a35ea59ed.yaml

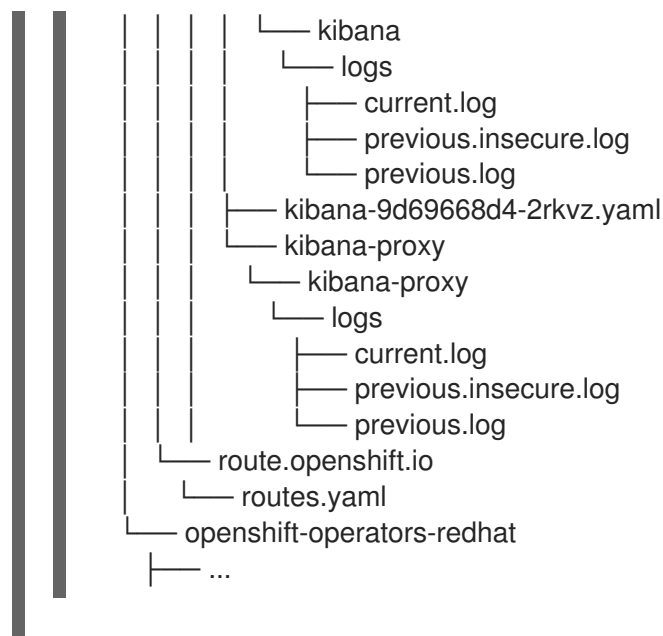
```



```

├── events.yaml
├── persistentvolumeclaims.yaml
├── pods.yaml
├── replicationcontrollers.yaml
├── secrets.yaml
├── services.yaml
├── openshift-logging.yaml
├── pods
│   ├── cluster-logging-operator-74dd5994f-6ttgt
│   │   ├── cluster-logging-operator
│   │   │   └── logs
│   │   │       ├── current.log
│   │   │       ├── previous.insecure.log
│   │   │       └── previous.log
│   │   └── cluster-logging-operator-74dd5994f-6ttgt.yaml
│   ├── cluster-logging-operator-registry-6df49d7d4-mxxff
│   │   ├── cluster-logging-operator-registry
│   │   │   └── logs
│   │   │       ├── current.log
│   │   │       ├── previous.insecure.log
│   │   │       └── previous.log
│   │   ├── cluster-logging-operator-registry-6df49d7d4-mxxff.yaml
│   │   ├── mutate-csv-and-generate-sqlite-db
│   │   │   └── logs
│   │   │       ├── current.log
│   │   │       ├── previous.insecure.log
│   │   │       └── previous.log
│   ├── curator-1596028500-zkz4s
│   ├── elasticsearch-cdm-lp8l38m0-1-794d6dd989-4jxms
│   ├── elasticsearch-delete-app-1596030300-bpgcx
│   │   ├── elasticsearch-delete-app-1596030300-bpgcx.yaml
│   │   ├── indexmanagement
│   │   │   └── logs
│   │   │       ├── current.log
│   │   │       ├── previous.insecure.log
│   │   │       └── previous.log
│   ├── fluentd-2tr64
│   │   ├── fluentd
│   │   │   └── logs
│   │   │       ├── current.log
│   │   │       ├── previous.insecure.log
│   │   │       └── previous.log
│   │   ├── fluentd-2tr64.yaml
│   │   ├── fluentd-init
│   │   │   └── logs
│   │   │       ├── current.log
│   │   │       ├── previous.insecure.log
│   │   │       └── previous.log
│   ├── kibana-9d69668d4-2rkvz
│   └── kibana

```



- 作業ディレクトリーに作成された **must-gather** ディレクトリーから圧縮ファイルを作成します。たとえば、Linux オペレーティングシステムを使用するコンピューターで以下のコマンドを実行します。

```
$ tar cvaf must-gather.tar.gz must-gather.local.5421342344627712289/ 1
```

- 1** **must-gather-local.5421342344627712289/** を実際のディレクトリー名に置き換えてください。

- 圧縮ファイルを [Red Hat カスタマーポータル](#) で作成したサポートケースに添付します。