Red Hat OpenStack Platform 17.1

director Operator を使用した Red Hat OpenShift Container Platform クラスターへのオーバークラウドのデプロイ

director Operator を使用して Red Hat OpenShift Container Platform で Red Hat OpenStack Platform オーバークラウドをデプロイおよび管理する

Last Updated: 2024-05-25

Red Hat OpenStack Platform 17.1 director Operator を使用した Red Hat OpenShift Container Platform クラスターへのオーバークラウドのデプロイ

director Operator を使用して Red Hat OpenShift Container Platform で Red Hat OpenStack Platform オーバークラウドをデプロイおよび管理する

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

RHOSP director Operator を Red Hat OpenShift Container Platform クラスターにインストールし、director Operator を使用して RHOSP オーバークラウドをデプロイする方法を学びます。 Red Hat OpenStack Platform director Operator のサポートは、アーキテクチャーが Red Hat サービスまたはテクニカルアカウントマネージャーによって承認された場合にのみ付与されます。この機能を導入する前に Red Hat にお問い合わせください。

目次

多様性を受け入れるオープンソースの強化

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

RED HAT ドキュメントへのフィードバック (英語のみ)

Red Hat ドキュメントに対するご意見をお聞かせください。ドキュメントの改善点があればお知らせください。

Jira でドキュメントのフィードバックを提供する

ドキュメントに関するフィードバックを提供するには、Create Issue フォームを使用します。Red Hat OpenStack Platform Jira プロジェクトで Jira Issue が作成され、フィードバックの進行状況を追跡できます。

1. Jira にログインしていることを確認してください。Jira アカウントをお持ちでない場合は、アカウントを作成してフィードバックを送信してください。
2. Create Issue をクリックして、**Create Issue** ページを開きます。
3. **Summary** フィールドと **Description** フィールドに入力します。**Description** フィールドに、ドキュメントの URL、章またはセクション番号、および問題の詳しい説明を入力します。フォーム内の他のフィールドは変更しないでください。
4. **Create** をクリックします。

第1章 DIRECTOR OPERATOR を使用した RHOSP オーバークラウドの作成とデプロイ

Red Hat OpenShift Container Platform (RHOCP) は、Operator のモジュラーシステムを使用して RHOCP クラスターの機能を拡張します。Red Hat OpenStack Platform (RHOSP) director Operator (OSPdO) は、RHOCP 内で RHOSP クラウドをインストールして実行する機能を追加します。OSPdO は、RHOSP ノードのインフラストラクチャーと設定をデプロイおよび管理する一連のカスタムリソース定義 (CRD) を管理します。OSPdO でデプロイされた RHOSP クラウドの基本アーキテクチャーには、以下の機能が含まれます。

仮想コントロールプレーン

コントローラーノードは、OSPdO が Red Hat OpenShift Virtualization で作成する仮想マシン (VM) です。

ベアメタルマシンのプロビジョニング

OSPdO は、RHOCP ベアメタルマシン管理を使用して、RHOSP クラウドの Compute ノードをプロビジョニングします。

ネットワーク

OSPdO は、RHOSP サービスの基盤となるネットワークを設定します。

Heat および Ansible ベースの設定

OSPdO はカスタム Heat 設定を RHOCP に保存し、director の **config-download** 機能を使用して設定を Ansible Playbook に変換します。保存された Heat 設定を変更すると、OSPdO は Ansible Playbook を自動的に再生成します。

CLI クライアント

OSPdO は、ユーザーが RHOSP CLI コマンドを実行し、RHOSP クラウドと対話するための **openstackclient** Pod を作成します。

OSPdO に固有のリソースを使用して、オーバークラウドインフラストラクチャーのプロビジョニング、オーバークラウド設定の生成、およびオーバークラウドの作成を行うことができます。OSPdO を使用して RHOSP オーバークラウドを作成するには、以下のタスクを完了する必要があります。

1. 稼働中の RHOCP クラスターに OSPdO をインストールします。
2. ベースオペレーティングシステムの RHOCP クラスターデータボリュームを作成し、リモート Git リポジトリーの認証の詳細を追加します。
3. **OpenStackNetConfig** CRD を使用して、コントロールプレーンと分離されたネットワークを含むオーバークラウドネットワークを作成します。
4. **ConfigMap** を作成して、オーバークラウド用のカスタム heat テンプレートおよび環境ファイルを保存します。
5. コントローラーノード用の 3 つの仮想マシンと、クライアント操作を実行するための Pod を含むコントロールプレーンを作成します。
6. ベアメタルの Compute ノードを作成します。
7. オーバークラウド設定用の Ansible Playbook をレンダリングするための **OpenStackConfigGenerator** カスタムリソースを作成します。
8. **openstackdeploy** を使用して、Ansible Playbook 設定をオーバークラウドノードに適用します。

1.1. DIRECTOR OPERATOR のカスタムリソース定義

Red Hat OpenStack Platform (RHOSP) director Operator (OSPdO) には、オーバークラウドリソースの管理に使用できるカスタムリソース定義 (CRD) のセットが含まれています。

- OSPdO CRD の完全なリストを表示するには、次のコマンドを使用します。

```
$ oc get crd | grep "^openstack"
```

- 特定の CRD の定義を表示するには、次のコマンドを使用します。

```
$ oc describe crd openstackbaremetalset
Name:         openstackbaremetalsets.osp-director.openstack.org
Namespace:
Labels:       operators.coreos.com/osp-director-operator.openstack=
Annotations:  cert-manager.io/inject-ca-from:
$(CERTIFICATE_NAMESPACE)/$(CERTIFICATE_NAME)
              controller-gen.kubebuilder.io/version: v0.3.0
API Version:  apiextensions.k8s.io/v1
Kind:         CustomResourceDefinition
...
```

- 特定の CRD の設定に使用できるフィールドの説明を表示するには、次のコマンドを使用します。

```
$ oc explain openstackbaremetalset.spec
KIND:     OpenStackBaremetalSet
VERSION:  osp-director.openstack.org/v1beta1

RESOURCE: spec <Object>

DESCRIPTION:
    <empty>

FIELDS:
  count               <Object>
  baseImageUrl        <Object>
  deploymentSSHSecret  <Object>
  ctlplaneInterface    <Object>
  networks            <[]Object>
  ...
```

OSPdO には、ハードウェアプロビジョニングとソフトウェア設定という 2 種類の CRD が含まれています。

ハードウェアプロビジョニング CRD

openstacknetattachment (internal)

OSPdO によって、ネットワークを仮想マシン (VM) に接続するために使用される **NodeNetworkConfigurationPolicy** および **NodeSriovConfigurationPolicy** CRD を管理するために使用されます。

openstacknetconfig

完全なネットワーク設定を記述する **openstacknetattachment** および **openstacknet** CRD を指定するために使用します。各ノードの予約された IP アドレスと MAC アドレスのセットがステータスに反映されます。

openstackbaremetalset

コンピューティングやストレージなど、特定の RHOSP ロール用のベアメタルホストのセットを作成するために使用します。

openstackcontrolplane

RHOSP コントロールプレーンを作成し、関連する **openstackvmset** CR を管理するために使用します。

openstacknet (internal)

IP を **openstackvmset** および **openstackbaremetalset** CR に割り当てるために使用されるネットワークを作成するために使用します。

openstackipset (internal)

特定のネットワークとロールの一連の IP が含まれています。OSPdO が IP アドレスを管理するために使用します。

openstackprovisionservers

Metal3 でベアメタルノードをプロビジョニングするためのカスタムイメージを提供するために使用します。

openstackvmset

コントローラー、データベース、ネットワークコントローラーなど、特定の RHOSP ロール用の OpenShift Virtualization VM のセットを作成するために使用します。

ソフトウェア設定 CRD

openstackconfiggenerator

デプロイメント用のカスタム **ConfigMap** をスケールアップまたは変更するときに、デプロイメント用の Ansible Playbook を自動的に生成するために使用します。

openstackconfigversion

実行可能な Ansible Playbook のセットを表すために使用します。

openstackdeploy

openstackconfigversion CR で定義された Ansible Playbook のセットを実行するために使用します。

openstackclient

RHOSP デプロイメントコマンドの実行に使用される Pod を作成します。

関連情報

- カスタムリソース定義か定義定義らのリソースの管理

1.2. CRD の命名規則

各カスタムリソース定義 (CRD) には、**spec.names** パラメーターを使用して複数の名前を定義できます。どの名前を使用するかは、実行するアクションのコンテキストによって異なります。

- リソースのマニフェストを作成して操作する場合は **kind** を使用します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
...
```

リソースマニフェストの **kind** 名は、それぞれの CRD の **kind** 名に相関します。

- 複数のリソースを操作する場合は **plural** を使用します。

```
$ oc get openstackbaremetalsets
```

- 単一リソースを操作する場合は、**singular** を使用します。

```
$ oc describe openstackbaremetalset/compute
```

- CLI での操作には **shortName** を使用します。

```
$ oc get osbmset
```

1.3. DIRECTOR OPERATOR でサポートされていない機能

ファイバーチャネルバックエンド

ファイバーチャネルを使用するバックエンドについては、Block Storage (cinder) のイメージからボリュームへの転送はサポートされません。Red Hat OpenShift Virtualization は、N_Port ID Virtualization (NPIV) をサポートしていません。したがって、デフォルトで **cinder-volume** が実行される、ストレージバックエンドからコントローラーに LUN をマッピングする必要がある Block Storage ドライバーは機能しません。**cinder-volume** 専用のロールを作成し、そのロールを仮想化コントローラーに含めるのではなく、そのロールを使用して物理ノードを作成する必要があります。詳細は、**Red Hat OpenStack Platform デプロイメントのカスタマイズ** ガイドの コンポーザブルサービスとカスタムロール を参照してください。

ロールベースの Ansible Playbook

Director Operator (OSPdO) は、ベアメタルノードのプロビジョニング後にロールベースのノード属性を設定するための Ansible Playbook の実行をサポートしていません。つまり、追加の Ansible 変数 **role_growvols_args** を使用して、Object Storage サービス (Swift) のディスクパーティション全体を設定することはできません。ロールベースの Ansible Playbook 設定は、ノード定義ファイルを使用してプロビジョニングされたベアメタルノードにのみ適用されます。

Red Hat Virtualization から OSPdO へのワークロードの移行

ワークロードを Red Hat Virtualization 環境から OSPdO 環境に移行することはできません。

コントロールプレーンネットワークに VLAN を使用する

TripleO は、コントロールプレーン (**ctlplane**) ネットワークでの VLAN の使用をサポートしません。

複数の Compute セル

OSPdO 環境に Compute セルを追加することはできません。

コントロールプレーン用の BGP

BGP は、OSPdO 環境のコントロールプレーンでサポートされていません。

PCI パススルーとハードウェアデバイスのコントローラー仮想マシンへの接続

SRIOV デバイスと FC SAN Storage をコントローラー仮想マシンに接続することはできません。

1.4. DIRECTOR OPERATOR デプロイメントの制限

director Operator (OSPdO) 環境には、次のサポート制限があります。

- シングルスタック IPv6 はサポートされていません。IPv4 のみが **ctlplane** ネットワークでサポートされています。
- NMState Operator は VLAN トランクを OSPdO コントローラー仮想マシンに接続できないため、専用のネットワークノードなしで VLAN プロバイダーネットワークを作成することはできません。したがって、VLAN プロバイダーネットワークを作成するには、ベアメタル上に専用の Networker ノードを作成する必要があります。詳細は、https://github.com/openstack/tripleo-heat-templates/blob/stable/wallaby/roles/Networker.yaml を参照してください。
- プロビジョニングネットワークを削除することはできません。
- SSH 接続にプロキシーを使用して Git リポジトリーと通信することはできません。
- HTTP または HTTPS を使用して Git リポジトリーに接続することはできません。

1.5. DIRECTOR OPERATOR デプロイメントに関する推奨事項

ストレージクラス

バックエンドのパフォーマンスを向上させるには、SSD/NVMe でバックアップされた低レイテンシーのストレージを使用して、コントローラー仮想マシン、クライアント Pod、およびイメージに必要な RWX/RWO ストレージクラスを作成します。

1.6. 関連情報

- Operator
- Operator のクラスターへの追加

第2章 DIRECTOR OPERATOR のインストールと準備

Red Hat OpenStack Platform (RHOSP) director Operator (OSPdO) を既存の稼働中の Red Hat Openshift Container Platform (RHOCP) クラスターにインストールします。OSPdO のインストールタスクとすべてのオーバークラウド作成タスクは、RHOCP クラスターにアクセスできるワークステーションで実行します。OSPdO をインストールした後、ベースオペレーティングシステムのデータボリュームを作成し、リモート Git リポジトリーの認証の詳細を追加する必要があります。また、ノードに root パスワードを設定することもできます。root パスワードを設定していない場合でも、**osp-controlplane-ssh-keys** シークレットで定義した SSH 鍵を使用してノードにログインすることができます。

注記

Red Hat OpenStack Platform director Operator のサポートは、アーキテクチャーが Red Hat サービスまたはテクニカルアカウントマネージャーによって承認された場合にのみ付与されます。この機能を導入する前に Red Hat にお問い合わせください。

2.1. 前提条件

- 稼働中の Red Hat Openshift Container Platform (RHOCP) クラスター、バージョン 4.12 以降。クラスターには **provisioning** ネットワークと次の Operator が含まれている必要があります。
  - **baremetal** クラスターの Operator。**baremetal** クラスター Operator を有効にする必要があります。**baremental** クラスターオペレーターの詳細は、ベアメタルクラスター Operator を参照してください。
  - OpenShift Virtualization Operator.OpenShift Virtualization Operator のインストールの詳細は、Web コンソールを使用した OpenShift Virtualization のインストール を参照してください。
  - SR-IOV Network Operator.
  - Kubernetes NMState Operator.すべての NMState CRD のインストールを完了するには、NMState インスタンスを作成する必要もあります。

    ```
    cat <<EOF | oc apply -f -
    apiVersion: nmstate.io/v1
    kind: NMState
    metadata:
      name: nmstate
      namespace: openshift-nmstate
    EOF
    ```

    Kubernetes NMState Operator のインストールの詳細は、Kubernetes NMState Operator のインストール を参照してください。
- **oc** コマンドラインツールがワークステーションにインストールされています。
- オーバークラウド用に生成された設定を保存するための OSPdO のリモート Git リポジトリー。
- Git リポジトリー用の SSH キーペアが生成され、公開キーが Git リポジトリーにアップロードされます。
- OSPdO が作成する永続ボリューム要求を満たすための次の永続ボリューム:

- 4G (**openstackclient-cloud-admin**)
- 1G (**openstackclient-hosts**)
- OSPdO が各コントローラー仮想マシンのクローンを作成する基本イメージ用に 500G。
- 最低 50 GB (コントローラー仮想マシンごと)。詳細情報は、コントローラーノードの要件を参照してください。

2.2. ベアメタルクラスター OPERATORS

インストーラープロビジョニングインフラストラクチャー (IPI) またはアシストインストール (AI) を使用してインストールする Red Hat Openshift Container Platform (RHOCP) クラスターは、**ベアメタル** プラットフォームタイプを使用し、**baremetal** クラスター Operator を有効にします。ユーザーがプロビジョニングするインフラストラクチャー (UPI) でインストールする RHOCP クラスターは、プラットフォームタイプ **none** を使用し、**baremetal** Operator が無効になる可能性があります。

クラスターのタイプが AI または IPI の場合、ベアメタルホストを管理するための Kubernetes API である **metal3** が使用されます。これは、利用可能なホストのインベントリーを **BareMetalHost** カスタムリソース定義 (CRD) のインスタンスとして維持します。ベアメタル Operator を使用して、次のタスクを実行できます。

- ホストのハードウェア詳細を検査し、それを対応する **BareMetalHost** CR に報告します。これには、CPU、RAM、ディスク、および NIC に関する情報が含まれます。
- 特定のイメージを使用してホストをプロビジョニングします。
- プロビジョニング前または後に、ホストのディスクコンテンツを消去します。

baremetal クラスター Operator が有効かどうかを確認するには、**Administration > Cluster Settings > ClusterOperators > baremetal** に移動し、**Conditions** セクションまでスクロールして **Disabled** ステータスを表示します。

RHOCP クラスターのプラットフォームタイプを確認するには、**Administration > Cluster Settings > Configuration > Infrastructure** に移動し、**YAML** ビューに切り替え、**Conditions** セクションまでスクロールして、**status.platformStatus** 値を表示します。

2.3. DIRECTOR OPERATOR のインストール

director Operator (OSPdO) をインストールするには、OSPdO の **openstack** プロジェクト (**namespace**) を作成し、プロジェクト内に次のカスタムリソース (CR) を作成する必要があります。

- **CatalogSource**: OSPdO カタログに使用するインデックスイメージを識別します。
- **OperatorGroup**: OSPdO の Operator グループを定義し、OSPdO をターゲット名前空間に制限します。
- OSPdO カタログ内の変更を追跡する **Subscription**。

手順

1. OSPdO プロジェクトを作成します。

```
$ oc new-project openstack
```

2. https://catalog.redhat.com/software/containers/search から最新の **osp-director-operator-bundle** イメージを取得します。

3. Operator Package Manager (**opm**) ツールを https://console.redhat.com/openshift/downloads からダウンロードします。

4. **opm** ツールを使用してインデックスイメージを作成します。

```
$ BUNDLE_IMG="registry.redhat.io/rhosp-rhel9/osp-director-operator-bundle:1.3.1"
$ INDEX_IMG="quay.io/<account>/osp-director-operator-index:x.y.z-a"
$ opm index add --bundles ${BUNDLE_IMG} --tag ${INDEX_IMG} -u podman --pull-tool
podman
```

5. インデックスイメージをレジストリーにプッシュします。

```
$ podman push ${INDEX_IMG}
```

6. OSPdO のインストールに必要な **CatalogSource**、**OperatorGroup**、および **Subscription** CR を設定する環境ファイル (例: **osp-director-operator.yaml**) を作成します。

7. **CatalogSource** CR を設定するには、次の設定を **osp-director-operator.yaml** に追加します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: osp-director-operator-index
  namespace: openstack
spec:
  sourceType: grpc
  image: quay.io/<account>/osp-director-operator-index:x.y.z-a
```

   Operator デプロイメントがイメージをプルできるように Quay 認証を適用する方法は、プライベートレジストリーから Operator のイメージにアクセスする を参照してください。

8. **OperatorGroup** CR を設定するには、次の設定を **osp-director-operator.yaml** に追加します。

```
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: "osp-director-operator-group"
  namespace: openstack
spec:
  targetNamespaces:
  - openstack
```

9. **サブスクリプション** CR を設定するには、次の設定を **osp-director-operator.yaml** に追加します。

```
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: osp-director-operator-subscription
  namespace: openstack
```

```
spec:
  config:
    env:
    - name: WATCH_NAMESPACE
      value: openstack,openshift-machine-api,openshift-sriov-network-operator
  source: osp-director-operator-index
  sourceNamespace: openstack
  name: osp-director-operator
```

10. **openstack** namespace 内に新規の **CatalogSource**、**Subscription**、および **Subscription** CR を作成します。

```
$ oc apply -f osp-director-operator.yaml
```

11. インストールされているオペレーターをリストして、OSPdO、**osp-director-operator.openstack** がインストールされていることを確認します。

```
$ oc get operators
NAME                                 AGE
osp-director-operator.openstack      5m
```

次のステップ

- ベースオペレーティングシステムのデータボリュームの作成

2.4. ベースオペレーティングシステムのデータボリュームの作成

コントローラー仮想マシン (VM) のベースオペレーティングシステムイメージを保存するには、Red Hat OpenShift Container Platform (RHOCP) クラスターを使用してデータボリュームを作成する必要があります。**OpenStackControlPlane** および **OpenStackVmSet** カスタムリソースを作成するときに、**baseImageVolumeName** パラメーターを使用してこのデータボリュームを指定します。

前提条件

- **virtctl** クライアントツールがワークステーションにインストールされます。このツールを Red Hat Enterprise Linux (RHEL) ワークステーションにインストールするには、次のコマンドを使用します。

```
$ sudo subscription-manager repos --enable=cnv-4.12-for-rhel-8-x86_64-rpms
$ sudo dnf install -y kubevirt-virtctl
```

- **virt-customize** クライアントツールがワークステーションにインストールされます。このツールを RHEL ワークステーションにインストールするには、次のコマンドを使用します。

```
$ dnf install -y libguestfs-tools-c
```

手順

1. Red Hat Customer Portal の 製品ダウンロード セクションから RHEL 9.2 QCOW2 イメージをワークステーションにダウンロードします。

2. オプション: カスタム CA 証明書を追加します。

```
$ sudo -s
$ export LIBGUESTFS_BACKEND=direct
$ virt-copy-in -a <local_path_to_image> <ca_certificate>.pem /etc/pki/ca-
trust/source/anchors/
```

Identity Service の LDAP 通信を保護したり、RHOSP 以外のシステムと通信したりするために、カスタム CA 証明書を追加することができます。

3. イメージをカスタマイズして、予測可能なネットワークインターフェイス名を割り当てるスクリプトを作成します。

```
#!/bin/bash
set -eux

if [ -e /etc/kernel/cmdline ]; then
  echo 'Updating /etc/kernel/cmdline'
  sed -i -e "s/^\(.*\)net\.ifnames=0\s*\(.*\)/\1\2/" /etc/kernel/cmdline
fi

source /etc/default/grub
if grep -q "net.ifnames=0" <<< "$GRUB_CMDLINE_LINUX"; then
  echo 'Updating /etc/default/grub'
  sed -i -e "s/^\(GRUB_CMDLINE_LINUX=.*\)net\.ifnames=0\s*\(.*\)/\1\2/" /etc/default/grub
fi
if [ "$GRUB_ENABLE_BLSCFG" == "true" ]; then
  echo 'Fixing BLS entries'
  find /boot/loader/entries -type f -exec sed -i -e "s/^\(.*\)net\.ifnames=0\s*\(.*\)/\1\2/" {} \;
fi
# Always do this, on RHEL8 with BLS we still need it as the BLS entry uses $kernelopts from
grubenv
echo 'Running grub2-mkconfig'
grub2-mkconfig -o /etc/grub2.cfg
grub2-mkconfig -o /etc/grub2-efi.cfg
rm -f /etc/sysconfig/network-scripts/ifcfg-ens* /etc/sysconfig/network-scripts/ifcfg-eth*
update-ca-trust extract
```

4. イメージのカスタマイズスクリプトを実行します。

```
$ sudo -s
$ export LIBGUESTFS_BACKEND=direct
$ chmod 755 customize_image.sh
$ virt-customize -a <local_path_to_image> --run customize_image.sh --truncate
/etc/machine-id
```

5. **virtctl** を使用してイメージを OpenShift Virtualization にアップロードします。

```
$ virtctl image-upload dv <datavolume_name> -n openstack \
 --size=<size> --image-path=<local_path_to_image> \
 --storage-class <storage_class> --access-mode <access_mode> --insecure
```

- **<datavolume_name>** をデータボリュームの名前 (例: **openstack-base-img**) に置き換えます。
- **<size>** を、環境に必要なデータボリュームのサイズ (たとえば、**500Gi**) に置き換えます。最小サイズは 500 GB です。

- **<storage_class>** をクラスターの必要なストレージクラスに置き換えます。次のコマンドを使用して、利用可能なストレージクラスを取得します。

```
$ oc get storageclass
```

- **<access_mode>** を、PVC のアクセスモードに置き換えます。デフォルト値は **ReadWriteOnce** です。

関連情報

- 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

次のステップ

- リモート Git リポジトリーの認証情報の追加

2.5. リモート GIT リポジトリーの認証情報の追加

director Operator (OSPdO) はレンダリングされた Ansible Playbook をリモート Git リポジトリーに保管し、このリポジトリーを使用してオーバークラウドの設定に対する変更を追跡します。SSH 認証をサポートする Git リポジトリーであれば、どのリポジトリーでも使用できます。Git リポジトリーの詳細を、**git-secret** という名前の Red Hat OpenShift Platform (RHOCP) Secret リソースとして指定する必要があります。

前提条件

- OSPdO Git リポジトリー用の SSH キーペアの秘密鍵。

手順

1. **git-secret** シークレットリソースを作成します。

```
$ oc create secret generic <secret_name> -n <namespace> \
 --from-file=git_ssh_identity=<path_to_private_SSH_key> \
 --from-literal=git_url=<git_server_URL>
```

   - **<secret_name>** をシークレットの名前 (この場合は **git-secret** に置き換えます。
   - **<namespace>** を **openstack** などにシークレットを作成するネームスペースの名前に置き換えます。
   - **<path_to_private_SSH_key>** を Git リポジトリーにアクセスするための秘密鍵へのパスに置き換えます。
   - **<git_server_URL>** を、OSPdO 設定を保存する git リポジトリーの SSH URL (**ssh://<user>@<server>:2202/repo.git** など) に置き換えます。

2. Secret リソースが作成されたことを確認します。

```
$ oc get secret/git-secret -n openstack
```

関連情報

- Pod への機密性の高いデータの提供

次のステップ

- director Operator とネットワークを構築する

2.6. ノードの ROOT パスワードの設定

各ノードのパスワードを使用して **root** ユーザーにアクセスするには、**userpassword** という名前の **Secret** リソースに **root** パスワードを設定します。ノードの root パスワードの設定はオプションです。**root** パスワードを設定していない場合には、**osp-controlplane-ssh-keys** シークレットで定義した SSH 鍵を使用してノードにログインすることができます。

注記

root パスワードを設定する場合は、**OpenStackControlPlane** および **OpenStackBaremetalSet** カスタムリソースを作成するときに、**passwordSecret** パラメーターを使用してこの **Secret** リソースの名前を指定する必要があります。このガイドの例では、**Secret** リソース名 **userpassword** を使用します。

手順

1. 選択したパスワードを base64 値に変換します。

```
$ echo -n "p@ssw0rd!" | base64
cEBzc3cwcmQh
```

重要

-n オプションは、echo 出力から末尾の改行を削除します。

2. ワークステーションに **openstack-userpassword.yaml** という名前のファイルを作成します。ファイルに、Secret の以下のリソース仕様を追加します。

```
apiVersion: v1
kind: Secret
metadata:
  name: <secret_name>
  namespace: openstack
data:
  NodeRootPassword: "<password>"
```

   - **<secret_name>** をこのシークレットリソースの名前 (**userpassword** など) に置き換えます。
   - **<password>** を、base64 でエンコードされたパスワードに置き換えます。

3. **userpassword** シークレットを作成します。

```
$ oc create -f openstack-userpassword.yaml -n openstack
```

関連情報

- Pod への機密性の高いデータの提供

次のステップ

- director Operator とネットワークを構築する

第3章 DIRECTOR OPERATOR とネットワークを構築する

OpenShift Virtualization ワーカーノード上にネットワークとブリッジを作成し、仮想マシン (VM) をこれらのネットワークに接続するには、**OpenStackNetConfig** カスタムリソース (CR) を定義し、オーバークラウドネットワークのすべてのサブネットを指定します。オーバークラウド用にコントロールプレーンネットワークを 1 つ作成する必要があります。オプションで追加のネットワークを作成して、コンポーザブルネットワークのネットワーク分離を実装することもできます。

3.1. OPENSTACKNETCONFIG CRD を使用したオーバークラウドネットワークの作成

OpenStackNetConfig CRD を使用して、オーバークラウド用に少なくとも 1 つのコントロールプレーンネットワークを定義する必要があります。オプションで、VLAN ネットワークを定義して、**InternalAPI**、**Storage**、**External** などの設定可能なネットワークのネットワーク分離を作成することもできます。各ネットワーク定義には、IP アドレスの割り当てと、**OpenStackNetAttachment** CRD のマッピング情報が含まれている必要があります。OpenShift Virtualization は、ネットワーク定義を使用して、仮想マシン (VM) をコントロールプレーンおよび VLAN ネットワークに接続します。

ヒント

OpenStackNetConfig CRD 定義と仕様スキーマを表示するには、次のコマンドを使用します。

```
$ oc describe crd openstacknetconfig

$ oc explain openstacknetconfig.spec
```

手順

1. ワークステーション上に **openstacknetconfig.yaml** という名前のファイルを作成します。
2. 次の設定を **openstacknetconfig.yaml** に追加して、**OpenStackNetConfig** カスタムリソース (CR) を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
```

3. ネットワークに必要なブリッジのネットワークアタッチメント定義を設定します。たとえば、次の設定を **openstacknetconfig.yaml** に追加して RHOSP ブリッジのネットワーク接続定義 **br-osp** を作成し、**nodeNetworkConfigurationPolicy** オプションを設定して Linux ブリッジを作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp: 1
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
```

```
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0 2
          description: Linux bridge with enp6s0 as a port
          name: br-osp 3
          state: up
          type: linux-bridge
          mtu: 1500 4
  # optional DnsServers list
  dnsServers:
  - 192.168.25.1
  # optional DnsSearchDomains list
  dnsSearchDomains:
  - osptest.test.metalkube.org
  - some.other.domain
  # DomainName of the OSP environment
  domainName: osptest.test.metalkube.org
```

1 **br-osp** のネットワーク接続定義。

2 各ホストに接続する NIC/イーサネットデバイス。

3 インターフェイスの名前。

4 1つのネットワークパケットで転送できるデータの最大量。

4. オプション: ブリッジにジャンボフレームを使用するには、ジャンボフレームを使用するようにブリッジインターフェイスを設定し、ブリッジの **mtu** の値を更新します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
          - bridge:
              options:
                stp:
                  enabled: false
              port:
              - name: enp6s0
            description: Linux bridge with enp6s0 as a port
            name: br-osp
            state: up
```

```
        type: linux-bridge
        mtu: 9000 (1)
      - name: enp6s0 (2)
        description: Configuring enp6s0 on workers
        type: ethernet
        state: up
        mtu: 9000
...
```

(1) ジャンボフレームを使用して、1つのネットワークパケットで転送できるデータの最大量を更新します。

(2) ジャンボフレームを使用するようにブリッジインターフェイスを設定します。

5. 各オーバークラウドネットワークを定義します。次の例では、**InternalAPI** トラフィック用にコントロールプレーンネットワークと分離されたネットワークを作成します。

```
spec:
  ...
  networks:
  - name: Control (1)
    nameLower: ctlplane (2)
    subnets: (3)
    - name: ctlplane (4)
      ipv4: (5)
        allocationEnd: 172.22.0.250
        allocationStart: 172.22.0.100
        cidr: 172.22.0.0/24
        gateway: 172.22.0.1
      attachConfiguration: br-osp (6)
  - name: InternalApi (7)
    nameLower: internal_api
    mtu: 1350
    subnets:
    - name: internal_api
      attachConfiguration: br-osp
      vlan: 20 (8)
      ipv4:
        allocationEnd: 172.17.0.250
        allocationStart: 172.17.0.10
        cidr: 172.17.0.0/24
...
```

(1) ネットワークの名前 (例: **Control**)。

(2) ネットワーク名の小文字バージョン (**ctlplane** など)。

(3) サブネットの仕様。

(4) サブネットの名前 (**ctlplane** など)。

(5) **allocationStart**、**allocationEnd**、**cidr**、**gateway**、オプションのルートリスト (**destination** と **nexthop** を含む) を含む ipv4 サブネットの詳細

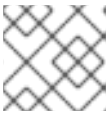
6 ネットワークを接続するネットワーク接続定義。この例では、RHOSP ブリッジ **br-osp** が各ワーカーの NIC に接続されています。

7 コンポーザブルネットワークのネットワーク定義。デフォルトの RHOSP ネットワークを使用するには、ネットワークごとに **OpenStackNetConfig** リソースを作成する必要があります。デフォルトの RHOSP ネットワークは、デフォルトの Red Hat OpenStack Platform ネットワーク を参照してください。異なるネットワークを使用するには、カスタム **network_data.yaml** ファイルを作成する必要があります。カスタムの **network_data.yaml** ファイルの作成は、オーバークラウドネットワークの設定 を参照してください。

8 ネットワーク VLAN。デフォルトの RHOSP ネットワークは、デフォルトの Red Hat OpenStack Platform ネットワーク を参照してください。**OpenStackNetConfig** CRD を使用した仮想マシンブリッジングの詳細は、OpenStackNetConfig CRD を使用した仮想マシンブリッジングについて を参照してください。

6. オプション: 特定のノード上のネットワーク用に静的 IP アドレスを予約します。

```
spec:
  ...
  reservations:
    controller-0:
      ipReservations:
        ctlplane: 172.22.0.120
    compute-0:
      ipReservations:
        ctlplane: 172.22.0.140
        internal_api: 172.17.0.40
        storage: 172.18.0.40
        tenant: 172.20.0.40
```



注記

予約は、自動生成された IP アドレスよりも優先されます。

7. **openstacknetconfig.yaml** 定義ファイルを保存します。
8. オーバークラウドネットワークを作成します。

```
$ oc create -f osnetconfig.yaml -n openstack
```

9. オーバークラウドネットワークが作成されたことを確認するには、オーバークラウドネットワークのリソースを表示します。

```
$ oc get openstacknetconfig/openstacknetconfig
```

10. **OpenStackNetConfig** API と子リソースを表示します。

```
$ oc get openstacknetconfig/openstacknetconfig -n openstack
$ oc get openstacknetattachment -n openstack
$ oc get openstacknet -n openstack
```

エラーが表示された場合は、基礎となる **network-attach-definition** とノードのネットワーク設定ポリシーを確認してください。

```
$ oc get network-attachment-definitions -n openstack
$ oc get nncp
```

次のステップ

- director Operator を使用したオーバークラウドのカスタマイズ

3.1.1. デフォルトの Red Hat OpenStack Platform ネットワーク

Network	VLAN	CIDR	割り当て
External	10	10.0.0.0/24	10.0.0.10 - 10.0.0.250
InternalApi	20	172.17.0.0/24	172.17.0.10 - 172.17.0.250
Storage	30	172.18.0.0/24	172.18.0.10 - 172.18.0.250
StorageMgmt	40	172.19.0.0/24	172.19.0.10 - 172.19.250
テナント	50	172.20.0.0/24	172.20.0.10 - 172.20.0.250

3.2. OPENSTACKNETCONFIG CRD を使用した仮想マシンのブリッジングについて

OpenStackVMSet CRD を使用して仮想マシン (VM) を作成する場合は、これらの VM を関連する Red Hat OpenStack Platform (RHOSP) ネットワークに接続する必要があります。**OpenStackNetConfig** CRD を使用すると、Red Hat OpenShift Container Platform (RHOCP) ワーカーノード上に必要なブリッジを作成し、コントローラー VM を RHOSP オーバークラウドネットワークに接続できます。RHOSP をデプロイメントするには専用の NIC が必要です。

OpenStackNetConfig CRD には、**nodeNetworkConfigurationPolicy** のハッシュである **attachConfigurations** オプションが含まれています。**OpenStackNetConfig** カスタムリソース (CR) で指定されたそれぞれの **attachConfiguration** は、**NetworkAttachmentDefinition** オブジェクトを作成し、ネットワークインターフェイスデータを RHOCP クラスターの **NodeNetworkConfigurationPolicy** リソースに渡します。**NodeNetworkConfigurationPolicy** リソースは、**nmstate** API を使用して、各 RHOCP ワーカーノードのネットワーク設定の最終状態を設定します。各ネットワークの **NetworkAttachmentDefinition** オブジェクトは、Multus CNI プラグイン設定を定義します。**NetworkAttachmentDefinition** オブジェクトの VLAN ID を指定すると、Multus CNI プラグインによってブリッジ上の **vlan-filtering** が有効になります。**OpenStackNetConfig** で設定された各ネットワークは、**attachConfigurations** の 1 つを参照します。VM 内には、ネットワークごとに 1 つのインターフェイスがあります。

次の例では、**br-ospattachConfiguration** を作成し、Linux ブリッジを作成し、そのブリッジを各ワーカーの NIC に接続するように、**nodeNetworkConfigurationPolicy** オプションを設定します。この設定を適用すると、**NodeNetworkConfigurationPolicy** オブジェクトは、必要な最終状態に一致するように

各 RHOCP ワーカーノードを設定します。各ワーカーには、各ホストの **enp6s0** NIC に接続される **br-osp** という名前の新しいブリッジが含まれます。すべての RHOSP コントローラー VM は、コントロールプレーンネットワークトラフィックの **br-osp** ブリッジに接続できます。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
          - bridge:
              options:
                stp:
                  enabled: false
              port:
              - name: enp6s0
            description: Linux bridge with enp6s0 as a port
            name: br-osp
            state: up
            type: linux-bridge
            mtu: 1500
…
  networks:
  - name: Control
    nameLower: ctlplane
    subnets:
    - name: ctlplane
      ipv4:
        allocationEnd: 192.168.25.250
        allocationStart: 192.168.25.100
        cidr: 192.168.25.0/24
        gateway: 192.168.25.1
      attachConfiguration: br-osp
```

VLAN 20 を介して内部 API ネットワークを指定する場合は、**attachConfiguration** オプションを設定して、各 RHOCP ワーカーノードのネットワーク設定を変更し、VLAN を既存の **br-osp** ブリッジに接続できます。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
  ...
  networks:
  ...
  - isControlPlane: false
```

```
    mtu: 1500
    name: InternalApi
    nameLower: internal_api
    subnets:
    - attachConfiguration: br-osp
      ipv4:
        allocationEnd: 172.17.0.250
        allocationStart: 172.17.0.10
        cidr: 172.17.0.0/24
        gateway: 172.17.0.1
        routes:
        - destination: 172.17.1.0/24
          nexthop: 172.17.0.1
        - destination: 172.17.2.0/24
          nexthop: 172.17.0.1
      name: internal_api
      vlan: 20
```

br-osp はすでに存在し、各ホストの **enp6s0** NIC に接続されているので、ブリッジ自体に変更は加えられません。ただし、**InternalAPI OpenStackNet** は VLAN 20 をこのネットワークに関連付けます。これは、RHOSP コントローラー VM が内部 API ネットワークトラフィックの **br-osp** ブリッジ上の VLAN 20 に接続できることを意味します。

OpenStackVMSet CRD を使用して VM を作成すると、VM は各ネットワークに接続された複数の Virtio デバイスを使用します。OpenShift Virtualization は、**default** ネットワーク (常に最初にくるインターフェイス) を除き、ネットワーク名をアルファベット順に並べ替えます。たとえば、**OpenStackNetConfig** を使用してデフォルトの RHOSP ネットワークを作成すると、コントローラー VM に対して次のインターフェイス設定が生成されます。

```
interfaces:
  - masquerade: {}
    model: virtio
    name: default
  - bridge: {}
    model: virtio
    name: ctlplane
  - bridge: {}
    model: virtio
    name: external
  - bridge: {}
    model: virtio
    name: internalapi
  - bridge: {}
    model: virtio
    name: storage
  - bridge: {}
    model: virtio
    name: storagemgmt
  - bridge: {}
    model: virtio
    name: tenant
```

この設定により、コントローラーノードの以下のネットワークとインターフェイスのマッピングが作成されます。

表3.1 デフォルトのネットワークからインターフェイスへのマッピング

Network	インターフェイス
default	nic1
ctlplane	nic2
external	nic3
internalapi	nic4
ストレージ	nic5
storagemgmt	nic6
tenant	nic7

注記

OpenStackVMSet で使用されるロール NIC テンプレートは自動生成されます。カスタム NIC テンプレートファイル **<role>-nic-template.j2** (たとえば、**controller-nic-template.j2**) のデフォルト設定を上書きできます。カスタム NIC ファイルを、OpenShift ConfigMap オブジェクトを使用して実装されるオーバークラウド設定を含む tarball ファイルに追加する必要があります。詳細は、第 4 章director Operator を使用したオーバークラウドのカスタマイズ。

関連情報

- ノードのネットワーク設定の更新

3.3. OPENSTACKNETCONFIG カスタムリソースファイルの例

次の **OpenStackNetConfig** カスタムリソース (CR) ファイルの例では、デフォルトの RHOSP デプロイメント用のコントロールプレーンネットワークと分離された VLAN ネットワークを含むオーバークラウドネットワークを定義します。この例では、特定のノード上のネットワーク用に静的 IP アドレスも予約します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
          - bridge:
              options:
                stp:
```

```
              enabled: false
            port:
            - name: enp7s0
          description: Linux bridge with enp7s0 as a port
          name: br-osp
          state: up
          type: linux-bridge
          mtu: 9000
        - name: enp7s0
          description: Configuring enp7s0 on workers
          type: ethernet
          state: up
          mtu: 9000
    br-ex:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
          - bridge:
              options:
                stp:
                  enabled: false
              port:
              - name: enp6s0
            description: Linux bridge with enp6s0 as a port
            name: br-ex
            state: up
            type: linux-bridge
            mtu: 1500
  # optional DnsServers list
  dnsServers:
  - 172.22.0.1
  # optional DnsSearchDomains list
  dnsSearchDomains:
  - osptest.test.metalkube.org
  - some.other.domain
  # DomainName of the OSP environment
  domainName: osptest.test.metalkube.org
networks:
- name: Control
  nameLower: ctlplane
  subnets:
  - name: ctlplane
    ipv4:
      allocationEnd: 172.22.0.250
      allocationStart: 172.22.0.10
      cidr: 172.22.0.0/24
      gateway: 172.22.0.1
    attachConfiguration: br-osp
- name: InternalApi
  nameLower: internal_api
  mtu: 1350
  subnets:
  - name: internal_api
    attachConfiguration: br-osp
```

```
    vlan: 20
    ipv4:
      allocationEnd: 172.17.0.250
      allocationStart: 172.17.0.10
      cidr: 172.17.0.0/24
      gateway: 172.17.0.1
      routes:
      - destination: 172.17.1.0/24
        nexthop: 172.17.0.1
      - destination: 172.17.2.0/24
        nexthop: 172.17.0.1
- name: External
  nameLower: external
  subnets:
  - name: external
    ipv4:
      allocationEnd: 10.0.0.250
      allocationStart: 10.0.0.10
      cidr: 10.0.0.0/24
      gateway: 10.0.0.1
    attachConfiguration: br-ex
- name: Storage
  nameLower: storage
  mtu: 1500
  subnets:
  - name: storage
    ipv4:
      allocationEnd: 172.18.0.250
      allocationStart: 172.18.0.10
      cidr: 172.18.0.0/24
    vlan: 30
    attachConfiguration: br-osp
- name: StorageMgmt
  nameLower: storage_mgmt
  mtu: 1500
  subnets:
  - name: storage_mgmt
    ipv4:
      allocationEnd: 172.19.0.250
      allocationStart: 172.19.0.10
      cidr: 172.19.0.0/24
    vlan: 40
    attachConfiguration: br-osp
- name: Tenant
  nameLower: tenant
  vip: False
  mtu: 1500
  subnets:
  - name: tenant
    ipv4:
      allocationEnd: 172.20.0.250
      allocationStart: 172.20.0.10
      cidr: 172.20.0.0/24
    vlan: 50
    attachConfiguration: br-osp
reservations:
```

```
compute-0:
  ipReservations:
    ctlplane: 172.22.0.140
    internal_api: 172.17.0.40
    storage: 172.18.0.40
    tenant: 172.20.0.40
  macReservations: {}
controller-0:
  ipReservations:
    ctlplane: 172.22.0.120
    external: 10.0.0.20
    internal_api: 172.17.0.20
    storage: 172.18.0.20
    storage_mgmt: 172.19.0.20
    tenant: 172.20.0.20
  macReservations: {}
controller-1:
  ipReservations:
    ctlplane: 172.22.0.130
    external: 10.0.0.30
    internal_api: 172.17.0.30
    storage: 172.18.0.30
    storage_mgmt: 172.19.0.30
    tenant: 172.20.0.30
  macReservations: {}

//The key for the ctlplane VIPs
controlplane:
  ipReservations:
    ctlplane: 172.22.0.110
    external: 10.0.0.10
    internal_api: 172.17.0.10
    storage: 172.18.0.10
    storage_mgmt: 172.19.0.10
  macReservations: {}
openstackclient-0:
  ipReservations:
    ctlplane: 172.22.0.251
    external: 10.0.0.251
    internal_api: 172.17.0.251
  macReservations: {}
```

第4章 DIRECTOR OPERATOR を使用したオーバークラウドのカスタマイズ

オーバークラウドのデプロイメントに含める Heat テンプレートと環境ファイルを作成することで、オーバークラウドをカスタマイズしたり、特定の機能を有効にしたりできます。Director Operator (OSPdO) オーバークラウドデプロイメントでは、オーバークラウドデプロイメントを実行する前に、これらのファイルを **ConfigMap** オブジェクトに保存します。

4.1. オーバークラウド設定へのカスタムテンプレートの追加

director Operator (OSPdO) は、各ノードで Red Hat OpenStack Platform (RHOSP) ソフトウェアを設定する準備が整うと、オーバークラウドヒートテンプレートのコアセットをプロビジョニングノードに適用する Ansible Playbook に変換します。独自のカスタム heat テンプレートおよびカスタムロールファイルをオーバークラウドデプロイメントに追加するには、テンプレートファイルを tarball ファイルにアーカイブし、tarball ファイルのバイナリーコンテンツを **tripleo-tarball-config** という名前の OpenShift **ConfigMap** に含める必要があります。この tarball ファイルには、テンプレートのコアセットを拡張するための複雑なディレクトリー構造を含めることができます。OSPdO は、tarball ファイルから、heat テンプレートのコアセットと同じディレクトリーにファイルとディレクトリーを展開します。カスタムテンプレートのいずれかがコアコレクションのテンプレートと名前が同じ場合には、カスタムテンプレートはコアテンプレートを上書きします。

注記

環境ファイル内のすべての参照は、tarball が抽出される TripleO heat テンプレートに関連している必要があります。

前提条件

- プロビジョニングされたノードに適用するカスタムオーバークラウドテンプレート。

手順

1. カスタムテンプレートの場所に移動します。

```
$ cd ~/custom_templates
```

2. テンプレートを gzip 圧縮された tarball にアーカイブします。

```
$ tar -cvzf custom-config.tar.gz *.yaml
```

3. **tripleo-tarball-config** CR を作成し、tarball をデータとして使用します。

```
$ oc create configmap tripleo-tarball-config --from-file=custom-config.tar.gz -n openstack
```

4. **ConfigMap** CR が作成されたことを確認します。

```
$ oc get configmap/tripleo-tarball-config -n openstack
```

関連情報

- 設定マップの作成および使用

- heat テンプレートの概要

次のステップ

- オーバークラウド設定へのカスタム環境ファイルの追加

4.2. オーバークラウド設定へのカスタム環境ファイルの追加

オーバークラウドで機能を有効にしたりパラメーターを設定するには、オーバークラウドのデプロイメントに環境ファイルを含める必要があります。director Operator (OSPdO) は **heat-env-config** という名前の **ConfigMap** オブジェクトを使用して、環境ファイルを保存および取得します。**ConfigMap** オブジェクトは、環境ファイルを次の形式で保存します。

```
...
data:
  <environment_file_name>: |+
    <environment_file_contents>
```

たとえば、次の **ConfigMap** には 2 つの環境ファイルが含まれています。

```
...
data:
  network_environment.yaml: |+
    parameter_defaults:
      ComputeNetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
  cloud_name.yaml: |+
    parameter_defaults:
      CloudDomain: ocp4.example.com
      CloudName: overcloud.ocp4.example.com
      CloudNameInternal: overcloud.internalapi.ocp4.example.com
      CloudNameStorage: overcloud.storage.ocp4.example.com
      CloudNameStorageManagement: overcloud.storagemgmt.ocp4.example.com
      CloudNameCtlplane: overcloud.ctlplane.ocp4.example.com
```

ディレクトリーからオーバークラウドデプロイメントの一部として追加する **ConfigMap** オブジェクトにカスタム環境ファイルのセットをアップロードします。

前提条件

- オーバークラウドデプロイメント用のカスタム環境ファイル。

手順

1. **heat-env-config ConfigMap** オブジェクトを作成します。

```
$ oc create configmap -n openstack heat-env-config \
 --from-file=~/<dir_custom_environment_files>/ \
 --dry-run=client -o yaml | oc apply -f -
```

- **<dir_custom_environment_files>** をオーバークラウドのデプロイメントで使用する環境ファイルが含まれるディレクトリーに置き換えます。**ConfigMap** オブジェクトは、これらを個別の **data** エントリーとして保存します。

2. **heat-env-config ConfigMap** オブジェクトに必要な環境ファイルがすべて含まれていることを確認します。

```
$ oc get configmap/heat-env-config -n openstack
```

4.3. 関連情報

- heat テンプレートの概要
- 環境ファイル
- 設定マップの作成および使用

第5章 DIRECTOR OPERATOR を使用したオーバークラウドノードの作成

Red Hat OpenStack Platform (RHOSP) オーバークラウドは、コントロールプレーンサービスを提供するコントロールノードや、コンピュートリソースを提供する Compute ノードなど、複数のノードで設定されます。高可用性に対応したオーバークラウドには、3 つのコントローラーノードと少なくとも 1 つの Compute ノードが必要です。**OpenStackControlPlane** カスタムリソース定義 (CRD) を使用してコントローラーノードを作成し、**OpenStackBaremetalSet** CRD を使用して Compute ノードを作成できます。

注記

Red Hat OpenShift Container Platform (RHOCP) は、RHOCP ワーカーノードの問題を自動検出しません。また、ワーカーノードに障害が発生したり問題が発生した場合に、RHOSP コントローラー VM をホストするワーカーノードの自動回復を実行しません。ホストワーカーノードに障害が発生したときにコントローラー VM Pod を自動的に再配置するには、RHOCP クラスターでヘルスチェックを有効にする必要があります。RHOCP ワーカーノードの問題を自動検出する方法は、マシンヘルスチェックのデプロイ を参照してください。

5.1. OPENSTACKCONTROLPLANE CRD を使用したコントロールプレーンの作成

Red Hat OpenStack Platform (RHOSP) コントロールプレーンには、オーバークラウドを管理する RHOSP サービスが含まれています。デフォルトのコントロールプレーンは 3 つのコントローラーノードで設定されます。設定可能なロールを使用して、専用コントローラー仮想マシン (VM) 上のサービスを管理できます。コンポーザブルロールの詳細は、コンポーザブルサービスとカスタムロール を参照してください。

OpenStackControlPlane カスタムリソース (CR) を定義して、コントローラーノードを OpenShift Virtualization 仮想マシン (VM) として作成します。

ヒント

OpenStackControlPlane CRD 定義と仕様スキーマを表示するには、次のコマンドを使用します。

```
$ oc describe crd openstackcontrolplane

$ oc explain openstackcontrolplane.spec
```

前提条件

- **OpenStackNetConfig** CR を使用して、コントロールプレーンネットワークと追加の分離ネットワークを作成している。

手順

1. ワークステーションに **openstack-controller.yaml** という名前のファイルを作成します。コントローラーノードのリソース仕様を含めます。次の例では、3 つのコントローラーノードで設定されるコントロールプレーンの仕様を定義します。

```
apiVersion: osp-director.openstack.org/v1beta2
```

```
kind: OpenStackControlPlane
metadata:
  name: overcloud (1)
  namespace: openstack (2)
spec: (3)
  openStackClientNetworks:
      - ctlplane
      - internal_api
      - external
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword (4)
  virtualMachineRoles:
    Controller:
      roleName: Controller
      roleCount: 3
      networks:
        - ctlplane
        - internal_api
        - external
        - tenant
        - storage
        - storage_mgmt
      cores: 12
      memory: 64
      rootDisk:
        diskSize: 500
        baseImageVolumeName: openstack-base-img (5)
        storageClass: host-nfs-storageclass (6)
        storageAccessMode:  ReadWriteMany
        storageVolumeMode: Filesystem
      # optional configure additional discs to be attached to the VMs,
      # need to be configured manually inside the VMs where to be used.
      additionalDisks:
        - name: datadisk
          diskSize: 500
          storageClass: host-nfs-storageclass
          storageAccessMode:  ReadWriteMany
          storageVolumeMode: Filesystem
  openStackRelease: "17.1"
```

(1) オーバークラウドのコントロールプレーンの名前 (例: **overcloud**)。

(2) OSPdO 名前空間 (例: **openstack**)。

(3) コントロールプレーンの設定。

(4) オプション: パスワードを使用してユーザーに各ノードでの root アクセスを提供する **Secret** リソース。

(5) Controller VM の基本オペレーティングシステムイメージを保存するデータボリュームの名前。データボリュームの作成の詳細は、ベースオペレーティングシステムのデータボリュームの作成 を参照してください。

(6) Red Hat OpenShift Container Platform (RHOCP)ストレージの設定は、動的プロビジョニング を参照してください。

2. **openstack-controller.yaml** ファイルを保存します。
3. コントロールプレーンを作成します。

```
$ oc create -f openstack-controller.yaml -n openstack
```

4. RHOCP が **OpenStackControlPlane** CR に関連するリソースを作成するまで待ちます。OSPdO は、リモートシェルを通じてアクセスして RHOSP コマンドを実行できる **OpenStackClient** Pod も作成します。

検証

1. コントロールプレーンのリソースを表示します。

```
$ oc get openstackcontrolplane/overcloud -n openstack
```

2. **OpenStackVMSet** リソースを表示して、コントロールプレーンの仮想マシンセットの作成を確認します。

```
$ oc get openstackvmsets -n openstack
```

3. VM を表示して、コントロールプレーン OpenShift Virtualization VM の作成を確認します。

```
$ oc get virtualmachines -n openstack
```

4. **openstackclient** リモートシェルにアクセスできるかをテストします。

```
$ oc rsh -n openstack openstackclient
```

5.2. OPENSTACKBAREMETALSET CRD を使用したコンピュートノードの作成

Compute ノードは Red Hat OpenStack Platform (RHOSP) 環境にコンピュートリソースを提供します。オーバークラウドには Compute ノードが少なくとも 1 台必要で、デプロイメント後にCompute ノードの数をスケーリングできます。

OpenStackBaremetalSet カスタムリソース (CR) を定義して、Red Hat OpenShift Container Platform (RHOCP) が管理するベアメタルマシンから Compute ノードを作成します。

ヒント

OpenStackBareMetalSet CRD 定義と仕様スキーマを表示するには、次のコマンドを使用します。

```
$ oc describe crd openstackbaremetalset

$ oc explain openstackbaremetalset.spec
```

前提条件

- **OpenStackNetConfig** CR を使用して、コントロールプレーンネットワークと追加の分離ネットワークを作成している。

- **OpenStackControlPlane** CRD を使用してコントロールプレーンを作成している。

手順

1. ワークステーションに **openstack-compute.yaml** という名前のファイルを作成します。Compute ノードのリソース仕様を含めます。次の例では、1 つの Compute ノードの仕様を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: compute 1
  namespace: openstack 2
spec: 3
  count: 1
  baseImageUrl: http://<source_host>/rhel-9.2-x86_64-kvm.qcow2
  deploymentSSHSecret: osp-controlplane-ssh-keys
  # If you manually created an OpenStackProvisionServer, you can use it here,
  # otherwise director Operator will create one for you (with `baseImageUrl` as the image that
it server)
  # to use with this OpenStackBaremetalSet
  # provisionServerName: openstack-provision-server
  ctlplaneInterface: enp2s0
  networks:
    - ctlplane
    - internal_api
    - tenant
    - storage
  roleName: Compute
  passwordSecret: userpassword 4
```

1. Compute ノードのベアメタルセットの名前 (例: **compute**)。
2. OSPdO 名前空間 (例: **openstack**)。
3. Compute ノードの設定。
4. オプション: パスワードを使用してユーザーに各ノードでの root アクセスを提供する **Secret** リソース。

2. **openstack-compute.yaml** ファイルを保存します。
3. Compute ノードを作成します。

```
$ oc create -f openstack-compute.yaml -n openstack
```

検証

1. Compute ノードのリソースを表示します。

```
$ oc get openstackbaremetalset/compute -n openstack
```

2. RHOCP が管理するベアメタルマシンを表示し、Compute ノードの作成を確認します。

```
$ oc get baremetalhosts -n openshift-machine-api
```

5.3. OPENSTACKPROVISIONSERVER CRD を使用したプロビジョニングサーバーの作成

プロビジョニングサーバーは、Red Hat OpenStack Platform (RHOSP) の Compute ノードをプロビジョニングするための特定の Red Hat Enterprise Linux (RHEL) QCOW2 イメージを提供します。**OpenStackProvisionServer** CR は、作成した **OpenStackBaremetalSet** CR に対して自動的に作成されます。**OpenStackProvisionServer** CR を手動で作成し、作成する **OpenStackBaremetalSet** CR に名前を指定できます。

OpenStackProvisionServer CRD は、特定の RHEL QCOW2 イメージ用に Red Hat OpenShift Container Platform (RHOCP) プロビジョニングネットワーク上に Apache サーバーを作成します。

手順

1. ワークステーションに **openstack-provision.yaml** という名前のファイルを作成します。プロビジョニングサーバーのリソース仕様を含めます。次の例では、特定の RHEL 9.2 QCOW2 イメージを使用してプロビジョニングサーバーの仕様を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackProvisionServer
metadata:
  name: openstack-provision-server 1
  namespace: openstack 2
spec:
  baseImageUrl: http://<source_host>/rhel-9.2-x86_64-kvm.qcow2 3
  port: 8080 4
```

1. **OpenStackProvisionServer** CR を識別する名前。
2. OSPdO 名前空間 (例: **openstack**)。
3. Provisioing サーバーの RHEL QCOW2 イメージの初期ソースを設定します。イメージは、サーバーの作成時にこのリモートソースからダウンロードされます。
4. プロビジョニングサーバーポート。デフォルトでは 8080 に設定されます。特定のポート設定に合わせて変更できます。

OpenStackProvisionServer CR の設定に使用できる値の説明は、**OpenStackProvisionServer** CRD 仕様スキーマを参照してください。

```
$ oc describe crd openstackprovisionserver
```

2. **openstack-provision.yaml** ファイルを保存します。
3. Provisioning サーバーを作成します。

```
$ oc create -f openstack-provision.yaml -n openstack
```

4. プロビジョニングサーバーのリソースが作成されたことを確認します。

```
$ oc get openstackprovisionserver/openstack-provision-server -n openstack
```

第6章 DIRECTOR OPERATOR を使用したオーバークラウドの設定とデプロイメント

オーバークラウドに仮想ノードとベアメタルノードをプロビジョニングした後、オーバークラウドノードを設定できます。**OpenStackConfigGenerator** リソースを作成して Ansible Playbook を生成し、ノードを Red Hat Customer Portal または Red Hat Satellite に登録してから、**OpenStackDeploy** リソースを作成して設定をノードに適用する必要があります。

6.1. OPENSTACKCONFIGGENERATOR CRD を使用したオーバークラウド設定用の ANSIBLE PLAYBOOK の作成

オーバークラウドのインフラストラクチャーをプロビジョニングした後に、オーバークラウドノード上に Red Hat OpenStack Platform (RHOSP) を設定する Ansible Playbook のセットを作成する必要があります。これらの Playbook を作成するには、**OpenStackConfigGenerator** カスタムリソース定義 (CRD) を使用します。**OpenStackConfigGenerator** CRD は、RHOSP ディレクターの **config-download** 機能を使用して、Heat 設定を Playbook に変換します。

ヒント

OpenStackConfigGenerator CRD 定義と仕様スキーマを表示するには、次のコマンドを使用します。

```
$ oc describe crd openstackconfiggenerator

$ oc explain openstackconfiggenerator.spec
```

前提条件

- **OpenStackControlPlane** CRD を使用してコントロールプレーンを作成している。
- **OpenStackBaremetalSets** CRD を使用して Compute ノードを作成している。
- カスタム Heat テンプレートを含む **ConfigMap** オブジェクトを作成している。
- カスタム環境ファイルを含む **ConfigMap** オブジェクトを作成している。

手順

1. ワークステーションに **openstack-config-generator.yaml** という名前のファイルを作成します。リソース仕様を追加して Ansible Playbook を生成します。次の例では、Playbook を生成するための仕様を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
metadata:
  name: default 1
  namespace: openstack
spec:
  enableFencing: true 2
  gitSecret: git-secret 3
  imageURL: registry.redhat.io/rhosp-rhel8/openstack-tripleoclient:17.1
  heatEnvConfigMap: heat-env-config 4
  # List of heat environment files to include from tripleo-heat-templates/environments
```

```
heatEnvs: 5
- ssl/tls-endpoints-public-dns.yaml
- ssl/enable-tls.yaml
tarballConfigMap: tripleo-tarball-config 6
```

1. 設定ジェネレータの名前。デフォルトではこれが **default** です。
2. **true** に設定すると、フェンシングを有効にするために必要な heat 環境ファイルの自動作成が有効になります。本番 RHOSP 環境では、フェンシングを有効にする必要があります。pacemaker を実行する仮想マシンには、**fence-agents-kubevirt** パッケージが必要です。
3. Git 認証認証情報 (デフォルトでは **git-secret**) を含む **ConfigMap** オブジェクトに設定します。
4. カスタム環境ファイルを含む **ConfigMap** オブジェクト (デフォルトでは **heat-env-config**)。
5. Playbook の生成に使用する、TripleO によって **TripleO-heat-templates/environments** ディレクトリーに提供される、デフォルトの Heat 環境ファイルのリスト。
6. カスタム Heat テンプレートを含む tarball を含む **ConfigMap** オブジェクト (デフォルトでは **Tripleo-tarball-config**)。

2. オプション: **OpenStackConfigGenerator** CR が一時的 Heat サービスの作成に使用するコンテナーイメージの場所を変更するには、**openstack-config-generator.yaml** ファイルに次の設定を追加します。

```
spec:
  ...
  ephemeralHeatSettings:
    heatAPIImageURL: <heat_api_image_location>
    heatEngineImageURL: <heat_engine_image_location>
    mariadbImageURL: <mariadb_image_location>
    rabbitImageURL: <rabbitmq_image_location>
```

- **<heat_api_image_location>** を、heat API イメージをホストするディレクトリー **openstack-heat-api** へのパスに置き換えます。
- **<heat_engine_image_location>** を、ヒートエンジンイメージをホストするディレクトリーへのパス、**openstack-heat-engine** に置き換えます。
- **<mariadb_image_location>** を、MariaDB イメージをホストするディレクトリー (**openstack-mariadb**) へのパスに置き換えます。
- **<rabbitmq_image_location>** を、RabbitMQ イメージをホストするディレクトリー **openstack-rabbitmq** へのパスに置き換えます。

3. オプション: デバッグモードで設定を生成するための Ansible Playbook を作成するには、次の設定を **openstack-config-generator.yaml** ファイルに追加します。

```
spec:
  ...
  interactive: true
```

対話モードでの **OpenStackConfigGenerator** Pod のデバッグの詳細は、設定生成のデバッグを参照してください。

4. **openstack-config-generator.yaml** ファイルを保存します。
5. Ansible 設定ジェネレーターを作成します。

```
$ oc create -f openstack-config-generator.yaml -n openstack
```

6. 設定ジェネレーターのリソースが作成されたことを確認します。

```
$ oc get openstackconfiggenerator/default -n openstack
```

6.2. オーバークラウドのオペレーティングシステムの登録

Director Operator (OSPdO) がオーバークラウドノードを設定する前に、すべてのノードのオペレーティングシステムを Red Hat Customer Portal または Red Hat Satellite Server に登録し、ノードのリポジトリーを有効にする必要があります。

OpenStackControlPlane CR の一部として、OSPdO は、リモートシェル (RSH) 経由でアクセスして Red Hat OpenStack Platform (RHOSP) コマンドを実行する **OpenStackClient** Pod を作成します。この Pod には、**/home/cloud-admin/ctlplane-ansible-inventory** という名前の Ansible インベントリースクリプトも含まれています。

ノードを登録するには、**redhat_subscription** Ansible モジュールを OpenStackClient Pod のインベントリースクリプトと共に使用できます。

手順

1. **OpenStackClient** Pod への RSH 接続を開きます。

```
$ oc rsh -n openstack openstackclient
```

2. **cloud-admin** ホームディレクトリーに移動します。

```
$ cd /home/cloud-admin
```

3. ノードを登録する **redhat_subscription** モジュールを使用して Playbook を作成します。たとえば、以下の Playbook はコントローラーノードを登録します。

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      - openstack-17.1-for-rhel-9-x86_64-rpms
      - fast-datapath-for-rhel-9-x86_64-rpms
      - rhceph-6-tools-for-rhel-9-x86_64-rpms
  tasks:
    - name: Register system 1
```

```
    redhat_subscription:
      username: myusername
      password: p@55w0rd!
      org_id: 1234567
      release: 9.2
      pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
  - name: Disable all repos (2)
    command: "subscription-manager repos --disable *"
  - name: Enable Controller node repos (3)
    command: "subscription-manager repos --enable {{ item }}"
    with_items: "{{ repos }}"
```

1. ノードを登録するタスク。
2. 自動有効化されたリポジトリーを無効にするタスク。
3. コントローラーノードに関連するリポジトリーのみを有効にするタスク。リポジトリーは **repos** 変数でリストされます。

4. 必要なリポジトリーにオーバークラウドノードを登録します。

```
$ ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory ./rhsm.yaml
```

関連情報

- edhat_subscription – subscription-manager コマンドを使用した RHSM への登録およびサブスクリプションの管理
- 手動による Ansible ベースの登録の実行
- オーバークラウドのリポジトリー

6.3. DIRECTOR OPERATOR を使用したオーバークラウドの設定の適用

コントロールプレーンを作成してベアメタルの Compute ノードをプロビジョニングし、各ノードにソフトウェアを設定する Ansible Playbook を生成してからしか、director Operator (OSPdO) でオーバークラウドを設定できません。**OpenStackDeploy** カスタムリソース (CR) を作成すると、OSPdO は Ansible Playbook を実行してオーバークラウドを設定するジョブを作成します。

ヒント

OpenStackDeploy CRD 定義と仕様スキーマを表示するには、次のコマンドを使用します。

```
$ oc describe crd openstackdeploy

$ oc explain openstackdeploy.spec
```

前提条件

- **OpenStackControlPlane** CRD を使用してコントロールプレーンを作成している。
- **OpenStackBaremetalSets** CRD を使用して Compute ノードを作成している。

- **OpenStackConfigGenerator** CRD を使用して、オーバークラウドの Ansible Playbook 設定を作成している。

手順

1. 最新の **OpenStackConfigVersion** オブジェクトの **hash/digest** を取得します。これは、オーバークラウドの設定に使用する必要がある Ansible Playbook を表します。

```
$ oc get -n openstack --sort-by {.metadata.creationTimestamp} openstackconfigversion -o
json
```

2. ワークステーション上に **openstack-deployment.yaml** という名前のファイルを作成し、リソース仕様を Ansible Playbook に含めます。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: default
spec:
  configVersion: <config_version>
  configGenerator: default
```

   - **<config_version>** を、ステップ 1 で取得した Ansible Playbook の **hash/digest** (例: **n5fch96h548h75hf4hbdhb8hfdh676h57bh96h5c5h59hf4h88h...**) に置き換えます。

3. **openstack-deployment.yaml** ファイルを保存します。

4. **OpenStackDeploy** リソースを作成します。

```
$ oc create -f openstack-deployment.yaml -n openstack
```

   デプロイメントが実行すると、Ansible Playbook を実行するための Kubernetes ジョブが作成されます。ジョブのログを表示して、Ansible Playbook の実行を確認できます。

```
$ oc logs -f jobs/deploy-openstack-default
```

   openstackclient Pod にログインすることで、実行された Ansible Playbook に手動でアクセスすることもできます。現在のデプロイメントの ansible Playbook と **ansible.log** ファイルは **/home/cloud-admin/work/directory** にあります。

6.4. 設定生成のデバッグ

設定生成操作をデバッグするには、対話型モードを使用するように **OpenStackConfigGenerator** CR を設定します。対話型モードでは、**OpenStackConfigGenerator** CR は Playbook のレンダリングを開始する環境を作成しますが、Playbook は自動的にレンダリングされません。

前提条件

- **OpenStackConfigGenerator** CR は対話モードで作成されました。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
metadata:
  name: default
```

```
  namespace: openstack
spec:
  …
  interactive: true
```

- 接頭辞 **generate-config** が付いた **OpenStackConfigGenerator** Pod が開始されました。

手順

1. **OpenStackConfigGenerator** Pod へのリモートシェル (RSH) 接続を開きます。

```
$ oc rsh $(oc get pod -o name -l job-name=generate-config-default)
```

2. ファイルと Playbook のレンダリングを検査します。

```
$ ls -la /home/cloud-admin/
...
config 1
config-custom 2
config-passwords 3
create-playbooks.sh 4
process-heat-environment.py 5
tht-tars 6
```

1. OSPdO によって自動レンダリングされたファイルを保存するディレクトリー。
2. **heatEnvConfigMap** オプションで指定された環境ファイルを格納するディレクトリー。
3. OSPdO によって作成されたオーバークラウドサービスのパスワードを保存するディレクトリー。
4. Ansible Playbook をレンダリングするスクリプト。
5. **create-playbooks** で使用される内部スクリプト。文書化されていない heat クライアントのマップパラメーターのマージを複製します。
6. **tarballConfigMap** オプションで指定された tarball を格納するディレクトリー。

第7章 DIRECTOR OPERATOR を使用した RHOSP ハイパーコンバージドインフラストラクチャー (HCI) のデプロイ

director Operator (OSPdO) を使用して、ハイパーコンバージドインフラストラクチャー (HCI) を備えたオーバークラウドをデプロイできます。HCI を備えたオーバークラウドは、Compute サービスと Red Hat Ceph Storage OSD サービスを同じノードに配置します。

7.1. 前提条件

- お使いの Compute HCI ノードに OSD として使用する追加のディスクが必要である。
- 稼働中の Red Hat OpenShift Container Platform (RHOCP) クラスターに OSPdO をインストールして準備している。詳細は、director Operator のインストールと準備 を参照してください。
- **OpenStackNetConfig** カスタムリソース定義 (CRD) を使用して、コントロールプレーンと分離されたネットワークを含むオーバークラウドネットワークを作成している。詳細は、director Operator を使用したネットワークの作成 を参照してください。
- オーバークラウドのカスタム heat テンプレートおよび環境ファイルを保存するために **ConfigMap** を作成している。詳細は、director Operator を使用したオーバークラウドのカスタマイズ を参照してください。
- オーバークラウド用のコントロールプレーンとベアメタル Compute ノードを作成している。詳細は、director Operator を使用したオーバークラウドノードの作成 を参照してください。
- オーバークラウド設定用の Ansible Playbook をレンダリングするための **OpenStackConfigGenerator** カスタムリソースを作成し、適用している。

7.2. ディレクターオペレーターの COMPUTE HCI ロールを使用して、ROLES_DATA.YAML ファイルを作成する

オーバークラウドに Compute HCI ロールの設定を追加するには、オーバークラウドのデプロイメントに含める **roles_data.yaml** ファイルに Compute HCI ロールを追加する必要があります。

注記

roles_data.yaml をファイル名として使用するようにしてください。

手順

1. **openstackclient** のリモートシェルにアクセスします。

```
$ oc rsh -n openstack openstackclient
```

2. **OS_CLOUD** 環境変数の設定を解除します。

```
$ unset OS_CLOUD
```

3. **cloud-admin** ディレクトリーに移動します。

```
$ cd /home/cloud-admin/
```

4. **Controller** ロールおよび **ComputeHCI** ロールを設定して、新しい **roles_data.yaml** ファイルを生成します。

```
$ openstack overcloud roles generate -o roles_data.yaml Controller ComputeHCI
```

5. **openstackclient** Pod を終了します。

```
$ exit
```

6. カスタムの **roles_data.yaml** ファイルを **openstackclient** Pod からカスタムテンプレートディレクトリーにコピーします。

```
$ oc cp openstackclient:/home/cloud-admin/roles_data.yaml
custom_templates/roles_data.yaml -n openstack
```

関連情報

- roles_data ファイルの作成

次のステップ

- director Operator での HCI ネットワークの設定

7.3. DIRECTOR OPERATOR での HCI ネットワークの設定

ワークステーション上にディレクトリーを作成してカスタムテンプレートと環境ファイルを保存し、Compute HCI ロールの NIC テンプレートを設定します。

手順

1. カスタムテンプレートのディレクトリーを作成します。

```
$ mkdir custom_templates
```

2. **custom_templates** ディレクトリーに **multiple_nics_vlans_dvr.j2** という名前のカスタムテンプレートファイルを作成します。

3. ベアメタルノードの NIC の設定を **multiple_nics_vlans_dvr.j2** ファイルに追加します。NIC 設定ファイルの例は、HCI Compute ノード用のカスタム NIC heat テンプレート を参照してください。

4. カスタム環境ファイルのディレクトリーを作成します。

```
$ mkdir custom_environment_files
```

5. オーバークラウドロールの NIC テンプレートを、**custom_environment_files** ディレクトリーの **network-environment.yaml** 環境ファイルにマップします。

```
parameter_defaults:
  ComputeHCINetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
```

関連情報

- カスタムネットワークインターフェイステンプレート

次のステップ

- オーバークラウド設定へのカスタムテンプレートの追加

7.4. HCI COMPUTE ノード用のカスタム NIC HEAT テンプレート

以下は、HCI Compute ノードの NIC 設定が含まれる heat テンプレート例です。heat テンプレートの設定は、ネットワークを次のブリッジとインターフェイスにマッピングします。

Networks	ブリッジ	インターフェイス
コントロールプレーン、ストレージ、内部 API	該当なし	**nic3**
外部、テナント	**br-ex**	**nic4**

デプロイメントで次のテンプレートを使用するには、この例をワークステーションの **custom_templates** ディレクトリーの **multiple_nics_vlans_dvr.j2** にコピーします。ベアメタルノードの NIC 設定に合わせてこの設定を変更できます。

例

```
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
# BMH provisioning interface used for ctlplane
- type: interface
  name: nic1
  mtu: 1500
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
# Disable OCP cluster interface
- type: interface
  name: nic2
  mtu: 1500
  use_dhcp: false
{% for network in networks_all if network not in networks_skip_config|default([]) %}
{% if network == 'External' %}
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  use_dhcp: false
{% if network in role_networks %}
```

```
  addresses:
  - ip_netmask:
      {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
  routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endif %}
  members:
  - type: interface
    name: nic3
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    primary: true
{% endif %}
{% endfor %}
- type: ovs_bridge
  name: br-tenant
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  members:
  - type: interface
    name: nic4
    mtu: {{ min_viable_mtu }}
    use_dhcp: false
    primary: true
{% for network in networks_all if network not in networks_skip_config|default([]) %}
{% if network not in ["External"] and network in role_networks %}
  - type: vlan
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask:
        {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endif %}
{% endfor %}
```

7.5. オーバークラウド設定へのカスタムテンプレートの追加

director Operator (OSPdO) は、各ノードで Red Hat OpenStack Platform (RHOSP) ソフトウェアを設定する準備が整うと、オーバークラウドヒートテンプレートのコアセットをプロビジョニングノードに適用する Ansible Playbook に変換します。独自のカスタム heat テンプレートおよびカスタムロールファイルをオーバークラウドデプロイメントに追加するには、テンプレートファイルを tarball ファイルにアーカイブし、tarball ファイルのバイナリーコンテンツを **tripleo-tarball-config** という名前の OpenShift **ConfigMap** に含める必要があります。この tarball ファイルには、テンプレートのコアセットを拡張するための複雑なディレクトリー構造を含めることができます。OSPdO は、tarball ファイルから、heat テンプレートのコアセットと同じディレクトリーにファイルとディレクトリーを展開します。カスタムテンプレートのいずれかがコアコレクションのテンプレートと名前が同じ場合には、カスタムテンプレートはコアテンプレートを上書きします。

注記

環境ファイル内のすべての参照は、tarball が抽出される TripleO heat テンプレートに関連している必要があります。

前提条件

- プロビジョニングされたノードに適用するカスタムオーバークラウドテンプレート。

手順

1. カスタムテンプレートの場所に移動します。

```
$ cd ~/custom_templates
```

2. テンプレートを gzip 圧縮された tarball にアーカイブします。

```
$ tar -cvzf custom-config.tar.gz *.yaml
```

3. **tripleo-tarball-config** CR を作成し、tarball をデータとして使用します。

```
$ oc create configmap tripleo-tarball-config --from-file=custom-config.tar.gz -n openstack
```

4. **ConfigMap** CR が作成されたことを確認します。

```
$ oc get configmap/tripleo-tarball-config -n openstack
```

関連情報

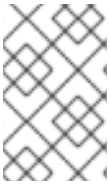
- 設定マップの作成および使用
- heat テンプレートの概要

次のステップ

- オーバークラウド設定へのカスタム環境ファイルの追加

7.6. DIRECTOR OPERATOR でハイパーコンバージドインフラストラクチャー (HCI) ストレージを設定するためのカスタム環境ファイル

以下の例は、Compute HCI ノード用の Ceph Storage 設定が含まれる環境ファイルです。この設定により、OSD ノードを **sdb**、**sdc**、**sdd** デバイスにマッピングし、**is_hci** オプションで HCI を有効にします。



注記

この設定は、ベアメタルノードのストレージ設定に合わせて変更できます。**CephPoolDefaultPgNum** パラメーターの値を確認するには、Ceph Placement Groups (PG) per Pool Calculator を使用します。

デプロイメントでこのテンプレートを使用するには、サンプルの内容を、ワークステーションの **custom_environment_files** ディレクトリーの **compute-hci.yaml** にコピーします。

```
resource_registry:
  OS::TripleO::Services::CephMgr: deployment/cephadm/ceph-mgr.yaml
  OS::TripleO::Services::CephMon: deployment/cephadm/ceph-mon.yaml
  OS::TripleO::Services::CephOSD: deployment/cephadm/ceph-osd.yaml
```

```
  OS::TripleO::Services::CephClient: deployment/cephadm/ceph-client.yaml

parameter_defaults:
  CephDynamicSpec: true
  CephSpecFqdn: true
  CephConfigOverrides:
    rgw_swift_enforce_content_length: true
    rgw_swift_versioning_enabled: true
    osd:
      osd_memory_target_autotune: true
      osd_numa_auto_affinity: true
    mgr:
      mgr/cephadm/autotune_memory_target_ratio: 0.2

  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: true
  CinderBackupBackend: ceph
  CinderEnableNfsBackend: false
  NovaEnableRbdBackend: true
  GlanceBackend: rbd
  CinderRbdPoolName: "volumes"
  NovaRbdPoolName: "vms"
  GlanceRbdPoolName: "images"
  CephPoolDefaultPgNum: 32
  CephPoolDefaultSize: 2
```

7.7. オーバークラウド設定へのカスタム環境ファイルの追加

オーバークラウドで機能を有効にしたりパラメーターを設定するには、オーバークラウドのデプロイメントに環境ファイルを含める必要があります。director Operator (OSPdO) は **heat-env-config** という名前の **ConfigMap** オブジェクトを使用して、環境ファイルを保存および取得します。**ConfigMap** オブジェクトは、環境ファイルを次の形式で保存します。

```
...
data:
  <environment_file_name>: |+
    <environment_file_contents>
```

たとえば、次の **ConfigMap** には 2 つの環境ファイルが含まれています。

```
...
data:
  network_environment.yaml: |+
    parameter_defaults:
      ComputeNetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
  cloud_name.yaml: |+
    parameter_defaults:
      CloudDomain: ocp4.example.com
      CloudName: overcloud.ocp4.example.com
      CloudNameInternal: overcloud.internalapi.ocp4.example.com
      CloudNameStorage: overcloud.storage.ocp4.example.com
      CloudNameStorageManagement: overcloud.storagemgmt.ocp4.example.com
      CloudNameCtlplane: overcloud.ctlplane.ocp4.example.com
```

ディレクトリーからオーバークラウドデプロイメントの一部として追加する **ConfigMap** オブジェクトにカスタム環境ファイルのセットをアップロードします。

前提条件

- オーバークラウドデプロイメント用のカスタム環境ファイル。

手順

1. **heat-env-config ConfigMap** オブジェクトを作成します。

```
$ oc create configmap -n openstack heat-env-config \
 --from-file=~/<dir_custom_environment_files>/ \
 --dry-run=client -o yaml | oc apply -f -
```

   - **<dir_custom_environment_files>** をオーバークラウドのデプロイメントで使用する環境ファイルが含まれるディレクトリーに置き換えます。**ConfigMap** オブジェクトは、これらを個別の **data** エントリーとして保存します。

2. **heat-env-config ConfigMap** オブジェクトに必要な環境ファイルがすべて含まれていることを確認します。

```
$ oc get configmap/heat-env-config -n openstack
```

7.8. HCI COMPUTE ノードの作成およびオーバークラウドのデプロイ

Compute ノードは Red Hat OpenStack Platform (RHOSP) 環境にコンピュートリソースを提供します。オーバークラウドには Compute ノードが少なくとも 1 台必要で、デプロイメント後にCompute ノードの数をスケーリングできます。

OpenStackBaremetalSet カスタムリソース (CR) を定義して、Red Hat OpenShift Container Platform (RHOCP) が管理するベアメタルマシンから Compute ノードを作成します。

ヒント

OpenStackBareMetalSet CRD 定義と仕様スキーマを表示するには、次のコマンドを使用します。

```
$ oc describe crd openstackbaremetalset

$ oc explain openstackbaremetalset.spec
```

前提条件

- **OpenStackNetConfig** CR を使用して、コントロールプレーンネットワークと追加の分離ネットワークを作成している。
- **OpenStackControlPlane** CRD を使用してコントロールプレーンを作成している。

手順

1. ワークステーションに **openstack-hcicompute.yaml** という名前のファイルを作成します。HCI Compute のリソース仕様を含めます。たとえば、3 つの HCI Compute ノードの仕様は次のとおりです。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: computehci (1)
  namespace: openstack (2)
spec: (3)
  count: 3
  baseImageUrl: http://<source_host>/rhel-9.2-x86_64-kvm.qcow2
  deploymentSSHSecret: osp-controlplane-ssh-keys
  ctlplaneInterface: enp8s0
  networks:
    - ctlplane
    - internal_api
    - tenant
    - storage
    - storage_mgmt
  roleName: ComputeHCI
  passwordSecret: userpassword (4)
```

(1) HCI Compute ノードのベアメタルセットの名前 (例: **computehci**)。

(2) OSPdO 名前空間 (例: **openstack**)。

(3) HCI Compute ノードの設定。

(4) オプション: パスワードを使用してユーザーに各ノードでの root アクセスを提供する **Secret** リソース。

2. **openstack-hcicompute.yaml** ファイルを保存します。
3. HCI Compute ノードを作成します。

```
$ oc create -f openstack-hcicompute.yaml -n openstack
```

4. HCI Compute ノードのリソースが作成されたことを確認します。

```
$ oc get openstackbaremetalset/computehci -n openstack
```

5. HCI Compute ノードの作成を確認するには、RHOCP が管理するベアメタルマシンを表示します。

```
$ oc get baremetalhosts -n openshift-machine-api
```

6. **OpenStackConfigGenerator** CRD を使用して、オーバークラウド設定用の Ansible Playbook を作成します。詳細は、OpenStackConfigGenerator CRD を使用したオーバークラウド設定用の Ansible Playbook の作成 を参照してください。
7. オーバークラウドのオペレーティングシステムを登録します。詳細は、オーバークラウドのオペレーティングシステムの登録 を参照してください。
8. オーバークラウド設定を適用します。詳細は、director Operator を使用したオーバークラウド設定の適用 を参照してください。

第8章 外部 RED HAT CEPH STORAGE クラスターを使用した RHOSP のデプロイ (DIRECTOR OPERATOR を使用)

director Operator (OSPdO) を使用して、外部の Red Hat Ceph Storage クラスターに接続するオーバークラウドをデプロイできます。

前提条件

- 外部 Red Hat Ceph Storage クラスターがあります。
- 稼働中の Red Hat OpenShift Container Platform (RHOCP) クラスターに OSPdO をインストールして準備している。詳細は、director Operator のインストールと準備 を参照してください。
- **OpenStackNetConfig** カスタムリソース定義 (CRD) を使用して、コントロールプレーンと分離されたネットワークを含むオーバークラウドネットワークを作成している。詳細は、director Operator を使用したネットワークの作成 を参照してください。
- オーバークラウドのカスタム heat テンプレートおよび環境ファイルを保存するために **ConfigMap** を作成している。詳細は、director Operator を使用したオーバークラウドのカスタマイズ を参照してください。
- オーバークラウド用のコントロールプレーンとベアメタル Compute ノードを作成している。詳細は、director Operator を使用したオーバークラウドノードの作成 を参照してください。
- オーバークラウド設定用の Ansible Playbook をレンダリングするための **OpenStackConfigGenerator** カスタムリソースを作成し、適用している。

8.1. DIRECTOR OPERATOR での COMPUTE ロールのネットワークの設定

ワークステーション上にディレクトリーを作成してカスタムテンプレートと環境ファイルを保存し、Compute ロールの NIC テンプレートを設定します。

手順

1. カスタムテンプレートのディレクトリーを作成します。

```
$ mkdir custom_templates
```

2. **custom_templates** ディレクトリーに **multiple_nics_vlans_dvr.j2** という名前のカスタムテンプレートファイルを作成します。
3. ベアメタル Compute ノードの NIC の設定を **multiple_nics_vlans_dvr.j2** ファイルに追加します。NIC 設定ファイルの例は、Compute ノード用のカスタム NIC heat テンプレート を参照してください。
4. カスタム環境ファイルのディレクトリーを作成します。

```
$ mkdir custom_environment_files
```

5. オーバークラウドロールの NIC テンプレートを、**custom_environment_files** ディレクトリーの **network-environment.yaml** 環境ファイルにマップします。

```
parameter_defaults:
  ComputeNetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
```

-

関連情報

- カスタムネットワークインターフェイステンプレート

8.2. COMPUTE ノード用のカスタム NIC HEAT テンプレート

次の例は、外部 Red Hat Ceph Storage クラスターに接続するオーバークラウド内の Compute ベアメタルノードの NIC 設定を含む Heat テンプレートです。heat テンプレートの設定は、ネットワークを次のブリッジとインターフェイスにマッピングします。

Networks	ブリッジ	interface
コントロールプレーン、ストレージ、内部 API	該当なし	**nic3**
外部、テナント	**br-ex**	**nic4**

デプロイメントで次のテンプレートを使用するには、この例をワークステーションの **custom_templates** ディレクトリーの **multiple_nics_vlans_dvr.j2** にコピーします。ベアメタルノードの NIC 設定に合わせてこの設定を変更できます。

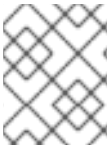
例

```
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
# BMH provisioning interface used for ctlplane
- type: interface
  name: nic1
  mtu: 1500
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
# Disable OCP cluster interface
- type: interface
  name: nic2
  mtu: 1500
  use_dhcp: false
{% for network in networks_all if network not in networks_skip_config|default([]) %}
{% if network == 'External' %}
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  use_dhcp: false
{% if network in role_networks %}
```

```
  addresses:
  - ip_netmask:
      {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
  routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endif %}
  members:
  - type: interface
    name: nic3
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    primary: true
{% endif %}
{% endfor %}
- type: ovs_bridge
  name: br-tenant
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  members:
  - type: interface
    name: nic4
    mtu: {{ min_viable_mtu }}
    use_dhcp: false
    primary: true
{% for network in networks_all if network not in networks_skip_config|default([]) %}
{% if network not in ["External"] and network in role_networks %}
  - type: vlan
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask:
        {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endif %}
{% endfor %}
```

8.3. オーバークラウド設定へのカスタムテンプレートの追加

director Operator (OSPdO) は、各ノードで Red Hat OpenStack Platform (RHOSP) ソフトウェアを設定する準備が整うと、オーバークラウドヒートテンプレートのコアセットをプロビジョニングノードに適用する Ansible Playbook に変換します。独自のカスタム heat テンプレートおよびカスタムロールファイルをオーバークラウドデプロイメントに追加するには、テンプレートファイルを tarball ファイルにアーカイブし、tarball ファイルのバイナリーコンテンツを **tripleo-tarball-config** という名前の OpenShift **ConfigMap** に含める必要があります。この tarball ファイルには、テンプレートのコアセットを拡張するための複雑なディレクトリー構造を含めることができます。OSPdO は、tarball ファイルから、heat テンプレートのコアセットと同じディレクトリーにファイルとディレクトリーを展開します。カスタムテンプレートのいずれかがコアコレクションのテンプレートと名前が同じ場合には、カスタムテンプレートはコアテンプレートを上書きします。



注記

環境ファイル内のすべての参照は、tarball が抽出される TripleO heat テンプレートに関連している必要があります。

前提条件

- プロビジョニングされたノードに適用するカスタムオーバークラウドテンプレート。

手順

1. カスタムテンプレートの場所に移動します。

```
$ cd ~/custom_templates
```

2. テンプレートを gzip 圧縮された tarball にアーカイブします。

```
$ tar -cvzf custom-config.tar.gz *.yaml
```

3. **tripleo-tarball-config** CR を作成し、tarball をデータとして使用します。

```
$ oc create configmap tripleo-tarball-config --from-file=custom-config.tar.gz -n openstack
```

4. **ConfigMap** CR が作成されたことを確認します。

```
$ oc get configmap/tripleo-tarball-config -n openstack
```

関連情報

- 設定マップの作成および使用
- heat テンプレートの概要

次のステップ

- オーバークラウド設定へのカスタム環境ファイルの追加

8.4. DIRECTOR OPERATOR で外部の CEPH STORAGE の使用状況を設定するためのカスタム環境ファイル

外部の Red Hat Ceph Storage クラスターと統合するには、次の例に示すようなパラメーターと値を持つ環境ファイルを含めます。この例では、オーバークラウドノードで **CephExternal** サービスと **CephClient** サービスを有効にし、さまざまな RHOSP サービスのプールを設定します。

注記

この設定は、ストレージ設定に合わせて変更できます。

デプロイメントでこのテンプレートを使用するには、サンプルの内容をワークステーションの **custom_environment_files** ディレクトリーの **ceph-ansible-external.yaml** にコピーします。

```
resource_registry:
  OS::TripleO::Services::CephExternal: deployment/cephadm/ceph-client.yaml

parameter_defaults:
  CephClusterFSID: '4b5c8c0a-ff60-454b-a1b4-9747aa737d19' 1
  CephClientKey: 'AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==' 2
```

```
  CephExternalMonHost: '172.16.1.7, 172.16.1.8' 3
  ExternalCeph: true

  # the following parameters enable Ceph backends for Cinder, Glance, Gnocchi and Nova
  NovaEnableRbdBackend: true
  CinderEnableRbdBackend: true
  CinderBackupBackend: ceph
  GlanceBackend: rbd
  # Uncomment below if enabling legacy telemetry
  # GnocchiBackend: rbd
  # If the Ceph pools which host VMs, Volumes and Images do not match these
  # names OR the client keyring to use is not named 'openstack',  edit the
  # following as needed.
  NovaRbdPoolName: vms
  CinderRbdPoolName: volumes
  CinderBackupRbdPoolName: backups
  GlanceRbdPoolName: images
  # Uncomment below if enabling legacy telemetry
  # GnocchiRbdPoolName: metrics
  CephClientUserName: openstack

  # finally we disable the Cinder LVM backend
  CinderEnableIscsiBackend: false
```

1 外部の Red Hat Ceph Storage クラスターのファイルシステム ID。

2 外部 Red Hat Ceph Storage クラスターの Red Hat Ceph Storage クライアントキー。

3 外部 Red Hat Ceph Storage クラスターの全 MON ホストの IP をコンマ区切りにしたリスト。

関連情報

- オーバークラウドの既存 Red Hat Ceph Storage クラスターとの統合
- Red Hat コンテナーレジストリーの認証

8.5. オーバークラウド設定へのカスタム環境ファイルの追加

オーバークラウドで機能を有効にしたりパラメーターを設定するには、オーバークラウドのデプロイメントに環境ファイルを含める必要があります。director Operator (OSPdO) は **heat-env-config** という名前の **ConfigMap** オブジェクトを使用して、環境ファイルを保存および取得します。**ConfigMap** オブジェクトは、環境ファイルを次の形式で保存します。

```
...
data:
  <environment_file_name>: |+
    <environment_file_contents>
```

たとえば、次の **ConfigMap** には 2 つの環境ファイルが含まれています。

```
...
data:
  network_environment.yaml: |+
    parameter_defaults:
```

```
    ComputeNetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
cloud_name.yaml: |+
  parameter_defaults:
    CloudDomain: ocp4.example.com
    CloudName: overcloud.ocp4.example.com
    CloudNameInternal: overcloud.internalapi.ocp4.example.com
    CloudNameStorage: overcloud.storage.ocp4.example.com
    CloudNameStorageManagement: overcloud.storagemgmt.ocp4.example.com
    CloudNameCtlplane: overcloud.ctlplane.ocp4.example.com
```

ディレクトリーからオーバークラウドデプロイメントの一部として追加する **ConfigMap** オブジェクトにカスタム環境ファイルのセットをアップロードします。

前提条件

- オーバークラウドデプロイメント用のカスタム環境ファイル。

手順

1. **heat-env-config ConfigMap** オブジェクトを作成します。

```
$ oc create configmap -n openstack heat-env-config \
 --from-file=~/<dir_custom_environment_files>/ \
 --dry-run=client -o yaml | oc apply -f -
```

   - **<dir_custom_environment_files>** をオーバークラウドのデプロイメントで使用する環境ファイルが含まれるディレクトリーに置き換えます。**ConfigMap** オブジェクトは、これらを個別の **data** エントリーとして保存します。

2. **heat-env-config ConfigMap** オブジェクトに必要な環境ファイルがすべて含まれていることを確認します。

```
$ oc get configmap/heat-env-config -n openstack
```

8.6. COMPUTE ノードの作成およびオーバークラウドのデプロイ

Compute ノードは Red Hat OpenStack Platform (RHOSP) 環境にコンピュートリソースを提供します。オーバークラウドには Compute ノードが少なくとも 1 台必要で、デプロイメント後にCompute ノードの数をスケーリングできます。

OpenStackBaremetalSet カスタムリソース (CR) を定義して、Red Hat OpenShift Container Platform (RHOCP) が管理するベアメタルマシンから Compute ノードを作成します。

ヒント

OpenStackBareMetalSet CRD 定義と仕様スキーマを表示するには、次のコマンドを使用します。

```
$ oc describe crd openstackbaremetalset

$ oc explain openstackbaremetalset.spec
```

前提条件

- **OpenStackNetConfig** CR を使用して、コントロールプレーンネットワークと追加の分離ネットワークを作成している。
- **OpenStackControlPlane** CRD を使用してコントロールプレーンを作成している。

手順

1. **OpenStackBaremetalSet** CRD を使用して Compute ノードを作成します。詳細は、OpenStackBaremetalSet CRD を使用した Compute ノードの作成を 参照してください。
2. **OpenStackConfigGenerator** CRD を使用して、オーバークラウド設定用の Ansible Playbook を作成します。詳細は、OpenStackConfigGenerator CRD を使用したオーバークラウド設定用の Ansible Playbook の作成 を参照してください。
3. オーバークラウドのオペレーティングシステムを登録します。詳細は、オーバークラウドのオペレーティングシステムの登録 を参照してください。
4. オーバークラウド設定を適用します。詳細は、director Operator を使用したオーバークラウド設定の適用 を参照してください。

第9章 DIRECTOR OPERATOR を使用してデプロイされたオーバークラウドへのアクセス

director Operator (OSPdO) を使用してオーバークラウドをデプロイした後に、**openstack** クライアントツールを使用してそのオーバークラウドにアクセスし、コマンドを実行できます。オーバークラウドの主なアクセスポイントは、作成した **OpenStackControlPlane** リソースの一部として OSPdO がデプロイする **OpenStackClient** Pod を経由します。

9.1. OPENSTACKCLIENT POD へのアクセス

OpenStackClient Pod は、オーバークラウドに対してコマンドを実行する主なアクセスポイントです。この Pod には、オーバークラウドに対してアクションを実行するのに必要なクライアントツールおよび認証情報が含まれます。ワークステーションから Pod にアクセスするには、ワークステーションで **oc** コマンドを使用して、Pod のリモートシェルに接続する必要があります。

注記

director Operator (OSPdO) なしでデプロイするオーバークラウドにアクセスする場合には、通常、**source ~/overcloudrc** コマンドを実行して、オーバークラウドにアクセスする環境変数を設定します。この手順は、OSPdO を使用してデプロイするオーバークラウドでは、必要ありません。

手順

1. **openstackclient** のリモートシェルにアクセスします。

```
$ oc rsh -n openstack openstackclient
```

2. **cloud-admin** ホームディレクトリーに移動します。

```
$ cd /home/cloud-admin
```

3. **openstack** コマンドを実行します。たとえば、以下のコマンドを使用して **default** ネットワークを作成できます。

```
$ openstack network create default
```

関連情報

- インスタンスの作成と管理
- Red Hat OpenStack Platform ネットワークの設定

9.2. オーバークラウドのダッシュボードへのアクセス

Director Operator (OSPdO) でデプロイしたオーバークラウドのダッシュボードには、標準のオーバークラウドと同じ方法 (Web ブラウザーを使用してコントロールプレーンによって予約された仮想 IP アドレスにアクセス) を使用してアクセスします。

手順

1. オプション: **admin** ユーザーとしてログインするには、**tripleo-passwords** シークレットにある **AdminPassword** パラメーターから admin パスワードを取得します。

```
$ oc get secret tripleo-passwords -o jsonpath='{.data.tripleo-overcloud-passwords\.yaml}' |
base64 -d
```

2. **OpenStackNetConfig** CR からコントロールプレーン用に予約されている IP アドレスを取得します。

```
spec:
  ...
  reservations:
    controlplane:
      ipReservations:
        ctlplane: 172.22.0.110
        external: 10.0.0.10
        internal_api: 172.17.0.10
        storage: 172.18.0.10
        storage_mgmt: 172.19.0.10
```

3. Web ブラウザーを開きます。

4. URL フィールドにコントロールプレーンの IP アドレスを入力します。

5. ユーザー名とパスワードを使用して Dashboard にログインします。

第10章 DIRECTOR OPERATOR を使用した COMPUTE ノードのスケーリング

必要なオーバークラウドのコンピュートリソース数が多い場合も少ない場合も、要件に応じて Compute ノード数をスケーリングできます。

10.1. DIRECTOR OPERATOR を使用したオーバークラウドの COMPUTE ノードの追加

Compute ノードにさらにオーバークラウドを追加するには、**compute OpenStackBaremetalSet** リソースのノード数を増やす必要があります。新しいノードがプロビジョニングされると、新しい **OpenStackConfigGenerator** リソースを作成して Ansible Playbook の新しいセットを生成し、次に **OpenStackConfigVersion** を使用して **OpenStackDeploy** オブジェクトを作成または更新し、Ansible 設定をオーバークラウドに再適用します。

手順

1. **openshift-machine-api** namespace の準備ができている状態にあるホストが十分に存在することを確認します。

```
$ oc get baremetalhosts -n openshift-machine-api
```

   ベアメタルホストの管理の詳細は、ベアメタルホストの管理 を参照してください。

2. l**compute OpenStackBaremetalSet** リソースの **count** パラメーターの値を増やします。

```
$ oc patch openstackbaremetalset compute --type=merge --patch '{"spec":{"count":3}}' -n openstack
```

   OpenStackBaremetalSet リソースは、Red Hat Enterprise Linux のベースオペレーティングシステムで新規ノードを自動的にプロビジョニングします。

3. プロビジョニングプロセスが完了するまで待ちます。ノードを定期的にチェックし、ノードの readiness を判別します。

```
$ oc get baremetalhosts -n openshift-machine-api
$ oc get openstackbaremetalset
```

4. オプション: 新しい Compute ノード上のネットワーク用に静的 IP アドレスを予約します。詳細は、 **OpenStackNetConfig** CRD を使用して追加した Compute ノードの静的 IP アドレスの予約 を参照してください。

5. **OpenStackConfigGenerator** を使用して Ansible Playbook を生成し、オーバークラウド設定を適用します。詳細は、director Operator を使用したオーバークラウドの設定とデプロイ を参照してください。

関連情報

- ベアメタルホストの管理

10.2. OPENSTACKNETCONFIG CRD を使用して追加した COMPUTE ノードの静的 IP アドレスの予約

OpenStackNetConfig CRD を使用して、オーバークラウドに追加した Compute ノード用に予約する IP アドレスを定義します。

ヒント

OpenStackNetConfig CRD 定義と仕様スキーマを表示するには、次のコマンドを使用します。

```
$ oc describe crd openstacknetconfig

$ oc explain openstacknetconfig.spec
```

手順

1. ワークステーション上でオーバークラウドの **openstacknetconfig.yaml** ファイルを開きます。
2. 次の設定を **openstacknetconfig.yaml** に追加して、**OpenStackNetConfig** カスタムリソース (CR) を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
```

3. 特定のノード上のネットワーク用に静的 IP アドレスを予約します。

```
spec:
  ...
  reservations:
    controller-0:
      ipReservations:
        ctlplane: 172.22.0.120
    compute-0:
      ipReservations:
        ctlplane: 172.22.0.140
        internal_api: 172.17.0.40
        storage: 172.18.0.40
        tenant: 172.20.0.40
    ...
    //The key for the ctlplane VIPs
    controlplane:
      ipReservations:
        ctlplane: 172.22.0.110
        external: 10.0.0.10
        internal_api: 172.17.0.10
        storage: 172.18.0.10
        storage_mgmt: 172.19.0.10
      macReservations: {}
```

注記

予約は、自動生成された IP アドレスよりも優先されます。

4. **openstacknetconfig.yaml** 定義ファイルを保存します。

5. オーバークラウドネットワーク設定を作成します。

```
$ oc create -f osnetconfig.yaml -n openstack
```

検証

1. オーバークラウドネットワーク設定が作成されたことを確認するには、オーバークラウドネットワーク設定のリソースを表示します。

```
$ oc get openstacknetconfig/openstacknetconfig
```

2. **OpenStackNetConfig** API と子リソースを表示します。

```
$ oc get openstacknetconfig/openstacknetconfig -n openstack
$ oc get openstacknetattachment -n openstack
$ oc get openstacknet -n openstack
```

エラーが表示された場合は、基礎となる **network-attach-definition** とノードのネットワーク設定ポリシーを確認してください。

```
$ oc get network-attachment-definitions -n openstack
$ oc get nncp
```

10.3. DIRECTOR OPERATOR を使用したオーバークラウドからの COMPUTE ノードの削除

オーバークラウドからCompute ノードを削除するには、Compute ノードを無効にして削除のマークを付け、**Compute OpenStackBaremetalSet** リソースのノード数を減らす必要があります。

注記

同じロールの新規ノードでオーバークラウドをスケーリングする場合に、ノードが一番小さい数値の ID 接尾辞で始まるホスト名と、対応の IP 予約を再利用する。

前提条件

- Compute ノード上のワークロードを他の Compute ノードに移行した。詳細は、コンピュートノード間の仮想マシンインスタンスの移行 を参照してください。

手順

1. **openstackclient** のリモートシェルにアクセスします。

```
$ oc rsh -n openstack openstackclient
```

2. 削除する Compute ノードを特定します。

```
$ openstack compute service list
```

3. ノード上の Compute サービスを無効にして、ノードが新しいインスタンスをスケジュールできないようにします。

```
$ openstack compute service set <hostname> nova-compute --disable
```

4. Metal [3] がノードを起動しないようにベアメタルノードにアノテーションを付けます。

```
$ oc annotate baremetalhost <node> baremetalhost.metal3.io/detached=true
$ oc logs --since=1h <metal3-pod> metal3-baremetal-operator | grep -i detach
$ oc get baremetalhost <node> -o json | jq .status.operationalStatus
"detached"
```

- **<node>** を **BareMetalHost** リソースの名前に置き換えます。
- **<metal3-pod>** を **metal3** Pod の名前に置き換えます。

5. **root** ユーザーとして Compute ノードにログインし、ベアメタルノードをシャットダウンします。

```
[root@compute-0 ~]# shutdown -h now
```

Compute ノードにアクセスできない場合は、次の手順を実行します。

a. コントローラーノードに **root** ユーザーとしてログインします。

b. インスタンス HA が有効になっている場合は、Compute ノードの STONITH デバイスを無効にします。

```
[root@controller-0 ~]# pcs stonith disable <stonith_resource_name>
```

- **<stonith_resource_name>** は、ノードに対応する STONITH リソースの名前に置き換えます。リソース名には、**<resource_agent>-<host_mac>** という形式が使用されます。リソースエージェントおよびホストの MAC アドレスは、**fencing.yaml** ファイルの **FencingConfig** セクションに記載されています。

c. IPMI を使用してベアメタルノードの電源をオフにします。詳細は、ハードウェアベンダーのドキュメントを参照してください。

6. 削除するノードに対応する **BareMetalHost** リソースを取得します。

```
$ oc get openstackbaremetalset compute -o json | jq '.status.baremetalHosts | to_entries[] | "\
(.key) => \(.value | .hostRef)"'
"compute-0, openshift-worker-3"
"compute-1, openshift-worker-4"
```

7. **OpenStackBaremetalSet** リソースの **annotatedForDeletion** パラメーターのステータスを **true** に変更するには、**BareMetalHost** リソースに **osp-director.openstack.org/delete-host=true** のアノテーションを付けます。

```
$ oc annotate -n openshift-machine-api bmh/openshift-worker-3 osp-
director.openstack.org/delete-host=true --overwrite
```

8. オプション: **OpenStackBaremetalSet** リソースで **annotatedForDeletion** ステータスが **true** に変更されたことを確認します。

```
$ oc get openstackbaremetalset compute -o json -n openstack | jq .status
{
```

```
 "baremetalHosts": {
  "compute-0": {
   "annotatedForDeletion": true,
   "ctlplaneIP": "192.168.25.105/24",
   "hostRef": "openshift-worker-3",
   "hostname": "compute-0",
   "networkDataSecretName": "compute-cloudinit-networkdata-openshift-worker-3",
   "provisioningState": "provisioned",
   "userDataSecretName": "compute-cloudinit-userdata-openshift-worker-3"
  },
  "compute-1": {
   "annotatedForDeletion": false,
   "ctlplaneIP": "192.168.25.106/24",
   "hostRef": "openshift-worker-4",
   "hostname": "compute-1",
   "networkDataSecretName": "compute-cloudinit-networkdata-openshift-worker-4",
   "provisioningState": "provisioned",
   "userDataSecretName": "compute-cloudinit-userdata-openshift-worker-4"
  }
 },
 "provisioningStatus": {
  "readyCount": 2,
  "reason": "All requested BaremetalHosts have been provisioned",
  "state": "provisioned"
 }
}
```


9. **compute OpenStackBaremetalSet** リソースの **count** パラメーターの値を増やします。

```
$ oc patch openstackbaremetalset compute --type=merge --patch '{"spec":{"count":1}}' -n openstack
```

OpenStackBaremetalSet リソースのリソース数を減らすと、対応するコントローラーがリソースの削除を処理するようトリガーされ、以下のアクションが発生します。

- director Operator は、削除されたノードの対応する IP 予約を **OpenStackIPSet** および **OpenStackNetConfig** から削除します。
- director Operator は、**OpenStackNet** リソースの IP 予約エントリーを削除済みとしてフラグします。


```
$ oc get osnet ctlplane -o json -n openstack | jq .reservations
{
  "compute-0": {
    "deleted": true,
    "ip": "172.22.0.140"
  },
  "compute-1": {
    "deleted": false,
    "ip": "172.22.0.100"
  },
  "controller-0": {
    "deleted": false,
    "ip": "172.22.0.120"
  },
  "controlplane": {
```

```
  "deleted": false,
  "ip": "172.22.0.110"
},
"openstackclient-0": {
  "deleted": false,
  "ip": "172.22.0.251"
}
```

10. オプション: 削除された **OpenStackBaremetalSet** リソースの IP 予約を他のロールが使用できるようにするには、**OpenStackNetConfig** オブジェクトで **spec.preserveservations** パラメーターの値を false に設定します。

11. **openstackclient** のリモートシェルにアクセスします。

```
$ oc rsh openstackclient -n openstack
```

12. オーバークラウドから Compute サービスエントリーを削除します。

```
$ openstack compute service list
$ openstack compute service delete <service-id>
```

13. オーバークラウドの Compute ネットワークエージェントエントリーをチェックして、このようなエントリーが存在する場合は削除します。

```
$ openstack network agent list
$ for AGENT in $(openstack network agent list --host <scaled-down-node> -c ID -f value) ;
do openstack network agent delete $AGENT ; done
```

14. **openstackclient** を終了します。

```
$ exit
```

第11章 DIRECTOR OPERATOR を使用して RHOSP オーバークラウドのマイナー更新を実行する

openstackclient Pod を更新した後、オーバークラウドとコンテナーイメージの準備デプロイメントを実行し、ノードを更新し、オーバークラウド更新収束デプロイメントを実行して、オーバークラウドを更新します。マイナー更新中は、コントロールプレーン API が利用可能です。

Red Hat OpenStack Platform (RHOSP) 環境のマイナー更新には、オーバークラウドノード上の RPM パッケージとコンテナーの更新が含まれます。一部のサービスの設定を更新する必要がある場合もあります。データプレーンとコントロールプレーンは、マイナー更新中に完全に利用可能になります。RHOSP 環境を更新するには、次の各手順を完了する必要があります。

1. マイナー更新用に RHOSP 環境を準備します。
2. オプション: **ovn-controller** コンテナーを更新します。
3. Pacemaker サービスを含むコントローラーノードとコンポーザブルノードを更新します。
4. コンピュートノードを更新します。
5. Red Hat Ceph Storage ノードを更新します。
6. Red Hat Ceph Storage クラスターを更新します。
7. オーバークラウドノードを再起動します。

前提条件

- RHOSP デプロイメントのバックアップがある。詳細は、director Operator を使用してデプロイしたオーバークラウドのバックアップと復元 を参照してください。

11.1. マイナー更新用の DIRECTOR オペレーターの準備

director Operator (OSPdO) を使用してマイナー更新を実行できるように Red Hat OpenStack Platform (RHOSP) 環境を準備するには、次のタスクを実行します。

1. RHOSP 環境を Red Hat Enterprise Linux (RHEL) リリースにロックします。
2. RHOSP リポジトリーを更新します。
3. コンテナーイメージ準備ファイルを更新します。
4. オーバークラウドでフェンシングを無効にします。

11.1.1. RHOSP 環境を RHEL リリースにロックする

Red Hat OpenStack Platform (RHOSP) 17.1 は Red Hat Enterprise Linux 9.2 (RHEL) でサポートされています。更新を実行する前に、オーバークラウドのリポジトリーを RHEL 9.2 リリースにロックして、オペレーティングシステムが新しいマイナーリリースにアップグレードされないようにします。

手順

1. オーバークラウドのサブスクリプション管理用環境ファイル **rhsm.yaml** を **openstackclient** にコピーします。

```
$ oc cp rhsm.yaml openstackclient:/home/cloud-admin/rhsm.yaml
```

2. **openstackclient** Pod のリモートシェルにアクセスします。

```
$ oc rsh openstackclient
```

3. **rhsm.yaml** ファイルを開き、Subscription Management 設定に **rhsm_release** パラメーターが含まれているかどうかを確認します。**rhsm_release** パラメーターが存在しない場合は、追加して **9.2** に設定します。

```
parameter_defaults:
  RhsmVars:
    …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "1a85f9223e3d5e43013e3d6e8ff506fd"
    rhsm_method: "portal"
    rhsm_release: "9.2"
```

4. **rhsm.yaml** ファイルを保存します。

5. すべてのノードでオペレーティングシステムのバージョンを RHEL 9.2 にロックするタスクを内包した、**set_release.yaml** という名前の Playbook を作成します。

```
- hosts: all
  gather_facts: false
  tasks:
    - name: set release to 9.2
      command: subscription-manager release --set=9.2
      become: true
```

6. **openstackclient** Pod で、 **set_release.yaml** Playbook を実行します。

```
$ ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory /home/cloud-
admin/set_release.yaml --limit Controller,Compute
```

--limit オプションを使用して、コンテンツをすべての RHOSP ノードに適用します。Red Hat Ceph Storage ノードには別のサブスクリプションがある可能性があるため、これらのノードに対してこの Playbook を実行しないでください。

注記

手動でノードを特定のバージョンにロックするには、ノードにログインして **subscription-manager release** コマンドを実行します。

```
$ sudo subscription-manager release --set=9.2
```

7. **openstackclient** Pod のリモートシェルを終了します。

```
$ exit
```

11.1.2. RHOSP リポジトリーの更新

Red Hat OpenStack Platform (RHOSP) 17.1 を使用するようにリポジトリーを更新します。

手順

1. **rhsm.yaml** ファイルを開き、**rhsm_repos** パラメーターを正しいリポジトリーバージョンに更新します。

```
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      - openstack-17.1-for-rhel-9-x86_64-rpms
      - fast-datapath-for-rhel-9-x86_64-rpms
```

2. **rhsm.yaml** ファイルを保存します。

3. **openstackclient** Pod のリモートシェルにアクセスします。

```
$ oc rsh openstackclient
```

4. すべてのノードで、リポジトリーを **RHOSP 17.1** に設定するタスクを含む Playbook を、**update_rhosp_repos.yaml** という名前で作成します。

```
- hosts: all
  gather_facts: false
  tasks:
    - name: change osp repos
      command: subscription-manager repos --enable=openstack-17.1-for-rhel-9-x86_64-rpms
      become: true
```

5. **openstackclient** Pod で **update_rhosp_repos.yaml** Playbook を実行します。

```
$ ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory /home/cloud-admin/update_rhosp_repos.yaml --limit Controller,Compute
```

--limit オプションを使用して、コンテンツをすべての RHOSP ノードに適用します。Red Hat Ceph Storage ノードは通常異なるサブスクリプションを使用するため、この Playbook を Ceph Storage ノードに対して実行しないでください。

6. すべての Red Hat Ceph Storage ノードで、リポジトリーを **RHOSP 17.1** に設定するタスクを含む Playbook を、**update_ceph_repos.yaml** という名前で作成します。

```
- hosts: all
  gather_facts: false
  tasks:
    - name: change ceph repos
      command: subscription-manager repos --enable=openstack-17.1-deployment-tools-for-rhel-9-x86_64-rpms
      become: true
```

7. **openstackclient** Pod で **update_ceph_repos.yaml** Playbook を実行します。

```
$ ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory /home/cloud-admin/update_ceph_repos.yaml --limit CephStorage
```

--limit オプションを使用して、コンテンツを Red Hat Ceph Storage ノードに適用します。

8. **openstackclient** Pod のリモートシェルを終了します。

```
$ exit
```

11.1.3. コンテナーイメージ準備ファイルの更新

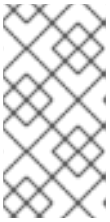
コンテナー準備ファイルは、**ContainerImagePrepare** パラメーターが含まれるファイルです。このファイルを使用して、オーバークラウドのコンテナーイメージを取得するためのルールを定義します。

環境を更新する前に、ファイルを確認して正しいイメージバージョンを取得するようにしてください。

手順

1. コンテナー準備ファイルを編集します。このファイルのデフォルト名は **containers-prepare-parameter.yaml** です。
2. それぞれのルールセットで、**tag** パラメーターが **17.1** に設定されていることを確認します。

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      ...
      tag: '17.1'
    tag_from_label: '{version}-{release}'
```



注記

更新に特定のタグ (**17.1** や **17.1.1** など) を使用しない場合は、**tag** キーと値のペアを削除し、**tag_from_label** のみを指定します。**tag_from_label** タグは、インストールされている Red Hat OpenStack Platform (RHOSP) バージョンを使用して、更新プロセスの一部として使用するタグの値を決定します。

3. **containers-prepare-parameter.yaml** ファイルを保存します。

11.1.4. オーバークラウドでのフェンシングの無効化

オーバークラウドを更新する前に、フェンシングが無効になっていることを確認します。

コントローラーノードの更新プロセス中にフェンシングが環境にデプロイされると、オーバークラウドは特定ノードが無効であることを検出し、フェンシング操作を試みる場合があります。これにより、意図しない結果が生じる可能性があります。

オーバークラウドでフェンシングを有効にした場合、更新中はフェンシングを一時的に無効にする必要があります。

手順

1. **openstackclient** Pod のリモートシェルにアクセスします。

```
$ oc rsh openstackclient
```

2. コントローラーノードにログインし、Pacemaker コマンドを実行してフェンシングを無効にします。

```
$ ssh <controller-0.ctlplane> "sudo pcs property set stonith-enabled=false"
```

   - **<controller-0.ctlplane>** をコントローラーノードの名前に置き換えます。

3. **openstackclient** Pod のリモートシェルを終了します。

```
$ exit
```

関連情報

- STONITH を使用したコントローラーノードのフェンシング

11.2. DIRECTOR OPERATOR のオーバークラウド更新準備を実行する

更新プロセスのためにオーバークラウドを準備するには、更新準備設定を生成します。これにより、更新された ansible Playbook が作成され、ノードが更新のために準備されます。

手順

1. **osconfiggenerator-update-prepare.yaml** という名前の **OpenStackConfigGenerator** を作成します。

```
$ cat <<EOF > osconfiggenerator-update-prepare.yaml
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
metadata:
  name: "update"
  namespace: openstack
spec:
  gitSecret: git-secret
  enableFencing: false
  heatEnvs:
    - lifecycle/update-prepare.yaml
  heatEnvConfigMap: heat-env-config-update
  tarballConfigMap: tripleo-tarball-config-update
EOF
```

2. 設定を適用します。

```
$ oc apply -f osconfiggenerator-update-prepare.yaml
```

3. 更新の準備プロセスが完了するまで待ちます。

11.3. すべてのオーバークラウドサーバーで OVN-CONTROLLER を更新する

Modular Layer 2 Open Virtual Network メカニズムドライバー (ML2/OVN) を使用してオーバークラウドをデプロイした場合は、**ovn-controller** コンテナーを最新の Red Hat OpenStack Platform (RHOSP) 17.1 バージョンに更新します。更新は、**ovn-controller** コンテナーを実行するすべてのオーバークラウドサーバーで行われます。

重要

次の手順ではコントローラーノードの **ovn-northd** サービスを更新する前に、コンピュートノードの **ovn-controller** コンテナーを更新します。この手順を実行する前に誤って **ovn-northd** サービスを更新すると、仮想マシンインスタンスにアクセスできなくなったり、新しいインスタンスや仮想ネットワークを作成できなくなったりする可能性があります。次の手順で接続を復元します。

手順

1. **osdeploy-ovn-update.yaml** という名前の **OpenStackDeploy** カスタムリソース (CR) を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: ovn-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: externalUpdate
  advancedSettings:
    tags:
      - ovn
```

2. 更新された設定を適用します。

```
$ oc apply -f osdeploy-ovn-update.yaml
```

3. **ovn-controller** コンテナーの更新が完了するまで待ちます。

11.4. すべてのコントローラーノードを更新する

すべてのコントローラーノードを、最新の Red Hat OpenStack Platform (RHOSP) 17.1 バージョンに更新します。

手順

1. **osdeploy-controller-update.yaml** という名前の **OpenStackDeploy** カスタムリソース (CR) を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: controller-update
spec:
  configVersion: <config_version>
  configGenerator: update
```

```
  mode: update
  advancedSettings:
    limit: Controller
```

2. 更新された設定を適用します。

```
$ oc apply -f osdeploy-controller-update.yaml
```

3. コントローラーノードの更新が完了するまで待ちます。

11.5. すべてのコンピュートノードを更新する

すべてのコンピュートノードを、最新の Red Hat OpenStack Platform (RHOSP) 17.1 バージョンに更新します。コンピューティングノードを更新するには、操作をコンピュートノードのみに制限する **limit: Compute** オプションを指定して、**OpenStackDeploy** カスタムリソース (CR) を作成します。

手順

1. **osdeploy-compute-update.yaml** という名前の **OpenStackDeploy** CR を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: compute-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: update
  advancedSettings:
    limit: Compute
```

2. 更新された設定を適用します。

```
$ oc apply -f osdeploy-compute-update.yaml
```

3. Compute ノードの更新が完了するまで待ちます。

11.6. すべての HCI コンピュートノードを更新する

ハイパーコンバージドインフラストラクチャー (HCI) コンピュートノードを最新の Red Hat OpenStack Platform (RHOSP) 17.1 バージョンに更新します。HCI コンピュートノードを更新するには、操作を HCI ノードのみに制限する **limit: ComputeHCI** オプションを指定して、**OpenStackDeploy** カスタムリソース (CR) を作成します。コンテナー化された Red Hat Ceph Storage 4 クラスターに更新する場合は、**mode: external-update** および **tags: ["ceph"]** オプションも指定して、**OpenStackDeploy** CR を作成する必要があります。

手順

1. **osdeploy-computehci-update.yaml** という名前の **OpenStackDeploy** CR を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
```

```
  name: computehci-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: update
  advancedSettings:
    limit: ComputeHCI
```

2. 更新された設定を適用します。

```
$ oc apply -f osdeploy-computehci-update.yaml
```

3. ComputeHCI ノードの更新が完了するまで待ちます。

4. **osdeploy-ceph-update.yaml** という名前の **OpenStackDeploy** CR を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: ceph-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: external-update
  advancedSettings:
    tags:
      - ceph
```

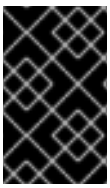
5. 更新された設定を適用します。

```
$ oc apply -f osdeploy-ceph-update.yaml
```

6. Red Hat Ceph Storage ノードの更新が完了するまで待ちます。

11.7. すべての RED HAT CEPH STORAGE ノードを更新する

Red Hat Ceph Storage ノードを、最新の Red Hat OpenStack Platform (RHOSP) 17.1 バージョンに更新します。



重要

RHOSP 17.1 は RHEL 9.2 でサポートされています。ただし、**Ceph Storage** ロールにマップされているホストは、最新のメジャー RHEL リリースに更新されます。詳細は、Red Hat Ceph Storage: サポートされる設定 を参照してください。

手順

1. **osdeploy-cephstorage-update.yaml** という名前の **OpenStackDeploy** カスタムリソース (CR) を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
```

```
  name: cephstorage-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: externalUpdate
  advancedSettings:
    limit: CephStorage
```

2. 更新された設定を適用します。

```
$ oc apply -f osdeploy-cephstorage-update.yaml
```

3. Red Hat Ceph Storage ノードの更新が完了するまで待ちます。

4. **osdeploy-ceph-update.yaml** という名前の **OpenStackDeploy** CR を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: ceph-update
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: externalUpdate
  advancedSettings:
    tags:
      - ceph
```

5. 更新された設定を適用します。

```
$ oc apply -f osdeploy-ceph-update.yaml
```

6. Red Hat Ceph Storage ノードの更新が完了するまで待ちます。

11.8. RED HAT CEPH STORAGE クラスターの更新

cephadm オーケストレーターを使用して、director を使用してデプロイされた Red Hat Ceph Storage クラスターを Red Hat OpenStack Platform (RHOSP) 17.1 と互換性のある最新バージョンに更新します。

手順

1. **openstackclient** Pod のリモートシェルにアクセスします。

```
$ oc rsh openstackclient
```

2. コントローラーノードにログインします。

```
$ ssh <controller-0.ctlplane>
```

- **<controller-0.ctlplane>** をコントローラーノードの名前に置き換えます。

3. **cephadm** シェルにログインします。

```
[cloud-admin@controller-0 ~]$ sudo cephadm shell
```

4. **cephadm** を使用して、Red Hat Ceph Storage クラスターをアップグレードします。詳細は、**Red Hat Ceph Storage 6 アップグレードガイド**の cephadm を使用して Red Hat Ceph Storage クラスターをアップグレードする を参照してください。

5. **openstackclient** Pod のリモートシェルを終了します。

```
$ exit
```

11.9. データベースのオンライン更新の実施

一部のオーバークラウドコンポーネントでは、データベーステーブルのオンライン更新 (または移行) が必要です。データベースのオンライン更新は、次のコンポーネントに適用されます。

- Block Storage サービス (cinder)
- Compute サービス (nova)

手順

1. **osdeploy-online-migration.yaml** という名前の **OpenStackDeploy** カスタムリソース (CR) を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: online-migration
spec:
  configVersion: <config_version>
  configGenerator: update
  mode: external-update
  advancedSettings:
    tags:
      - online_upgrade
```

2. 更新された設定を適用します。

```
$ oc apply -f osdeploy-online-migration.yaml
```

11.10. オーバークラウドでのフェンシングの再有効化

最新の Red Hat OpenStack Platform (RHOSP) 17.1 に更新するには、オーバークラウドでフェンシングを再度有効にする必要があります。

手順

1. **openstackclient** Pod のリモートシェルにアクセスします。

```
$ oc rsh openstackclient
```

2. コントローラーノードにログインし、Pacemaker コマンドを実行してフェンシングを有効にします。

```
$ ssh <controller-0.ctlplane> "sudo pcs property set stonith-enabled=true"
```

   - **<controller-0.ctlplane>** をコントローラーノードの名前に置き換えます。

3. **openstackclient** Pod のリモートシェルを終了します。

```
$ exit
```

11.11. オーバークラウドの再起動

最新の 17.1 バージョンへの Red Hat Open Stack Platform (RHOSP) のマイナー更新実行後、オーバークラウドを再起動します。リブートにより、関連付けられたカーネル、システムレベル、およびコンテナーコンポーネントの更新と共にノードがリフレッシュされます。これらの更新により、パフォーマンスとセキュリティー上のメリットが得られます。ダウンタイムを計画して、再起動手順を実施します。

以下のガイドを使用して、さまざまなノードのタイプを再起動する方法を説明します。

- 1 つのロールで全ノードを再起動する場合は、各ノードを個別に再起動します。ロールの全ノードを同時に再起動すると、その操作中サービスにダウンタイムが生じる場合があります。
- 次の順序でノードの再起動の手順を完了します。
  1. 「コントローラーノードおよびコンポーザブルノードの再起動」
  2. 「Ceph Storage (OSD) クラスターの再起動」
  3. 「Compute ノードの再起動」

11.11.1. コントローラーノードおよびコンポーザブルノードの再起動

設定可能なロールに基づいて Controller ノードとスタンドアロンノードを再起動し、Compute ノードと Ceph ストレージノードを除外します。

手順

1. リブートするノードにログインします。
2. オプション: ノードが Pacemaker リソースを使用している場合は、クラスターを停止します。

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo pcs cluster stop
```

3. ノードをリブートします。

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo reboot
```

4. ノードがブートするまで待ちます。

検証

1. サービスが有効になっていることを確認します。

a. ノードが Pacemaker サービスを使用している場合は、ノードがクラスターに再度加わったか確認します。

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo pcs status
```

b. ノードが Systemd サービスを使用している場合は、すべてのサービスが有効化されていることを確認します。

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo systemctl status
```

c. ノードがコンテナー化されたサービスを使用している場合は、ノード上の全コンテナーがアクティブであることを確認します。

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo podman ps
```

11.11.2. Ceph Storage (OSD) クラスターの再起動

Ceph Storage (OSD) ノードのクラスターを再起動するには、以下の手順を実施します。

前提条件

- **ceph-mon** サービスを実行している Ceph Monitor または Controller ノードで、Red Hat Ceph Storage クラスターのステータスが正常であり、pg ステータスが **active+clean** であることを確認する。

```
$ sudo cephadm -- shell ceph status
```

Ceph クラスターが正常な場合、**HEALTH_OK** のステータスが返されます。

Ceph クラスターのステータスが異常な場合、**HEALTH_WARN** または **HEALTH_ERR** のステータスが返されます。トラブルシューティングのガイダンスは、Red Hat Ceph Storage 5 トラブルシューティングガイド または Red Hat Ceph Storage 6 トラブルシューティングガイド を参照してください。

手順

1. **ceph-mon** サービスを実行している Ceph Monitor または Controller ノードにログインし、Ceph Storage クラスターのリバランスを一時的に無効にします。

```
$ sudo cephadm shell -- ceph osd set noout
$ sudo cephadm shell -- ceph osd set norebalance
```

注記

マルチスタックまたは分散コンピュートノード (DCN) アーキテクチャーを使用している場合は、**noout** フラグと **norebalance** フラグの設定時に Ceph クラスター名を指定する必要があります。例: **sudo cephadm shell -c /etc/ceph/<cluster>.conf -k /etc/ceph/<cluster>.client.keyring**。

2. 再起動する最初の Ceph Storage ノードを選択し、そのノードにログインします。

3. ノードをリブートします。

```
$ sudo reboot
```

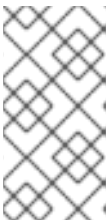
4. ノードがブートするまで待ちます。
5. ノードにログインし、Ceph クラスターのステータスを確認します。

```
$ sudo cephadm -- shell ceph status
```

pgmap により、すべての **pgs** が正常な状態 (**active+clean**) として報告されることを確認します。

6. ノードからログアウトして、次のノードを再起動し、ステータスを確認します。全 Ceph Storage ノードが再起動されるまで、このプロセスを繰り返します。
7. 完了したら、**ceph-mon** サービスを実行している Ceph Monitor または Controller ノードにログインし、クラスターのリバランスを有効にします。

```
$ sudo cephadm shell -- ceph osd unset noout
$ sudo cephadm shell -- ceph osd unset norebalance
```



注記

マルチスタックまたは分散コンピュートノード (DCN) アーキテクチャーを使用している場合は、**noout** フラグと **norebalance** フラグの設定解除時に Ceph クラスター名を指定する必要があります。例: **sudo cephadm shell -c /etc/ceph/<cluster>.conf -k /etc/ceph/<cluster>.client.keyring**。

8. 最終のステータスチェックを実行して、クラスターが **HEALTH_OK** を報告していることを確認します。

```
$ sudo cephadm shell ceph status
```

11.11.3. Compute ノードの再起動

Red Hat Open Stack Platform 環境でのインスタンスのダウンタイムを最小限に抑えるために、インスタンスの移行ワークフローでは、再起動するコンピュートノードからインスタンスを移行する時に必要な手順を概説します。

インスタンスの移行ワークフロー

1. Compute ノードを再起動する前に、インスタンスを別のノードに移行するか決定します。
2. 再起動する Compute ノードを選択して無効にし、新規インスタンスをプロビジョニングしないようにします。
3. インスタンスを別の Compute ノードに移行します。
4. 空の Compute ノードを再起動します。
5. 空の Compute ノードを有効にします。

前提条件

- Compute ノードを再起動する前に、ノードの再起動中にインスタンスを別の Compute ノードに移行するか決定する。
  Compute ノード間で仮想マシンインスタンスを移行する際に発生する可能性のある移行の制約リストを確認する。詳細は、**インスタンス作成のための Compute サービスの設定** の 移行の制約 を参照してください。

注記

Multi-RHEL 環境があり、RHEL 9.2 を実行しているコンピュートノードから RHEL 8.4 を実行しているコンピュートノードに仮想マシンを移行する場合は、コールドマイグレーションのみがサポートされます。コールドマイグレーションの詳細は、**インスタンス作成のための Compute サービスの設定** の インスタンスのコールドマイグレーション を参照してください。

- インスタンスを移行できない場合は、以下のコアテンプレートパラメーターを設定して、Compute ノード再起動後のインスタンスの状態を制御する。

NovaResumeGuestsStateOnHostBoot

リブート後の Compute ノードで、インスタンスを同じ状態に戻すか定義します。**False** に設定すると、インスタンスは停止した状態を維持し、手動で起動する必要があります。デフォルト値は **False** です。

NovaResumeGuestsShutdownTimeout

再起動する前に、インスタンスのシャットダウンを待つ秒数。この値を **0** に設定することは推奨されません。デフォルト値は **300** です。
オーバークラウドパラメーターおよびその使用方法の詳細は、オーバークラウドのパラメーター を参照してください。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. コンピュートノードのリストを取得して、再起動するノードのホスト名を特定します。

```
(undercloud)$ source ~/overcloudrc
(overcloud)$ openstack compute service list
```

再起動するコンピュートノードのホスト名を特定します。

3. 再起動するコンピュートノード上のコンピュートサービスを無効にします。

```
(overcloud)$ openstack compute service list
(overcloud)$ openstack compute service set <hostname> nova-compute --disable
```

- **<hostname>** は、コンピュートノードのホスト名に置き換えます。

4. Compute ノード上の全インスタンスをリスト表示します。

```
(overcloud)$ openstack server list --host <hostname> --all-projects
```

5. オプション: インスタンスを別のコンピュートノードに移行するには、次の手順を実行します。
   a. インスタンスを別の Compute ノードに移行する場合は、以下のコマンドのいずれかを使用します。

- インスタンスを別のホストに移行するには、次のコマンドを実行します。

```
(overcloud) $ openstack server migrate <instance_id> --live <target_host> --wait
```

  - **<instance_id>** は、インスタンス ID に置き換えます。
  - **<target_host>** は、インスタンスの移行先のホストに置き換えます。

- **nova-scheduler** がターゲットホストを自動的に選択できるようにします。

```
(overcloud) $ nova live-migration <instance_id>
```

- すべてのインスタンスを一度にライブマイグレーションします。

```
$ nova host-evacuate-live <hostname>
```

注記

nova コマンドで非推奨の警告が表示される可能性がありますが、無視しても問題ありません。

b. 移行が完了するまで待ちます。

c. 移行が正常に完了したことを確認します。

```
(overcloud) $ openstack server list --host <hostname> --all-projects
```

d. Compute ノードのインスタンスがなくなるまで、移行を続けます。

6. コンピュートノードにログインして、ノードをリブートします。

```
[tripleo-admin@overcloud-compute-0 ~]$ sudo reboot
```

7. ノードがブートするまで待ちます。

8. Compute ノードを再度有効にします。

```
$ source ~/overcloudrc
(overcloud) $ openstack compute service set <hostname>  nova-compute --enable
```

9. Compute ノードが有効であることを確認します。

```
(overcloud) $ openstack compute service list
```

11.11.4. オーバークラウドの更新後の RHOSP の検証

Red Hat OpenStack Platform (RHOSP) 環境を更新した後、**tripleo-validations** Playbook を使用してオーバークラウドを検証します。

検証の詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理**の 検証フレームワークの使用 を参照してください。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

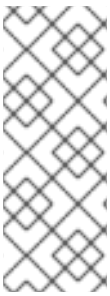
3. 検証を実行します。

```
$ validation run -i ~/overcloud-deploy/<stack>/tripleo-ansible-inventory.yaml --group post-update
```

- <stack> をスタックの名前に置き換えます。

検証

1. 検証レポートの結果を表示するには、**director を使用した Red Hat OpenStack Platform のインストールと管理** の 検証履歴の表示 を参照してください。



注記

検証の実行時にホストが見つからない場合、コマンドはステータスを **SKIPPED** と報告します。**SKIPPED** のステータスは、検証が実行されないことを意味しますが、これは想定内です。さらに、検証の合格基準が満たされていない場合、コマンドはステータスを **FAILED** と報告します。検証が **FAILED** であっても、更新された RHOSP 環境を使用できないことはありません。ただし、検証が **FAILED** の場合は、環境に問題があることを示している可能性があります。

第12章 DIRECTOR OPERATOR を使用してパブリックエンドポイントに TLS をデプロイする

TLS を使用してオーバークラウドをデプロイし、director Operator (OSPdO) のパブリックエンドポイント IP または DNS 名を作成します。

前提条件

- 稼働中の Red Hat OpenShift Container Platform (RHOCP) クラスターに OSPdO をインストールしている。
- ワークステーションに **oc** コマンドラインツールをインストールしている。
- 認証局、キー、および証明書を作成している。詳細は、オーバークラウドのパブリックエンドポイントでの SSL/TLS の有効化 を参照してください。

12.1. パブリックエンドポイント IP アドレスの TLS

パブリックエンドポイント IP アドレスを参照するには、CA 証明書を保存する **ConfigMap** リソースを作成し、**OpenStackControlPlane** リソースでその **ConfigMap** リソースを参照することにより、CA 証明書を **openstackclient** Pod に追加します。

手順

1. CA 証明書を保存するための **ConfigMap** リソースを作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cacerts
  namespace: openstack
data:
 local_CA: |
    -----BEGIN CERTIFICATE-----
    …
   -----END CERTIFICATE-----
  another_CA: |
    -----BEGIN CERTIFICATE-----
    …
   -----END CERTIFICATE-----
```

2. **OpenStackControlPlane** リソースを作成し、**ConfigMap** リソースを参照します。

```
apiVersion: osp-director.openstack.org/v1beta2
kind: OpenStackControlPlane
metadata:
  name: <overcloud>
  namespace: openstack
spec:
  caConfigMap: cacerts
```

- **<overcloud>** をオーバークラウドのコントロールプレーンの名前に置き換えます。

3. ~/**custom_environment_files** ディレクトリーに **tls-certs.yaml** という名前のファイルを作成します。このファイルは、**SSLCertificate**、**SSLIntermediateCertificate**、**SSLKey**、および **CAMap** パラメーターを使用して、デプロイメント用に生成された証明書を指定します。

4. **heatEnvConfigMap** を更新して **tls-certs.yaml** ファイルを追加します。

```
$ oc create configmap -n openstack heat-env-config --from-file=~/custom_environment_files/
--dry-run=client -o yaml | oc apply -f -
```

5. **OpenStackConfigGenerator** リソースを作成し、必要な **heatEnvs** 設定ファイルを追加して、パブリックエンドポイント IP の TLS を設定します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
…
spec:
  …
  heatEnvs:
    - ssl/tls-endpoints-public-ip.yaml
    - ssl/enable-tls.yaml
  …
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
```

6. **OpenStackConfigGenerator** を使用して Ansible Playbook を生成し、オーバークラウド設定を適用します。詳細は、director Operator を使用したオーバークラウドの設定とデプロイ を参照してください。

12.2. パブリックエンドポイント DNS 名の TLS

パブリックエンドポイント DNS 名を参照するには、CA 証明書を保存する **ConfigMap** リソースを作成し、**OpenStackControlPlane** リソースでその **ConfigMap** リソースを参照することにより、CA 証明書を **openstackclient** Pod に追加します。

手順

1. CA 証明書を保存するための **ConfigMap** リソースを作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cacerts
  namespace: openstack
data:
 local_CA: |
    -----BEGIN CERTIFICATE-----
    …
   -----END CERTIFICATE-----
  another_CA: |
    -----BEGIN CERTIFICATE-----
    …
   -----END CERTIFICATE-----
```

2. **OpenStackControlPlane** リソースを作成し、**ConfigMap** リソースを参照します。

```
apiVersion: osp-director.openstack.org/v1beta2
kind: OpenStackControlPlane
metadata:
  name: <overcloud>
  namespace: openstack
spec:
  caConfigMap: cacerts
```

- **<overcloud>** をオーバークラウドのコントロールプレーンの名前に置き換えます。

3. ~/**custom_environment_files** ディレクトリーに **tls-certs.yaml** という名前のファイルを作成します。このファイルは、**SSLCertificate**、**SSLIntermediateCertificate**、**SSLKey**、および **CAMap** パラメーターを使用して、デプロイメント用に生成された証明書を指定します。

4. **heatEnvConfigMap** を更新して **tls-certs.yaml** ファイルを追加します。

```
$ oc create configmap -n openstack heat-env-config --from-file=~/custom_environment_files/
--dry-run=client -o yaml | oc apply -f -
```

5. **OpenStackConfigGenerator** リソースを作成し、必要な **heatEnvs** 設定ファイルを追加して、パブリックエンドポイント DNS 名の TLS を設定します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
…
spec:
  …
  heatEnvs:
    - ssl/tls-endpoints-public-dns.yaml
    - ssl/enable-tls.yaml
  …
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
```

6. **OpenStackConfigGenerator** を使用して Ansible Playbook を生成し、オーバークラウド設定を適用します。詳細は、director Operator を使用したオーバークラウドの設定とデプロイ を参照してください。

第13章 DIRECTOR OPERATOR を使用してサービスアカウントのパスワードを変更する

Red Hat OpenStack Platform (RHOSP) サービスとそれらが使用するデータベースは、Identity サービス (keystone) の認証情報によって認証されます。Identity サービスは、RHOSP の初回デプロイプロセスを実行する間に RHOSP パスワードを生成します。脅威の軽減やセキュリティーコンプライアンスのために、パスワードを定期的に更新する必要がある場合もあります。director Operator (OSPdO) ネイティブのツールを使用して、RHOSP 環境をデプロイした後に多くの生成済みパスワードを変更できます。

13.1. DIRECTOR OPERATOR を使用してオーバークラウドサービスアカウントのパスワードをローテーションする

director Operator (OSPdO) がデプロイされた Red Hat OpenStack Platform (RHOSP) 環境で使用されるオーバークラウドのサービスアカウントパスワードをローテーションできます。

手順

1. 現在の **Tripleo-passwords** シークレットのバックアップを作成します。

```
$ oc get secret tripleo-passwords -n openstack -o yaml > tripleo-passwords_backup.yaml
```

2. **Tripleo-overcloud-passwords_preserve_list** という名前のプレインテキストファイルを作成して、次のサービスのパスワードをローテーションしないように指定します。

```
parameter_defaults
BarbicanSimpleCryptoKek
KeystoneCredential0
KeystoneCredential1
KeystoneFernetKey0
KeystoneFernetKey1
KeystoneFernetKeys
CephClientKey
CephClusterFSID
CephManilaClientKey
CephRgwKey
HeatAuthEncryptionKey
MysqlClustercheckPassword
MysqlMariabackupPassword
PacemakerRemoteAuthkey
PcsdPassword
```

   パスワードを保持したいサービスが他にもある場合は、そのサービスをリストに追加できます。

3. 変更すべきでないパスワードをリストするためのパスワードパラメーターファイル (**Tripleo-overcloud-passwords.yaml**) を作成します。

```
$ oc get secret tripleo-passwords -n openstack \
-o jsonpath='{.data.tripleo-overcloud-passwords\.yaml}' \
| base64 -d | grep -f ./tripleo-overcloud-passwords_preserve_list > tripleo-overcloud-passwords.yaml
```

4. **Tripleo-overcloud-passwords.yaml** ファイルに、ローテーションしないパスワードが含まれていることを検証します。

5. **tripleo-password** シークレットを更新します。

```
$ oc create secret generic tripleo-passwords -n openstack \
--from-file=./tripleo-overcloud-passwords.yaml \
--dry-run=client -o yaml | oc apply -f -
```

6. Ansible Playbook を作成し、OpenStackConfigGenerator CRD を使用してオーバークラウドを設定します。詳細は、OpenStackConfigGenerator CRD を使用したオーバークラウド設定用の Ansible Playbook の作成 を参照してください。

7. 更新された設定を適用します。詳細は、director Operator を使用したオーバークラウド設定の適用 を参照してください。

検証

シークレット内の新しい **NovaPassword** を、コントローラーノードにインストールされたものと比較します。

1. 更新されたシークレットからパスワードを取得します。

```
$ oc get secret tripleo-passwords -n openstack -o jsonpath='{.data.tripleo-overcloud-passwords\.yaml}' | base64 -d | grep NovaPassword
```

出力例:

```
NovaPassword: hp4xpt7t2p79ktqjjnxpqwbp6
```

2. コントローラーノード上で稼働している Compute サービス (nova) のパスワードを取得します。

   a. **openstackclient** リモートシェルにアクセスします。

```
$ oc rsh openstackclient -n openstack
```

   b. ホームディレクトリーにいることを確認します。

```
$ cd
```

   c. Compute サービスのパスワードを取得します。

```
$ ansible -i /home/cloud-admin/ctlplane-ansible-inventory Controller -b -a "grep ^connection /var/lib/config-data/puppet-generated/nova/etc/nova/nova.conf"
```

出力例:

```
172.22.0.120 | CHANGED | rc=0 >>
connection=mysql+pymysql://nova_api:hp4xpt7t2p79ktqjjnxpqwbp6@172.17.0.10/nova_api?read_default_file=/etc/my.cnf.d/tripleo.cnf&read_default_group=tripleo
connection=mysql+pymysql://nova:hp4xpt7t2p79ktqjjnxpqwbp6@172.17.0.10/nova?read_default_file=/etc/my.cnf.d/tripleo.cnf&read_default_group=tripleo
```

第14章 DIRECTOR OPERATOR を使用したスパインリーフ設定のノードのデプロイ

スパインリーフネットワーキングアーキテクチャーを備えたノードをデプロイメントして、環境内の広範なネットワークトポロジーを複製します。現在の制限により、**Metal3** のプロビジョニングネットワークは 1 つだけ許可されます。

14.1. OPENSTACKNETCONFIG カスタムリソースを作成または更新して、すべてのサブネットを定義する

OpenStackNetConfig カスタムリソース (CR) を定義し、オーバークラウドネットワークのサブネットを指定します。次に、Red Hat OpenStack Platform (RHOSP) director Operator (OSPdO) は設定をレンダリングし、ネットワークトポロジーを作成または更新します。

前提条件

- 稼働中の Red Hat OpenShift Container Platform (RHOCP) クラスターに OSPdO をインストールしている。
- ワークステーションに **oc** コマンドラインツールをインストールしている。

手順

1. **openstacknetconfig.yaml** という名前の設定ファイルを作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNetConfig
metadata:
  name: openstacknetconfig
spec:
  attachConfigurations:
    br-osp:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
          - bridge:
              options:
                stp:
                  enabled: false
              port:
              - name: enp7s0
            description: Linux bridge with enp7s0 as a port
            name: br-osp
            state: up
            type: linux-bridge
            mtu: 1500
    br-ex:
      nodeNetworkConfigurationPolicy:
        nodeSelector:
          node-role.kubernetes.io/worker: ""
        desiredState:
          interfaces:
```

```
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp6s0 as a port
          name: br-ex
          state: up
          type: linux-bridge
          mtu: 1500
# optional DnsServers list
dnsServers:
- 192.168.25.1
# optional DnsSearchDomains list
dnsSearchDomains:
- osptest.test.metalkube.org
- some.other.domain
# DomainName of the OSP environment
domainName: osptest.test.metalkube.org
networks:
- name: Control
  nameLower: ctlplane
  subnets:
  - name: ctlplane
    ipv4:
      allocationEnd: 192.168.25.250
      allocationStart: 192.168.25.100
      cidr: 192.168.25.0/24
      gateway: 192.168.25.1
    attachConfiguration: br-osp
- name: InternalApi
  nameLower: internal_api
  mtu: 1350
  subnets:
  - name: internal_api
    ipv4:
      allocationEnd: 172.17.0.250
      allocationStart: 172.17.0.10
      cidr: 172.17.0.0/24
      routes:
      - destination: 172.17.1.0/24
        nexthop: 172.17.0.1
      - destination: 172.17.2.0/24
        nexthop: 172.17.0.1
    vlan: 20
    attachConfiguration: br-osp
  - name: internal_api_leaf1
    ipv4:
      allocationEnd: 172.17.1.250
      allocationStart: 172.17.1.10
      cidr: 172.17.1.0/24
      routes:
      - destination: 172.17.0.0/24
        nexthop: 172.17.1.1
      - destination: 172.17.2.0/24
```

```
        nexthop: 172.17.1.1
    vlan: 21
    attachConfiguration: br-osp
  - name: internal_api_leaf2
    ipv4:
      allocationEnd: 172.17.2.250
      allocationStart: 172.17.2.10
      cidr: 172.17.2.0/24
      routes:
      - destination: 172.17.1.0/24
        nexthop: 172.17.2.1
      - destination: 172.17.0.0/24
        nexthop: 172.17.2.1
    vlan: 22
    attachConfiguration: br-osp
- name: External
  nameLower: external
  subnets:
  - name: external
    ipv4:
      allocationEnd: 10.0.0.250
      allocationStart: 10.0.0.10
      cidr: 10.0.0.0/24
      gateway: 10.0.0.1
    attachConfiguration: br-ex
- name: Storage
  nameLower: storage
  mtu: 1350
  subnets:
  - name: storage
    ipv4:
      allocationEnd: 172.18.0.250
      allocationStart: 172.18.0.10
      cidr: 172.18.0.0/24
      routes:
      - destination: 172.18.1.0/24
        nexthop: 172.18.0.1
      - destination: 172.18.2.0/24
        nexthop: 172.18.0.1
    vlan: 30
    attachConfiguration: br-osp
  - name: storage_leaf1
    ipv4:
      allocationEnd: 172.18.1.250
      allocationStart: 172.18.1.10
      cidr: 172.18.1.0/24
      routes:
      - destination: 172.18.0.0/24
        nexthop: 172.18.1.1
      - destination: 172.18.2.0/24
        nexthop: 172.18.1.1
    vlan: 31
    attachConfiguration: br-osp
  - name: storage_leaf2
    ipv4:
      allocationEnd: 172.18.2.250
```

```
      allocationStart: 172.18.2.10
      cidr: 172.18.2.0/24
      routes:
      - destination: 172.18.0.0/24
        nexthop: 172.18.2.1
      - destination: 172.18.1.0/24
        nexthop: 172.18.2.1
    vlan: 32
    attachConfiguration: br-osp
  - name: StorageMgmt
    nameLower: storage_mgmt
    mtu: 1350
    subnets:
    - name: storage_mgmt
      ipv4:
        allocationEnd: 172.19.0.250
        allocationStart: 172.19.0.10
        cidr: 172.19.0.0/24
        routes:
        - destination: 172.19.1.0/24
          nexthop: 172.19.0.1
        - destination: 172.19.2.0/24
          nexthop: 172.19.0.1
      vlan: 40
      attachConfiguration: br-osp
    - name: storage_mgmt_leaf1
      ipv4:
        allocationEnd: 172.19.1.250
        allocationStart: 172.19.1.10
        cidr: 172.19.1.0/24
        routes:
        - destination: 172.19.0.0/24
          nexthop: 172.19.1.1
        - destination: 172.19.2.0/24
          nexthop: 172.19.1.1
      vlan: 41
      attachConfiguration: br-osp
    - name: storage_mgmt_leaf2
      ipv4:
        allocationEnd: 172.19.2.250
        allocationStart: 172.19.2.10
        cidr: 172.19.2.0/24
        routes:
        - destination: 172.19.0.0/24
          nexthop: 172.19.2.1
        - destination: 172.19.1.0/24
          nexthop: 172.19.2.1
      vlan: 42
      attachConfiguration: br-osp
  - name: Tenant
    nameLower: tenant
    vip: False
    mtu: 1350
    subnets:
    - name: tenant
      ipv4:
```

```
      allocationEnd: 172.20.0.250
      allocationStart: 172.20.0.10
      cidr: 172.20.0.0/24
      routes:
      - destination: 172.20.1.0/24
        nexthop: 172.20.0.1
      - destination: 172.20.2.0/24
        nexthop: 172.20.0.1
    vlan: 50
    attachConfiguration: br-osp
  - name: tenant_leaf1
    ipv4:
      allocationEnd: 172.20.1.250
      allocationStart: 172.20.1.10
      cidr: 172.20.1.0/24
      routes:
      - destination: 172.20.0.0/24
        nexthop: 172.20.1.1
      - destination: 172.20.2.0/24
        nexthop: 172.20.1.1
    vlan: 51
    attachConfiguration: br-osp
  - name: tenant_leaf2
    ipv4:
      allocationEnd: 172.20.2.250
      allocationStart: 172.20.2.10
      cidr: 172.20.2.0/24
      routes:
      - destination: 172.20.0.0/24
        nexthop: 172.20.2.1
      - destination: 172.20.1.0/24
        nexthop: 172.20.2.1
    vlan: 52
    attachConfiguration: br-osp
```

2. 内部 API ネットワークを作成します。

```
$ oc create -f openstacknetconfig.yaml -n openstack
```

3. **OpenStackNetConfig** リソースのリソースと子リソースが作成されていることを確認します。

```
$ oc get openstacknetconfig/openstacknetconfig -n openstack
$ oc get openstacknetattachment -n openstack
$ oc get openstacknet -n openstack
```

14.2. デプロイメントにリーフネットワークのロールを追加する

リーフネットワークのロールをデプロイに追加するには、**roles_data.yaml** 設定ファイルを更新します。リーフネットワークロールに異なる NIC 設定がある場合、ロールごとに Ansible NIC テンプレートを作成してスパイン/リーフネットワークを設定し、NIC テンプレートを登録して、**ConfigMap** カスタムリソースを作成できます。

注記

ファイル名として **roles_data.yaml** を使用する必要があります。

手順

1. **roles_data.yaml** ファイルを更新します。

```
...
################################################################################
####
# Role: ComputeLeaf1                                           #
################################################################################
####
- name: ComputeLeaf1
  description: |
    Basic ComputeLeaf1 Node role
  # Create external Neutron bridge (unset if using ML2/OVS without DVR)
  tags:
    - compute
    - external_bridge
  networks:
    InternalApi:
      subnet: internal_api_leaf1
    Tenant:
      subnet: tenant_leaf1
    Storage:
      subnet: storage_leaf1
  HostnameFormatDefault: '%stackname%-novacompute-leaf1-%index%'
...
################################################################################
####
# Role: ComputeLeaf2                                           #
################################################################################
####
- name: ComputeLeaf2
  description: |
    Basic ComputeLeaf1 Node role
  # Create external Neutron bridge (unset if using ML2/OVS without DVR)
  tags:
    - compute
    - external_bridge
  networks:
    InternalApi:
      subnet: internal_api_leaf2
    Tenant:
      subnet: tenant_leaf2
    Storage:
      subnet: storage_leaf2
  HostnameFormatDefault: '%stackname%-novacompute-leaf2-%index%'
...
```

2. Compute ロールごとに NIC テンプレートを作成します。Ansible NIC テンプレートの例は、https://github.com/openstack/tripleo-ansible/tree/stable/wallaby/tripleo_ansible/roles/tripleo_network_config/templates を参照してください。

3. 新しいノードの NIC テンプレートを環境ファイルに追加します。

```
parameter_defaults:
  ComputeNetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
  ComputeLeaf1NetworkConfigTemplate: 'multiple_nics_vlans_dvr.j2'
  ComputeLeaf2NetworkConfigTemplate: 'multiple_nics_compute_leaf_2_vlans_dvr.j2'
```

4. ~/**custom_environment_files** ディレクトリーで、**roles_data.yaml** ファイル、環境ファイル、NIC テンプレートを tarball にアーカイブします。

```
$ tar -cvzf custom-spine-leaf-config.tar.gz *.yaml
```

5. **tripleo-tarball-config ConfigMap** リソースを作成します。

```
$ oc create configmap tripleo-tarball-config --from-file=custom-spine-leaf-config.tar.gz -n
openstack
```

14.3. 複数のルーティングされたネットワークを使用したオーバークラウドのデプロイ

複数のルーテッドネットワークセットを使用してオーバークラウドをデプロイするには、スパインリーフネットワークのコントロールプレーンと Compute ノードを作成し、Ansible Playbook をレンダリングして適用します。コントロールプレーンを作成するには、Controller ノードのリソースを指定します。ベアメタルマシンからリーフの Compute ノードを作成するには、**OpenStackBaremetalSet** カスタムリソースにリソース仕様を含めます。

手順

1. ワークステーションに **openstack-controller.yaml** という名前のファイルを作成します。コントローラーノードのリソース仕様を含めます。次の例は、3 つのコントローラーノードで設定されるコントロールプレーンの仕様を示しています。

```
apiVersion: osp-director.openstack.org/v1beta2
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  gitSecret: git-secret
  openStackClientImageURL: registry.redhat.io/rhosp-rhel9/openstack-tripleoclient:17.1
  openStackClientNetworks:
    - ctlplane
    - external
    - internal_api
    - internal_api_leaf1 # optionally the openstackclient can also be connected to subnets
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  domainName: ostest.test.metalkube.org
  virtualMachineRoles:
    Controller:
      roleName: Controller
      roleCount: 1
      networks:
```

```
      - ctlplane
      - internal_api
      - external
      - tenant
      - storage
      - storage_mgmt
    cores: 6
    memory: 20
    rootDisk:
      diskSize: 500
      baseImageVolumeName: openstack-base-img
      storageClass: host-nfs-storageclass
      storageAccessMode: ReadWriteMany
      storageVolumeMode: Filesystem
  enableFencing: False
```

2. コントロールプレーンを作成します。

```
$ oc create -f openstack-controller.yaml -n openstack
```

3. Red Hat OpenShift Container Platform (RHOCP) が **OpenStackControlPlane** リソースに関連するリソースを作成するまで待ちます。

4. Compute リーフごとにワークステーション上にファイル (例: **openstack-computeleaf1.yaml**) を作成します。リーフの Compute ノードのリソース仕様を含めます。次の例は、1 つの Compute ノードを含む 1 つの計算リーフの仕様を示しています。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: computeleaf1
  namespace: openstack
spec:
  # How many nodes to provision
  count: 1
  # The image to install on the provisioned nodes
  baseImageUrl: http://<source_host>/rhel-9.2-x86_64-kvm.qcow2
  # The secret containing the SSH pub key to place on the provisioned nodes
  deploymentSSHSecret: osp-controlplane-ssh-keys
  # The interface on the nodes that will be assigned an IP from the mgmtCidr
  ctlplaneInterface: enp7s0
  # Networks to associate with this host
  networks:
      - ctlplane
      - internal_api_leaf1
      - external
      - tenant_leaf1
      - storage_leaf1
  roleName: ComputeLeaf1
  passwordSecret: userpassword
```

5. 各リーフの Compute ノードを作成します。

```
$ oc create -f openstack-computeleaf1.yaml -n openstack
```

6. **OpenStackConfigGenerator** を使用して Ansible Playbook を生成し、オーバークラウド設定を適用します。詳細は、director Operator を使用したオーバークラウドの設定とデプロイ を参照してください。

検証

1. コントロールプレーンのリソースを表示します。

```
$ oc get openstackcontrolplane/overcloud -n openstack
```

2. **OpenStackVMSet** リソースを表示して、コントロールプレーンの仮想マシン (VM) セットの作成を確認します。

```
$ oc get openstackvmsets -n openstack
```

3. VM リソースを表示して、OpenShift Virtualization でのコントロールプレーン VM の作成を確認します。

```
$ oc get virtualmachines -n openstack
```

4. **openstackclient** Pod リモートシェルへのアクセスをテストします。

```
$ oc rsh -n openstack openstackclient
```

5. 各 Compute リーフのリソースを表示します。

```
$ oc get openstackbaremetalset/computeleaf1 -n openstack
```

6. RHOCP によって管理されるベアメタルマシンを表示して、Compute ノードの作成を確認します。

```
$ oc get baremetalhosts -n openshift-machine-api
```

第15章 DIRECTOR OPERATOR を使用してデプロイしたオーバークラウドのバックアップと復元

Red Hat OpenStack Platform (RHOSP) の Director Operator (OSPdO) は、デプロイメントのバックアップと復元のためのカスタムリソース定義 (CRD) を提供します。複数の設定を手動でエクスポートおよびインポートする必要はありません。OSPdO は、すべてのリソースの状態を認識しているため、**ConfigMap** および **Secret** CR を含むどのカスタムリソース (CR) が完全なバックアップを作成する必要があるかを認識しています。したがって、OSPdO は、不完全またはエラー状態にある設定をバックアップしません。

OSPdO デプロイメントをバックアップおよび復元するには、**OpenStackBackupRequest** CR を作成して、バックアップの作成または復元を開始します。**OpenStackBackupRequest** CR は、指定された名前空間のカスタムリソース (CR)、**ConfigMap**、および **Secret** 設定のバックアップを保存する **OpenStackBackup** CR を作成します。

15.1. DIRECTOR OPERATOR のバックアップ

バックアップを作成するには、名前空間の **OpenStackBackupRequest** カスタムリソース (CR) を作成する必要があります。**OpenStackBackup** CR は、**OpenStackBackupRequest** オブジェクトが **保存** モードで作成されるときに作成されます。

手順

1. ワークステーション上に **openstack_backup.yaml** という名前のファイルを作成します。
2. 以下の設定を **openstack_backup.yaml** ファイルに追加して、**OpenStackBackupRequest** カスタムリソース (CR) を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBackupRequest
metadata:
  name: openstackbackupsave
  namespace: openstack
spec:
  mode: save (1)
  additionalConfigMaps: [] (2)
  additionalSecrets: [] (3)
```

(1) **OpenStackBackup** CR の作成を要求するには、**mode** を **save** に設定します。

(2) オプション: 手動で作成した **ConfigMap** リソースを含めます。

(3) オプション: 手動で作成した **Secret** リソースを含めます。

注記

OSPdO は、**OpenStackControlPlane** や **OpenStackBaremetalSet** など、OSPdO CR に関連付けられたすべての **ConfigMap** オブジェクトと **Secret** オブジェクトを名前空間に含めようとします。これらを追加リストに含める必要はありません。

3. **openstack_backup.yaml** ファイルを保存します。

4. **OpenStackBackupRequest** CR を作成します。

```
$ oc create -f openstack_backup.yaml -n openstack
```

5. **OpenStackBackupRequest** CR の作成ステータスを監視します。

```
$ oc get openstackbackuprequest openstackbackupsave -n openstack
```

- **Quiescing** 状態は、OSPdO が CR が終了状態に達するのを待っていることを示します。CR の数は、バックアップの作成が完了するまでにかかる時間に影響を与える可能性があります。

```
NAME                 OPERATION   SOURCE   STATUS     COMPLETION TIMESTAMP
openstackbackupsave      save               Quiescing
```

ステータスが予想よりも長く **Quiescing** 状態のままである場合は、OSPdO ログを調査して進行状況を確認できます。

```
$ oc logs <operator_pod> -c manager -f
2022-01-11T18:26:15.180Z        INFO    controllers.OpenStackBackupRequest
Quiesce for save for OpenStackBackupRequest openstackbackupsave is waiting for:
[OpenStackBaremetalSet: compute, OpenStackControlPlane: overcloud,
OpenStackVMSet: controller]
```

  - **<operator_pod>** を Operator Pod の名前に置き換えます。

- **Saved** 状態は、**OpenStackBackup** CR が作成されたことを示します。

```
NAME                 OPERATION   SOURCE   STATUS   COMPLETION TIMESTAMP
openstackbackupsave      save               Saved    2022-01-11T19:12:58Z
```

- **Error** 状態は、バックアップの作成に失敗したことを示します。リクエストの内容を確認してエラーを見つけます。

```
$ oc get openstackbackuprequest openstackbackupsave -o yaml -n openstack
```

6. **OpenStackBackup** リソースを表示して、存在することを確認します。

```
$ oc get openstackbackup -n openstack
NAME                            AGE
openstackbackupsave-1641928378      6m7s
```

15.2. バックアップからの DIRECTOR OPERATOR の復元

バックアップの復元をリクエストすると、Red Hat OpenStack Platform (RHOSP) ディレクターオペレーター (OSPdO) は、指定された **OpenStackBackup** リソースの内容を取得し、それらを namespace 内に存在するすべての既存のカスタムリソース (CR)、**ConfigMap**、および **Secret** リソースに適用しようとします。。OSPdO は、namespace 内の既存のリソースを上書きし、namesapce 内に見つからないリソースに対して新しいリソースを作成します。

手順

1. 利用可能なバックアップを一覧表示します。

```
$ oc get osbackup
```

2. 特定のバックアップの詳細を調べます。

```
$ oc get backup <name> -o yaml
```

- **<name>** を検査するバックアップの名前に置き換えます。

3. ワークステーション上に **openstack_restore.yaml** という名前のファイルを作成します。

4. 以下の設定を **openstack_restore.yaml** ファイルに追加して、**OpenStackBackupRequest** カスタムリソース (CR) を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBackupRequest
metadata:
  name: openstackbackuprestore
  namespace: openstack
spec:
  mode: <mode>
  restoreSource: <restore_source>
```

- **<mode>** を次のいずれかのオプションに置き換えます。
  - **restore**: 既存の **OpenStackBackup** からのリストアを要求します。
  - **cleanRestore**: 既存の **OpenStackBackup** から新しいリソースを復元して作成する前に、namespace 内の既存の OSPdO リソースを完全に消去します。
- **<restore_source>** を復元する **OpenStackBackup** の ID に置き換えます (例: **openstackbackupsave-1641928378**)。

5. **openstack_restore.yaml** ファイルを保存します。

6. **OpenStackBackupRequest** CR を作成します。

```
$ oc create -f openstack_restore.yaml -n openstack
```

7. **OpenStackBackupRequest** CR の作成ステータスを監視します。

```
$ oc get openstackbackuprequest openstackbackuprestore -n openstack
```

- **Loading** 状態は、**OpenStackBackup** のすべてのリソースがクラスターに対して適用されていることを示します。

```
NAME                     OPERATION  SOURCE                       STATUS    COMPLETION TIMESTAMP
openstackbackuprestore   restore    openstackbackupsave-1641928378   Loading
```

- **Reconciling** 状態は、すべてのリソースがロードされ、OSPdO がすべてのリソースのプロビジョニングを試みるために調整を開始したことを示します。

```
NAME                    OPERATION  SOURCE                          STATUS      COMPLETION
TIMESTAMP
openstackbackuprestore  restore    openstackbackupsave-1641928378  Reconciling
```

- **Restored** 状態は、**OpenStackBackup** CR が復元されたことを示します。

```
NAME                    OPERATION  SOURCE                          STATUS    COMPLETION
TIMESTAMP
openstackbackuprestore  restore    openstackbackupsave-1641928378  Restored
2022-01-12T13:48:57Z
```

- **Error** 状態は、復元が失敗したことを示します。リクエストの内容を確認してエラーを見つけます。

```
$ oc get openstackbackuprequest openstackbackuprestore -o yaml -n openstack
```

第16章 DIRECTOR OPERATOR を使用して仮想マシンのリソースを変更する

OpenStackVMSet カスタムリソース (CR) の CPU、RAM、およびディスクリソースを変更するには、**OpenStackControlPlane** CRD を使用します。

16.1. OPENSTACKVMSET の CPU または RAM を変更する

OpenStackControlPlane CRD を使用して、**OpenStackVMSet** カスタムリソース (CR) の CPU または RAM を変更できます。

手順

1. Controller virtualMachineRole コアの数を 8 に変更します。

```
$ oc patch -n openstack osctlplane overcloud --type='json' -p='[{"op": "add", "path": "/spec/virtualMachineRoles/controller/cores", "value": 8 }]'
```

2. コントローラーの virtualMachineRole RAM サイズを 22GB に変更します。

```
$ oc patch -n openstack osctlplane overcloud --type='json' -p='[{"op": "add", "path": "/spec/virtualMachineRoles/controller/memory", "value": 22 }]'
```

3. virtualMachineRole リソースを検証します。

```
$ oc get osvmset
NAME         CORES   RAM   DESIRED   READY   STATUS        REASON
controller   8       22    1         1       Provisioned   All requested VirtualMachines have been provisioned
```

4. 仮想マシンの内部から、正常なシャットダウンを実行します。更新された各仮想マシンを 1 つずつシャットダウンします。

5. 仮想マシンをパワーオンします。

```
$ `virtctl start <VM>` to power on the virtual machine.
```

- **<VM>** を仮想マシンの名前に置き換えます。

16.2. OPENSTACKVMSET CR にディスクを追加する

additionalDisks プロパティーを編集することで、**OpenStackControlPlane** CRD を使用して仮想マシンにディスクを追加できます。

手順

1. **OpenStackControlPlane** オブジェクトで **additionalDisks** パラメーターを追加または更新します。

```
spec:
  ...
  virtualMachineRoles:
```

```
    Controller:
      ...
      additionalDisks:
      - baseImageVolumeName: openstack-base-img
        dedicatedIOThread: false
        diskSize: 10
        name: "data-disk1"
        storageAccessMode: ReadWriteMany
        storageClass: host-nfs-storageclass
        storageVolumeMode: Filesystem
```

2. パッチを適用します。

```
$ oc patch -n openstack osctlplane overcloud --patch-file controller_add_data_disk1.yaml
```

3. virtualMachineRole リソースを検証します。

```
$ oc get osvmset controller -o json | jq .spec.additionalDisks
[
  {
    "baseImageVolumeName": "openstack-base-img",
    "dedicatedIOThread": false,
    "diskSize": 10,
    "name": "data-disk1",
    "storageAccessMode": "ReadWriteMany",
    "storageClass": "host-nfs-storageclass",
    "storageVolumeMode": "Filesystem"
  }
]
```

4. 仮想マシンの内部から、正常なシャットダウンを実行します。更新された各仮想マシンを 1 つずつシャットダウンします。

5. 仮想マシンをパワーオンします。

```
$ `virtctl start <VM>` to power on the virtual machine.
```

- **<VM>** を仮想マシンの名前に置き換えます。

第17章 エアギャップ環境

エアギャップ環境は、他のネットワークやシステムから物理的に分離することでセキュリティーを確保します。director Operator をエアギャップ環境にインストールして、セキュリティーを確保し、特定の規制要件を提供できます。

17.1. 前提条件

- 稼働中の Red Hat Openshift Container Platform (RHOCP) クラスター、バージョン 4.12 以降。クラスターには **provisioning** ネットワークと次の Operator が含まれている必要があります。
  - **baremetal** クラスターの Operator。**baremetal** クラスター Operator を有効にする必要があります。**baremental** クラスターオペレーターの詳細は、ベアメタルクラスター Operator を参照してください。
  - OpenShift Virtualization Operator.OpenShift Virtualization Operator のインストールの詳細は、Web コンソールを使用した OpenShift Virtualization のインストール を参照してください。
  - SR-IOV Network Operator.
- docker v2 スキーマに準拠する切断されたレジストリーがあります。詳細は、切断されたインストールのイメージのミラーリング を参照してください。
- オーバークラウドノードの登録とパッケージのインストールに使用される Satellite Server またはその他のリポジトリーにアクセスできます。
- **oc** コマンドラインツールがワークステーションにインストールされています。
- ローカルの git リポジトリーにアクセスして、デプロイアーティファクトを保存できます。
- ワークステーションに **podman** および **skopeo** コマンドラインツールをインストールしました。

17.2. エアギャップ環境の設定

エアギャップ環境を設定するには、**registry.redhat.io** とエアギャップ環境のレジストリーの両方にアクセスできる必要があります。両方のレジストリーにアクセスする方法の詳細は、エアギャップレジストリーへのカタログコンテンツのミラーリング を参照してください。

手順

1. **openstack** namespace を作成します。

```
$ oc new-project openstack
```

2. インデックスイメージを作成し、レジストリーにプッシュします。

```
$ podman login registry.redhat.io
$ podman login your.registry.local
$ BUNDLE_IMG="registry.redhat.io/rhosp-rhel8/osp-director-operator-
bundle@sha256:c19099ac3340d364307a43e0ae2be949a588fefe8fcb17663049342e7587f055
"
```

注記

最新のバンドルイメージは次の場所から取得できます: Certified container images .**osp-director-operator-bundle** を検索します。

3. オペレータインデックスイメージに基づいて、関連するイメージをミラーリングします。

```
$ oc adm catalog mirror ${INDEX_IMG} your.registry.local --insecure --index-filter-by-os='Linux/x86_64'
```

4. ミラーリングが完了すると、現在のディレクトリーに **manifests-osp-director-operator-index-<random_number>** という **manifests** ディレクトリーが生成されます。作成した ImageContentSourcePolicy をクラスターに適用します。

```
$ os apply -f manifests-osp-director-operator-index-<random_number>/imageContentSourcePolicy.yaml
```

- **<random_number>** をランダムに生成された数値に置き換えます。

5. **osp-director-operator.yaml** という名前のファイルを作成し、次の YAML コンテンツを含めて、director Operator のインストールに必要な 3 つのリソースを設定します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: osp-director-operator-index
  namespace: openstack
spec:
  sourceType: grpc
  image: your.registry.local/osp-director-operator-index:1.3.x-y
---
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: "osp-director-operator-group"
  namespace: openstack
spec:
  targetNamespaces:
  - openstack
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: osp-director-operator-subscription
  namespace: openstack
spec:
  config:
    env:
    - name: WATCH_NAMESPACE
      value: openstack,openshift-machine-api,openshift-sriov-network-operator
  source: osp-director-operator-index
  sourceNamespace: openstack
  name: osp-director-operator
```

6. **openstack** namespace に新しいリソースを作成します。

```
$ oc apply -f osp-director-operator.yaml
```

7. 必要なオーバークラウドイメージをリポジトリーにコピーします。

```
$ for i in $(podman search --limit 1000 "registry.redhat.io/rhosp-rhel9/openstack" --format="{{ .Name }}" | awk '{print $1 ":" "17.1.0"}' | awk -F "/" '{print $2 "/" $3}'); do skopeo copy --all docker://registry.redhat.io/$i docker://your.registry.local/$i;done
```

注記

Red Hat Satellite がローカルレジストリーとして使用されている場合は、コンテナーイメージ用の Satellite Server の準備 を参照できます。

8. これで、director Operator のインストールと準備 に進むことができます。

検証

1. director Operator が正常にインストールされたことを確認します。

```
$ oc get operators
NAME                                 AGE
osp-director-operator.openstack      5m
```

関連情報

- CLI を使用した OperatorHub からのインストール
- 非接続クラスターで使用する Operator カタログのミラーリング
- カタログコンテンツをエアギャップされたレジストリーへのミラーリング
- コンテナーイメージ管理用 Satellite サーバーの準備
- プライベートレジストリーからのコンテナーイメージの取得

第18章 DIRECTOR OPERATOR を使用した RED HAT OPENSHIFT CONTAINER PLATFORM クラスター上のオーバークラウドのアップグレード (16.2 から 17.1)

インプレースフレームワークアップグレード (FFU) ワークフローを使用して、Red Hat OpenStack Platform (RHOSP) 16.2 オーバークラウドを、director Operator (OSPdO) を使用した RHOSP 17.1 オーバークラウドにアップグレードできます。

アップグレードを実行するには、次のタスクを実行する必要があります。

1. アップグレードに向けて環境を準備します。
2. カスタムの **roles_data** ファイルを、RHOSP 17.1 でサポートされるコンポーザブルサービスに更新します。
3. オプション: Red Hat Ceph Storage をアップグレードし、**cephadm** を導入します。
4. RHEL 8 上で RHOSP 17.1 コンテナーを実行するようにオーバークラウドノードをアップグレードします。
5. RHEL 9 上で RHOSP 17.1 コンテナーを実行するようにオーバークラウドノードをアップグレードします。
6. アップグレード後のタスクを実行します。

18.1. 前提条件

- OSPdO の最新バージョンを使用している。
- オーバークラウドノードが、最新の RHOSP 16.2 バージョンにより最新の状態になっている。マイナー更新を実行する方法については、director Operator を使用して RHOSP オーバークラウドのマイナー更新を実行する を参照してください。
- オーバークラウドノードが最新の RHEL 8 カーネルを実行している。

18.2. DIRECTOR OPERATOR の更新

オーバークラウドのアップグレードを実行する前に、director Operator (OSPdO) を最新の 17.1 バージョンに更新する必要があります。OSPdO を更新するには、まず現在の OSPdO を削除して再インストールする必要があります。OSPdO を削除するには、OSPdO サブスクリプションと CSV を削除します。

手順

1. **currentCSV** フィールドで、director Operator の現在のバージョンを確認します。

```
$ oc get subscription osp-director-operator.openstack -n openstack -o yaml | grep currentCSV
```

2. ターゲット名前空間内の director Operator の CSV を削除します。

```
$ oc delete clusterserviceversion <current_CSV> -n openstack
```

- **<current_CSV>** は、手順 1 の **currentCSV** の値に置き換えます。

3. サブスクリプションを削除します。

```
$ oc delete subscription  osp-director-operator.openstack -n openstack
```

4. 最新の 17.1 director Operator をインストールします。詳細は、director Operator のインストール を参照してください。

18.3. DIRECTOR OPERATOR 環境のアップグレードの準備

RHOSP 17.1 へのアップグレードに向けて、director Operator (OSPdO) でデプロイされた Red Hat OpenStack Platform (RHOSP) 環境を準備する必要があります。

手順

1. **openstackcontrolplane** CR で **openStackRelease** を 17.1 に設定します。

```
$ oc patch openstackcontrolplane -n openstack overcloud --type=json -p="[{'op': 'replace',
'path': '/spec/openStackRelease', 'value': '17.1'}]"
```

2. OSPdO **ClusterServiceVersion** (**csv**) CR を取得します。

```
$ oc get csv -n openstack
```

3. **OpenStackConfigGenerator** CR のすべてのインスタンスを削除します。

```
$ oc delete -n openstack openstackconfiggenerator --all
```

4. デプロイメントに HCI が含まれている場合は、RHEL8 の **openstackclient** イメージで RHOSP 17.1 を使用して、**ceph-ansible** から **cephadm** への変更を実行する必要があります。

```
$ oc patch openstackclient -n openstack openstackclient --type=json -p="[{'op': 'replace',
'path': '/spec/imageURL', 'value': 'registry.redhat.io/rhosp-rhel8/openstack-tripleoclient:17.1'}]"
```

デプロイメントに HCI が含まれていない場合、または **cephadm** の導入がすでに完了している場合は、**openstackclient** CR から現在の **imageURL** を削除して、17.1 OSPdO のデフォルトの **openstackclient** イメージに切り替えます。

```
$ oc patch openstackclient -n openstack openstackclient --type=json -p="[{'op': 'remove',
'path': '/spec/imageURL'}]"
```

5. オーバークラウドでフェンシングを有効にしている場合は、アップグレード中にコントローラーノードの 1 つでフェンシングを一時的に無効にする必要があります。

```
$ oc rsh -n openstack openstackclient
$ ssh controller-0.ctlplane "sudo pcs property set stonith-enabled=false"
```

18.4. カスタム ROLES_DATA ファイルのコンポーザブルサービスの更新

roles_data ファイルを、サポートされている Red Hat OpenStack Platform (RHOSP) 17.1 コンポーザブルサービスに更新する必要があります。詳細は、**16.2 から 17.1 へのアップグレードフレームワーク** ガイドの カスタム roles_data ファイルのコンポーザブルサービスの更新 を参照してください。

手順

1. すべてのロールから次のサービスを削除します。

```
`OS::TripleO::Services::CinderBackendDellEMCXTREMIOISCSI`
`OS::TripleO::Services::CinderBackendDellPs`
`OS::TripleO::Services::CinderBackendVRTSHyperScale`
`OS::TripleO::Services::Ec2Api`
`OS::TripleO::Services::Fluentd`
`OS::TripleO::Services::FluentdAlt`
`OS::TripleO::Services::Keepalived`
`OS::TripleO::Services::MistralApi`
`OS::TripleO::Services::MistralEngine`
`OS::TripleO::Services::MistralEventEngine`
`OS::TripleO::Services::MistralExecutor`
`OS::TripleO::Services::NeutronLbaasv2Agent`
`OS::TripleO::Services::NeutronLbaasv2Api`
`OS::TripleO::Services::NeutronML2FujitsuCfab`
`OS::TripleO::Services::NeutronML2FujitsuFossw`
`OS::TripleO::Services::NeutronSriovHostConfig`
`OS::TripleO::Services::NovaConsoleauth`
`OS::TripleO::Services::Ntp`
`OS::TripleO::Services::OpenDaylightApi`
`OS::TripleO::Services::OpenDaylightOvs`
`OS::TripleO::Services::OpenShift::GlusterFS`
`OS::TripleO::Services::OpenShift::Infra`
`OS::TripleO::Services::OpenShift::Master`
`OS::TripleO::Services::OpenShift::Worker`
`OS::TripleO::Services::PankoApi`
`OS::TripleO::Services::Rear`
`OS::TripleO::Services::SaharaApi`
`OS::TripleO::Services::SaharaEngine`
`OS::TripleO::Services::SensuClient`
`OS::TripleO::Services::SensuClientAlt`
`OS::TripleO::Services::SkydiveAgent`
`OS::TripleO::Services::SkydiveAnalyzer`
`OS::TripleO::Services::Tacker`
`OS::TripleO::Services::TripleoUI`
`OS::TripleO::Services::UndercloudMinionMessaging`
`OS::TripleO::Services::UndercloudUpgradeEphemeralHeat`
`OS::TripleO::Services::Zaqar`
```

2. **OS::TripleO::Services::GlanceApiInternal** サービスをコントローラーロールに追加します。

3. コンピュートロールの **OS::TripleO::Services::NovaLibvirt** サービスを **OS::TripleO::Services::NovaLibvirtLegacy** に更新します。

4. 環境に Red Hat Ceph Storage が含まれている場合は、**DeployedCeph** パラメーターを **false** に設定して、director 管理の **cephadm** デプロイメントを有効にします。

5. ネットワーク設定テンプレートに以下の機能が含まれている場合は、オーバークラウドをアップグレードする前に、NIC テンプレートを Jinja2 Ansible 形式に手動で変換する必要があります。次の関数は自動変換ではサポートされていません。

```
'get_file'
'get_resource'
'digest'
'repeat'
'resource_facade'
'str_replace'
'str_replace_strict'
'str_split'
'map_merge'
'map_replace'
'yaql'
'equals'
'if'
'not'
'and'
'or'
'filter'
'make_url'
'contains'
```

NIC テンプレートの手動変換に関する詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理** の NIC テンプレートの Jinja2 Ansible 形式への手動変換 を参照してください。

18.5. RED HAT CEPH STORAGE のアップグレードと CEPHADM の導入

環境に Red Hat Ceph Storage デプロイメントが含まれている場合は、デプロイメントを Red Hat Ceph Storage 5 にアップグレードする必要があります。バージョン 5 へのアップグレードにより、**cephadm** は、**ceph-ansible** の代わりに Red Hat Ceph Storage を管理するようになりました。

手順

1. **ceph-admin-user-playbook.yaml** という名前の Ansible Playbook ファイルを作成して、オーバークラウドノードに **ceph-admin** ユーザーを作成します。

2. 次の設定を **ceph-admin-user-playbook.yaml** ファイルに追加します。

```
- hosts: localhost
  gather_facts: false
  tasks:
    - name: set ssh key path facts
      set_fact:
        private_key: "{{ lookup('env', 'HOME') }}/.ssh/{{ tripleo_admin_user }}-id_rsa"
        public_key: "{{ lookup('env', 'HOME') }}/.ssh/{{ tripleo_admin_user }}-id_rsa.pub"
      run_once: true
    - name: stat private key
      stat:
        path: "{{ private_key }}"
      register: private_key_stat
    - name: create private key if it does not exist
      shell: "ssh-keygen -t rsa -q -N '' -f {{ private_key }}"
```

```
      no_log: true
      when:
        - not private_key_stat.stat.exists
    - name: stat public key
      stat:
        path: "{{ public_key }}"
      register: public_key_stat
    - name: create public key if it does not exist
      shell: "ssh-keygen -y -f {{ private_key }} > {{ public_key }}"
      when:
        - not public_key_stat.stat.exists

- hosts: overcloud
  gather_facts: false
  become: true
  pre_tasks:
    - name: Get local private key
      slurp:
        src: "{{ hostvars['localhost']['private_key'] }}"
      register: private_key_get
      delegate_to: localhost
      no_log: true
    - name: Get local public key
      slurp:
        src: "{{ hostvars['localhost']['public_key'] }}"
      register: public_key_get
      delegate_to: localhost
  roles:
    - role: tripleo_create_admin
      tripleo_admin_user: "{{ tripleo_admin_user }}"
      tripleo_admin_pubkey: "{{ public_key_get['content'] | b64decode }}"
      tripleo_admin_prikey: "{{ private_key_get['content'] | b64decode }}"
      no_log: true
```

3. Playbook を **openstackclient** コンテナーにコピーします。

```
$ oc cp -n openstack ceph-admin-user-playbook.yml openstackclient:/home/cloud-admin/ceph-admin-user-playbook.yml
```

4. **openstackclient** コンテナーで Playbook を実行します。

```
$ oc rsh -n openstack openstackclient
$ ansible-playbook -i /home/cloud-admin/ctlplane-ansible-inventory -e tripleo_admin_user=ceph-admin -e distribute_private_key=true /home/cloud-admin/ceph-admin-user-playbook.yml
```

5. デプロイメントで使用する Red Hat Ceph Storage のバージョンに合わせて、**containers-prepare-parameter.yaml** ファイル内の Red Hat Ceph Storage コンテナーイメージのパラメーターを更新します。

```
ceph_namespace: registry.redhat.io/rhceph
ceph_image: <ceph_image_file>
ceph_tag: latest
```

```
ceph_grafana_image: <grafana_image_file>
ceph_grafana_namespace: registry.redhat.io/rhceph
ceph_grafana_tag: latest
```

- **<ceph_image_file>** は、デプロイメントで使用する Red Hat Ceph Storage のバージョンのイメージファイルの名前に置き換えます。
  - Red Hat Ceph Storage 5: **rhceph-5-rhel8**
- **<grafana_image_file>** をデプロイメントで使用する Red Hat Ceph Storage のバージョンのイメージファイルの名前に置き換えます。
  - Red Hat Ceph Storage 5: **rhceph-5-dashboard-rhel8**

6. デプロイメントに HCI が含まれている場合は、**compute-hci.yaml** の **CephAnsibleRepo** パラメーターを "rhelosp-ceph-5-tools" に更新します。
7. **upgrade.yaml** という名前の環境ファイルを作成し、そのファイルに次の設定を追加します。

```
parameter_defaults:
  UpgradeInitCommand: |
    sudo subscription-manager repos --disable *
    if $( grep -q 9.2 /etc/os-release )
    then
      sudo subscription-manager repos --enable=rhel-9-for-x86_64-baseos-eus-rpms --enable=rhel-9-for-x86_64-appstream-eus-rpms --enable=rhel-9-for-x86_64-highavailability-eus-rpms --enable=openstack-17.1-for-rhel-9-x86_64-rpms --enable=fast-datapath-for-rhel-9-x86_64-rpms
      sudo podman ps | grep -q ceph && subscription-manager repos --enable=rhceph-5-tools-for-rhel-9-x86_64-rpms
      sudo subscription-manager release --set=9.2
    else
      sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-tus-rpms --enable=rhel-8-for-x86_64-appstream-tus-rpms --enable=rhel-8-for-x86_64-highavailability-tus-rpms --enable=openstack-17.1-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-x86_64-rpms
      sudo podman ps | grep -q ceph && subscription-manager repos --enable=rhceph-5-tools-for-rhel-8-x86_64-rpms
      sudo subscription-manager release --set=8.4
    fi
    sudo dnf -y install cephadm
```

8. **ceph-upgrade** という名前の新しい **OpenStackConfigGenerator** CR を作成して、更新した環境ファイルの ConfigMap と tripleo-tarball ConfigMap を含めます。
9. ワークステーション上に **openstack-ceph-upgrade.yaml** という名前のファイルを作成して、Red Hat Ceph Storage 4 から 5 へのアップグレード用の **OpenStackDeploy** CR を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: ceph-upgrade
spec:
  configVersion: <config_version>
  configGenerator: ceph-upgrade
```

```
  mode: externalUpgrade
  advancedSettings:
   skipTags:
   - ceph_health
   - opendev-validation
   - ceph_ansible_remote_tmp
   tags:
   - ceph
   - facts
```

10. **openstack-ceph-upgrade.yaml** ファイルを保存します。
11. **OpenStackDeploy** リソースを作成します。

```
$ oc create -f openstack-ceph-upgrade.yaml -n openstack
```

12. デプロイが完了するまで待ちます。
13. ワークステーション上に **openstack-ceph-upgrade-packages.yaml** という名前のファイルを作成して、Red Hat Ceph Storage パッケージをアップグレードする **OpenStackDeploy** CR を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: ceph-upgrade-packages
spec:
  configVersion: <config_version>
  configGenerator: ceph-upgrade
  mode: upgrade
  advancedSettings:
    limit: ceph_osd,ceph_mon,Undercloud
    playbook:
    - upgrade_steps_playbook.yaml
    skipTags:
    - ceph_health
    - opendev-validation
    - ceph_ansible_remote_tmp
    tags:
    - setup_packages
```

14. **openstack-ceph-upgrade-packages.yaml** ファイルを保存します。
15. **OpenStackDeploy** リソースを作成します。

```
$ oc create -f openstack-ceph-upgrade-packages.yaml -n openstack
```

16. デプロイが完了するまで待ちます。
17. ワークステーション上に **openstack-ceph-upgrade-to-cephadm.yaml** という名前のファイルを作成して、**cephadm** の導入を実行する **OpenStackDeploy** CR を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
```

```
  name: ceph-upgrade-to-cephadm
spec:
  configVersion: <config_version>
  configGenerator: ceph-upgrade
  mode: externalUpgrade
  advancedSettings:
    skipTags:
    - ceph_health
    - opendev-validation
    - ceph_ansible_remote_tmp
    tags:
    - cephadm_adopt
```

18. **openstack-ceph-upgrade-to-cephadm.yaml** ファイルを保存します。
19. **OpenStackDeploy** リソースを作成します。

```
$ oc create -f openstack-ceph-upgrade-to-cephadm.yaml -n openstack
```

20. デプロイが完了するまで待ちます。
21. **openstackclient** CR から現在の **imageURL** を削除して、**openstackclient** イメージを RHEL9 コンテナーイメージに更新します。

```
$ oc patch openstackclient -n openstack openstackclient --type=json -p="[{'op': 'remove',
'path': '/spec/imageURL'}]"
```

18.6. オーバークラウドを RHEL8 の RHOSP17.1 にアップグレードする

RHEL 8 で RHOSP 17.1 コンテナーを実行できるようにオーバークラウドノードをアップグレードするには、コンテナー準備ファイル (**ContainerImagePrepare** パラメーターを含むファイル) を更新する必要があります。このファイルを使用して、オーバークラウドのコンテナーイメージを取得するためのルールを定義します。

RHEL 8 ホストと RHEL 9 ホストの両方のコンテナー準備ファイルを更新する必要があります。

- RHEL 9 ホスト: すべてのコンテナーが RHEL9 ベースです。
- RHEL 8 ホスト: **libvirt** と **collectd** を除くすべてのコンテナーが RHEL9 ベースです。**libvirt** コンテナーと **collectd** コンテナーは、ホストと同じベースを使用する必要があります。

その後、更新をデプロイする前に、新しい **OpenStackConfigGenerator** CR を生成する必要があります。

手順

1. コンテナー準備ファイル (**containers-prepare-parameter.yaml**) を開き、正しいイメージバージョンが取得されることを確認します。
2. **ContainerImagePrepareRhel8** パラメーターを **containers-prepare-parameter.yaml** に追加します。

```
parameter_defaults:
  #default container image configuration for RHEL 9 hosts
  ContainerImagePrepare:
```

```
  - push_destination: false
    set: &container_image_prepare_rhel9_contents
      tag: 17.1.2
      name_prefix: openstack-
      namespace: registry.redhat.io/rhosp-rhel9
      ceph_namespace: registry.redhat.io/rhceph
      ceph_image: rhceph-5-rhel8
      ceph_tag: latest
      ceph_alertmanager_image: ose-prometheus-alertmanager
      ceph_alertmanager_namespace: registry.redhat.io/openshift4
      ceph_alertmanager_tag: v4.10
      ceph_grafana_image: rhceph-5-dashboard-rhel8
      ceph_grafana_namespace: registry.redhat.io/rhceph
      ceph_grafana_tag: latest
      ceph_node_exporter_image: ose-prometheus-node-exporter
      ceph_node_exporter_namespace: registry.redhat.io/openshift4
      ceph_node_exporter_tag: v4.10
      ceph_prometheus_image: ose-prometheus
      ceph_prometheus_namespace: registry.redhat.io/openshift4
      ceph_prometheus_tag: v4.10

  # RHEL8 hosts pin the collectd and libvirt containers to rhosp-rhel8
  # To apply the following configuration, reference the followingparameter
  # in the role specific parameters below: <Role>ContainerImagePrepare
  ContainerImagePrepareRhel8: &container_image_prepare_rhel8
  - push_destination: false
    set: *container_image_prepare_rhel9_contents
    excludes:
    - collectd
    - nova-libvirt
  - push_destination: false
    set:
      tag: 17.1.2
      name_prefix: openstack-
      namespace: registry.redhat.io/rhosp-rhel8
      ceph_namespace: registry.redhat.io/rhceph
      ceph_image: rhceph-5-rhel8
      ceph_tag: latest
      ceph_alertmanager_image: ose-prometheus-alertmanager
      ceph_alertmanager_namespace: registry.redhat.io/openshift4
      ceph_alertmanager_tag: v4.10
      ceph_grafana_image: rhceph-5-dashboard-rhel8
      ceph_grafana_namespace: registry.redhat.io/rhceph
      ceph_grafana_tag: latest
      ceph_node_exporter_image: ose-prometheus-node-exporter
      ceph_node_exporter_namespace: registry.redhat.io/openshift4
      ceph_node_exporter_tag: v4.10
      ceph_prometheus_image: ose-prometheus
      ceph_prometheus_namespace: registry.redhat.io/openshift4
      ceph_prometheus_tag: v4.10
    includes:
    - collectd
    - nova-libvirt
  # Initially all hosts are RHEL 8 so set the role specific container
  # image prepare parameter to the RHEL 8 configuration
```

```
  ControllerContainerImagePrepare: *container_image_prepare_rhel8
  ComputeContainerImagePrepare: *container_image_prepare_rhel8
...
```

3. **upgrade.yaml** という名前の環境ファイルを作成します。
4. 次の設定を **upgrade.yaml** ファイルに追加します。

```
parameter_defaults:
  UpgradeInitCommand: |
    sudo subscription-manager repos --disable *
      if $( grep -q  9.2  /etc/os-release )
      then
        sudo subscription-manager repos --enable=rhel-9.2-for-x86_64-baseos-eus-rpms --
enable=rhel-9.2-for-x86_64-appstream-eus-rpms --enable=rhel-9.2-for-x86_64-
highavailability-eus-rpms --enable=openstack-17.1-for-rhel-9-x86_64-rpms --enable=fast-
datapath-for-rhel-9-x86_64-rpms
      else
        sudo subscription-manager repos --enable=rhel-8-for-x86_64-baseos-eus-rpms --
enable=rhel-8-for-x86_64-appstream-eus-rpms --enable=rhel-8-for-x86_64-highavailability-
eus-rpms --enable=openstack-17.1-for-rhel-8-x86_64-rpms --enable=fast-datapath-for-rhel-8-
x86_64-rpms
        fi
```

5. **disable_compute_service_check.yaml** という名前の環境ファイルを作成します。
6. 次の設定を **disable_compute_service_check.yaml** ファイルに追加します。

```
parameter_defaults:
  ExtraConfig:
    nova::workarounds::disable_compute_service_check_for_ffu: true

parameter_merge_strategies:
  ExtraConfig: merge
```

7. デプロイメントに HCI が含まれている場合は、Red Hat Ceph Storage および HCI のパラメーターを RHOSP 16.2 の **ceph-ansible** 値から RHOSP 17.1 の **cephadm** 値に更新します。詳細は、director Operator でハイパーコンバージドインフラストラクチャー (HCI) ストレージを設定するためのカスタム環境ファイル 参照してください。
8. ワークステーション上に **openstack-configgen-upgrade.yaml** という名前のファイルを作成して、"upgrade" という名前の新しい **OpenStackConfigGenerator** CR を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackConfigGenerator
metadata:
  name: "upgrade"
  namespace: openstack
spec:
  enableFencing: False
  gitSecret: git-secret
  heatEnvs:
    - ssl/tls-endpoints-public-dns.yaml
    - ssl/enable-tls.yaml
    - nova-hw-machine-type-upgrade.yaml
```

```
    - lifecycle/upgrade-prepare.yaml
  heatEnvConfigMap: heat-env-config-upgrade
  tarballConfigMap: tripleo-tarball-config-upgrade
```

9. ワークステーション上に **openstack-upgrade.yaml** という名前のファイルを作成して、オーバークラウドアップグレード用の **OpenStackDeploy** CR を作成します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: upgrade
spec:
  configVersion: <config_version>
  configGenerator: upgrade
  mode: upgrade
```

10. **openstack-upgrade.yaml** ファイルを保存します。

11. **OpenStackDeploy** リソースを作成します。

```
$ oc create -f openstack-upgrade.yaml -n openstack
```

12. デプロイが完了するまで待ちます。オーバークラウドノードが RHEL8 上で 17.1 コンテナーを実行するようになります。

18.7. オーバークラウドを RHEL 9 にアップグレードする

RHEL 9 で RHOSP 17.1 コンテナーを実行できるようにオーバークラウドノードをアップグレードするには、コンテナー準備ファイル (**ContainerImagePrepare** パラメーターを含むファイル) を更新する必要があります。このファイルを使用して、オーバークラウドのコンテナーイメージを取得するためのルールを定義します。その後、更新をデプロイする前に、新しい **OpenStackConfigGenerator** CR を生成する必要があります。

手順

1. コンテナー準備ファイル (**containers-prepare-parameter.yaml**) を開き、正しいイメージバージョンが取得されることを確認します。

2. 次のロール固有のオーバーライドを **containers-prepare-paramater.yaml** ファイルから削除します。

```
  ControllerContainerImagePrepare: *container_image_prepare_rhel8
  ComputeContainerImagePrepare: *container_image_prepare_rhel8
```

3. **roles_data.yaml** ファイルを開き、**OS::TripleO::Services::NovaLibvirtLegacy** を **OS::TripleO::Services::NovaLibvirt** に置き換えます。

4. **skip_rhel_release.yaml** という名前の環境ファイルを作成し、次の設定を追加します。

```
parameter_defaults:
  SkipRhelEnforcement: false
```

5. **system_upgrade.yaml** という名前の環境ファイルを作成し、次の設定を追加します。

```
parameter_defaults:
  NICsPrefixesToUdev: ['en']
  UpgradeLeappDevelSkip: "LEAPP_UNSUPPORTED=1
LEAPP_DEVEL_SKIP_CHECK_OS_RELEASE=1 LEAPP_NO_NETWORK_RENAMING=1
LEAPP_DEVEL_TARGET_RELEASE=9.2"
  UpgradeLeappDebug: false
  UpgradeLeappEnabled: true
  LeappActorsToRemove:
['checkifcfg','persistentnetnamesdisable','checkinstalledkernels','biosdevname']
  LeappRepoInitCommand: |
    sudo subscription-manager repos --disable=*
    subscription-manager repos --enable rhel-8-for-x86_64-baseos-tus-rpms --enable rhel-8-
for-x86_64-appstream-tus-rpms --enable openstack-17.1-for-rhel-8-x86_64-rpms
    subscription-manager release --set=8.4
  UpgradeLeappCommandOptions:
"--enablerepo=rhel-9-for-x86_64-baseos-eus-rpms --enablerepo=rhel-9-for-x86_64-
appstream-eus-rpms --enablerepo=rhel-9-for-x86_64-highavailability-eus-rpms --
enablerepo=openstack-17.1-for-rhel-9-x86_64-rpms --enablerepo=fast-datapath-for-rhel-9-
x86_64-rpms"
  LeappInitCommand: |
    sudo subscription-manager repos --disable=*
    sudo subscription-manager repos
  --enable=rhel-9-for-x86_64-baseos-eus-rpms --enable=rhel-9-for-x86_64-appstream-eus-
rpms --enable=rhel-9-for-x86_64-highavailability-eus-rpms --enable=openstack-17.1-for-rhel-
9-x86_64-rpms --enable=fast-datapath-for-rhel-9-x86_64-rpms

    leapp answer --add --section check_vdo.confirm=True

    dnf -y remove irb
```

推奨される Leapp パラメーターの詳細は、**16.2 から 17.1 へのアップグレードフレームワーク** ガイドの アップグレードのパラメーター を参照してください。

6. **system-upgrade** という名前の新しい **OpenStackConfigGenerator** CR を作成して、更新した heat 環境 ConfigMap と tripleo tarball ConfigMap を含めます。

7. ワークステーション上に **openstack-controller0-upgrade.yaml** という名前のファイルを作成して、最初のコントローラーノードの **OpenStackDeploy** CR を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: system-upgrade-controller-0
spec:
  configVersion: <config_version>
  configGenerator: system-upgrade
  mode: upgrade
  advancedSettings:
    limit: Controller[0]
    tags:
    - system_upgrade
```

8. **openstack-controller0-upgrade.yaml** ファイルを保存します。

9. コントローラー 0 でシステムアップグレードを実行するための **OpenStackDeploy** リソースを作成します。

```
$ oc create -f openstack-controller0-upgrade.yaml -n openstack
```

10. デプロイが完了するまで待ちます。

11. ワークステーション上に **openstack-controller1-upgrade.yaml** という名前のファイルを作成して、2 番目のコントローラーノードの **OpenStackDeploy** CR を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: system-upgrade-controller-1
spec:
  configVersion: <config_version>
  configGenerator: system-upgrade
  mode: upgrade
  advancedSettings:
    limit: Controller[1]
    tags:
    - system_upgrade
```

12. **openstack-controller1-upgrade.yaml** ファイルを保存します。

13. コントローラー 1 でシステムアップグレードを実行するための **OpenStackDeploy** リソースを作成します。

```
$ oc create -f openstack-controller1-upgrade.yaml -n openstack
```

14. デプロイが完了するまで待ちます。

15. ワークステーション上に **openstack-controller2-upgrade.yaml** という名前のファイルを作成して、3 番目のコントローラーノードの **OpenStackDeploy** CR を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: system-upgrade-controller-2
spec:
  configVersion: <config_version>
  configGenerator: system-upgrade
  mode: upgrade
  advancedSettings:
    limit: Controller[2]
    tags:
    - system_upgrade
```

16. **openstack-controller2-upgrade.yaml** ファイルを保存します。

17. コントローラー 1 でシステムアップグレードを実行するための **OpenStackDeploy** リソースを作成します。

```
$ oc create -f openstack-controller2-upgrade.yaml -n openstack
```

18. デプロイが完了するまで待ちます。

19. ワークステーション上に **openstack-computes-upgrade.yaml** という名前のファイルを作成して、すべての Compute ノードをアップグレードする **OpenStackDeploy** CR を定義します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackDeploy
metadata:
  name: system-upgrade-computes
spec:
  configVersion: <config_version>
  configGenerator: system-upgrade
  mode: upgrade
  advancedSettings:
    limit: Compute
    tags:
    - system_upgrade
```

20. **openstack-computes-upgrade.yaml** ファイルを保存します。

21. Compute ノードでシステムアップグレードを実行するための **OpenStackDeploy** リソースを作成します。

```
$ oc create -f openstack-computes-upgrade.yaml -n openstack
```

22. デプロイが完了するまで待ちます。

18.8. アップグレード後のタスクの実行

アップグレードを完了するには、オーバークラウドのアップグレードが正常に完了した後に、アップグレード後のタスクをいくつか実行する必要があります。

手順

1. **OpenStackProvisionServer** CR および **OpenStackBaremetalSet** CR の **baseImageUrl** パラメーターを RHEL 9.2 ゲストイメージに更新します。

2. コントローラーでフェンシングを再度有効にします。

```
$ oc rsh -n openstack openstackclient
$ ssh controller-0.ctlplane "sudo pcs property set stonith-enabled=true"
```

3. 環境に関連するその他のアップグレード後の操作を実行します。詳細は、**16.2 から 17.1 へのアップグレードフレームワーク** ガイドの アップグレード後操作の実施 を参照してください。