Red Hat OpenStack Platform 16.2

OpenShift Container Platform 向けの RHOSP director Operator

OpenShift Container Platform クラスターでの Red Hat OpenStack Platform オーバークラウドのデプロイ

Last Updated: 2022-05-25

Red Hat OpenStack Platform 16.2 OpenShift Container Platform 向けの RHOSP director Operator

OpenShift Container Platform クラスターでの Red Hat OpenStack Platform オーバークラウドのデプロイ

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

RHOSP director Operator を OpenShift Container Platform クラスターにインストールし、Operator を使用して RHOSP オーバークラウドをデプロイする方法について説明します。 この機能は、本リリースでは テクノロジープレビュー として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト目的のみでご利用いただく機能で、実稼働環境にデプロイすべきではありません。

目次

前書き

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、弊社 の CTO、Chris Wright のメッセージ を参照してください。

RED HAT ドキュメントへのフィードバック (英語のみ)

弊社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。

ドキュメントへのダイレクトフィードバック (DDF) 機能の使用 (英語版のみ)

特定の文章、段落、またはコードブロックに対して直接コメントを送付するには、DDF の **Add Feedback** 機能を使用してください。なお、この機能は英語版のドキュメントでのみご利用いただけます。

1. **Multi-page HTML** 形式でドキュメントを表示します。
2. ドキュメントの右上隅に **Feedback** ボタンが表示されていることを確認してください。
3. コメントするテキスト部分をハイライト表示します。
4. **Add Feedback** をクリックします。
5. **Add Feedback** フィールドにコメントを入力します。
6. (オプション) ドキュメントチームが連絡を取り問題についてお伺いできるように、ご自分のメールアドレスを追加します。
7. **送信** をクリックします。

テクノロジープレビュー

この機能は、本リリースでは **テクノロジープレビュー** として提供しているため、Red Hat では全面的にはサポートしていません。これは、テスト目的のみでご利用いただく機能で、実稼働環境にデプロイすべきではありません。テクノロジープレビュー機能についての詳しい情報は、「対象範囲の詳細」を参照してください。

第1章 RED HAT OPENSTACK PLATFORM DIRECTOR OPERATOR の概要

OpenShift Container Platform (OCP) は、Operator のモジュラーシステムを使用して OCP クラスターの機能を拡張します。Red Hat OpenStack Platform (RHOSP) director Operator は、OCP 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。この Operator は、RHOSP ノードのインフラストラクチャーおよび設定を管理してデプロイするためのカスタムリソース定義 (CRD) のセットを管理します。operator 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

director Operator は、コントローラーノードとして機能するように OpenShift Virtualization に仮想マシンのセットを作成します。

ベアメタルマシンのプロビジョニング

director Operator は、OCP ベアメタルマシン管理を使用して、operator でデプロイされた RHOSP クラウドにコンピュートノードをプロビジョニングします。

ネットワーキング

director Operator は、RHOSP サービスの基礎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

director Operator は、カスタム Heat 設定を OCP 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、director Operator は Ansible Playbook を自動的に再生成します。

CLI クライアント

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

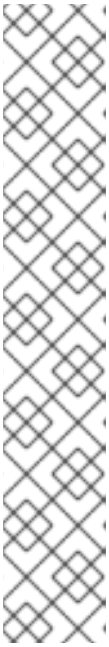
関連情報

- Red Hat OpenShift 上の Operator

1.1. DIRECTOR OPERATOR の前提条件

Red Hat OpenStack Platform (RHOSP) director Operator をインストールする前に、以下の前提条件のタスクを完了する必要があります。

- 有効な **ベアメタル** のクラスター Operator と **プロビジョニング** ネットワークを含む OpenshiftContainer Platform (OCP) 4.6 以降のクラスターをインストールします。



注記

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 OCP 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、**ベアメタル** 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 OCP 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。

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。

OCP 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 > Global Configuration > Infrastructure** の順に移動し、**YAML** ビューに切り替えて、**Conditions** セクションまでスクロールして **status.platformStatus** の値を表示します。

- OCP クラスターで、以下の前提条件の Operator を OperatorHub からインストールします。
  - OpenShift Virtualization 2.6 以降
  - SR-IOV ネットワーク Operator
- director オペレーターのリモート Git リポジトリーを設定して、オーバークラウド向けに生成された設定を保存します。
- 以下の永続ボリュームを作成し、director Operator が作成する以下の永続ボリューム要求 (PVC) に対応します。
  - 4G (**openstackclient-cloud-admin**)
  - 1G (**openstackclient-hosts**)
  - 50G (director Operator がコントローラー仮想マシンごとに複製するベースイメージの場合)
  - 最低 50 GB (コントローラー仮想マシンごと)

関連情報

- 「Operator のクラスターへの追加」

1.2. DIRECTOR OPERATOR のインストール

director Operator をインストールするには、Operator の namespace を作成し、以下の 3 つのリソースを namespace 内に作成する必要があります。

- CatalogSource。director Operator 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。
- サブスクリプション。director Operator カタログの変更を追跡します。
- OperatorGroup。director Operator の Operator グループを定義し、director Operator をターゲット namespace に制限します。

前提条件

前提条件

- OpenShift Container Platform クラスターが機能していることを確認する。
- OperatorHub から以下の前提条件 Operator をインストールする。
  - OpenShift Virtualization 2.6 以降
  - SR-IOV ネットワーク Operator
- **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ることを確認する。

手順

1. **openstack** namespace を作成します。

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

2. **osp-director-operator.yaml** という名前のファイルを作成し、3 つのリソースを設定して director Operator をインストールするように設定するための以下の YAML コンテンツを追加します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: osp-director-operator-index
  namespace: openstack
spec:
  sourceType: grpc
  image: quay.io/openstack-k8s-operators/osp-director-operator-index:1.0.0-1
---
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
  startingCSV: osp-director-operator.v0.0.1
  channel: alpha
```

3. **openstack** namespace 内に 3 つの新規リソースを作成します。

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

検証

1. director Operator が正常に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
```

関連情報

- 「CLI を使用した OperatorHub からのインストール」

1.3. DIRECTOR OPERATOR のカスタムリソース定義

director Operator には、オーバークラウドのリソース管理に使用できるカスタムリソース定義 (CRD) のセットが含まれます。CRD 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 つのタイプがあります。

ハードウェアプロビジョニング CRD

openstackbaremetalsets
: Compute および Ceph Storage など、特定のオーバークラウドロールのベアメタルホストのセットを作成します。

openstackcontrolplanes
: オーバークラウドのコントロールプレーンを作成して、関連する **openstackvmsets** を管理します。

openstackipsets
: ネットワークおよびロールの IP アドレスのセットを定義します。OpenShift Container Platform (OCP) は、この CRD を使用して IP アドレスを管理します。

openstacknets
: **openstackvmset** および **openstackbaremetalset** リソースに IP アドレスを割り当てるネットワークを作成します。

openstackprovisionservers
: ベアメタルプロビジョニング用のカスタムイメージを提供します。

openstackvmsets
: Controller、Database、Networker などの特定のオーバークラウドロールの仮想マシンセットを作成します。

ソフトウェア設定 CRD

openstackplaybookgenerators
: デプロイ用の Ansible Playbook を作成し、オーバークラウドをスケーリングしたり、カスタム ConfigMapに変更を加えたりすると、Playbook を再生成します。

openstackclients
: デプロイメントおよび管理コマンドを実行する Pod を作成します。

director Operator CRD の表示

- **oc get crd** コマンドを使用してこれらの CRD の一覧を表示します。

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

- **oc describe crd** コマンドを使用して、特定の CRD の定義を表示します。

```
$ 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
...
```

CRD の命名規則

各 CRD には **spec.names** セクションに複数の名前が含まれます。アクションのコンテキストに応じて、これらの名前を使用します。

- 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
```

関連情報

- カスタムリソース定義からのリソース管理

1.4. DIRECTOR OPERATOR を使用したオーバークラウドデプロイメントのワークフロー

Red Hat OpenStack Platform director Operator のインストール後に、director Operator 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、オーバークラウドの設定の生成、オーバークラウドの作成を行うことができます。

以下のワークフローで、オーバークラウド作成の一般的なプロセスを概説します。

1. コントロールプレーンおよびすべての分離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。
2. 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。
3. 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。
4. ベアメタルのコンピュートノードを作成します。
5. ジェネレーターを作成して、オーバークラウドの設定用の Ansible Playbook をレンダリングします。
6. Ansible Playbook の設定をオーバークラウドノードに適用します。

パート I. 一般的なデプロイメント操作

第2章 DIRECTOR OPERATOR を使用したオーバークラウドのデプロイメントの準備

director Operator を使用してオーバークラウドをデプロイする前に、ベースオペレーティングシステム用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情報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。

2.1. 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のベースオペレーティングシステムのイメージを保存するには、OpenShift Container Platform (OCP) クラスターでデータボリュームを作成する必要があります。

前提条件

- Red Hat Enterprise Linux 8 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。このイメージは、Red Hat カスタマーポータルの製品のダウンロードセクションからダウンロードできます。
- **virtctl** クライアントツールをワークステーションにインストールします。以下のコマンドを実行して、このツールを Red Hat Enterprise Linux ワークステーションにインストールできます。

```
$ sudo subscription-manager repos --enable=cnv-2.6-for-rhel-8-x86_64-rpms
$ sudo dnf install -y kubevirt-virtctl
```

- **virt-customize** クライアントツールをワークステーションにインストールします。このツールは、以下のコマンドを使用して Red Hat Enterprise Linux ワークステーションにインストールできます。

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

手順

1. access.redhat.com からダウンロードしたデフォルトの QCOW2 イメージでは、biosdev の予測可能なネットワークインターフェース名は使用されません。biosdev の予測可能なネットワークインターフェース名を使用するように、**virt-customize** でイメージを変更します。

```
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(kernelopts=.*\)net.ifnames=0 \(.*\)/\1\2/" /boot/grub2/grubenv'
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(GRUB_CMDLINE_LINUX=.*\)net.ifnames=0 \(.*\)/\1\2/" /etc/default/grub'
```

2. **virtctl** でイメージを OpenShift Virtualization にアップロードします。

```
$ virtctl image-upload dv openstack-base-img -n openstack --size=50Gi --image-path=<local
path to image> --storage-class <storage class> --insecure
```

--storage-class オプションの場合は、クラスターからストレージクラスを選択します。以下のコマンドを使用してストレージクラスの一覧を表示します。

```
$ oc get storageclass
```

3. OpenStackControlPlane リソースおよび個別の OpenStackVmSet リソースの作成時に、**baseImageVolumeName** パラメーターをデータボリューム名に設定します。

```
...
spec:
  ...
  baseImageVolumeName: openstack-base-img
...
```

関連情報

- virtctl ツールを使用したローカルディスクイメージのアップロード

2.2. リモート GIT リポジトリーの認証情報の追加

director Operator 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-secret** という名前の OpenShift Secret リソースとして Git リポジトリーの詳細を指定する必要があります。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- director オペレーターのリモート Git リポジトリーを作成して、オーバークラウド向けに生成された設定を保存します。
- SSH キーペアを準備します。公開鍵を Git リポジトリーにアップロードし、秘密鍵を利用できる状態にして、**git-secret** Secret リソースに追加します。

手順

- Secret リソースを作成します。

```
$ oc create secret generic git-secret -n openstack --from-file=git_ssh_identity=
<path_to_private_SSH_key> --from-literal=git_url=<git_server_URL>
```

git-secret Secret リソースには、2 つのキーと値のペアが含まれます。

git_ssh_identity

Git リポジトリーにアクセスするための秘密鍵**--from-file** オプションは、SSH 秘密鍵ファイルの内容を保存します。

git_url

設定を保存する git リポジトリーの SSH URL。**--from-literal** オプションは、このキーに入力した URL を保存します。

検証

- Secret リソースを表示します。

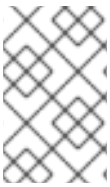
```
$ oc get secret/git-secret -n openstack
```

関連情報

- Pod への機密データの指定

2.3. 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。



注記

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。

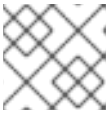
前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。

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
```



注記

-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: userpassword
  namespace: openstack
data:
  NodeRootPassword: "cEBzc3cwcmQh"
```

NodeRootPassword パラメーターを base64 でエンコードされたパスワードに設定します。

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 への機密データの指定

第3章 DIRECTOR OPERATOR を使用したネットワークの作成

OpenStackNet リソースを使用して OpenShift Virtualization ワーカーノードでネットワークおよびブリッジを作成し、仮想マシンをこれらのネットワークに接続します。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には、オーバークラウドおよび追加のネットワーク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。

3.1. OPENSTACKNET での仮想マシンのブリッジについて

OpenStackVMSet リソースで仮想マシンを作成する場合には、これらの仮想マシンを関連する Red Hat OpenStack Platform (RHOSP) ネットワークに接続する必要があります。OpenStackNet リソースには **nodeNetworkConfigurationPolicy** オプションが含まれており、ネットワークインターフェースデータを OpenShift Virtualization 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 OCP ワーカーノードでネットワーク設定の最終状態を設定します。この方法では、OCP ノードでブリッジを作成して、コントローラー仮想マシンを RHOSP ネットワークに接続できます。

たとえば、コントロールプレーンネットワークを作成し、**nodeNetworkConfigurationPolicy** オプションを設定して Linux ブリッジを作成し、各ワーカーの NIC にブリッジに接続する場合には、NodeNetworkConfigurationPolicy リソースは、この必要な終了状態になるように、それぞれの OCP ワーカーノードを設定します。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: ctlplane
spec:
  ...
  attachConfiguration:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
    nodeNetworkConfigurationPolicy:
      ...
      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
```

この設定を適用すると、各ワーカーに **br-osp** という名前の新規ブリッジが追加され、各ホストの **enp6s0** NIC に接続されます。すべての RHOSP コントローラー仮想マシンは、コントロールプレーンの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。

後に VLAN 20 経由で Internal API ネットワークを作成する場合は、**nodeNetworkConfigurationPolicy** オプションを設定して、各 OCP 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: OpenStackNet
metadata:
  name: internalapi
spec:
  ...
  vlan: 20
  attachConfiguration:
    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
```

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、OpenStackNet は VLAN 20 をこのネットワークに関連付けるので、RHOSP コントローラーの仮想マシンは、内部 API ネットワークトラフィック用の **br-osp** ブリッジ上の VLAN 20 に接続できます。

OpenStackVMSet リソースで仮想マシンを作成すると、仮想マシンは、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ース) 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、OpenStackNet を使用してデフォルトの RHOSP ネットワークを作成する場合に、コントローラー仮想マシンのインターフェース設定は以下の例のようになります。

```
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ー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ースへのマッピング

ネットワーク	インターフェース
default	nic1
ctlplane	nic2
external	nic3
InternalApi	nic4
storage	nic5
StorageMgmt	nic6
テナント	nic7

コントローラー NIC 用の heat テンプレート作成時には、テンプレートに各ネットワークに対応する NIC 構成が含まれていることを確認してください。

```
...
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:

              ## NIC2 - Control Plane ##
              - type: interface
                name: nic2
                ...

              ## NIC3 - External ##
              - type: ovs_bridge
                name: bridge_name
                ...
                members:
```

```
    - type: interface
      name: nic3

  ## NIC4 - Internal API ##
  - type: interface
    name: nic4
    ...

  ## NIC5 - Storage ##
  - type: interface
    name: nic5
    ...

  ## NIC6 - StorageMgmt ##
  - type: interface
    name: nic6
    ...

  ## NIC7 - Tenant ##
  - type: ovs_bridge
    name: br-isolated
    ...
    members:
    - type: interface
      name: nic7
      ...
```

関連情報

- ノードのネットワーク設定の更新

3.2. OPENSTACKNET を使用したオーバークラウドのコントロールプレーンネットワークの作成

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。OpenStackNet リソースには、IP アドレスの割り当てに加えて、OpenShift Virtualization が仮想マシンをネットワークにアタッチするために使用するネットワーク設定ポリシーを定義する情報が含まれます。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。

手順

1. ワークステーションに **ctlplane-network.yaml** という名前のファイルを作成します。**ctlplane** という名前のコントロールプレーンネットワークのリソース仕様を含めます。たとえば、各ワーカーノードの **enp6s0** イーサネットデバイスに接続された Linux ブリッジを使用するコントロールプレーンの仕様は以下のようになります。

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

```
kind: OpenStackNet
metadata:
  name: ctlplane
spec:
  cidr: 192.168.25.0/24
  allocationStart: 192.168.25.100
  allocationEnd: 192.168.25.250
  gateway: 192.168.25.1
  attachConfiguration:
    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
```

リソース仕様に以下の値を設定します。

metadata.name

コントロールプレーンネットワークの名 (**ctlplane**) を設定します。

spec

コントロールプレーンネットワークのネットワーク設定を指定します。このセクションで使用できる値の詳細は、**openstacknet** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstacknet
```

ネットワーク仕様の設定が完了したら、ファイルを保存します。

2. コントロールプレーンネットワークを作成します。

```
$ oc create -f ctlplane-network.yaml -n openstack
```

検証

- コントロールプレーンネットワークのリソースを表示します。

```
$ oc get openstacknet/ctlplane
```

3.3. OPENSTACKNET を使用したネットワーク分離用の VLAN ネットワークの作成

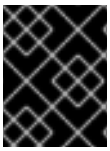
コンポーザブルネットワークのネットワーク分離を実装するには、追加のネットワークを作成する必要

があります。このネットワーク分離を実現するには、コンポーザブルネットワークを個別の VLAN ネットワークに配置できます。OpenStackNet リソースには、IP アドレスの割り当てに加えて、OpenShift Virtualization が仮想マシンを VLAN ネットワークにアタッチするために使用するネットワーク設定ポリシーを定義する情報が含まれます。

デフォルトの Red Hat OpenStack Platform ネットワークを使用するには、ネットワークごとに OpenStackNet リソースを作成する必要があります。

表3.2 デフォルトの Red Hat OpenStack Platform ネットワーク

ネットワーク	VLAN	CIDR	割り当て
外部	10	10.0.0.0/24	10.0.0.4 - 10.0.0.250
InternalApi	20	172.16.2.0/24	172.16.2.4 - 172.16.2.250
ストレージ	30	172.16.1.0/24	172.16.1.4 - 172.16.1.250
StorageMgmt	40	172.16.3.0/24	172.16.3.4 - 172.16.3.250
Tenant	50	172.16.0.0/24	172.16.0.4 - 172.16.0.250



重要

各ネットワークごとに異なるネットワークの詳細を使用するには、カスタム **network_data.yaml** ファイルを作成する必要があります。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。

手順

1. 新規ネットワークのファイルを作成します。たとえば、内部 API ネットワークの場合は、ワークステーションに **internalapi-network.yaml** という名前のファイルを作成します。VLAN ネットワークのリソース仕様を含めます。たとえば、以下は、各ワーカーノードの **enp6s0** イーサネットデバイスに接続された Linux ブリッジ経由で VLAN タグ付けされたトラフィックを管理する内部 API ネットワークの仕様です。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: internalapi
spec:
  cidr: 172.16.2.0/24
  vlan: 20
  allocationStart: 172.16.2.4
  allocationEnd: 172.16.2.250
  attachConfiguration:
    nodeNetworkConfigurationPolicy:
```

```
    nodeSelector:
      node-role.kubernetes.io/worker: ""
    desiredState:
      interfaces:
      - bridge:
          options:
            stp:
              enabled: false
          port:
          - name: enp6s0
        description: Linux bridge with enp7s0 as a port
        name: br-osp
        state: up
        type: linux-bridge
```

linux-bridge でネットワーク分離に VLAN を使用する場合に、以下のような処理が行われます。

- director Operator は、リソースに指定されたブリッジインターフェースの Node Network Configuration Policy を作成します。このポリシーは、**nmstate** を使用してワーカーノードにブリッジを設定します。
- director Operator は、Multus CNI プラグイン設定を定義するネットワークごとにネットワーク接続定義を作成します。ネットワーク接続定義で VLAN ID を指定する場合には、Multus CNI プラグインはブリッジで **vlan-filtering** を有効にします。
- director Operator は、仮想マシン上の各ネットワーク専用のインターフェースを割り当てます。これは、**OSVMSet** のネットワークテンプレートがマルチ NIC ネットワークテンプレートであることを意味します。

リソース仕様に以下の値を設定します。

metadata.name

コントロールプレーンネットワークの名 (**ctlplane**) を設定します。

spec

コントロールプレーンネットワークのネットワーク設定を指定します。このセクションで使用できる値の詳細は、**openstacknet** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstacknet
```

ネットワーク仕様の設定が完了したら、ファイルを保存します。

2. 内部 API ネットワークを作成します。

```
$ oc create -f internalapi-network.yaml -n openstack
```

検証

- 内部 API ネットワークのリソースを表示します。

```
$ oc get openstacknet/internalapi -n openstack
```

第4章 DIRECTOR OPERATOR を使用した HEAT テンプレートおよび環境ファイルの追加

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する場合には、希望するカスタマイズ似合わせて heat テンプレートおよび環境ファイルを作成する必要があります。デプロイメントにカスタムのテンプレートおよび環境ファイルを追加して、オーバークラウドを設定できます。director Operator のコンテキストでは、オーバークラウドのデプロイメントを実行する前にこれらのファイルを ConfigMap リソースに保存します。

4.1. DIRECTOR OPERATOR を使用したカスタムテンプレートの使用方法について

director Operator は、各ノードで Red Hat OpenStack Platform ソフトウェアを設定する準備が整うと、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。

独自のカスタムテンプレート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、**tripleo-tarball-config** という名前の OpenShift ConfigMap に tarball ファイルのバイナリーコンテンツを組み込みます。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。director Operator 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。

たとえば、tarball ファイルに以下の YAML ファイルが含まれる場合があります。

```
$ tar -tf custom-config.tar.gz
custom-template.yaml
net-config-static-bridge.yaml
net-config-static.yaml
network_data.yaml
```

custom-template.yaml ファイルは、新しいファイルで、既存のテンプレートを上書きしません。ただし、**net-config-static-bridge.yaml** ファイルおよび **net-config-static.yaml** ファイルは、事前にプロビジョニングされたノードネットワーク設定のデフォルトの heat テンプレートを、**network_data.yaml** ファイルはデフォルトのコンポーザブルネットワーク設定を上書きします。

関連情報

- Heat テンプレート

4.2. 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

カスタムテンプレートを tarball ファイルにアーカイブし、これらのテンプレートをオーバークラウドデプロイメントの一部として追加できるようにします。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- プロビジョニングしたノードに適用するカスタムテンプレートを作成します。

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 tarball にアーカイブします。

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

3. **tripleo-tarball-config** ConfigMap 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
```

検証

- ConfigMap を表示します。

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

関連情報

- configmap の作成および使用
- Heat テンプレート

4.3. DIRECTOR OPERATOR を使用したカスタム環境ファイルの使用方法について

オーバークラウドに機能またはパラメーターを設定するには、デプロイメントの実行に環境ファイルを追加する必要があります。director Operator は **heat-env-config** という名前の ConfigMap を使用して環境ファイルを保存し、取得します。**heat-env-config** ConfigMap のデータには、以下の構文を使用します。

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

たとえば、**heat-env-config** ConfigMap には 2 つの環境ファイルが含まれる可能性があります。

```
...
data:
  network_environment.yaml: |+
    resource_registry:
      OS::TripleO::Controller::Net::SoftwareConfig: net-config-static-bridge.yaml
      OS::TripleO::Compute::Net::SoftwareConfig: net-config-static-bridge-compute.yaml
  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
```

- 最初の環境ファイルは **network_environment.yaml** という名前で、このファイルにはネットワークインターフェース設定を適切な heat テンプレートにマッピングする **resource_registry** セクションが含まれています。
- 2 番目の環境ファイルは **cloud_name.yaml** という名前で、オーバークラウドのホスト名に関連するパラメーターを設定する **parameter_defaults** セクションが含まれます。
- director Operator がオーバークラウドをデプロイする時に、Operator には **heat-env-config** ConfigMap からの両方のファイルがデプロイメントで追加されます。

関連情報

- 環境ファイル

4.4. 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

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 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

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- 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を作成します。

手順

1. **heat-env-config** ConfigMap を作成し、環境ファイルが含まれるディレクトリーをデータとして使用します。

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

検証

- ConfigMap を表示します。

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

関連情報

- configmap の作成および使用
- 環境ファイル

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 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コンピュートノードなど、複数のノードで構成されます。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 つのコンピュートノードが必要です。OpenStackBaremetalSet リソースでコンピュートノードを、OpenStackControlPlane リソースでコントローラーノードを作成することができます。

5.1. OPENSTACKCONTROLPLANE でのコントロールプレーンの作成

オーバークラウドのコントロールプレーンには、オーバークラウドの機能を管理するメインの Red Hat OpenStack Platform サービスが含まれます。コントロールプレーンは、通常 3 つのコントローラーノードで構成されており、他のコントロールプレーンベースのコンポーザブルロールにスケーリングできます。コンポーザブルロールを使用する場合には、各サービスは 3 つの追加の専用ノードで実行される必要があり、Pacemaker のクォーラムを維持するために、コントロールプレーン内のノードの合計数を追加する必要があります。

OpenStackControlPlane カスタムリソースは、コントロールプレーンベースのノードを OpenShift Virtualization 内の仮想マシンとして作成します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackNet リソースを使用して、コントロールプレーンネットワークおよび追加の分離ネットワークを作成します。

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。たとえば、3 つのコントローラーノードで構成されるコントロールプレーンの仕様は以下のとおりです。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  openStackClientImageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  openStackClientNetworks:
        - ctlplane
        - external
        - internalapi
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  gitSecret: git-secret
  virtualMachineRoles:
    controller:
      roleName: Controller
```

```
    roleCount: 3
    networks:
      - ctlplane
      - internalapi
      - external
      - tenant
      - storage
      - storagemgmt
    cores: 6
    memory: 12
    diskSize: 50
    baseImageVolumeName: openstack-base-img
    storageClass: host-nfs-storageclass
```

リソース仕様に以下の値を設定します。

metadata.name

オーバークラウドコントロールプレーンの名前を **overcloud** に設定します。

metadata.namespace

director Operator namespace を **openstack** に設定します。

spec

コントロールプレーンの設定を指定します。このセクションで使用できる値の詳細は、**openstackcontrolplane** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstackcontrolplane
```

コントロールプレーンの仕様の設定が完了したら、ファイルを保存します。

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
```

OCP が OpenStackControlPlane リソースに関連するリソースを作成するまで待機します。

director Operator は、OpenStackControlPlane リソースの一部として、リモートシェルからアクセスして 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. 仮想マシンリソースを表示し、OpenShift Virtualization でのコントロールプレーンの仮想マシンの作成を確認します。

```
$ oc get virtualmachines
```

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 openstackclient -n openstack
```

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

コンピュートノードは Red Hat OpenStack Platform 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コンピュートノードが少なくとも 1 台必要で、デプロイメント後にコンピュートノードの数をスケーリングできます。

OpenStackBaremetalSet カスタムリソースは、OpenShift Container Platform が管理するベアメタルマシンからコンピュートノードを作成します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackNet リソースを使用して、コントロールプレーンネットワークおよび追加の分離ネットワークを作成します。

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。コンピュートノードのリソース仕様を含めます。たとえば、コンピュートノード 1 つの仕様は以下のようになります。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: compute
  namespace: openstack
spec:
  count: 1
  baseImageUrl: http://host/images/rhel-image-8.4.x86_64.qcow2
  deploymentSSHSecret: osp-controlplane-ssh-keys
  ctlplaneInterface: enp2s0
  networks:
    - ctlplane
    - internalapi
    - tenant
    - storage
  roleName: Compute
  passwordSecret: userpassword
```

リソース仕様に以下の値を設定します。

metadata.name

コンピュートノードのベアメタルセットの名前を **overcloud** に設定します。

metadata.namespace

director Operator namespace を **openstack** に設定します。

spec

コンピュートノードの設定を定義します。このセクションで使用できる値の詳細は、**openstackbaremetalset** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstackbaremetalset
```

コンピュートノードの仕様の設定が完了したら、ファイルを保存します。

2. コンピュートノードを作成します。

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

検証

1. コンピュートノードのリソースを表示します。

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

2. OpenShift が管理するベアメタルマシンを表示し、コンピュートノードの作成を確認します。

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

第6章 DIRECTOR OPERATOR を使用したオーバークラウドソフトウェアの設定

オーバークラウドの仮想ノードおよびベアメタルノードをプロビジョニングしたら、オーバークラウドを設定できます。OpenStackPlaybookGenerator リソースを作成して Ansible Playbook を生成し、Red Hat カスタマーポータルまたは Red Hat Satellite のいずれかにノードを登録し、**tripleo-deploy.sh** スクリプトを実行して設定をノードに適用する必要があります。

6.1. OPENSTACKPLAYBOOKGENERATOR を使用したオーバークラウドの設定用の 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) ソフトウェアを設定する Ansible Playbook のセットを作成する必要があります。これらの Playbook を OpenStackPlaybookGenerator リソースで作成し、RHOSP director の **config-download** 機能を使用して heat 設定を Playbook に変換します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- リモート Git リポジトリーの認証情報が含まれる **git-secret** シークレットを設定します。
- カスタム heat テンプレートが含まれる **tripleo-tarball-config** ConfigMap を設定します。
- カスタム環境ファイルが含まれる **heat-env-config** ConfigMap を設定します。

手順

1. ワークステーションに **openstack-playbook-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。たとえば、Playbook を生成する仕様は以下のようになります。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackPlaybookGenerator
metadata:
  name: default
  namespace: openstack
spec:
  imageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  gitSecret: git-secret
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
```

リソース仕様に以下の値を設定します。

metadata.name

コンピュートノードのベアメタルセットの名前を **default** に設定します。

metadata.namespace

director Operator namespace を **openstack** に設定します。

spec.gitSecret

Git 認証認証情報 (**git-secret**) を含む ConfigMap に設定します。

spec.tarballConfigMap

カスタムの heat テンプレートを使用して tarball が含まれる ConfigMap (**tripleo-tarball-config**) に設定します。

spec.heatEnvConfigMap

カスタム環境ファイルが含まれる ConfigMap (**heat-env-config**) に設定します。

spec セクションで使用できる値の詳細は、**openstackplaybookgenerator** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstackplaybookgenerator
```

Ansible Playbook ジェネレーター仕様の設定が完了したら、ファイルを保存します。

2. Ansible Playbook ジェネレーターを作成します。

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

検証

- Playbook ジェネレーターのリソースを表示します。

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

6.2. 一時 HEAT コンテナーイメージのソースパラメーター

一時 heat サービスを作成するには、OpenStackPlaybookGenerator リソースに registry.redhat.io からの固有のコンテナーイメージが 4 つ必要です。

- **openstack-heat-api**
- **openstack-heat-engine**
- **openstack-mariadb**
- **openstack-rabbitmq**

これらのイメージのソースの場所は、**spec.ephemeralHeatSettings** パラメーターで変更できます。たとえば、これらのイメージまたは Red Hat Satellite Server をホストする場合には、これらのイメージのソースとして Red Hat Satellite Server を使用するように、**spec.ephemeralHeatSettings** パラメーターおよびサブパラメーターを変更できます。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackPlaybookGenerator
metadata:
  name: default
  namespace: openstack
spec:
  imageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  gitSecret: git-secret
  heatEnvConfigMap: heat-env-config
```

```
  tarballConfigMap: tripleo-tarball-config
  ephemeralHeatSettings:
    heatAPIImageURL: <heat_api_image_location>
    heatEngineImageURL: <heat_engine_image_location>
    mariadbImageURL: <mariadb_image_location>
    rabbitImageURL: <rabbitmq_image-location>
```

リソース仕様に以下の値を設定します。

spec.ephemeralHeatSettings.heatAPIImageURL

heat API のイメージの場所。

spec.ephemeralHeatSettings.heatEngineImageURL

heat エンジンのイメージの場所。

spec.ephemeralHeatSettings.mariadbImageURL

MariaDB のイメージの場所。

spec.ephemeralHeatSettings.rabbitImageURL

RabbitMQ のイメージの場所。

6.3. PLAYBOOK 生成のインタラクティブモードパラメーター

Playbook 生成操作をデバッグするには、OpenStackPlaybookGenerator リソースを対話モードを使用するように設定できます。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackPlaybookGenerator
metadata:
  name: default
  namespace: openstack
spec:
  imageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  gitSecret: git-secret
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
  interactive: true
```

このモードでは、OpenStackPlaybookGenerator リソースは、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。

6.4. TRIPLEO-DEPLOY.SH スクリプトの使用

OpenStackControlPlane リソースの一部として、director Operator はリモートシェル経由でアクセスする OpenStackClient Pod を作成し、Red Hat OpenStack Platform コマンドを実行します。この Pod には **tripleo-deploy.sh** という名前のスクリプトが含まれており、この設定を更新して、最新の設定をオーバークラウドノードに適用します。

以下のオプションを指定して **tripleo-deploy.sh** スクリプトを実行します。

-d

Git diff は、レンダリングされた Playbook の最新バージョンを、以前の対応バージョンと比較します。

-a

レンダリングされた Playbook の利用可能な最新バージョンを受け入れて **latest** とタグ付けします。

-p

オーバークラウドノードに対して Ansible Playbook を適用します。

6.5. 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 がノードにオーバークラウドソフトウェアを設定する前に、全ノードのオペレーティングシステムを Red Hat カスタマーポータルまたは Red Hat Satellite Server のいずれかに登録して、ノードのリポジトリーを有効にする必要があります。ノードを登録するには、OpentackPlaybookGenerator リソースで作成されるインベントリーファイルを使用して、**redhat_subscription** Ansible モジュールを使用します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackControlPlane リソースを使用してコントロールプレーンを作成します。
- OpenStackBareMetalSet リソースを使用して、ベアメタルのコンピュートノードを作成します。
- OpentackPlaybookGenerator を使用して、オーバークラウドの Ansible Playbook 設定を作成します。

手順

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

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

2. **cloud-admin** ホームディレクトリーに移動します。

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

3. オプション: オーバークラウドの Ansible Playbook の diff をチェックします。

```
$ ./tripleo-deploy.sh -d
```

4. レンダリングされた Ansible Playbook の最新バージョンを受け入れて **latest** としてタグ付けします。

```
$ ./tripleo-deploy.sh -a
```

最新バージョンの Playbook には、オーバークラウド設定用のインベントリーファイルが含まれます。このインベントリーファイルは、OpenStackClient Pod の **/home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml** にあります。

5. 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-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      - ansible-2.9-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-16.1-for-rhel-8-x86_64-rpms
      - rhceph-4-mon-for-rhel-8-x86_64-rpms
      - fast-datapath-for-rhel-8-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 8.4
        pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
    - name: Disable all repos
      command: "subscription-manager repos --disable *"
    - name: Enable Controller node repos
      command: "subscription-manager repos --enable {{ item }}"
      with_items: "{{ repos }}"
```

このプレイには、以下の 3 つのタスクが含まれます。

- ノードを登録する。
- 自動的に有効化されるリポジトリーをすべて無効にする。
- コントローラーノードに関連するリポジトリーだけを有効にする。リポジトリーは **repos** 変数でリストされます。

6. 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-playpook -i /home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml ./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

6.6. 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コンピュート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 でオーバークラウドを設定できません。director Operator は、OpenStackControlPlane リソースの作成時に、リモートシェル経由でアクセスする OpenStackClient Pod を作成し、**tripleo-deploy.sh** スクリプトを実行してオーバークラウドを設定します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackControlPlane リソースを使用してコントロールプレーンを作成します。
- OpenStackBareMetalSet リソースを使用して、ベアメタルのコンピュートノードを作成します。
- OpentackPlaybookGenerator を使用して、オーバークラウドの Ansible Playbook 設定を作成します。

手順

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

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

2. **cloud-admin** ホームディレクトリーに移動します。

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

3. オプション: オーバークラウドの Ansible Playbook の diff をチェックします。

```
$ ./tripleo-deploy.sh -d
```

4. レンダリングされた Ansible Playbook の最新バージョンを受け入れて **latest** としてタグ付けします。

```
$ ./tripleo-deploy.sh -a
```

5. オーバークラウドノードに対して Ansible Playbook を適用します。

```
$ ./tripleo-deploy.sh -p
```

注記

openstackclient Pod 内の ~/**ansible.log** で Ansible 設定のログを表示できます。

パート II. デプロイメントシナリオ

第7章 DIRECTOR OPERATOR のデプロイメントシナリオ: ハイパーコンバージドインフラストラクチャー (HCI) が含まれるオーバークラウド

director Operator 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) を設定してオーバークラウドをデプロイできます。このシナリオでは、Compute サービスと Ceph Storage OSD サービスの両方を同じノードにインストールします。

前提条件

- OpenStackClient リソースの **openStackClientImageURL** に **quay.io/openstack-k8s-operators/rhosp16-openstack-tripleoclient:16.2_20210521.1** 以降のイメージを使用している。
- お使いのコンピュート 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。

7.1. 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のベースオペレーティングシステムのイメージを保存するには、OpenShift Container Platform (OCP) クラスターでデータボリュームを作成する必要があります。

前提条件

- Red Hat Enterprise Linux 8 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。このイメージは、Red Hat カスタマーポータルの製品のダウンロードセクションからダウンロードできます。
- **virtctl** クライアントツールをワークステーションにインストールします。以下のコマンドを実行して、このツールを Red Hat Enterprise Linux ワークステーションにインストールできます。

```
$ sudo subscription-manager repos --enable=cnv-2.6-for-rhel-8-x86_64-rpms
$ sudo dnf install -y kubevirt-virtctl
```

- **virt-customize** クライアントツールをワークステーションにインストールします。このツールは、以下のコマンドを使用して Red Hat Enterprise Linux ワークステーションにインストールできます。

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

手順

1. access.redhat.com からダウンロードしたデフォルトの QCOW2 イメージでは、biosdev の予測可能なネットワークインターフェース名は使用されません。biosdev の予測可能なネットワークインターフェース名を使用するように、**virt-customize** でイメージを変更します。

```
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(kernelopts=.*\)net.ifnames=0 \(.*\)/\1\2/" /boot/grub2/grubenv'
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(GRUB_CMDLINE_LINUX=.*\)net.ifnames=0 \(.*\)/\1\2/" /etc/default/grub'
```

2. **virtctl** でイメージを OpenShift Virtualization にアップロードします。

```
$ virtctl image-upload dv openstack-base-img -n openstack --size=50Gi --image-path=<local path to image> --storage-class <storage class> --insecure
```

--storage-class オプションの場合は、クラスターからストレージクラスを選択します。以下のコマンドを使用してストレージクラスの一覧を表示します。

```
$ oc get storageclass
```

3. OpenStackControlPlane リソースおよび個別の OpenStackVmSet リソースの作成時に、**baseImageVolumeName** パラメーターをデータボリューム名に設定します。

```
...
spec:
  ...
  baseImageVolumeName: openstack-base-img
...
```

関連情報

- virtctl ツールを使用したローカルディスクイメージのアップロード

7.2. リモート GIT リポジトリーの認証情報の追加

director Operator 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-secret** という名前の OpenShift Secret リソースとして Git リポジトリーの詳細を指定する必要があります。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- director オペレーターのリモート Git リポジトリーを作成して、オーバークラウド向けに生成された設定を保存します。
- SSH キーペアを準備します。公開鍵を Git リポジトリーにアップロードし、秘密鍵を利用できる状態にして、**git-secret** Secret リソースに追加します。

手順

- Secret リソースを作成します。

```
$ oc create secret generic git-secret -n openstack --from-file=git_ssh_identity=<path_to_private_SSH_key> --from-literal=git_url=<git_server_URL>
```

git-secret Secret リソースには、2 つのキーと値のペアが含まれます。

git_ssh_identity

Git リポジトリーにアクセスするための秘密鍵**--from-file** オプションは、SSH 秘密鍵ファイルの内容を保存します。

git_url

設定を保存する git リポジトリーの SSH URL。**--from-literal** オプションは、このキーに入力した URL を保存します。

検証

- Secret リソースを表示します。

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

関連情報

- Pod への機密データの指定

7.3. 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。

注記

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。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。

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
```

注記

-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: userpassword
```

```
  namespace: openstack
data:
  NodeRootPassword: "cEBzc3cwcmQh"
```

NodeRootPassword パラメーターを base64 でエンコードされたパスワードに設定します。

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 への機密データの指定

7.4. OPENSTACKNET を使用したオーバークラウドのコントロールプレーンネットワークの作成

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。OpenStackNet リソースには、IP アドレスの割り当てに加えて、OpenShift Virtualization が仮想マシンをネットワークにアタッチするために使用するネットワーク設定ポリシーを定義する情報が含まれます。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。

手順

1. ワークステーションに **ctlplane-network.yaml** という名前のファイルを作成します。**ctlplane** という名前のコントロールプレーンネットワークのリソース仕様を含めます。たとえば、各ワーカーノードの **enp6s0** イーサネットデバイスに接続された Linux ブリッジを使用するコントロールプレーンの仕様は以下のようになります。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: ctlplane
spec:
  cidr: 192.168.25.0/24
  allocationStart: 192.168.25.100
  allocationEnd: 192.168.25.250
  gateway: 192.168.25.1
  attachConfiguration:
    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
```

リソース仕様に以下の値を設定します。

metadata.name

コントロールプレーンネットワークの名 (**ctlplane**) を設定します。

spec

コントロールプレーンネットワークのネットワーク設定を指定します。このセクションで使用できる値の詳細は、**openstacknet** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstacknet
```

ネットワーク仕様の設定が完了したら、ファイルを保存します。

2. コントロールプレーンネットワークを作成します。

```
$ oc create -f ctlplane-network.yaml -n openstack
```

検証

- コントロールプレーンネットワークのリソースを表示します。

```
$ oc get openstacknet/ctlplane
```

7.5. OPENSTACKNET を使用したネットワーク分離用の VLAN ネットワークの作成

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には、追加のネットワークを作成する必要があります。このネットワーク分離を実現するには、コンポーザブルネットワークを個別の VLAN ネットワークに配置できます。OpenStackNet リソースには、IP アドレスの割り当てに加えて、OpenShift Virtualization が仮想マシンを VLAN ネットワークにアタッチするために使用するネットワーク設定ポリシーを定義する情報が含まれます。

デフォルトの Red Hat OpenStack Platform ネットワークを使用するには、ネットワークごとに OpenStackNet リソースを作成する必要があります。

表7.1 デフォルトの Red Hat OpenStack Platform ネットワーク

ネットワーク	VLAN	CIDR	割り当て
外部	10	10.0.0.0/24	10.0.0.4 - 10.0.0.250
InternalApi	20	172.16.2.0/24	172.16.2.4 - 172.16.2.250

ネットワーク	VLAN	CIDR	割り当て
ストレージ	30	172.16.1.0/24	172.16.1.4 - 172.16.1.250
StorageMgmt	40	172.16.3.0/24	172.16.3.4 - 172.16.3.250
Tenant	50	172.16.0.0/24	172.16.0.4 - 172.16.0.250

重要

各ネットワークごとに異なるネットワークの詳細を使用するには、カスタム **network_data.yaml** ファイルを作成する必要があります。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。

手順

1. 新規ネットワークのファイルを作成します。たとえば、内部 API ネットワークの場合は、ワークステーションに **internalapi-network.yaml** という名前のファイルを作成します。VLAN ネットワークのリソース仕様を含めます。たとえば、以下は、各ワーカーノードの **enp6s0** イーサネットデバイスに接続された Linux ブリッジ経由で VLAN タグ付けされたトラフィックを管理する内部 API ネットワークの仕様です。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: internalapi
spec:
  cidr: 172.16.2.0/24
  vlan: 20
  allocationStart: 172.16.2.4
  allocationEnd: 172.16.2.250
  attachConfiguration:
    nodeNetworkConfigurationPolicy:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp7s0 as a port
```

```
      name: br-osp
      state: up
      type: linux-bridge
```

linux-bridge でネットワーク分離に VLAN を使用する場合に、以下のような処理が行われます。

- director Operator は、リソースに指定されたブリッジインターフェースの Node Network Configuration Policy を作成します。このポリシーは、**nmstate** を使用してワーカーノードにブリッジを設定します。
- director Operator は、Multus CNI プラグイン設定を定義するネットワークごとにネットワーク接続定義を作成します。ネットワーク接続定義で VLAN ID を指定する場合には、Multus CNI プラグインはブリッジで **vlan-filtering** を有効にします。
- director Operator は、仮想マシン上の各ネットワーク専用のインターフェースを割り当てます。これは、**OSVMSet** のネットワークテンプレートがマルチ NIC ネットワークテンプレートであることを意味します。

リソース仕様に以下の値を設定します。

metadata.name

コントロールプレーンネットワークの名 (**ctlplane**) を設定します。

spec

コントロールプレーンネットワークのネットワーク設定を指定します。このセクションで使用できる値の詳細は、**openstacknet** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstacknet
```

ネットワーク仕様の設定が完了したら、ファイルを保存します。

2. 内部 API ネットワークを作成します。

```
$ oc create -f internalapi-network.yaml -n openstack
```

検証

- 内部 API ネットワークのリソースを表示します。

```
$ oc get openstacknet/internalapi -n openstack
```

7.6. OPENSTACKCONTROLPLANE でのコントロールプレーンの作成

オーバークラウドのコントロールプレーンには、オーバークラウドの機能を管理するメインの Red Hat OpenStack Platform サービスが含まれます。コントロールプレーンは、通常 3 つのコントローラーノードで構成されており、他のコントロールプレーンベースのコンポーザブルロールにスケーリングできます。コンポーザブルロールを使用する場合には、各サービスは 3 つの追加の専用ノードで実行される必要があり、Pacemaker のクォーラムを維持するために、コントロールプレーン内のノードの合計数を追加する必要があります。

OpenStackControlPlane カスタムリソースは、コントロールプレーンベースのノードを OpenShift Virtualization 内の仮想マシンとして作成します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackNet リソースを使用して、コントロールプレーンネットワークおよび追加の分離ネットワークを作成します。

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。たとえば、3 つのコントローラーノードで構成されるコントロールプレーンの仕様は以下のとおりです。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  openStackClientImageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  openStackClientNetworks:
        - ctlplane
        - external
        - internalapi
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  gitSecret: git-secret
  virtualMachineRoles:
    controller:
      roleName: Controller
      roleCount: 3
      networks:
        - ctlplane
        - internalapi
        - external
        - tenant
        - storage
        - storagemgmt
      cores: 6
      memory: 12
      diskSize: 50
      baseImageVolumeName: openstack-base-img
      storageClass: host-nfs-storageclass
```

リソース仕様に以下の値を設定します。

metadata.name

オーバークラウドコントロールプレーンの名前を **overcloud** に設定します。

metadata.namespace

director Operator namespace を **openstack** に設定します。

spec

コントロールプレーンの設定を指定します。このセクションで使用できる値の詳細は、**openstackcontrolplane** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstackcontrolplane
```

コントロールプレーンの仕様の設定が完了したら、ファイルを保存します。

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
```

OCP が OpenStackControlPlane リソースに関連するリソースを作成するまで待機します。

director Operator は、OpenStackControlPlane リソースの一部として、リモートシェルからアクセスして 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. 仮想マシンリソースを表示し、OpenShift Virtualization でのコントロールプレーンの仮想マシンの作成を確認します。

```
$ oc get virtualmachines
```

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 openstackclient -n openstack
```

7.7. テンプレートおよび環境ファイル用のディレクトリーの作成

ワークステーションにディレクトリーを作成し、カスタムテンプレートおよび環境ファイルを保管します。このファイルは、OpenShift Container Platform (OCP) の ConfigMap にアップロードします。

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. 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
```

7.8. コントローラーノード用のカスタム NIC HEAT テンプレート

以下の例は、コントローラー仮想マシンの NIC 設定が含まれる heat テンプレートです。

```
heat_template_version: rocky
description: >
  Software Config to drive os-net-config to configure multiple interfaces for the Controller role.
parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ControlPlaneSubnetCidr:
    default: ''
    description: >
      The subnet CIDR of the control plane network. (The parameter is
      automatically resolved from the ctlplane subnet's cidr attribute.)
    type: string
  ControlPlaneDefaultRoute:
    default: ''
    description: The default route of the control plane network. (The parameter
      is automatically resolved from the ctlplane subnet's gateway_ip attribute.)
    type: string
  ControlPlaneStaticRoutes:
    default: []
    description: >
      Routes for the ctlplane network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  ControlPlaneMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the network.
      (The parameter is automatically resolved from the ctlplane network's mtu attribute.)
    type: number
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageNetworkVlanID:
    default: 30
    description: Vlan ID for the storage network traffic.
    type: number
  StorageMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      Storage network.
    type: number
  StorageInterfaceRoutes:
    default: []
```

```
  description: >
    Routes for the storage network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
StorageMgmtIpSubnet:
  default: ''
  description: IP address/subnet on the storage_mgmt network
  type: string
StorageMgmtNetworkVlanID:
  default: 40
  description: Vlan ID for the storage_mgmt network traffic.
  type: number
StorageMgmtMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    StorageMgmt network.
  type: number
StorageMgmtInterfaceRoutes:
  default: []
  description: >
    Routes for the storage_mgmt network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
InternalApiIpSubnet:
  default: ''
  description: IP address/subnet on the internal_api network
  type: string
InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal_api network traffic.
  type: number
InternalApiMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    InternalApi network.
  type: number
InternalApiInterfaceRoutes:
  default: []
  description: >in your `custom_templates` directory.
    Routes for the internal_api network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
TenantNetworkVlanID:
  default: 50
```

```
    description: Vlan ID for the tenant network traffic.
    type: number
  TenantMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      Tenant network.
    type: number
  TenantInterfaceRoutes:
    default: []
    description: >
      Routes for the tenant network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  ExternalNetworkVlanID:
    default: 10
    description: Vlan ID for the external network traffic.
    type: number
  ExternalMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      External network.
    type: number
  ExternalInterfaceDefaultRoute:
    default: ''
    description: default route for the external network
    type: string
  ExternalInterfaceRoutes:
    default: []
    description: >
      Routes for the external network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  DnsServers: # Override this via parameter_defaults
    default: []
    description: >
      DNS servers to use for the Overcloud (2 max for some implementations).
      If not set the nameservers configured in the ctlplane subnet's
      dns_nameservers attribute will be used.
    type: comma_delimited_list
  DnsSearchDomains: # Override this via parameter_defaults
    default: []
    description: A list of DNS search domains to be added (in order) to resolv.conf.
    type: comma_delimited_list
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
```

```
properties:
  group: script
  config:
    str_replace:
      template:
        get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
      params:
        $network_config:
          network_config:

          ## NIC2 - Control Plane ##
          - type: interface
            name: nic2
            mtu:
              get_param: ControlPlaneMtu
            use_dhcp: false
            dns_servers:
              get_param: DnsServers
            domain:
              get_param: DnsSearchDomains
            addresses:
            - ip_netmask:
                list_join:
                - /
                - - get_param: ControlPlaneIp
                  - get_param: ControlPlaneSubnetCidr
            routes:
              list_concat_unique:
                - get_param: ControlPlaneStaticRoutes

          ## NIC3 - External ##
          - type: ovs_bridge
            name: bridge_name
            mtu:
              get_param: ExternalMtu
            dns_servers:
              get_param: DnsServers
            use_dhcp: false
            addresses:
            - ip_netmask:
                get_param: ExternalIpSubnet
            routes:
              list_concat_unique:
                - get_param: ExternalInterfaceRoutes
                - - default: true
                    next_hop:
                      get_param: ExternalInterfaceDefaultRoute
            members:
            - type: interface
              name: nic3
              mtu:
                get_param: ExternalMtu
              use_dhcp: false
              primary: true

          ## NIC4 - Internal API ##
```

```
- type: interface
  name: nic4
  mtu:
    get_param: InternalApiMtu
  use_dhcp: false
  addresses:
  - ip_netmask:
      get_param: InternalApiIpSubnet
  routes:
    list_concat_unique:
      - get_param: InternalApiInterfaceRoutes

## NIC5 - Storage ##
- type: interface
  name: nic5
  mtu:
    get_param: StorageMtu
  use_dhcp: false
  addresses:
  - ip_netmask:
      get_param: StorageIpSubnet
  routes:
    list_concat_unique:
      - get_param: StorageInterfaceRoutes

## NIC6 - StorageMgmt ##
- type: interface
  name: nic6
  mtu:
    get_param: StorageMgmtMtu
  use_dhcp: false
  addresses:
  - ip_netmask:
      get_param: StorageMgmtIpSubnet
  routes:
    list_concat_unique:
      - get_param: StorageMgmtInterfaceRoutes

## NIC7 - Tenant ##
- type: ovs_bridge
  name: br-isolated
  mtu:
    get_param: TenantMtu
  dns_servers:
    get_param: DnsServers
  use_dhcp: false
  addresses:
  - ip_netmask:
      get_param: TenantIpSubnet
  routes:
    list_concat_unique:
      - get_param: TenantInterfaceRoutes
  members:
  - type: interface
    name: nic7
    mtu:
```

```
                get_param: TenantMtu
              use_dhcp: false
              primary: true

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl
```

この設定により、ネットワークが以下のブリッジおよびインターフェースにマッピングされます。

ネットワーク	ブリッジ	interface
コントロールプレーン		**nic2**
外部	**br-ex**	**nic3**
InternalApi		**nic4**
ストレージ		**nic5**
StorageMgmt		**nic6**
Tenant	**br-isolated**	**nic7**

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_templates** ディレクトリーの **net-config-multi-nic-controller.yaml** にコピーします。

7.9. HCI コンピュートノード用のカスタム NIC HEAT テンプレート

以下は、HCI コンピュートノードの NIC 設定が含まれる heat テンプレート例です。

```
heat_template_version: rocky
description: >
  Software Config to drive os-net-config to configure VLANs for the Compute role.
parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ControlPlaneSubnetCidr:
    default: ''
    description: >
      The subnet CIDR of the control plane network. (The parameter is
      automatically resolved from the ctlplane subnet's cidr attribute.)
    type: string
  ControlPlaneDefaultRoute:
    default: ''
    description: The default route of the control plane network. (The parameter
      is automatically resolved from the ctlplane subnet's gateway_ip attribute.)
```

```
  type: string
ControlPlaneStaticRoutes:
  default: []
  description: >
    Routes for the ctlplane network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
ControlPlaneMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the network.
    (The parameter is automatically resolved from the ctlplane network's mtu attribute.)
  type: number
StorageIpSubnet:
  default: ''
  description: IP address/subnet on the storage network
  type: string
StorageNetworkVlanID:
  default: 30
  description: Vlan ID for the storage network traffic.
  type: number
StorageMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    Storage network.
  type: number
StorageInterfaceRoutes:
  default: []
  description: >
    Routes for the storage network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
StorageMgmtIpSubnet:
  default: ''
  description: IP address/subnet on the storage_mgmt network
  type: string
StorageMgmtNetworkVlanID:
  default: 40
  description: Vlan ID for the storage_mgmt network traffic.
  type: number
StorageMgmtMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    StorageMgmt network.
  type: number
StorageMgmtInterfaceRoutes:
  default: []
  description: >
    Routes for the storage_mgmt network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
```

```
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
InternalApiIpSubnet:
  default: ''
  description: IP address/subnet on the internal_api network
  type: string
InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal_api network traffic.
  type: number
InternalApiMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    InternalApi network.
  type: number
InternalApiInterfaceRoutes:
  default: []
  description: >
    Routes for the internal_api network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
TenantNetworkVlanID:
  default: 50
  description: Vlan ID for the tenant network traffic.
  type: number
TenantMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    Tenant network.
  type: number
TenantInterfaceRoutes:
  default: []
  description: >
    Routes for the tenant network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
ExternalMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    External network.
  type: number
DnsServers: # Override this via parameter_defaults
  default: []
  description: >
```

```
    DNS servers to use for the Overcloud (2 max for some implementations).
    If not set the nameservers configured in the ctlplane subnet's
    dns_nameservers attribute will be used.
  type: comma_delimited_list
DnsSearchDomains: # Override this via parameter_defaults
  default: []
  description: A list of DNS search domains to be added (in order) to resolv.conf.
  type: comma_delimited_list

resources:

 MinViableMtu:
  # This resource resolves the minimum viable MTU for interfaces, bonds and
  # bridges that carry multiple VLANs. Each VLAN may have different MTU. The
  # bridge, bond or interface must have an MTU to allow the VLAN with the
  # largest MTU.
  type: OS::Heat::Value
  properties:
    type: number
    value:
      yaql:
        expression: $.data.max()
        data:
          - {get_param: ControlPlaneMtu}
          - {get_param: StorageMtu}
          - {get_param: StorageMgmtMtu}
          - {get_param: InternalApiMtu}
          - {get_param: TenantMtu}
          - {get_param: ExternalMtu}

 OsNetConfigImpl:
  type: OS::Heat::SoftwareConfig
  properties:
    group: script
    config:
      str_replace:
        template:
          get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
        params:
          $network_config:
            network_config:
            - type: ovs_bridge
              name: br-isolated
              mtu:
                get_attr: [MinViableMtu, value]
              use_dhcp: false
              dns_servers:
                get_param: DnsServers
              domain:
                get_param: DnsSearchDomains
              addresses:
              - ip_netmask:
                  list_join:
                  - /
                  - - get_param: ControlPlaneIp
                    - get_param: ControlPlaneSubnetCidr
```

```
      routes:
        list_concat_unique:
          - get_param: ControlPlaneStaticRoutes
          - - default: true
              next_hop:
                get_param: ControlPlaneDefaultRoute
      members:
      - type: interface
        name: nic4
        mtu:
          get_attr: [MinViableMtu, value]
        # force the MAC address of the bridge to this interface
        primary: true
      - type: vlan
        mtu:
          get_param: StorageMtu
        vlan_id:
          get_param: StorageNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: StorageIpSubnet
        routes:
          list_concat_unique:
            - get_param: StorageInterfaceRoutes
      - type: vlan
        mtu:
          get_param: StorageMgmtMtu
        vlan_id:
          get_param: StorageMgmtNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: StorageMgmtIpSubnet
        routes:
          list_concat_unique:
            - get_param: StorageMgmtInterfaceRoutes
      - type: vlan
        mtu:
          get_param: InternalApiMtu
        vlan_id:
          get_param: InternalApiNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: InternalApiIpSubnet
        routes:
          list_concat_unique:
            - get_param: InternalApiInterfaceRoutes
      - type: vlan
        mtu:
          get_param: TenantMtu
        vlan_id:
          get_param: TenantNetworkVlanID
        addresses:
        - ip_netmask:
            get_param: TenantIpSubnet
        routes:
          list_concat_unique:
```

```
                - get_param: TenantInterfaceRoutes
            - type: ovs_bridge
              # This will default to br-ex, anything else requires specific
              # brige mapping entries for it to be used.
              name: bridge_name
              mtu:
                get_param: ExternalMtu
              use_dhcp: false
              members:
              - type: interface
                name: nic3
                mtu:
                  get_param: ExternalMtu
                use_dhcp: false
                primary: true
outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl
```

この設定により、ネットワークが以下のブリッジおよびインターフェースにマッピングされます。

ネットワーク	ブリッジ	interface
Control Plane、Storage、StorageMgmt、Internal API、Tenant	**br-isolated**	**nic4**
外部	**br-ex**	**nic3**

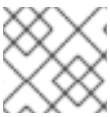
注記

この設定は、ベアメタルノードの NIC 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_templates** ディレクトリーの **net-config-two-nic-vlan-computehci.yaml** にコピーします。

7.10. DIRECTOR OPERATOR の COMPUTE HCI ロールを使用した ROLES_DATA.YAML ファイルの作成

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。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。

- **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ることを確認する。
- OpenStackControlPlane リソースを使用してコントロールプレーンを作成します。

手順

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

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

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 Controller ComputeHCI > roles_data.yaml
```

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

7.11. 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

カスタムテンプレートを tarball ファイルにアーカイブし、これらのテンプレートをオーバークラウドデプロイメントの一部として追加できるようにします。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- プロビジョニングしたノードに適用するカスタムテンプレートを作成します。

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 tarball にアーカイブします。

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

3. **tripleo-tarball-config** ConfigMap 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
```

検証

- ConfigMap を表示します。

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

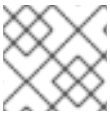
関連情報

- configmap の作成および使用
- Heat テンプレート

7.12. DIRECTOR OPERATOR での HCI ネットワークを設定するためのカスタム環境ファイル

以下の例は、ネットワークソフトウェア設定リソースをオーバークラウドの NIC テンプレートにマッピングする環境ファイルです。

```
resource_registry:
  OS::TripleO::Controller::Net::SoftwareConfig: net-config-multi-nic-controller.yaml
  OS::TripleO::ComputeHCI::Net::SoftwareConfig: net-config-two-nic-vlan-computehci.yaml
```



注記

parameter_defaults セクションに別のネットワーク設定を追加します。

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 **network-environment.yaml** にコピーします。

関連情報

- カスタムネットワーク環境ファイル
- ネットワーク環境パラメーター

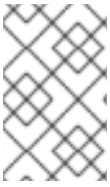
7.13. DIRECTOR OPERATOR でのハイパーコンバージドインフラストラクチャー (HCI) ストレージを設定するためのカスタム環境ファイル

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。

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

parameter_defaults:
  # needed for now because of the repo used to create tripleo-deploy image
  CephAnsibleRepo: "rhelosp-ceph-4-tools"
  CephAnsiblePlaybookVerbosity: 3
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: true
  CinderBackupBackend: ceph
  CinderEnableNfsBackend: false
  NovaEnableRbdBackend: true
  GlanceBackend: rbd
  CinderRbdPoolName: "volumes"
  NovaRbdPoolName: "vms"
  GlanceRbdPoolName: "images"
  CephPoolDefaultPgNum: 32
  CephPoolDefaultSize: 2
  CephAnsibleDisksConfig:
    devices:
      - '/dev/sdb'
      - '/dev/sdc'
      - '/dev/sdd'
    osd_scenario: lvm
    osd_objectstore: bluestore
  CephAnsibleExtraConfig:
    is_hci: true
  CephConfigOverrides:
    rgw_swift_enforce_content_length: true
    rgw_swift_versioning_enabled: true
```

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。

関連情報

- ハイパーコンバージドノード上におけるリソース分離の設定

7.14. 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

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 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

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- 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を作成します。

手順

1. **heat-env-config** ConfigMap を作成し、環境ファイルが含まれるディレクトリーをデータとして使用します。

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

検証

- ConfigMap を表示します。

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

関連情報

- configmap の作成および使用
- 環境ファイル

7.15. OPENSTACKBAREMETALSET を使用した HCI コンピュートノードの作成

コンピュートノードは Red Hat OpenStack Platform 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コンピュートノードが少なくとも 1 台必要で、デプロイメント後にコンピュートノードの数をスケーリングできます。

OpenStackBaremetalSet カスタムリソースは、OpenShift Container Platform が管理するベアメタルマシンからコンピュートノードを作成します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackNet リソースを使用して、コントロールプレーンネットワークおよび追加の分離ネットワークを作成します。

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。コンピュートノードのリソース仕様を含めます。たとえば、コンピュートノード 1 つの仕様は以下のようになります。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: computehci
  namespace: openstack
spec:
  replicas: 3
  baseImageUrl: http://host/images/rhel-image-8.4.x86_64.qcow2
  deploymentSSHSecret: osp-controlplane-ssh-keys
  ctlplaneInterface: enp8s0
  networks:
    - ctlplane
    - internalapi
    - tenant
    - storage
    - storagemgmt
  roleName: ComputeHCI
  passwordSecret: userpassword
```

リソース仕様に以下の値を設定します。

metadata.name

コンピュートノードのベアメタルセットの名前を **overcloud** に設定します。

metadata.namespace

director Operator namespace を **openstack** に設定します。

spec

コンピュートノードの設定を定義します。このセクションで使用できる値の詳細は、**openstackbaremetalset** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstackbaremetalset
```

コンピュートノードの仕様の設定が完了したら、ファイルを保存します。

2. コンピュートノードを作成します。

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

検証

1. コンピュート HCI ノードのリソースを表示します。

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

2. OpenShift が管理するベアメタルマシンを表示して、コンピュート HCI ノードの作成を確認します。

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

7.16. OPENSTACKPLAYBOOKGENERATOR を使用したオーバークラウドの設定用の 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) ソフトウェアを設定する Ansible Playbook のセットを作成する必要があります。これらの Playbook を OpenStackPlaybookGenerator リソースで作成し、RHOSP director の **config-download** 機能を使用して heat 設定を Playbook に変換します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- リモート Git リポジトリーの認証情報が含まれる **git-secret** シークレットを設定します。
- カスタム heat テンプレートが含まれる **tripleo-tarball-config** ConfigMap を設定します。
- カスタム環境ファイルが含まれる **heat-env-config** ConfigMap を設定します。

手順

1. ワークステーションに **openstack-playbook-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。たとえば、Playbook を生成する仕様は以下のようになります。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackPlaybookGenerator
metadata:
  name: default
  namespace: openstack
spec:
  imageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  gitSecret: git-secret
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
```

リソース仕様に以下の値を設定します。

metadata.name

コンピュートノードのベアメタルセットの名前を **default** に設定します。

metadata.namespace

director Operator namespace を **openstack** に設定します。

spec.gitSecret

Git 認証認証情報 (**git-secret**) を含む ConfigMap に設定します。

spec.tarballConfigMap

カスタムの heat テンプレートを使用して tarball が含まれる ConfigMap (**tripleo-tarball-config**) に設定します。

spec.heatEnvConfigMap

カスタム環境ファイルが含まれる ConfigMap (**heat-env-config**) に設定します。

spec セクションで使用できる値の詳細は、**openstackplaybookgenerator** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstackplaybookgenerator
```

Ansible Playbook ジェネレーター仕様の設定が完了したら、ファイルを保存します。

2. Ansible Playbook ジェネレーターを作成します。

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

検証

- Playbook ジェネレーターのリソースを表示します。

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

7.17. 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 がノードにオーバークラウドソフトウェアを設定する前に、全ノードのオペレーティングシステムを Red Hat カスタマーポータルまたは Red Hat Satellite Server のいずれかに登録して、ノードのリポジトリーを有効にする必要があります。ノードを登録するには、OpentackPlaybookGenerator リソースで作成されるインベントリーファイルを使用して、**redhat_subscription** Ansible モジュールを使用します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackControlPlane リソースを使用してコントロールプレーンを作成します。
- OpenStackBareMetalSet リソースを使用して、ベアメタルのコンピュートノードを作成します。
- OpentackPlaybookGenerator を使用して、オーバークラウドの Ansible Playbook 設定を作成します。

手順

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

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

2. **cloud-admin** ホームディレクトリーに移動します。

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

3. オプション: オーバークラウドの Ansible Playbook の diff をチェックします。

```
$ ./tripleo-deploy.sh -d
```

4. レンダリングされた Ansible Playbook の最新バージョンを受け入れて **latest** としてタグ付けします。

```
$ ./tripleo-deploy.sh -a
```

最新バージョンの Playbook には、オーバークラウド設定用のインベントリーファイルが含まれます。このインベントリーファイルは、OpenStackClient Pod の **/home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml** にあります。

5. 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-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      - ansible-2.9-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-16.1-for-rhel-8-x86_64-rpms
      - rhceph-4-mon-for-rhel-8-x86_64-rpms
      - fast-datapath-for-rhel-8-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 8.4
        pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
    - name: Disable all repos
      command: "subscription-manager repos --disable *"
    - name: Enable Controller node repos
      command: "subscription-manager repos --enable {{ item }}"
      with_items: "{{ repos }}"
```

このプレイには、以下の 3 つのタスクが含まれます。

- ノードを登録する。
- 自動的に有効化されるリポジトリーをすべて無効にする。
- コントローラーノードに関連するリポジトリーだけを有効にする。リポジトリーは **repos** 変数でリストされます。

6. 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-playpook -i /home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml ./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

7.18. 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コンピュート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 でオーバークラウドを設定できません。director Operator は、OpenStackControlPlane リソースの作成時に、リモートシェル経由でアクセスする OpenStackClient Pod を作成し、**tripleo-deploy.sh** スクリプトを実行してオーバークラウドを設定します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackControlPlane リソースを使用してコントロールプレーンを作成します。
- OpenStackBareMetalSet リソースを使用して、ベアメタルのコンピュートノードを作成します。
- OpentackPlaybookGenerator を使用して、オーバークラウドの Ansible Playbook 設定を作成します。

手順

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

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

2. **cloud-admin** ホームディレクトリーに移動します。

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

3. オプション: オーバークラウドの Ansible Playbook の diff をチェックします。

```
$ ./tripleo-deploy.sh -d
```

4. レンダリングされた Ansible Playbook の最新バージョンを受け入れて **latest** としてタグ付けします。

```
$ ./tripleo-deploy.sh -a
```

5. オーバークラウドノードに対して Ansible Playbook を適用します。

```
$ ./tripleo-deploy.sh -p
```

注記

openstackclient Pod 内の **~/ansible.log** で Ansible 設定のログを表示できます。

第8章 DIRECTOR OPERATOR のデプロイメントシナリオ: 外部の CEPH STORAGE を使用するオーバークラウド

director Operator 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

- 外部の Red Hat Ceph Storage クラスター

8.1. 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のベースオペレーティングシステムのイメージを保存するには、OpenShift Container Platform (OCP) クラスターでデータボリュームを作成する必要があります。

前提条件

- Red Hat Enterprise Linux 8 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。このイメージは、Red Hat カスタマーポータルの製品のダウンロードセクションからダウンロードできます。
- **virtctl** クライアントツールをワークステーションにインストールします。以下のコマンドを実行して、このツールを Red Hat Enterprise Linux ワークステーションにインストールできます。

```
$ sudo subscription-manager repos --enable=cnv-2.6-for-rhel-8-x86_64-rpms
$ sudo dnf install -y kubevirt-virtctl
```

- **virt-customize** クライアントツールをワークステーションにインストールします。このツールは、以下のコマンドを使用して Red Hat Enterprise Linux ワークステーションにインストールできます。

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

手順

1. access.redhat.com からダウンロードしたデフォルトの QCOW2 イメージでは、biosdev の予測可能なネットワークインターフェース名は使用されません。biosdev の予測可能なネットワークインターフェース名を使用するように、**virt-customize** でイメージを変更します。

```
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(kernelopts=.*\)net.ifnames=0 \(.*\)/\1\2/" /boot/grub2/grubenv'
$ sudo virt-customize -a <local path to image> --run-command 'sed -i -e "s/^\
(GRUB_CMDLINE_LINUX=.*\)net.ifnames=0 \(.*\)/\1\2/" /etc/default/grub'
```

2. **virtctl** でイメージを OpenShift Virtualization にアップロードします。

```
$ virtctl image-upload dv openstack-base-img -n openstack --size=50Gi --image-path=<local
path to image> --storage-class <storage class> --insecure
```

--storage-class オプションの場合は、クラスターからストレージクラスを選択します。以下のコマンドを使用してストレージクラスの一覧を表示します。

```
$ oc get storageclass
```

3. OpenStackControlPlane リソースおよび個別の OpenStackVmSet リソースの作成時に、**baseImageVolumeName** パラメーターをデータボリューム名に設定します。

```
...
spec:
  ...
  baseImageVolumeName: openstack-base-img
...
```

関連情報

- virtctl ツールを使用したローカルディスクイメージのアップロード

8.2. リモート GIT リポジトリーの認証情報の追加

director Operator 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-secret** という名前の OpenShift Secret リソースとして Git リポジトリーの詳細を指定する必要があります。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- director オペレーターのリモート Git リポジトリーを作成して、オーバークラウド向けに生成された設定を保存します。
- SSH キーペアを準備します。公開鍵を Git リポジトリーにアップロードし、秘密鍵を利用できる状態にして、**git-secret** Secret リソースに追加します。

手順

- Secret リソースを作成します。

```
$ oc create secret generic git-secret -n openstack --from-file=git_ssh_identity=
<path_to_private_SSH_key> --from-literal=git_url=<git_server_URL>
```

git-secret Secret リソースには、2 つのキーと値のペアが含まれます。

git_ssh_identity

Git リポジトリーにアクセスするための秘密鍵**--from-file** オプションは、SSH 秘密鍵ファイルの内容を保存します。

git_url

設定を保存する git リポジトリーの SSH URL。**--from-literal** オプションは、このキーに入力した URL を保存します。

検証

- Secret リソースを表示します。

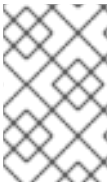
```
$ oc get secret/git-secret -n openstack
```

関連情報

- Pod への機密データの指定

8.3. 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。



注記

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。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。

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
```

注記

-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: userpassword
  namespace: openstack
data:
  NodeRootPassword: "cEBzc3cwcmQh"
```

NodeRootPassword パラメーターを base64 でエンコードされたパスワードに設定します。

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 への機密データの指定

8.4. OPENSTACKNET を使用したオーバークラウドのコントロールプレーンネットワークの作成

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。OpenStackNet リソースには、IP アドレスの割り当てに加えて、OpenShift Virtualization が仮想マシンをネットワークにアタッチするために使用するネットワーク設定ポリシーを定義する情報が含まれます。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。

手順

1. ワークステーションに **ctlplane-network.yaml** という名前のファイルを作成します。**ctlplane** という名前のコントロールプレーンネットワークのリソース仕様を含めます。たとえば、各ワーカーノードの **enp6s0** イーサネットデバイスに接続された Linux ブリッジを使用するコントロールプレーンの仕様は以下のようになります。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: ctlplane
spec:
  cidr: 192.168.25.0/24
  allocationStart: 192.168.25.100
  allocationEnd: 192.168.25.250
  gateway: 192.168.25.1
  attachConfiguration:
    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
```

リソース仕様に以下の値を設定します。

metadata.name

コントロールプレーンネットワークの名 (**ctlplane**) を設定します。

spec

コントロールプレーンネットワークのネットワーク設定を指定します。このセクションで使用できる値の詳細は、**openstacknet** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstacknet
```

ネットワーク仕様の設定が完了したら、ファイルを保存します。

2. コントロールプレーンネットワークを作成します。

```
$ oc create -f ctlplane-network.yaml -n openstack
```

検証

- コントロールプレーンネットワークのリソースを表示します。

```
$ oc get openstacknet/ctlplane
```

8.5. OPENSTACKNET を使用したネットワーク分離用の VLAN ネットワークの作成

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には、追加のネットワークを作成する必要があります。このネットワーク分離を実現するには、コンポーザブルネットワークを個別の VLAN ネットワークに配置できます。OpenStackNet リソースには、IP アドレスの割り当てに加えて、OpenShift Virtualization が仮想マシンを VLAN ネットワークにアタッチするために使用するネットワーク設定ポリシーを定義する情報が含まれます。

デフォルトの Red Hat OpenStack Platform ネットワークを使用するには、ネットワークごとに OpenStackNet リソースを作成する必要があります。

表8.1 デフォルトの Red Hat OpenStack Platform ネットワーク

ネットワーク	VLAN	CIDR	割り当て
外部	10	10.0.0.0/24	10.0.0.4 - 10.0.0.250
InternalApi	20	172.16.2.0/24	172.16.2.4 - 172.16.2.250
ストレージ	30	172.16.1.0/24	172.16.1.4 - 172.16.1.250
StorageMgmt	40	172.16.3.0/24	172.16.3.4 - 172.16.3.250
Tenant	50	172.16.0.0/24	172.16.0.4 - 172.16.0.250

重要

各ネットワークごとに異なるネットワークの詳細を使用するには、カスタム **network_data.yaml** ファイルを作成する必要があります。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。

手順

1. 新規ネットワークのファイルを作成します。たとえば、内部 API ネットワークの場合は、ワークステーションに **internalapi-network.yaml** という名前のファイルを作成します。VLAN ネットワークのリソース仕様を含めます。たとえば、以下は、各ワーカーノードの **enp6s0** イーサネットデバイスに接続された Linux ブリッジ経由で VLAN タグ付けされたトラフィックを管理する内部 API ネットワークの仕様です。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackNet
metadata:
  name: internalapi
spec:
  cidr: 172.16.2.0/24
  vlan: 20
  allocationStart: 172.16.2.4
  allocationEnd: 172.16.2.250
  attachConfiguration:
    nodeNetworkConfigurationPolicy:
      nodeSelector:
        node-role.kubernetes.io/worker: ""
      desiredState:
        interfaces:
        - bridge:
            options:
              stp:
                enabled: false
            port:
            - name: enp6s0
          description: Linux bridge with enp7s0 as a port
          name: br-osp
          state: up
          type: linux-bridge
```

linux-bridge でネットワーク分離に VLAN を使用する場合に、以下のような処理が行われます。

- director Operator は、リソースに指定されたブリッジインターフェースの Node Network Configuration Policy を作成します。このポリシーは、**nmstate** を使用してワーカーノードにブリッジを設定します。
- director Operator は、Multus CNI プラグイン設定を定義するネットワークごとにネットワーク接続定義を作成します。ネットワーク接続定義で VLAN ID を指定する場合には、Multus CNI プラグインはブリッジで **vlan-filtering** を有効にします。

- director Operator は、仮想マシン上の各ネットワーク専用のインターフェースを割り当てます。これは、**OSVMSet** のネットワークテンプレートがマルチ NIC ネットワークテンプレートであることを意味します。

リソース仕様に以下の値を設定します。

metadata.name

コントロールプレーンネットワークの名 (**ctlplane**) を設定します。

spec

コントロールプレーンネットワークのネットワーク設定を指定します。このセクションで使用できる値の詳細は、**openstacknet** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstacknet
```

ネットワーク仕様の設定が完了したら、ファイルを保存します。

2. 内部 API ネットワークを作成します。

```
$ oc create -f internalapi-network.yaml -n openstack
```

検証

- 内部 API ネットワークのリソースを表示します。

```
$ oc get openstacknet/internalapi -n openstack
```

8.6. OPENSTACKCONTROLPLANE でのコントロールプレーンの作成

オーバークラウドのコントロールプレーンには、オーバークラウドの機能を管理するメインの Red Hat OpenStack Platform サービスが含まれます。コントロールプレーンは、通常 3 つのコントローラーノードで構成されており、他のコントロールプレーンベースのコンポーザブルロールにスケーリングできます。コンポーザブルロールを使用する場合には、各サービスは 3 つの追加の専用ノードで実行される必要があり、Pacemaker のクォーラムを維持するために、コントロールプレーン内のノードの合計数を追加する必要があります。

OpenStackControlPlane カスタムリソースは、コントロールプレーンベースのノードを OpenShift Virtualization 内の仮想マシンとして作成します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackNet リソースを使用して、コントロールプレーンネットワークおよび追加の分離ネットワークを作成します。

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。たとえば、3 つのコントローラーノードで構成されるコントロールプレーンの仕様は以下のとおりです。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackControlPlane
metadata:
  name: overcloud
  namespace: openstack
spec:
  openStackClientImageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  openStackClientNetworks:
        - ctlplane
        - external
        - internalapi
  openStackClientStorageClass: host-nfs-storageclass
  passwordSecret: userpassword
  gitSecret: git-secret
  virtualMachineRoles:
    controller:
      roleName: Controller
      roleCount: 3
      networks:
        - ctlplane
        - internalapi
        - external
        - tenant
        - storage
        - storagemgmt
      cores: 6
      memory: 12
      diskSize: 50
      baseImageVolumeName: openstack-base-img
      storageClass: host-nfs-storageclass
```

リソース仕様に以下の値を設定します。

metadata.name

オーバークラウドコントロールプレーンの名前を **overcloud** に設定します。

metadata.namespace

director Operator namespace を **openstack** に設定します。

spec

コントロールプレーンの設定を指定します。このセクションで使用できる値の詳細は、**openstackcontrolplane** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstackcontrolplane
```

コントロールプレーンの仕様の設定が完了したら、ファイルを保存します。

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
```

OCP が OpenStackControlPlane リソースに関連するリソースを作成するまで待機します。

director Operator は、OpenStackControlPlane リソースの一部として、リモートシェルからアクセスして 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. 仮想マシンリソースを表示し、OpenShift Virtualization でのコントロールプレーンの仮想マシンの作成を確認します。

```
$ oc get virtualmachines
```

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 openstackclient -n openstack
```

8.7. テンプレートおよび環境ファイル用のディレクトリーの作成

ワークステーションにディレクトリーを作成し、カスタムテンプレートおよび環境ファイルを保管します。このファイルは、OpenShift Container Platform (OCP) の ConfigMap にアップロードします。

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. 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
```

8.8. コントローラーノード用のカスタム NIC HEAT テンプレート

以下の例は、コントローラー仮想マシンの NIC 設定が含まれる heat テンプレートです。

```
heat_template_version: rocky
description: >
  Software Config to drive os-net-config to configure multiple interfaces for the Controller role.
parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
```

```
  ControlPlaneSubnetCidr:
    default: ''
    description: >
      The subnet CIDR of the control plane network. (The parameter is
      automatically resolved from the ctlplane subnet's cidr attribute.)
    type: string
  ControlPlaneDefaultRoute:
    default: ''
    description: The default route of the control plane network. (The parameter
      is automatically resolved from the ctlplane subnet's gateway_ip attribute.)
    type: string
  ControlPlaneStaticRoutes:
    default: []
    description: >
      Routes for the ctlplane network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  ControlPlaneMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the network.
      (The parameter is automatically resolved from the ctlplane network's mtu attribute.)
    type: number
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageNetworkVlanID:
    default: 30
    description: Vlan ID for the storage network traffic.
    type: number
  StorageMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      Storage network.
    type: number
  StorageInterfaceRoutes:
    default: []
    description: >
      Routes for the storage network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage_mgmt network
    type: string
  StorageMgmtNetworkVlanID:
    default: 40
    description: Vlan ID for the storage_mgmt network traffic.
    type: number
  StorageMgmtMtu:
```

```
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    StorageMgmt network.
  type: number
StorageMgmtInterfaceRoutes:
  default: []
  description: >
    Routes for the storage_mgmt network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
InternalApiIpSubnet:
  default: ''
  description: IP address/subnet on the internal_api network
  type: string
InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal_api network traffic.
  type: number
InternalApiMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    InternalApi network.
  type: number
InternalApiInterfaceRoutes:
  default: []
  description: >in your `custom_templates` directory.
    Routes for the internal_api network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
TenantNetworkVlanID:
  default: 50
  description: Vlan ID for the tenant network traffic.
  type: number
TenantMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    Tenant network.
  type: number
TenantInterfaceRoutes:
  default: []
  description: >
    Routes for the tenant network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
```

```
    type: json
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  ExternalNetworkVlanID:
    default: 10
    description: Vlan ID for the external network traffic.
    type: number
  ExternalMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      External network.
    type: number
  ExternalInterfaceDefaultRoute:
    default: ''
    description: default route for the external network
    type: string
  ExternalInterfaceRoutes:
    default: []
    description: >
      Routes for the external network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  DnsServers: # Override this via parameter_defaults
    default: []
    description: >
      DNS servers to use for the Overcloud (2 max for some implementations).
      If not set the nameservers configured in the ctlplane subnet's
      dns_nameservers attribute will be used.
    type: comma_delimited_list
  DnsSearchDomains: # Override this via parameter_defaults
    default: []
    description: A list of DNS search domains to be added (in order) to resolv.conf.
    type: comma_delimited_list
resources:
  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template:
            get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
          params:
            $network_config:
              network_config:

              ## NIC2 - Control Plane ##
              - type: interface
                name: nic2
                mtu:
                  get_param: ControlPlaneMtu
```

```
      use_dhcp: false
      dns_servers:
        get_param: DnsServers
      domain:
        get_param: DnsSearchDomains
      addresses:
      - ip_netmask:
          list_join:
          - /
          - - get_param: ControlPlaneIp
            - get_param: ControlPlaneSubnetCidr
      routes:
        list_concat_unique:
          - get_param: ControlPlaneStaticRoutes

    ## NIC3 - External ##
    - type: ovs_bridge
      name: bridge_name
      mtu:
        get_param: ExternalMtu
      dns_servers:
        get_param: DnsServers
      use_dhcp: false
      addresses:
      - ip_netmask:
          get_param: ExternalIpSubnet
      routes:
        list_concat_unique:
          - get_param: ExternalInterfaceRoutes
          - - default: true
              next_hop:
                get_param: ExternalInterfaceDefaultRoute
      members:
      - type: interface
        name: nic3
        mtu:
          get_param: ExternalMtu
        use_dhcp: false
        primary: true

    ## NIC4 - Internal API ##
    - type: interface
      name: nic4
      mtu:
        get_param: InternalApiMtu
      use_dhcp: false
      addresses:
      - ip_netmask:
          get_param: InternalApiIpSubnet
      routes:
        list_concat_unique:
          - get_param: InternalApiInterfaceRoutes

    ## NIC5 - Storage ##
    - type: interface
      name: nic5
```

```
            mtu:
              get_param: StorageMtu
            use_dhcp: false
            addresses:
            - ip_netmask:
                get_param: StorageIpSubnet
            routes:
              list_concat_unique:
                - get_param: StorageInterfaceRoutes

          ## NIC6 - StorageMgmt ##
          - type: interface
            name: nic6
            mtu:
              get_param: StorageMgmtMtu
            use_dhcp: false
            addresses:
            - ip_netmask:
                get_param: StorageMgmtIpSubnet
            routes:
              list_concat_unique:
                - get_param: StorageMgmtInterfaceRoutes

          ## NIC7 - Tenant ##
          - type: ovs_bridge
            name: br-isolated
            mtu:
              get_param: TenantMtu
            dns_servers:
              get_param: DnsServers
            use_dhcp: false
            addresses:
            - ip_netmask:
                get_param: TenantIpSubnet
            routes:
              list_concat_unique:
                - get_param: TenantInterfaceRoutes
            members:
            - type: interface
              name: nic7
              mtu:
                get_param: TenantMtu
              use_dhcp: false
              primary: true

outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl
```

この設定により、ネットワークが以下のブリッジおよびインターフェースにマッピングされます。

ネットワーク	ブリッジ	interface
コントロールプレーン		**nic2**
外部	**br-ex**	**nic3**
InternalApi		**nic4**
ストレージ		**nic5**
StorageMgmt		**nic6**
Tenant	**br-isolated**	**nic7**

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_templates** ディレクトリーの **net-config-multi-nic-controller.yaml** にコピーします。

8.9. コンピュートノード用のカスタム NIC HEAT テンプレート

以下は、コンピュートノードの NIC 設定が含まれる heat テンプレート例です。

```
heat_template_version: rocky
description: >
  Software Config to drive os-net-config to configure VLANs for the Compute role.
parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ControlPlaneSubnetCidr:
    default: ''
    description: >
      The subnet CIDR of the control plane network. (The parameter is
      automatically resolved from the ctlplane subnet's cidr attribute.)
    type: string
  ControlPlaneDefaultRoute:
    default: ''
    description: The default route of the control plane network. (The parameter
      is automatically resolved from the ctlplane subnet's gateway_ip attribute.)
    type: string
  ControlPlaneStaticRoutes:
    default: []
    description: >
      Routes for the ctlplane network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  ControlPlaneMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
```

```
    guaranteed to pass through the data path of the segments in the network.
    (The parameter is automatically resolved from the ctlplane network's mtu attribute.)
  type: number
StorageIpSubnet:
  default: ''
  description: IP address/subnet on the storage network
  type: string
StorageNetworkVlanID:
  default: 30
  description: Vlan ID for the storage network traffic.
  type: number
StorageMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    Storage network.
  type: number
StorageInterfaceRoutes:
  default: []
  description: >
    Routes for the storage network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
InternalApiIpSubnet:
  default: ''
  description: IP address/subnet on the internal_api network
  type: string
InternalApiNetworkVlanID:
  default: 20
  description: Vlan ID for the internal_api network traffic.
  type: number
InternalApiMtu:
  default: 1500
  description: The maximum transmission unit (MTU) size(in bytes) that is
    guaranteed to pass through the data path of the segments in the
    InternalApi network.
  type: number
InternalApiInterfaceRoutes:
  default: []
  description: >
    Routes for the internal_api network traffic.
    JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
    Unless the default is changed, the parameter is automatically resolved
    from the subnet host_routes attribute.
  type: json
TenantIpSubnet:
  default: ''
  description: IP address/subnet on the tenant network
  type: string
TenantNetworkVlanID:
  default: 50
  description: Vlan ID for the tenant network traffic.
  type: number
TenantMtu:
```

```
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      Tenant network.
    type: number
  TenantInterfaceRoutes:
    default: []
    description: >
      Routes for the tenant network traffic.
      JSON route e.g. [{'destination':'10.0.0.0/16', 'nexthop':'10.0.0.1'}]
      Unless the default is changed, the parameter is automatically resolved
      from the subnet host_routes attribute.
    type: json
  ExternalMtu:
    default: 1500
    description: The maximum transmission unit (MTU) size(in bytes) that is
      guaranteed to pass through the data path of the segments in the
      External network.
    type: number
  DnsServers: # Override this via parameter_defaults
    default: []
    description: >
      DNS servers to use for the Overcloud (2 max for some implementations).
      If not set the nameservers configured in the ctlplane subnet's
      dns_nameservers attribute will be used.
    type: comma_delimited_list
  DnsSearchDomains: # Override this via parameter_defaults
    default: []
    description: A list of DNS search domains to be added (in order) to resolv.conf.
    type: comma_delimited_list

resources:

  MinViableMtu:
    # This resource resolves the minimum viable MTU for interfaces, bonds and
    # bridges that carry multiple VLANs. Each VLAN may have different MTU. The
    # bridge, bond or interface must have an MTU to allow the VLAN with the
    # largest MTU.
    type: OS::Heat::Value
    properties:
      type: number
      value:
        yaql:
          expression: $.data.max()
          data:
            - {get_param: ControlPlaneMtu}
            - {get_param: StorageMtu}
            - {get_param: InternalApiMtu}
            - {get_param: TenantMtu}
            - {get_param: ExternalMtu}

  OsNetConfigImpl:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
```

```
str_replace:
  template:
    get_file: /usr/share/openstack-tripleo-heat-templates/network/scripts/run-os-net-config.sh
  params:
    $network_config:
      network_config:
      - type: ovs_bridge
        name: br-isolated
        mtu:
          get_attr: [MinViableMtu, value]
        use_dhcp: false
        dns_servers:
          get_param: DnsServers
        domain:
          get_param: DnsSearchDomains
        addresses:
        - ip_netmask:
            list_join:
            - /
            - - get_param: ControlPlaneIp
              - get_param: ControlPlaneSubnetCidr
        routes:
          list_concat_unique:
            - get_param: ControlPlaneStaticRoutes
            - - default: true
                next_hop:
                  get_param: ControlPlaneDefaultRoute
        members:
        - type: interface
          name: nic4
          mtu:
            get_attr: [MinViableMtu, value]
          # force the MAC address of the bridge to this interface
          primary: true
        - type: vlan
          mtu:
            get_param: StorageMtu
          vlan_id:
            get_param: StorageNetworkVlanID
          addresses:
          - ip_netmask:
              get_param: StorageIpSubnet
          routes:
            list_concat_unique:
              - get_param: StorageInterfaceRoutes
        - type: vlan
          mtu:
            get_param: InternalApiMtu
          vlan_id:
            get_param: InternalApiNetworkVlanID
          addresses:
          - ip_netmask:
              get_param: InternalApiIpSubnet
          routes:
            list_concat_unique:
              - get_param: InternalApiInterfaceRoutes
```

```
            - type: vlan
              mtu:
                get_param: TenantMtu
              vlan_id:
                get_param: TenantNetworkVlanID
              addresses:
              - ip_netmask:
                  get_param: TenantIpSubnet
              routes:
                list_concat_unique:
                  - get_param: TenantInterfaceRoutes
          - type: ovs_bridge
            # This will default to br-ex, anything else requires specific
            # brige mapping entries for it to be used.
            name: bridge_name
            mtu:
              get_param: ExternalMtu
            use_dhcp: false
            members:
            - type: interface
              name: nic3
              mtu:
                get_param: ExternalMtu
              use_dhcp: false
              primary: true
outputs:
  OS::stack_id:
    description: The OsNetConfigImpl resource.
    value:
      get_resource: OsNetConfigImpl
```

この設定により、ネットワークが以下のブリッジおよびインターフェースにマッピングされます。

ネットワーク	ブリッジ	interface
コントロールプレーン、ストレージ、内部 API、テナント	**br-isolated**	**nic4**
外部	**br-ex**	**nic3**

注記

この設定は、ベアメタルノードの NIC 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_templates** ディレクトリーの **net-config-two-nic-vlan-compute.yaml** にコピーします。

8.10. 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

カスタムテンプレートを tarball ファイルにアーカイブし、これらのテンプレートをオーバークラウドデプロイメントの一部として追加できるようにします。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- プロビジョニングしたノードに適用するカスタムテンプレートを作成します。

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 tarball にアーカイブします。

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

3. **tripleo-tarball-config** ConfigMap 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
```

検証

- ConfigMap を表示します。

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

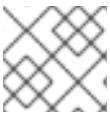
関連情報

- configmap の作成および使用
- Heat テンプレート

8.11. DIRECTOR OPERATOR でネットワークを設定するためのカスタム環境ファイル

以下の例は、ネットワークソフトウェア設定リソースをオーバークラウドの NIC テンプレートにマッピングする環境ファイルです。

```
resource_registry:
  OS::TripleO::Controller::Net::SoftwareConfig: net-config-multi-nic-controller.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: net-config-two-nic-vlan-compute.yaml
```



注記

parameter_defaults セクションに別のネットワーク設定を追加します。

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 **network-environment.yaml** にコピーします。

関連情報

- カスタムネットワーク環境ファイル
- ネットワーク環境パラメーター

8.12. 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

以下の例は、外部の Ceph Storage クラスターと統合するための設定が含まれる環境ファイルです。

```
resource_registry:
  OS::TripleO::Services::CephExternal: deployment/ceph-ansible/ceph-external.yaml

parameter_defaults:
  # needed for now because of the repo used to create tripleo-deploy image
  CephAnsibleRepo: "rhelosp-ceph-4-tools"
  CephAnsiblePlaybookVerbosity: 3

  NovaEnableRbdBackend: true
  GlanceBackend: rbd
  CinderEnableRbdBackend: true
  CinderBackupBackend: ceph
  CinderEnableIscsiBackend: false

  # Change the following values for your external ceph cluster
  NovaRbdPoolName: vms
  CinderRbdPoolName: volumes
  CinderBackupRbdPoolName: backups
  GlanceRbdPoolName: images
  CephClientUserName: openstack
  CephClientKey: AQDLOh1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==
  CephClusterFSID: 4b5c8c0a-ff60-454b-a1b4-9747aa737d19
  CephExternalMonHost: 172.16.1.7, 172.16.1.8, 172.16.1.9
```

この設定には、ノード上で **CephExternal** サービスおよび **CephClient** サービスを有効にし、異なる RHOSP サービスのプールを設定するためのパラメーターが含まれており、以下のパラメーターを使用して外部 Ceph Storage クラスターを統合します。

CephClientKey
外部の Ceph Storage クラスターの Ceph クライアントキー。

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

CephExternalMonHost
外部 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。

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。

関連情報

- 既存の Ceph Storage クラスターとの統合

8.13. 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

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 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

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- 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を作成します。

手順

1. **heat-env-config** ConfigMap を作成し、環境ファイルが含まれるディレクトリーをデータとして使用します。

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

検証

- ConfigMap を表示します。

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

関連情報

- configmap の作成および使用
- 環境ファイル

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

コンピュートノードは Red Hat OpenStack Platform 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コンピュートノードが少なくとも 1 台必要で、デプロイメント後にコンピュートノードの数をスケーリングできます。

OpenStackBaremetalSet カスタムリソースは、OpenShift Container Platform が管理するベアメタルマシンからコンピュートノードを作成します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。

- OpenStackNet リソースを使用して、コントロールプレーンネットワークおよび追加の分離ネットワークを作成します。

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。コンピュートノードのリソース仕様を含めます。たとえば、コンピュートノード 1 つの仕様は以下のようになります。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackBaremetalSet
metadata:
  name: compute
  namespace: openstack
spec:
  count: 1
  baseImageUrl: http://host/images/rhel-image-8.4.x86_64.qcow2
  deploymentSSHSecret: osp-controlplane-ssh-keys
  ctlplaneInterface: enp2s0
  networks:
    - ctlplane
    - internalapi
    - tenant
    - storage
  roleName: Compute
  passwordSecret: userpassword
```

リソース仕様に以下の値を設定します。

metadata.name

コンピュートノードのベアメタルセットの名前を **overcloud** に設定します。

metadata.namespace

director Operator namespace を **openstack** に設定します。

spec

コンピュートノードの設定を定義します。このセクションで使用できる値の詳細は、**openstackbaremetalset** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstackbaremetalset
```

コンピュートノードの仕様の設定が完了したら、ファイルを保存します。

2. コンピュートノードを作成します。

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

検証

1. コンピュートノードのリソースを表示します。

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

2. OpenShift が管理するベアメタルマシンを表示し、コンピュートノードの作成を確認します。

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

8.15. OPENSTACKPLAYBOOKGENERATOR を使用したオーバークラウドの設定用の 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) ソフトウェアを設定する Ansible Playbook のセットを作成する必要があります。これらの Playbook を OpenStackPlaybookGenerator リソースで作成し、RHOSP director の **config-download** 機能を使用して heat 設定を Playbook に変換します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- リモート Git リポジトリーの認証情報が含まれる **git-secret** シークレットを設定します。
- カスタム heat テンプレートが含まれる **tripleo-tarball-config** ConfigMap を設定します。
- カスタム環境ファイルが含まれる **heat-env-config** ConfigMap を設定します。

手順

1. ワークステーションに **openstack-playbook-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。たとえば、Playbook を生成する仕様は以下のようになります。

```
apiVersion: osp-director.openstack.org/v1beta1
kind: OpenStackPlaybookGenerator
metadata:
  name: default
  namespace: openstack
spec:
  imageURL: registry.redhat.io/rhosp-beta/openstack-tripleoclient:16.2
  gitSecret: git-secret
  heatEnvConfigMap: heat-env-config
  tarballConfigMap: tripleo-tarball-config
```

リソース仕様に以下の値を設定します。

metadata.name

コンピュートノードのベアメタルセットの名前を **default** に設定します。

metadata.namespace

director Operator namespace を **openstack** に設定します。

spec.gitSecret

Git 認証認証情報 (**git-secret**) を含む ConfigMap に設定します。

spec.tarballConfigMap

カスタムの heat テンプレートを使用して tarball が含まれる ConfigMap (**tripleo-tarball-config**) に設定します。

spec.heatEnvConfigMap

カスタム環境ファイルが含まれる ConfigMap (**heat-env-config**) に設定します。

spec セクションで使用できる値の詳細は、**openstackplaybookgenerator** CRD のカスタムリソース定義の仕様スキーマを確認します。

```
$ oc describe crd openstackplaybookgenerator
```

Ansible Playbook ジェネレーター仕様の設定が完了したら、ファイルを保存します。

2. Ansible Playbook ジェネレーターを作成します。

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

検証

- Playbook ジェネレーターのリソースを表示します。

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

8.16. 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 がノードにオーバークラウドソフトウェアを設定する前に、全ノードのオペレーティングシステムを Red Hat カスタマーポータルまたは Red Hat Satellite Server のいずれかに登録して、ノードのリポジトリーを有効にする必要があります。ノードを登録するには、OpentackPlaybookGenerator リソースで作成されるインベントリーファイルを使用して、**redhat_subscription** Ansible モジュールを使用します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackControlPlane リソースを使用してコントロールプレーンを作成します。
- OpenStackBareMetalSet リソースを使用して、ベアメタルのコンピュートノードを作成します。
- OpentackPlaybookGenerator を使用して、オーバークラウドの Ansible Playbook 設定を作成します。

手順

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

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

2. **cloud-admin** ホームディレクトリーに移動します。

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

3. オプション: オーバークラウドの Ansible Playbook の diff をチェックします。

```
$ ./tripleo-deploy.sh -d
```

4. レンダリングされた Ansible Playbook の最新バージョンを受け入れて **latest** としてタグ付けします。

```
$ ./tripleo-deploy.sh -a
```

最新バージョンの Playbook には、オーバークラウド設定用のインベントリーファイルが含まれます。このインベントリーファイルは、OpenStackClient Pod の **/home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml** にあります。

5. 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-8-for-x86_64-baseos-eus-rpms
      - rhel-8-for-x86_64-appstream-eus-rpms
      - rhel-8-for-x86_64-highavailability-eus-rpms
      - ansible-2.9-for-rhel-8-x86_64-rpms
      - advanced-virt-for-rhel-8-x86_64-rpms
      - openstack-16.1-for-rhel-8-x86_64-rpms
      - rhceph-4-mon-for-rhel-8-x86_64-rpms
      - fast-datapath-for-rhel-8-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 8.4
        pool_ids: 1a85f9223e3d5e43013e3d6e8ff506fd
    - name: Disable all repos
      command: "subscription-manager repos --disable *"
    - name: Enable Controller node repos
      command: "subscription-manager repos --enable {{ item }}"
      with_items: "{{ repos }}"
```

このプレイには、以下の 3 つのタスクが含まれます。

- ノードを登録する。
- 自動的に有効化されるリポジトリーをすべて無効にする。
- コントローラーノードに関連するリポジトリーだけを有効にする。リポジトリーは **repos** 変数でリストされます。

6. 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-playpook -i /home/cloud-admin/playbooks/tripleo-ansible/inventory.yaml ./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

8.17. 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コンピュート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 でオーバークラウドを設定できません。director Operator は、OpenStackControlPlane リソースの作成時に、リモートシェル経由でアクセスする OpenStackClient Pod を作成し、**tripleo-deploy.sh** スクリプトを実行してオーバークラウドを設定します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認します。
- **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ることを確認する。
- OpenStackControlPlane リソースを使用してコントロールプレーンを作成します。
- OpenStackBareMetalSet リソースを使用して、ベアメタルのコンピュートノードを作成します。
- OpentackPlaybookGenerator を使用して、オーバークラウドの Ansible Playbook 設定を作成します。

手順

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

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

2. **cloud-admin** ホームディレクトリーに移動します。

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

3. オプション: オーバークラウドの Ansible Playbook の diff をチェックします。

```
$ ./tripleo-deploy.sh -d
```

4. レンダリングされた Ansible Playbook の最新バージョンを受け入れて **latest** としてタグ付けします。

```
$ ./tripleo-deploy.sh -a
```

5. オーバークラウドノードに対して Ansible Playbook を適用します。

```
$ ./tripleo-deploy.sh -p
```

注記

openstackclient Pod 内の ~/**ansible.log** で Ansible 設定のログを表示できます。

パート III. デプロイメント後操作

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 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メインアクセスポイントは、OpenStackClient Pod で、director Operator を使用して、作成する OpenStackControlPlane リソースの一部として この Pod をデプロイします。

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 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、director Operator 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。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認する。
- OCP クラスターで実行されるオーバークラウドをデプロイして設定する。
- **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ることを確認する。

手順

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

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

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

- 基本的なオーバークラウドフレーバーの作成
- デフォルトのテナントネットワークの作成
- デフォルトの Floating IP ネットワークの作成

- デフォルトのプロバイダーネットワークの作成

9.2. オーバークラウドのダッシュボードへのアクセス

標準のオーバークラウドと同じ手法を使用して、director Operator でデプロイするオーバークラウドのダッシュボードにアクセスします。Web ブラウザーを使用して、オーバークラウドホスト名またはパブリック仮想 IP アドレス (この IP アドレスは、通常 **PublicVirtualFixedIPs** heat パラメーターで設定) にアクセスし、ユーザー名およびパスワードでオーバークラウドのダッシュボードにログインします。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認する。
- OCP クラスターで実行されるオーバークラウドをデプロイして設定する。
- **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ることを確認する。

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. Web ブラウザーを開きます。
3. URL フィールドに、オーバークラウドダッシュボードのホスト名またはパブリック仮想 IP を入力します。
4. 選択した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 を使用したコンピュート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、要件に応じてコンピュート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コンピュートノードの追加

コンピュートノードにさらにオーバークラウドを追加するには、**コンピュート** の OpenStackBaremetalSet リソースのノード数を増やす必要があります。ノード数を増やすと、OpenStackPlaybookGenerator リソースは Ansible Playbook の新しいセットを再生成します。**tripleo-deploy.sh** スクリプトを実行して、新しい Ansible 設定をオーバークラウドに再適用します。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認する。
- OCP クラスターで実行されるオーバークラウドをデプロイして設定する。
- **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ることを確認する。
- **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する。**oc get baremetalhosts -n openshift-machine-api** コマンドを実行して、利用可能なホストを確認します。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。

手順

1. **Compute** OpenStackBaremetalSet の YAML 設定を変更し、リソースの **count** パラメーターを増やします。

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

2. 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。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
```

3. OpenStackPlaybookGenerator リソースは、設定用に新規の Ansible Playbook を自動的に生成します。再生成プロセスが完了するまで待ちます。OpenStackPlaybookGenerator リソースを定期的に確認し、Playbook の Readiness を確認します。

```
$ oc describe openstackplaybookgenerator/default -n openstack
```

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

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

5. **cloud-admin** ホームディレクトリーに移動します。

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

6. オプション: オーバークラウドの Ansible Playbook の diff をチェックします。

```
$ ./tripleo-deploy.sh -d
```

7. レンダリングされた Ansible Playbook の最新バージョンを受け入れて **latest** としてタグ付けします。

```
$ ./tripleo-deploy.sh -a
```

8. オーバークラウドノードに対して Ansible Playbook を適用します。

```
$ ./tripleo-deploy.sh -p
```

関連情報

- 管理対象ベアメタルホスト

10.2. DIRECTOR OPERATOR を使用したオーバークラウドからのコンピュートノードの削除

オーバークラウドからコンピュートノードを削除するには、コンピュート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。

前提条件

- OpenShift Container Platform クラスターが稼働し、director Operator が正しくインストールされていることを確認する。
- OCP クラスターで実行されるオーバークラウドをデプロイして設定する。
- **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ることを確認する。

手順

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

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

2. 新規インスタンスをスケジューリングできないように、ノード上の Compute サービスを削除して無効にするコンピュートノードを特定します。

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

3. **openstackclient** を終了します。

```
$ exit
```

4. 削除するノードに該当する BareMetalHost リソースに、**osp-director.openstack.org/delete-host=true** 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
```

OpenStackBaremetalSet リソースの **annotatedForDeletion** ステータスが変更されます。

```
$ oc get osbms 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"
  }
}
```

5. **Compute** OpenStackBaremetalSet リソースの YAML 設定を変更し、リソースの **count** パラメーターを減らします。

```
oc patch osbms 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、ノードの対応する OpenStackIPSet を削除します。
- 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 .status.roleReservations.compute
{
  "addToPredictableIPs": true,
  "reservations": [
    {
      "deleted": true,
      "hostname": "compute-0",
      "ip": "192.168.25.105",
```

```
      "vip": false
    },
    {
      "deleted": false,
      "hostname": "compute-1",
      "ip": "192.168.25.106",
      "vip": false
    }
  ]
}
```

以下の結果は、ノードの削除および IP 予約の変更が原因で発生します。

- この IP を別のロールでは使用できない。
- 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 サフィックスで始まるホスト名と、対応の IP 予約を再利用する。
- OpenStackBaremetalSet リソースを削除すると、対応するロールの IP 予約がすべて削除されるので、他のロールがこれらの IP アドレスを使用できる。

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

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

7. 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>
```

8. 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
```

9. **openstackclient** を終了します。

```
$ exit
```