Red Hat OpenStack Platform 17.1

Red Hat OpenStack Platform デプロイメントのカスタマイズ

お使いの環境および要件に合わせて、コアの Red Hat OpenStack Platform デプロイメントのカスタマイズ

Last Updated: 2024-05-24

Red Hat OpenStack Platform 17.1 Red Hat OpenStack Platform デプロイメントのカスタマイズ

お使いの環境および要件に合わせて、コアの Red Hat OpenStack Platform デプロイメントのカスタマイズ

OpenStack Team
rhos-docs@redhat.com

法律上の通知

概要

基本的な Red Hat OpenStack Platform デプロイメントを環境と要件に合わせてカスタマイズする方法、および Ansible と Orchestration サービスの使用方法に関するガイダンス

目次

多様性を受け入れるオープンソースの強化

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

RED HAT ドキュメントへのフィードバック (英語のみ)

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

Jira でドキュメントのフィードバックを提供する

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

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

第1章 カスタムのアンダークラウド機能の計画

アンダークラウドに Director を設定してインストールする前に、アンダークラウドにカスタム機能を含めることを計画できます。

1.1. 文字のエンコーディング設定

Red Hat OpenStack Platform には、ロケール設定の一部として特殊文字のエンコーディングに関する要件があります。

- すべてのノードで UTF-8 エンコーディングを使用します。すべてのノードで **LANG** 環境変数を **en_US.UTF-8** に設定するようにします。
- Red Hat OpenStack Platform リソース作成の自動化に Red Hat Ansible Tower を使用している場合は、非 ASCII 文字を使用しないでください。

1.2. プロキシーを使用してアンダークラウドを実行する際の考慮事項

プロキシーを使用してアンダークラウドを実行する場合は特定の制限があります。Red Hat は、レジストリーおよびパッケージ管理に Red Hat Satellite を使用することを推奨します。

ただし、ご自分の環境でプロキシーを使用している場合は、以下の考慮事項を確認して、Red Hat OpenStack Platform の一部とプロキシーを統合する際のさまざまな設定手法、およびそれぞれの手法の制限事項を十分に理解するようにしてください。

システム全体のプロキシー設定

アンダークラウド上のすべてのネットワークトラフィックに対してプロキシー通信を設定するには、この手法を使用します。プロキシー設定を定義するには、**/etc/environment** ファイルを編集して以下の環境変数を設定します。

http_proxy
: 標準の HTTP リクエストに使用するプロキシー

https_proxy
: HTTPs リクエストに使用するプロキシー

no_proxy
: プロキシー通信から除外するドメインのコンマ区切りリスト

システム全体のプロキシー手法には、以下の制限事項があります。

- **pam_env** PAM モジュールのバッファーが固定されているため、**no_proxy** の最大長は 1024 文字です。

dnf プロキシー設定

すべてのトラフィックがプロキシーを通過するように **dnf** を設定するには、この手法を使用します。プロキシー設定を定義するには、**/etc/dnf/dnf.conf** ファイルを編集して以下のパラメーターを設定します。

proxy
: プロキシーサーバーの URL

proxy_username
: プロキシーサーバーへの接続に使用するユーザー名

proxy_password
　プロキシーサーバーへの接続に使用するパスワード

proxy_auth_method
　プロキシーサーバーが使用する認証方法

これらのオプションの詳細については、**man dnf.conf** を実行してください。

dnf プロキシー手法には、以下の制限事項があります。

- この手法では、**dnf** に対してのみプロキシーがサポートされます。
- **dnf** プロキシー手法には、特定のホストをプロキシー通信から除外するオプションは含まれていません。

Red Hat Subscription Manager プロキシー

すべてのトラフィックがプロキシーを通過するように Red Hat Subscription Manager を設定するには、この手法を使用します。プロキシー設定を定義するには、**/etc/rhsm/rhsm.conf** ファイルを編集して以下のパラメーターを設定します。

proxy_hostname
　プロキシーのホスト

proxy_scheme
　プロキシーをリポジトリー定義に書き出す際のプロキシーのスキーム

proxy_port
　プロキシーのポート

proxy_username
　プロキシーサーバーへの接続に使用するユーザー名

proxy_password
　プロキシーサーバーへの接続に使用するパスワード

no_proxy
　プロキシー通信から除外する特定ホストのホスト名接尾辞のコンマ区切りリスト

これらのオプションの詳細については、**man rhsm.conf** を実行してください。

Red Hat Subscription Manager プロキシー手法には、以下の制限事項があります。

- この手法では、Red Hat Subscription Manager に対してのみプロキシーがサポートされます。
- Red Hat Subscription Manager プロキシー設定の値は、システム全体の環境変数に設定されたすべての値をオーバーライドします。

透過プロキシー

アプリケーション層のトラフィックを管理するのにネットワークで透過プロキシーが使用される場合は、プロキシー管理が自動的に行われるため、アンダークラウド自体をプロキシーと対話するように設定する必要はありません。透過プロキシーは、Red Hat OpenStack Platform のクライアントベースのプロキシー設定に関連する制限に対処するのに役立ちます。

第2章 コンポーザブルサービスとカスタムロール

オーバークラウドは、通常コントローラーノードやコンピュートノードなどの事前定義済みロールのノードと、異なる種別のストレージノードで設定されます。これらのデフォルトの各ロールには、director ノード上にあるコアの heat テンプレートコレクションで定義されているサービスセットが含まれます。ただし、特定のサービスのセットが含まれるカスタムロールを作成することもできます。

この柔軟性により、異なるロール上に異なるサービスの組み合わせを作成することができます。本章では、カスタムロールのアーキテクチャー、コンポーザブルサービス、およびそれらを使用する方法について説明します。

2.1. サポートされるロールアーキテクチャー

カスタムロールとコンポーザブルサービスを使用する場合には、以下のアーキテクチャーを使用することができます。

デフォルトアーキテクチャー

デフォルトの **roles_data** ファイルを使用します。すべての Controller サービスが単一の Controller ロールに含まれます。

カスタムコンポーザブルサービス

専用のロールを作成し、それらを使用してカスタムの **roles_data** ファイルを生成します。限られたコンポーザブルサービスの組み合わせしかテスト/検証されていない点に注意してください。Red Hat では、すべてのコンポーザブルサービスの組み合わせに対してサポートを提供することはできません。

2.2. EXAMINING THE ROLES_DATA FILE

roles_data ファイルには、director がノードにデプロイする YAML 形式のロールリストが含まれます。それぞれのロールには、そのロールを設定するすべてのサービスの定義が含まれます。以下のスニペット例を使用して、**roles_data** の構文を説明します。

```
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
- name: Compute
  description: |
    Basic Compute Node role
  ServicesDefault:
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    ...
```

コア heat テンプレートコレクションには、デフォルトの **roles_data** ファイルが **/usr/share/openstack-tripleo-heat-templates/roles_data.yaml** に含まれています。デフォルトのファイルには、以下のロール種別の定義が含まれます。

- **Controller**
- **Compute**
- **BlockStorage**
- **ObjectStorage**
- **CephStorage**

openstack overcloud deploy コマンドにより、デプロイ中にデフォルトの **roles_data.yaml** ファイルが追加されます。ただし、**-r** 引数を使用して、このファイルをカスタムの **roles_data** ファイルでオーバーライドすることができます。

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-custom.yaml
```

2.3. ROLES_DATA ファイルの作成

カスタムの **roles_data** ファイルは、手動で作成することができますが、個別のロールテンプレートを使用して自動生成することも可能です。director には **openstack overcloud role generate** コマンドがあり、複数の事前定義済みロールを結合し、カスタムの **roles_data** ファイルを自動生成します。

手順

1. デフォルトロールのテンプレートをリスト表示します。

```
$ openstack overcloud role list
BlockStorage
CephStorage
Compute
ComputeHCI
ComputeOvsDpdk
Controller
...
```

2. ロール定義を表示します。

```
$ openstack overcloud role show Compute
```

3. **Controller** ロール、**Compute** ロール、および **Networker** ロールが含まれるカスタムの **roles_data.yaml** ファイルを生成します。

```
$ openstack overcloud roles \
 generate -o <custom_role_file> \
 Controller Compute Networker
```

- **<custom_role_file>** を、生成する新しいロールファイルの名前と場所 (**/home/stack/templates/roles_data.yaml** など) に置き換えます。

注記

Controller ロールおよび **Networker** ロールには、同じネットワークエージェントが含まれます。つまり、ネットワークサービスは **Controller** ロールから **Networker** ロールにスケーリングされ、オーバークラウドは **Controller** ノードと **Networker** ノード間にネットワークサービスの負荷のバランスを取ります。

この **Networker** ロールをスタンドアロンにするには、独自のカスタム **Controller** ロールと、その他の必要なロールを作成することができます。これにより、独自のカスタムロールから **roles_data.yaml** ファイルを生成できるようになります。

4. コア heat テンプレートコレクションから **roles** ディレクトリーを **stack** ユーザーのホームディレクトリーにコピーします。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles/. /home/stack/templates/roles/
```

5. このディレクトリー内でカスタムロールファイルを追加または変更します。このディレクトリーをカスタムロールのソースとして使用するには、ロールのサブコマンドに **--roles-path** オプションを指定します。

```
$ openstack overcloud role \
 generate -o my_roles_data.yaml \
 --roles-path /home/stack/templates/roles \
 Controller Compute Networker
```

このコマンドにより、~/**roles** ディレクトリー内の個々のロールから、単一の **my_roles_data.yaml** ファイルが生成されます。

注記

デフォルトのロールコレクションには、**ControllerOpenstack** ロールも含まれます。このロールには、**Networker**、**Messaging**、および **Database** ロールのサービスは含まれません。**ControllerOpenstack** は、スタンドアロンの **Networker**、**Messaging**、**Database** ロールと組み合わせて使用することができます。

2.4. サポートされるカスタムロール

以下の表で、利用可能なカスタムロールについて説明します。カスタムロールテンプレートは、**/usr/share/openstack-tripleo-heat-templates/roles** ディレクトリーにあります。

ロール	説明	ファイル
BlockStorage	OpenStack Block Storage (cinder) ノード	**BlockStorage.yaml**
CephAll	完全なスタンドアロンの Ceph Storage ノード。OSD、MON、Object Gateway (RGW)、Object Operations (MDS)、Manager (MGR)、および RBD Mirroring が含まれます。	**CephAll.yaml**

ロール	説明	ファイル
CephFile	スタンドアロンのスケールアウト Ceph Storage ファイルロール。OSD および Object Operations (MDS) が含まれます。	**CephFile.yaml**
CephObject	スタンドアロンのスケールアウト Ceph Storage オブジェクトロール。OSD および Object Gateway (RGW) が含まれます。	**CephObject.yaml**
CephStorage	Ceph Storage OSD ノードロール	**CephStorage.yaml**
ComputeAlt	代替のコンピュートノードロール	**ComputeAlt.yaml**
ComputeDVR	DVR 対応のコンピュートノードロール	**ComputeDVR.yaml**
ComputeHCI	ハイパーコンバージドインフラストラクチャーを持つコンピュートノード。Compute および Ceph OSD サービスが含まれます。	**ComputeHCI.yaml**
ComputeInstanceHA	コンピュートインスタンス HA ノードロール。**environments/compute-instanceha.yaml** 環境ファイルと共に使用します。	**ComputeInstanceHA.yaml**
ComputeLiquidio	Cavium Liquidio Smart NIC を持つコンピュートノード	**ComputeLiquidio.yaml**
ComputeOvsDpdkRT	コンピュート OVS DPDK RealTime ロール	**ComputeOvsDpdkRT.yaml**
ComputeOvsDpdk	コンピュート OVS DPDK ロール	**ComputeOvsDpdk.yaml**
ComputeRealTime	リアルタイムのパフォーマンスに最適化された Compute ロール。このロールを使用する場合には、**overcloud-realtime-compute** イメージが利用可能で、ロール固有のパラメーター **IsolCpusList**、**NovaComputeCpuDedicatedSet**、および **NovaComputeCpuSharedSet** がリアルタイムコンピュートノードのハードウェアに応じて設定されている必要があります。	**ComputeRealTime.yaml**
ComputeSriovRT	コンピュート SR-IOV RealTime ロール	**ComputeSriovRT.yaml**
ComputeSriov	コンピュート SR-IOV ロール	**ComputeSriov.yaml**
Compute	標準のコンピュートノードロール	**Compute.yaml**

ロール	説明	ファイル
ControllerAllNovaStandalone	データベース、メッセージング、ネットワーク設定、および OpenStack Compute (nova) コントロールコンポーネントを持たない Controller ロール。**Database**、**Messaging**、**Networker**、および **Novacontrol** ロールと組み合わせて使用します。	**ControllerAllNovaStandalone.yaml**
ControllerNoCeph	コア Controller サービスは組み込まれているが Ceph Storage (MON) コンポーネントを持たない Controller ロール。このロールはデータベース、メッセージング、およびネットワーク機能を処理しますが、Ceph Storage 機能は処理しません。	**ControllerNoCeph.yaml**
ControllerNovaStandalone	OpenStack Compute (nova) コントロールコンポーネントが含まれない Controller ロール。**Novacontrol** ロールと組み合わせて使用します。	**ControllerNovaStandalone.yaml**
ControllerOpenstack	データベース、メッセージング、およびネットワーク設定コンポーネントが含まれない Controller ロール。**Database**、**Messaging**、および **Networker** ロールと組み合わせて使用します。	**ControllerOpenstack.yaml**
ControllerStorageNfs	すべてのコアサービスが組み込まれ、Ceph NFS を使用する Controller ロール。このロールはデータベース、メッセージング、およびネットワーク機能を処理します。	**ControllerStorageNfs.yaml**
Controller	すべてのコアサービスが組み込まれた Controller ロール。このロールはデータベース、メッセージング、およびネットワーク機能を処理します。	**Controller.yaml**
ControllerSriov (ML2/OVN)	通常の Controller ロールと同じですが、OVN Metadata エージェントがデプロイされます。	**ControllerSriov.yaml**
Database	スタンドアロンのデータベースロール。Pacemaker を使用して Galera クラスターとして管理されるデータベース。	**Database.yaml**
HciCephAll	ハイパーコンバージドインフラストラクチャーおよびすべての Ceph Storage サービスを持つコンピュートノード。OSD、MON、Object Gateway (RGW)、Object Operations (MDS)、Manager (MGR)、および RBD Mirroring が含まれます。	**HciCephAll.yaml**
HciCephFile	ハイパーコンバージドインフラストラクチャーおよび Ceph Storage ファイルサービスを持つコンピュートノード。OSD および Object Operations (MDS) が含まれます。	**HciCephFile.yaml**

ロール	説明	ファイル
HciCephMon	ハイパーコンバージドインフラストラクチャーおよび Ceph Storage ブロックサービスを持つコンピュートノード。OSD、MON、および Manager が含まれます。	**HciCephMon.yaml**
HciCephObject	ハイパーコンバージドインフラストラクチャーおよび Ceph Storage オブジェクトサービスを持つコンピュートノード。OSD および Object Gateway (RGW) が含まれます。	**HciCephObject.yaml**
IronicConductor	Ironic Conductor ノードロール	**IronicConductor.yaml**
Messaging	スタンドアロンのメッセージングロール。Pacemaker を使用して管理される RabbitMQ。	**Messaging.yaml**
Networker	スタンドアロンのネットワーク設定ロール。単独で OpenStack Networking (neutron) エージェントを実行します。デプロイメントで ML2/OVN メカニズムドライバーを使用する場合は、ML2/OVN を使用したカスタムロールのデプロイ の追加手順を参照してください。	**Networker.yaml**
NetworkerSriov	通常の Networker ロールと同じですが、OVN Metadata エージェントがデプロイされます。ML2/OVN を使用したカスタムロールのデプロイ の追加手順を参照してください。	**NetworkerSriov.yaml**
Novacontrol	スタンドアロンの **nova-control** ロール。単独で OpenStack Compute (nova) コントロールエージェントを実行します。	**Novacontrol.yaml**
ObjectStorage	Swift オブジェクトストレージノードロール	**ObjectStorage.yaml**
Telemetry	すべてのメトリックおよびアラームサービスを持つ Telemetry ロール	**Telemetry.yaml**

2.5. ロールパラメーターの考察

それぞれのロールには以下のパラメーターが含まれます。

name

(必須) 空白または特殊文字を含まないプレーンテキスト形式のロール名。選択した名前により、他のリソースとの競合が発生しないことを確認します。たとえば、**Network** の代わりに **Networker** を名前に使用します。

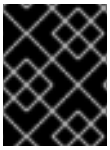
description

(オプション) プレーンテキスト形式のロールの説明

tags

(オプション) ロールのプロパティーを定義するタグの YAML リスト。このパラメーターを使用して **controller** と **primary** タグの両方で、プライマリーロールを定義します。

```
- name: Controller
  ...
  tags:
    - primary
    - controller
  ...
```



重要

プライマリーロールをタグ付けしない場合には、最初に定義するロールがプライマリーロールになります。このロールが Controller ロールとなるようにしてください。

networks

ロール上で設定するネットワークの YAML リストまたはディクショナリー。YAML リストを使用する場合には、各コンポーザブルネットワークのリストを含めます。

```
networks:
  - External
  - InternalApi
  - Storage
  - StorageMgmt
  - Tenant
```

ディクショナリーを使用する場合は、各ネットワークをコンポーザブルネットワークの特定の **subnet** にマッピングします。

```
networks:
  External:
    subnet: external_subnet
  InternalApi:
    subnet: internal_api_subnet
  Storage:
    subnet: storage_subnet
  StorageMgmt:
    subnet: storage_mgmt_subnet
  Tenant:
    subnet: tenant_subnet
```

デフォルトのネットワークに
は、**External**、**InternalApi**、**Storage**、**StorageMgmt**、**Tenant**、**Management** が含まれます。

CountDefault

(任意) このロールにデプロイするデフォルトのノード数を定義します。

HostnameFormatDefault

(任意) このロールに対するホスト名のデフォルトの形式を定義します。デフォルトの命名規則では、以下の形式が使用されます。

```
[STACK NAME]-[ROLE NAME]-[NODE ID]
```

たとえば、コントローラーノード名はデフォルトで以下のように命名されます。

```
overcloud-controller-0
overcloud-controller-1
overcloud-controller-2
...
```

disable_constraints

(任意) director によるデプロイ時に OpenStack Compute (nova) および OpenStack Image Storage (glance) の制約を無効にするかどうかを定義します。事前にプロビジョニングされたノードでオーバークラウドをデプロイする場合に、このパラメーターを使用します。詳細は、**Director のインストールおよび使用** ガイドの 事前にプロビジョニングされたノードを使用した基本的なオーバークラウドの設定 を参照してください。

update_serial

(任意) OpenStack の更新オプション時に同時に更新するノードの数を定義します。**roles_data.yaml** ファイルのデフォルト設定は以下のとおりです。

- コントローラー、オブジェクトストレージ、および Ceph Storage ノードのデフォルトは **1** です。
- コンピュートおよび Block Storage ノードのデフォルトは **25** です。

このパラメーターをカスタムロールから省いた場合のデフォルトは **1** です。

ServicesDefault

(任意) ノード上で追加するデフォルトのサービスリストを定義します。詳細は、「コンポーザブルサービスアーキテクチャーの考察」 を参照してください。

これらのパラメーターを使用して、新規ロールを作成すると共にロールに追加するサービスを定義することができます。

openstack overcloud deploy コマンドは、**roles_data** ファイルのパラメーターをいくつかの Jinja2 ベースのテンプレートに統合します。たとえば、特定の時点で **overcloud.j2.yaml** heat テンプレートは **roles_data.yaml** のロールのリストを繰り返し適用し、対応する各ロール固有のパラメーターとリソースを作成します。

たとえば、**overcloud.j2.yaml** heat テンプレートの各ロールのリソース定義は、以下のスニペットのようになります。

```
{{role.name}}:
  type: OS::Heat::ResourceGroup
  depends_on: Networks
  properties:
    count: {get_param: {{role.name}}Count}
    removal_policies: {get_param: {{role.name}}RemovalPolicies}
    resource_def:
      type: OS::TripleO::{{role.name}}
      properties:
        CloudDomain: {get_param: CloudDomain}
```

```
        ServiceNetMap: {get_attr: [ServiceNetMap, service_net_map]}
        EndpointMap: {get_attr: [EndpointMap, endpoint_map]}
...
```

このスニペットには、Jinja2 ベースのテンプレートが **{{role.name}}** の変数を組み込み、各ロール名を **OS::Heat::ResourceGroup** リソースとして定義しているのが示されています。これは、次に **roles_data** ファイルのそれぞれの **name** パラメーターを使用して、対応する各 **OS::Heat::ResourceGroup** リソースを命名します。

2.6. 新規ロールの作成

コンポーザブルサービスアーキテクチャーを使用して、デプロイメントの要件に従ってベアメタルノードにロールを割り当てることができます。たとえば、OpenStack Dashboard (**horizon**) だけをホストする新しい **Horizon** ロールを作成するケースを考えます。

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
[stack@director ~]$ source ~/stackrc
```

3. コア heat テンプレートコレクションから **roles** ディレクトリーを **stack** ユーザーのホームディレクトリーにコピーします。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles/. /home/stack/templates/roles/
```

4. **home/stack/templates/roles** に **Horizon.yaml** という名前の新規のファイルを作成します。
5. 以下の設定を **Horizon.yaml** に追加して、ベースおよびコアの OpenStack Dashboard サービスが含まれる新しい **Horizon** ロールを作成します。

```
- name: Horizon (1)
  CountDefault: 1 (2)
  HostnameFormatDefault: '%stackname%-horizon-%index%'
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::TripleoPackages
    - OS::TripleO::Services::TripleoFirewall
    - OS::TripleO::Services::SensuClient
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::AuditD
    - OS::TripleO::Services::Collectd
    - OS::TripleO::Services::MySQLClient
    - OS::TripleO::Services::Apache
    - OS::TripleO::Services::Horizon
```

1. **name** パラメーターをカスタムロールの名前に設定します。カスタムロール名の最大長は 47 文字です。

2. **CountDefault** パラメーターを **1** に設定して、デフォルトのオーバークラウドに常に **Horizon** ノードが含まれるようにすると良いでしょう。

6. (オプション) 既存のオーバークラウド内でサービスをスケーリングする場合は、既存のサービスを **Controller** ロール上に保持します。新規オーバークラウドを作成して、OpenStack Dashboard がスタンドアロンのロールに残るようにするには、**Controller** ロールの定義から OpenStack Dashboard コンポーネントを削除します。

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    ...
    - OS::TripleO::Services::GnocchiMetricd
    - OS::TripleO::Services::GnocchiStatsd
    - OS::TripleO::Services::HAproxy
    - OS::TripleO::Services::HeatApi
    - OS::TripleO::Services::HeatApiCfn
    - OS::TripleO::Services::HeatApiCloudwatch
    - OS::TripleO::Services::HeatEngine
    # - OS::TripleO::Services::Horizon          # Remove this service
    - OS::TripleO::Services::IronicApi
    - OS::TripleO::Services::IronicConductor
    - OS::TripleO::Services::Iscsid
    - OS::TripleO::Services::Keepalived
    ...
```

7. **Controller**、**Compute**、および **Horizon** のロールを含む、**roles_data_horizon.yaml** という名前の新しいロールデータファイルを生成します。

```
(undercloud)$ openstack overcloud roles \
 generate -o /home/stack/templates/roles_data_horizon.yaml \
 --roles-path /home/stack/templates/roles \
 Controller Compute Horizon
```

8. オプション: **overcloud-baremetal-deploy.yaml** ノード定義ファイルを編集して、Horizon ノードの配置を設定します。

```
- name: Controller
  count: 3
  instances:
  - hostname: overcloud-controller-0
    name: node00
  ...
- name: Compute
  count: 3
  instances:
  - hostname: overcloud-novacompute-0
    name: node04
  ...
- name: Horizon
  count: 1
```

```
instances:
- hostname: overcloud-horizon-0
  name: node07
```

2.7. ガイドラインおよび制限事項

コンポーザブルロールのアーキテクチャーには、以下のガイドラインおよび制限事項があることに注意してください。

Pacemaker により管理されないサービスの場合:

- スタンドアロンのカスタムロールにサービスを割り当てることができます。
- 初回のデプロイメント後に追加のカスタムロールを作成してそれらをデプロイし、既存のサービスをスケーリングすることができます。

Pacemaker により管理されるサービスの場合:

- スタンドアロンのカスタムロールに Pacemaker のマネージドサービスを割り当てることができます。
- Pacemaker のノード数の上限は 16 です。Pacemaker サービス (**OS::TripleO::Services::Pacemaker**) を 16 のノードに割り当てた場合には、それ以降のノードは、代わりに Pacemaker Remote サービス (**OS::TripleO::Services::PacemakerRemote**) を使用する必要があります。同じロールで Pacemaker サービスと Pacemaker Remote サービスを割り当てることはできません。
- Pacemaker のマネージドサービスが割り当てられていないロールには、Pacemaker サービス (**OS::TripleO::Services::Pacemaker**) を追加しないでください。
- **OS::TripleO::Services::Pacemaker** または **OS::TripleO::Services::PacemakerRemote** のサービスが含まれているカスタムロールはスケールアップまたはスケールダウンできません。

一般的な制限事項

- メジャーバージョン間のアップグレードプロセス中にカスタムロールとコンポーザブルサービスを変更することはできません。
- オーバクラウドのデプロイ後には、ロールのサービスリストを変更することはできません。オーバークラウドのデプロイ後にサービスリストを変更すると、デプロイでエラーが発生して、ノード上に孤立したサービスが残ってしまう可能性があります。

2.8. コンテナー化されたサービスのアーキテクチャー

director は、OpenStack Platform のコアサービスをオーバークラウド上にコンテナーとしてインストールします。コンテナー化されたサービス用のテンプレートは、**/usr/share/openstack-tripleo-heat-templates/deployment/** にあります。

コンテナー化されたサービスを使用するすべてのノードで、ロールの **OS::TripleO::Services::Podman** サービスを有効にする必要があります。カスタムロール設定用の **roles_data.yaml** ファイルを作成する際には、ベースコンポーザブルサービスとともに **OS::TripleO::Services::Podman** サービスを追加します。たとえば、**IronicConductor** ロールには、以下の定義を使用します。

```
- name: IronicConductor
  description: |
```

```
  Ironic Conductor node role
networks:
  InternalApi:
    subnet: internal_api_subnet
  Storage:
    subnet: storage_subnet
HostnameFormatDefault: '%stackname%-ironic-%index%'
ServicesDefault:
  - OS::TripleO::Services::Aide
  - OS::TripleO::Services::AuditD
  - OS::TripleO::Services::BootParams
  - OS::TripleO::Services::CACerts
  - OS::TripleO::Services::CertmongerUser
  - OS::TripleO::Services::Collectd
  - OS::TripleO::Services::Docker
  - OS::TripleO::Services::Fluentd
  - OS::TripleO::Services::IpaClient
  - OS::TripleO::Services::Ipsec
  - OS::TripleO::Services::IronicConductor
  - OS::TripleO::Services::IronicPxe
  - OS::TripleO::Services::Kernel
  - OS::TripleO::Services::LoginDefs
  - OS::TripleO::Services::MetricsQdr
  - OS::TripleO::Services::MySQLClient
  - OS::TripleO::Services::ContainersLogrotateCrond
  - OS::TripleO::Services::Podman
  - OS::TripleO::Services::Rhsm
  - OS::TripleO::Services::SensuClient
  - OS::TripleO::Services::Snmp
  - OS::TripleO::Services::Timesync
  - OS::TripleO::Services::Timezone
  - OS::TripleO::Services::TripleoFirewall
  - OS::TripleO::Services::TripleoPackages
  - OS::TripleO::Services::Tuned
```

2.9. コンテナー化されたサービスのパラメーター

コンテナー化されたサービスのテンプレートにはそれぞれ、**outputs** セクションがあります。このセクションでは、OpenStack Orchestration (heat) サービスに渡すデータセットを定義します。テンプレートには、標準のコンポーザブルサービスパラメーターに加えて、コンテナー設定に固有のパラメーターセットが含まれます。

puppet_config

サービスの設定時に Puppet に渡すデータ。初期のオーバークラウドデプロイメントステップでは、director は、コンテナー化されたサービスが実際に実行される前に、サービスの設定に使用するコンテナーのセットを作成します。このパラメーターには以下のサブパラメーターが含まれます。

- **config_volume**: 設定を格納するマウント済みのボリューム
- **puppet_tags**: 設定中に Puppet に渡すタグ。OpenStack は、これらのタグを使用して Puppet の実行を特定サービスの設定リソースに制限します。たとえば、OpenStack Identity (keystone) のコンテナー化されたサービスは、**keystone_config** タグを使用して、設定コンテナーで **keystone_config** Puppet リソースを実行します。

- **step_config**: Puppet に渡される設定データ。これは通常、参照されたコンポーザブルサービスから継承されます。
- **config_image**: サービスを設定するためのコンテナーイメージ

kolla_config

設定ファイルの場所、ディレクトリーのパーミッション、およびサービスを起動するためにコンテナー上で実行するコマンドを定義するコンテナー固有のデータセット

docker_config

サービスの設定コンテナーで実行するタスク。すべてのタスクは以下に示すステップにグループ化され、director が段階的にデプロイメントを行うのに役立ちます。

- **ステップ 1**: ロードバランサーの設定
- **ステップ 2**: コアサービス (データベース、Redis)
- **ステップ 3**: OpenStack Platform サービスの初期設定
- **ステップ 4**: OpenStack Platform サービスの全般設定
- **ステップ 5**: サービスのアクティブ化

host_prep_tasks

ベアメタルノードがコンテナー化されたサービスに対応するための準備タスク

2.10. コンポーザブルサービスアーキテクチャーの考察

コア heat テンプレートコレクションには、コンポーザブルサービスのテンプレートセットが 2 つ含まれています。

- **deployment** には、主要な OpenStack サービスのテンプレートが含まれます。
- **puppet/services** には、コンポーザブルサービスを設定するためのレガシーテンプレートが含まれます。互換性を維持するために、一部のコンポーザブルサービスは、このディレクトリーからのテンプレートを使用する場合があります。多くの場合、コンポーザブルサービスは **deployment** ディレクトリーのテンプレートを使用します。

各テンプレートには目的を特定する記述が含まれています。たとえば、**deployment/time/ntp-baremetal-puppet.yaml** サービステンプレートには以下のような記述が含まれます。

```
description: >
  NTP service deployment using puppet, this YAML file
  creates the interface between the HOT template
  and the puppet manifest that actually installs
  and configure NTP.
```

これらのサービステンプレートは、Red Hat OpenStack Platform デプロイメント固有のリソースとして登録されます。これは、**overcloud-resource-registry-puppet.j2.yaml** ファイルで定義されている一意な heat リソース名前空間を使用して、各リソースを呼び出すことができることを意味します。サービスはすべて、リソース種別に **OS::TripleO::Services** 名前空間を使用します。

一部のリソースは、直接コンポーザブルサービスのベーステンプレートを使用します。

```
resource_registry:
```

```
  ...
  OS::TripleO::Services::Ntp: deployment/time/ntp-baremetal-puppet.yaml
  ...
```

ただし、コアサービスにはコンテナーが必要なので、コンテナー化されたサービステンプレートを使用します。たとえば、コンテナー化された **keystone** サービスでは、以下のリソースを使用します。

```
resource_registry:
  ...
  OS::TripleO::Services::Keystone: deployment/keystone/keystone-container-puppet.yaml
  ...
```

通常、これらのコンテナー化されたテンプレートは、依存関係を追加するために他のテンプレートを参照します。たとえば、**deployment/keystone/keystone-container-puppet.yaml** テンプレートは、**ContainersCommon** リソースにベーステンプレートの出力を保管します。

```
resources:
  ContainersCommon:
    type: ../containers-common.yaml
```

これにより、コンテナー化されたテンプレートは、**containers-common.yaml** テンプレートからの機能やデータを取り込むことができます。

overcloud.j2.yaml heat テンプレートには、**roles_data.yaml** ファイル内の各カスタムロールのサービスリストを定義するための Jinja2-based コードのセクションが含まれています。

```
{{role.name}}Services:
  description: A list of service resources (configured in the heat
               resource_registry) which represent nested stacks
               for each service that should get installed on the {{role.name}} role.
  type: comma_delimited_list
  default: {{role.ServicesDefault|default([])}}
```

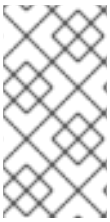
デフォルトのロールの場合は、これにより次のサービスリストパラメーターが作成されます: **ControllerServices**、**ComputeServices**、**BlockStorageServices**、**ObjectStorageServices**、**CephStorageServices**

roles_data.yaml ファイル内の各カスタムロールのデフォルトのサービスを定義します。たとえば、デフォルトの Controller ロールには、以下の内容が含まれます。

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
    - OS::TripleO::Services::CinderBackup
    - OS::TripleO::Services::CinderScheduler
    - OS::TripleO::Services::CinderVolume
    - OS::TripleO::Services::Core
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Keystone
```

```
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
...
```

これらのサービスは、次に **ControllerServices** パラメーターのデフォルトリストとして定義されます。



注記

環境ファイルを使用してサービスパラメーターのデフォルトリストを上書きすることもできます。たとえば、環境ファイルで **ControllerServices** を **parameter_default** として定義して、**roles_data.yaml** ファイルからのサービスリストを上書きすることができます。

2.11. ロールへのサービスの追加と削除

サービスの追加と削除の基本的な方法では、ノードロールのデフォルトサービスリストのコピーを作成してからサービスを追加/削除します。たとえば、OpenStack Orchestration (heat) をコントローラーノードから削除するケースを考えます。

手順

1. デフォルトの **roles** ディレクトリーのカスタムコピーを作成します。

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/roles ~/.
```

2. ~/**roles**/**Controller.yaml** ファイルを編集して、**ServicesDefault** パラメーターのサービスリストを変更します。OpenStack Orchestration のサービスまでスクロールしてそれらを削除します。

```
    - OS::TripleO::Services::GlanceApi
    - OS::TripleO::Services::GlanceRegistry
    - OS::TripleO::Services::HeatApi            # Remove this service
    - OS::TripleO::Services::HeatApiCfn         # Remove this service
    - OS::TripleO::Services::HeatApiCloudwatch  # Remove this service
    - OS::TripleO::Services::HeatEngine         # Remove this service
    - OS::TripleO::Services::MySQL
    - OS::TripleO::Services::NeutronDhcpAgent
```

3. 新しい **roles_data** ファイルを生成します。

```
$ openstack overcloud roles generate -o roles_data-no_heat.yaml \
  --roles-path ~/roles \
  Controller Compute Networker
```

4. **openstack overcloud deploy** コマンドを実行する際には、この新しい **roles_data** ファイルを指定します。

```
$ openstack overcloud deploy --templates -r ~/templates/roles_data-no_heat.yaml
```

このコマンドにより、コントローラーノードには OpenStack Orchestration のサービスがインストールされない状態でオーバークラウドがデプロイされます。

注記

また、カスタムの環境ファイルを使用して、**roles_data** ファイル内のサービスを無効にすることもできます。無効にするサービスを **OS::Heat::None** リソースにリダイレクトします。以下に例を示します。

```
resource_registry:
  OS::TripleO::Services::HeatApi: OS::Heat::None
  OS::TripleO::Services::HeatApiCfn: OS::Heat::None
  OS::TripleO::Services::HeatApiCloudwatch: OS::Heat::None
  OS::TripleO::Services::HeatEngine: OS::Heat::None
```

2.12. 無効化されたサービスの有効化

一部のサービスはデフォルトで無効化されています。これらのサービスは、**overcloud-resource-registry-puppet.j2.yaml** ファイルで null 操作 (**OS::Heat::None**) として登録されます。たとえば、Block Storage のバックアップサービス (**cinder-backup**) は無効化されています。

```
OS::TripleO::Services::CinderBackup: OS::Heat::None
```

このサービスを有効化するには、**puppet/services** ディレクトリー内の対応する heat テンプレートにリソースをリンクする環境ファイルを追加します。一部のサービスには、**environments** ディレクトリー内に事前定義済みの環境ファイルがあります。たとえば、Block Storage のバックアップサービスは、以下のような内容を含む **environments/cinder-backup.yaml** ファイルを使用します。

手順

1. **CinderBackup** サービスを **cinder-backup** 設定を含む heat テンプレートにリンクする環境ファイルにエントリーを追加します。

   ```
   resource_registry:
     OS::TripleO::Services::CinderBackup: ../podman/services/pacemaker/cinder-backup.yaml
   ...
   ```

   このエントリーにより、デフォルトの null 操作のリソースが上書きされ、これらのサービスが有効になります。

2. **openstack overcloud deploy** コマンドの実行時に、この環境ファイルを指定します。

   ```
   $ openstack overcloud deploy --templates -e /usr/share/openstack-tripleo-heat-
   templates/environments/cinder-backup.yaml
   ```

第3章 検証フレームワークの使用

Red Hat OpenStack Platform (RHOSP) には検証フレームワークが含まれており、アンダークラウドおよびオーバークラウドの要件と機能の検証に使用することができます。フレームワークには、以下に示す 2 つの検証種別が含まれます。

- **validation** コマンドセットを使用して実行する手動の Ansible ベースの検証。
- インフライトの自動検証: デプロイメントプロセス中に実行されます。

実行する検証を理解し、環境に関係のない検証をスキップする必要があります。たとえば、事前デプロイメントの検証には、どこでも TLS のテストが含まれます。環境を TLS 用に設定する予定がない場合、どこでも、このテストは失敗します。**validation run** コマンドで **--validation** オプションを使用して、環境に応じて検証を調整します。

3.1. ANSIBLE ベースの検証

Red Hat OpenStack Platform (RHOSP) director のインストール時に、director は **openstack-tripleo-validations** パッケージから Playbook のセットもインストールします。それぞれの Playbook には、特定のシステム要件のテストおよびグループが含まれます。このグループを使用して、OpenStack Platform のライフサイクル中の特定のステージにタグ付けされた一連のテストを実施することができます。

no-op

no-op (操作なし) タスクを実行してワークフローが正しく機能していることを確かめる検証。これらの検証は、アンダークラウドとオーバークラウドの両方で実行されます。

prep

アンダークラウドノードのハードウェア設定を確認する検証。**openstack undercloud install** コマンドを実行する前に、これらの検証を実行します。

openshift-on-openstack

環境が要件を満たし OpenShift on OpenStack をデプロイできることを確認する検証

pre-introspection

Ironic Inspector を使用するノードのイントロスペクション前に実行する検証

pre-deployment

openstack overcloud deploy コマンドの前に実行する検証

post-deployment

オーバークラウドのデプロイメントが完了した後に実行する検証

pre-update

更新前に RHOSP デプロイメントを検証するための検証。

post-update

更新後に RHOSP デプロイメントを検証するための検証。

pre-upgrade

アップグレード前に RHOSP デプロイメントを検証するための検証。

post-upgrade

アップグレード後に RHOSP デプロイメントを検証するための検証。

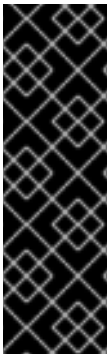
3.2. 検証設定ファイルの変更

検証設定ファイルは、検証の実行とリモートマシン間の通信のあらゆる側面を制御するために編集できる .ini ファイルです。

次のいずれかの方法で、デフォルトの設定値を変更できます。

- デフォルトの /etc/validations.cfg ファイルを編集します。
- デフォルトの **/etc/validations.cfg** ファイルの独自のコピーを作成して編集し、CLI から **--config** 引数を使用して渡します。設定ファイルの独自のコピーを作成する場合は、**--config** を使用して、実行のたびに CLI でこのファイルを指定します。

デフォルトでは、検証設定ファイルの場所は **/etc/validation.cfg** です。



重要

設定ファイルを正しく編集していることを確認してください。そうしないと、検証が次のようなエラーで失敗する可能性があります。

- 検出されない検証
- 異なる場所に書き込まれたコールバック
- 誤って解析されたログ

前提条件

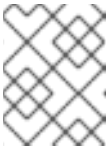
- 環境を検証する方法を完全に理解している。

手順

1. オプション: 編集用に検証設定ファイルのコピーを作成します。
   a. **/etc/validation.cfg** をホームディレクトリーにコピーします。
   b. 新しい設定ファイルに必要な編集を加えます。
2. 検証コマンドを実行します。

```
$ validation run --config <configuration-file>
```

   - <configuration-file> は、使用する設定ファイルへのファイルパスに置き換えます。



注記

検証を実行すると、出力の **Reasons** 列は 79 文字に制限されます。検証結果を完全に表示するには、検証ログファイルを表示します。

3.3. 検証のリスト表示

検証リスト コマンドを実行して、利用可能なさまざまな種類の検証を一覧表示します。

手順

1. source コマンドで **stackrc** ファイルを読み込みます。

```
$ source ~/stackrc
```

2. **検証リスト** コマンドを実行します。
   - すべての検証をリスト表示するには、オプションを設定せずにコマンドを実行します。

     ```
     $ validation list
     ```

   - グループの検証をリスト表示するには、**--group** オプションを指定してコマンドを実行します。

     ```
     $ validation list --group prep
     ```

注記

オプションの完全なリストについては、**validation list --help** を実行してください。

3.4. 検証の実行

検証または検証グループを実行するには、**validation run** コマンドを使用します。オプションの完全なリストを表示するには、**validation run --help** コマンドを使用します。

注記

検証を実行すると、出力の **Reasons** 列は 79 文字に制限されます。検証結果を完全に表示するには、検証ログファイルを表示します。

手順

1. **stackrc** ファイルを取得します。

   ```
   $ source ~/stackrc
   ```

2. **tripleo-ansible-inventory.yaml** という静的インベントリーファイルを検証します。

   ```
   $ validation run --group pre-introspection -i tripleo-ansible-inventory.yaml
   ```

注記

インベントリーファイルは、スタンドアロンまたはアンダークラウドデプロイメントの場合は **~/tripleo-deploy/<stack>** ディレクトリー、オーバークラウドデプロイメントの場合は **~/overcloud-deploy/<stack>** ディレクトリーにあります。

3. **検証実行** コマンドを入力します。
   - 単一の検証を実行するには、**--validation** オプションで検証の名前を指定してコマンドを入力します。たとえば、各ノードのメモリー要件を確認するには、**--validation check-ram** を入力します。

     ```
     $ validation run --validation check-ram
     ```

複数の特定の検証を実行するには、実行する検証のコンマ区切りリストとともに **--validation** オプションを使用します。使用可能な検証のリストを表示する方法の詳細については、Listing validations を参照してください。

- グループのすべての検証を実行するには、**--group** オプションを指定してコマンドを入力します。

```
$ validation run --group prep
```

特定の検証からの詳細な出力を表示するには、レポートからの特定の検証 UUID に対して **validation history get --full** コマンドを実行します。

```
$ validation history get --full <UUID>
```

3.5. 検証の作成

validation init コマンドを使用して検証を作成できます。コマンドを実行すると、新しい検証用の基本テンプレートが作成されます。要件に合わせて新しい検証ロールを編集できます。

重要

Red Hat は、ユーザー作成の検証をサポートしていません。

前提条件

- 環境を検証する方法を完全に理解している。
- コマンドを実行するディレクトリーへのアクセス権がある。

手順

1. 検証を作成します。

```
$ validation init <my-new-validation>
```

- **<my-new-validation>** は、新しい検証の名前に置き換えます。
このコマンドを実行すると、次のディレクトリーとサブディレクトリーが作成されます。

```
/home/stack/community-validations
├── library
├── lookup_plugins
├── playbooks
└── roles
```

注記

The Community Validations are disabled by default というエラーメッセージが表示された場合は、検証設定ファイルで **enable_community_validations** パラメーターが **True** に設定されていることを確認してください。このファイルのデフォルトの名前と場所は **/etc/validation.cfg** です。

2. 要件に合わせてロールを編集します。

関連情報

- 「検証設定ファイルの変更」.

3.6. 検証履歴の表示

検証または検証グループの実行後に、director は各検証の結果を保存します。**validation history list** コマンドを使用して、過去の検証結果を表示します。

前提条件

- 検証または検証グループを実行している。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. すべての検証または最新の検証のリストを表示できます。
   - すべての検証のリストを表示します。

```
$ validation history list
```

   - **--validation** オプションを使用して、特定の検証タイプの履歴を表示します。

```
$ validation history get --validation <validation-type>
```

     - **<validation-type>** は、検証のタイプ (ntp など) に置き換えます。
4. 特定の検証 UUID のログを表示します。

```
$ validation show run --full 7380fed4-2ea1-44a1-ab71-aab561b44395
```

関連情報

- Assembly_using-the-validation-framework[検証フレームワークの使用]

3.7. 検証フレームワークのログ形式

検証または検証グループの実行後に、director は各検証の JSON 形式のログを **/var/logs/validations** ディレクトリーに保存します。ファイルを手動で表示するか、**validation history get --full** コマンドを使用して、特定の検証 UUID のログを表示できます。

それぞれの検証ログファイルは、特定の形式に従います。

- **<UUID>_<Name>_<Time>**

  UUID

  検証用の Ansible UUID

名前
検証の Ansible 名

Time
検証を実行した時の開始日時

それぞれの検証ログには、3 つの主要部分が含まれます。

- plays
- stats
- validation_output

plays

plays セクションには、検証の一部として director が実行したタスクに関する情報が含まれます。

play
プレイはタスクのグループです。それぞれの **play** セクションには、開始/終了時刻、期間、プレイのホストグループ、ならびに検証 ID およびパス等の、特定のタスクグループに関する情報が含まれます。

tasks
検証を行うために director が実行する個別の Ansible タスク。それぞれの **tasks** セクションには **hosts** セクションが含まれ、それぞれの個別ホストで生じたアクションおよびアクションの実行結果が含まれます。**tasks** セクションには **task** セクションも含まれ、これにはタスクの期間が含まれます。

stats

stats セクションには、各ホストで実施した全タスクの結果の基本概要 (成功したタスクおよび失敗したタスク等) が含まれます。

validation_output

検証中にいずれかのタスクが失敗したか、警告メッセージが発行された場合、**validation_output** にその失敗または警告の出力が含まれます。

3.8. 検証フレームワークのログ出力形式

検証フレームワークのデフォルト動作では、検証ログを JSON 形式で保存します。ログの出力は、**ANSIBLE_STDOUT_CALLBACK** 環境変数を使用して変更できます。

検証の出力ログ形式を変更するには、検証を実行し、**--extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback>** オプションを追加します。

```
$ validation run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=<callback> --validation check-ram
```

- **<callback>** を Ansible 出力コールバックに置き換えます。標準的な Ansible 出力コールバックのリストを表示するには、以下のコマンドを実行します。

```
$ ansible-doc -t callback -l
```

検証フレームワークには、以下の追加のコールバックが含まれます。

validation_json

フレームワークは、JSON 形式の検証結果をログファイルとして **/var/logs/validations** に保存します。これは、検証フレームワークのデフォルトコールバックです。

validation_stdout

フレームワークは、画面に JSON 形式の検証結果を表示します。

http_json

フレームワークは、JSON 形式の検証結果を外部ロギングサーバーに送信します。このコールバック用に、追加の環境変数も追加する必要があります。

HTTP_JSON_SERVER

外部サーバーの URL

HTTP_JSON_PORT

外部サーバーの API エントリーポイントのポート。デフォルトポートは 8989 です。

追加の **--extra-env-vars** オプションを使用して、これらの環境変数を設定します。

```
$ validation run --extra-env-vars ANSIBLE_STDOUT_CALLBACK=http_json \
    --extra-env-vars HTTP_JSON_SERVER=http://logserver.example.com \
    --extra-env-vars HTTP_JSON_PORT=8989 \
    --validation check-ram
```

重要

http_json コールバックを使用する前に、**ansible.cfg** ファイルの **callback_whitelist** パラメーターに **http_json** を追加する必要があります。

```
callback_whitelist = http_json
```

3.9. インフライト検証

Red Hat OpenStack Platform (RHOSP) では、コンポーザブルサービスのテンプレートに、インフライト検証が含まれています。インフライト検証により、オーバークラウドのデプロイメントプロセスの主要ステップにおけるサービスの動作ステータスを確認します。

インフライト検証は、デプロイメントプロセスの一部として自動的に実行されます。一部のインフライト検証は、**openstack-tripleo-validations** パッケージからのロールも使用します。

第4章 その他のイントロスペクション操作

状況によっては、標準のオーバークラウドデプロイメントワークフローの外部でイントロスペクションを実行したい場合があります。たとえば、既存の未使用ノードのハードウェアを交換した後、新しいノードをイントロスペクトしたり、イントロスペクションデータを更新したりすることができます。

4.1. ノードイントロスペクションの個別実行

available の状態のノードで個別にイントロスペクションを実行するには、ノードを管理モードに設定して、イントロスペクションを実行します。

手順

1. すべてのノードを **manageable** 状態に設定します。

```
(undercloud) $ openstack baremetal node manage [NODE UUID]
```

2. イントロスペクションを実行します。

```
(undercloud) $ openstack overcloud node introspect [NODE UUID] --provide
```

イントロスペクションが完了すると、ノードの状態が **available** に変わります。

4.2. 初回のイントロスペクション後のノードイントロスペクションの実行

--provide オプションを指定したので、初回のイントロスペクションの後には、全ノードが **available** の状態になります。最初のイントロスペクションの後にすべてのノードでイントロスペクションを実行するには、ノードを管理モードに設定してイントロスペクションを実行します。

手順

1. すべてのノードを **manageable** 状態に設定します

```
(undercloud) $ for node in $(openstack baremetal node list --fields uuid -f value) ; do
openstack baremetal node manage $node ; done
```

2. bulk introspection コマンドを実行します。

```
(undercloud) $ openstack overcloud node introspect --all-manageable --provide
```

イントロスペクション完了後には、すべてのノードが **available** の状態に変わります。

4.3. ネットワークイントロスペクションの実行によるインターフェイス情報の取得

ネットワークイントロスペクションにより、Link Layer Discovery Protocol (LLDP) データがネットワークスイッチから取得されます。以下のコマンドにより、ノード上の全インターフェイスに関する LLDP 情報のサブセット、または特定のノードおよびインターフェイスに関するすべての情報が表示されます。この情報は、トラブルシューティングに役立ちます。director では、デフォルトで LLDP データ収集が有効になっています。

手順

1. ノード上のインターフェイスをリスト表示するには、以下のコマンドを実行します。

```
(undercloud) $ openstack baremetal introspection interface list [NODE UUID]
```

以下に例を示します。

```
(undercloud) $ openstack baremetal introspection interface list c89397b7-a326-41a0-907d-79f8b86c7cd9
+-----------+-------------------+------------------------+-------------------+----------------+
| Interface | MAC Address       | Switch Port VLAN IDs   | Switch Chassis ID | Switch Port ID |
+-----------+-------------------+------------------------+-------------------+----------------+
| p2p2      | 00:0a:f7:79:93:19 | [103, 102, 18, 20, 42] | 64:64:9b:31:12:00 | 510            |
| p2p1      | 00:0a:f7:79:93:18 | [101]                  | 64:64:9b:31:12:00 | 507            |
| em1       | c8:1f:66:c7:e8:2f | [162]                  | 08:81:f4:a6:b3:80 | 515            |
| em2       | c8:1f:66:c7:e8:30 | [182, 183]             | 08:81:f4:a6:b3:80 | 559            |
+-----------+-------------------+------------------------+-------------------+----------------+
```

2. インターフェイスのデータおよびスイッチポートの情報を表示するには、以下のコマンドを実行します。

```
(undercloud) $ openstack baremetal introspection interface show [NODE UUID] [INTERFACE]
```

以下に例を示します。

```
(undercloud) $ openstack baremetal introspection interface show c89397b7-a326-41a0-907d-79f8b86c7cd9 p2p1
+--------------------------------------+------------------------------------------------------------------------------------------------+
| Field                                | Value                                                                                          |
+--------------------------------------+------------------------------------------------------------------------------------------------+
| interface                            | p2p1                                                                                           |
| mac                                  | 00:0a:f7:79:93:18                                                                              |
| node_ident                           | c89397b7-a326-41a0-907d-79f8b86c7cd9                                                           |
| switch_capabilities_enabled          | [u'Bridge', u'Router']                                                                         |
| switch_capabilities_support          | [u'Bridge', u'Router']                                                                         |
| switch_chassis_id                    | 64:64:9b:31:12:00                                                                              |
| switch_port_autonegotiation_enabled  | True                                                                                           |
| switch_port_autonegotiation_support  | True                                                                                           |
| switch_port_description              | ge-0/0/2.0                                                                                     |
| switch_port_id                       | 507                                                                                            |
```

```
| switch_port_link_aggregation_enabled | False
|
| switch_port_link_aggregation_id      | 0
|
| switch_port_link_aggregation_support | True
|
| switch_port_management_vlan_id       | None
|
| switch_port_mau_type                 | Unknown
|
| switch_port_mtu                      | 1514
|
| switch_port_physical_capabilities    | [u'1000BASE-T fdx', u'100BASE-TX fdx', u'100BASE-
TX hdx', u'10BASE-T fdx', u'10BASE-T hdx', u'Asym and Sym PAUSE fdx'] |
| switch_port_protocol_vlan_enabled    | None
|
| switch_port_protocol_vlan_ids        | None
|
| switch_port_protocol_vlan_support    | None
|
| switch_port_untagged_vlan_id         | 101
|
| switch_port_vlan_ids                 | [101]
|
| switch_port_vlans                    | [{u'name': u'RHOS13-PXE', u'id': 101}]
|
| switch_protocol_identities           | None
|
| switch_system_name                   | rhos-compute-node-sw1
|
+--------------------------------------+----------------------------------------------------------------------
-----------------------------------------------+
```

4.4. ハードウェアイントロスペクション情報の取得

Bare Metal サービスでは、オーバークラウド設定の追加ハードウェア情報を取得する機能がデフォルトで有効です。**undercloud.conf** ファイル内の **Inspection_extras** パラメーターの詳細は、Director 設定パラメーター を参照してください。

たとえば、**numa_topology** コレクターは、追加ハードウェアイントロスペクションの一部で、各 NUMA ノードに関する以下の情報が含まれます。

- RAM (キロバイト単位)
- 物理 CPU コアおよびそのシブリングスレッド
- NUMA ノードに関連付けられた NIC

手順

- 上記の情報を取得するには、<UUID> をベアメタルノードの UUID に置き換えて、以下のコマンドを実行します。

```
# openstack baremetal introspection data save <UUID> | jq .numa_topology
```

取得されるベアメタルノードの NUMA 情報の例を以下に示します。


```
{
  "cpus": [
    {
      "cpu": 1,
      "thread_siblings": [
        1,
        17
      ],
      "numa_node": 0
    },
    {
      "cpu": 2,
      "thread_siblings": [
        10,
        26
      ],
      "numa_node": 1
    },
    {
      "cpu": 0,
      "thread_siblings": [
        0,
        16
      ],
      "numa_node": 0
    },
    {
      "cpu": 5,
      "thread_siblings": [
        13,
        29
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        15,
        31
      ],
      "numa_node": 1
    },
    {
      "cpu": 7,
      "thread_siblings": [
        7,
        23
      ],
      "numa_node": 0
    },
    {
      "cpu": 1,
      "thread_siblings": [
        9,
```

```
      25
    ],
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      6,
      22
    ],
    "numa_node": 0
  },
  {
    "cpu": 3,
    "thread_siblings": [
      11,
      27
    ],
    "numa_node": 1
  },
  {
    "cpu": 5,
    "thread_siblings": [
      5,
      21
    ],
    "numa_node": 0
  },
  {
    "cpu": 4,
    "thread_siblings": [
      12,
      28
    ],
    "numa_node": 1
  },
  {
    "cpu": 4,
    "thread_siblings": [
      4,
      20
    ],
    "numa_node": 0
  },
  {
    "cpu": 0,
    "thread_siblings": [
      8,
      24
    ],
    "numa_node": 1
  },
  {
    "cpu": 6,
    "thread_siblings": [
      14,
```

```
      30
     ],
     "numa_node": 1
    },
    {
     "cpu": 3,
     "thread_siblings": [
      3,
      19
     ],
     "numa_node": 0
    },
    {
     "cpu": 2,
     "thread_siblings": [
      2,
      18
     ],
     "numa_node": 0
    }
  ],
  "ram": [
    {
     "size_kb": 66980172,
     "numa_node": 0
    },
    {
     "size_kb": 67108864,
     "numa_node": 1
    }
  ],
  "nics": [
    {
     "name": "ens3f1",
     "numa_node": 1
    },
    {
     "name": "ens3f0",
     "numa_node": 1
    },
    {
     "name": "ens2f0",
     "numa_node": 0
    },
    {
     "name": "ens2f1",
     "numa_node": 0
    },
    {
     "name": "ens1f1",
     "numa_node": 0
    },
    {
     "name": "ens1f0",
     "numa_node": 0
    },
```

```
    {
      "name": "eno4",
      "numa_node": 0
    },
    {
      "name": "eno1",
      "numa_node": 0
    },
    {
      "name": "eno3",
      "numa_node": 0
    },
    {
      "name": "eno2",
      "numa_node": 0
    }
  ]
}
```

第5章 ベアメタルノードの自動検出

自動検出を使用すると、オーバークラウドノードを登録してそのメタデータを生成するのに、**instackenv.json** ファイルを作成する必要がありません。この改善は、ノードに関する情報を取得するのに費す時間を短縮するのに役立ちます。たとえば、自動検出を使用する場合、IPMI IP アドレスを照合し、その後に **instackenv.json** を作成する必要がありません。

5.1. 自動検出の有効化

Bare Metal 自動検出を有効にして設定し、PXE でブートするときにプロビジョニングネットワークに参加するノードを自動的に検出してインポートします。

手順

1. **undercloud.conf** ファイルで、ベアメタルの自動検出を有効にします。

```
enable_node_discovery = True
discovery_default_driver = ipmi
```

   - **enable_node_discovery**: 有効にすると、PXE を使用して introspection ramdisk をブートするすべてのノードが、自動的に Bare Metal サービス (ironic) に登録されます。
   - **discovery_default_driver**: 検出されたノードに使用するドライバーを設定します。例: **ipmi**

2. IPMI の認証情報を ironic に追加します。
   a. IPMI の認証情報を **ipmi-credentials.json** という名前のファイルに追加します。この例の **SampleUsername**、**RedactedSecurePassword**、および **bmc_address** の値を、実際の環境に応じて置き換えてください。

```
[
  {
    "description": "Set default IPMI credentials",
    "conditions": [
      {"op": "eq", "field": "data://auto_discovered", "value": true}
    ],
    "actions": [
      {"action": "set-attribute", "path": "driver_info/ipmi_username",
       "value": "SampleUsername"},
      {"action": "set-attribute", "path": "driver_info/ipmi_password",
       "value": "RedactedSecurePassword"},
      {"action": "set-attribute", "path": "driver_info/ipmi_address",
       "value": "{data[inventory][bmc_address]}"}
    ]
  }
]
```

3. IPMI の認証情報ファイルを ironic にインポートします。

```
$ openstack baremetal introspection rule import ipmi-credentials.json
```

5.2. 自動検出のテスト

PXE は、プロビジョニングネットワークに接続されているノードを起動して、Bare Metal 自動検出機能をテストします。

手順

1. 必要なノードの電源をオンにします。
2. **openstack baremetal node list** コマンドを実行します。新しいノードが **enroll** の状態でリストに表示されるはずです。

```
$ openstack baremetal node list
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| UUID                                 | Name | Instance UUID | Power State | Provisioning State |
Maintenance |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
| c6e63aec-e5ba-4d63-8d37-bd57628258e8 | None | None          | power off   | enroll             |
False       |
| 0362b7b2-5b9c-4113-92e1-0b34a2535d9b | None | None          | power off   | enroll             |
False       |
+--------------------------------------+------+--------------+-------------+--------------------+-------------
-+
```

3. 各ノードにリソースクラスを設定します。

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node set $NODE --resource-class baremetal ; done
```

4. 各ノードにカーネルと ramdisk を設定します。

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node manage $NODE ; done
$ openstack overcloud node configure --all-manageable
```

5. 全ノードを利用可能な状態に設定します。

```
$ for NODE in `openstack baremetal node list -c UUID -f value` ; do openstack baremetal
node provide $NODE ; done
```

5.3. ルールを使用した異なるベンダーハードウェアの検出

異種のハードウェアが混在する環境では、イントロスペクションルールを使用して、認証情報の割り当てやリモート管理を行うことができます。たとえば、DRAC を使用する Dell ノードを処理するには、別の検出ルールが必要になる場合があります。

手順

1. 以下の内容で、**dell-drac-rules.json** という名前のファイルを作成します。

```
[
  {
    "description": "Set default IPMI credentials",
```

```
        "conditions": [
            {"op": "eq", "field": "data://auto_discovered", "value": true},
            {"op": "ne", "field": "data://inventory.system_vendor.manufacturer",
             "value": "Dell Inc."}
        ],
        "actions": [
            {"action": "set-attribute", "path": "driver_info/ipmi_username",
             "value": "SampleUsername"},
            {"action": "set-attribute", "path": "driver_info/ipmi_password",
             "value": "RedactedSecurePassword"},
            {"action": "set-attribute", "path": "driver_info/ipmi_address",
             "value": "{data[inventory][bmc_address]}"}
        ]
    },
    {
        "description": "Set the vendor driver for Dell hardware",
        "conditions": [
            {"op": "eq", "field": "data://auto_discovered", "value": true},
            {"op": "eq", "field": "data://inventory.system_vendor.manufacturer",
             "value": "Dell Inc."}
        ],
        "actions": [
            {"action": "set-attribute", "path": "driver", "value": "idrac"},
            {"action": "set-attribute", "path": "driver_info/drac_username",
             "value": "SampleUsername"},
            {"action": "set-attribute", "path": "driver_info/drac_password",
             "value": "RedactedSecurePassword"},
            {"action": "set-attribute", "path": "driver_info/drac_address",
             "value": "{data[inventory][bmc_address]}"}
        ]
    }
]
```

- この例のユーザー名およびパスワードの値を、実際の環境に応じて置き換えてください。

2. ルールを ironic にインポートします。

```
$ openstack baremetal introspection rule import dell-drac-rules.json
```

第6章 プロファイルの自動タグ付けの設定

イントロスペクションプロセスでは、一連のベンチマークテストを実行します。director は、これらのテストのデータを保存します。このデータをさまざまな方法で使用するポリシーセットを作成することができます。

- ポリシーにより、パフォーマンスの低いノードまたは不安定なノードを特定して、これらのノードがオーバークラウドで使用されないように隔離することができます。
- ポリシーにより、ノードを自動的に特定のプロファイルにタグ付けするかどうかを定義することができます。

6.1. ポリシーファイルの構文

ポリシーファイルは JSON 形式で、ルールセットが記載されます。各ルールでは、説明、条件、およびアクションが定義されます。**説明** はプレーンテキストで記述されたルールの説明で、**条件** はキー/値のパターンを使用して評価を定義し、**アクション** は条件のパフォーマンスを表します。

説明

これは、プレーンテキストで記述されたルールの説明です。

例:

```
"description": "A new rule for my node tagging policy"
```

conditions

ここでは、以下のキー/値のパターンを使用して評価を定義します。

field

評価するフィールドを定義します。

- **memory_mb**: ノードのメモリーサイズ (MB 単位)
- **cpus**: ノードの CPU の合計スレッド数
- **cpu_arch**: ノードの CPU のアーキテクチャー
- **local_gb**: ノードのルートディスクの合計ストレージ容量

op

評価に使用する演算を定義します。これには、以下の属性が含まれます。

- **eq**: 等しい
- **ne**: 等しくない
- **lt**: より小さい
- **gt**: より大きい
- **le**: より小さいか等しい
- **ge**: より大きいか等しい

- **in-net**: IP アドレスが指定のネットワーク内にあることを確認します。
- **matches**: 指定の正規表現と完全に一致する必要があります。
- **contains**: 値には、指定の正規表現が含まれる必要があります。
- **is-empty**: **field** が空欄であることを確認します。

invert
: 評価の結果をインバージョン (反転) するかどうかを定義するブール値

multiple
: 複数の結果が存在する場合に、使用する評価を定義します。このパラメーターには以下の属性が含まれます。

  - **any**: いずれかの結果が一致する必要があります。
  - **all**: すべての結果が一致する必要があります。
  - **first**: 最初の結果が一致する必要があります。

value
: 評価する値を定義します。フィールド、演算および値の条件が満たされる場合には、true の結果を返します。そうでない場合には、条件は false の結果を返します。

例:

```
"conditions": [
  {
    "field": "local_gb",
    "op": "ge",
    "value": 1024
  }
],
```

アクション

条件が **true** の場合には、ポリシーはアクションを実行します。アクションでは、**action** キーおよび **action** の値に応じて追加のキーが使用されます。

- **fail**: イントロスペクションが失敗します。失敗のメッセージには、**message** パラメーターが必要です。
- **set-attribute**: ironic ノードの属性を設定します。ironic の属性へのパス (例: **/driver_info/ipmi_address**) を指定する **path** フィールドおよび設定する **value** が必要です。
- **set-capability**: ironic ノードのケイパビリティーを設定します。新しいケイパビリティーの名前と値を指定する **name** および **value** フィールドが必要です。これにより、このケイパビリティーの既存の値が置き換えられます。たとえば、これを使用してノードのプロファイルを定義します。
- **extend-attribute**: **set-attribute** と同じですが、既存の値をリストとして扱い、そのリストに値を追記します。オプションの **unique** パラメーターを True に設定すると、対象の値がすでにリストに含まれている場合には何も追加しません。

例:

```
"actions": [
  {
    "action": "set-capability",
    "name": "profile",
    "value": "swift-storage"
  }
]
```

6.2. ポリシーファイルの例

イントロスペクションルールを記載した JSON ファイル (**rules.json**) の例を以下に示します。

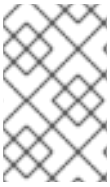
```
[
  {
    "description": "Fail introspection for unexpected nodes",
    "conditions": [
      {
        "op": "lt",
        "field": "memory_mb",
        "value": 4096
      }
    ],
    "actions": [
      {
        "action": "fail",
        "message": "Memory too low, expected at least 4 GiB"
      }
    ]
  },
  {
    "description": "Assign profile for object storage",
    "conditions": [
      {
        "op": "ge",
        "field": "local_gb",
        "value": 1024
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "profile",
        "value": "swift-storage"
      }
    ]
  },
  {
    "description": "Assign possible profiles for compute and controller",
    "conditions": [
      {
        "op": "lt",
        "field": "local_gb",
        "value": 1024
      },
      {
```

```
        "op": "ge",
        "field": "local_gb",
        "value": 40
      }
    ],
    "actions": [
      {
        "action": "set-capability",
        "name": "compute_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "control_profile",
        "value": "1"
      },
      {
        "action": "set-capability",
        "name": "profile",
        "value": null
      }
    ]
  }
]
```

上記の例は、3 つのルールで設定されています。

- メモリーが 4096 MiB 未満の場合には、イントロスペクションが失敗します。クラウドから特定のノードを除外する場合は、このルール種別を適用することができます。
- ハードドライブのサイズが 1 TiB 以上のノードの場合は swift-storage プロファイルが無条件で割り当てられます。
- ハードドライブが 1 TiB 未満だが 40 GiB を超えているノードは、Compute ノードまたは Controller ノードのいずれかに割り当てることができます。**openstack overcloud profiles match** コマンドを使用して後で最終選択できるように、2 つのケイパビリティー (**compute_profile** および **control_profile**) を割り当てています。このプロセスが機能するためには、既存のプロファイルケイパビリティーを削除する必要があります。削除しないと、既存のプロファイルケイパビリティーが優先されます。

プロファイルマッチングルールは、他のノードを変更しません。



注記

イントロスペクションルールを使用して **profile** 機能を割り当てる場合は常に、既存の値よりこの割り当てた値が優先されます。ただし、すでにプロファイルケイパビリティーを持つノードについては、**[PROFILE]_profile** ケイパビリティーは無視されます。

6.3. ポリシーファイルを DIRECTOR にインポートする

ポリシー JSON ファイルで定義したポリシールールを適用するには、ポリシーファイルを director にインポートする必要があります。

手順

1. ポリシーファイルを director にインポートします。

```
$ openstack baremetal introspection rule import <policy_file>
```

   - **<policy_file>** をポリシールールファイルの名前 (例: **rules.json**) に置き換えます。

2. イントロスペクションのプロセスを実行します。

```
$ openstack overcloud node introspect --all-manageable
```

3. ポリシールールが適用されるノードの UUID を取得します。

```
$ openstack baremetal node list
```

4. ポリシールールファイルで定義されたプロファイルがノードに割り当てられていることを確認します。

```
$ openstack baremetal node show <node_uuid>
```

5. イントロスペクションルールを間違えた場合は、すべてのルールを削除します。

```
$ openstack baremetal introspection rule purge
```

第7章 コンテナーイメージのカスタマイズ

Red Hat OpenStack Platform (RHOSP) サービスはコンテナーで実行されるため、RHOSP サービスをデプロイするにはコンテナーイメージを取得する必要があります。RHOSP デプロイメント用のコンテナーイメージを準備する環境ファイルを生成およびカスタマイズできます。

7.1. DIRECTOR インストール用のコンテナーイメージの準備

Red Hat では、オーバークラウド用コンテナーイメージの管理に関して、以下の方法をサポートしています。

- コンテナーイメージを Red Hat Container Catalog からアンダークラウド上の **image-serve** レジストリーにプルし、続いてそのイメージを **image-serve** レジストリーからプルする。イメージをアンダークラウドにプルする際には、複数のオーバークラウドノードが外部接続を通じて同時にコンテナーイメージをプルする状況を避けてください。

- Satellite 6 サーバーからコンテナーイメージをプルする。ネットワークトラフィックは内部になるため、これらのイメージを Satellite から直接プルすることができます。

アンダークラウドのインストールには、コンテナーイメージの取得先およびその保存方法を定義するための環境ファイルが必要です。director のインストールの準備時に、デフォルトのコンテナーイメージ準備ファイルを生成します。デフォルトのコンテナーイメージ準備ファイルをカスタマイズできます。

7.1.1. コンテナーイメージ準備のパラメーター

コンテナー準備用のデフォルトファイル (**containers-prepare-parameter.yaml**) には、**ContainerImagePrepare** heat パラメーターが含まれます。このパラメーターで、イメージのセットを準備するためのさまざまな設定を定義します。

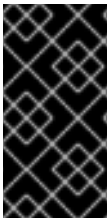
```
parameter_defaults:
  ContainerImagePrepare:
  - (strategy one)
  - (strategy two)
  - (strategy three)
  ...
```

それぞれの設定では、サブパラメーターのセットにより使用するイメージやイメージの使用方法を定義することができます。以下の表には、**ContainerImagePrepare** ストラテジーの各設定で使用することのできるサブパラメーターの情報をまとめています。

パラメーター	説明
excludes	設定からイメージ名を除外する正規表現のリスト
includes	設定に含める正規表現のリスト。少なくとも 1 つのイメージ名が既存のイメージと一致している必要があります。**includes** パラメーターを指定すると、**excludes** の設定はすべて無視されます。

パラメーター	説明
modify_append_tag	対象となるイメージのタグに追加する文字列。たとえば、17.1.0-5.161 のタグが付けられたイメージをプルし、**modify_append_tag** を **-hotfix** に設定すると、director は最終イメージを 17.1.0-5.161-hotfix とタグ付けします。
modify_only_with_labels	変更するイメージを絞り込むイメージラベルのディクショナリー。イメージが定義したラベルと一致する場合には、director はそのイメージを変更プロセスに含めます。
modify_role	イメージのアップロード中 (ただし目的のレジストリーにプッシュする前) に実行する Ansible ロール名の文字列
modify_vars	**modify_role** に渡す変数のディクショナリー
push_destination	アップロードプロセス中にイメージをプッシュするレジストリーの名前空間を定義します。 ● **true** に設定すると、**push_destination** はホスト名を使用してアンダークラウドレジストリーの名前空間に設定されます。これが推奨される方法です。 ● **false** に設定すると、ローカルレジストリーへのプッシュは実行されず、ノードはソースから直接イメージをプルします。 ● カスタムの値に設定すると、director はイメージを外部のローカルレジストリーにプッシュします。 実稼働環境でこのパラメーターを **false** に設定した場合、イメージを Red Hat Container Catalog から直接プルする際に、すべてのオーバークラウドノードが同時に外部接続を通じて Red Hat Container Catalog からイメージをプルするため、帯域幅の問題が発生する可能性があります。コンテナーイメージをホストする Red Hat Satellite Server から直接プルする場合にのみ、**false** を使用します。 **push_destination** パラメーターが **false** に設定されているか、定義されておらずリモートレジストリーで認証が必要な場合は、**ContainerImageRegistryLogin** パラメーターを **true** に設定し、**ContainerImageRegistryCredentials** パラメーターで認証情報を追加します。
pull_source	元のコンテナーイメージをプルするソースレジストリー

パラメーター	説明
set	初期イメージの取得場所を定義する、**キー: 値** 定義のディクショナリー
tag_from_label	指定したコンテナーイメージのメタデータラベルの値を使用して、すべてのイメージのタグを作成し、そのタグが付けられたイメージをプルします。たとえば、**tag_from_label: {version}-{release}** を設定すると、director は **version** および **release** ラベルを使用して新しいタグを作成します。あるコンテナーについて、**version** を 17.1.0 に設定し、**release** を **5.161** に設定した場合、タグは 17.1.0-5.161 となります。**set** ディクショナリーで **tag** を定義していない場合に限り、director はこのパラメーターを使用します。



重要

イメージをアンダークラウドにプッシュする場合は、**push_destination: UNDERCLOUD_IP:PORT** の代わりに **push_destination: true** を使用します。**push_destination: true** 手法を使用することで、IPv4 アドレスおよび IPv6 アドレスの両方で一貫性が確保されます。

set パラメーターには、複数の **キー: 値** 定義を設定することができます。

キー	説明
ceph_image	Ceph Storage コンテナーイメージの名前
ceph_namespace	Ceph Storage コンテナーイメージの名前空間
ceph_tag	Ceph Storage コンテナーイメージのタグ
ceph_alertmanager_image **ceph_alertmanager_namespace** **ceph_alertmanager_tag**	Ceph Storage Alert Manager コンテナーイメージの名前、namespace、およびタグ。
ceph_grafana_image **ceph_grafana_namespace** **ceph_grafana_tag**	Ceph Storage Grafana コンテナーイメージの名前、namespace、およびタグ。

キー	説明
ceph_node_exporter_image **ceph_node_exporter_namespace** **ceph_node_exporter_tag**	Ceph Storage Node Exporter コンテナーイメージの名前、namespace、およびタグ。
ceph_prometheus_image **ceph_prometheus_namespace** **ceph_prometheus_tag**	Ceph Storage Prometheus コンテナーイメージの名前、namespace、およびタグ。
name_prefix	各 OpenStack サービスイメージの接頭辞
name_suffix	各 OpenStack サービスイメージの接尾辞
namespace	各 OpenStack サービスイメージの名前空間
neutron_driver	使用する OpenStack Networking (neutron) コンテナーを定義するのに使用するドライバー。標準の **neutron-server** コンテナーに設定するには、null 値を使用します。OVN ベースのコンテナーを使用するには、**ovn** に設定します。
tag	ソースからの全イメージに特定のタグを設定します。定義されていない場合は、director は Red Hat OpenStack Platform のバージョン番号をデフォルト値として使用します。このパラメーターは、**tag_from_label** の値よりも優先されます。

注記

コンテナーイメージでは、Red Hat OpenStack Platform のバージョンに基づいたマルチストリームタグが使用されます。したがって、今後 **latest** タグは使用されません。

7.1.2. コンテナーイメージタグ付けのガイドライン

Red Hat コンテナーレジストリーでは、すべての Red Hat OpenStack Platform コンテナーイメージをタグ付けするのに、特定のバージョン形式が使用されます。この形式は、**version-release** のように各コンテナーのラベルのメタデータに従います。

version

Red Hat OpenStack Platform のメジャーおよびマイナーバージョンに対応します。これらのバージョンは、1 つまたは複数のリリースが含まれるストリームとして機能します。

release

バージョンストリーム内の、特定コンテナーイメージバージョンのリリースに対応します。

たとえば、Red Hat OpenStack Platform の最新バージョンが 17.1.0 で、コンテナーイメージのリリースが **5.161** の場合、コンテナーイメージのタグは 17.1.0-5.161 となります。

Red Hat コンテナーレジストリーでは、コンテナーイメージバージョンの最新リリースとリンクするメジャーおよびマイナー **version** タグのセットも使用されます。たとえば、17.1 と 17.1.0 の両方が、17.1.0 コンテナーストリームの最新 **release** とリンクします。17.1 の新規マイナーリリースが公開されると、17.1 タグは新規マイナーリリースストリームの最新 **release** とリンクします。一方、17.1.0 タグは、引き続き 17.1.0 ストリーム内の最新の **release** とリンクします。

ContainerImagePrepare パラメーターには 2 つのサブパラメーターが含まれ、これを使用してダウンロードするコンテナーイメージを定義することができます。これらのサブパラメーターは、**set** ディクショナリー内の **tag** パラメーターおよび **tag_from_label** パラメーターです。以下のガイドラインを使用して、**tag** または **tag_from_label** のどちらを使用するかを判断してください。

- **tag** のデフォルト値は、お使いの OpenStack Platform のメジャーバージョンです。本バージョンでは、17.1 です。これは常に最新のマイナーバージョンおよびリリースに対応します。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 17.1
      ...
```

- OpenStack Platform コンテナーイメージの特定マイナーバージョンに変更するには、タグをマイナーバージョンに設定します。たとえば、17.1.2 に変更するには、**tag** を 17.1.2 に設定します。

```
parameter_defaults:
  ContainerImagePrepare:
  - set:
      ...
      tag: 17.1.2
      ...
```

- **tag** を設定すると、インストールおよび更新時に、director は必ず **tag** で設定したバージョンの最新のコンテナーイメージ **release** をダウンロードします。

- **tag** を設定しないと、director は最新のメジャーバージョンと共に **tag_from_label** の値を使用します。

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

- **tag_from_label** パラメーターは、Red Hat コンテナーレジストリーから検査する最新コンテナーイメージリリースのラベルメタデータからタグを生成します。たとえば、特定のコンテナーのラベルは、以下の **version** および **release** メタデータを使用します。

```
  "Labels": {
    "release": "5.161",
    "version": "17.1.0",
    ...
  }
```

- **tag_from_label** のデフォルト値は **{version}-{release}** です。これは、各コンテナーイメージのバージョンおよびリリースのメタデータラベルに対応します。たとえば、コンテナーイメージの **version** に 17.1.0 が、 **release** に 5.161 が、それぞれ設定されている場合、コンテナーイメージのタグは 17.1.0-5.161 となります。

- **tag** パラメーターは、常に **tag_from_label** パラメーターよりも優先されます。**tag_from_label** を使用するには、コンテナー準備の設定で **tag** パラメーターを省略します。

- **tag** および **tag_from_label** の主な相違点は、次のとおりです。director が **tag** を使用してイメージをプルする場合は、Red Hat コンテナーレジストリーがバージョンストリーム内の最新イメージリリースとリンクするメジャーまたはマイナーバージョンのタグだけに基づきます。一方、**tag_from_label** を使用する場合は、director がタグを生成して対応するイメージをプルできるように、各コンテナーイメージのメタデータの検査を行います。

7.1.3. Ceph Storage コンテナーイメージの除外

デフォルトのオーバークラウドロール設定では、デフォルトの Controller ロール、Compute ロール、および Ceph Storage ロールが使用されます。ただし、デフォルトのロール設定を使用して Ceph Storage ノードを持たないオーバークラウドをデプロイする場合、director は引き続き Ceph Storage コンテナーイメージを Red Hat コンテナーレジストリーからプルします。イメージがデフォルト設定の一部として含まれているためです。

オーバークラウドで Ceph Storage コンテナーが必要ない場合は、Red Hat コンテナーレジストリーから Ceph Storage コンテナーイメージをプルしないように director を設定することができます。

手順

1. **containers-prepare-parameter.yaml** ファイルを編集し、**ceph_images: false** パラメーターを追加します。
   このファイルの例を以下に示します。上記のパラメーターを太字で示しています。

```
parameter_defaults:
  ContainerImagePrepare:
  - tag_from_label: {version}-{release}
    set:
      name_prefix: rhosp17-openstack-
      name_suffix: ''
      tag: 17.1_20231214.1
      rhel_containers: false
      neutron_driver: ovn
      ceph_images: false
    push_destination: true
```

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

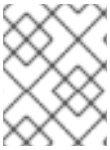
3. オーバークラウドのデプロイに使用する新しいコンテナーイメージファイルを作成します。
   sudo openstack tripleo container image prepare -e containers-prepare-parameter.yaml --output-env-file <new_container_images_file>

   - **<new_container_images_file>** は、新しいパラメーターを含む出力ファイルに置き換えます。

4. 新しいコンテナーイメージファイルをオーバークラウドデプロイメント環境ファイルのリストに追加します。

7.1.4. 準備プロセスにおけるイメージの変更

イメージの準備中にイメージを変更し、変更したそのイメージで直ちにオーバークラウドをデプロイすることが可能です。



注記

Red Hat OpenStack Platform (RHOSP) ディレクターは、Ceph コンテナーではなく、RHOSP コンテナーの準備中にイメージを変更することをサポートします。

イメージを変更するシナリオを以下に示します。

- デプロイメント前にテスト中の修正でイメージが変更される、継続的インテグレーションのパイプラインの一部として。
- ローカルの変更をテストおよび開発のためにデプロイしなければならない、開発ワークフローの一部として。
- 変更をデプロイしなければならないが、イメージビルドパイプラインでは利用することができない場合。たとえば、プロプライエタリーアドオンの追加または緊急の修正など。

準備プロセス中にイメージを変更するには、変更する各イメージで Ansible ロールを呼び出します。ロールはソースイメージを取得して必要な変更を行い、その結果をタグ付けします。prepare コマンドでイメージを目的のレジストリーにプッシュし、変更したイメージを参照するように heat パラメーターを設定することができます。

Ansible ロール **tripleo-modify-image** は要求されたロールインターフェイスに従い、変更のユースケースに必要な処理を行います。**ContainerImagePrepare** パラメーターの変更固有のキーを使用して、変更をコントロールします。

- **modify_role** では、変更する各イメージについて呼び出す Ansible ロールを指定します。
- **modify_append_tag** は、ソースイメージタグの最後に文字列を追加します。これにより、そのイメージが変更されていることが明確になります。すでに **push_destination** レジストリーに変更されたイメージが含まれている場合には、このパラメーターを使用して変更を省略します。イメージを変更する場合には、必ず **modify_append_tag** を変更します。
- **modify_vars** は、ロールに渡す Ansible 変数のディクショナリーです。

tripleo-modify-image ロールが処理するユースケースを選択するには、**tasks_from** 変数をそのロールで必要なファイルに設定します。

イメージを変更する **ContainerImagePrepare** エントリーを開発およびテストする場合には、イメージが想定どおりに変更されることを確認するために、追加のオプションを指定せずにイメージの準備コマンドを実行します。

```
sudo openstack tripleo container image prepare \
  -e ~/containers-prepare-parameter.yaml
```

重要

openstack tripleo container image prepare コマンドを使用するには、アンダークラウドに実行中の **image-serve** レジストリーが含まれている必要があります。したがって、**image-serve** レジストリーがインストールされないため、新しいアンダークラウドのインストール前にこのコマンドを実行することはできません。アンダークラウドが正常にインストールされた後に、このコマンドを実行することができます。

7.1.5. コンテナーイメージの既存パッケージの更新

Red Hat OpenStack Platform (RHOSP) コンテナーのコンテナーイメージ上の既存パッケージを更新できます。

注記

Red Hat OpenStack Platform (RHOSP) ディレクターは、Ceph コンテナーではなく、RHOSP コンテナーのコンテナーイメージ上の既存のパッケージの更新をサポートします。

手順

1. コンテナーイメージにインストールするための RPM パッケージをダウンロードします。
2. **containers-prepare-parameter.yaml** ファイルを編集して、コンテナーイメージ上のすべてのパッケージを更新します。

```
ContainerImagePrepare:
- push_destination: true
  ...
  modify_role: tripleo-modify-image
  modify_append_tag: "-updated"
  modify_vars:
    tasks_from: yum_update.yml
    compare_host_packages: true
    yum_repos_dir_path: /etc/yum.repos.d
  ...
```

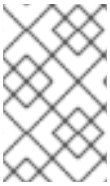
3. **containers-prepare-parameter.yaml** ファイルを保存します。
4. **openstack overclouddeploy** コマンドを実行する際に、**containers-prepare-parameter.yaml** ファイルを含めます。

7.1.6. コンテナーイメージへの追加 RPM ファイルのインストール

RPM ファイルのディレクトリーをコンテナーイメージにインストールすることができます。この機能は、ホットフィックスやローカルパッケージビルドなど、パッケージリポジトリーからは入手できないパッケージのインストールに役立ちます。

注記

Red Hat OpenStack Platform (RHOSP) ディレクターは、Ceph コンテナーではなく、RHOSP コンテナーのコンテナーイメージへの追加の RPM ファイルのインストールをサポートします。



注記

既存のデプロイメントでコンテナーイメージを変更する場合は、変更後にマイナー更新を実行して変更をオーバークラウドに適用する必要があります。詳細は、Red Hat OpenStack Platform のマイナー更新の実行 を参照してください。

手順

- 次の **ContainerImagePrepare** エントリーの例では、いくつかのホットフィックスパッケージを **nova-compute** イメージにのみインストールします。

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
  modify_vars:
    tasks_from: rpm_install.yml
    rpms_path: /home/stack/nova-hotfix-pkgs
  ...
```

7.1.7. カスタム Dockerfile を使用したコンテナーイメージの変更

Dockerfile を含むディレクトリーを指定して、必要な変更を加えることができます。**tripleo-modify-image** ロールを呼び出すと、ロールは **Dockerfile.modified** ファイルを生成し、これにより **FROM** ディレクティブが変更され新たな **LABEL** ディレクティブが追加されます。

注記

Red Hat OpenStack Platform (RHOSP) ディレクターは、Ceph コンテナーではなく、RHOSP コンテナー用のカスタム Dockerfile を使用したコンテナーイメージの変更をサポートします。

手順

1. 以下の例では、**nova-compute** イメージでカスタム Dockerfile が実行されます。

```
ContainerImagePrepare:
- push_destination: true
  ...
  includes:
  - nova-compute
  modify_role: tripleo-modify-image
  modify_append_tag: "-hotfix"
  modify_vars:
    tasks_from: modify_image.yml
    modify_dir_path: /home/stack/nova-custom
  ...
```

2. **/home/stack/nova-custom/Dockerfile** ファイルの例を以下に示します。**USER** root ディレクティブを実行した後は、元のイメージのデフォルトユーザーに戻す必要があります。

```
FROM registry.redhat.io/rhosp-rhel9/openstack-nova-compute:latest

USER "root"

COPY customize.sh /tmp/
RUN /tmp/customize.sh

USER "nova"
```

7.1.8. コンテナーイメージ管理用 Satellite サーバーの準備

Red Hat Satellite 6 には、レジストリーの同期機能が備わっています。これにより、複数のイメージを Satellite Server にプルし、アプリケーションライフサイクルの一環として管理することができます。また、他のコンテナー対応システムも Satellite をレジストリーとして使うことができます。コンテナーイメージ管理についての詳細は、**Red Hat Satellite 6 コンテンツ管理ガイド**の コンテナーイメージの管理 を参照してください。

以下の手順は、Red Hat Satellite 6 の **hammer** コマンドラインツールを使用した例を示しています。組織には、例として **ACME** という名称を使用しています。この組織は、実際に使用する Satellite 6 の組織に置き換えてください。

注記

この手順では、**registry.redhat.io** のコンテナーイメージにアクセスするために認証情報が必要です。Red Hat では、個人のユーザー認証情報を使用する代わりに、レジストリーサービスアカウントを作成し、それらの認証情報を使用して **registry.redhat.io** コンテンツにアクセスすることを推奨します。詳しくは、Red Hat コンテナーレジストリーの認証 を参照してください。

手順

1. すべてのコンテナーイメージのリストを作成します。

```
$ sudo podman search --limit 1000 "registry.redhat.io/rhosp-rhel9" --format="{{ .Name }}" |
sort > satellite_images
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep
<ceph_dashboard_image_file>
$ sudo podman search --limit 1000 "registry.redhat.io/rhceph" | grep <ceph_image_file>
$ sudo podman search --limit 1000 "registry.redhat.io/openshift4" | grep ose-prometheus
```

- **<ceph_dashboard_image_file>** は、デプロイメントで使用する Red Hat Ceph Storage のバージョンのイメージファイルの名前に置き換えます。
  - Red Hat Ceph Storage 5: **rhceph-5-dashboard-rhel8**
  - Red Hat Ceph Storage 6: **rhceph-6-dashboard-rhel9**
- **<ceph_image_file>** は、デプロイメントで使用する Red Hat Ceph Storage のバージョンのイメージファイルの名前に置き換えます。
  - Red Hat Ceph Storage 5: **rhceph-5-rhel8**
  - Red Hat Ceph Storage 6: **rhceph-6-rhel9**

注記

openstack-ovn-bgp-agent イメージは **registry.redhat.io/rhosp-rhel9/openstack-ovn-bgp-agent-rhel9:17.1** にあります。

- Ceph をインストールして Ceph Dashboard を有効にする場合は、次の ose-prometheus コンテナーが必要です。

```
registry.redhat.io/openshift4/ose-prometheus-node-exporter:v4.12
registry.redhat.io/openshift4/ose-prometheus:v4.12
registry.redhat.io/openshift4/ose-prometheus-alertmanager:v4.12
```

2. Satellite 6 の **hammer** ツールがインストールされているシステムに **satellite_images** ファイルをコピーします。あるいは、Hammer CLI ガイド に記載の手順に従って、アンダークラウドに **hammer** ツールをインストールします。

3. 以下の **hammer** コマンドを実行して、実際の Satellite 組織に新規製品 (**OSP Containers**) を作成します。

```
$ hammer product create \
  --organization "ACME" \
  --name "OSP Containers"
```

このカスタム製品に、イメージを保管します。

4. **satellite_images** ファイルからオーバークラウドのコンテナーイメージを追加します。

```
$ while read IMAGE; do \
  IMAGE_NAME=$(echo $IMAGE | cut -d"/" -f3 | sed "s/openstack-//g") ; \
  IMAGE_NOURL=$(echo $IMAGE | sed "s/registry.redhat.io\///g") ; \
  hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name $IMAGE_NOURL \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name $IMAGE_NAME ; done < satellite_images
```

5. Ceph Storage コンテナーイメージを追加します。

```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/<ceph_image_name> \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name <ceph_image_name>
```

- **<ceph_image_file>** は、デプロイメントで使用する Red Hat Ceph Storage のバージョンのイメージファイルの名前に置き換えます。

- Red Hat Ceph Storage 5: **rhceph-5-rhel8**
- Red Hat Ceph Storage 6: **rhceph-6-rhel9**

注記

Ceph ダッシュボードをインストールする場合は、**hammer repository create** コマンドに **--name <ceph_dashboard_image_name>** を含めます。

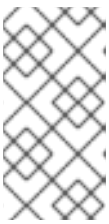
```
$ hammer repository create \
  --organization "ACME" \
  --product "OSP Containers" \
  --content-type docker \
  --url https://registry.redhat.io \
  --docker-upstream-name rhceph/<ceph_dashboard_image_name> \
  --upstream-username USERNAME \
  --upstream-password PASSWORD \
  --name <ceph_dashboard_image_name>
```

- **<ceph_dashboard_image_file>** は、デプロイメントで使用する Red Hat Ceph Storage のバージョンのイメージファイルの名前に置き換えます。
  - Red Hat Ceph Storage 5: **rhceph-5-dashboard-rhel8**
  - Red Hat Ceph Storage 6: **rhceph-6-dashboard-rhel9**

6. コンテナーイメージを同期します。

```
$ hammer product synchronize \
  --organization "ACME" \
  --name "OSP Containers"
```

Satellite Server が同期を完了するまで待ちます。



注記

設定によっては、**hammer** から Satellite Server のユーザー名およびパスワードが要求される場合があります。設定ファイルを使用して自動的にログインするように **hammer** を設定することができます。詳しくは、**Red Hat Satellite Hammer CLI ガイド** の 認証 セクションを参照してください。

7. お使いの Satellite 6 サーバーでコンテンツビューが使われている場合には、新たなバージョンのコンテンツビューを作成してイメージを反映し、アプリケーションライフサイクルの環境に従ってプロモートします。この作業は、アプリケーションライフサイクルをどのように設定するかに大きく依存します。たとえば、ライフサイクルに **production** という名称の環境があり、その環境でコンテナーイメージを利用可能にする場合には、コンテナーイメージを含むコンテンツビューを作成し、そのコンテンツビューを **production** 環境にプロモートします。詳しくは、Red Hat Satellite コンテンツ管理ガイドの コンテンツビューの管理 を参照してください。

8. **base** イメージに使用可能なタグを確認します。

```
$ hammer docker tag list --repository "base" \
  --organization "ACME" \
  --lifecycle-environment "production" \
  --product "OSP Containers"
```

このコマンドにより、特定環境のコンテンツビューでの OpenStack Platform コンテナーイメージのタグが表示されます。

9. アンダークラウドに戻り、Satellite サーバーをソースとして使用して、イメージを準備するデフォルトの環境ファイルを生成します。以下のサンプルコマンドを実行して環境ファイルを生成します。

```
$ sudo openstack tripleo container image prepare default \
  --output-env-file containers-prepare-parameter.yaml
```

- **--output-env-file**: 環境ファイルの名前です。このファイルには、アンダークラウド用コンテナーイメージを準備するためのパラメーターが含まれます。ここでは、ファイル名は **containers-prepare-parameter.yaml** です。

10. **containers-prepare-parameter.yaml** ファイルを編集して以下のパラメーターを変更します。

- **push_destination**: 選択したコンテナーイメージの管理手段に応じて、これを **true** または **false** に設定します。このパラメーターを **false** に設定すると、オーバークラウドノードはイメージを直接 Satellite からプルします。このパラメーターを **true** に設定すると、director はイメージを Satellite からアンダークラウドレジストリーにプルし、オーバークラウドはそのイメージをアンダークラウドレジストリーからプルします。

- **namespace**: Satellite サーバー上のレジストリーの URL。

- **name_prefix**: 接頭辞は Satellite 6 の命名規則に基づきます。これは、コンテンツビューを使用するかどうかによって異なります。

  - コンテンツビューを使用する場合、設定は **[org]-[environment]-[content view]-[product]-** です。例: **acme-production-myosp17-osp_containers-**.

  - コンテンツビューを使用しない場合、設定は **[org]-[product]-** です。例: **acme-osp_containers-**。

- **ceph_namespace**、**ceph_image**、**ceph_tag**: Ceph Storage を使用する場合には、Ceph Storage コンテナーイメージの場所を定義するこれらの追加パラメーターを指定します。**ceph_image** に Satellite 固有の接頭辞が追加された点に注意してください。この接頭辞は、**name_prefix** オプションと同じ値です。

Satellite 固有のパラメーターが含まれる環境ファイルの例を、以下に示します。

```
parameter_defaults:
  ContainerImagePrepare:
  - push_destination: false
    set:
      ceph_image: acme-production-myosp17_1-osp_containers-rhceph-6
      ceph_namespace: satellite.example.com:443
      ceph_tag: latest
      name_prefix: acme-production-myosp17_1-osp_containers-
      name_suffix: ''
      namespace: satellite.example.com:5000
```

```
neutron_driver: null
tag: '17.1'
...
```

注記

Red Hat SatelliteServer に保存されている特定のコンテナーイメージのバージョンを使用するには、**tag** のキーと値のペアを **set** ディクショナリー内の特定のバージョンに設定します。たとえば、17.1.2 イメージストリームを使用するには、**set** ディクショナリーに **tag: 17.1.2** を設定します。

undercloud.conf 設定ファイルで **containers-prepare-parameter.yaml** 環境ファイルを定義する必要があります。定義しないと、アンダークラウドはデフォルト値を使用します。

```
container_images_file = /home/stack/containers-prepare-parameter.yaml
```

7.1.9. ベンダープラグインのデプロイ

一部のサードパーティーハードウェアをブロックストレージのバックエンドとして使用するには、ベンダープラグインをデプロイする必要があります。以下の例で、Dell EMC ハードウェアをブロックストレージのバックエンドとして使用するために、ベンダープラグインをデプロイする方法について説明します。

手順

1. オーバークラウド用に新たなコンテナーイメージファイルを作成します。

```
$ sudo openstack tripleo container image prepare default \
    --local-push-destination \
    --output-env-file containers-prepare-parameter-dellemc.yaml
```

2. **containers-prepare-parameter-dellemc.yaml** ファイルを編集します。

3. メインの Red Hat OpenStack Platform コンテナーイメージの設定に **exclude** パラメーターを追加します。このパラメーターを使用して、ベンダーコンテナーイメージで置き換えるコンテナーイメージを除外します。以下の例では、コンテナーイメージは **cinder-volume** イメージです。

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      excludes:
      - cinder-volume
      set:
        namespace: registry.redhat.io/rhosp-rhel9
        name_prefix: openstack-
        name_suffix: ''
        tag: 17.1
        ...
      tag_from_label: "{version}-{release}"
```

4. **ContainerImagePrepare** パラメーターに、ベンダープラグインの代替コンテナーイメージが含まれる新しい設定を追加します。

```
parameter_defaults:
  ContainerImagePrepare:
    ...
    - push_destination: true
      includes:
        - cinder-volume
      set:
        namespace: registry.connect.redhat.com/dellemc
        name_prefix: openstack-
        name_suffix: -dellemc-rhosp16
        tag: 16.2-2
        ...
```

5. **ContainerImageRegistryCredentials** パラメーターに registry.connect.redhat.com レジストリーの認証情報を追加します。

```
parameter_defaults:
  ContainerImageRegistryCredentials:
    registry.redhat.io:
      [service account username]: [service account password]
    registry.connect.redhat.com:
      [service account username]: [service account password]
```

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

7. **openstack overcloud deploy** などのデプロイメントコマンドに **containers-prepare-parameter-dellemc.yaml** ファイルを追加します。

```
$ openstack overcloud deploy --templates
    ...
    -e containers-prepare-parameter-dellemc.yaml
    ...
```

director がオーバークラウドをデプロイする際に、オーバークラウドは標準のコンテナーイメージの代わりにベンダーのコンテナーイメージを使用します。

重要

containers-prepare-parameter-dellemc.yaml ファイルは、オーバークラウドデプロイメント内の標準の **containers-prepare-parameter.yaml** ファイルを置き換えます。オーバークラウドのデプロイメントに、標準の **containers-prepare-parameter.yaml** ファイルを含めないでください。アンダークラウドのインストールおよび更新には、標準の **containers-prepare-parameter.yaml** ファイルを維持します。

7.2. 高度なコンテナーイメージ管理の実施

デフォルトのコンテナーイメージ設定は、ほとんどの環境に対応します。状況によっては、コンテナーイメージ設定にバージョンの固定などのカスタマイズが必要になる場合があります。

7.2.1. アンダークラウド用コンテナーイメージの固定

特定の状況では、アンダークラウド用に特定のコンテナーイメージバージョンのセットが必要な場合があります。そのような場合には、イメージを特定のバージョンに固定する必要があります。イメージに固定するには、コンテナー設定ファイルを生成および変更し、続いてアンダークラウドのロールデータ

をコンテナー設定ファイルと組み合わせ、サービスとコンテナーイメージのマッピングが含まれる環境ファイルを生成する必要があります。次に、この環境ファイルを **undercloud.conf** ファイルの **custom_env_files** パラメーターに追加します。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **--output-env-file** オプションを指定して **openstack tripleo container image prepare default** コマンドを実行し、デフォルトのイメージ設定が含まれるファイルを生成します。

```
$ sudo openstack tripleo container image prepare default \
--output-env-file undercloud-container-image-prepare.yaml
```

3. 環境の要件に応じて、**undercloud-container-image-prepare.yaml** ファイルを変更します。
   a. **tag:** パラメーターを削除して、director が **tag_from_label:** パラメーターを使用できるようにします。director はこのパラメーターを使用して各コンテナーイメージの最新バージョンを特定し、それぞれのイメージをプルし、director のコンテナーレジストリーの各イメージをタグ付けします。
   b. アンダークラウドの Ceph ラベルを削除します。
   c. **neutron_driver:** パラメーターが空であることを確認します。OVN はアンダークラウドでサポートされないため、このパラメーターを **OVN** に設定しないでください。
   d. コンテナーイメージレジストリーの認証情報を追加します。

```
ContainerImageRegistryCredentials:
  registry.redhat.io:
    myser: 'p@55w0rd!'
```

注記

image-serve レジストリーがまだインストールされていないので、新しいアンダークラウドのアンダークラウドレジストリーにコンテナーイメージをプッシュすることはできません。**push_destination** の値を **false** に設定するか、カスタム値を使用して、イメージを直接ソースからプルする必要があります。詳細は、コンテナーイメージ準備のパラメーター を参照してください。

4. カスタムの **undercloud-container-image-prepare.yaml** ファイルと組み合わせたアンダークラウドのロールファイルを使用する、新たなコンテナーイメージ設定ファイルを作成します。

```
$ sudo openstack tripleo container image prepare \
-r /usr/share/openstack-tripleo-heat-templates/roles_data_undercloud.yaml \
-e undercloud-container-image-prepare.yaml \
--output-env-file undercloud-container-images.yaml
```

undercloud-container-images.yaml ファイルは、サービスパラメーターのコンテナーイメージへのマッピングが含まれる環境ファイルです。たとえば、OpenStack Identity (keystone) は、**ContainerKeystoneImage** パラメーターを使用してそのコンテナーイメージを定義します。

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel9/openstack-keystone:17.1
```

コンテナーイメージタグは **{version}-{release}** 形式に一致することに注意してください。

5. **undercloud.conf** ファイルの **custom_env_files** パラメーターに **undercloud-container-images.yaml** ファイルを追加します。アンダークラウドのインストールを実施する際に、アンダークラウドサービスはこのファイルから固定されたコンテナーイメージのマッピングを使用します。

7.2.2. オーバークラウド用コンテナーイメージのピニング

特定の状況では、オーバークラウド用に特定のコンテナーイメージバージョンのセットが必要な場合があります。そのような場合には、イメージを特定のバージョンに固定する必要があります。イメージに固定するには、**containers-prepare-parameter.yaml** ファイルを作成して、このファイルを使用してアンダークラウドレジストリーにコンテナーイメージをプルし、固定されたイメージのリストが含まれる環境ファイルを生成する必要があります。

たとえば、**containers-prepare-parameter.yaml** ファイルに以下の内容が含まれる場合があります。

```
parameter_defaults:
  ContainerImagePrepare:
    - push_destination: true
      set:
        name_prefix: openstack-
        name_suffix: ''
        namespace: registry.redhat.io/rhosp-rhel9
        neutron_driver: ovn
      tag_from_label: '{version}-{release}'

  ContainerImageRegistryCredentials:
    registry.redhat.io:
      myuser: 'p@55w0rd!'
```

ContainerImagePrepare パラメーターには、単一のルール **set** が含まれます。このルール **set** には **tag** パラメーターを含めないでください。各コンテナーイメージの最新バージョンとリリースを特定するのに、**tag_from_label** パラメーターを使用する必要があります。director はこのルール **set** を使用して各コンテナーイメージの最新バージョンを特定し、それぞれのイメージをプルし、director のコンテナーレジストリーの各イメージをタグ付けします。

手順

1. **openstack tripleo container image prepare** コマンドを実行します。このコマンドは、**containers-prepare-parameter.yaml** ファイルで定義されたソースからすべてのイメージをプルします。固定されたコンテナーイメージのリストが含まれる出力ファイルを指定するには、**--output-env-file** を追加します。

```
$ sudo openstack tripleo container image prepare -e /home/stack/templates/containers-prepare-parameter.yaml --output-env-file overcloud-images.yaml
```

overcloud-images.yaml ファイルは、サービスパラメーターのコンテナーイメージへのマッピングが含まれる環境ファイルです。たとえば、OpenStack Identity (keystone) は、**ContainerKeystoneImage** パラメーターを使用してそのコンテナーイメージを定義します。

```
ContainerKeystoneImage: undercloud.ctlplane.localdomain:8787/rhosp-rhel9/openstack-keystone:17.1
```

コンテナーイメージタグは **{version}-{release}** 形式に一致することに注意してください。

2. **openstack overcloud deploy** コマンドを実行する際に、**containers-prepare-parameter.yaml** および **overcloud-images.yaml** ファイルを特定の順序で環境ファイルコレクションに含めます。

```
$ openstack overcloud deploy --templates \
  ...
  -e /home/stack/containers-prepare-parameter.yaml \
  -e /home/stack/overcloud-images.yaml \
  ...
```

オーバークラウドサービスは、**overcloud-images.yaml** ファイルにリスト表示されている固定されたイメージを使用します。

第8章 RED HAT OPENSTACK PLATFORM 環境用ネットワークのカスタマイズ

Red Hat OpenStack Platform (RHOSP) 環境のアンダークラウドおよびオーバークラウドの物理ネットワークをカスタマイズできます。

8.1. アンダークラウドネットワークのカスタマイズ

アンダークラウドのネットワーク設定をカスタマイズして、特定のネットワーク機能を備えたアンダークラウドをインストールできます。IPv6 ノードとインフラストラクチャーがある場合は、IPv4 ではなく IPv6 を使用するようにアンダークラウドとプロビジョニングネットワークを設定することもできます。

8.1.1. アンダークラウドネットワークインターフェイスの設定

特定のネットワーク機能を持つアンダークラウドをインストールするには、**undercloud.conf** ファイルにカスタムネットワーク設定を追加します。たとえば、一部のインターフェイスは DHCP を持ちません。このような場合は、アンダークラウドのインストールプロセス中に **os-net-config** が設定を適用できるように、**undercloud.conf** ファイルでこれらのインターフェイスの DHCP を無効にする必要があります。

手順

1. アンダークラウドのホストにログインします。
2. 新規ファイル **undercloud-os-net-config.yaml** を作成し、必要なネットワーク設定を追加します。
   addresses セクションには、**172.20.0.1/26** などの **local_ip** を含めます。アンダークラウドで TLS が有効になっている場合は、**undercloud_public_host** (**172.20.0.2/32** など) と **undercloud_admin_host** (**172.20.0.3/32** など) も含める必要があります。

   以下に例を示します。

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  - ip_netmask: 172.20.0.2/32
  - ip_netmask: 172.20.0.3/32
  members:
  - type: interface
    name: nic2
```

   特定のインターフェイスのネットワークボンディングを作成するには、次のサンプルを使用します。

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
  domain: lab.example.com
  ovs_extra:
  - "br-set-external-id br-ctlplane bridge-id br-ctlplane"
  addresses:
  - ip_netmask: 172.20.0.1/26
  - ip_netmask: 172.20.0.2/32
  - ip_netmask: 172.20.0.3/32
  members:
  - name: bond-ctlplane
    type: linux_bond
    use_dhcp: false
    bonding_options: "mode=active-backup"
    mtu: 1500
    members:
    - type: interface
      name: nic2
    - type: interface
      name: nic3
```

3. **undercloud.conf** ファイルの **net_config_override** パラメーターに、**undercloud-os-net-config.yaml** ファイルへのパスを追加します。

```
[DEFAULT]
...
net_config_override=undercloud-os-net-config.yaml
...
```

注記

director は、**net_config_override** パラメーターに追加するファイルをテンプレートとして使用し、**/etc/os-net-config/config.yaml** ファイルを生成します。**os-net-config** はテンプレートで定義するインターフェイスを管理するので、このファイルですべてのアンダークラウドネットワークインターフェイスのカスタマイズを実行する必要があります。

4. アンダークラウドをインストールします。

検証

- アンダークラウドのインストールが正常に完了したら、**/etc/os-net-config/config.yaml** ファイルに該当する設定が含まれていることを確認します。

```
network_config:
- name: br-ctlplane
  type: ovs_bridge
  use_dhcp: false
  dns_servers:
  - 192.168.122.1
```

```
domain: lab.example.com
ovs_extra:
- "br-set-external-id br-ctlplane bridge-id br-ctlplane"
addresses:
- ip_netmask: 172.20.0.1/26
- ip_netmask: 172.20.0.2/32
- ip_netmask: 172.20.0.3/32
members:
- type: interface
  name: nic2
```

8.1.2. IPv6 を使用してベアメタルをプロビジョニングするためのアンダークラウド設定

IPv6 ノードおよびインフラストラクチャーがある場合には、IPv4 ではなく IPv6 を使用するようにアンダークラウドおよびプロビジョニングネットワークを設定することができます。これにより、director は IPv6 ノードに Red Hat OpenStack Platform をプロビジョニングおよびデプロイすることができます。ただし、いくつかの考慮事項があります。

- デュアルスタック IPv4/6 は利用できません。
- tempest 検証が正しく動作しない可能性があります。
- アップグレード時に IPv4 から IPv6 に移行することはできません。

undercloud.conf ファイルを変更して、Red Hat OpenStack Platform で IPv6 プロビジョニングを有効にします。

前提条件

- アンダークラウドの IPv6 アドレス。詳細は、**オーバークラウドの IPv6 ネットワーク**の アンダークラウドの IPv6 アドレスの設定 を参照してください。

手順

1. **undercloud.conf** ファイルを開きます。
2. IPv6 アドレスモードをステートレスまたはステートフルのいずれかに指定します。

```
[DEFAULT]
ipv6_address_mode = <address_mode>
...
```

- NIC がサポートするモードに基づいて、**<address_mode>** を **dhcpv6-stateless** または **dhcpv6-stateful** に置き換えます。

注記

ステートフルアドレスモードを使用する場合、ファームウェア、チェーンローダー、およびオペレーティングシステムは、DHCP サーバーが追跡する ID を生成するために異なるアルゴリズムを使用する場合があります。DHCPv6 は MAC によってアドレスを追跡せず、リクエスターからの ID 値が変更されても、MAC アドレスが同じままである場合、同じアドレスを提供しません。したがって、ステートフル DHCPv6 を使用する場合は、次の手順を実行してネットワークインターフェイスを設定する必要もあります。

3. ステートフル DHCPv6 を使用するようにアンダークラウドを設定した場合は、ベアメタルノードに使用するネットワークインターフェイスを指定します。

```
[DEFAULT]
ipv6_address_mode = dhcpv6-stateful
ironic_enabled_network_interfaces = neutron,flat
...
```

4. ベアメタルノードのデフォルトのネットワークインターフェイスを設定します。

```
[DEFAULT]
...
ironic_default_network_interface = neutron
...
```

5. アンダークラウドがプロビジョニングネットワーク上にルーターを作成するかどうかを指定します。

```
[DEFAULT]
...
enable_routed_networks: <true/false>
...
```

- **<true/false>** を **true** に置き換えて、ルーティングされたネットワークを有効にし、アンダークラウドがプロビジョニングネットワーク上にルーターを作成しないようにします。**true** の場合、データセンタールーターはルーターアドバタイズメントを提供する必要があります。

- **<true/false>** を **false** に置き換えて、ルーティングされたネットワークを無効にし、プロビジョニングネットワーク上にルーターを作成します。

6. ローカル IP アドレス、および SSL/TLS を介した director Admin API および Public API エンドポイントの IP アドレスを設定します。

```
[DEFAULT]
...
local_ip = <ipv6_address>
undercloud_admin_host = <ipv6_address>
undercloud_public_host = <ipv6_address>
...
```

- **<ipv6_address>** をアンダークラウドの IPv6 アドレスに置き換えます。

7. オプション: director がインスタンスの管理に使用するプロビジョニングネットワークを設定します。

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
...
```

- **<ipv6_address>** を、デフォルトのプロビジョニングネットワークを使用していないときにインスタンスの管理に使用するネットワークの IPv6 アドレスに置き換えます。

- **<ipv6_prefix>** を、デフォルトのプロビジョニングネットワークを使用していないときにインスタンスの管理に使用するネットワークの IP アドレス接頭辞に置き換えます。

8. プロビジョニングノードの DHCP 割り当て範囲を設定します。

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
...
```

- **<ipv6_address_dhcp_start>** を、オーバークラウドノードに使用するネットワーク範囲の開始点の IPv6 アドレスに置き換えます。
- **<ipv6_address_dhcp_end>** を、オーバークラウドノードに使用するネットワーク範囲の終わりの IPv6 アドレスに置き換えます。

9. オプション: トラフィックを External ネットワークに転送するようにゲートウェイを設定します。

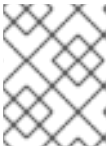
```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
...
```

- デフォルトゲートウェイを使用しない場合は、**<ipv6_gateway_address>** をゲートウェイの IPv6 アドレスに置き換えます。

10. 検査プロセス中に使用する DHCP 範囲を設定します。

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
gateway = <ipv6_gateway_address>
inspection_iprange = <ipv6_address_inspection_start>,<ipv6_address_inspection_end>
...
```

- **<ipv6_address_inspection_start>** を、検査プロセス中に使用するネットワーク範囲の開始点の IPv6 アドレスに置き換えます。
- **<ipv6_address_inspection_end>** を、検査プロセス中に使用するネットワーク範囲の終わりの IPv6 アドレスに置き換えます。



注記

この範囲は、**dhcp_start** と **dhcp_end** で定義された範囲と重複することはできませんが、同じ IP サブネット内になければなりません。

11. サブネットの IPv6 ネームサーバーを設定します。

```
[ctlplane-subnet]
cidr = <ipv6_address>/<ipv6_prefix>
dhcp_start = <ipv6_address_dhcp_start>
dhcp_end = <ipv6_address_dhcp_end>
```

```
gateway = <ipv6_gateway_address>
inspection_iprange = <ipv6_address_inspection_start>,<ipv6_address_inspection_end>
dns_nameservers = <ipv6_dns>
```

- **<ipv6_dns>** をサブネットに固有の DNS ネームサーバーに置き換えます。

12. **virt-customize** ツールを使用してオーバークラウドイメージを変更し、**cloud-init** ネットワーク設定を無効にします。詳細は、Red Hat ナレッジベースで Modifying the Red Hat Linux OpenStack Platform Overcloud Image with virt-customize のソリューションを参照してください。

8.2. オーバークラウドネットワークのカスタマイズ

オーバークラウドの物理ネットワークの設定をカスタマイズできます。たとえば、Jinja2 ansible 形式 (**j2)** の NIC テンプレートファイルを使用して、ネットワークインターフェイスコントローラー (NIC) の設定ファイルを作成できます。

8.2.1. カスタムネットワークインターフェイステンプレートの定義

カスタムネットワークインターフェイステンプレートのセットを作成して、オーバークラウド環境の各ノードの NIC レイアウトを定義できます。オーバークラウドのコアテンプレートコレクションには、さまざまなユースケース向けのデフォルトの NIC レイアウトのセットが含まれています。拡張子が **.j2.yaml** の Jinja2 形式のファイルを使用して、カスタム NIC テンプレートを作成できます。director は、デプロイメント中に Jinja2 ファイルを YAML 形式に変換します。

その後、ノード定義ファイル **overcloud-baremetal-deploy.yaml** の **network_config** プロパティーをカスタム NIC テンプレートに設定して、特定のノードのネットワークをプロビジョニングできます。詳細は、オーバークラウド用のベアメタルノードのプロビジョニング を参照してください。

8.2.1.1. カスタム NIC テンプレートの作成

オーバークラウド環境の各ノードの NIC レイアウトをカスタマイズするための NIC テンプレートを作成します。

手順

1. 必要なサンプルネットワーク設定テンプレートを **/usr/share/ansible/roles/tripleo_network_config/templates/** から環境ファイルディレクトリーにコピーします。

```
$ cp /usr/share/ansible/roles/tripleo_network_config/templates/<sample_NIC_template>
/home/stack/templates/<NIC_template>
```

   - **<sample_NIC_template>** は、コピーするサンプル NIC テンプレートの名前 (**single_nic_vlans/single_nic_vlans.j2** など) に置き換えます。
   - **<NIC_template>** は、カスタム NIC テンプレートファイルの名前 (例: **single_nic_vlans.j2)** に置き換えます。

2. オーバークラウドのネットワーク環境の要件に合わせて、カスタム NIC テンプレートのネットワーク設定を更新します。NIC テンプレートの設定に使用できるプロパティーについては、ネットワークインターフェイス設定オプション を参照してください。NIC テンプレートの例は、カスタムネットワークインターフェイスの例 を参照してください。

3. 既存の環境ファイルを作成または更新して、カスタム NIC 設定テンプレートを有効にします。

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/single_nic_vlans.j2'
  CephStorageNetworkConfigTemplate: '/home/stack/templates/single_nic_vlans_storage.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/single_nic_vlans.j2'
```

4. オーバークラウドがデフォルトの内部負荷分散を使用する場合は、以下の設定を環境ファイルに追加して、Redis および OVNDB に予測可能な仮想 IP を割り当てます。

```
parameter_defaults:
  RedisVirtualFixedIPs: [{'ip_address':'<vip_address>'}]
  OVNDBsVirtualFixedIPs: [{'ip_address':'<vip_address>'}]
```

- **<vip_address>** は、割り当てプール範囲外の IP アドレスに置き換えます。

8.2.1.2. ネットワークインターフェイスの設定オプション

次の表では、ネットワークインターフェイスの設定に使用できるオプションを説明しています。

interface

単一のネットワークインターフェイスを定義します。ネットワークインターフェイスの **name** には、実際のインターフェイス名 (**eth0**、**eth1**、**enp0s25**) または一連の番号付きインターフェイス (**nic1**、**nic2**、**nic3**) が使用されます。**eth0** や **eno2** などの名前付きインターフェイスではなく、**nic1** や **nic2** などの番号付きインターフェイスを使用する場合、ロール内のホストのネットワークインターフェイスがまったく同じである必要はありません。たとえば、あるホストに **em1** と **em2** のインターフェイスが指定されており、別のホストには **eno1** と **eno2** が指定されていても、両ホストの NIC は **nic1** および **nic2** として参照することができます。

番号付きのインターフェイスの順序は、名前付きのネットワークインターフェイスのタイプの順序と同じです。

- **eth0**、**eth1** などの **ethX**。これらは、通常オンボードのインターフェイスです。
- **eno0**、**eno1** などの **enoX**。これらは、通常オンボードのインターフェイスです。
- **enp3s0**、**enp3s1**、**ens3** などの英数字順の **enX** インターフェイス。これらは、通常アドオンのインターフェイスです。

番号による NIC スキームには、アクティブなインターフェイスだけが含まれます (たとえば、インターフェイスにスイッチに接続されたケーブルがあるかどうかが考慮されます)。4 つのインターフェイスを持つホストと、6 つのインターフェイスを持つホストがある場合は、**nic1** から **nic4** を使用して各ホストで 4 本のケーブルだけを結線します。

```
- type: interface
  name: nic2
```

表8.1 interface のオプション

オプション	デフォルト	説明

オプション	デフォルト	説明
name		インターフェイスの名前。ネットワークインターフェイスの **name** には、実際のインターフェイス名 (**eth0**、**eth1**、**enp0s25**) または一連の番号付きインターフェイス (**nic1**、**nic2**、**nic3**) が使用されます。
use_dhcp	False	DHCP を使用して IP アドレスを取得します。
use_dhcpv6	False	DHCP を使用して v6 の IP アドレスを取得します。
addresses		インターフェイスに割り当てられる IP アドレスのリスト
routes		インターフェイスに割り当てられるルートのリスト。詳細は、routes を参照してください。
mtu	1500	接続の最大伝送単位 (MTU: Maximum Transmission Unit)
primary	False	プライマリーインターフェイスとしてインターフェイスを定義します。
persist_mapping	False	システム名の代わりにデバイスのエイリアス設定を記述します。
dhclient_args	なし	DHCP クライアントに渡す引数
dns_servers	なし	インターフェイスに使用する DNS サーバーのリスト
ethtool_opts		特定の NIC で VXLAN を使用する際にスループットを向上させるには、このオプションを **"rx-flow-hash udp4 sdfn"** に設定します。

vlan

VLAN を定義します。**parameters** セクションから渡された VLAN ID およびサブネットを使用します。

以下に例を示します。

```
  - type: vlan
    device: nic{{ loop.index + 1 }}
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask:
      {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars', networks_lower[network] ~
'_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
```

表8.2 vlan のオプション

オプション	デフォルト	説明
vlan_id		VLAN ID
device		VLAN の接続先となる親デバイス。VLAN が OVS ブリッジのメンバーではない場合に、このパラメーターを使用します。たとえば、このパラメーターを使用して、ボンディングされたインターフェイスデバイスに VLAN を接続します。
use_dhcp	False	DHCP を使用して IP アドレスを取得します。
use_dhcpv6	False	DHCP を使用して v6 の IP アドレスを取得します。
addresses		VLAN に割り当てられる IP アドレスのリスト
routes		VLAN に割り当てられるルートのリスト。詳細は、routes を参照してください。
mtu	1500	接続の最大伝送単位 (MTU: Maximum Transmission Unit)
primary	False	プライマリーインターフェイスとして VLAN を定義します。
persist_mapping	False	システム名の代わりにデバイスのエイリアス設定を記述します。
dhclient_args	なし	DHCP クライアントに渡す引数
dns_servers	なし	VLAN に使用する DNS サーバーのリスト

ovs_bond

Open vSwitch で、複数の **インターフェイス** を結合するボンディングを定義します。これにより、冗長性や帯域幅が向上します。

以下に例を示します。

```
members:
  - type: ovs_bond
    name: bond1
    mtu: {{ min_viable_mtu }}
    ovs_options: {{ bond_interface_ovs_options }}
    members:
    - type: interface
      name: nic2
      mtu: {{ min_viable_mtu }}
      primary: true
    - type: interface
      name: nic3
      mtu: {{ min_viable_mtu }}
```

表8.3 ovs_bond のオプション

オプション	デフォルト	説明
name		ボンディング名
use_dhcp	False	DHCP を使用して IP アドレスを取得します。
use_dhcpv6	False	DHCP を使用して v6 の IP アドレスを取得します。
addresses		ボンディングに割り当てられる IP アドレスのリスト
routes		ボンディングに割り当てられるルートのリスト。詳細は、routes を参照してください。
mtu	1500	接続の最大伝送単位 (MTU: Maximum Transmission Unit)
primary	False	プライマリーインターフェイスとしてインターフェイスを定義します。
members		ボンディングで使用するインターフェイスオブジェクトのリスト
ovs_options		ボンディング作成時に OVS に渡すオプションのセット

オプション	デフォルト	説明
ovs_extra		ボンディングのネットワーク設定ファイルで OVS_EXTRA パラメーターとして設定するオプションのセット
defroute	True	DHCP サービスにより提供されるデフォルトのルートを使用します。**use_dhcp** または **use_dhcpv6** を選択した場合に限り有効です。
persist_mapping	False	システム名の代わりにデバイスのエイリアス設定を記述します。
dhclient_args	なし	DHCP クライアントに渡す引数
dns_servers	なし	ボンディングに使用する DNS サーバーのリスト

ovs_bridge

Open vSwitch で、複数の **interface**、**ovs_bond**、**vlan** オブジェクトを接続するブリッジを定義します。

ネットワークインターフェイス種別 **ovs_bridge** には、パラメーター **name** を使用します。

注記

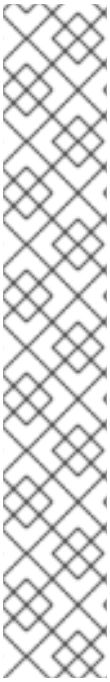
複数のブリッジがある場合は、デフォルト名の **bridge_name** を受け入れるのではなく、個別のブリッジ名を使用する必要があります。個別の名前を使用しないと、コンバージフェーズ時に 2 つのネットワークボンディングが同じブリッジに配置されます。

外部の tripleo ネットワークに OVS ブリッジを定義している場合は、**bridge_name** および **interface_name** の値を維持します。デプロイメントフレームワークが、これらの値を自動的にそれぞれ外部ブリッジ名および外部インターフェイス名に置き換えるためです。

以下に例を示します。

```
- type: ovs_bridge
  name: br-bond
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: ovs_bond
    name: bond1
    mtu: {{ min_viable_mtu }}
    ovs_options: {{ bound_interface_ovs_options }}
    members:
    - type: interface
      name: nic2
```

```
    mtu: {{ min_viable_mtu }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu }}
```



注記

OVS ブリッジは、Networking サービス (neutron) サーバーに接続して設定データを取得します。OpenStack の制御トラフィック (通常 Control Plane および Internal API ネットワーク) が OVS ブリッジに配置されていると、OVS がアップグレードされたり、管理ユーザーやプロセスによって OVS ブリッジが再起動されたりする度に、neutron サーバーへの接続が失われます。これにより、ダウンタイムが発生します。このような状況でダウンタイムが許容されない場合は、コントロールグループのネットワークを OVS ブリッジではなく別のインターフェイスまたはボンディングに配置する必要があります。

- Internal API ネットワークをプロビジョニングインターフェイス上の VLAN および 2 番目のインターフェイス上の OVS ブリッジに配置すると、最小の設定を行うことができます。
- ボンディングを実装する場合は、少なくとも 2 つのボンディング (4 つのネットワークインターフェイス) が必要です。コントロールグループを Linux ボンディング (Linux ブリッジ) に配置します。PXE ブート用のシングルインターフェイスへの LACP フォールバックをスイッチがサポートしていない場合には、このソリューションには少なくとも 5 つの NIC が必要となります。

表8.4 ovs_bridge のオプション

オプション	デフォルト	説明
name		ブリッジ名
use_dhcp	False	DHCP を使用して IP アドレスを取得します。
use_dhcpv6	False	DHCP を使用して v6 の IP アドレスを取得します。
addresses		ブリッジに割り当てられる IP アドレスのリスト
routes		ブリッジに割り当てられるルートのリスト。詳細は、routes を参照してください。
mtu	1500	接続の最大伝送単位 (MTU: Maximum Transmission Unit)
members		ブリッジで使用するインターフェイス、VLAN、およびボンディングオブジェクトのリスト

オプション	デフォルト	説明
ovs_options		ブリッジ作成時に OVS に渡すオプションのセット
ovs_extra		ブリッジのネットワーク設定ファイルで OVS_EXTRA パラメーターとして設定するオプションのセット
defroute	True	DHCP サービスにより提供されるデフォルトのルートを使用します。**use_dhcp** または **use_dhcpv6** を選択した場合に限り有効です。
persist_mapping	False	システム名の代わりにデバイスのエイリアス設定を記述します。
dhclient_args	なし	DHCP クライアントに渡す引数
dns_servers	なし	ブリッジに使用する DNS サーバーのリスト

linux_bond

複数の **インターフェイス** を結合する Linux ボンディングを定義します。これにより、冗長性や帯域幅が向上します。**bonding_options** パラメーターには、カーネルベースのボンディングオプションを指定するようにしてください。

以下に例を示します。

```
- type: linux_bond
  name: bond1
  mtu: {{ min_viable_mtu }}
  bonding_options: "mode=802.3ad lacp_rate=fast updelay=1000 miimon=100
xmit_hash_policy=layer3+4"
  members:
    type: interface
    name: ens1f0
    mtu: {{ min_viable_mtu }}
    primary: true
  type: interface
    name: ens1f1
    mtu: {{ min_viable_mtu }}
```

表8.5 linux_bond のオプション

オプション	デフォルト	説明
name		ボンディング名

オプション	デフォルト	説明
use_dhcp	False	DHCP を使用して IP アドレスを取得します。
use_dhcpv6	False	DHCP を使用して v6 の IP アドレスを取得します。
addresses		ボンディングに割り当てられる IP アドレスのリスト
routes		ボンディングに割り当てられるルートのリスト。routesを参照してください。
mtu	1500	接続の最大伝送単位 (MTU: Maximum Transmission Unit)
primary	False	プライマリーインターフェイスとしてインターフェイスを定義します。
members		ボンディングで使用するインターフェイスオブジェクトのリスト
bonding_options		ボンディングを作成する際のオプションのセット
defroute	True	DHCP サービスにより提供されるデフォルトのルートを使用します。**use_dhcp** または **use_dhcpv6** を選択した場合に限り有効です。
persist_mapping	False	システム名の代わりにデバイスのエイリアス設定を記述します。
dhclient_args	なし	DHCP クライアントに渡す引数
dns_servers	なし	ボンディングに使用する DNS サーバーのリスト

linux_bridge

複数の **interface**、**linux_bond**、**vlan** オブジェクトを接続する Linux ブリッジを定義します。外部のブリッジは、パラメーターに 2 つの特殊な値も使用します。

- **bridge_name**: 外部ブリッジ名に置き換えます。

- **interface_name**: 外部インターフェイスに置き換えます。

以下に例を示します。

```
- type: linux_bridge
    name: bridge_name
    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
```

表8.6 linux_bridge のオプション

オプション	デフォルト	説明
name		ブリッジ名
use_dhcp	False	DHCP を使用して IP アドレスを取得します。
use_dhcpv6	False	DHCP を使用して v6 の IP アドレスを取得します。
addresses		ブリッジに割り当てられる IP アドレスのリスト
routes		ブリッジに割り当てられるルートのリスト。詳細は、routes を参照してください。
mtu	1500	接続の最大伝送単位 (MTU: Maximum Transmission Unit)
members		ブリッジで使用するインターフェイス、VLAN、およびボンディングオブジェクトのリスト

オプション	デフォルト	説明
defroute	True	DHCP サービスにより提供されるデフォルトのルートを使用します。**use_dhcp** または **use_dhcpv6** を選択した場合に限り有効です。
persist_mapping	False	システム名の代わりにデバイスのエイリアス設定を記述します。
dhclient_args	なし	DHCP クライアントに渡す引数
dns_servers	なし	ブリッジに使用する DNS サーバーのリスト

routes

ネットワークインターフェイス、VLAN、ブリッジ、またはボンディングに適用するルートのリストを定義します。

以下に例を示します。

```
- type: linux_bridge
  name: bridge_name
  ...
  routes: {{ [ctlplane_host_routes] | flatten | unique }}
```

オプション	デフォルト	説明
ip_netmask	なし	接続先ネットワークの IP およびネットマスク
default	False	このルートをデフォルトルートに設定します。**ip_netmask: 0.0.0.0/0** の設定と等価です。
next_hop	なし	接続先ネットワークに到達するのに使用するルーターの IP アドレス

8.2.1.3. カスタムネットワークインターフェイスの例

次の例は、ネットワークインターフェイステンプレートをカスタマイズする方法を示しています。

制御グループと OVS ブリッジを分離する例

以下に示すコントローラーノードの NIC テンプレートの例では、OVS ブリッジとは別に制御グループを設定します。このテンプレートは、5 つのネットワークインターフェイスを使用し、タグ付けされた複数の VLAN デバイスを、番号によるインターフェイスに割り当てます。テンプレートは、**nic4** およ

び **nic5** に OVS ブリッジを作成します。

```
network_config:
- type: interface
  name: nic1
  mtu: {{ ctlplane_mtu }}
  use_dhcp: false
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: {{ bond_interface_ovs_options }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
{% for network in role_networks if not network.startswith('Tenant') %}
- type: vlan
  device: bond_api
  mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
  vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
  addresses:
  - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
  routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endfor %}
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  members:
  - type: linux_bond
    name: bond-data
    mtu: {{ min_viable_mtu_dataplane }}
    bonding_options: {{ bond_interface_ovs_options }}
    members:
    - type: interface
      name: nic4
      mtu: {{ min_viable_mtu_dataplane }}
      primary: true
    - type: interface
      name: nic5
      mtu: {{ min_viable_mtu_dataplane }}
{% for network in role_networks if network.startswith('Tenant') %}
  - type: vlan
    device: bond-data
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
```

```
    addresses:
    - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
```

複数の NIC の例

次の例では、2 番目の NIC を使用して、DHCP アドレスを持つインフラストラクチャーネットワークに接続し、ボンド用に別の NIC を使用します。

```
network_config:
  # Add a DHCP infrastructure network to nic2
  - type: interface
    name: nic2
    mtu: {{ tenant_mtu }}
    use_dhcp: true
    primary: true
  - type: vlan
    mtu: {{ tenant_mtu }}
    vlan_id: {{ tenant_vlan_id }}
    addresses:
    - ip_netmask: {{ tenant_ip }}/{{ tenant_cidr }}
    routes: {{ [tenant_host_routes] | flatten | unique }}
  - type: ovs_bridge
    name: br-bond
    mtu: {{ external_mtu }}
    dns_servers: {{ ctlplane_dns_nameservers }}
    use_dhcp: false
    members:
      - type: interface
        name: nic10
        mtu: {{ external_mtu }}
        use_dhcp: false
        primary: true
      - type: vlan
        mtu: {{ external_mtu }}
        vlan_id: {{ external_vlan_id }}
        addresses:
        - ip_netmask: {{ external_ip }}/{{ external_cidr }}
        routes: {{ [external_host_routes, [{'default': True, 'next_hop': external_gateway_ip}]] | flatten |
unique }}
```

8.2.1.4. 事前にプロビジョニングされたノードの NIC マッピングのカスタマイズ

事前にプロビジョニングされたノードを使用している場合は、次のいずれかの方法を使用して、特定のノードの **os-net-config** マッピングを指定できます。

- 環境ファイルで **NetConfigDataLookup** heat パラメーターを設定し、**--network-config** を指定せずに **openstack overcloud node provision** コマンドを実行します。
- ノード定義ファイル **overcloud-baremetal-deploy.yaml** で **net_config_data_lookup** プロパティーを設定し、**--network-config** を指定して **openstack overcloud node provision** コマンドを実行します。

注記

事前にプロビジョニングされたノードを使用していない場合は、ノード定義ファイルで NIC マッピングを設定する必要があります。**net_config_data_lookup** プロパティーの設定の詳細は、ベアメタルノードのプロビジョニング属性 を参照してください。

各ノードの物理インターフェイスにエイリアスを割り当てて、**nic1** または **nic2** などの特定のエイリアスにマッピングする物理 NIC を事前に定義したり、MAC アドレスを特定のエイリアスにマッピングしたりできます。MAC アドレスまたは DMI キーワードを使用して特定のノードをマップしたり、DMI キーワードを使用してノードのグループをマップしたりできます。次の例では、物理インターフェイスへのエイリアスを持つ 3 つのノードと 2 つのノードグループを設定します。得られた設定は、**os-net-config** により適用されます。これぞれのノードで、適用された設定が **/etc/os-net-config/mapping.yaml** ファイルの **interface_mapping** セクションに表示されます。

例 1: os-net-config-mappings.yaml で NetConfigDataLookup パラメーターを設定する

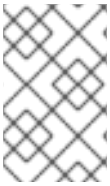
```
NetConfigDataLookup:
  node1: (1)
    nic1: "00:c8:7c:e6:f0:2e"
  node2:
    nic1: "00:18:7d:99:0c:b6"
  node3: (2)
    dmiString: "system-uuid" (3)
    id: 'A8C85861-1B16-4803-8689-AFC62984F8F6'
    nic1: em3
  # Dell PowerEdge
  nodegroup1: (4)
    dmiString: "system-product-name"
    id: "PowerEdge R630"
    nic1: em3
    nic2: em1
    nic3: em2
  # Cisco UCS B200-M4"
  nodegroup2:
    dmiString: "system-product-name"
    id: "UCSB-B200-M4"
    nic1: enp7s0
    nic2: enp6s0
```

(1) 指定の MAC アドレスに **node1** をマップし、このノードの MAC アドレスのエイリアスとして **nic1** を割り当てます。

(2) **node3** をシステムUUID "A8C85861-1B16-4803-8689-AFC62984F8F6" を持つノードにマップし、このノード上の **em3** インターフェイスのエイリアスとして **nic1** を割り当てます。

(3) **dmiString** パラメーターは有効な文字列キーワードに設定する必要があります。有効な文字列キーワードのリストは、DMIDECODE(8) のマニュアルページを参照してください。

(4) **nodegroup1** 内のすべてのノードを製品名 PowerEdge R630 のノードにマップし、これらのノード上の名前付きインターフェイスのエイリアスとして **nic1**、**nic2**、および **nic3** を割り当てます。



注記

通常、**os-net-config** はすでに接続済みの **UP** 状態のインターフェイスしか登録しません。ただし、カスタムマッピングファイルを使用してインターフェイスをハードコーディングすると、**DOWN** 状態のインターフェイスであっても登録されます。

例 2: overcloud-baremetal-deploy.yaml で net_config_data_lookup プロパティーを設定する (特定のノード)

```
- name: Controller
  count: 3
  defaults:
    network_config:
      net_config_data_lookup:
        node1:
          nic1: "00:c8:7c:e6:f0:2e"
        node2:
          nic1: "00:18:7d:99:0c:b6"
        node3:
          dmiString: "system-uuid"
          id: 'A8C85861-1B16-4803-8689-AFC62984F8F6'
          nic1: em3
        # Dell PowerEdge
        nodegroup1:
          dmiString: "system-product-name"
          id: "PowerEdge R630"
          nic1: em3
          nic2: em1
          nic3: em2
        # Cisco UCS B200-M4"
        nodegroup2:
          dmiString: "system-product-name"
          id: "UCSB-B200-M4"
          nic1: enp7s0
          nic2: enp6s0
```

例 3: overcloud-baremetal-deploy.yaml で net_config_data_lookup プロパティーを設定する (ロールの全ノード)

```
- name: Controller
  count: 3
  defaults:
    network_config:
      template: templates/net_config_bridge.j2
      default_route_network:
      - external
  instances:
  - hostname: overcloud-controller-0
    network_config:
      <name/groupname>:
        nic1: 'XX:XX:XX:XX:XX:XX'
        nic2: 'YY:YY:YY:YY:YY:YY'
        nic3: 'ens1f0'
```

8.2.2. コンポーザブルネットワーク

特定のネットワークトラフィックを異なるネットワークでホストする場合は、カスタムコンポーザブルネットワークを作成することができます。Director は、ネットワーク分離が有効になっているデフォルトのネットワークトポロジーを提供します。この設定は **/usr/share/openstack-tripleo-heat-templates/network-data-samples/default-network-isolation.yaml** にあります。

デフォルトのオーバークラウドでは、以下に示す事前定義済みのネットワークセグメントのセットが使用されます。

- Internal API
- Storage
- Storage Management
- Tenant
- External

コンポーザブルネットワークを使用して、さまざまなサービス用のネットワークを追加することができます。たとえば、NFS トラフィック専用のネットワークがある場合には、複数のロールに提供できます。

director は、デプロイメントおよび更新フェーズでのカスタムネットワークの作成をサポートしています。このような追加のネットワークを、ベアメタルノード、システム管理に使用したり、異なるロール用に別のネットワークを作成するのに使用したりすることができます。また、これは、トラフィックが複数のネットワーク間でルーティングされる、分離型のデプロイメント用に複数のネットワークセットを作成するのに使用することもできます。

8.2.2.1. コンポーザブルネットワークの追加

コンポーザブルネットワークを使用して、さまざまなサービス用のネットワークを追加します。たとえば、ストレージバックアップトラフィック専用のネットワークがある場合には、ネットワークを複数のロールに提供できます。

手順

1. 利用可能なサンプルネットワーク設定ファイルをリストします。

```
$ ll /usr/share/openstack-tripleo-heat-templates/network-data-samples/
-rw-r--r--. 1 root root 1554 May 11 23:04 default-network-isolation-ipv6.yaml
-rw-r--r--. 1 root root 1181 May 11 23:04 default-network-isolation.yaml
-rw-r--r--. 1 root root 1126 May 11 23:04 ganesha-ipv6.yaml
-rw-r--r--. 1 root root 1100 May 11 23:04 ganesha.yaml
-rw-r--r--. 1 root root 3556 May 11 23:04 legacy-routed-networks-ipv6.yaml
-rw-r--r--. 1 root root 2929 May 11 23:04 legacy-routed-networks.yaml
-rw-r--r--. 1 root root  383 May 11 23:04 management-ipv6.yaml
-rw-r--r--. 1 root root  290 May 11 23:04 management.yaml
-rw-r--r--. 1 root root  136 May 11 23:04 no-networks.yaml
-rw-r--r--. 1 root root 2725 May 11 23:04 routed-networks-ipv6.yaml
-rw-r--r--. 1 root root 2033 May 11 23:04 routed-networks.yaml
-rw-r--r--. 1 root root  943 May 11 23:04 vip-data-default-network-isolation.yaml
-rw-r--r--. 1 root root  848 May 11 23:04 vip-data-fixed-ip.yaml
-rw-r--r--. 1 root root 1050 May 11 23:04 vip-data-routed-networks.yaml
```

2. 必要なサンプルネットワーク設定ファイルを **/usr/share/openstack-tripleo-heat-templates/network-data-samples** から環境ファイルディレクトリーにコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/network-data-samples/default-network-isolation.yaml /home/stack/templates/network_data.yaml
```

3. **network_data.yaml** 設定ファイルを編集して、新しいネットワークのセクションを追加します。

```
- name: StorageBackup
  vip: false
  name_lower: storage_backup
  subnets:
    storage_backup_subnet:
      ip_subnet: 172.16.6.0/24
      allocation_pools:
        - start: 172.16.6.4
        - end: 172.16.6.250
      gateway_ip: 172.16.6.1
```

環境のその他のネットワーク VIP 属性を設定します。ネットワーク属性の設定に使用できるプロパティーの詳細は、ネットワーク定義ファイル設定オプション (https://access.redhat.com/documentation/ja-jp/red_hat_openstack_platform/17.1/html/installing_and_managing_red_hat_openstack_platform_w overcloud-networking_installing-director-on-the-undercloud#ref_network-definition-file-configuration-options_overcloud_networking) を参照してください。

4. 仮想 IP を含むコンポーザブルネットワークを追加し、一部の API サービスをこのネットワークにマッピングする場合は、**CloudName{network.name}** 定義を使用して API エンドポイントの DNS 名を設定します。

```
CloudName{{network.name}}
```

以下に例を示します。

```
parameter_defaults:
  ...
  CloudNameOcProvisioning: baremetal-vip.example.com
```

5. 必要なサンプルネットワーク VIP 定義テンプレートを **/usr/share/openstack-tripleo-heat-templates/network-data-samples** から環境ファイルディレクトリーにコピーします。次の例では、**vip-data-default-network-isolation.yaml** を **vip_data.yaml** という名前のローカル環境ファイルにコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/network-data-samples/vip-data-default-network-isolation.yaml  /home/stack/templates/vip_data.yaml
```

6. **vip_data.yaml** 設定ファイルを編集します。仮想 IP データは仮想 IP アドレス定義のリストであり、それぞれに IP アドレスが割り当てられているネットワークの名前が含まれています。

```
- network: storage_mgmt
  dns_name: overcloud
- network: internal_api
  dns_name: overcloud
```

```
- network: storage
  dns_name: overcloud
- network: external
  dns_name: overcloud
  ip_address: <vip_address>
- network: ctlplane
  dns_name: overcloud
```

- **<vip_address>** は、必要な仮想 IP アドレスに置き換えます。

VIP 定義ファイルでネットワーク VIP 属性を設定するために使用できるプロパティーの詳細は、ネットワーク VIP 属性のプロパティー を参照してください。

7. サンプルネットワーク設定テンプレートをコピーします。Jinja2 テンプレートは、NIC 設定テンプレートを定義するために使用されます。**/usr/share/ansible/roles/tripleo_network_config/templates/** ディレクトリーにある例を参照してください。例の 1 つが要件に一致する場合は、それを使用してください。例が要件に合わない場合は、サンプル設定ファイルをコピーし、必要に応じて変更します。

```
$ cp
/usr/share/ansible/roles/tripleo_network_config/templates/single_nic_vlans/single_nic_vlans.j2
/home/stack/templates/
```

8. **single_nic_vlans.j2** 設定ファイルを編集します。

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: ovs_bridge
  name: {{ neutron_physical_bridge_name }}
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ ctlplane_host_routes }}
  members:
  - type: interface
    name: nic1
    mtu: {{ min_viable_mtu }}
    # force the MAC address of the bridge to this interface
    primary: true
{% for network in role_networks %}
  - type: vlan
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask:
        {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
```

```
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{% endfor %}
```

9. **overcloud-baremetal-deploy.yaml** 設定ファイルで **network_config** テンプレートを設定します。

```
- name: CephStorage
  count: 3
  defaults:
    networks:
    - network: storage
    - network: storage_mgmt
    - network: storage_backup
    network_config:
      template: /home/stack/templates/single_nic_vlans.j2
```

10. オーバークラウドネットワークをプロビジョニングします。このアクションにより、オーバークラウドのデプロイ時に環境ファイルで使用される出力ファイルが生成されます。

```
(undercloud)$ openstack overcloud network provision --output <deployment_file>
/home/stack/templates/<networks_definition_file>.yaml
```

- **<networks_definition_file>** は、ネットワーク定義ファイルの名前 (**network_data.yaml** など) に置き換えます。
- **<deployment_file>** は、デプロイメントコマンドに追加するために生成する heat 環境ファイルの名前に置き換えます (例 **:/home/stack/templates/overcloud-networks-deployed.yaml)**。

11. ネットワーク VIP をプロビジョニングし、**vip-deployed-environment.yaml** ファイルを生成します。オーバークラウドをデプロイするときに、このファイルを使用します。

```
(overcloud)$ openstack overcloud network vip provision  --stack <stack> --output
<deployment_file> /home/stack/templates/vip_data.yaml
```

- **<stack>** は、ネットワーク VIP がプロビジョニングされるスタックの名前に置き換えます。指定しない場合、デフォルトは overcloud です。
- **<deployment_file>** は、デプロイメントコマンドに含めるために生成する heat 環境ファイルの名前に置き換えます (例 **:/home/stack/templates/overcloud-vip-deployed.yaml)**。

8.2.2.2. ロールへのコンポーザブルネットワークの追加

コンポーザブルネットワークをご自分の環境で定義したオーバークラウドロールに割り当てることができます。たとえば、カスタム **StorageBackup** ネットワークを Ceph Storage ノードに追加することができます。

手順

1. カスタム **roles_data.yaml** ファイルがまだない場合には、デフォルトをご自分のホームディレクトリーにコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/roles_data.yaml
/home/stack/templates/roles_data.yaml
```

2. カスタムの **roles_data.yaml** ファイルを編集します。

3. ネットワークを追加するロールの **networks** リストにネットワーク名を追加します。たとえば、**StorageBackup** ネットワークを Ceph Storage ロールに追加するには、以下のスニペット例を使用します。

```
- name: CephStorage
  description: |
    Ceph OSD Storage node role
  networks:
    Storage
      subnet: storage_subnet
    StorageMgmt
      subnet: storage_mgmt_subnet
    StorageBackup
      subnet: storage_backup_subnet
```

4. カスタムネットワークを対応するロールに追加したら、ファイルを保存します。

openstack overcloud deploy コマンドを実行する際に、**-r** オプションを使用してカスタムの **roles_data.yaml** ファイルを指定します。**-r** オプションを設定しないと、デプロイメントコマンドはデフォルトのロールセットとそれに対応する割り当て済みのネットワークを使用します。

8.2.2.3. コンポーザブルネットワークへの OpenStack サービスの割り当て

各 OpenStack サービスは、リソースレジストリーでデフォルトのネットワーク種別に割り当てられます。これらのサービスは、そのネットワーク種別に割り当てられたネットワーク内の IP アドレスにバインドされます。OpenStack サービスはこれらのネットワークに分割されますが、実際の物理ネットワーク数はネットワーク環境ファイルに定義されている数と異なる可能性があります。環境ファイル (たとえば **/home/stack/templates/service-reassignments.yaml**) で新たにネットワークマッピングを定義することで、OpenStack サービスを異なるネットワーク種別に再割り当てすることができます。**ServiceNetMap** パラメーターにより、各サービスに使用するネットワーク種別が決定されます。

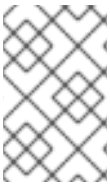
たとえば、ハイライトしたセクションを変更して、Storage Management ネットワークサービスを Storage Backup ネットワークに再割り当てすることができます。

```
parameter_defaults:
  ServiceNetMap:
    SwiftStorageNetwork: storage_backup
    CephClusterNetwork: storage_backup
```

これらのパラメーターを **storage_backup** に変更すると、対象のサービスは Storage Management ネットワークではなく、Storage Backup ネットワークに割り当てられます。つまり、**parameter_defaults** セットを設定するのは Storage Backup ネットワーク用だけで、Storage Management ネットワーク用に設定する必要はありません。

director はカスタムの **ServiceNetMap** パラメーターの定義を **ServiceNetMapDefaults** から取得したデフォルト値の事前定義済みリストにマージして、デフォルト値を上書きします。director はカスタマイズされた設定を含む完全なリストを **ServiceNetMap** に返し、そのリストは多様なサービスのネットワーク割り当ての設定に使用されます。

サービスマッピングは、Pacemaker を使用するノードの **network_data.yaml** ファイルで **vip: true** と設定されているネットワークに適用されます。オーバークラウドの負荷分散サービスにより、トラフィックが仮想 IP から特定のサービスのエンドポイントにリダイレクトされます。



注記

デフォルトのサービスの全リストは、**/usr/share/openstack-tripleo-heat-templates/network/service_net_map.j2.yaml** ファイル内の **ServiceNetMapDefaults** パラメーターの箇所に記載されています。

8.2.2.4. カスタムコンポーザブルネットワークの有効化

デフォルトの NIC テンプレートの 1 つを使用して、カスタムのコンポーザブルネットワークを有効にします。この例では、Single NIC with VLANs テンプレート (**custom_single_nic_vlans**) を使用します。

手順

1. **stackrc** アンダークラウド認証情報ファイルを入手します。

```
$ source ~/stackrc
```

2. オーバークラウドネットワークをプロビジョニングします。

```
$ openstack overcloud network provision \
  --output overcloud-networks-deployed.yaml \
  custom_network_data.yaml
```

3. ネットワーク VIP をプロビジョニングします。

```
$ openstack overcloud network vip provision \
    --stack overcloud \
    --output overcloud-networks-vips-deployed.yaml \
     custom_vip_data.yaml
```

4. オーバークラウドノードをプロビジョニングします。

```
$ openstack overcloud node provision \
    --stack overcloud \
    --output overcloud-baremetal-deployed.yaml \
    overcloud-baremetal-deploy.yaml
```

5. 以下の例のように、設定ファイルとテンプレートを必要な順序で指定して、**openstack overcloud deploy** コマンドを作成します。

```
$ openstack overcloud deploy --templates \
  --networks-file network_data_v2.yaml \
  -e overcloud-networks-deployed.yaml \
  -e overcloud-networks-vips-deployed.yaml \
  -e overcloud-baremetal-deployed.yaml
  -e custom-net-single-nic-with-vlans.yaml
```

上記の例に示すコマンドにより、追加のカスタムネットワークを含め、コンポーザブルネットワークがオーバークラウドのノード全体にデプロイされます。

8.2.2.5. デフォルトネットワークの名前変更

network_data.yaml ファイルを使用して、デフォルトネットワークのユーザー表示名を変更することができます。

- InternalApi
- External
- Storage
- StorageMgmt
- Tenant

これらの名前を変更するのに、**name** フィールドを変更しないでください。代わりに、**name_lower** フィールドをネットワークの新しい名前に変更し、新しい名前で ServiceNetMap を更新します。

手順

1. **network_data.yaml** ファイルで、名前を変更する各ネットワークの **name_lower** パラメーターに新しい名前を入力します。

```
- name: InternalApi
  name_lower: MyCustomInternalApi
```

2. **service_net_map_replace** パラメーターに、**name_lower** パラメーターのデフォルト値を追加します。

```
- name: InternalApi
  name_lower: MyCustomInternalApi
  service_net_map_replace: internal_api
```

8.2.3. 追加のオーバークラウドネットワーク設定

本章では、「カスタムネットワークインターフェイステンプレートの定義」で説明した概念および手順に続いて、オーバークラウドネットワークの要素を設定する際に役立つその他の情報を提供します。

8.2.3.1. ルートおよびデフォルトルートの設定

ホストのデフォルトルートは、2 つの方法のどちらかで設定することができます。インターフェイスが DHCP を使用しており、DHCP サーバーがゲートウェイアドレスを提供している場合には、システムはそのゲートウェイのデフォルトルートを使用します。それ以外の場合には、固定 IP が指定されたインターフェイスにデフォルトのルートを設定することができます。

Linux カーネルは複数のデフォルトゲートウェイをサポートしますが、最も低いメトリックのゲートウェイだけを使用します。複数の DHCP インターフェイスがある場合には、どのデフォルトゲートウェイが使用されるかが推測できなくなります。このような場合には、デフォルトルートを使用しないインターフェイスに **defroute: false** を設定することを推奨します。

たとえば、DHCP インターフェイス (**nic3**) をデフォルトのルートに指定する場合があります。そのためには、以下の YAML スニペットを使用して別の DHCP インターフェイス (**nic2**) 上のデフォルトルートを無効にします。

```
# No default route on this DHCP interface
```

```
- type: interface
  name: nic2
  use_dhcp: true
  defroute: false
# Instead use this DHCP interface as the default route
- type: interface
  name: nic3
  use_dhcp: true
```

注記

defroute パラメーターは、DHCP で取得したルートにのみ適用されます。

固定 IP が指定されたインターフェイスに静的なルートを設定するには、サブネットへのルートを指定します。たとえば、Internal API ネットワーク上のゲートウェイ 172.17.0.1 を経由するサブネット 10.1.2.0/24 にルートを設定します。

```
- type: vlan
  device: bond1
  vlan_id: 9
  addresses:
  - ip_netmask: 172.17.0.100/16
  routes:
  - ip_netmask: 10.1.2.0/24
    next_hop: 172.17.0.1
```

8.2.3.2. ポリシーベースのルーティングの設定

コントローラーノードで、異なるネットワークからの無制限のアクセスを設定するには、ポリシーベースのルーティングを設定します。複数のインターフェイスを持つホストでは、ポリシーベースのルーティングはルーティングテーブルを使用し、送信元のアドレスに応じて特定のインターフェイス経由でトラフィックを送信することができます。送信先が同じであっても、異なる送信元からのパケットを異なるネットワークにルーティングすることができます。

たとえば、デフォルトのルートが External ネットワークの場合でも、パケットの送信元アドレスに基づいてトラフィックを Internal API ネットワークに送信するようにルートを設定することができます。インターフェイスごとに特定のルーティングルールを定義することもできます。

Red Hat OpenStack Platform では **os-net-config** ツールを使用してオーバークラウドノードのネットワーク属性を設定します。**os-net-config** ツールは、コントローラーノードの以下のネットワークルーティングを管理します。

- **/etc/iproute2/rt_tables** ファイルのルーティングテーブル
- **/etc/sysconfig/network-scripts/rule-{ifname}** ファイルの IPv4 ルール
- **/etc/sysconfig/network-scripts/rule6-{ifname}** ファイルの IPv6 ルール
- **/etc/sysconfig/network-scripts/route-{ifname}** のルーティングテーブル固有のルート

前提条件

- アンダークラウドが正常にインストールされている。詳細は、**director を使用した Red Hat OpenStack Platform のインストールと管理ガイド** の アンダークラウドへの director のインストール を参照してください。

手順

1. **/home/stack/templates/custom-nics** ディレクトリーからのカスタム NIC テンプレートに **interface** エントリーを作成し、インターフェイスのルートを定義し、デプロイメントに関連するルールを定義します。

```
network_config:
- type: interface
  name: em1
  use_dhcp: false
  addresses:
    - ip_netmask: {{ external_ip }}/{{ external_cidr}}
  routes:
    - default: true
      next_hop: {{ external_gateway_ip }}
    - ip_netmask: {{ external_ip }}/{{ external_cidr}}
      next_hop: {{ external_gateway_ip }}
      table: 2
      route_options: metric 100
  rules:
    - rule: "iif em1 table 200"
      comment: "Route incoming traffic to em1 with table 200"
    - rule: "from 192.0.2.0/24 table 200"
      comment: "Route all traffic from 192.0.2.0/24 with table 200"
    - rule: "add blackhole from 172.19.40.0/24 table 200"
    - rule: "add unreachable iif em1 from 192.168.1.0/24"
```

2. ご自分のデプロイメントに該当するその他の環境ファイルと共に、カスタム NIC 設定およびネットワーク環境ファイルをデプロイメントコマンドに追加します。

```
$ openstack overcloud deploy --templates \
-e /home/stack/templates/<custom-nic-template>
-e <OTHER_ENVIRONMENT_FILES>
```

検証

- コントローラーノードで以下のコマンドを入力して、ルーティング設定が正しく機能していることを確認します。

```
$ cat /etc/iproute2/rt_tables
$ ip route
$ ip rule
```

8.2.3.3. ジャンボフレームの設定

最大伝送単位 (MTU) の設定は、単一の Ethernet フレームで転送されるデータの最大量を決定します。各フレームはヘッダー形式でデータを追加するため、より大きい値を指定すると、オーバーヘッドが少なくなります。デフォルト値が 1500 で、1500 より高い値を使用する場合には、ジャンボフレームをサポートするスイッチポートの設定が必要になります。大半のスイッチは、9000 以上の MTU 値をサポートしていますが、それらの多くはデフォルトで 1500 に指定されています。

VLAN の MTU は、物理インターフェイスの MTU を超えることができません。ボンディングまたはインターフェイスで MTU 値を含めるようにしてください。

ジャンボフレームは、Storage、Storage Management、Internal API、Tenant ネットワークのすべてにメリットをもたらします。

jinja2 テンプレートまたは **network_data.yaml** ファイルで **mtu** の値を変更できます。**network_data.yaml** ファイルに値を設定すると、デプロイ中にレンダリングされます。

警告

ルーターは、通常レイヤー 3 の境界を超えてジャンボフレームでのデータを転送することができません。接続性の問題を回避するには、プロビジョニングインターフェイス、外部インターフェイス、および Floating IP インターフェイスのデフォルト MTU を変更しないでください。

```
---
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
network_config:
- type: ovs_bridge
  name: bridge_name
  mtu: {{ min_viable_mtu }}
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  addresses:
  - ip_netmask: {{ ctlplane_ip }}/{{ ctlplane_subnet_cidr }}
  routes: {{ [ctlplane_host_routes] | flatten | unique }}
  members:
  - type: interface
    name: nic1
    mtu: {{ min_viable_mtu }}
    primary: true
  - type: vlan
    mtu: 9000 (1)
    vlan_id: {{ storage_vlan_id }}
    addresses:
    - ip_netmask: {{ storage_ip }}/{{ storage_cidr }}
    routes: {{ [storage_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ storage_mgmt_mtu }} (2)
    vlan_id: {{ storage_mgmt_vlan_id }}
    addresses:
    - ip_netmask: {{ storage_mgmt_ip }}/{{ storage_mgmt_cidr }}
    routes: {{ [storage_mgmt_host_routes] | flatten | unique }}
  - type: vlan
```

```
    mtu: {{ internal_api_mtu }}
    vlan_id: {{ internal_api_vlan_id }}
    addresses:
    - ip_netmask: {{ internal_api_ip }}/{{ internal_api_cidr }}
    routes: {{ [internal_api_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ tenant_mtu }}
    vlan_id: {{ tenant_vlan_id }}
    addresses:
    - ip_netmask: {{ tenant_ip }}/{{ tenant_cidr }}
    routes: {{ [tenant_host_routes] | flatten | unique }}
  - type: vlan
    mtu: {{ external_mtu }}
    vlan_id: {{ external_vlan_id }}
    addresses:
    - ip_netmask: {{ external_ip }}/{{ external_cidr }}
    routes: {{ [external_host_routes, [{'default': True, 'next_hop': external_gateway_ip}]] | flatten |
unique }}
```

1. **jinja2** テンプレートで直接更新される mtu 値。
2. mtu 値は、デプロイ中に **network_data.yaml** ファイルから取得されます。

8.2.3.4. ジャンボフレームを細分化するための ML2/OVN ノースバウンドパス MTU 検出の設定

Internal ネットワーク上の仮想マシンがジャンボフレームを External ネットワークに送信する場合、Internal ネットワークの最大伝送単位 (MTU) が External ネットワークの MTU より大きいと、ノースバウンドフレームが External ネットワークの容量を容易に超過してしまいます。

ML2/OVS はこのオーバーサイズパケットの問題を自動的に処理し、ML2/OVN は TCP パケットについてこの問題を自動的に処理します。

ただし、ML2/OVN メカニズムドライバーを使用するデプロイメントで、オーバーズのノースバウンド UDP パケットを適切に処理するには、追加の設定手順を実施する必要があります。

以下の手順により、ML2/OVN ルーターが ICMP の fragmentation needed パケットを送信元の仮想マシンに返すように設定します。この場合、送信元アプリケーションはペイロードをより小さなパケットに分割することができます。

注記

East-West トラフィックでは、RHOSP ML2/OVN デプロイメント は East-West パスの最少 MTU を超えるパケットの断片化をサポートしていません。以下に例を示します。

- VM1 は、MTU が 1300 に設定された Network1 上にある。
- VM2 は、MTU が 1200 に設定された Network2 上にある。
- サイズが 1171 以下の VM1/VM2 間の ping は、どちらの方向も成功します。サイズが 1171 を超える ping は、100% パケットロスになります。
  このタイプの断片化に対するお客様の要件が特定されていないため、Red Hat はサポートを追加する予定はありません。

手順

1. ml2_conf.ini の ovn セクションに次の値を設定します。

```
ovn_emit_need_to_frag = True
```

8.2.3.5. トランキングされたインターフェイスでのネイティブ VLAN の設定

トランキングされたインターフェイスまたはボンディングに、ネイティブ VLAN を使用したネットワークがある場合には、IP アドレスはブリッジに直接割り当てられ、VLAN インターフェイスはありません。

次の例では、External ネットワークがネイティブ VLAN 上にあるボンディングインターフェイスを設定します。

```
network_config:
- type: ovs_bridge
  name: br-ex
  addresses:
  - ip_netmask: {{ external_ip }}/{{ external_cidr }}
  routes: {{ external_host_routes }}
  members:
  - type: ovs_bond
    name: bond1
    ovs_options: {{ bond_interface_ovs_options }}
    members:
    - type: interface
      name: nic3
      primary: true
    - type: interface
      name: nic4
```

注記

アドレスまたはルートのステートメントをブリッジに移動する場合は、対応する VLAN インターフェイスをそのブリッジから削除します。該当する全ロールに変更を加えます。External ネットワークはコントローラーのみに存在するため、変更する必要があるのはコントローラーのテンプレートだけです。Storage ネットワークは全ロールにアタッチされているため、Storage ネットワークがデフォルトの VLAN の場合には、全ロールを変更する必要があります。

8.2.3.6. netfilter が追跡する接続の最大数を増やす

Red Hat OpenStack Platform (RHOSP) Networking サービス (neutron) は、netfilter 接続追跡を使用してステートフルファイアウォールを構築し、仮想ネットワークでネットワークアドレス変換 (NAT) を提供します。カーネルスペースが最大接続制限に達し、**nf_conntrack: table full, dropping packet** などのエラーが発生する状況がいくつかあります。接続追跡 (conntrack) の制限を増やして、これらのタイプのエラーを回避できます。RHOSP デプロイメントで、1 つ以上のロール、またはすべてのノードの conntrack 制限を増やすことができます。

前提条件

- RHOSP アンダークラウドのインストールが成功しました。

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. source コマンドでアンダークラウドの認証情報ファイルを読み込みます。

```
$ source ~/stackrc
```

3. カスタム YAML 環境ファイルを作成します。

例

```
$ vi /home/stack/templates/custom-environment.yaml
```

4. 環境ファイルには、キーワード **parameter_defaults** および **ExtraSysctlSettings** が含まれている必要があります。netfilter が追跡できる接続の最大数の新しい値を変数 **net.nf_conntrack_max** に入力します。

例

この例では、RHOSP デプロイメント内のすべてのホストにわたって conntrack 制限を設定できます。

```
parameter_defaults:
  ExtraSysctlSettings:
    net.nf_conntrack_max:
      value: 500000
```

<role>Parameter パラメーターを使用して、特定のロールの conntrack 制限を設定します。

```
parameter_defaults:
  <role>Parameters:
    ExtraSysctlSettings:
      net.nf_conntrack_max:
        value: <simultaneous_connections>
```

- **<role>** をロールの名前に置き換えます。
  たとえば、**ControllerParameters** を使用して Controller ロールの conntrack 制限を設定するか、**ComputeParameters** を使用して Compute ロールの conntrack 制限を設定します。
- **<simultaneous_connections>** を、許可する同時接続数に置き換えます。

例

この例では、RHOSP デプロイメントの Controller ロールのみに conntrack 制限を設定できます。

```
parameter_defaults:
  ControllerParameters:
    ExtraSysctlSettings:
      net.nf_conntrack_max:
        value: 500000
```

注記

net.nf_conntrack_max のデフォルト値は **500000** 接続です。最大値は **4294967295** です。

5. コア heat テンプレート、環境ファイル、およびこの新しいカスタム環境ファイルを指定して、deployment コマンドを実行します。

重要

後で実行される環境ファイルで定義されているパラメーターとリソースが優先されることになるため、環境ファイルの順序は重要となります。

例

```
$ openstack overcloud deploy --templates \
-e /home/stack/templates/custom-environment.yaml
```

関連情報

- 環境ファイル
- オーバークラウド作成時の環境ファイルの追加

8.2.4. ネットワークインターフェイスボンディング

カスタムネットワーク設定では、さまざまなボンディングオプションを使用することができます。

8.2.4.1. オーバークラウドノードのネットワークインターフェイスボンディング

複数の物理 NIC をバンドルして、単一の論理チャネルを形成することができます。この設定はボンディングとも呼ばれます。ボンディングを設定して、高可用性システム用の冗長性またはスループットの向上を実現することができます。

Red Hat OpenStack Platform では、Open vSwitch (OVS) カーネルボンディング、OVS-DPDK ボンディング、および Linux カーネルボンディングがサポートされます。

表8.7 サポート対象のインターフェイスボンディングの種別

ボンディング種別	種別の値	許可されるブリッジ種別	許可されるメンバー
OVS カーネルボンディング	**ovs_bond**	**ovs_bridge**	**interface**
OVS-DPDK ボンディング	**ovs_dpdk_bond**	**ovs_user_bridge**	**ovs_dpdk_port**
Linux カーネルボンディング	**linux_bond**	**ovs_bridge** または **linux_bridge**	**interface**

重要

ovs_bridge と **ovs_user_bridge** を同じノード上で組み合わせないでください。

8.2.4.2. Open vSwitch (OVS) ボンディングの作成

ネットワークインターフェイステンプレートで OVS ボンディングを作成します。たとえば、OVS ユーザースペースブリッジの一部としてボンディングを作成できます。

```
- type: ovs_user_bridge
  name: br-dpdk0
  members:
  - type: ovs_dpdk_bond
    name: dpdkbond0
    rx_queue: {{ num_dpdk_interface_rx_queues }}
    members:
    - type: ovs_dpdk_port
      name: dpdk0
      members:
      - type: interface
        name: nic4
    - type: ovs_dpdk_port
      name: dpdk1
      members:
      - type: interface
        name: nic5
```

以下の例では、2 つの DPDK ポートからボンディングを作成します。

ovs_options パラメーターには、ボンディングオプションが含まれます。ネットワーク環境ファイルの **BondInterfaceOvsOptions** パラメーターを使用して、ボンディングオプションを設定することができます。

```
environment_parameters:
  BondInterfaceOvsOptions: "bond_mode=active_backup"
```

8.2.4.3. Open vSwitch (OVS) 結合オプション

NIC テンプレートファイルの **ovs_options** heat パラメーターを使用して、さまざまな Open vSwitch (OVS) ボンディングオプションを設定することができます。

bond_mode=balance-slb

送信元負荷分散 (slb) は、送信元 MAC アドレスと出力 VLAN に基づいてフローのバランスを取り、トラフィックパターンの変化に応じて定期的に再調整します。**balance-slb** ボンディングオプションを使用して結合を設定する場合は、リモートスイッチで必要な設定はありません。Networking サービス (neutron) は、ソース MAC と VLAN の各ペアをリンクに割り当て、その MAC と VLAN からのすべてのパケットをそのリンクを介して送信します。トラフィックパターンの変化に応じて定期的にリバランスを行う、送信元 MAC アドレスと VLAN の番号に基づいた簡単なハッシュアルゴリズム。**balance-slb** モードは、Linux ボンディングドライバーで使用されるモード 2 ボンドに似ています。このモードを使用すると、スイッチが LACP を使用するように設定されていない場合でも、負荷分散機能を有効にすることができます。

bond_mode=active-backup

active-backup ボンドモードを使用してボンドを設定すると、Networking サービスは 1 つの NIC をスタンバイ状態に保ちます。アクティブな接続に障害が発生すると、スタンバイ NIC がネットワー

ク操作を再開します。物理スイッチに提示される MAC アドレスは 1 つのみです。このモードはスイッチ設定を必要とせず、リンクが別のスイッチに接続されている場合に機能します。このモードは、負荷分散機能は提供しません。

lacp=[active | passive | off]

Link Aggregation Control Protocol (LACP) の動作を制御します。LACP をサポートしているのは特定のスイッチのみです。お使いのスイッチが LACP に対応していない場合には **bond_mode=balance-slb** または **bond_mode=active-backup** を使用してください。

other-config:lacp-fallback-ab=true

LACP が失敗した場合は、ボンドモードとしてアクティブバックアップを設定します。

other_config:lacp-time=[fast | slow]

LACP のハートビートを 1 秒 (高速) または 30 秒 (低速) に設定します。デフォルトは低速です。

other_config:bond-detect-mode=[miimon | carrier]

リンク検出に miimon ハートビート (miimon) またはモニターキャリア (carrier) を設定します。デフォルトは carrier です。

other_config:bond-miimon-interval=100

miimon を使用する場合には、ハートビートの間隔をミリ秒単位で設定します。

bond_updelay=1000

フラッピングを防止するためにリンクがアクティブになっている必要がある間隔 (ミリ秒) を設定します。

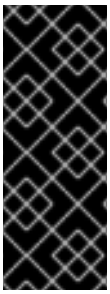
other_config:bond-rebalance-interval=10000

ボンドメンバー間でフローがリバランスする間隔 (ミリ秒) を設定します。この値をゼロに設定すると、ボンドメンバー間のフローのリバランスが無効になります。

8.2.4.4. Open vSwitch (OVS) ボンディングモードでの Link Aggregation Control Protocol (LACP) の使用

ボンディングを、オプションの Link Aggregation Control Protocol (LACP) と共に使用することができます。LACP は動的ボンディングを作成するネゴシエーションプロトコルで、これにより負荷分散機能および耐障害性を持たせることができます。

以下の表を使用して、LACP オプションと組み合わせた OVS カーネルおよび OVS-DPDK ボンディングインターフェイスのサポート互換性について説明します。



重要

Control ネットワークおよび Storage ネットワークの場合、Red Hat では VLAN を使用する Linux ボンディングを LACP と組み合わせて使用することを推奨します。OVS ボンディングを使用すると、更新、ホットフィックス等の理由により OVS または neutron エージェントが再起動すると、コントロールプレーンの中断が生じる可能性があるためです。Linux ボンディング/LACP/VLAN の設定であれば、OVS の中断を懸念することなく NIC を管理できます。

表8.8 OVS カーネルおよび OVS-DPDK ボンディングモードの LACP オプション

目的	OVS ボンディングモード	互換性のある LACP オプション	備考
高可用性 (active-passive)	**active-backup**	**active**、**passive**、または **off**	

スループットの向上 (active-active)	**balance-slb**	**active**、**passive**、または **off**	• パフォーマンスは、パケットあたりの追加パース量の影響を受けます。 • vhost-user ロック競合が生じる可能性があります。
	balance-tcp	**active** または **passive**	• balance-slb と同様に、パフォーマンスはパケットあたりの追加パース量の影響を受け、vhost-user ロック競合が生じる可能性があります。 • LACP を設定して有効にする必要があります。 • **lb-output-action=true** を設定します。以下に例を示します。 `ovs-vsctl set port <bond port> other_config:lb-output-action=true`

8.2.4.5. Linux ボンディングの作成

ネットワークインターフェイステンプレートで linux ボンディングを作成します。たとえば、2 つのインターフェイスをボンディングする linux ボンディングを作成することができます。

```
- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: {{ bond_interface_ovs_options }}
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
```

```
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
```

bonding_options パラメーターは、Linux ボンディング用の特定のボンディングオプションを設定します。

mode

ボンディングモードを設定します。この例では、**802.3ad** モードまたは LACP モードです。Linux ボンディングモードの詳細は、Red Hat Enterprise Linux 9 ネットワークの設定と管理ガイドの ボンディングモードに応じたアップストリームの切り替え設定 を参照してください。

lacp_rate

LACP パケットの送信間隔を 1 秒または 30 秒に定義します。

updelay

インターフェイスをトラフィックに使用する前にそのインターフェイスがアクティブである必要のある最低限の時間を定義します。この最小設定は、ポートフラッピングによる停止を軽減するのに役立ちます。

miimon

ドライバーの MIIMON 機能を使用してポートの状態を監視する間隔 (ミリ秒単位)

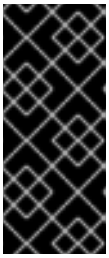
以下の追加の例をガイドとして使用し、独自の Linux ボンディングを設定します。

- 1 つの VLAN を持つ **active-backup** モードに設定された Linux ボンディング

```
....

- type: linux_bond
  name: bond_api
  mtu: {{ min_viable_mtu_ctlplane }}
  use_dhcp: false
  bonding_options: "mode=active-backup"
  dns_servers: {{ ctlplane_dns_nameservers }}
  domain: {{ dns_search_domains }}
  members:
  - type: interface
    name: nic2
    mtu: {{ min_viable_mtu_ctlplane }}
    primary: true
  - type: interface
    name: nic3
    mtu: {{ min_viable_mtu_ctlplane }}
  - type: vlan
    mtu: {{ internal_api_mtu }}
    vlan_id: {{ internal_api_vlan_id }}
    addresses:
    - ip_netmask:
        {{ internal_api_ip }}/{{ internal_api_cidr }}
      routes:
        {{ internal_api_host_routes }}
```

- OVS ブリッジ上の Linux ボンディング。1 つの VLAN を持つ **802.3ad** LACP モードに設定されたボンディング

```
- type: linux_bond
  name: bond_tenant
  mtu: {{ min_viable_mtu_ctlplane }}
  bonding_options: "mode=802.3ad updelay=1000 miimon=100"
  use_dhcp: false
  dns_servers: {{ ctlplane_dns_nameserver }}
  domain: {{ dns_search_domains }}
  members:
    - type: interface
      name: p1p1
      mtu: {{ min_viable_mtu_ctlplane }}
    - type: interface
      name: p1p2
      mtu: {{ min_viable_mtu_ctlplane }}
    - type: vlan
      mtu: {{ tenant_mtu }}
      vlan_id: {{ tenant_vlan_id }}
      addresses:
        - ip_netmask:
            {{ tenant_ip }}/{{ tenant_cidr }}
      routes:
        {{ tenant_host_routes }}
```



重要

min_viable_mtu_ctlplane を使用する前に、設定する必要があります。**/usr/share/ansible/roles/tripleo_network_config/templates/2_linux_bonds_vlans.j2** をテンプレートディレクトリーにコピーし、必要に応じて変更します。詳細は、コンポーザブルネットワーク を参照し、ネットワーク設定テンプレートに関連する手順を参照してください。

8.2.5. ネットワーク設定ファイルのフォーマットの更新

Red Hat OpenStack Platform (RHOSP) 17.0 では、ネットワーク設定の **yaml** ファイルの形式が変更されました。ネットワーク設定ファイル **network_data.yaml** の構造が変更され、NIC テンプレートファイルの形式が **yaml** ファイル形式から Jinja2 ansible 形式の **j2** に変更されました。

次の変換ツールを使用して、現在の展開の既存のネットワーク設定ファイルを RHOSP 17+ 形式に変換できます。

- **convert_v1_net_data.py**
- **convert_heat_nic_config_to_ansible_j2.py**

既存の NIC テンプレートファイルを手動で変換することもできます。

変換する必要があるファイルは次のとおりです。

- **network_data.yaml**
- コントローラー NIC テンプレート

- コンピュート NIC テンプレート
- その他のカスタムネットワークファイル

8.2.5.1. ネットワーク設定ファイルのフォーマットの更新

Red Hat OpenStack Platform (RHOSP) 17.0 では、ネットワーク設定 **yaml** ファイルの形式が変更されました。**convert_v1_net_data.py** 変換ツールを使用して、現在のデプロイメントの既存のネットワーク設定ファイルを RHOSP 17+ 形式に変換できます。

手順

1. 変換ツールをダウンロードします。
   - **/usr/share/openstack-tripleo-heat-templates/tools/convert_v1_net_data.py**
2. RHOSP 16+ ネットワーク設定ファイルを RHOSP 17+ 形式に変換します。

```
$ python3 convert_v1_net_data.py <network_config>.yaml
```

   - **<network_config>** は、変換する既存の設定ファイルの名前 (**network_data.yaml** など) に置き換えます。

8.2.5.2. NIC テンプレートの Jinja2 Ansible 形式への自動変換

Red Hat OpenStack Platform (RHOSP) 17.0 では、NIC テンプレートのファイル形式が **yaml** ファイル形式から Jinja2 Ansible 形式の **j2** に変更されました。

convert_heat_nic_config_to_ansible_j2.py 変換ツールを使用して、現在のデプロイの既存の NIC テンプレートファイルを Jinja2 形式に変換できます。

既存の NIC テンプレートファイルを手動で変換することもできます。詳細は、NIC テンプレートの Jinja2 Ansible 形式への手動変換 を参照してください。

変換する必要があるファイルは次のとおりです。

- コントローラー NIC テンプレート
- コンピュート NIC テンプレート
- その他のカスタムネットワークファイル

手順

1. アンダークラウドに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
[stack@director ~]$ source ~/stackrc
```

3. 変換ツールをアンダークラウドの現在のディレクトリーにコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-
templates/tools/convert_heat_nic_config_to_ansible_j2.py .
```

4. コンピュートとコントローラーの NIC テンプレートファイル、およびその他のカスタムネットワークファイルを Jinja2 Ansible 形式に変換します。

```
$ python3 convert_heat_nic_config_to_ansible_j2.py \
 [--stack <overcloud> | --standalone] --networks_file <network_config.yaml> \
 <network_template>.yaml
```

- **<overcloud>** は、オーバークラウドスタックの名前または UUID に置き換えてください。**--stack** が指定されていない場合、スタックはデフォルトで **overcloud** になります。

注記

--stack オプションは、アンダークラウドノードでオーケストレーションサービス (heat) を実行する必要があるため、RHOSP 16 デプロイメントでのみ使用できます。RHOSP 17 以降、RHOSP デプロイメントは、コンテナー内でオーケストレーションサービスを実行する一時的な Heat を使用します。オーケストレーションサービスが利用できない場合、またはスタックがない場合は、**--stack** の代わりに **--standalone** オプションを使用します。

- **<network_config.yaml>** は、ネットワークデプロイを記述する設定ファイルの名前 (**network_data.yaml** など) に置き換えます。
- **<network_template>** は、変換するネットワーク設定ファイルの名前に置き換えます。

すべてのカスタムネットワーク設定ファイルを変換するまで、このコマンドを繰り返します。**convert_heat_nic_config_to_ansible_j2.py** スクリプトは、変換用に渡す **yaml** ごとに **.j2** ファイルを生成します。

5. 生成された各 **.j2** ファイルを検査して、設定が環境に対して正しく、また完全であることを確認します。次に、手動で、変換できなかった設定を強調表示するツールが生成したコメントに、手動で対応します。NIC 設定を Jinja2 形式に手動で変換する方法は、Heat パラメーターから Ansible 変数へのマッピング を参照してください。

6. 生成された **.j2** ファイルを指すように、**network-environment.yaml** ファイルの ***NetworkConfigTemplate** パラメーターを設定します。

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/custom-nics/controller.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/custom-nics/compute.j2'
```

7. 古いネットワーク設定ファイルの **network-environment.yaml** ファイルから **resource_registry** マッピングを削除します。

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

8.2.5.3. NIC テンプレートの Jinja2 Ansible 形式への手動変換

Red Hat OpenStack Platform (RHOSP) 17.0 では、NIC テンプレートのファイル形式が **yaml** ファイル形式から Jinja2 Ansible 形式の **j2** に変更されました。

既存の NIC テンプレートファイルを手動で変換できます。

convert_heat_nic_config_to_ansible_j2.py 変換ツールを使用して、現在のデプロイの既存の NIC テンプレートファイルを Jinja2 形式に変換することもできます。詳細は、NIC テンプレートの Jinja2 ansible 形式への自動変換 を参照してください。

変換する必要があるファイルは次のとおりです。

- コントローラー NIC テンプレート
- コンピュート NIC テンプレート
- その他のカスタムネットワークファイル

手順

1. Jinja2 テンプレートを作成します。アンダークラウドノードの **/usr/share/ansible/roles/tripleo_network_config/templates/** ディレクトリーからサンプルテンプレートをコピーして、新しいテンプレートを作成できます。
2. Heat 固有の関数は、Jinja2 フィルターに置き換えます。たとえば、次のフィルターを使用して **min_viable_mtu** を計算します。

```
{% set mtu_list = [ctlplane_mtu] %}
{% for network in role_networks %}
{{ mtu_list.append(lookup('vars', networks_lower[network] ~ '_mtu')) }}
{%- endfor %}
{% set min_viable_mtu = mtu_list | max %}
```

3. Ansible 変数を使用して、デプロイメントのネットワークプロパティーを設定します。個々のネットワークを手動で設定するか、**role_networks** を反復して各ネットワークをプログラムで設定できます。
   - 各ネットワークを手動で設定するには、各 **get_param** 関数を同等の Ansible 変数に置き換えます。たとえば、現在のデプロイで **get_param: InternalApiNetworkVlanID** を使用して **vlan_id** を設定している場合は、次の設定をテンプレートに追加します。

```
vlan_id: {{ internal_api_vlan_id }}
```

表8.9 heat パラメーターから Ansiblevars へのネットワークプロパティーマッピングの例

yaml ファイル形式	Jinja2 ansible 形式、j2
- type: vlan device: nic2 vlan_id: get_param: InternalApiNetworkVlanID addresses: - ip_netmask: get_param: InternalApiIpSubnet	- type: vlan device: nic2 vlan_id: {{ internal_api_vlan_id }} addresses: - ip_netmask: {{ internal_api_ip }}/{{ internal_api_cidr }}

- 各ネットワークをプログラムで設定するには、**role_networks** を使用してロール名で使用可能なネットワークを取得するテンプレートに、Jinja2 for-loop 構造を追加します。

  例

```
{% for network in role_networks %}
  - type: vlan
    mtu: {{ lookup('vars', networks_lower[network] ~ '_mtu') }}
    vlan_id: {{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}
    addresses:
    - ip_netmask: {{ lookup('vars', networks_lower[network] ~ '_ip') }}/{{ lookup('vars',
networks_lower[network] ~ '_cidr') }}
    routes: {{ lookup('vars', networks_lower[network] ~ '_host_routes') }}
{%- endfor %}
```

heat パラメーターから同等の Ansible **vars** へのマッピングをすべて記載したリストについては、Heat パラメーターから Ansible 変数へのマッピング を参照してください。

4. 生成された **.j2** ファイルを指すように、**network-environment.yaml** ファイルの ***NetworkConfigTemplate** パラメーターを設定します。

```
parameter_defaults:
  ControllerNetworkConfigTemplate: '/home/stack/templates/custom-nics/controller.j2'
  ComputeNetworkConfigTemplate: '/home/stack/templates/custom-nics/compute.j2'
```

5. 古いネットワーク設定ファイルの **network-environment.yaml** ファイルから **resource_registry** マッピングを削除します。

```
resource_registry:
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-
configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-
configs/controller.yaml
```

8.2.5.4. Heat パラメーターから Ansible 変数へのマッピング

Red Hat OpenStack Platform (RHOSP) 17.x では、NIC テンプレートのファイル形式が **yaml** ファイル形式から Jinja2 ansible 形式 **j2** に変更されました。

既存の NIC テンプレートファイルを手動で Jinja2 ansible 形式に変換するには、heat パラメーターを Ansible 変数にマッピングして、デプロイメントで事前にプロビジョニングされたノードのネットワークプロパティーを設定します。**--network-config** オプションの引数を指定せずに **openstack overcloud node provision** を実行すると、heat パラメーターを Ansible 変数にマップすることもできます。

たとえば、現在のデプロイで **get_param: InternalApiNetworkVlanID** を使用して **vlan_id** を設定している場合は、新しい Jinja2 テンプレートで次の設定に置き換えます。

```
vlan_id: {{ internal_api_vlan_id }}
```

注記

--network-config オプションの引数を指定し、**openstack overcloud node provision** を実行してノードをプロビジョニングする場合は、**overcloud-baremetal-deploy.yaml** のパラメーターを使用して、デプロイ用のネットワークプロパティーを設定する必要があります。詳細は、定義ファイルマッピングをプロビジョニングする heat パラメーターを参照してください。

次の表は、heat パラメーターから同等の Ansible **vars** へのマッピングで利用できるものを一覧にしています。

表8.10 heat パラメーターから Ansible vars へのマッピング

Heat パラメーター	Ansible vars
BondInterfaceOvsOptions	**{{ bond_interface_ovs_options }}**
ControlPlaneIp	**{{ ctlplane_ip }}**
ControlPlaneDefaultRoute	**{{ ctlplane_gateway_ip }}**
ControlPlaneMtu	**{{ ctlplane_mtu }}**
ControlPlaneStaticRoutes	**{{ ctlplane_host_routes }}**
ControlPlaneSubnetCidr	**{{ ctlplane_subnet_cidr }}**
DnsSearchDomains	**{{ dns_search_domains }}**
DnsServers	**{{ ctlplane_dns_nameservers }}** **注記** この Ansible 変数には、**DEFAULT/undercloud_nameservers** および **%SUBNET_SECTION%/dns_nameservers** の **undercloud.conf** で設定された IP アドレスが入力されます。**%SUBNET_SECTION%/dns_nameservers** の設定は、**DEFAULT/undercloud_nameservers** の設定をオーバーライド。これにより、アンダークラウドおよびオーバークラウドに異なる DNS サーバーを使用し、異なるプロビジョニングサブネットのノードに異なる DNS サーバーを使用することができます。
NumDpdkInterfaceRxQueues	**{{ num_dpdk_interface_rx_queues }}**

表に記載されていない Heat パラメーターの設定

表に記載されていない heat パラメーターを設定するには、パラメーターを

{{role.name}}ExtraGroupVars として設定する必要があります。このパラメーターを **{{role.name}}ExtraGroupVars** パラメーターとして設定しないと、新しいテンプレートで使用できません。たとえば、**StorageSupernet** パラメーターを設定するには、次の設定をネットワーク設定ファイルに追加します。

```
parameter_defaults:
  ControllerExtraGroupVars:
    storage_supernet: 172.16.0.0/16
```

その後、**{{ storage_supernet }}** を Jinja2 テンプレートに追加できます。

警告

ノードのプロビジョニングで **--network-config** オプションが使用されている場合には、このプロセスは機能しません。カスタム変数を必要とするユーザーは、**--network-config** オプションを使用しないでください。代わりに、Heat スタックの作成後に、ノードのネットワーク設定を **config-download** ansible 実行に適用します。

Ansible 変数構文を変換することによる各ネットワークのプログラム設定

Jinja2 の for ループ構造を使用して、**role_networks** を反復処理し、利用可能なネットワークをロール名で取得する場合には、各ネットワークロールの小文字の名前を取得して、各プロパティーの先頭に追加する必要があります。次の構造を使用して、上記の表の Ansible **vars** を必要な構文に変換します。

{{ lookup('vars', networks_lower[network] ~ '_<property>') }}

- **<property>** は、設定するプロパティー (**ip**、**vlan_id**、または **mtu** など) に置き換えます。

たとえば、各 **NetworkVlanID** の値を動的に入力するには、**{{ <network_name>_vlan_id }}** を次の設定に置き換えます。

```
{{ lookup('vars', networks_lower[network] ~ '_vlan_id') }}`
```

8.2.5.5. 定義ファイルマッピングをプロビジョニングする heat パラメーター

openstack overcloud node provision コマンドに **--network-config** オプションの引数を指定して実行し、ノードをプロビジョニングする場合、ノード定義ファイル **overcloud-baremetal-deploy.yaml** のパラメーターを使用して、デプロイメントのネットワークプロパティーを設定する必要があります。

デプロイメントで事前にプロビジョニングされたノードを使用する場合は、heat パラメーターを Ansible 変数にマッピングして、ネットワーク属性を設定します。**--network-config** オプションの引数を指定せずに **openstack overcloud node provision** を実行すると、heat パラメーターを Ansible 変数にマップすることもできます。Ansible 変数を使用したネットワークプロパティーの設定に関する詳細は、Heat パラメーターから Ansible 変数へのマッピング を参照してください。

以下の表には、heat パラメーターからノード定義ファイル **overcloud-baremetal-deploy.yaml** に相当する **network_config** プロパティーへの利用可能なマッピングをまとめています。

表8.11 heat パラメーターからノード定義ファイル overcloud-baremetal-deploy.yaml へのマッピング

heat パラメーター	network_config プロパティー
BondInterfaceOvsOptions	**bond_interface_ovs_options**
DnsSearchDomains	**dns_search_domains**
NetConfigDataLookup	**net_config_data_lookup**
NeutronPhysicalBridge	**physical_bridge_name**
NeutronPublicInterface	**public_interface_name**
NumDpdkInterfaceRxQueues	**num_dpdk_interface_rx_queues**
{{role.name}}NetworkConfigUpdate	**network_config_update**

以下の表では、heat パラメーターからネットワーク定義ファイル **network_data.yaml** に相当するプロパティーへの利用可能なマッピングをまとめています。

表8.12 heat パラメーターからネットワーク定義ファイル network_data.yamlへのマッピング

heat パラメーター	IPv4 **network_data.yaml** プロパティー	IPv6 **network_data.yaml** プロパティー
<network_name>IpSubnet	- name: <network_name> subnets: subnet01: ip_subnet: 172.16.1.0/24	- name: <network_name> subnets: subnet01: ipv6_subnet: 2001:db8:a::/64
<network_name>NetworkVlanID	- name: <network_name> subnets: subnet01: ... vlan: <vlan_id>	- name: <network_name> subnets: subnet01: ... vlan: <vlan_id>
<network_name>Mtu	- name: <network_name> mtu:	- name: <network_name> mtu:

heat パラメーター	IPv4 **network_data.yaml** プロパティー	IPv6 **network_data.yaml** プロパティー
<network_name>InterfaceDefaultRoute	- name: <network_name> subnets: subnet01: ip_subnet: 172.16.16.0/24 gateway_ip: 172.16.16.1	- name: <network_name> subnets: subnet01: ipv6_subnet: 2001:db8:a::/64 gateway_ipv6: 2001:db8:a::1
<network_name>InterfaceRoutes	- name: <network_name> subnets: subnet01: ... routes: - destination: 172.18.0.0/24 nexthop: 172.18.1.254	- name: <network_name> subnets: subnet01: ... routes_ipv6: - destination: 2001:db8:b::/64 nexthop: 2001:db8:a::1

8.2.5.6. ネットワークデータスキーマの変更

ネットワークデータスキーマは、Red Hat OpenStack Platform (RHOSP) 17 で更新されました。RHOSP 16 以前で使用されていたネットワークデータスキーマと、RHOSP 17 以降で使用されていたネットワークデータスキーマの主な違いは次のとおりです。

- ベースサブネットは、**サブネット** マップに移動されました。これにより、スパインリーフネットワーキングなど、ルーティングされないデプロイメントとルーティングされるデプロイメントの設定が調整されます。
- 無効なネットワークを無視するために、**enabled** オプションが使用されなくなりました。代わりに、設定ファイルから無効なネットワークを削除する必要があります。
- **compat_name** オプションは、このオプションを使用していた heat リソースが削除されたため、不要になりました。
- **ip_subnet**、**gateway_ip**、**allocation_pools**、**routes**、**ipv6_subnet**、**gateway_ipv6**、**ipv6_allocation_pools**、および **routes_ipv6** のパラメーターは、ネットワークレベルでは無効になりました。これらのパラメーターは、サブネットレベルで引き続き使用されます。
- **metalsmith** で Ironic ポートを作成するために使用される新しいパラメーター **physical_network** が導入されました。
- 新しいパラメーター **network_type** および **segmentation_id** は、ネットワークタイプを **vlan** に設定するために使用される **{{network.name}}NetValueSpecs** の後継となります。
- 次のパラメーターは、RHOSP 17 で非推奨になりました。
  - **{{network.name}}NetCidr**

- **{{network.name}}SubnetName**
- **{{network.name}}Network**
- **{{network.name}}AllocationPools**
- **{{network.name}}Routes**
- **{{network.name}}SubnetCidr_{{subnet}}**
- **{{network.name}}AllocationPools_{{subnet}}**
- **{{network.name}}Routes_{{subnet}}**

第9章 ANSIBLE を使用した RED HAT OPENSTACK PLATFORM の設定と管理

Ansible を使用して、オーバークラウドの設定と登録、およびコンテナーの管理を行うことができます。

9.1. ANSIBLE ベースのオーバークラウド登録

director は、Ansible ベースのメソッドを使用して、オーバークラウドノードを Red Hat カスタマーポータルまたは Red Hat Satellite Server に登録します。

以前のバージョンの Red Hat OpenStack Platform の **rhel-registration** メソッドを使用していた場合は、それを無効にして Ansible ベースのメソッドに切り替える必要があります。詳しい情報は、「rhsm コンポーザブルサービスへの切り替え」 および 「rhel-registration から rhsm へのマッピング」 を参照してください。

director ベースの登録メソッドに加えて、デプロイメント後に手動で登録することもできます。詳細は、「手動による Ansible ベースの登録の実行」 を参照してください。

9.1.1. Red Hat Subscription Manager (RHSM) コンポーザブルサービス

rhsm コンポーザブルサービスを使用して、Ansible を介してオーバークラウドノードを登録することができます。デフォルトの **roles_data** ファイルの各ロールには、**OS::TripleO::Services::Rhsm** リソースが含まれており、これはデフォルトで無効になっています。サービスを有効にするには、このリソースを **rhsm** コンポーザブルサービスのファイルに登録します。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
```

rhsm コンポーザブルサービスは **RhsmVars** パラメーターを受け入れます。これを使用して、登録に必要な複数のサブパラメーターを定義することができます。

```
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      …
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_release: 9.2
```

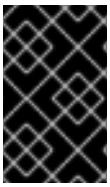
RhsmVars パラメーターをロール固有のパラメーター (例: **ControllerParameters**) と共に使用することにより、異なるノードタイプ用の特定のリポジトリーを有効化する場合に柔軟性を提供することもできます。

RhsmVars サブパラメーター

rhsm コンポーザブルサービスを設定する際に、以下のサブパラメーターを **RhsmVars** パラメーターの一部として使用します。利用可能な Ansible パラメーターについての詳細は、ロールに関するドキュメント を参照してください。

rhsm	説明
rhsm_method	登録の方法を選択します。**portal**、**satellite**、または **disable** のいずれかです。
rhsm_org_id	登録に使用する組織。この ID を特定するには、アンダークラウドノードから **sudo subscription-manager orgs** を実行します。プロンプトが表示されたら Red Hat の認証情報を入力して、出力される **Key** の値を使用します。組織の ID についての詳細は、Red Hat Subscription Management における組織 ID について理解する を参照してください。
rhsm_pool_ids	使用するサブスクリプションプール ID。サブスクリプションを自動でアタッチしない場合は、このパラメーターを使用します。この ID を特定するには、アンダークラウドノードから **sudo subscription-manager list --available --all --matches="*Red Hat OpenStack*"** を実行して、出力される **Pool ID** 値を使用します。
rhsm_activation_key	登録に使用するアクティベーションキー
rhsm_autosubscribe	このパラメーターを使用して、互換性のあるサブスクリプションを自動的にこのシステムにアタッチします。この機能を有効にするには、値を **true** に設定します。
rhsm_baseurl	コンテンツを取得するためのベース URL。デフォルトの URL は Red Hat コンテンツ配信ネットワークです。Satellite サーバーを使用している場合は、この値を Satellite サーバーコンテンツリポジトリーのベース URL に変更します。
rhsm_server_hostname	登録用のサブスクリプション管理サービスのホスト名。デフォルトは Red Hat Subscription Management のホスト名です。Satellite サーバーを使用している場合は、この値を Satellite サーバーのホスト名に変更します。
rhsm_repos	有効にするリポジトリーのリスト
rhsm_username	登録用のユーザー名。可能な場合には、登録にアクティベーションキーを使用します。
rhsm_password	登録用のパスワード。可能な場合には、登録にアクティベーションキーを使用します。
rhsm_release	リポジトリー固定用の Red Hat Enterprise Linux リリース。Red Hat OpenStack Platform の場合、このパラメーターは 9.2 に設定されます。
rhsm_rhsm_proxy_hostname	HTTP プロキシーのホスト名。たとえば、**proxy.example.com**。
rhsm_rhsm_proxy_port	HTTP プロキシー通信用のポート。たとえば、**8080**。
rhsm_rhsm_proxy_user	HTTP プロキシーにアクセスするためのユーザー名

rhsm	説明
rhsm_rhsm_proxy_password	HTTP プロキシーにアクセスするためのパスワード



重要

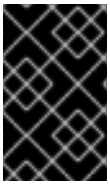
rhsm_method が **portal** に設定されている場合に限り、**rhsm_activation_key** と **rhsm_repos** を使用できます。**rhsm_method** を satellite に設定すると、**rhsm_activation_key** または **rhsm_repos** のいずれかを使用できます。

9.1.2. RhsmVars サブパラメーター

rhsm コンポーザブルサービスを設定する際に、以下のサブパラメーターを **RhsmVars** パラメーターの一部として使用します。利用可能な Ansible パラメーターについての詳細は、ロールに関するドキュメント を参照してください。

rhsm	説明
rhsm_method	登録の方法を選択します。**portal**、**satellite**、または **disable** のいずれかです。
rhsm_org_id	登録に使用する組織。この ID を特定するには、アンダークラウドノードから **sudo subscription-manager orgs** を実行します。プロンプトが表示されたら Red Hat の認証情報を入力して、出力される **Key** の値を使用します。組織の ID についての詳細は、Red Hat Subscription Management における組織 ID について理解する を参照してください。
rhsm_pool_ids	使用するサブスクリプションプール ID。サブスクリプションを自動でアタッチしない場合は、このパラメーターを使用します。この ID を特定するには、アンダークラウドノードから **sudo subscription-manager list --available --all --matches="*Red Hat OpenStack*"** を実行して、出力される **Pool ID** 値を使用します。
rhsm_activation_key	登録に使用するアクティベーションキー
rhsm_autosubscribe	このパラメーターを使用して、互換性のあるサブスクリプションを自動的にこのシステムにアタッチします。この機能を有効にするには、値を **true** に設定します。
rhsm_baseurl	コンテンツを取得するためのベース URL。デフォルトの URL は Red Hat コンテンツ配信ネットワークです。Satellite サーバーを使用している場合は、この値を Satellite サーバーコンテンツリポジトリーのベース URL に変更します。
rhsm_server_hostname	登録用のサブスクリプション管理サービスのホスト名。デフォルトは Red Hat Subscription Management のホスト名です。Satellite サーバーを使用している場合は、この値を Satellite サーバーのホスト名に変更します。

rhsm	説明
rhsm_repos	有効にするリポジトリーのリスト
rhsm_username	登録用のユーザー名。可能な場合には、登録にアクティベーションキーを使用します。
rhsm_password	登録用のパスワード。可能な場合には、登録にアクティベーションキーを使用します。
rhsm_release	リポジトリー固定用の Red Hat Enterprise Linux リリース。Red Hat OpenStack Platform の場合、このパラメーターは 9.2 に設定されます。
rhsm_rhsm_proxy_hostname	HTTP プロキシーのホスト名。たとえば、**proxy.example.com**。
rhsm_rhsm_proxy_port	HTTP プロキシー通信用のポート。たとえば、**8080**。
rhsm_rhsm_proxy_user	HTTP プロキシーにアクセスするためのユーザー名
rhsm_rhsm_proxy_password	HTTP プロキシーにアクセスするためのパスワード



重要

rhsm_method が **portal** に設定されている場合に限り、**rhsm_activation_key** と **rhsm_repos** を使用できます。**rhsm_method** を **satellite** に設定すると、**rhsm_activation_key** または **rhsm_repos** のいずれかを使用できます。

9.1.3. rhsm コンポーザブルサービスを使用したオーバークラウドの登録

rhsm コンポーザブルサービスを有効にして設定する環境ファイルを作成します。director はこの環境ファイルを使用して、ノードを登録し、サブスクライブします。

手順

1. 設定を保存するための環境ファイル (**templates/rhsm.yml**) を作成します。
2. 環境ファイルに設定を追加します。以下に例を示します。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      …
```

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

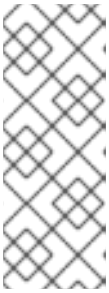
- **resource_registry** セクションは、各ロールで利用可能な **OS::TripleO::Services::Rhsm** リソースに **rhsm** コンポーザブルサービスを関連付けます。
- **RhsmVars** の変数は、Red Hat の登録を設定するためにパラメーターを Ansible に渡します。

3. **rhsm** コンポーザブルサービスをロールごとに適用するには、環境ファイルに設定を含めます。たとえば、コントローラーノード、コンピュートノード、および Ceph Storage ノードに、異なる設定セットを適用できます。

```
parameter_defaults:
  ControllerParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17.1-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
        - rhceph-6-tools-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.2
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17.1-for-rhel-9-x86_64-rpms
        - rhceph-6-tools-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.2
  CephStorageParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-rpms
        - rhel-9-for-x86_64-appstream-rpms
        - rhel-9-for-x86_64-highavailability-rpms
        - openstack-17.1-deployment-tools-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
```

```
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "68790a7aa2dc9dc50a9bc39fabc55e0d"
    rhsm_method: "portal"
    rhsm_release: 9.2
```

ControllerParameters、**ComputeParameters**、および **CephStorageParameters** パラメーターはいずれも、個別の **RhsmVars** パラメーターを使用してサブスクリプションの情報をそれぞれのロールに渡します。



注記

Red Hat Ceph Storage のサブスクリプションおよび Ceph Storage 固有のリポジトリーを使用するように、**CephStorageParameters** パラメーター内の **RhsmVars** パラメーターを設定します。**rhsm_repos** パラメーターに、コントローラーノードおよび Compute ノードに必要な Extended Update Support (EUS) リポジトリーではなく、標準の Red Hat Enterprise Linux リポジトリーが含まれるようにします。

4. 環境ファイルを保存します。

9.1.4. 異なるロールに対する rhsm コンポーザブルサービスの適用

rhsm コンポーザブルサービスをロールごとに適用することができます。たとえば、コントローラーノード、Compute ノード、および Ceph Storage ノードに、異なる設定セットを適用することができます。

手順

1. 設定を保存するための環境ファイル (**templates/rhsm.yml**) を作成します。
2. 環境ファイルに設定を追加します。以下に例を示します。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  ControllerParameters:
    RhsmVars:
      rhsm_repos:
        - rhel-9-for-x86_64-baseos-eus-rpms
        - rhel-9-for-x86_64-appstream-eus-rpms
        - rhel-9-for-x86_64-highavailability-eus-rpms
        - openstack-17.1-for-rhel-9-x86_64-rpms
        - fast-datapath-for-rhel-9-x86_64-rpms
        - rhceph-6-tools-for-rhel-9-x86_64-rpms
      rhsm_username: "myusername"
      rhsm_password: "p@55w0rd!"
      rhsm_org_id: "1234567"
      rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
      rhsm_method: "portal"
      rhsm_release: 9.2
  ComputeParameters:
    RhsmVars:
      rhsm_repos:
```

```
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      - openstack-17.1-for-rhel-9-x86_64-rpms
      - rhceph-6-tools-for-rhel-9-x86_64-rpms
      - fast-datapath-for-rhel-9-x86_64-rpms
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "55d251f1490556f3e75aa37e89e10ce5"
    rhsm_method: "portal"
    rhsm_release: 9.2
CephStorageParameters:
  RhsmVars:
    rhsm_repos:
      - rhel-9-for-x86_64-baseos-rpms
      - rhel-9-for-x86_64-appstream-rpms
      - rhel-9-for-x86_64-highavailability-rpms
      - openstack-17.1-deployment-tools-for-rhel-9-x86_64-rpms
    rhsm_username: "myusername"
    rhsm_password: "p@55w0rd!"
    rhsm_org_id: "1234567"
    rhsm_pool_ids: "68790a7aa2dc9dc50a9bc39fabc55e0d"
    rhsm_method: "portal"
    rhsm_release: 9.2
```

resource_registry は、各ロールで利用可能な **OS::TripleO::Services::Rhsm** リソースに **rhsm** コンポーザブルサービスを関連付けます。

ControllerParameters、**ComputeParameters**、および **CephStorageParameters** パラメーターはいずれも、個別の **RhsmVars** パラメーターを使用してサブスクリプションの情報をそれぞれのロールに渡します。

注記

Red Hat Ceph Storage のサブスクリプションおよび Ceph Storage 固有のリポジトリーを使用するように、**CephStorageParameters** パラメーター内の **RhsmVars** パラメーターを設定します。**rhsm_repos** パラメーターに、コントローラーノードおよび Compute ノードに必要な Extended Update Support (EUS) リポジトリーではなく、標準の Red Hat Enterprise Linux リポジトリーが含まれるようにします。

3. 環境ファイルを保存します。

9.1.5. Red Hat Satellite Server へのオーバークラウドの登録

ノードを Red Hat カスタマーポータルではなく Red Hat Satellite に登録するには、**rhsm** コンポーザブルサービスを有効にして設定する環境ファイルを作成します。

手順

1. 設定を保存するための環境ファイル (**templates/rhsm.yml**) を作成します。
2. 環境ファイルに設定を追加します。以下に例を示します。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-
templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
parameter_defaults:
  RhsmVars:
    rhsm_activation_key: "myactivationkey"
    rhsm_method: "satellite"
    rhsm_org_id: "ACME"
    rhsm_server_hostname: "satellite.example.com"
    rhsm_baseurl: "https://satellite.example.com/pulp/repos"
    rhsm_release: 9.2
```

resource_registry は、各ロールで利用可能な **OS::TripleO::Services::Rhsm** リソースに **rhsm** コンポーザブルサービスを関連付けます。

RhsmVars の変数は、Red Hat の登録を設定するためにパラメーターを Ansible に渡します。

3. 環境ファイルを保存します。

9.1.6. rhsm コンポーザブルサービスへの切り替え

従来の **rhel-registration** メソッドは、bash スクリプトを実行してオーバークラウドの登録を処理します。このメソッド用のスクリプトと環境ファイルは、**/usr/share/openstack-tripleo-heat-templates/extraconfig/pre_deploy/rhel-registration/** のコア Heat テンプレートコレクションにあります。

rhel-registration メソッドを **rhsm** コンポーザブルサービスに切り替えるには、以下の手順を実施します。

手順

1. **rhel-registration** 環境ファイルは、今後のデプロイメント操作から除外します。通常は、以下のファイルを除外します。
   - **rhel-registration/environment-rhel-registration.yaml**
   - **rhel-registration/rhel-registration-resource-registry.yaml**
2. カスタムの **roles_data** ファイルを使用する場合には、**roles_data** ファイルの各ロールに必ず **OS::TripleO::Services::Rhsm** コンポーザブルサービスを含めてください。以下に例を示します。

```
- name: Controller
  description: |
    Controller role that has all the controller services loaded and handles
    Database, Messaging and Network functions.
  CountDefault: 1
  ...
  ServicesDefault:
    ...
    - OS::TripleO::Services::Rhsm
    ...
```

3. **rhsm** コンポーザブルサービスのパラメーター用の環境ファイルを今後のデプロイメント操作に追加します。

このメソッドは、**rhel-registration** パラメーターを **rhsm** サービスのパラメーターに置き換えて、サービスを有効化する Heat リソースを変更します。

```
resource_registry:
  OS::TripleO::NodeExtraConfig: rhel-registration.yaml
```

必要に応じて、以下を行ってください。

```
resource_registry:
  OS::TripleO::Services::Rhsm: /usr/share/openstack-tripleo-heat-templates/deployment/rhsm/rhsm-baremetal-ansible.yaml
```

デプロイメントに **/usr/share/openstack-tripleo-heat-templates/environments/rhsm.yaml** 環境ファイルを追加して、サービスを有効にすることもできます。

9.1.7. rhel-registration から rhsm へのマッピング

rhel-registration メソッドから **rhsm** メソッドへの情報の移行を容易に行うには、以下の表を使用してパラメーターとその値をマッピングします。

rhel-registration	rhsm / RhsmVars
rhel_reg_method	**rhsm_method**
rhel_reg_org	**rhsm_org_id**
rhel_reg_pool_id	**rhsm_pool_ids**
rhel_reg_activation_key	**rhsm_activation_key**
rhel_reg_auto_attach	**rhsm_autosubscribe**
rhel_reg_sat_url	**rhsm_satellite_url**
rhel_reg_repos	**rhsm_repos**
rhel_reg_user	**rhsm_username**
rhel_reg_password	**rhsm_password**
rhel_reg_release	**rhsm_release**
rhel_reg_http_proxy_host	**rhsm_rhsm_proxy_hostname**
rhel_reg_http_proxy_port	**rhsm_rhsm_proxy_port**
rhel_reg_http_proxy_username	**rhsm_rhsm_proxy_user**
rhel_reg_http_proxy_password	**rhsm_rhsm_proxy_password**

9.1.8. rhsm コンポーザブルサービスを使用したオーバークラウドのデプロイ

Ansible がオーバークラウドノードの登録プロセスを制御するように、**rhsm** コンポーザブルサービスを使用してオーバークラウドをデプロイします。

手順

1. **openstack overcloud deploy** コマンドに **rhsm.yml** 環境ファイルを追加します。

```
openstack overcloud deploy \
  <other cli args> \
  -e ~/templates/rhsm.yaml
```

   これにより、Ansible のオーバークラウドの設定と、Ansible ベースの登録が有効化されます。

2. オーバークラウドのデプロイメントが完了するまで待ちます。

3. オーバークラウドノードのサブスクリプション情報を確認します。たとえば、コントローラーノードにログインして、以下のコマンドを実行します。

```
$ sudo subscription-manager status
$ sudo subscription-manager list --consumed
```

9.1.9. 手動による Ansible ベースの登録の実行

director ノードで動的インベントリースクリプトを使用して、デプロイしたオーバークラウドで、手動による Ansible ベースの登録を行うことができます。このスクリプトを使用して、ホストグループとしてノードロールを定義します。続いて **ansible-playbook** を使用して定義したノードロールに対して Playbook を実行します。コントローラーノードを手動で登録するには、以下の例の Playbook を使用します。

手順

1. 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に適用されます。

```
---
- name: Register Controller nodes
  hosts: Controller
  become: yes
  vars:
    repos:
      - rhel-9-for-x86_64-baseos-eus-rpms
      - rhel-9-for-x86_64-appstream-eus-rpms
      - rhel-9-for-x86_64-highavailability-eus-rpms
      - openstack-17.1-for-rhel-9-x86_64-rpms
      - fast-datapath-for-rhel-9-x86_64-rpms
  tasks:
    - name: Register system
      redhat_subscription:
        username: myusername
        password: p@55w0rd!
        org_id: 1234567
        release: 9.2
        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 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。

2. オーバークラウドのデプロイ後には、以下のコマンドを実行して、Ansible がオーバークラウドに対して Playbook (**ansible-osp-registration.yml**) を実行することができます。

```
$ ansible-playbook -i /usr/bin/tripleo-ansible-inventory ansible-osp-registration.yml
```

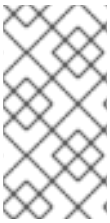
このコマンドにより、以下のアクションが行われます。

- 動的インベントリースクリプトを実行し、ホストとそのグループのリストを取得する。
- Playbook の **hosts** パラメーターで定義されているグループ (この場合はコントローラーグループ) 内のノードに、その Playbook のタスクを適用する。

9.2. ANSIBLE を使用したオーバークラウドの設定

Ansible は、オーバークラウドの設定を適用する主要な方法です。本章では、オーバークラウドの Ansible 設定を操作する方法について説明します。

director は Ansible Playbook を自動生成しますが、Ansible の構文を十分に理解しておくと役立ちます。Ansible の使用についての詳細は、Ansible のドキュメント を参照してください。



注記

Ansible でもロールの概念を使用します。これは、OpenStack Platform director のロールとは異なります。**Ansible のロール** は再利用可能な Playbook のコンポーネントを形成しますが、director のロールには OpenStack サービスのノード種別へのマッピングが含まれます。

9.2.1. Ansible ベースのオーバークラウド設定 (config-download)

director は、**config-download** 機能を使用してオーバークラウドを設定します。director は、**config-download** を OpenStack Orchestration (heat) と組み合わせて使用して、ソフトウェア設定を生成し、その設定を各オーバークラウドノードに適用します。heat は **SoftwareDeployment** リソースから全デプロイメントデータを作成して、オーバークラウドのインストールと設定を行いますが、設定の適用は一切行いません。heat は、heat API から設定データの提供のみを行います。

結果として、**openstack overcloud deploy** コマンドを実行すると、以下のプロセスが実行されます。

- director は **openstack-tripleo-heat-templates** を元に新たなデプロイメントプランを作成し、プランをカスタマイズするための環境ファイルおよびパラメーターをすべて追加します。

- director は heat を使用してデプロイメントプランを翻訳し、オーバークラウドスタックとすべての子リソースを作成します。これには、OpenStack Bare Metal サービス (ironic) を使用したノードのプロビジョニングも含まれます。
- heat はデプロイメントプランからソフトウェア設定も作成します。director はこのソフトウェア設定から Ansible Playbook をコンパイルします。
- director は、特に Ansible SSH アクセス用としてオーバークラウドノードに一時ユーザー (**tripleo-admin**) を生成します。
- director は heat ソフトウェア設定をダウンロードし、heat の出力を使用して Ansible Playbook のセットを生成します。
- director は、**ansible-playbook** を使用してオーバークラウドノードに Ansible Playbook を適用します。

9.2.2. config-download の作業ディレクトリー

ansible-playbook コマンドは、Ansible プロジェクトディレクトリーを作成します。デフォルト名は **~/config-download/overcloud** です。このプロジェクトディレクトリーには、heat からダウンロードしたソフトウェア設定が保存されます。これには、**ansible-playbook** を実行してオーバークラウドを設定するために必要なすべての Ansible 関連ファイルが含まれています。

ディレクトリーの内容は次のとおりです。

- tripleo-ansible-inventory.yaml - すべてのオーバークラウドノードの **hosts** と **vars** を含む Ansible インベントリーファイル。
- ansible.log - **ansible-playbook** の最新の実行からのログファイル。
- ansible.cfg - **ansible-playbook の** 実行時に使用される設定ファイル。
- ansible-playbook-command.sh - **ansible-playbook** の再実行に使用される実行可能スクリプト。
- ssh_private_key - オーバークラウドノードへのアクセスに使用されるプライベート ssh キー。

1. ansible-playbook の再現

プロジェクトディレクトリーが作成されたら、**ansible-playbook-command.sh** コマンドを実行してデプロイメントを再現します。

```
$ ./ansible-playbook-command.sh
```

チェックモード **--check**、ホストの制限 **--limit**、変数のオーバーライド **-e** などの追加の引数を指定してスクリプトを実行できます。

```
$ ./ansible-playbook-command.sh --check
```

9.2.3. config-download ログの確認

config-download プロセス中に、Ansible はアンダークラウドの **/home/stack** ディレクトリーに **ansible.log** という名前のログファイルを作成します。

手順

1. **less** コマンドでログを表示します。

```
$ less ~/ansible.log
```

9.2.4. 作業ディレクトリーでの Git 操作の実施

config-download の作業ディレクトリーは、ローカルの Git リポジトリーです。デプロイメント操作を実行するたびに、director は該当する変更に関する Git コミットを作業ディレクトリーに追加します。Git 操作を実施して、さまざまなステージでのデプロイメント設定を表示したり、異なるデプロイメント間で設定を比較したりすることができます。

作業ディレクトリーには制限がある点に注意してください。たとえば、Git を使用して **config-download** の作業ディレクトリーを前のバージョンに戻しても、この操作は作業ディレクトリー内の設定にしか影響を及ぼしません。したがって、以下の設定は影響を受けません。

- **オーバークラウドデータスキーマ:** 作業ディレクトリーのソフトウェア設定の前のバージョンを適用しても、データ移行およびスキーマ変更は取り消されません。
- **オーバークラウドのハードウェアレイアウト:** 以前のソフトウェア設定に戻しても、スケールアップ/ダウン等のオーバークラウドハードウェアに関する変更は取り消されません。
- **heat スタック:** 作業ディレクトリーを前のバージョンに戻しても、heat スタックに保管された設定は影響を受けません。heat スタックは新たなバージョンのソフトウェア設定を作成し、それがオーバークラウドに適用されます。オーバークラウドに永続的な変更を加えるには、**openstack overcloud deploy** コマンドを再度実行する前に、オーバークラウドスタックに適用する環境ファイルを変更します。

config-download の作業ディレクトリー内の異なるコミットを比較するには、以下の手順を実施します。

手順

1. オーバークラウドの **config-download** の作業ディレクトリー (通常は **overcloud** という名前) に移動します。

```
$ cd ~/config-download/overcloud
```

2. **git log** コマンドを実行して、作業ディレクトリー内のコミットのリストを表示します。ログの出力に日付が表示されるようにフォーマットを設定することもできます。

```
$ git log --format=format:"%h%x09%cd%x09"
a7e9063 Mon Oct 8 21:17:52 2018 +1000
dfb9d12 Fri Oct 5 20:23:44 2018 +1000
d0a910b Wed Oct 3 19:30:16 2018 +1000
...
```

   デフォルトでは、最新のコミットから順に表示されます。

3. 2 つのコミットのハッシュに対して **git diff** コマンドを実行し、デプロイメント間の違いをすべて表示します。

```
$ git diff a7e9063 dfb9d12
```

9.2.5. config-download を使用するデプロイメント方式

オーバークラウドのデプロイメントに関して、**config-download** を使用する方式は以下の 4 つに大別されます。

標準のデプロイメント

openstack overcloud deploy コマンドを実行して、プロビジョニングステージの後に設定ステージを自動的に実行します。これは、**openstack overcloud deploy** コマンドを実行する際のデフォルトの方式です。

プロビジョニングと設定の分離

特定のオプションを指定して **openstack overcloud deploy** コマンドを実行し、プロビジョニングステージと設定ステージを分離します。

デプロイメント後の ansible-playbook-command.sh スクリプトの実行

プロビジョニングステージと設定ステージを分離または組み合わせて **openstack overcloud deploy** コマンドを実行し、続いて **config-download** の作業ディレクトリーに用意されている **ansible-playbook-command.sh** スクリプトを実行し、設定ステージを再度適用します。

ノードのプロビジョニング、config-download の手動作成、および Ansible の実行

特定のオプションを指定して **openstack overcloud deploy** コマンドを実行し、ノードをプロビジョニングしてから、**deploy_steps_playbook.yaml** を指定して **ansible-playbook** コマンドを実行します。

9.2.6. 標準デプロイメントでの config-download の実行

config-download を実行するためのデフォルトの方法は、**openstack overcloud deploy** コマンドを実行することです。この方式は、ほとんどの環境に適します。

前提条件

- アンダークラウドの正常なインストール。
- デプロイ可能なオーバークラウドノード
- 実際のオーバークラウドカスタマイズに該当する Heat 環境ファイル

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. デプロイメントコマンドを実行します。オーバークラウドに必要なすべての環境ファイルを追加します。

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
```

4. デプロイメントプロセスが完了するまで待ちます。

デプロイプロセス中に、director は **config-download** ファイルを **~/config-download/overcloud** の作

業ディレクトリーに生成します。デプロイメントプロセスが終了したら、作業ディレクトリーの Ansible Playbooks を表示して、オーバークラウドを設定するために director が実行したタスクを確認します。

9.2.7. プロビジョニングと設定を分離した config-download の実行

openstack overcloud deploy コマンドは、heat ベースのプロビジョニングプロセスの後に、**config-download** 設定プロセスを実行します。各プロセスを個別に実施するように、デプロイメントコマンドを実行することもできます。独立したプロセスとしてオーバークラウドノードをプロビジョニングするには、この方式を使用します。これにより、オーバークラウドの設定プロセスを実施する前に、ノードで手動の事前設定タスクを実行することができます。

前提条件

- アンダークラウドの正常なインストール。
- デプロイ可能なオーバークラウドノード
- 実際のオーバークラウドカスタマイズに該当する Heat 環境ファイル

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. **--stack-only** オプションを指定してデプロイメントコマンドを実行します。オーバークラウドに必要なすべての環境ファイルを追加します。

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --stack-only
```

4. プロビジョニングプロセスが完了するまで待ちます。
5. **tripleo-admin** ユーザーによるアンダークラウドからオーバークラウドへの SSH アクセスを有効にします。**config-download** プロセスでは、**tripleo-admin** ユーザーを使用して Ansible ベースの設定を実施します。

```
$ openstack overcloud admin authorize
```

6. ノードで手動の事前設定タスクを実行します。設定に Ansible を使用する場合は、**tripleo-admin** ユーザーを使用してノードにアクセスします。
7. **--config-download-only** オプションを指定してデプロイメントコマンドを実行します。オーバークラウドに必要なすべての環境ファイルを追加します。

```
$ openstack overcloud deploy \
  --templates \
```

```
-e environment-file1.yaml \
-e environment-file2.yaml \
...
--config-download-only
```

8. 設定プロセスが完了するまで待ちます。

設定段階で、director は **config-download** ファイルを ~/**config-download**/**overcloud** 作業ディレクトリーに生成します。デプロイメントプロセスが終了したら、作業ディレクトリーの Ansible Playbooks を表示して、オーバークラウドを設定するために director が実行したタスクを確認します。

9.2.8. ansible-playbook-command.sh スクリプトを使用した config-download の実行

標準の方式または個別のプロビジョニングおよび設定プロセスを使用してオーバークラウドをデプロイすると、director は ~/**config-download**/**overcloud** に作業ディレクトリーを生成します。このディレクトリーには、設定プロセスを再度実行するのに必要な Playbook およびスクリプトが含まれています。

前提条件

- 以下の方式のいずれかでデプロイされたオーバークラウド
  - プロビジョニングプロセスと設定プロセスをまとめて実施する標準の方式
  - プロビジョニングプロセスと設定プロセスを分離する方式

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **ansible-playbook-command.sh** スクリプトを実行します。
   このスクリプトには追加の Ansible 引数を渡すことができ、それらの引数は、そのまま **ansible-playbook** コマンドに渡されます。これにより、チェックモード (**--check**)、ホストの限定 (**--limit**)、変数のオーバーライド (**-e**) など、Ansible の機能を更に活用することが可能となります。以下に例を示します。

```
$ ./ansible-playbook-command.sh --limit Controller
```

警告

--limit を使用して大規模にデプロイする場合、実行に含まれるホストのみがノード全体の SSH **known_hosts** ファイルに追加されます。したがって、ライブマイグレーションなどの一部の操作は、**known_hosts** ファイルにないノード間では機能しない場合があります。

注記

すべてのノードで **/etc/hosts** ファイルが最新であることを確認するには、**stack** ユーザーとして次のコマンドを実行します。

```
(undercloud)$ cd /home/stack/overcloud-deploy/overcloud/config-download/overcloud
(undercloud)$ ANSIBLE_REMOTE_USER="tripleo-admin" ansible allovercloud \
  -i /home/stack/overcloud-deploy/overcloud/tripleo-ansible-inventory.yaml \
  -m include_role \
  -a name=tripleo_hosts_entries \
  -e @global_vars.yaml
```

3. 設定プロセスが完了するまで待ちます。

関連情報

- 作業ディレクトリーには、オーバークラウドの設定タスクを管理する **deploy_steps_playbook.yaml** という名前の Playbook が含まれています。この Playbook を表示するには、以下のコマンドを実行します。

  ```
  $ less deploy_steps_playbook.yaml
  ```

  Playbook は、作業ディレクトリーに含まれているさまざまなタスクファイルを使用します。タスクファイルには、OpenStack Platform の全ロールに共通するものと、特定の OpenStack Platform ロールおよびサーバー固有のものがあります。

- 作業ディレクトリーには、オーバークラウドの **roles_data** ファイルで定義する各ロールに対応するサブディレクトリーも含まれます。以下に例を示します。

  ```
  $ ls Controller/
  ```

  各 OpenStack Platform ロールにディレクトリーには、そのロール種別の個々のサーバー用のサブディレクトリーも含まれます。これらのディレクトリーには、コンポーザブルロールのホスト名の形式を使用します。

  ```
  $ ls Controller/overcloud-controller-0
  ```

- **deploy_steps_playbook.yaml** の Ansible タスクはタグ付けされます。タグの全リストを確認するには、**ansible-playbook** で CLI オプション **--list-tags** を使用します。

  ```
  $ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags deploy_steps_playbook.yaml
  ```

  次に、**ansible-playbook-command.sh** スクリプトで **--tags**、**--skip-tags**、**--start-at-task** のいずれかを使用して、タグ付けした設定を適用します。

  ```
  $ ./ansible-playbook-command.sh --tags overcloud
  ```

4. オーバークラウドに対して **config-download** Playbook を実行すると、それぞれのホストの SSH フィンガープリントに関するメッセージが表示される場合があります。これらのメッセージを回避するには、**ansible-playbook-command.sh** スクリプトの実行時に、**--ssh-common-**

args="-o StrictHostKeyChecking=no" を追加します。

```
$ ./ansible-playbook-command.sh --tags overcloud --ssh-common-args="-o
StrictHostKeyChecking=no"
```

9.2.9. 手動で作成した Playbook を使用した config-download の実行

標準のワークフローとは別に、専用の **config-download** ファイルを作成することができます。たとえば、**--stack-only** オプションを指定して **openstack overcloud deploy** コマンドを実行し、ノードをプロビジョニングしてから、別途 Ansible 設定を手動で適用することができます。

前提条件

- アンダークラウドの正常なインストール。
- デプロイ可能なオーバークラウドノード
- 実際のオーバークラウドカスタマイズに該当する Heat 環境ファイル

手順

1. アンダークラウドホストに **stack** ユーザーとしてログインします。
2. **stackrc** ファイルを取得します。

```
$ source ~/stackrc
```

3. **--stack-only** オプションを指定してデプロイメントコマンドを実行します。オーバークラウドに必要なすべての環境ファイルを追加します。

```
$ openstack overcloud deploy \
  --templates \
  -e environment-file1.yaml \
  -e environment-file2.yaml \
  ...
  --stack-only
```

4. プロビジョニングプロセスが完了するまで待ちます。
5. **tripleo-admin** ユーザーによるアンダークラウドからオーバークラウドへの SSH アクセスを有効にします。**config-download** プロセスでは、**tripleo-admin** ユーザーを使用して Ansible ベースの設定を実施します。

```
$ openstack overcloud admin authorize
```

6. **config-download** ファイルを生成します。

```
$ openstack overcloud deploy \
  --stack overcloud --stack-only \
  --config-dir  ~/overcloud-deploy/overcloud/config-download/overcloud/
```

- **--stack** は、オーバークラウドの名前を指定します。

- **--stack-only** は、確実にコマンドが Heat スタックのみをデプロイし、ソフトウェア設定をスキップするようにします。
- **--config-dir** は、**config-download** ファイルの場所を指定します。

7. **config-download** ファイルが含まれるディレクトリーに移動します。

```
$ cd ~/config-download
```

8. 静的なインベントリーファイルを生成します。

```
$ tripleo-ansible-inventory \
  --stack <overcloud> \
  --ansible_ssh_user tripleo-admin \
  --static-yaml-inventory inventory.yaml
```

- **<overcloud>** を実際のオーバークラウドの名前に置き換えてください。

9. ~/**overcloud-deploy**/**overcloud**/**config-download**/**overcloud** ファイルと静的インベントリーファイルを使用して設定を実行します。デプロイメント用の Playbook を実行するには、**ansible-playbook** コマンドを実行します。

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  ~/overcloud-deploy/overcloud/config-download/overcloud/deploy_steps_playbook.yaml
```

注記

オーバークラウドに対して **config-download**/**overcloud** Playbook を実行すると、それぞれのホストの SSH フィンガープリントに関するメッセージが表示される場合があります。これらのメッセージを回避するには、**--ssh-common-args="-o StrictHostKeyChecking=no"** を **ansible-playbook** コマンドに追加します。

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --ssh-common-args="-o StrictHostKeyChecking=no" \
  --become \
  --tags overcloud \
  ~/overcloud-deploy/overcloud/config-
download/overcloud/deploy_steps_playbook.yaml
```

10. 設定プロセスが完了するまで待ちます。

11. ansible ベースの設定から手動で **overcloudrc** ファイルを生成します。

```
$ openstack action execution run \
```

```
--save-result \
--run-sync \
tripleo.deployment.overcloudrc \
'{"container":"overcloud"}' \
| jq -r '.["result"]["overcloudrc.v3"]' > overcloudrc.v3
```

12. デプロイメントステータスを手動で success に設定します。

```
$ openstack workflow execution create
tripleo.deployment.v1.set_deployment_status_success '{"plan": "<overcloud>"}'
```

- **<overcloud>** を実際のオーバークラウドの名前に置き換えてください。

注記

~/**overcloud-deploy**/**overcloud**/**config-download**/**overcloud**/ ディレクトリーには、Playbook **deploy_steps_playbook.yaml** が含まれています。Playbook は、作業ディレクトリーに含まれているさまざまなタスクファイルを使用します。すべての Red Hat OpenStack Platform (RHOSP) ロールに共通のタスクファイルもあれば、特定の RHOSP ロールおよびサーバーに固有のタスクファイルもあります。

~/**overcloud-deploy**/**overcloud**/**config-download**/**overcloud**/ ディレクトリーには、オーバークラウドの **roles_data** ファイルで定義した各ロールに対応するサブディレクトリーも含まれています。各 RHOSP ロールディレクトリーには、そのロールタイプの個々のサーバーのサブディレクトリーも含まれます。ディレクトリーは、**Controller**/**overcloud-controller-0** などの設定可能なロールのホスト名形式を使用します。

deploy_steps_playbook.yaml の Ansible タスクはタグ付けされます。タグの全リストを確認するには、**ansible-playbook** で CLI オプション **--list-tags** を使用します。

```
$ ansible-playbook -i tripleo-ansible-inventory.yaml --list-tags
deploy_steps_playbook.yaml
```

次に、**ansible-playbook-command.sh** スクリプトで **--tags**、**--skip-tags**、**--start-at-task** のいずれかを使用して、タグ付けした設定を適用できます。

```
$ ansible-playbook \
  -i inventory.yaml \
  -e gather_facts=true \
  -e @global_vars.yaml \
  --private-key ~/.ssh/id_rsa \
  --become \
  --tags overcloud \
  ~/overcloud-deploy/overcloud/config-
download/overcloud/deploy_steps_playbook.yaml
```

9.2.10. config-download の制限事項

config-download 機能にはいくつかの制限があります。

- **--tags**、**--skip-tags**、**--start-at-task** などの ansible-playbook CLI 引数を使用する場合には、デプロイメントの設定は、間違った順序で実行したり適用したりしないでください。これらの CLI 引数は、以前に失敗したタスクを再度実行する場合や、初回のデプロイメントを繰り返す

場合に便利な方法です。ただし、デプロイメントの一貫性を保証するには、**deploy_steps_playbook.yaml** の全タスクを順番どおりに実行する必要があります。

- タスク名に変数を使用する特定のタスクに **--start-at-task** 引数を使用することはできません。たとえば、**--start-at-task** 引数は、以下の Ansible タスクでは機能しません。

```
- name: Run puppet host configuration for step {{ step }}
```

- オーバークラウドのデプロイメントに director でデプロイされた Ceph Storage クラスターが含まれる場合、**external_deploy_steps** のタスクも省略しない限り、**--check** オプションを使用する際に **step1** のタスクを省略することはできません。

- **--forks** オプションを使用して、同時に実施する Ansible タスクの数を設定することができます。ただし、同時タスクが 25 を超えると、**config-download** 操作のパフォーマンスが低下します。このため、**--forks** オプションに 25 を超える値を設定しないでください。

9.2.11. config-download の主要ファイル

config-download の作業ディレクトリー内の主要なファイルを以下に示します。

Ansible の設定および実行

config-download の作業ディレクトリー内の以下のファイルは、Ansible を設定/実行するための専用ファイルです。

ansible.cfg
: **ansible-playbook** 実行時に使用する設定ファイル

ansible.log
: 最後に実行した **ansible-playbook** に関するログファイル

ansible-errors.json
: デプロイメントエラーが含まれる JSON 構造のファイル

ansible-playbook-command.sh
: 最後のデプロイメント操作の **ansible-playbook** コマンドを再実行するための実行可能スクリプト

ssh_private_key
: Ansible がオーバークラウドノードにアクセスする際に使用する SSH 秘密鍵

tripleo-ansible-inventory.yaml
: すべてのオーバークラウドノードのホストおよび変数が含まれる Ansible インベントリーファイル

overcloud-config.tar.gz
: 作業ディレクトリーのアーカイブ

Playbook

以下のファイルは、**config-download** の作業ディレクトリー内の Playbook です。

deploy_steps_playbook.yaml
: デプロイメントのメインステップ。この Playbook により、オーバークラウド設定の主要な操作が実施されます。

pre_upgrade_rolling_steps_playbook.yaml
: メジャーアップグレードのための事前アップグレードステップ

upgrade_steps_playbook.yaml

メジャーアップグレードのステップ

post_upgrade_steps_playbook.yaml

メジャーアップグレードに関するアップグレード後ステップ

update_steps_playbook.yaml

マイナー更新のステップ

fast_forward_upgrade_playbook.yaml

Fast Forward Upgrade のタスク。Red Hat OpenStack Platform のロングライフバージョンから次のロングライフバージョンにアップグレードする場合にのみ、この Playbook 使用します。

9.2.12. config-download のタグ

Playbook では、オーバークラウドに適用されるタスクを管理するのにタグ付けされたタスクを使用します。**ansible-playbook** CLI の引数 **--tags** または **--skip-tags** でタグを使用して、実行するタスクを管理します。デフォルトで有効なタグに関する情報を、以下のリストに示します。

facts

ファクト収集操作

common_roles

すべてのノードに共通な Ansible ロール

overcloud

オーバークラウドデプロイメント用のすべてのプレイ

pre_deploy_steps

deploy_steps の操作の前に実施されるデプロイメント

host_prep_steps

ホスト準備のステップ

deploy_steps

デプロイメントのステップ

post_deploy_steps

deploy_steps の操作の後に実施される手順

external

すべての外部デプロイメントタスク

external_deploy_steps

アンダークラウドでのみ実行される外部デプロイメントタスク

9.2.13. config-download のデプロイメントステップ

deploy_steps_playbook.yaml Playbook により、オーバークラウドが設定されます。この Playbook により、オーバークラウドデプロイメントプランに基づき完全なオーバークラウドをデプロイするのに必要なすべてのソフトウェア設定が適用されます。

本項では、この Playbook で使用されるさまざまな Ansible プレイの概要について説明します。本項のプレイと同じ名前が、Playbook 内で使用され **ansible-playbook** の出力にも表示されます。本項では、それぞれのプレイに設定される Ansible タグについても説明します。

Gather facts from undercloud

アンダークラウドノードからファクトを収集します。
タグ: **facts**

Gather facts from overcloud

オーバークラウドノードからファクトを収集します。
タグ: **facts**

Load global variables

global_vars.yaml からすべての変数を読み込みます。
タグ: **always**

Common roles for TripleO servers

共通の Ansible ロールをすべてのオーバークラウドノードに適用します。これには、ブートストラップパッケージをインストールする tripleo-bootstrap および ssh の既知のホストを設定する tripleo-ssh-known-hosts が含まれます。
タグ: **common_roles**

Overcloud deploy step tasks for step 0

deploy_steps_tasks テンプレートインターフェイスからのタスクを適用します。
タグ: **overcloud**、**deploy_steps**

Server deployments

ネットワーク設定や hieradata 等の設定に、サーバー固有の heat デプロイメントを適用します。これには、NetworkDeployment、<Role>Deployment、<Role>AllNodesDeployment 等が含まれます。
タグ: **overcloud**、**pre_deploy_steps**

Host prep steps

host_prep_steps テンプレートインターフェイスからのタスクを適用します。
タグ: **overcloud**、**host_prep_steps**

External deployment step [1,2,3,4,5]

external_deploy_steps_tasks テンプレートインターフェイスからのタスクを適用します。Ansible は、アンダークラウドノードに対してのみこれらのタスクを実行します。
タグ: **external**、**external_deploy_steps**

Overcloud deploy step tasks for [1,2,3,4,5]

deploy_steps_tasks テンプレートインターフェイスからのタスクを適用します。
タグ: **overcloud**、**deploy_steps**

Overcloud common deploy step tasks [1,2,3,4,5]

puppet ホストの設定、**container-puppet.py**、および **tripleo-container-manage** (コンテナーの設定と管理) など、各ステップで実行される一般的なタスクを適用します。
タグ: **overcloud**、**deploy_steps**

Server Post Deployments

5 ステップのデプロイメントプロセス後に実施される設定に、サーバー固有の heat デプロイメントを適用します。
タグ: **overcloud**、**post_deploy_steps**

External deployment Post Deploy tasks

external_post_deploy_steps_tasks テンプレートインターフェイスからのタスクを適用します。Ansible は、アンダークラウドノードに対してのみこれらのタスクを実行します。

タグ: external、external_deploy_steps

9.3. ANSIBLE を使用したコンテナーの管理

Red Hat OpenStack Platform 17.1 は、**tripleo_container_manage** Ansible ロールを使用してコンテナーの管理操作を実行します。カスタム Playbook を作成して、特定のコンテナー管理操作を実行することもできます。

- heat が生成するコンテナー設定データを収集する。**tripleo_container_manage** ロールは、このデータを使用してコンテナーのデプロイメントをオーケストレーションします。
- コンテナーを起動する。
- コンテナーを停止する。
- コンテナーを更新する。
- コンテナーを削除する。
- 特定の設定でコンテナーを実行する。

director はコンテナー管理を自動的に実施しますが、コンテナー設定をカスタマイズしなければならない場合や、オーバークラウドを再デプロイせずにコンテナーにホットフィックスを適用しなければならない場合があります。

注記

このロールがサポートするのは Podman コンテナー管理だけです。

9.3.1. tripleo-container-manage ロールのデフォルトと変数

次の抜粋は、**tripleo_container_manage** Ansible ロールのデフォルトと変数を示しています。

```
# All variables intended for modification should place placed in this file.
tripleo_container_manage_hide_sensitive_logs: '{{ hide_sensitive_logs | default(true)
  }}'
tripleo_container_manage_debug: '{{ ((ansible_verbosity | int) >= 2) | bool }}'
tripleo_container_manage_clean_orphans: true

# All variables within this role should have a prefix of "tripleo_container_manage"
tripleo_container_manage_check_puppet_config: false
tripleo_container_manage_cli: podman
tripleo_container_manage_concurrency: 1
tripleo_container_manage_config: /var/lib/tripleo-config/
tripleo_container_manage_config_id: tripleo
tripleo_container_manage_config_overrides: {}
tripleo_container_manage_config_patterns: '*.json'
# Some containers where Puppet is run, can take up to 10 minutes to finish
# in slow environments.
tripleo_container_manage_create_retries: 120
# Default delay is 5s so 120 retries makes a timeout of 10 minutes which is
# what we have observed a necessary value for nova and neutron db-sync execs.
tripleo_container_manage_exec_retries: 120
```

```
tripleo_container_manage_healthcheck_disabled: false
tripleo_container_manage_log_path: /var/log/containers/stdouts
tripleo_container_manage_systemd_teardown: true
```

9.3.2. tripleo-container-manage molecule のシナリオ

Molecule は **tripleo_container_manage** ロールをテストするために使用されます。以下は、**molecule** のデフォルトのインベントリーを示しています。

```
hosts:
  all:
    hosts:
      instance:
        ansible_host: localhost
        ansible_connection: local
        ansible_distribution: centos8
```

使用方法

Red Hat OpenStack 17.1 は、このロールで Podman のみをサポートします。Docker のサポートはロードマップ上にあります。

Molecule Ansible ロールは、次のタスクを実行します。

- TripleO Heat テンプレートによって生成されたコンテナー設定データを収集します。このデータは、信頼できる情報源として使用されます。コンテナーがすでに **Molecule** で管理されている場合には、現在の状態に関係なく、設定データは必要に応じてコンテナーを再設定します。
- **systemd** シャットダウンファイルを管理します。ノードのシャットダウン時または起動時のサービス注文に必要な **TripleO Container systemd** サービスを作成します。また、**netns-placeholder** サービスも管理します。
- 不要になったコンテナーまたは再設定が必要なコンテナーを削除します。コンテナーを削除する必要があるかどうかを判断するための一連のルールが含まれる、**needs_delete ()** という名前のカスタムフィルターを使用します。
  - コンテナーが **tripleo_ansible** によって管理されていない場合、またはコンテナーの **config_id** が入力 ID と一致しない場合には、コンテナーは削除されません。
  - コンテナーに **config_data** がない場合、またはコンテナーに入力データと一致しない **config_data** がある場合には、コンテナーは削除されます。コンテナーが削除されると、そのロールも **systemd** サービスとヘルスチェックを無効にして削除することに注意してください。
- **start_order** コンテナー設定で定義された特定の順序でコンテナーを作成します。デフォルトは 0 です。
  - コンテナーが **exec** の場合には、複数の **exec** を **同時に実行できるように async を** 使用して、exec 専用の Playbook が実行されます。
  - それ以外の場合、**podman_container** が非同期で使用され、コンテナーが作成されます。コンテナーに **再起動ポリシー** がある場合、**systemd** サービスが設定されます。コンテナーにヘルスチェックスクリプトがある場合には、**systemd healthcheck** サービスが設定されます。

注記

tripleo_container_manage_concurrency パラメーターはデフォルトで 1 に設定されており、2 より大きい値を設定すると、Podman ロックに関する問題が発生する可能性があります。

Playbook の例:

```
- name: Manage step_1 containers using tripleo-ansible
  block:
    - name: "Manage containers for step 1 with tripleo-ansible"
      include_role:
        name: tripleo_container_manage
      vars:
        tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-config/step_1"
        tripleo_container_manage_config_id: "tripleo_step1"
```

9.3.3. tripleo_container_manage ロールの変数

tripleo_container_manage Ansible ロールには、以下の変数が含まれます。

表9.1 ロール変数

名前	デフォルト値	説明
tripleo_container_manage_check_puppet_config	false	Ansible で Puppet コンテナー設定を確認する場合は、この変数を使用します。Ansible は、設定ハッシュを使用して更新されたコンテナー設定を識別することができます。コンテナーに Puppet からの新規設定がある場合は、この変数を **true** に設定します。これにより、Ansible は新規設定を検出し、Ansible が再起動しなければならないコンテナーリストにコンテナーを追加することができます。
tripleo_container_manage_cli	podman	この変数を使用して、コンテナーを管理するのに使用するコマンドラインインターフェイスを設定します。**tripleo_container_manage** ロールがサポートするのは Podman だけです。
tripleo_container_manage_concurrency	1	この変数を使用して、同時に管理するコンテナーの数を設定します。

名前	デフォルト値	説明
tripleo_container_manage_config	/var/lib/tripleo-config/	この変数を使用して、コンテナー設定ディレクトリーへのパスを設定します。
tripleo_container_manage_config_id	tripleo	この変数を使用して、特定の設定ステップの ID を設定します。たとえば、デプロイメントのステップ 2 のコンテナーを管理するには、この値を **tripleo_step2** に設定します。
tripleo_container_manage_config_patterns	*.json	この変数を使用して、コンテナー設定ディレクトリーの設定ファイルを識別する bash 正規表現を設定します。
tripleo_container_manage_debug	false	この変数を使用して、デバッグモードを有効または無効にします。特定の一度限りの設定でコンテナーを実行する場合、コンテナーのライフサイクルを管理するコンテナーコマンドを出力する場合、またはテストおよび検証目的で操作を行わずにコンテナー管理を実施する場合に、**tripleo_container_manage** ロールをデバッグモードで実行します。
tripleo_container_manage_healthcheck_disable	false	この変数を使用して、ヘルスチェックを有効または無効にします。
tripleo_container_manage_log_path	/var/log/containers/stdouts	この変数を使用して、コンテナーの stdout ログパスを設定します。
tripleo_container_manage_systemd_order	false	この変数を使用して、Ansible による systemd のシャットダウン順序を有効または無効にします。
tripleo_container_manage_systemd_teardown	true	この変数を使用して、使用されなくなったコンテナーのクリーンアップをトリガーします。

名前	デフォルト値	説明
tripleo_container_manage_config_overrides	{}	この変数を使用して、コンテナー設定をオーバーライドします。この変数では、値にディクショナリー形式が使用されます。ここで、それぞれのキーはコンテナー名およびオーバーライドするパラメーター (例: コンテナーイメージ、ユーザー) です。この変数は、JSON コンテナー設定ファイルにカスタムオーバーライドを書き込みません。したがって、コンテナーが新たにデプロイ、更新、またはアップグレードされると、JSON 設定ファイルの内容に戻ります。
tripleo_container_manage_valid_exit_code	[]	この変数を使用して、コンテナーが終了コードを返すかどうかを確認します。この値はリストにする必要があります (例: **[0,3]**)。

9.3.4. tripleo-container-manage ヘルスチェック

Red Hat OpenStack 17.1 までは、コンテナーのヘルスチェックは **systemd** タイマーで実装され、**podman exec** を実行して特定のコンテナーが正常かどうかを判断していました。現在は、Podman のネイティブの **healthcheck** インターフェイスを使用しており、統合と使用がさらに簡単になっています。

コンテナー (keystone など) が正常かどうかを確認するには、次のコマンドを実行します。

```
$ sudo podman healthcheck run keystone
```

戻りコードは **0** および **“healthy”** である必要があります。

```
"Healthcheck": {
    "Status": "healthy",
    "FailingStreak": 0,
    "Log": [
        {
            "Start": "2020-04-14T18:48:57.272180578Z",
            "End": "2020-04-14T18:48:57.806659104Z",
            "ExitCode": 0,
            "Output": ""
        },
        (...)
    ]
}
```

9.3.5. tripleo-container-manage デバッグ

tripleo_container_manage Ansible ロールを使用すると、特定のコンテナーに対して特定のアクションを実行できます。これは次の目的で使用できます。

- 特定の 1 回限りの設定でコンテナーを実行します。
- コンテナーのライフサイクル管理用のコンテナーコマンドを出力します。
- Ansible がコンテナーに加えられた変更を出力します。

注記

1 つのコンテナーを管理するには、次の 2 つのことを知っておく必要があります。

- オーバークラウドのデプロイ中のどのステップでコンテナーがデプロイされたか。
- コンテナー設定を含む、生成された JSON ファイルの名前。

以下は、イメージ設定をオーバーライドする **ステップ 1** で **HAproxy** コンテナーを管理する Playbook の例です。

```
- hosts: localhost
  become: true
  tasks:
    - name: Manage step_1 containers using tripleo-ansible
      block:
        - name: "Manage HAproxy container at step 1 with tripleo-ansible"
          include_role:
            name: tripleo_container_manage
          vars:
            tripleo_container_manage_config_patterns: 'haproxy.json'
            tripleo_container_manage_config: "/var/lib/tripleo-config/container-startup-config/step_1"
            tripleo_container_manage_config_id: "tripleo_step1"
            tripleo_container_manage_clean_orphans: false
            tripleo_container_manage_config_overrides:
              haproxy:
                image: quay.io/tripleoRed_Hat_OpenStack_Platform-17.1-
Customizing_your_Red_Hat_OpenStack_Platform_deployment.entos9/centos-binary-haproxy:hotfix
```

Ansible が **check mode** で実行されている場合に、コンテナーは削除または作成されませんが、Playbook の実行の最後にコマンドのリストが表示され、Playbook で出される可能性のある結果が示されます。これはデバッグに役立ちます。

```
$ ansible-playbook haproxy.yaml --check
```

diff mode を追加すると、Ansible によってコンテナーに加えられた変更が表示されます。

```
$ ansible-playbook haproxy.yaml --check --diff
```

tripleo_container_manage_clean_orphans パラメーターはオプションです。false に設定できます。これは、固有の **config_id** が割り当てられた、孤立したコンテナーが削除されないことを意味します。このパラメーターを使用して、**config_id** が同じ、別の実行中のコンテナーに影響を与えずに、1 つのコンテナーを管理できます。

tripleo_container_manage_config_overrides パラメーターはオプションであり、イメージやコンテナーユーザーなどの特定のコンテナー属性をオーバーライドするために使用できます。このパラメーターは、コンテナー名とオーバーライドするパラメーターを使用してディクショナリーを作成します。これらのパラメーターは存在する必要があり、TripleO Heat テンプレートでコンテナー設定を定義します。

ディクショナリーは JSON ファイルのオーバーライドを更新しないことに注意してください。そのため、更新またはアップグレードが実行された場合に、コンテナーは JSON ファイルの設定で再構成されます。

第10章 オーケストレーションサービス (HEAT) を使用したオーバークラウドの設定

オーケストレーションサービス (heat) を使用して、heat テンプレートと環境ファイルにカスタムのオーバークラウド設定を作成できます。

10.1. HEAT テンプレートの概要

本章のカスタム設定では、heat テンプレートおよび環境ファイルを使用して、オーバークラウドの特定の機能を定義します。本項には、Red Hat OpenStack Platform director に関連した heat テンプレートの構造や形式を理解するための基本的な説明を記載します。

10.1.1. heat テンプレート

director は、Heat Orchestration Template (HOT) をオーバークラウドデプロイメントプランのテンプレート形式として使用します。HOT 形式のテンプレートは、通常 YAML 形式で表現されます。テンプレートの目的は、OpenStack Orchestration (heat) が作成するリソースのコレクションであるスタックを定義および作成し、リソースを設定することです。リソースとは、コンピュートリソース、ネットワーク設定、セキュリティーグループ、スケーリングルール、カスタムリソースなどの Red Hat OpenStack Platform (RHOSP) のオブジェクトを指します。

heat テンプレートは、3 つの主要なセクションで設定されます。

parameters

これらは、heat に渡される設定 (スタックのカスタマイズが可能) およびパラメーターのデフォルト値 (値を渡さない場合) です。これらの設定がテンプレートの **parameters** セクションで定義されます。

resources

resources セクションを使用して、このテンプレートを使用してスタックをデプロイする際に作成することができるリソース (コンピュートインスタンス、ネットワーク、ストレージボリューム等) を定義します。Red Hat OpenStack Platform (RHOSP) には、全コンポーネントに対応するコアリソースのセットが含まれています。これらは、スタックの一部として作成/設定する固有のオブジェクトです。RHOSP には、全コンポーネントに対応するコアリソースのセットが含まれています。これらがテンプレートの **resources** セクションで定義されます。

outputs

outputs セクションを使用して、スタックの作成後にクラウドユーザーがアクセスできるアウトプットパラメーターを宣言します。クラウドユーザーはこれらのパラメーターを使用して、デプロイしたインスタンスの IP アドレスやスタックの一部としてデプロイされた Web アプリケーションの URL 等のスタックの詳細を要求することができます。

基本的な heat テンプレートの例:

```
heat_template_version: 2013-05-23

description: > A very basic Heat template.

parameters:
  key_name:
    type: string
    default: lars
    description: Name of an existing key pair to use for the instance
  flavor:
```

```
    type: string
    description: Instance type for the instance to be created
    default: m1.small
  image:
    type: string
    default: cirros
    description: ID or name of the image to use for the instance

resources:
  my_instance:
    type: OS::Nova::Server
    properties:
      name: My Cirros Instance
      image: { get_param: image }
      flavor: { get_param: flavor }
      key_name: { get_param: key_name }

output:
  instance_name:
    description: Get the instance's name
    value: { get_attr: [ my_instance, name ] }
```

このテンプレートは、リソース種別 **type: OS::Nova::Server** を使用して、クラウドユーザーが指定する特定のフレーバー、イメージ、およびキーで **my_instance** というインスタンスを作成します。このスタックは、**My Cirros Instance** という **instance_name** の値を返すことができます。

heat がテンプレートを処理する際には、テンプレートのスタックとリソーステンプレートの子スタックセットを作成します。これにより、テンプレートで定義するメインのスタックを最上位とするスタックの階層が作成されます。以下のコマンドを使用して、スタックの階層を表示することができます。

```
$ openstack stack list --nested
```

10.1.2. 環境ファイル

環境ファイルとは特別な種類のテンプレートで、これを使用して heat テンプレートをカスタマイズすることができます。コアの heat テンプレートに加えて、環境ファイルをデプロイメントコマンドに追加することができます。環境ファイルには、3 つの主要なセクションが含まれます。

resource_registry

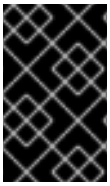
このセクションでは、他の heat テンプレートにリンクしたカスタムのリソース名を定義します。これにより、コアリソースコレクションに存在しないカスタムのリソースを作成することができます。

parameters

これらは、最上位のテンプレートのパラメーターに適用する共通設定です。たとえば、入れ子状のスタックをデプロイするテンプレートの場合には (リソースレジストリーマッピング等)、パラメーターは最上位のテンプレートにのみ適用され、入れ子状のリソースのテンプレートには適用されません。

parameter_defaults

これらのパラメーターは、全テンプレートのパラメーターのデフォルト値を変更します。たとえば、入れ子状のスタックをデプロイする heat テンプレートの場合には (リソースレジストリーマッピングなど)、パラメーターのデフォルト値がすべてのテンプレートに適用されます。



重要

パラメーターがオーバークラウドのすべてのスタックテンプレートに適用されるように、オーバークラウド用にカスタムの環境ファイルを作成する場合には、**parameters** ではなく **parameter_defaults** を使用します。

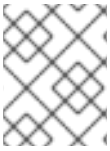
基本的な環境ファイルの例:

```
resource_registry:
  OS::Nova::Server::MyServer: myserver.yaml

parameter_defaults:
  NetworkName: my_network

parameters:
  MyIP: 192.168.0.1
```

特定の heat テンプレート (**my_template.yaml**) からスタックを作成する際に、この環境ファイル (**my_env.yaml**) を追加します。**my_env.yaml** ファイルにより、**OS::Nova::Server::MyServer** という新しいリソース種別が作成されます。**myserver.yaml** ファイルは、このリソース種別を実装する heat テンプレートファイルで、このファイルでの設定が元の設定よりも優先されます。**my_template.yaml** ファイルに **OS::Nova::Server::MyServer** リソースを含めることができます。

MyIP は、この環境ファイルと共にデプロイを行うメインの heat テンプレートにしかパラメーターを適用しません。この例では、**MyIP** は **my_template.yaml** のパラメーターにのみ適用します。

NetworkName はメインの heat テンプレート (**my_template.yaml**) とメインのテンプレートに含まれるリソースに関連付けられたテンプレート (上記の例では **OS::Nova::Server::MyServer** リソースとその **myserver.yaml** テンプレート) の両方に適用されます。



注記

RHOSP が heat テンプレートファイルをカスタムテンプレートリソースとして使用するには、ファイルの拡張子を .yaml または .template のいずれかにする必要があります。

10.1.3. オーバークラウドのコア heat テンプレート

director には、オーバークラウド用のコア heat テンプレートコレクションおよび環境ファイルコレクションが含まれます。このコレクションは、**/usr/share/openstack-tripleo-heat-templates** に保存されています。

このテンプレートコレクションの主なファイルおよびディレクトリーは、以下のとおりです。

overcloud.j2.yaml

オーバークラウド環境を作成するのに director が使用するメインのテンプレートファイル。このファイルでは Jinja2 構文を使用してテンプレートの特定セクションを繰り返し、カスタムロールを作成します。Jinja2 フォーマットは、オーバークラウドのデプロイメントプロセス中に YAML にレンダリングされます。

overcloud-resource-registry-puppet.j2.yaml

オーバークラウド環境を作成するのに director が使用するメインの環境ファイル。このファイルは、オーバークラウドイメージ上に保存される Puppet モジュールの設定セットを提供します。director により各ノードにオーバークラウドのイメージが書き込まれると、heat はこの環境ファイルに登録されているリソースを使用して各ノードの Puppet 設定を開始します。このファイルでは

Jinja2 構文を使用してテンプレートの特定セクションを繰り返し、カスタムロールを作成します。Jinja2 フォーマットは、オーバークラウドのデプロイメントプロセス中に YAML にレンダリングされます。

roles_data.yaml

このファイルにはオーバークラウド内のロールの定義が含まれ、サービスを各ロールにマッピングします。

network_data.yaml

このファイルには、オーバークラウド内のネットワーク定義、およびそれらのサブネット、割り当てプール、仮想 IP のステータス等の属性が含まれます。デフォルトの **network_data.yaml** ファイルにはデフォルトのネットワーク (External、Internal Api、Storage、Storage Management、Tenant、および Management) が含まれます。カスタムの **network_data.yaml** ファイルを作成して、**openstack overcloud deploy** コマンドに **-n** オプションで追加することができます。

plan-environment.yaml

このファイルには、オーバークラウドプランのメタデータの定義が含まれます。これには、プラン名、使用するメインのテンプレート、およびオーバークラウドに適用する環境ファイルが含まれます。

capabilities-map.yaml

このファイルには、オーバークラウドプランの環境ファイルのマッピングが含まれます。

デプロイメント

このディレクトリーには、heat テンプレートが含まれます。**overcloud-resource-registry-puppet.j2.yaml** 環境ファイルは、このディレクトリーのファイルを使用して、各ノードに Puppet の設定が適用されるようにします。

environments

このディレクトリーには、オーバークラウドの作成に使用可能なその他の heat 環境ファイルが含まれます。これらの環境ファイルは、作成された Red Hat OpenStack Platform (RHOSP) 環境の追加の機能を有効にします。たとえば、ディレクトリーには Cinder NetApp のバックエンドストレージ (**cinder-netapp-config.yaml**) を有効にする環境ファイルが含まれています。

network

このディレクトリーには、分離ネットワークおよびポートを作成するのに使用できる heat テンプレートのセットが含まれます。

puppet

このディレクトリーには、Puppet 設定を制御するテンプレートが含まれます。**overcloud-resource-registry-puppet.j2.yaml** 環境ファイルは、このディレクトリーのファイルを使用して、各ノードに Puppet の設定が適用されるようにします。

puppet/services

このディレクトリーには、全サービス設定用のレガシー heat テンプレートが含まれます。**puppet/services** ディレクトリー内のほとんどのテンプレートが、**deployment** ディレクトリーのテンプレートに置き換えられています。

extraconfig

このディレクトリーには、追加機能を有効にするのに使用できるテンプレートが含まれます。

10.1.4. オーバークラウド作成時の環境ファイルの追加

-e オプションを使用して、デプロイメントコマンドに環境ファイルを追加します。必要に応じていくつでも環境ファイルを追加することができます。ただし、後で指定する環境ファイルで定義されるパラメーターとリソースが優先されることになるため、環境ファイルの順番は重要です。この例では、両環境ファイルに共通のリソース種別 (**OS::TripleO::NodeExtraConfigPost**) と共通のパラメーター (**TimeZone**) が含まれています。

environment-file-1.yaml

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/template-1.yaml

parameter_defaults:
  RabbitFDLimit: 65536
  TimeZone: 'Japan'
```

environment-file-2.yaml

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/template-2.yaml

parameter_defaults:
  TimeZone: 'Hongkong'
```

デプロイメントコマンドに両方の環境ファイルを含めます。

```
$ openstack overcloud deploy --templates -e environment-file-1.yaml -e environment-file-2.yaml
```

openstack overcloud deploy コマンドは、以下のプロセスを順に実行します。

1. コア heat テンプレートコレクションからデフォルト設定を読み込みます。
2. **environment-file-1.yaml** の設定を適用します。この設定により、デフォルト設定と共通している設定は上書きされます。
3. **environment-file-2.yaml** の設定を適用します。この設定により、デフォルト設定および **environment-file-1.yaml** と共通している設定は上書きされます。

これにより、オーバークラウドのデフォルト設定が以下のように変更されます。

- **OS::TripleO::NodeExtraConfigPost** リソースは、**environment-file-2.yaml** で指定されているとおりに **/home/stack/templates/template-2.yaml** に設定されます。
- **TimeZone** パラメーターは、**environment-file-2.yaml** で指定されているとおりに **Hongkong** に設定されます。
- **RabbitFDLimit** パラメーターは、**environment-file-1.yaml** に定義されるように **65536** に設定されます。**environment-file-2.yaml** はこの値を変更しません。

この手法を使用して、複数の環境ファイルの値が競合することなく、オーバークラウドのカスタム設定を定義することができます。

10.1.5. カスタムのコア heat テンプレートの使用

オーバークラウドの作成時に、director は **/usr/share/openstack-tripleo-heat-templates** にある heat テンプレートのコアセットを使用します。このコアテンプレートコレクションをカスタマイズする場合は、以下の git ワークフローを使用してカスタムテンプレートコレクションを管理します。

手順

- heat テンプレートコレクションが含まれる初期 git リポジトリーを作成します。

a. テンプレートコレクションを **/home/stack/templates** ディレクトリーにコピーします。

```
$ cd ~/templates
$ cp -r /usr/share/openstack-tripleo-heat-templates .
```

b. カスタムテンプレートのディレクトリーに移動して git リポジトリーを初期化します。

```
$ cd ~/templates/openstack-tripleo-heat-templates
$ git init .
```

c. Git のユーザー名およびメールアドレスを設定します。

```
$ git config --global user.name "<USER_NAME>"
$ git config --global user.email "<EMAIL_ADDRESS>"
```

- **<USER_NAME>** を使用するユーザー名に置き換えます。
- **<EMAIL_ADDRESS>** をご自分のメールアドレスに置き換えます。

a. 初期コミットに向けて全テンプレートをステージします。

```
$ git add *
```

b. 初期コミットを作成します。

```
$ git commit -m "Initial creation of custom core heat templates"
```

これで、最新のコアテンプレートコレクションを格納する初期 **master** ブランチが作成されます。このブランチは、カスタムブランチのベースとして使用し、新規テンプレートバージョンをこのブランチにマージします。

- カスタムブランチを使用して、コアテンプレートコレクションの変更を保管します。以下の手順に従って **my-customizations** ブランチを作成し、カスタマイズを追加します。

a. **my-customizations** ブランチを作成して、そのブランチに切り替えます。

```
$ git checkout -b my-customizations
```

b. カスタムブランチ内のファイルを編集します。

c. 変更を git にステージします。

```
$ git add [edited files]
```

d. カスタムブランチに変更をコミットします。

```
$ git commit -m "[Commit message for custom changes]"
```

このコマンドにより、変更がコミットとして **my-customizations** ブランチに追加されます。**master** ブランチを更新するには、**master** から **my-customizations** にリベースすると、git はこれらのコミットを更新されたテンプレートに追加します。これは、カスタマイズをトラッキングして、今後テンプレートが更新された際にそれらを再生するのに役立ちます。

- アンダークラウドの更新時には、**openstack-tripleo-heat-templates** パッケージも更新を受け取る可能性があります。このような場合には、カスタムテンプレートコレクションも更新する必要があります。

  a. **openstack-tripleo-heat-templates** パッケージのバージョンを環境変数として保存します。

  ```
  $ export PACKAGE=$(rpm -qv openstack-tripleo-heat-templates)
  ```

  b. テンプレートコレクションのディレクトリーに移動して、更新されたテンプレート用に新規ブランチを作成します。

  ```
  $ cd ~/templates/openstack-tripleo-heat-templates
  $ git checkout -b $PACKAGE
  ```

  c. そのブランチの全ファイルを削除して、新しいバージョンに置き換えます。

  ```
  $ git rm -rf *
  $ cp -r /usr/share/openstack-tripleo-heat-templates/* .
  ```

  d. 初期コミット用にすべてのテンプレートを追加します。

  ```
  $ git add *
  ```

  e. パッケージ更新のコミットを作成します。

  ```
  $ git commit -m "Updates for $PACKAGE"
  ```

  f. このブランチを master にマージします。git 管理システム (例: GitLab) を使用している場合には、管理ワークフローを使用してください。git をローカルで使用している場合には、**master** ブランチに切り替えてから **git merge** コマンドを実行してマージします。

  ```
  $ git checkout master
  $ git merge $PACKAGE
  ```

master ブランチに最新のコアテンプレートコレクションが含まれるようになりました。これで、**my-customization** ブランチを更新されたコレクションからリベースできます。

- **my-customization** ブランチを更新します。

  a. **my-customizations** ブランチに切り替えます。

  ```
  $ git checkout my-customizations
  ```

  b. このブランチを **master** からリベースします。

  ```
  $ git rebase master
  ```

  これにより、**my-customizations** ブランチが更新され、このブランチに追加されたカスタムコミットが再生されます。

- リベース中に発生する競合を解決します。

  a. どのファイルで競合が発生しているかを確認します。

```
$ git status
```

b. 特定したテンプレートファイルで競合を解決します。

c. 解決したファイルを追加します。

```
$ git add [resolved files]
```

d. リベースを続行します。

```
$ git rebase --continue
```

- カスタムテンプレートコレクションをデプロイします。

a. 必ず **my-customization** ブランチに切り替えた状態にします。

```
git checkout my-customizations
```

b. **openstack overcloud deploy** コマンドに **--templates** オプションを付けて、ローカルのテンプレートディレクトリーを指定して実行します。

```
$ openstack overcloud deploy --templates /home/stack/templates/openstack-tripleo-heat-templates [OTHER OPTIONS]
```

注記

ディレクトリーの指定をせずに **--templates** オプションを使用すると、director はデフォルトのテンプレートディレクトリー (**/usr/share/openstack-tripleo-heat-templates**) を使用します。

重要

Red Hat は、heat テンプレートコレクションを変更する代わりに 「設定フック」 に記載の方法を使用することを推奨します。

10.1.6. Jinja2 構文のレンダリング

/usr/share/openstack-tripleo-heat-templates のコア heat テンプレートには、 **j2.yaml** の拡張子が付いた多数のファイルが含まれています。これらのファイルには Jinja2 テンプレート構文が含まれ、director はこれらのファイルを **.yaml** 拡張子を持つ等価な静的 heat テンプレートにレンダリングします。たとえば、メインの **overcloud.j2.yaml** ファイルは **overcloud.yaml** にレンダリングされます。director はレンダリングされた **overcloud.yaml** ファイルを使用します。

Jinja2 タイプの heat テンプレートでは、Jinja2 構文を使用して反復値のパラメーターおよびリソースを作成します。たとえば、**overcloud.j2.yaml** ファイルには以下のスニペットが含まれます。

```
parameters:
...
{% for role in roles %}
  ...
  {{role.name}}Count:
    description: Number of {{role.name}} nodes to deploy
    type: number
```

```
    default: {{role.CountDefault|default(0)}}
  ...
{% endfor %}
```

director が Jinja2 構文をレンダリングする場合、director は **roles_data.yaml** ファイルで定義されるロールを繰り返し処理し、**{{role.name}}Count** パラメーターにロール名を代入します。デフォルトの **roles_data.yaml** ファイルには 5 つのロールが含まれ、ここでの例からは以下のパラメーターが作成されます。

- **ControllerCount**
- **ComputeCount**
- **BlockStorageCount**
- **ObjectStorageCount**
- **CephStorageCount**

レンダリング済みバージョンのパラメーターの例を以下に示します。

```
parameters:
  ...
  ControllerCount:
    description: Number of Controller nodes to deploy
    type: number
    default: 1
  ...
```

director がレンダリングするのは、コア heat テンプレートディレクトリー内からの Jinja2 タイプのテンプレートおよび環境ファイルだけです。Jinja2 テンプレートをレンダリングする際の正しい設定方法を、以下のユースケースで説明します。

ユースケース 1: デフォルトのコアテンプレート

テンプレートのディレクトリー: /usr/share/openstack-tripleo-heat-templates/

環境ファイル: /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.j2.yaml

Director はデフォルトのコアテンプレートの場所 (**--templates**) を使用し、**enable-internal-tls.j2.yaml** ファイルを **enable-internal-tls.yaml** にレンダリングします。**openstack overcloud deploy** コマンドを実行するときに、**-e** オプションを使用して、レンダリングされた **enable-internal-tls.yaml** ファイルの名前を含めます。

```
$ openstack overcloud deploy --templates \
    -e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml
    ...
```

ユースケース 2: カスタムコアテンプレート

テンプレートディレクトリー: /home/stack/tripleo-heat-installer-templates

環境ファイル: /home/stack/tripleo-heat-installer-templates/environments/ssl/enable-internal-tls.j2.yaml

Director はカスタムコアテンプレートの場所 (**--templates /home/stack/tripleo-heat-templates**) を使用し、director はカスタムコアテンプレート内の **enable-internal-tls.j2.yaml** ファイルを **enable-internal-tls.yaml** にレンダリングします。**openstack overcloud deploy** コマンドを実行するときに、**-e** オプションを使用して、レンダリングされた **enable-internal-tls.yaml** ファイルの名前を含めます。

```
$ openstack overcloud deploy --templates /home/stack/tripleo-heat-templates \
    -e /home/stack/tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml
    ...
```

ユースケース 3: 誤った設定

テンプレートのディレクトリー: /usr/share/openstack-tripleo-heat-templates/

環境ファイル: /home/stack/tripleo-heat-installer-templates/environments/ssl/enable-internal-tls.j2.yaml

director はカスタムコアテンプレートの場所を使用します (**--templates /home/stack/tripleo-heat-installer-templates**)。ただし、選択された **enable-internal-tls.j2.yaml** はカスタムコアテンプレート内に配置されていないため、**enable-internal-tls.yaml** にレンダリングされません。この設定ではデプロイメントに失敗します。

Jinja2 構文の静的テンプレートへの処理

process-templates.py スクリプトを使用して、**openstack-tripleo-heat-templates** の Jinja2 構文を静的テンプレートセットにレンダリングします。**process-templates.py** スクリプトで **openstack-tripleo-heat-templates** コレクションのコピーをレンダリングするには、**openstack-tripleo-heat-templates** ディレクトリーに移動します。

```
$ cd /usr/share/openstack-tripleo-heat-templates
```

静的コピーを保存するカスタムディレクトリーを定義する **-o** オプションを指定して、**tools** ディレクトリーにある **process-templates.py** スクリプトを実行します。

```
$ ./tools/process-templates.py -o ~/openstack-tripleo-heat-templates-rendered
```

これにより、すべての Jinja2 テンプレートがレンタリング済みの YAML バージョンに変換され、結果が **~/openstack-tripleo-heat-templates-rendered** に保存されます。

10.2. HEAT パラメーター

director テンプレートコレクション内の各 heat テンプレートには、**parameters** セクションがあります。このセクションには、特定のオーバークラウドサービス固有の全パラメーターの定義が含まれます。これには、以下のパラメーターが含まれます。

- **overcloud.j2.yaml**: デフォルトのベースパラメーター
- **roles_data.yaml**: コンポーザブルロールのデフォルトパラメーター
- **deployment/*.yaml**: 特定のサービスのデフォルトパラメーター

これらのパラメーターの値は、以下の方法で変更することができます。

1. カスタムパラメーター用の環境ファイルを作成します。
2. その環境ファイルの **parameter_defaults** セクションにカスタムのパラメーターを追加します。

3. **openstack overcloud deploy** コマンドでその環境ファイルを指定します。

10.2.1. 例 1: タイムゾーンの設定

タイムゾーンを設定するための Heat テンプレート (**puppet/services/time/timezone.yaml**) には **TimeZone** パラメーターが含まれています。**TimeZone** パラメーターの値を空白のままにすると、オーバークラウドはデフォルトで時刻を **UTC** に設定します。

タイムゾーンのリストを取得するには、**timedatectl list-timezones** コマンドを実行します。アジアのタイムゾーンを取得するコマンド例を以下に示します。

```
$ sudo timedatectl list-timezones|grep "Asia"
```

タイムゾーンを特定したら、環境ファイルの **TimeZone** パラメーターを設定します。以下に示す環境ファイルの例では、**TimeZone** の値を **Asia/Tokyo** に設定しています。

```
parameter_defaults:
  TimeZone: 'Asia/Tokyo'
```

10.2.2. 例 2: RabbitMQ ファイル記述子の上限の設定

特定の設定では、RabbitMQ サーバーのファイル記述子の上限を高くする必要がある場合があります。**deployment/rabbitmq/rabbitmq-container-puppet.yaml** の heat テンプレートを使用して **RabbitFDLimit** パラメーターで新しい制限を設定します。環境ファイルに以下のエントリーを追加します。

```
parameter_defaults:
  RabbitFDLimit: 65536
```

10.2.3. 例 3: パラメーターの有効化および無効化

デプロイメント時にパラメーターを初期設定し、それ以降のデプロイメント操作 (更新またはスケーリング操作など) ではそのパラメーターを無効にしなければならない場合があります。たとえば、オーバークラウドの作成時にカスタム RPM を含めるには、環境ファイルに以下のエントリーを追加します。

```
parameter_defaults:
  DeployArtifactURLs: ["http://www.example.com/myfile.rpm"]
```

それ以降のデプロイメントでこのパラメーターを無効にするには、パラメーターを削除するだけでは不十分です。削除するのではなく、パラメーターに空の値を設定する必要があります。

```
parameter_defaults:
  DeployArtifactURLs: []
```

これにより、それ以降のデプロイメント操作ではパラメーターは設定されなくなります。

10.2.4. 例 4: ロールベースのパラメーター

[ROLE]Parameters パラメーターを使用して特定のロールのパラメーターを設定します。ここで、**[ROLE]** はコンポーザブルロールに置き換えてください。

たとえば、director はコントローラーノードとコンピュートノードの両方に **sshd** を設定します。コントローラーノードとコンピュートノードに異なる **sshd** パラメーターを設定するには、**ControllerParameters** と **ComputeParameters** パラメーターの両方が含まれる環境ファイルを作成し、特定のロールごとに **sshd** パラメーターを設定します。

```
parameter_defaults:
  ControllerParameters:
    BannerText: "This is a Controller node"
  ComputeParameters:
    BannerText: "This is a Compute node"
```

10.2.5. 変更するパラメーターの特定

Red Hat OpenStack Platform director は、設定用のパラメーターを多数提供しています。場合によっては、設定する特定のオプションとそれに対応する director のパラメーターを特定するのが困難なことがあります。director を使用してオプションを設定するには、以下のワークフローに従ってオプションを確認し、特定のオーバークラウドパラメーターにマッピングしてください。

1. 設定するオプションを特定します。そのオプションを使用するサービスを書き留めておきます。
2. このオプションに対応する Puppet モジュールを確認します。Red Hat OpenStack Platform 用の Puppet モジュールは director ノードの **/etc/puppet/modules** にあります。各モジュールは、特定のサービスに対応しています。たとえば、**keystone** モジュールは OpenStack Identity (keystone) に対応しています。
   - 選択したオプションを制御する変数が Puppet モジュールに含まれている場合には、次のステップに進んでください。
   - 選択したオプションを制御する変数が Puppet モジュールに含まれていない場合には、そのオプションには hieradata は存在しません。可能な場合には、オーバークラウドがデプロイメントを完了した後でオプションを手動で設定することができます。
3. コア heat テンプレートコレクションに hieradata 形式の Puppet 変数が含まれているかどうかを確認します。**deployment/*** は通常、同じサービスの Puppet モジュールに対応します。たとえば、**deployment/keystone/keystone-container-puppet.yaml** テンプレートは、**keystone** モジュールの hieradata を提供します。
   - heat テンプレートが Puppet 変数用の hieradata を設定している場合には、そのテンプレートは変更することのできる director ベースのパラメーターも開示する必要があります。
   - heat テンプレートが Puppet 変数用の hieradata を設定していない場合には、環境ファイルを使用して、設定フックにより hieradata を渡します。hieradata のカスタマイズに関する詳細は、「Puppet: ロール用 hieradata のカスタマイズ」 を参照してください。

手順

1. OpenStack Identity (keystone) の通知の形式を変更するには、ワークフローを使用して、以下の手順を実施します。
   a. 設定する OpenStack パラメーターを特定します (**notification_format**)。
   b. **keystone** Puppet モジュールで **notification_format** の設定を検索します。

```
$ grep notification_format /etc/puppet/modules/keystone/manifests/*
```

この場合は、**keystone** モジュールは **keystone::notification_format** の変数を使用してこのオプションを管理します。

c. **keystone** サービステンプレートでこの変数を検索します。

```
$ grep "keystone::notification_format" /usr/share/openstack-tripleo-heat-templates/deployment/keystone/keystone-container-puppet.yaml
```

このコマンドの出力には、director が **KeystoneNotificationFormat** パラメーターを使用して **keystone::notification_format** hieradata を設定していると表示されます。

最終的なマッピングは、以下の表のとおりです。

director のパラメーター	Puppet hieradata	OpenStack Identity (keystone) のオプション
KeystoneNotificationFormat	**keystone::notification_format**	**notification_format**

オーバークラウドの環境ファイルで **KeystoneNotificationFormat** を設定すると、オーバークラウドの設定中に **keystone.conf** ファイルの **notification_format** オプションが設定されます。

10.3. 設定フック

設定フックを使用して、オーバークラウドのデプロイメントプロセスに独自のカスタム設定関数を挿入します。メインのオーバークラウドサービスの設定前後にカスタム設定を挿入するためのフックや、Puppet ベースの設定を変更/追加するためのフックを作成することができます。

10.3.1. 事前設定: 特定のオーバークラウドロールのカスタマイズ

オーバークラウドは、OpenStack コンポーネントのコア設定に Puppet を使用します。director は、コア設定を開始する前に特定のノードロールのカスタム設定を実行するために使用できるフックのセットを提供します。これらのフックには以下の設定が含まれます。

重要

本書の以前のバージョンでは、**OS::TripleO::Tasks::*PreConfig** リソースを使用してロールごとに事前設定フックを提供していました。heat テンプレートコレクションでは、これらのフックを特定の用途に使用する必要があるので、これらを個別の用途に使用すべきではありません。その代わりに、以下に概要を示す **OS::TripleO::*ExtraConfigPre** フックを使用してください。

OS::TripleO::ControllerExtraConfigPre

Puppet のコア設定前にコントローラーノードに適用される追加の設定

OS::TripleO::ComputeExtraConfigPre

Puppet のコア設定前にコンピュートノードに適用される追加の設定

OS::TripleO::CephStorageExtraConfigPre

Puppet のコア設定前に Ceph Storage ノードに適用される追加の設定

OS::TripleO::ObjectStorageExtraConfigPre

Puppet のコア設定前にオブジェクトストレージノードに適用される追加の設定

OS::TripleO::BlockStorageExtraConfigPre

Puppet のコア設定前に Block Storage ノードに適用される追加の設定

OS::TripleO::[ROLE]ExtraConfigPre

Puppet のコア設定前にカスタムノードに適用される追加の設定。**[ROLE]** をコンポーザブルロール名に置き換えてください。

以下の例では、特定ロールの全ノードの **resolv.conf** ファイルに変数のネームサーバーを追加します。

手順

1. ノードの **resolv.conf** ファイルに変数のネームサーバーを書き込むスクリプトを実行するために、基本的な heat テンプレート **~/templates/nameserver.yaml** を作成します。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  server:
    type: string
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

resources:
  CustomExtraConfigPre:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" > /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeploymentPre:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: {get_param: server}
      config: {get_resource: CustomExtraConfigPre}
      actions: ['CREATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

outputs:
  deploy_stdout:
    description: Deployment reference, used to trigger pre-deploy on changes
    value: {get_attr: [CustomExtraDeploymentPre, deploy_stdout]}
```

上記の例では、**resources** セクションには以下のパラメーターが含まれます。

CustomExtraConfigPre

ここでは、ソフトウェア設定を定義します。上記の例では、Bash **スクリプト** を定義し、heat が **_NAMESERVER_IP_** を **nameserver_ip** パラメーターに保管された値に置き換えます。

CustomExtraDeploymentPre

この設定により、**CustomExtraConfigPre** リソースで定義したソフトウェア設定を実行します。以下の点に注意してください。

- **config** パラメーターは **CustomExtraConfigPre** リソースを参照し、適用する設定を heat が理解できるようにします。
- **server** パラメーターは、オーバークラウドノードのマッピングを取得します。これは親テンプレートにより提供されるパラメーターで、このフックのテンプレートには必須です。
- **actions** パラメーターは、設定を適用するタイミングを定義します。上記の例では、オーバークラウドが作成または更新された時に設定を適用します。設定可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND**、および **RESUME** です。
- **input_values** では **deploy_identifier** というパラメーターを定義し、親テンプレートからの **DeployIdentifier** を格納します。このパラメーターにより、各デプロイメント更新のリソースにタイムスタンプが提供されます。これにより、これ以降のオーバークラウド更新時にリソースが再度適用されるようになります。

2. heat テンプレートをロールベースのリソース種別として登録する環境ファイル (**~/templates/pre_config.yaml**) を作成します。たとえば、コントローラーノードだけに設定を適用するには、**ControllerExtraConfigPre** フックを使用します。

```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. その他の環境ファイルと共に環境ファイルをスタックに追加します。

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/pre_config.yaml \
    ...
```

これにより、オーバークラウドの初回作成またはそれ以降の更新において、コア設定前にすべてのコントローラーノードに設定が適用されます。

重要

各リソースを登録することができるのは、1 つのフックにつき 1 つの heat テンプレートだけです。別の heat テンプレートに登録すると、使用する heat テンプレートがそのテンプレートに変わります。

10.3.2. 事前設定: 全オーバークラウドロールのカスタマイズ

オーバークラウドは、OpenStack コンポーネントのコア設定に Puppet を使用します。director は、コア設定を開始する前にすべてのノードタイプを設定するために使用できるフックを提供します。

OS::TripleO::NodeExtraConfig

Puppet のコア設定前に全ノードロールに適用される追加の設定

以下の例では、各ノードの **resolv.conf** ファイルに変数のネームサーバーを追加します。

手順

1. 各ノードの **resolv.conf** に変数のネームサーバーを追加するスクリプトを実行するために、まず基本的な heat テンプレート (~/**templates**/**nameserver.yaml**) を作成します。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  server:
    type: string
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

resources:
  CustomExtraConfigPre:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeploymentPre:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: {get_param: server}
      config: {get_resource: CustomExtraConfigPre}
      actions: ['CREATE']
      input_values:
        deploy_identifier: {get_param: DeployIdentifier}

outputs:
  deploy_stdout:
    description: Deployment reference, used to trigger pre-deploy on changes
    value: {get_attr: [CustomExtraDeploymentPre, deploy_stdout]}
```

上記の例では、**resources** セクションには以下のパラメーターが含まれます。

CustomExtraConfigPre

このパラメーターは、ソフトウェア設定を定義します。上記の例では、Bash **スクリプト** を定義し、heat が **_NAMESERVER_IP_** を **nameserver_ip** パラメーターに保管された値に置き換えます。

CustomExtraDeploymentPre

このパラメーターにより、**CustomExtraConfigPre** リソースで定義したソフトウェア設定を実行します。以下の点に注意してください。

- **config** パラメーターは **CustomExtraConfigPre** リソースを参照し、適用する設定を heat が理解できるようにします。
- **server** パラメーターは、オーバークラウドノードのマッピングを取得します。これは親テンプレートにより提供されるパラメーターで、このフックのテンプレートには必須です。
- **actions** パラメーターは、設定を適用するタイミングを定義します。上記の例では、オーバークラウドが作成または更新された時にのみ設定を適用します。設定可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND**、および **RESUME** です。
- **input_values** パラメーターでは **deploy_identifier** というサブパラメーターを定義し、親テンプレートからの **DeployIdentifier** を格納します。このパラメーターにより、各デプロイメント更新のリソースにタイムスタンプが提供されます。これにより、これ以降のオーバークラウド更新時にリソースが再度適用されるようになります。

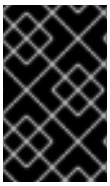
2. **OS::TripleO::NodeExtraConfig** リソース種別として heat テンプレートを登録する環境ファイル (**~/templates/pre_config.yaml**) を作成します。

```
resource_registry:
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. その他の環境ファイルと共に環境ファイルをスタックに追加します。

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/pre_config.yaml \
    ...
```

これにより、オーバークラウドの初回作成またはそれ以降の更新において、コア設定前にすべてのノードに設定が適用されます。



重要

OS::TripleO::NodeExtraConfig を登録することができるのは 1 つの heat テンプレートだけです。別の heat テンプレートに登録すると、使用する heat テンプレートがそのテンプレートに変わります。

10.3.3. 設定後: 全オーバークラウドロールのカスタマイズ

重要

本書の以前のバージョンでは、**OS::TripleO::Tasks::*PostConfig** リソースを使用してロールごとに設定後フックを提供していました。heat テンプレートコレクションでは、これらのフックを特定の用途に使用する必要があるので、これらを個別の用途に使用すべきではありません。その代わりに、以下に概要を示す **OS::TripleO::NodeExtraConfigPost** フックを使用してください。

オーバークラウドの初回作成時または更新時において、オーバークラウドの作成が完了してからすべてのロールに設定の追加が必要となる可能性があります。このような場合には、以下の設定後フックを使用します。

OS::TripleO::NodeExtraConfigPost

Puppet のコア設定後に全ノードロールに適用される追加の設定

以下の例では、各ノードの **resolv.conf** ファイルに変数のネームサーバーを追加します。

手順

1. 各ノードの **resolv.conf** に変数のネームサーバーを追加するスクリプトを実行するために、まず基本的な heat テンプレート (**~/templates/nameserver.yaml**) を作成します。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

parameters:
  servers:
    type: json
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string
  EndpointMap:
    default: {}
    type: json

resources:
  CustomExtraConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolv.conf
          params:
            _NAMESERVER_IP_: {get_param: nameserver_ip}

  CustomExtraDeployments:
    type: OS::Heat::SoftwareDeploymentGroup
    properties:
      servers:  {get_param: servers}
```

```
    config: {get_resource: CustomExtraConfig}
    actions: ['CREATE']
    input_values:
      deploy_identifier: {get_param: DeployIdentifier}
```

上記の例では、**resources** セクションには以下のパラメーターが含まれます。

CustomExtraConfig

ここでは、ソフトウェア設定を定義します。上記の例では、Bash **スクリプト** を定義し、heat が **_NAMESERVER_IP_** を **nameserver_ip** パラメーターに保管された値に置き換えます。

CustomExtraDeployments

この設定により、**CustomExtraConfig** リソースで定義したソフトウェア設定を実行します。以下の点に注意してください。

- **config** パラメーターは **CustomExtraConfig** リソースを参照し、適用する設定を heat が理解できるようにします。
- **servers** パラメーターは、オーバークラウドノードのマッピングを取得します。これは親テンプレートにより提供されるパラメーターで、このフックのテンプレートには必須です。
- **actions** パラメーターは、設定を適用するタイミングを定義します。上記の例では、オーバークラウドが作成または更新された時に設定を適用します。設定可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND**、および **RESUME** です。
- **input_values** では **deploy_identifier** というパラメーターを定義し、親テンプレートからの **DeployIdentifier** を格納します。このパラメーターにより、各デプロイメント更新のリソースにタイムスタンプが提供されます。これにより、これ以降のオーバークラウド更新時にリソースが再度適用されるようになります。

2. **OS::TripleO::NodeExtraConfigPost:** リソース種別として heat テンプレートを登録する環境ファイル (**~/templates/post_config.yaml**) を作成します。

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

3. その他の環境ファイルと共に環境ファイルをスタックに追加します。

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/post_config.yaml \
    ...
```

これにより、オーバークラウドの初回作成またはそれ以降の更新において、コア設定の完了後にすべてのノードに設定が適用されます。

重要

OS::TripleO::NodeExtraConfigPost を登録することができるのは 1 つの heat テンプレートだけです。別の heat テンプレートに登録すると、使用する heat テンプレートがそのテンプレートに変わります。

10.3.4. Puppet: ロール用 hieradata のカスタマイズ

heat テンプレートコレクションにはパラメーターのセットが含まれ、これを使用して特定のノード種別に追加の設定を渡すことができます。これらのパラメーターでは、ノードの Puppet 設定用 hieradata として設定を保存します。

ControllerExtraConfig

すべてのコントローラーノードに追加する設定

ComputeExtraConfig

すべてのコンピュートノードに追加する設定

BlockStorageExtraConfig

すべての Block Storage ノードに追加する設定

ObjectStorageExtraConfig

すべてのオブジェクトストレージノードに追加する設定

CephStorageExtraConfig

すべての Ceph Storage ノードに追加する設定

[ROLE]ExtraConfig

コンポーザブルロールに追加する設定。**[ROLE]** をコンポーザブルロール名に置き換えてください。

ExtraConfig

すべてのノードに追加する設定

手順

1. デプロイ後の設定プロセスに設定を追加するには、**parameter_defaults** セクションにこれらのパラメーターが記載された環境ファイルを作成します。たとえば、コンピュートホストに確保するメモリーを 1024 MB に増やし VNC キーマップを日本語に指定するには、**ComputeExtraConfig** パラメーターの以下のエントリーを使用します。

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::reserved_host_memory: 1024
    nova::compute::vnc_keymap: ja
```

2. ご自分のデプロイメントに該当するその他の環境ファイルと共に、この環境ファイルを **openstack overcloud deploy** コマンドに追加します。

重要

それぞれのパラメーターを定義することができるのは 1 度だけです。さらに定義すると、以前の値が上書きされます。

10.3.5. Puppet: 個別のノードの hieradata のカスタマイズ

heat テンプレートコレクションを使用して、個別のノードの Puppet hieradata を設定することができます。

手順

1. ノードのイントロスペクションデータからシステム UUID を特定します。

```
$ openstack baremetal introspection data save 9dcc87ae-4c6d-4ede-81a5-9b20d7dc4a14 |
jq .extra.system.product.uuid
```

このコマンドは、システム UUID を返します。以下に例を示します。

```
"f5055c6c-477f-47fb-afe5-95c6928c407f"
```

2. ノード固有の hieradata を定義して **per_node.yaml** テンプレートを事前設定フックに登録する環境ファイルを作成します。**NodeDataLookup** パラメーターに、設定するノードのシステム UUID を指定します。

```
resource_registry:
  OS::TripleO::ComputeExtraConfigPre: /usr/share/openstack-tripleo-heat-
templates/puppet/extraconfig/pre_deploy/per_node.yaml
parameter_defaults:
  NodeDataLookup: '{"f5055c6c-477f-47fb-afe5-95c6928c407f":
{"nova::compute::vcpu_pin_set": [ "2", "3" ]}}'
```

3. ご自分のデプロイメントに該当するその他の環境ファイルと共に、この環境ファイルを **openstack overcloud deploy** コマンドに追加します。

per_node.yaml テンプレートは、各システム UUID に対応するノード上に heiradata ファイルのセットを生成して、定義した hieradata を含めます。UUID が定義されていない場合には、生成される hieradata ファイルは空になります。上記の例では、**per_node.yaml** テンプレートは (**OS::TripleO::ComputeExtraConfigPre** フックに従って) 全コンピュートノード上で実行されますが、システム UUID が **f5055c6c-477f-47fb-afe5-95c6928c407f** のコンピュートノードのみが hieradata を受け取ります。

このメカニズムを使用して、特定の要件に応じて各ノードを個別に調整することができます。

10.3.6. Puppet: カスタムのマニフェストの適用

特定の状況では、追加のコンポーネントをオーバークラウドノードにインストールして設定する場合があります。そのためには、カスタムの Puppet マニフェストを使用して、主要な設定が完了してからノードに適用します。基本的な例として、各ノードに **motd** をインストールするケースを考えます。

手順

1. Puppet 設定を起動する heat テンプレート **~/templates/custom_puppet_config.yaml** を作成します。

```
heat_template_version: 2014-10-16

description: >
  Run Puppet extra configuration to set new MOTD

parameters:
```

```
  servers:
    type: json
  DeployIdentifier:
    type: string
  EndpointMap:
    default: {}
    type: json

resources:
  ExtraPuppetConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      config: {get_file: motd.pp}
      group: puppet
      options:
        enable_hiera: True
        enable_facter: False

  ExtraPuppetDeployments:
    type: OS::Heat::SoftwareDeploymentGroup
    properties:
      config: {get_resource: ExtraPuppetConfig}
      servers: {get_param: servers}
```

この例は、テンプレート内に **/home/stack/templates/motd.pp** を追加し、設定するノードに渡します。**motd.pp** ファイルには、**motd** のインストールと設定に必要な Puppet クラスが含まれています。

2. **OS::TripleO::NodeExtraConfigPost:** リソース種別として heat テンプレートを登録する環境ファイル (**~templates/puppet_post_config.yaml**) を作成します。

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/custom_puppet_config.yaml
```

3. ご自分のデプロイメントに該当するその他の環境ファイルと共に、この環境ファイルを **openstack overcloud deploy** コマンドに追加します。

```
$ openstack overcloud deploy --templates \
    ...
    -e /home/stack/templates/puppet_post_config.yaml \
    ...
```

これにより、**motd.pp** からの設定がオーバークラウド内の全ノードに適用されます。