Red Hat OpenStack Platform 13

パートナーのソリューションの統合

Red Hat OpenStack Platform 環境におけるサードパーティーのソフトウェアおよびハードウェアの統合および認定

Last Updated: 2022-01-17

Red Hat OpenStack Platform 13 パートナーのソリューションの統合

Red Hat OpenStack Platform 環境におけるサードパーティーのソフトウェアおよびハードウェアの統合および認定

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

法律上の通知

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

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

本ガイドは、サードパーティーコンポーネントを Red Hat OpenStack Platform 環境に統合および認定する際のガイドラインを提供します。これには、オーバークラウドのイメージへのコンポーネントの追加、director を使用したデプロイメントの設定の作成、およびこれらのコンポーネントを Red Hat Connect Build Service で認定することが含まれます。

目次

第1章 サードパーティーコンポーネントを統合する理由 4
1.1. パートナーインテグレーションの前提条件 4

第2章 DIRECTOR のアーキテクチャー 5
2.1. コアコンポーネントとオーバークラウド 5
2.1.1. OpenStack Bare Metal Provisioning サービス (ironic) 6
2.1.2. heat 6
2.1.3. Puppet 7
2.1.4. TripleO および TripleO heat テンプレート 8
2.1.5. コンポーザブルサービス 9
2.1.6. コンテナー化されたサービスおよび Kolla 9
2.1.7. Ansible 9

第3章 オーバークラウドイメージに関する操作 10
3.1. オーバークラウドイメージの取得 10
3.2. INITRD: 最初の RAMDISK の変更 10
3.3. QCOW: DIRECTOR への VIRT-CUSTOMIZE のインストール 11
3.4. QCOW: オーバークラウドイメージの検査 12
3.5. QCOW: ROOT パスワードの設定 12
3.6. QCOW: イメージの登録 12
3.7. QCOW: サブスクリプションのアタッチと RED HAT リポジトリーの有効化 13
3.8. QCOW: カスタムリポジトリーファイルのコピー 13
3.9. QCOW: RPM のインストール 14
3.10. QCOW: サブスクリプションプールの消去 14
3.11. QCOW: イメージの登録解除 14
3.12. QCOW: マシン ID のリセット 15
3.13. DIRECTOR へのイメージのアップロード 15

第4章 OPENSTACK PUPPET モジュールへの設定追加 17
4.1. パペットの構文とモジュールの構造 17
4.1.1. Puppet モジュールの構造 17
4.1.2. サービスのインストール 18
4.1.3. サービスの起動と有効化 18
4.1.4. サービスの設定 19
4.2. OPENSTACK PUPPET モジュールの取得 21
4.3. PUPPET モジュールの設定例 21
4.4. PUPPET 設定への HIERA データの追加例 23

第5章 オーケストレーション 25
5.1. HEAT テンプレートの基礎知識 25
5.1.1. heat テンプレートの概要 25
5.1.2. 環境ファイルの概要 26
5.2. デフォルトの DIRECTOR テンプレートの取得 28
5.3. 初回起動: 初回起動設定のカスタマイズ 30
5.4. 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 32
5.5. 事前設定: 全オーバークラウドロールのカスタマイズ 36
5.6. 設定後: 全オーバークラウドロールのカスタマイズ 39
5.7. PUPPET: オーバークラウドへのカスタム設定の適用 43
5.8. PUPPET: ロール用 HIERADATA のカスタマイズ 44
5.9. オーバークラウドのデプロイメントへの環境ファイルの追加 45

第6章 コンポーザブルサービス 47

6.1. コンポーザブルサービスアーキテクチャーの考察 47
6.2. ユーザー定義のコンポーザブルサービスの作成 49
6.3. ユーザー定義のコンポーザブルサービスの追加 51

第7章 認定済みコンテナーイメージのビルド 53
7.1. コンテナープロジェクトの追加 53
7.2. コンテナー認定チェックリストへの準拠 55
7.3. DOCKERFILE の要件 58
7.4. プロジェクト詳細の設定 59
7.5. ビルドサービスを使用したコンテナーイメージのビルド 62
7.6. エラーの発生したスキャン結果の修正 64
7.7. コンテナーイメージの公開 65
7.8. ベンダープラグインのデプロイ 65

第8章 OPENSTACK コンポーネントの統合と DIRECTOR およびオーバークラウドとの関係 67
8.1. BARE METAL PROVISIONING (IRONIC) 67
8.2. NETWORKING (NEUTRON) 69
8.3. BLOCK STORAGE (CINDER) 71
8.4. IMAGE STORAGE (GLANCE) 73
8.5. SHARED FILE SYSTEMS (MANILA) 75
8.6. OPENSHIFT ON OPENSTACK 76

付録A コンポーザブルサービスのパラメーター 77
A.1. すべてのコンポーザブルサービス 78
A.2. コンテナー化されたコンポーザブルサービス 80

第1章 サードパーティーコンポーネントを統合する理由

Red Hat OpenStack Platform (RHOSP) を使用して、ソリューションを RHOSP director と統合できます。RHOSP director を使用して、RHOSP 環境のデプロイメントライフサイクルをインストールおよび管理します。リソースを最適化し、デプロイメントに要する時間を短縮し、ライフサイクル管理コストを削減できます。

RHOSP director インテグレーションにより、既存のエンタープライズ管理システムおよびプロセスを統合します。CloudForms 等の Red Hat 製品により director との統合プロセスを把握し、サービスデプロイメントの管理を広範囲に公開することが期待されます。

1.1. パートナーインテグレーションの前提条件

director で操作を実行する前に、いくつかの前提条件を満たす必要があります。パートナーインテグレーションの目的は、Red Hat のエンジニアリングチーム、パートナーのマネージャー、サポート要員が協調してテクノロジーの統合を効率的に行えるように、統合全体について共通理解を形作ることです。

Red Hat OpenStack Platform director にサードパーティーのコンポーネントを含めるには、Red Hat OpenStack Platform でパートナーソリューションを認証する必要があります。

OpenStack 用プラグインの認定に関するガイド

- 『Red Hat OpenStack Certification Policy Guide』
- 『Red Hat OpenStack Certification Workflow Guide』

OpenStack 用アプリケーションの認定に関するガイド

- 『Red Hat OpenStack Application and VNF Policy Guide』
- 『Red Hat OpenStack Application and VNF Workflow Guide』

OpenStack 用ベアメタルの認定に関するガイド

- 『Red Hat OpenStack Platform Hardware Bare Metal Certification Policy Guide』
- 『Red Hat OpenStack Platform Hardware Bare Metal Certification Workflow Guide』

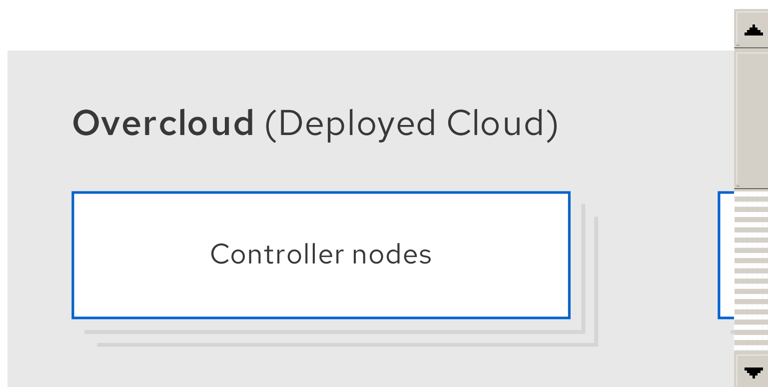
第2章 DIRECTOR のアーキテクチャー

Red Hat OpenStack Platform director は、OpenStack API を使用して、Red Hat OpenStack Platform (RHOSP) 環境の設定、デプロイ、および管理を行います。つまり、director との統合では、これらの OpenStack API およびサポートコンポーネントと統合する必要があります。これらの API のメリットは、十分に文書化されていること、アップストリームで統合テストが幅広く行われていること、成熟していること、また RHOSP 基本知識を持つユーザーであればより簡単に director の機能の仕組みを理解できることなどです。director は、OpenStack のコア機能拡張、セキュリティー修正プログラム、バグの修正を自動的に継承します。

director は、完全な RHOSP 環境のインストールと管理に使用するツールセットです。director は、主に OpenStack プロジェクト TripleO (「OpenStack-On-OpenStack」の略語) をベースとしてます。このプロジェクトは、RHOSP のコンポーネントを使用して、完全に機能する RHOSP 環境をインストールします。これには、OpenStack ノードとして使用するベアメタルシステムのプロビジョニングや制御を行う新たな OpenStack のコンポーネントが含まれます。director により、効率的で堅牢性の高い、完全な RHOSP 環境を簡単にインストールできます。

director は、アンダークラウドとオーバークラウドという 2 つの主要な概念を採用しています。director は、アンダークラウドとして知られている単一システムの OpenStack 環境を形成する OpenStack コンポーネントのサブセットです。アンダークラウドは、ワークロードを実行できるように実稼働レベルのクラウドを構築できる管理システムとして機能します。この実稼働レベルのクラウドはオーバークラウドです。オーバークラウドおよびアンダークラウドに関する詳しい情報は、『director のインストールと使用方法』を参照してください。

図2.1 アンダークラウドおよびオーバークラウドのアーキテクチャー



director には、オーバークラウド構成を構築するのに使用できるツール、ユーティリティー、テンプレートのサンプルが含まれています。director は、設定データ、パラメーター、ネットワークトポロジーの情報を取得し、ironic、heat、Puppet などのコンポーネントとともにその情報を使用して、オーバークラウドのインストール環境をオーケストレーションします。

2.1. コアコンポーネントとオーバークラウド

オーバークラウドの作成に貢献する Red Hat OpenStack Platform director のコアコンポーネントを以下に示します。

- OpenStack Bare Metal Provisioning サービス (ironic)
- OpenStack Orchestration サービス (heat)
- Puppet
- TripleO および TripleO heat テンプレート

- コンポーザブルサービス
- コンテナー化されたサービスおよび Kolla
- Ansible

2.1.1. OpenStack Bare Metal Provisioning サービス (ironic)

Bare Metal Provisioning サービスは、セルフサービスのプロビジョニングを使用してエンドユーザーに専用のベアメタルホストを提供します。director は、Bare Metal Provisioning を使用してオーバークラウドのベアメタルハードウェアのライフサイクルを管理します。Bare Metal Provisioning は、自己の API を使用してベアメタルノードを定義します。

director で OpenStack 環境をプロビジョニングするには、特定のドライバーを使用して、Bare Metal Provisioning にノードを登録する必要があります。ハードウェアの多くで Intelligent Platform Management Interface (IPMI) 電源管理機能がサポートされているため、IPMI が主要なサポートドライバーとなっています。しかし、Bare Metal Provisioning には HP iLO、Cisco UCS または Dell DRAC などのベンダー固有のドライバーも含まれています。

Bare Metal Provisioning は、ノードの電源管理を制御し、イントロスペクションメカニズムを使用して、ハードウェアの情報やファクトを収集します。director は、イントロスペクションプロセスからの情報を使用して、コントローラーノード、コンピュートノード、ストレージノードなど、さまざまな OpenStack 環境のロールとノードを照合します。たとえば、ディスクが 10 個あるノードが検出された場合は、通常ストレージノードとしてプロビジョニングされます。

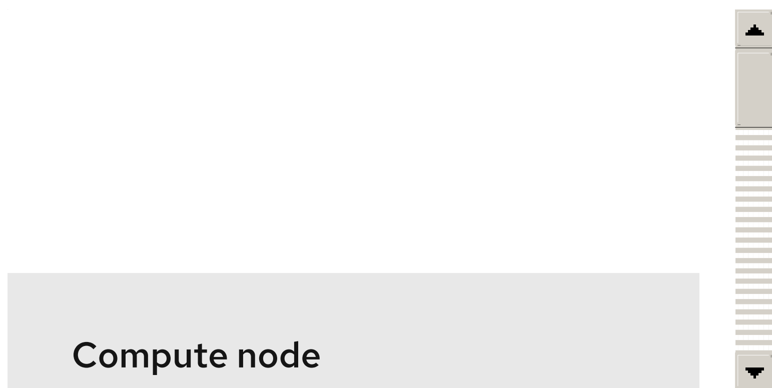
図2.2 Bare Metal Provisioning サービスを使用したノードの電源管理の制御

ハードウェアで director サポートを必要とする場合は、Bare Metal Provisioning サービスでドライバーカバレッジを設定する必要があります。

2.1.2. heat

heat は、アプリケーションスタックのオーケストレーションエンジンです。heat を使用して、クラウドにデプロイする前に、アプリケーションの要素を定義できます。複数のインフラストラクチャーリソース (例: インスタンス、ネットワーク、ストレージボリューム、Elastic IP アドレスなど) や設定用のパラメーターセットなどが含まれるスタックテンプレートを作成します。heat を使用して、特定の依存関係チェーンに基づいてこれらのリソースを作成し、リソースの可用性を監視し、必要に応じてスケーリングします。これらのテンプレートを使用して、アプリケーションスタックを移植可能にし、常に同じ結果が得られるようにすることができます。

図2.3 heat サービスを使用した、クラウドにデプロイする前のアプリケーション要素の定義



director は、ネイティブの OpenStack heat API を使用して、オーバークラウドデプロイメントに関連するリソースのプロビジョニングおよび管理を行います。これには、1 ノードロールあたりのプロビジョニングするノードの数、各ノードに設定するソフトウェアコンポーネント、それらのコンポーネントとノードの種別を director が設定する順序の定義などの詳細情報が含まれます。director は、デプロイメントのトラブルシューティングやデプロイメント後の変更を行うためにも heat を使用します。

以下の例は、コントローラーノードのパラメーターを定義する heat テンプレートのスニペットです。

```
NeutronExternalNetworkBridge:
    description: Name of bridge used for external network traffic.
    type: string
    default: 'br-ex'
NeutronBridgeMappings:
    description: >
      The OVS logical->physical bridge mappings to use. See the Neutron
      documentation for details. Defaults to mapping br-ex - the external
      bridge on hosts - to a physical name 'datacentre' which can be used
      to create provider networks (and we use this for the default floating
      network) - if changing this either use different post-install network
      scripts or be sure to keep 'datacentre' as a mapping network name.
    type: string
    default: "datacentre:br-ex"
```

Heat は、director に含まれるテンプレートで ironic を呼び出してノードの電源を入れるなど、オーバークラウドの作成を簡素化します。標準の tools ツールを使用して、進行中のオーバークラウドのリソースとステータスを表示できます。たとえば、heat ツールを使用して、入れ子状のアプリケーションスタックとしてオーバークラウドを表示することができます。実稼働向けの OpenStack クラウドを宣言および作成するには、heat テンプレートの構文を使用します。すべてのパートナーインテグレーションのユースケースには heat テンプレートが必要であるため、パートナーインテグレーションのための事前の理解と習熟が必要です。

2.1.3. Puppet

Puppet は、マシンの終了状態を記述および維持するために使用できる構成管理および適用ツールです。この最終的な状態は、Puppet マニフェストで定義します。Puppet では、以下の 2 つのモードがサポートされています。

- マニフェスト形式の手順をローカルで実行するスタンドアロンモード
- Puppet マスターと呼ばれる中央サーバーから Puppet がマニフェストを取得するサーバーモード

次の 2 つの方法で変更を行うことができます。

- 新しいマニフェストをノードにアップロードし、ローカルで実行する。
- Puppet マスターのクライアント/サーバーモデルで変更を加える。

directorでは、次の領域でパペットが使用されます。

- アンダークラウドホスト上で、undercloud **.conf** ファイルの設定に応じてローカルにパッケージをインストールおよび設定します。
- **openstack-puppet-modules** パッケージをベースのオーバークラウドイメージに挿入することで、Puppet モジュールはデプロイメント後の設定の準備が整います。デフォルトでは、ノードごとにすべての OpenStack サービスが含まれたイメージを作成します。
- 追加の Puppet マニフェストと heat パラメーターをノードに渡して、オーバークラウドのデプロイメントの後にその設定を適用します。これには、ノード種別に応じて設定を有効化および開始するサービスが含まれます。
- ノードに Puppet hieradata を渡します。Puppet モジュールやマニフェストには、マニフェストの一貫性を確保するためのサイトやノード固有のパラメーターはありません。hieradata はパラメーター値の形式で機能し、Puppet モジュールをプッシュして、他のエリアで参照することができます。たとえば、マニフェスト内の MySQL パスワードを参照するには、この情報を hieradata として保存して、マニフェスト内でこの hieradata を参照します。
  hieradata を表示するには、以下のコマンドを入力します。

  ```
  [root@localhost ~]# grep mysql_root_password hieradata.yaml # View the data in the
  hieradata file
  openstack::controller::mysql_root_password: ‘redhat123'
  ```

  Puppet マニフェストで hieradata を参照するには、以下のコマンドを入力します。

  ```
  [root@localhost ~]# grep mysql_root_password example.pp # Now referenced in the Puppet
  manifest
  mysql_root_password  => hiera(‘openstack::controller::mysql_root_password')
  ```

パートナーが統合するサービスで、パッケージをインストールしたり、サービスを有効化したりする必要がある場合には、その要件を満たすための Puppet モジュールを作成することができます。最新の OpenStack Puppet モジュールおよび例の取得の詳細については、「OpenStack Puppet モジュールの取得」を参照してください。

2.1.4. TripleO および TripleO heat テンプレート

director はアップストリームの TripleO プロジェクトをベースにしています。このプロジェクトは、以下のために OpenStack サービスセットを統合します。

- Image サービス (glance) を使用したオーバークラウドイメージの保存
- Orchestration サービス (heat) を使用してオーバークラウドのオーケストレーション
- Bare Metal Provisioning (ironic) および Compute (nova) サービスを使用したベアメタルマシンのプロビジョニング

TripleO には、Red Hat がサポートするオーバークラウド環境を定義する heat テンプレートコレクションが含まれます。director は、heat を使用してこのテンプレートコレクションを読み込み、オーバークラウドスタックをオーケストレーションします。

2.1.5. コンポーザブルサービス

Red Hat OpenStack Platform の各機能側面は、コンポーザブルサービスに細分化されます。つまり、異なるサービスの組み合わせを使用するさまざまなロールを定義できるということです。たとえば、ネットワークエージェントをデフォルトのコントローラーノードからスタンドアロンのネットワーカーノードに移すことができます。

コンポーザブルサービスのアーキテクチャーに関する詳しい情報は、「*6章コンポーザブルサービス*」を参照してください。

2.1.6. コンテナー化されたサービスおよび Kolla

Red Hat OpenStack Platform (RHOSP) の各主要サービスは、コンテナー内で実行されます。このことにより、それぞれのサービスが、ホストから独立した専用の分離名前空間内に維持されます。これには次の効果があります。

- デプロイ中、RHOSP は Red Hat カスタマーポータルからコンテナーイメージをプルして実行する。
- **podman** コマンドは、サービスの起動や停止などの管理機能を操作します。
- コンテナーをアップグレードするには、新しいコンテナーイメージをプルし、既存のコンテナーを新しいバージョンのコンテナーに置き換える必要がある。

Red Hat OpenStack Platform は、K **olla** ツールセットによりビルド/管理されるコンテナーセットを使用します。

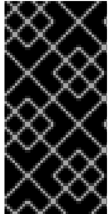
2.1.7. Ansible

Red Hat OpenStack Platform では、Ansible を使用してコンポーザブルサービスのアップグレードに関する特定の機能がアクティブ化されます。この機能には、サービスの起動/停止やデータベースアップグレードの実施が含まれます。これらのアップグレードタスクは、コンポーザブルサービスのテンプレートで定義されます。

第3章 オーバークラウドイメージに関する操作

Red Hat OpenStack Platform (RHOSP) director は、オーバークラウドのイメージを提供します。このコレクションの QCOW イメージには、ベースのソフトウェアコンポーネントが含まれており、これらを統合してコンピュート、コントローラー、ストレージノードなどさまざまなオーバークラウドのロールを形成します。場合によっては、追加のコンポーネントをノードにインストールするなど、ニーズにあわせてオーバークラウドイメージの特定の機能を変更することもできます。

virt-customize ツールを使用して、既存のコントローラーノードを増強するために既存のオーバークラウドイメージを変更することができます。たとえば、以下の手順を使用して、初期イメージには装備されていない **ml2** プラグイン、Cinder バックエンド、監視エージェントを追加でインストールします。



重要

サードパーティー製のソフトウェアを追加するために変更を加えたオーバークラウドのイメージを使用中に発生した問題を Red Hat に報告する場合には、弊社の一般サードパーティーサポートポリシー (https://access.redhat.com/articles/1067) に従って、変更を加えていないイメージで問題を再現するように依頼する場合があります。

3.1. オーバークラウドイメージの取得

director では、オーバークラウドのノードをプロビジョニングするのに、複数のディスクイメージが必要です。

- **イントロスペクションカーネルおよび ramdisk**: PXE ブートでのベアメタルシステムのイントロスペクション用
- **デプロイメントカーネルおよび ramdisk**: システムのプロビジョニングおよびデプロイメント用
- **オーバークラウドカーネル、ramdisk、および完全なイメージ**: director がノードのハードディスクに書き込むベースのオーバークラウドシステム

手順

1. これらのイメージを取得するには、**rhosp-director-images** および **rhosp-director-images-ipa** パッケージをインストールします。

```
$ sudo yum install rhosp-director-images rhosp-director-images-ipa
```

2. **stack** ユーザーのホームの **images** ディレクトリー(**/home/stack/images**)にアーカイブを展開します。

```
$ cd ~/images
$ for i in /usr/share/rhosp-director-images/overcloud-full-latest-13.0.tar /usr/share/rhosp-director-images/ironic-python-agent-latest-13.0.tar; do tar -xvf $i; done
```

3.2. INITRD: 最初の RAMDISK の変更

場合によっては、最初の ramdisk を変更する必要がある可能性があります。たとえば、イントロスペクションまたはプロビジョニングプロセス中にノードをブートする際には、特定のドライバーを利用できるようにする必要がある場合があります。オーバークラウドにおいては、これには以下の ramdisk のいずれかが含まれます。

- イントロスペクション ramdisk: **ironic-python-agent.initramfs**
- プロビジョニング ramdisk: **overcloud-full.initrd**

以下の手順では、例として **ironic-python-agent.initramfs** ramdisk に追加の RPM パッケージを追加します。

手順

1. **root** ユーザーとしてログインし、ramdisk の一時ディレクトリーを作成します。

```
# mkdir ~/ipa-tmp
# cd ~/ipa-tmp
```

2. **skipcpio** および **cpio** コマンドを使用して、一時ディレクトリーに ramdisk を展開します。

```
# /usr/lib/dracut/skipcpio ~/images/ironic-python-agent.initramfs | zcat | cpio -ivd | pax -r
```

3. 展開したコンテンツに RPM パッケージをインストールします。

```
# rpm2cpio ~/RPMs/python-proliantutils-2.1.7-1.el7ost.noarch.rpm | pax -r
```

4. 新しい ramdisk を再作成します。

```
# find . 2>/dev/null | cpio --quiet -c -o | gzip -8  > /home/stack/images/ironic-python-
agent.initramfs
# chown stack: /home/stack/images/ironic-python-agent.initramfs
```

5. ramdisk に新しいパッケージが存在することを確認します。

```
# lsinitrd /home/stack/images/ironic-python-agent.initramfs | grep proliant
```

3.3. QCOW: DIRECTOR への VIRT-CUSTOMIZE のインストール

libguestfs-tools パッケージには **virt-customize** ツールが含まれます。

手順

- **rhel-8-for -x86_64-appstream-eus-rpms リポジトリーから libguestfs- tools** をインストールします。

```
$ sudo yum install libguestfs-tools
```

重要

アンダークラウドに **libguestfs-tools** パッケージをインストールする場合は、iscsid **.socket** を無効にして、アンダークラウドの **tripleo_iscsid** サービスとポートの競合を回避します。

```
$ sudo systemctl disable --now iscsid.socket
```

3.4. QCOW: オーバークラウドイメージの検査

overcloud-full.qcow2 イメージの内容を確認する前に、このイメージを使用する仮想マシンを作成する必要があります。

手順

1. **overcloud-full.qcow2** イメージを使用する仮想マシンインスタンスを作成するには、**guestmount** コマンドを使用します。

   ```
   $ mkdir ~/overcloud-full
   $ guestmount -a overcloud-full.qcow2 -i --ro ~/overcloud-full
   ```

   QCOW2 イメージの内容は、~/**overcloud-full** で確認できます。

2. または、virt-manager を使用して、以下の起動オプションで仮想マシンを作成できます。
   - **カーネルのパス:** /overcloud-full.vmlinuz
   - **initrd のパス:** /overcloud-full.initrd
   - **カーネルの引数:** root=/dev/sda

3.5. QCOW: ROOT パスワードの設定

root パスワードを設定して、コンソールを使用してノードにアクセスする際に管理者レベルの権限を提供します。

手順

- イメージで **root** ユーザーのパスワードを設定します。

  ```
  $ virt-customize --selinux-relabel -a overcloud-full.qcow2 --root-password password:test
  [  0.0] Examining the guest ...
  [ 18.0] Setting a random seed
  [ 18.0] Setting passwords
  [ 19.0] Finishing off
  ```

3.6. QCOW: イメージの登録

Red Hat コンテンツ配信ネットワークにオーバークラウドのイメージを登録します。

手順

1. イメージを一時的に登録して、カスタマイズに適切な Red Hat のリポジトリーを有効にします。

   ```
   $ virt-customize --selinux-relabel -a overcloud-full.qcow2 --run-command 'subscription-manager register --username=[username] --password=[password]'
   [  0.0] Examining the guest ...
   [ 10.0] Setting a random seed
   [ 10.0] Running: subscription-manager register --username=[username] --password=[password]
   [ 24.0] Finishing off
   ```

2. **[username]** および **[password]** を、ご自分の Red Hat カスタマーアカウント情報に置き換えてください。これで、イメージに対して以下のコマンドが実行されます。

```
subscription-manager register --username=[username] --password=[password]
```

3.7. QCOW: サブスクリプションのアタッチと RED HAT リポジトリーの有効化

手順

1. アカウントのサブスクリプションからプール ID の一覧を検索します。

```
$ sudo subscription-manager list
```

2. サブスクリプションプール ID を選択して、その ID をイメージにアタッチします。

```
$ virt-customize --selinux-relabel -a overcloud-full.qcow2 --run-command 'subscription-manager attach --pool [subscription-pool]'
[  0.0] Examining the guest ...
[ 12.0] Setting a random seed
[ 12.0] Running: subscription-manager attach --pool [subscription-pool]
[ 52.0] Finishing off
```

3. **[subscription-pool]** は選択したサブスクリプションプール ID に置き換えてください。

```
subscription-manager attach --pool [subscription-pool]
```

これにより、リポジトリーを有効にできるように、イメージにプールが追加されます。

4. Red Hat リポジトリーを有効にします。

```
$ subscription-manager repos --enable=[repo-id]
```

3.8. QCOW: カスタムリポジトリーファイルのコピー

サードパーティー製のソフトウェアをイメージに追加するには、追加のリポジトリーが必要です。以下は、OpenDaylight リポジトリーの内容を使用する設定が含まれたリポジトリーファイルの例です。

手順

1. **opendaylight.repo** ファイルの内容を一覧表示します。

```
$ cat opendaylight.repo

[opendaylight]
name=OpenDaylight Repository
baseurl=https://nexus.opendaylight.org/content/repositories/opendaylight-yum-epel-6-x86_64/
gpgcheck=0
```

2. リポジトリーファイルをイメージにコピーします。

```
$ virt-customize --selinux-relabel -a overcloud-full.qcow2 --upload
opendaylight.repo:/etc/yum.repos.d/
[  0.0] Examining the guest ...
[ 12.0] Setting a random seed
[ 12.0] Copying: opendaylight.repo to /etc/yum.repos.d/
[ 13.0] Finishing off
```

--upload オプションは、リポジトリーファイルをオーバークラウドイメージの **/etc/yum.repos.d/** にコピーします。

重要: Red Hat は、認定を受けていないベンダーからのソフトウェアに対するサポートは提供していません。インストールするソフトウェアがサポートされていることを、Red Hat のサポート担当者に確認してください。

3.9. QCOW: RPM のインストール

手順

- **virt-customize** コマンドを使用して、イメージにパッケージをインストールします。

```
$ virt-customize --selinux-relabel -a overcloud-full.qcow2 --install opendaylight
[  0.0] Examining the guest ...
[ 11.0] Setting a random seed
[ 11.0] Installing packages: opendaylight
[ 91.0] Finishing off
```

--install オプションを使用して、インストールするパッケージを指定します。

3.10. QCOW: サブスクリプションプールの消去

手順

- 必要なパッケージをインストールしてイメージをカスタマイズした後に、サブスクリプションプールを削除して、イメージの登録を解除します。

```
$ virt-customize --selinux-relabel -a overcloud-full.qcow2 --run-command 'subscription-
manager remove --all'
[  0.0] Examining the guest ...
[ 12.0] Setting a random seed
[ 12.0] Running: subscription-manager remove --all
[ 18.0] Finishing off
```

3.11. QCOW: イメージの登録解除

手順

- オーバークラウドのデプロイメントプロセスでイメージをノードにデプロイして、各ノードを個別に登録できるように、イメージの登録を解除します。

```
$ virt-customize --selinux-relabel -a overcloud-full.qcow2 --run-command 'subscription-
```

```
manager unregister'
[  0.0] Examining the guest ...
[ 11.0] Setting a random seed
[ 11.0] Running: subscription-manager unregister
[ 17.0] Finishing off
```

3.12. QCOW: マシン ID のリセット

手順

- このイメージを使用するマシンが重複するマシン ID を使用しないように、イメージのマシン ID をリセットします。

```
$ virt-sysprep --operation machine-id -a overcloud-full.qcow2
```

3.13. DIRECTOR へのイメージのアップロード

イメージを変更したら、director にアップロードする必要があります。

手順

1. **source コマンドで stackrc ファイルを読み込み**、コマンドラインから director にアクセスできるようにします。

```
$ source stackrc
```

2. オーバークラウドのデプロイに使用するデフォルトの director イメージをアップロードします。

```
$ openstack overcloud image upload --image-path /home/stack/images/
```

このコマンドにより、以下のイメージが director にアップロードされます。

- bm-deploy-kernel
- bm-deploy-ramdisk
- overcloud-full
- overcloud-full-initrd
- overcloud-full-vmlinuz
スクリプトにより、director の PXE サーバー上にイントロスペクションイメージもインストールされます。

3. CLI でイメージ一覧を表示します。

```
$ openstack image list
+--------------------------------------+------------------------+
| ID                                   | Name                   |
+--------------------------------------+------------------------+
| 765a46af-4417-4592-91e5-a300ead3faf6 | bm-deploy-ramdisk      |
| 09b40e3d-0382-4925-a356-3a4b4f36b514 | bm-deploy-kernel       |
```

```
| ef793cd0-e65c-456a-a675-63cd57610bd5 | overcloud-full         |
| 9a51a6cb-4670-40de-b64b-b70f4dd44152 | overcloud-full-initrd  |
| 4f7e33f4-d617-47c1-b36f-cbe90f132e5d | overcloud-full-vmlinuz |
+--------------------------------------+------------------------+
```

この一覧には、イントロスペクションの PXE イメージ (agent.*) は表示されません。director は、これらのファイルを **/httpboot** にコピーします。

```
[stack@host1 ~]$ ls /httpboot -l
total 151636
-rw-r--r--. 1 ironic ironic       269 Sep 19 02:43 boot.ipxe
-rw-r--r--. 1 root   root         252 Sep 10 15:35 inspector.ipxe
-rwxr-xr-x. 1 root   root     5027584 Sep 10 16:32 agent.kernel
-rw-r--r--. 1 root   root   150230861 Sep 10 16:32 agent.ramdisk
drwxr-xr-x. 2 ironic ironic      4096 Sep 19 02:45 pxelinux.cfg
```

第4章 OPENSTACK PUPPET モジュールへの設定追加

本章では、OpenStack Puppet モジュールに設定を追加する方法を考察します。これには、Puppet モジュール開発の基本指針も含まれます。

4.1. パペットの構文とモジュールの構造

次のセクションでは、Puppet の構文および Puppet のモジュールの構造を理解するのに役立つ基本事項を説明します。

4.1.1. Puppet モジュールの構造

OpenStack モジュールに貢献する前に、Puppet モジュールを作成するコンポーネントについて理解する必要があります。

マニフェスト

マニフェストとは、リソースセットおよび属性を定義するコードが含まれるファイルのことです。リソースは、システムの設定可能なコンポーネントです。リソースの例には、パッケージ、サービス、ファイル、ユーザー、グループ、SELinux 設定、SSH キー認証、cron ジョブなどが挙げられます。マニフェストは、属性のキーと値のペアのセットを使用して必要な各リソースを定義します。

```
package { 'httpd':
  ensure => installed,
}
```

たとえば、この宣言では、**httpd** パッケージがインストールされているかどうかを確認します。インストールされていない場合は、マニフェストが **dnf** を実行してインストールします。マニフェストは、モジュールの manifest ディレクトリーに置かれています。また Puppet モジュールは、テストマニフェストのテストディレクトリーを使用します。これらのマニフェストを使用して、正式なマニフェストに含まれている特定のクラスをテストします。

クラス

クラスは、マニフェスト内の複数のリソースを統合します。たとえば、HTTP サーバーをインストールして設定する場合には、HTTP サーバーパッケージをインストールするリソース、HTTP サーバーを設定するリソース、サーバーを起動または有効化するリソースの 3 つのリソースでクラスを作成します。また、他のモジュールからのクラスを参照して設定に適用することもできます。たとえば、Web サーバーも必要なアプリケーションを設定する必要がある場合に、上述した HTTP サーバーのクラスを参照することができます。

静的ファイル

モジュールには、システムの特定の場所に、Puppet がコピーできる静的ファイルが含まれます。マニフェストのファイルリソース宣言を使用して、場所やアクセス権限などのその他の属性を定義します。
静的ファイルは、モジュールの files ディレクトリーに配置されています。

テンプレート

設定ファイルにはカスタムのコンテンツが必要な場合があります。このような場合にユーザーは静的ファイルの代わりにテンプレートを使用します。静的ファイルと同じように、テンプレートはマニフェストで定義され、システム上の場所にコピーされます。相違点は、テンプレートでは Ruby 表現でカスタマイズのコンテンツや変数入力を定義することができる点です。たとえば、カスタマイズ可能なポートで httpd を設定する場合には、設定ファイルのテンプレートには以下が含まれます。

```
Listen <%= @httpd_port %>
```

この場合には、**httpd_port** 編集はこのテンプレートを参照するマニフェストに定義されています。

テンプレートは、モジュールの templates ディレクトリーに配置されています。

プラグイン

Puppet のコア機能を超える要素については、プラグインを使用します。たとえば、プラグインを使用してカスタムファクト、カスタムリソース、または新機能を定義することができます。また、データベースの管理者が、PostgreSQL データベース向けのリソース種別を必要とする場合があります。プラグインを使用すると、データベース管理者は PostgreSQL のインストール後に新規データベースセットで PostgreSQL にデータを投入しやすくなります。その結果、データベース管理者は、PostgreSQL のインストールとその後のデータベース作成を確実に行う Puppet マニフェストのみを作成するだけで良くなります。
プラグインは、モジュールの lib ディレクトリーに配置されています。このディレクトリーには、プラグインの種別に応じたサブディレクトリーセットが含まれます。

- **/lib/facter**: カスタムファクトの場所
- **/lib/puppet/type**: 属性のキーと値のペアを記述するカスタムリソース種別の定義の場所
- **/lib/puppet/provider**: リソースを制御するためのリソース種別の定義と併用するカスタムリソースプロバイダーの場所
- **/lib/puppet/parser/functions**: カスタム関数の場所

4.1.2. サービスのインストール

一部のソフトウェアには、パッケージのインストールが必要です。これは、Puppet モジュールが実行可能な機能です。これには、特定のパッケージの設定を定義するリソース定義が必要です。

たとえば、**mymodule** モジュールを使用して **httpd** パッケージをインストールするには、mymodule **モジュールの** Puppet マニフェストに以下のコンテンツを追加します。

```
class mymodule::httpd {
  package { 'httpd':
    ensure => installed,
  }
}
```

このコードは、**httpd** パッケージのリソース宣言を定義する **httpd** と呼ばれる **mymodule** サブクラスを定義します。**ensure => installed** の属性は、パッケージがインストールされているかどうかを確認するように Puppet に指示を出します。インストールされていない場合には、Puppet は **yum** を実行してパッケージをインストールします。

4.1.3. サービスの起動と有効化

パッケージのインストール後に、サービスを起動します。**service** と呼ばれる別のリソース宣言を使用します。以下の内容が含まれるようにマニフェストを編集します。

```
class mymodule::httpd {
  package { 'httpd':
    ensure => installed,
```

```
  }
  service { 'httpd':
    ensure => running,
    enable => true,
    require => Package["httpd"],
  }
}
```

結果:

- **ensure => running** 属性は、サービスが実行さているかどうかを確認します。実行されていない場合は Puppet により有効化されます。
- **enable => true** 属性は、システムの起動時にサービスが実行されるように設定します。
- **require => Package["httpd"]** 属性は、リソース宣言同士の順序関係を定義します。今回の場合は、httpd サービスが **httpd パッケージのインストールの後に起動されるようにします。この属性により、サービスと関連のパッケージの間で依存関係が生まれます。**

4.1.4. サービスの設定

HTTP サーバーは、ポート 80 に Web ホストを提供する /etc/httpd/conf/httpd.conf にデフォルト設定をいくつか提供しています。ただし、ユーザー指定のポートに追加の Web ホストを提供するために、さらに設定を追加することができます。

手順

1. **HTTP 設定ファイルを保存するには、テンプレートファイルを使用する必要があります。これは、ユーザ定義ポートに変数入力が必要なためです。モジュールの templates ディレクトリーに、以下の内容を含む myserver.conf.erb というファイルを追加します。**

   ```
   Listen <%= @httpd_port %>
   NameVirtualHost *:<%= @httpd_port %>
   <VirtualHost *:<%= @httpd_port %>>
     DocumentRoot /var/www/myserver/
     ServerName *:<%= @fqdn %>>
     <Directory "/var/www/myserver/">
       Options All Indexes FollowSymLinks
       Order allow,deny
       Allow from all
     </Directory>
   </VirtualHost>
   ```

   このテンプレートは、Apache Web 設定の標準構文に準拠します。唯一の相違点は、モジュールから変数を注入する際に Ruby のエスケープ文字が含まれる点です。たとえば、Web サーバーポートを指定するのに使用する httpd_port などがあります。

fqdn が含まれる 変数は、システムの完全修飾ドメイン名を保存する変数です。これは、システムの完全修飾ドメイン名を保存する変数で、システムファクト として知られています。システムファクトは、システムの各 **Puppet** カタログを生成する前に各システムから取得します。**Puppet** は **facter** コマンドを使用して、これらのシステムファクトを収集します。また、これらのファクトの一覧を表示するには、**facter** を実行します。

2. **Save myserver.conf.erb.**

3. モジュールの **Puppet** マニフェストにリソースを追加します。

```
class mymodule::httpd {
  package { 'httpd':
    ensure => installed,
  }
  service { 'httpd':
    ensure => running,
    enable => true,
    require => Package["httpd"],
  }
  file {'/etc/httpd/conf.d/myserver.conf':
  notify => Service["httpd"],
    ensure => file,
    require => Package["httpd"],
    content => template("mymodule/myserver.conf.erb"),
  }
  file { "/var/www/myserver":
    ensure => "directory",
  }
}
```

結果:

- サーバー設定ファイル (**/etc/httpd/conf.d/myserver.conf**)のファイルリソース宣言を追加します。このファイルのコンテンツは、作成した **myserver.conf.erb** テンプレートです。

- このファイルを追加する前に、**httpd** パッケージがインストールされていることを確認します。

- **Web** サーバーのディレクトリー **/var/www/myserver** を作成する **2** 番目のファイルリソース宣言を追加します。

- **notify => Service["httpd"]** 属性を使用して、設定ファイルと **httpd** サービスの間の関係を追加します。これにより、設定ファイルへの変更の有無がチェックされます。ファイルが変更

された場合には、Puppet によりサービスが再起動されます。

4.2. OPENSTACK PUPPET モジュールの取得

Red Hat OpenStack Platform は、正式な OpenStack Puppet モジュールを使用します。OpenStack Puppet モジュールを取得するには、Github の openstack グループを参照してください。

手順

1. ブラウザーで、https://github.com/openstack に移動します。
2. フィルターセクションで、Puppet を検索します。すべての Puppet モジュールには puppet- の接頭辞が使用されます。
3. 必要な Puppet モジュールのクローンを作成します。たとえば、公式の OpenStack Block Storage (cinder) モジュールの場合は、次のようになります。

```
$ git clone https://github.com/openstack/puppet-cinder.git
```

4.3. PUPPET モジュールの設定例

OpenStack モジュールでは、主にコアサービスを設定します。モジュールの多くには、backends、agents または plugins として知られる追加のサービスを設定するための追加のマニフェストも含まれます。たとえば、cinder モジュールには backends と呼ばれるディレクトリーがあり、この中には NFS、iSCSI、Red Hat Ceph Storage など異なるストレージの設定オプションが含まれます。

たとえば、manifests/backends/nfs.pp ファイルには以下の設定が含まれます。

```
define cinder::backend::nfs (
  $volume_backend_name  = $name,
  $nfs_servers          = [],
  $nfs_mount_options    = undef,
  $nfs_disk_util        = undef,
  $nfs_sparsed_volumes  = undef,
  $nfs_mount_point_base = undef,
  $nfs_shares_config    = '/etc/cinder/shares.conf',
  $nfs_used_ratio       = '0.95',
  $nfs_oversub_ratio    = '1.0',
  $extra_options        = {},
) {
```

```
  file {$nfs_shares_config:
    content => join($nfs_servers, "\n"),
    require => Package['cinder'],
    notify  => Service['cinder-volume']
  }

  cinder_config {
    "${name}/volume_backend_name":  value => $volume_backend_name;
    "${name}/volume_driver":        value =>
      'cinder.volume.drivers.nfs.NfsDriver';
    "${name}/nfs_shares_config":    value => $nfs_shares_config;
    "${name}/nfs_mount_options":    value => $nfs_mount_options;
    "${name}/nfs_disk_util":        value => $nfs_disk_util;
    "${name}/nfs_sparsed_volumes":  value => $nfs_sparsed_volumes;
    "${name}/nfs_mount_point_base": value => $nfs_mount_point_base;
    "${name}/nfs_used_ratio":       value => $nfs_used_ratio;
    "${name}/nfs_oversub_ratio":    value => $nfs_oversub_ratio;
  }

  create_resources('cinder_config', $extra_options)

}
```

結果:

- **define ステートメントでは、cinder::backend::nfs と呼ばれる定義型が作成されます。定義型はクラスによく似ていますが、主な相違点は Puppet は定義型を複数回評価する点です。たとえば、複数の NFS バックエンドが必要なため、この設定では NFS 共有ごとに評価を複数回実行する必要があります。**

- **次の数行では、設定内のパラメーターとそのデフォルト値を定義します。cinder::backend::nfs の定義型に新しい値が渡された場合には、デフォルト値は上書きされます。**

- **file 関数は、ファイルの作成を呼び出すリソース宣言です。このファイルには NFS 共有の一覧が含まれており、このファイルの名前はパラメーター $nfs_shares_config = '/etc/cinder/shares.conf で定義されます。以下は追加の属性です。**

  - **content 属性は 、$nfs_servers パラメーターを使用して一覧を作成します。**

  - **require 属性は、cinder パッケージが確実にインストールされるようにします。**

  - **notify 属性は cinder-volume サービスにリセットするように指示を出します。**

- **cinder_config** 関数は、モジュールの **lib/puppet/** ディレクトリーからプラグインを使用するリソース宣言です。このプラグインは **/etc/cinder/cinder.conf** ファイルに設定を追加します。このリソースのそれぞれの行により、**cinder.conf** ファイルの適切なセクションに設定オプションが追加されます。たとえば、**$name** パラメーターが **mynfs** の場合には、属性は以下のようになります。

```
"${name}/volume_backend_name":  value => $volume_backend_name;
"${name}/volume_driver":        value =>
  'cinder.volume.drivers.nfs.NfsDriver';
"${name}/nfs_shares_config":    value => $nfs_shares_config;
```

  以下のスニペットを **cinder.conf** ファイルに保存します。

```
[mynfs]
volume_backend_name=mynfs
volume_driver=cinder.volume.drivers.nfs.NfsDriver
nfs_shares_config=/etc/cinder/shares.conf
```

- **create_resources** 関数は、ハッシュをリソースセットに変換します。この場合は、マニフェストにより **$extra_options** ハッシュがバックエンドの追加設定オプションに変換されます。これは、マニフェストのコアパラメーターに含まれていない設定オプションを追加する柔軟な方法を提供します。

これにより、ハードウェアの OpenStack ドライバーを設定するマニフェストを追加することの重要性が分かります。マニフェストは、director がハードウェアに適した設定オプションを追加する方法を提供します。マニフェストは、director がハードウェアをオーバークラウドで使用できるように設定する際の主要な統合ポイントの役割を果たします。

4.4. PUPPET 設定への HIERA データの追加例

Puppet には、hiera と呼ばれるツールが含まれています。このツールはノード固有の設定を提供するキー/値のシステムとして機能します。これらのキーと値は通常、**/etc/puppet/hieradata** に配置されるファイルに保管されています。**/etc/puppet/hiera.yaml** ファイルは、Puppet が **hieradata** ディレクトリーのファイルを読み込む順序を定義します。

オーバークラウドの設定中、Puppet は hiera データを使用して特定の Puppet クラスのデフォルト値を上書きします。たとえば、**puppet-cinder** にある **cinder::backend::nfs** の NFS のマウントオプションはデフォルトでは未定義になっています。

```
$nfs_mount_options    = undef,
```

ただし、**cinder::backend::nfs** の定義する型を呼び出す独自のマニフェストを作成して、このオプションを **hiera** データに置き換えることができます。

```
cinder::backend::nfs { $cinder_nfs_backend:
  nfs_mount_options   => hiera('cinder_nfs_mount_options'),
}
```

これは、**nfs_mount_options** パラメーターが **cinder_nfs_mount_options** キーから取得した **hiera** データの値を使用することを意味します。

```
cinder_nfs_mount_options: rsize=8192,wsize=8192
```

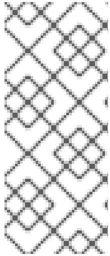
または、**hiera** データを使用して **cinder::backend::nfs::nfs_mount_options** パラメーターを直接上書きし、**NFS** 設定の全評価に適用することができます。

```
cinder::backend::nfs::nfs_mount_options: rsize=8192,wsize=8192
```

上記の **hiera** データは **cinder::backend::nfs** の各評価上にあるこのパラメーターを上書きします。

第5章 オーケストレーション

Red Hat OpenStack Platform (RHOSP) 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、Heat が作成するリソース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 RHOSP のオブジェクトを指します。



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。

本章では、独自のテンプレートファイルを作成できるように HOT 構文を理解するための基本を説明します。

5.1. HEAT テンプレートの基礎知識

5.1.1. heat テンプレートの概要

Heat テンプレートは、3 つの主要なセクションで構成されます。

パラメーター

これらは、スタックをカスタマイズするために heat に渡される設定です。heat パラメータを使用して、デフォルト値をカスタマイズすることもできます。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。

リソース

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。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。これらがテンプレートの resources セクションで定義されます。

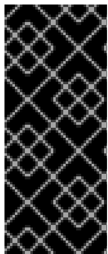
出力

これらは、スタックの作成後に heat から渡される値です。これらの値には、heat API またはクライアントツールを使用してアクセスすることができます。これらがテンプレートの output セクションで定義されます。

以下に、基本的な heat テンプレートの例を示します。

```
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、特定のフレーバー、イメージ、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。



重要

heat テンプレートは、利用可能な関数や使用する構文のバージョンを定義する **heat_template_version** パラメーターも必要とします。詳しい情報は Heat の正式なドキュメント を参照してください。

5.1.2. 環境ファイルの概要

環境ファイルとは、heat テンプレートをカスタマイズする特別な種類のテンプレートです。このファイルは、3 つの主要な部分で構成されます。

リソースレジストリー

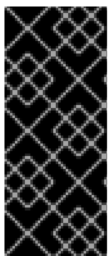
このセクションでは、他の heat テンプレートにリンクしたカスタムリソースの名前を定義します。これにより、コアリソースコレクションに存在しないカスタムのリソースを作成することができます。この設定は、環境ファイルの **resource_registry** セクションで定義されます。

パラメーター

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。これらの設定は、環境ファイルの **parameters** セクションで定義します。

パラメーターのデフォルト

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。パラメーターのデフォルト値は、環境ファイルの **parameter_defaults** セクションで定義します。



重要

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際に、パラメーター の代わりに **parameter_defaults** を使用します。パラメーターが、オーバークラウドの全スタックテンプレートに適用されるようにするためです。

基本的な環境ファイルの例:

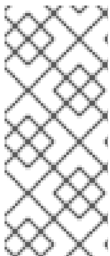
```
resource_registry:
  OS::Nova::Server::MyServer: myserver.yaml

parameter_defaults:
  NetworkName: my_network

parameters:
  MyIP: 192.168.0.1
```

heat テンプレート **my_ template.yaml** からスタックを作成する際に、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、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（例： OS::Nova::Server::MyServer リソースおよびその 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。

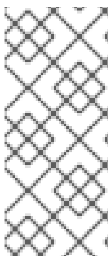
5.2. デフォルトの DIRECTOR テンプレートの取得

director は、オーバークラウドを作成するのに高度な heat テンプレートコレクションを使用します。このコレクションは、openstack-tripleo-heat-templates リポジトリーの Github にある openstack グループから入手できます。

手順

- このテンプレートコレクションのクローンを取得するには、以下のコマンドを入力します。

```
$ git clone https://github.com/openstack/tripleo-heat-templates.git
```



注記

このテンプレートコレクションの Red Hat 固有のバージョンは、openstack-tripleo-heat-template パッケージから取得できます。このパッケージは、コレクションを /usr/share/openstack-tripleo-heat-templates にインストールします。

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

オーバークラウド内のロールを定義して、サービスを各ロールにマッピングするファイル

network_data.yaml

サブネット、割り当てプール、VIP ステータスなどのオーバークラウド内のネットワークとそれらのプロパティーを定義するファイル。デフォルトの network_data 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、Management) が含まれます。カスタムの network_data 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

オーバークラウドプラン用のメタデータを定義するファイル。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

オーバークラウドプラン用の環境ファイルのマッピング。director の Web UI で環境ファイルを記述および有効化するには、このファイルを使用します。オーバークラウドプラン内の environments ディレクトリーで検出されるカスタムの環境ファイルの中で、capabilities-map.yaml では定義されていないファイルは、Web UI の 2 デプロイメントの設定の指定 > 全体の設定 の Other サブタブに一覧表示されます。

environments

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ます。capabilities-map.yaml ファイルで定義されていない、このディレクトリーで検出される環境ファイルはいずれも、director の Web UI の 2 デプロイメントの設定の指定 > 全体の設定 の Other サブタブに一覧表示されます。

network

分離ネットワークおよびポートの作成に役立つ heat テンプレートセット

puppet

主に Puppet を使用した設定により動作するテンプレート。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

コンポーザブルサービスアーキテクチャー内の全サービス用の heat テンプレートが含まれるディレクトリー

extraconfig

追加機能を有効にするテンプレート

firstboot

ノードを最初に作成する際に director が使用する first_boot スクリプトの例を提供します。

この章では、director がオーバークラウドの作成のオーケストレーションに使用するテンプレートの概要を説明しました。次の複数の項では、オーバークラウドのデプロイメントに追加可能なカスタムのテンプレートや環境ファイルを作成する方法を説明します。

5.3. 初回起動: 初回起動設定のカスタマイズ

director は、オーバークラウドの初回作成時に全ノードに対して設定を行います。そのために、director は OS::TripleO::NodeUserData リソース種別を使用して呼び出すことのできる cloud-init を使用します。

以下の例では、全ノード上でカスタム IP アドレスを使用してネームサーバーを更新します。まず基本的な heat テンプレート (/home/stack/templates/nameserver.yaml) を作成する必要があります。OS::TripleO::MultipartMime リソース種別を使用して、この設定スクリプトを送信することができます。

```
heat_template_version: 2014-10-16

description: >
  Extra hostname configuration

resources:
  userdata:
    type: OS::Heat::MultipartMime
    properties:
```

```
      parts:
      - config: {get_resource: nameserver_config}

  nameserver_config:
    type: OS::Heat::SoftwareConfig
    properties:
      config: |
        #!/bin/bash
        echo "nameserver 192.168.1.1" >> /etc/resolv.conf

outputs:
  OS::stack_id:
    value: {get_resource: userdata}
```

次に、OS::TripleO::NodeUserData 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 (/home/stack/templates/firstboot.yaml) を作成します。

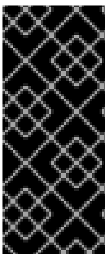
```
resource_registry:
  OS::TripleO::NodeUserData: /home/stack/templates/nameserver.yaml
```

初回起動の設定を追加するには、最初にオーバークラウドを作成する際に、その他の環境ファイルと共にこの環境ファイルをスタックに追加します。たとえば、以下のようになります。

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

-e を使用して、オーバークラウドスタックに環境ファイルを適用します。

これにより、ノード作成後の初回起動時に設定がすべてのノードに追加されます。これ以降は (たとえば、オーバークラウドスタックの更新時)、これらのテンプレートを追加してもこれらのスクリプトは実行されません。



重要

OS::TripleO::NodeUserData 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。

これにより、以下の操作が実行されます。

1. **OS::TripleO::NodeUserData** は、コレクション内の他のテンプレートで使用する director ベースの Heat リソースで、全ノードに対して初回起動の設定を適用します。このリソースは、cloud-init で使用するデータを渡します。デフォルトの NodeUserData は、空の値 (firstboot/userdata_default.yaml) を指定する heat テンプレートを参照します。この例では、firstboot.yaml の環境ファイルは、このデフォルトを独自の nameserver.yaml ファイルへの参照に置き換えます。

2. **nameserver_config** は、初回起動で実行する Bash スクリプトを定義します。OS::Heat::SoftwareConfig リソースは、適用する設定としてこれを定義します。

3. **userdata** は、OS::Heat::MultipartMime リソースを使用して、nameserver_config から複数のパートからなる MIME メッセージに設定を変換します。

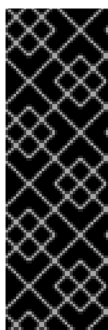
4. **outputs** では、output パラメーターの OS::stack_id が提供され、userdata から MIME メッセージを呼び出している Heat テンプレート/リソースに渡します。

これにより、各ノードは初回起動時に以下の Bash スクリプトを実行します。

```
#!/bin/bash
echo "nameserver 192.168.1.1" >> /etc/resolve.conf
```

この例では、Heat テンプレートがあるリソースから別のリソースに設定を渡して変更する方法を示しています。また、新規 Heat リソースの登録または既存のリソースの変更を行う環境ファイルの使用方法も説明します。

5.4. 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



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。director の 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。

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 にはフックのセットが用意されており、初回のブートが完了してコア設定が開始する前に、特定ノードロールのカスタム設定が提供されます。これには、以下のフックが含まれます。

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 のコア設定前にブロックストレージ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、まず基本的な heat テンプレート (/home/stack/templates/nameserver.yaml) を作成します。このテンプレートは、ノードの resolv.conf に変数のネームサーバーを書き込むスクリプトを実行します。

```
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','UPDATE']
      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、適用する設定を Heat が理解できるように **CustomExtraConfigPre** リソースを参照します。
- **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、オーバークラウドの作成または更新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** 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、それ以降のオーバークラウド更新に必ずリソースが再度適用されます。

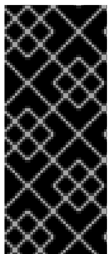
次に、Heat テンプレートをロールベースのリソース種別に登録する環境ファイル (/home/stack/templates/pre_config.yaml) を作成します。たとえば、コントローラーノードだけに適用するには、ControllerExtraConfigPre フックを使用します。

```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

この設定を適用するには、オーバークラウドの作成時または更新時に、その他の環境ファイルと共にこの環境ファイルをスタックに追加します。以下に例を示します。

```
$ 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。

これにより、以下の操作が実行されます。

1. OS::TripleO::ControllerExtraConfigPre は、Heat テンプレートコレクション内の設定テンプレートで使用する director ベースの Heat リソースです。このリソースは、各コントローラーノードに設定を渡します。デフォルトの ControllerExtraConfigPre は、空の値 (puppet/extraconfig/pre_deploy/default.yaml) を指定する heat テンプレートを参照します。この例では、pre_config.yaml 環境ファイルは、このデフォルトを独自の nameserver.yaml ファイルへの参照に置き換えます。

2. 環境ファイルは、この環境の parameter_default の値として nameserver_ip を渡します。

これは、ネームサーバーの IP アドレスを保存するパラメーターです。nameserver.yaml の Heat テンプレートは、parameters セクションで定義したように、このパラメーターを受け入れます。

3. このテンプレートは、 OS::Heat::SoftwareConfig を使用して設定リソースとして CustomExtraConfigPre を定義します。group: script プロパティーに注意してください。group は、使用するソフトウェア設定ツールを定義します。この場合は、script フックは、SoftwareConfig リソースで config プロパティーとして定義される実行可能なスクリプトを実行します。

4. このスクリプト自体は、/etc/resolve.conf にネームサーバーの IP アドレスを追加します。str_replace の属性に注意してください。この場合は、NAMESERVER_IP をネームサーバーの IP アドレスに設定します。スクリプト内の同じ変数はこの IP アドレスに置き換えられます。その結果、スクリプトは以下のようになります。

```
#!/bin/sh
echo "nameserver 192.168.1.1" >> /etc/resolve.conf
```

この例は、コアの設定の前に OS::Heat::SoftwareConfig と OS::Heat::SoftwareDeployments で設定を定義してデプロイする heat テンプレートの作成方法を示します。また、環境ファイルでパラメーターを定義して、設定でテンプレートを渡す方法も示します。

5.5. 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 にはフックが用意されており、初回のブートが完了してコア設定が開始する前に、すべてのノード種別が設定されます。

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、まず基本的な heat テンプレート (/home/stack/templates/nameserver.yaml) を作成します。このテンプレートは、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','UPDATE']
      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、適用する設定を Heat が理解できるように CustomExtraConfigPre リソースを参照します。

- **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、オーバークラウドの作成または更新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、それ以降のオーバークラウド更新に必ずリソースが再度適用されます。

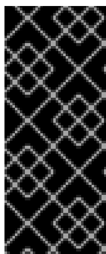
次に、**OS::TripleO::NodeExtraConfig** リソース種別として Heat テンプレートを登録する環境ファイル (**/home/stack/templates/pre_config.yaml**) を作成します。

```
resource_registry:
  OS::TripleO::NodeExtraConfig: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

この設定を適用するには、オーバークラウドの作成時または更新時に、その他の環境ファイルと共にこの環境ファイルをスタックに追加します。以下に例を示します。

```
$ 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。

これにより、以下の操作が実行されます。

1.
OS::TripleO::NodeExtraConfig は、Heat テンプレートコレクション内の設定テンプレートで使用する director ベースの Heat リソースです。このリソースは、各ノードに設定を渡します。デフォルトの NodeExtraConfig は、空の値 (puppet/extraconfig/pre_deploy/default.yaml) を指定する heat テンプレートを参照します。この例では、pre_config.yaml 環境ファイルは、このデフォルトを独自の nameserver.yaml ファイルへの参照に置き換えます。

2.
環境ファイルは、この環境の parameter_default の値として nameserver_ip を渡します。これは、ネームサーバーの IP アドレスを保存するパラメーターです。nameserver.yaml の Heat テンプレートは、parameters セクションで定義したように、このパラメーターを受け入れます。

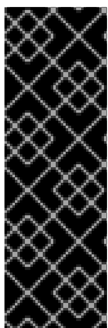
3.
このテンプレートは、 OS::Heat::SoftwareConfig を使用して設定リソースとして CustomExtraConfigPre を定義します。group: script プロパティーに注意してください。group は、使用するソフトウェア設定ツールを定義します。この場合は、script フックは、SoftwareConfig リソースで config プロパティーとして定義される実行可能なスクリプトを実行します。

4.
このスクリプト自体は、/etc/resolve.conf にネームサーバーの IP アドレスを追加します。str_replace の属性に注意してください。この場合は、NAMESERVER_IP をネームサーバーの IP アドレスに設定します。スクリプト内の同じ変数はこの IP アドレスに置き換えられます。その結果、スクリプトは以下のようになります。

```
#!/bin/sh
echo "nameserver 192.168.1.1" >> /etc/resolve.conf
```

この例は、コアの設定の前に OS::Heat::SoftwareConfig と OS::Heat::SoftwareDeployments で設定を定義してデプロイする heat テンプレートの作成方法を示します。また、環境ファイルでパラメーターを定義して、設定でテンプレートを渡す方法も示します。

5.6. 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。director の 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に適用される追加の設定

以下の例では、まず基本的な heat テンプレート (/home/stack/templates/nameserver.yaml) を作成します。このテンプレートは、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します。

```
description: >
  Extra hostname configuration

parameters:
  servers:
    type: json
  nameserver_ip:
    type: string
  DeployIdentifier:
    type: string

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','UPDATE']
      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、適用する設定を Heat が理解できるように CustomExtraConfig リソースを参照します。
- 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 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、それ以降のオーバークラウド更新に必ずリソースが再度適用されます。

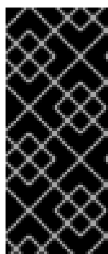
次に、OS::TripleO::NodeExtraConfigPost: リソース種別として heat テンプレートを登録する環境ファイル (/home/stack/templates/post_config.yaml) を作成します。

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: /home/stack/templates/nameserver.yaml

parameter_defaults:
  nameserver_ip: 192.168.1.1
```

この設定を適用するには、オーバークラウドの作成時または更新時に、その他の環境ファイルと共にこの環境ファイルをスタックに追加します。以下に例を示します。

```
$ 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。

これにより、以下の操作が実行されます。

1. OS::TripleO::NodeExtraConfigPost は、コレクション内の設定後のテンプレートで使用する director ベースの Heat リソースです。このリソースは、*-post.yaml を使用して各ノード種別に設定を渡します。デフォルトの NodeExtraConfigPost は、空の値 (extraconfig/post_deploy/default.yaml) を指定する heat テンプレートを参照します。この例では、post_config.yaml の環境ファイルは、このデフォルトを独自の nameserver.yaml ファイルへの参照に置き換えます。

2. 環境ファイルは、この環境の parameter_default の値として nameserver_ip を渡します。これは、ネームサーバーの IP アドレスを保存するパラメーターです。nameserver.yaml の Heat テンプレートは、parameters セクションで定義したように、このパラメーターを受け入れます。

3. このテンプレートは、OS::Heat::SoftwareConfig を使用して設定リソースとして CustomExtraConfig を定義します。group: script プロパティーに注意してください。group は、使用するソフトウェア設定ツールを定義します。この場合は、script フックは、SoftwareConfig リソースで config プロパティーとして定義される実行可能なスクリプトを実行します。

4. このスクリプト自体は、/etc/resolve.conf にネームサーバーの IP アドレスを追加します。str_replace の属性に注意してください。この場合は、NAMESERVER_IP をネームサーバーの IP アドレスに設定します。スクリプト内の同じ変数はこの IP アドレスに置き換えられます。その結果、スクリプトは以下のようになります。

```
#!/bin/sh
echo "nameserver 192.168.1.1" >> /etc/resolve.conf
```

この例は、OS::Heat::SoftwareConfig と OS::Heat::SoftwareDeployments で設定を定義してデプロイする heat テンプレートの作成方法を示します。また、環境ファイルでパラメーターを定義して、設定でテンプレートを渡す方法も示します。

5.7. PUPPET: オーバークラウドへのカスタム設定の適用

これまで、新規バックエンドの設定を OpenStack Puppet モジュールに追加する方法を説明しました。本項では、director が新規設定を適用する方法を説明します。

Heat テンプレートは、OS::Heat::SoftwareConfig リソースで Puppet 設定を適用可能なフックを提供します。このプロセスは、Bash スクリプトを追加して実行する方法に似ています。ただし、group: script フックを使用するのではなく、group: puppet フックを使用します。

たとえば、公式の Cinder Puppet モジュールを使用して NFS Cinder バックエンドを有効化する Puppet マニフェスト (example-puppet-manifest.pp) があるとします。

```
cinder::backend::nfs { 'mynfsserver':
  nfs_servers          => ['192.168.1.200:/storage'],
}
```

Puppet の設定は、cinder::backend::nfs の定義型を使用して新規リソースを作成します。Heat を使用してこのリソースを適用するには、Puppet マニフェストを実行する基本的な heat テンプレート (puppet-config.yaml) を作成します。

```
heat_template_version: 2014-10-16

parameters:
  servers:
    type: json

resources:
  ExtraPuppetConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: puppet
      config:
        get_file: example-puppet-manifest.pp
      options:
        enable_hiera: True
        enable_facter: False

  ExtraPuppetDeployment:
    type: OS::Heat::SoftwareDeployments
    properties:
      config: {get_resource: ExtraPuppetConfig}
      servers: {get_param: servers}
      actions: ['CREATE','UPDATE']
```

次に、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 (puppet_config.yaml) を作成します。

```
resource_registry:
  OS::TripleO::NodeExtraConfigPost: puppet_config.yaml
```

この例は、前項の script フックの例から SoftwareConfig および SoftwareDeployments を使用する点で似ています。ただし、この例では以下の点が異なります。

1. puppet フックを実行するために group: puppet を設定します。

2. config 属性は get_file 属性を使用して、追加の設定が含まれる Puppet マニフェストを参照します。

3. options 属性には、Puppet 設定固有のオプションが含まれます。

   - enable_hiera オプションは、Puppet 設定で Hiera データを使用できるようにします。

   - enable_facter オプションは、facter コマンドからシステムファクトを使用する Puppet 設定を有効にします。

この例では、Puppet マニフェストをオーバークラウドのソフトウェア設定の一部として追加する方法を示します。これにより、オーバークラウドのイメージで既存の Puppet モジュールから特定の設定クラスを適用する方法ができ、特定のソフトウェアやハードウェアを使用するようにオーバークラウドをカスタマイズしやすくなります。

5.8. 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、特定のノード種別に追加の設定を渡すためのパラメーターセットが含まれています。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。これらのパラメーターは以下のとおりです。

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

すべてのブロックストレージ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。

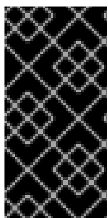
ExtraConfig

すべてのノードに追加する設定

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、以下のように設定します。

```
parameter_defaults:
  ComputeExtraConfig:
    nova::compute::reserved_host_memory: 1024
    nova::compute::vnc_keymap: ja
```

openstack overcloud deploy を実行する際に、この環境ファイルを指定します。



重要

それぞれのパラメーターを定義できるのは一度だけです。さらに定義すると、以前の値が上書きされます。

5.9. オーバークラウドのデプロイメントへの環境ファイルの追加

カスタム設定に関連する環境ファイルセットを開発した後に、オーバークラウドデプロイメントにこ

れらのファイルを追加します。これには、**-e** オプションの後に環境ファイルを指定して **openstack overcloud deploy** コマンドを実行します。カスタマイズに必要な回数だけ、**-e** オプションを指定することができます。たとえば、以下のようになります。

```
$ openstack overcloud deploy --templates -e network-configuration.yaml -e storage-configuration.yaml -e first-boot.yaml
```

重要

環境ファイルは、順序通りにスタックされます。これは、主要な heat テンプレートコレクションとこれまでの全環境ファイル両方の上に後続のファイルがスタックされることを意味します。この方法により、リソースの定義の上書きが可能となります。たとえば、オーバークラウドのデプロイメントにある全環境ファイルは **NodeExtraConfigPost** リソースを定義し、その後に Heat は最後の環境ファイルで定義した **NodeExtraConfigPost** を使用します。そのため、環境ファイルの順序は重要です。環境ファイルを正しく処理してスタックできるように、環境ファイルは順序付けてください。

警告

-e オプションを使用してオーバークラウドに追加した環境ファイルはいずれも、オーバークラウドのスタック定義の一部となります。director は、再デプロイおよびデプロイ後の機能にこれらの環境ファイルを必要とします。これらのファイルが含まれていない場合には、オーバークラウドが破損する場合があります。

第6章 コンポーザブルサービス

Red Hat OpenStack Platform (RHOSP) には、カスタムのロールとロール上のコンポーザブルサービスの組み合わせを定義する機能が実装されています。詳細は、『オーバークラウドの高度なカスタマイズ』の「コンポーザブルサービスとカスタムロール」を参照してください。統合の一環として、独自のカスタムサービスを定義して、選択したロールに追加することができます。

6.1. 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。

- **puppet/services** には、コンポーザブルサービスを設定するためのベーステンプレートが含まれます。
- **docker/services** には、主要な OpenStack Platform サービス用のコンテナー化されたテンプレートが含まれます。これらのテンプレートは、一部ベーステンプレートの機能を補足する働きをし、ベーステンプレートを後方参照します。

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、**ntp.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、RHOSP デプロイメントに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: puppet/services/time/ntp.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、以下のテンプレートを使用します。

```
resource_registry:
  ...
  OS::TripleO::Services::Keystone: docker/services/keystone.yaml
  ...
```

通常、これらのコンテナー化されたテンプレートは、**Puppet** 設定を含めるためにベーステンプレートを後方参照します。たとえば、**docker/services/keystone.yaml** テンプレートは、**KeystoneBase** パラメーターにベーステンプレートの出力を保管します。

```
KeystoneBase:
  type: ../../puppet/services/keystone.yaml
```

これにより、コンテナー化されたテンプレートは、ベース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一覧を定義するための **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一覧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一覧として定義されます。

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一覧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一覧を上書きすることができます。

6.2. ユーザー定義のコンポーザブルサービスの作成

本項では、ユーザー定義のコンポーザブルサービスの作成方法を考察し、その日のメッセージ (motd: message of the day) サービスの実装に重点を置いて説明します。以下の例では、設定フックを使用するか、オーバークラウドイメージを編集して、そのイメージにカスタムの **motd** Puppet モジュールが読み込まれています。詳細は、「3章*オーバークラウドイメージに関する操作*」を参照してください。

独自のサービスを作成する場合は、サービスの heat テンプレートで次の項目を定義する必要があります。

parameters

以下のパラメーターは、サービステンプレートに追加する必要のある必須パラメーターです。

- **ServiceNetMap:** サービスからネットワークへのマッピング。このパラメーターは、親の Heat テンプレートからの値で上書きされるので、空のハッシュ (**{}**) を **default** 値として使用します。

- **DefaultPasswords:** デフォルトパスワードの一覧。このパラメーターは、親の Heat テンプレートからの値で上書きされるので、空のハッシュ (**{}**) を **default** 値として使用します。

- **EndpointMap:** OpenStack サービスエンドポイントからプロトコルへのマッピングの

一覧。このパラメーターは、親の Heat テンプレートからの値で上書きされるので、空のハッシュ ({}) を default 値として使用します。

作成するサービスが必要とする追加のパラメーターを定義してください。

outputs

以下の出力パラメーターは、ホスト上でのサービス設定を定義します。詳細は、「*付録A コンポーザブルサービスのパラメーター*」を参照してください。

以下は、motd サービス用の heat テンプレート(service.yaml)の一例です。

```
heat_template_version: 2016-04-08

description: >
  Message of the day service configured with Puppet

parameters:
  ServiceNetMap:
    default: {}
    type: json
  DefaultPasswords:
    default: {}
    type: json
  EndpointMap:
    default: {}
    type: json
  MotdMessage: (1)
    default: |
      Welcome to my Red Hat OpenStack Platform environment!

    type: string
    description: The message to include in the motd

outputs:
  role_data:
    description: Motd role using composable services.
    value:
      service_name: motd
      config_settings: (2)
        motd::content: {get_param: MotdMessage}
      step_config: | (3)
        if hiera('step') >= 2 {
          include ::motd
        }
```

(1)

このテンプレートには、その日 のメッセージを定義する **MotdMessage** パラメーターが含まれています。このパラメーターにはデフォルトのメッセージが含まれていますが、カスタムの環境ファイルで同じパラメーターを使用して上書きすることができます。

2

outputs セクションは、**config_settings** 内の一部のサービスの hieradata を定義します。**motd::content** hieradata には、**MotdMessage** パラメーターからのコンテンツが保管されます。**motd** Puppet クラスは、最終的にこの hieradata を読み取り、ユーザー定義のメッセージを **/etc/motd** ファイルに渡します。

3

outputs セクションの **step_config** には、Puppet マニフェストのスニペットが記載されています。このスニペットは、設定がステップ 2 に達したかどうかをチェックし、達している場合には、**motd** Puppet クラスを実行します。

6.3. ユーザー定義のコンポーザブルサービスの追加

カスタム **motd** サービスは、オーバークラウドのコントローラーノードでのみ設定できます。そのためには、カスタムの環境ファイルとカスタムのロールデータファイルをデプロイメントに追加する必要があります。実際の要件に応じて、この手順の入力例を置き換えます。

手順

1. **OS::TripleO::Services** 名前空間内の登録済み heat リソースとして、新規サービスを環境ファイル **env-motd.yaml** に追加します。この例では、**motd** サービスのリソース名は **OS::TripleO::Services::Motd** です。

```
resource_registry:
  OS::TripleO::Services::Motd: /home/stack/templates/motd.yaml

parameter_defaults:
  MotdMessage: |
    You have successfully accessed my Red Hat OpenStack Platform environment!
```

このカスタム環境ファイルには、デフォルトの **MotdMessage** を上書きする新しいメッセージ も含まれています。

デプロイメントに **motd** サービスが追加されました。ただし、この新規サービスを必要とする各ロールは、カスタムの **roles_data.yaml** ファイルにある **ServicesDefault** リストを更新する必要があります。

2. **デフォルトの roles_data.yaml ファイルのコピーを作成します。**

```
$ cp /usr/share/openstack-tripleo-heat-templates/roles_data.yaml ~/custom_roles_data.yaml
```

3. **このファイルを編集するには、Controller ロールにスクロールし、ServicesDefault リストにサービスを追加します。**

```
- name: Controller
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::CephExternal
...
    - OS::TripleO::Services::FluentdClient
    - OS::TripleO::Services::VipHosts
    - OS::TripleO::Services::Motd          # Add the service to the end
```

4. **オーバークラウドの作成時には、編集した環境ファイルと custom_roles_data.yaml ファイルを他の環境ファイルおよびデプロイメントオプションとともに追加します。**

```
$ openstack overcloud deploy --templates -e /home/stack/templates/env-motd.yaml -r
~/custom_roles_data.yaml [OTHER OPTIONS]
```

このコマンドにより、デプロイメントにカスタムの motd サービスが追加され、コントローラーノードのみでサービスが設定されます。

第7章 認定済みコンテナーイメージのビルド

パートナー向け ビルドサービス を使用して、認定用にアプリケーションコンテナーをビルドすることができます。ビルドサービス では、SSH キーによりパブリックまたはプライベートにインターネットアクセス可能な Git リポジトリーから、コンテナーをビルドします。

本項では、Red Hat OpenStack and NFV Zone の一部である自動 ビルドサービス を使用して、コンテナー化されたパートナープラットフォームプラグインを Red Hat OpenStack Platform 13 ベースコンテナーに自動的にビルドする手順について説明します。

前提条件

- Red Hat Connect for Technology Partners に登録する
- Red Hat OpenStack & NFV ゾーンへのゾーンアクセスを申請する
- 製品を作成する(提供された情報は、当社のカタログに認定を公開する際に使用されます)
- コンテナーに含める Dockerfile およびあらゆるコンポーネントと共に、プラグイン用の git リポジトリーを作成する

Red Hat Connect サイトに登録またはアクセスする際に問題が発生した場合は、Red Hat Technology Partner Success Desk にお問い合わせください。

7.1. コンテナープロジェクトの追加

1 つのプロジェクトが 1 つのパートナーイメージに対応します。イメージが複数ある場合には、複数のプロジェクトを作成する必要があります。

手順

1. Red Hat Connect for Technology Partners にログインし、Zones をクリックします。
2. 下方向にスクロールして、Red Hat OpenStack & NFV ゾーンを選択します。ボックスのどこかをクリックします。

3. **Certify** をクリックし、ご自分の会社の既存製品およびプロジェクトにアクセスします。

4. **Add Project** をクリックし、新規プロジェクトを作成します。

5. **Project Name** を設定します。

   - プロジェクト名は、システム外には公開されません。

   - プロジェクト名には [product][version]-[extended-base-container-image]-[your-plugin] が含まれている必要があります。

   - OpenStack の場合、フォーマットは rhospXX-baseimage-myplugin です。

   - 例: rhosp13-openstack-cinder-volume-myplugin

6. ご自分の製品またはプラグイン、およびそのバージョンを元に、**Product**、**Product Version**、および **Release Category** を選択します。

   - プロジェクトを作成する前に、製品とそのバージョンを作成します。

   - ラベルのリリースカテゴリーは、**Tech Preview** に設定します。Red Hat Certification を使用した API テストが完了するまで、**Generally Available** オプションを選択することはできません。コンテナーイメージが認定されたら、プラグイン認定要件を参照してください。

7. パートナープラグインで変更するベースイメージを元に、**Red Hat Product** および **Red Hat Product Version** を選択します。今回のリリースでは、**Red Hat OpenStack Platform** および **13** を選択してください。

8. **Submit** をクリックし、新規プロジェクトを作成します。

結果:

Red Hat はプロジェクトを評価しその認定を確認します。

アップストリームコードに関してプラグインが ツリー内 か ツリー外 かを記載して、connect@redhat.com にメールを送信してください。

- ツリー内 とは、プラグインが OpenStack アップストリームコードベースに含まれ、プラグインイメージが Red Hat によりビルドされ Red Hat OpenStack Platform 16.1 で配布されることを意味します。
- ツリー外 とは、プラグインイメージが OpenStack アップストリームコードベースに含まれず、RHOSP 16.1 では配布されないことを意味します。

7.2. コンテナー認定チェックリストへの準拠

認定済みコンテナーは、パッケージング、配布、およびメンテナンスに関する Red Hat の基準を満たす必要があります。Red Hat によって認定されたコンテナーは、高いレベルの信頼性と Red Hat OpenStack Platform (RHOSP) を含むコンテナー対応プラットフォームからのサポート性を提供します。これを維持するには、パートナーはイメージを最新の状態に保つ必要があります。

手順

1. Certification Checklist をクリックします。
2. チェックリストのすべてのセクションを完了します。チェックリストの項目の詳細情報が必要な場合は、左側のドロップダウン矢印をクリックして、項目の情報や他のリソースへのリンクを表示してください。

Certified

Update your company profile	✓	EDIT
Update your product profile	✓	EDIT
Accept the OpenStack Appendix	✓	EDIT
Update your project profile	✓	EDIT
Package and test your application as a con...		LEARN MORE
Upload documentation and marketing mat...		START
Provide a container registry namespace	✓	EDIT
Provide sales contact information	✓	EDIT
Obtain distribution approval from Red Hat		START
Configure Automated Build Service		START

チェックリストには、以下の項目が含まれます。

Update your company profile

会社プロファイルが最新の状態であることを確認してください。

Update your product profile

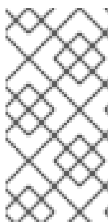
このページは、製品種別、説明、リポジトリーの URL、バージョン、および連絡先リストなどの製品プロファイルの詳細を定義します。

Accept the OpenStack Appendix

コンテナーに関する諸条件です。

Update project profile

自動公開、レジストリー名前空間、リリースカテゴリー、サポート対象プラットフォームなどのイメージ設定が正しいことを確認してください。



注記

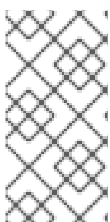
Supported Platforms セクションでは、このページの他の必須フィールドを保存できるように、オプションを選択する必要があります。

Package and test your application as a container

このページの指示に従って、ビルドサービスを設定します。ビルドサービスを使用するには、これまでのステップを完了している必要があります。

Upload documentation and marketing materials

これにより、製品ページにリダイレクトされます。下にスクロールし、**Add new Collateral** をクリックして製品情報をアップロードします。



注記

少なくとも 3 つのマテリアルを指定する必要があります。最初のマテリアルは ドキュメント タイプである必要があります。

Provide a container registry namespace

この名前空間は、プロジェクトプロファイルページと同じです。

Provide sales contact information

この情報は、会社プロファイルと同じです。

Obtain distribution approval from Red Hat

Red Hat は、このステップの許可を与えます。

Configure Automated Build Service

コンテナーイメージのビルドおよびスキャンを実施するための設定情報です。

チェックリストの最後の項目は Configure Automated Build Service です。このサービスを設定するためには、プロジェクトに Red Hat の認定基準に適合する Dockerfile が含まれていなければなりません。

7.3. DOCKERFILE の要件

イメージビルドプロセスの一環として、ビルドサービスはビルドイメージをスキャンし、Red Hat の基準に適合していることを確認します。プロジェクトに含める Dockerfile のガイドラインを以下に示します。

- ベースイメージは、Red Hat イメージでなければなりません。Ubuntu、Debian、および CentOS をベースとして使用するイメージはすべて、スキャナーをパスしません。
- 必須のラベルを設定する必要があります。
  - name
  - maintainer
  - vendor
  - version
  - release
  - summary
-

イメージ内に、テキストファイル形式のソフトウェアライセンスを含める必要があります。ソフトウェアライセンスは、プロジェクトのルート下の ライセンス ディレクトリーに追加します。

- **root** ユーザーではないユーザーを設定する必要があります。

スキャンに必要な情報を、以下の **Dockerfile** の例に示します。

```
FROM registry.redhat.io/rhosp13/openstack-cinder-volume
MAINTAINER VenderX Systems Engineering <maintainer@vendorX.com>

###Required Labels
LABEL name="rhosp13/openstack-cinder-volume-vendorx-plugin" \
    maintainer="maintainer@vendorX.com" \
    vendor="VendorX" \
    version="3.7" \
    release="1" \
    summary="Red Hat OpenStack Platform 13.0 cinder-volume VendorX PluginY" \
    description="Red Hat OpenStack Platform 13.0 cinder-volume VendorX PluginY"


USER root

###Adding package
###repo exmple
COPY vendorX.repo /etc/yum.repos.d/vendorX.repo

###adding package with curl
RUN curl -L -o /verdorX-plugin.rpm http://vendorX.com/vendorX-plugin.rpm

###adding local package
COPY verdorX-plugin.rpm /

# Enable a repo to install a package
RUN yum clean all
RUN yum-config-manager --enable rhel-7-server-openstack-13-rpms
RUN yum install -y vendorX-plugin
RUN yum-config-manager --disable rhel-7-server-openstack-13-rpms

# Add required license as text file in Liceses directory (GPL, MIT, APACHE, Partner End User
Agreement, etc)
RUN mkdir /licenses
COPY licensing.txt /licenses

USER cinder
```

7.4. プロジェクト詳細の設定

コンテナーイメージの名前空間とレジストリーを含め、プロジェクトの詳細を設定する必要がありま

す。

手順

1. **Project Settings** をクリックします。

2. プロジェクト名が正しい形式であることを確認します。認定に合格したコンテナーを自動的に公開する場合には、オプションとして **Auto-Publish** を **ON** に設定します。認定済みコンテナーは、**Red Hat Container Catalog** に公開されます。

Project Name *

MyProject

Current project name: **OS 13+ Test Project**

Auto-Publish

Once a container is certified it is automatically published. Auto-publish must be enabled in order to set up automatic rebuilds.

ON

A container must be pushed to begin the auto-publish process.
Auto-publish is always enabled when **auto-rebuilding** is enabled.

3. **Container Registry Namespace** を設定するには、オンラインの指示に従ってください。

Container Registry Namespace

mycompany

This should be your company name or abbreviation. For example, if your company is *Acme Corporation*, you can use names like *acme*, *acmecorp*, or *acme-corp*. This value is only editable when your company has no published containers in any project.

- Must be unique.
- Must be lowercase.
- Cannot contain special characters other than hyphens (-).
- Must start with a letter.
- Must be 64 characters or less.

- コンテナーレジストリー名前空間には、ご自分の会社の名前を設定します。

- 最終的なレジストリー URL は registry.connect.redhat.com/namespace/repository:tag です。

- 例: registry.connect.redhat.com/mycompany/rhosp16-openstack-cinder-volume-myplugin:1.0

4. **Outbound Repository Name** および **Outbound Repository Descriptions** を設定するには、画面に表示される指示に従ってください。アウトバウンドリポジトリー名は、プロジェクト名と同じでなければなりません。

Outbound Repository Name

rhosp13-openstack-cinder-volume-myplugin

This should represent your product (or the component if your product consists of multiple containers) and a major version. For example, you could use names like *jboss-server7*, or *agent5*. This value is only editable when there are no published containers in this project.

- Must be unique.
- Must be lowercase.
- Cannot contain special characters other than hyphens (-).
- Must start with a letter.
- Must be 64 characters or less.

- [product][version]-[extended_base_container_image]-[your_plugin]
- OpenStack の場合、フォーマットは rhospXX-baseimage-myplugin です。
- 最終的なレジストリーの URL は、registry.connect.redhat.com/namespace/repository:tag になります。
- 例: registry.connect.redhat.com/mycompany/rhosp13-openstack-cinder-volume-myplugin:1.0

5. 該当するフィールドに、プロジェクトに関する補足情報を追加します。
   - Repository Description
   - Supporting Documentation for Primed
6. Submit をクリックします。

7.5. ビルドサービスを使用したコンテナーイメージのビルド

パートナープラグインのコンテナーイメージをビルドします。

手順

1. Build Service をクリックします。
2. Configure Build Service をクリックして、ビルドの詳細を設定します。
   a. Red Hat Container Build を必ず ON に設定します。

b. **Git Source URL を追加します。お使いの git リポジトリーが保護されている場合には、オプションとして Source Code SSH Key を追加します。お使いの git リポジトリーが保護されている場合には、オプションとして Source Code SSH Key を追加します。保護されている git リポジトリーの場合には、SSH を使用する必要があります。**

c. **オプション： Dockerfile Name を追加します。Dockerfile の名前が Dockerfile の場合には、空欄のままにします 。**

d. **(オプション) Docker ビルドのコンテキストルートが git リポジトリーのルートではない場合は、Context Directory を追加します。そうでなければ、このフィールドは空欄のままにします。**

e. **コンテナーイメージのベースとする git リポジトリーの Branch を設定します。**

f. **Submit をクリックして、Build Service の設定を確定します。**

3. **Start Build をクリックします。**

4. **Tag Name を追加し、Submit をクリックします。ビルドが完了するのに、6 分程度かかる場合があります。**

- **タグ名は、プラグインのバージョンに設定する必要があります。**

- **最終的な参照先 URL は、registry.connect.redhat.com/namespace/repository:tag になります。**

- **例: registry.connect.redhat.com/mycompany/rhosp13-openstack-cinder-volume-myplugin:1.0**

5. **Refresh をクリックし、ご自分のビルドが完了したことを確認します。(オプション) 対応する Build ID をクリックして、ビルド情報およびログを表示します。**

6. **ビルドサービスは、イメージのビルドおよびスキャンの両方を行います。このプロセスには、通常 10 - 15 分かかります。スキャンが完了したら、View リンクをクリックしてスキャン**

結果を展開します。

7.6. エラーの発生したスキャン結果の修正

Scan Details のページには、失敗した項目を含めスキャン結果が表示されます。イメージのスキャンにより **FAILED** のステータスが報告される場合には、以下の手順を使用して、エラーを修正する方法を確認してください。

手順

1. **Container Information** のページで **View** のリンクをクリックし、スキャン結果を展開します。

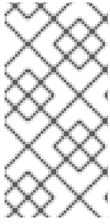
2. エラーの発生した項目をクリックします。たとえば、以下のスクリーンショットでは、**has_licenses** のチェック に失敗します。

Scan Details

▾ Assessments

Name	Value ▲
has_licenses	X
not_running_privileged	✓
rpm_list_successful	✓
rpm_verify_successful	✓
is_rhel	✓
vendor_label_exists	✓
free_of_critical_vulnerabilities	✓
good_tags	✓
good_layer_count	✓
release_label_exists	✓
not_running_as_root	✓
version_label_exists	✓
name_label_exists	✓

3. エラーの発生した項目をクリックして **Policy Guide** の該当するセクションを表示し、問題を修正する方法の詳細を確認します。



注記

Policy Guide にアクセスする際に Access Denied の警告が表示される場合には、connect@redhat.com にメールしてください。

7.7. コンテナーイメージの公開

コンテナーイメージがスキャンに合格したら、コンテナーイメージを公開することができます。

手順

1. Container Information のページで、Publish のリンクをクリックしてコンテナーイメージを一般に公開します。

2. Publish のリンクが Unpublish に変わります。コンテナーの公開を取り消すには、Unpublish のリンクをクリックします。

リンクを公開したら、プラグインの認定に関する詳細を認定ドキュメントで確認してください。認定ドキュメントへのその他のリンクについては、「パートナーインテグレーションの前提条件」を参照してください。

7.8. ベンダープラグインのデプロイ

サードパーティーの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。

1. registry.connect.redhat.com カタログにログインします。

```
$ docker login registry.connect.redhat.com
```

2. プラグインをダウンロードします。

```
$ docker pull registry.connect.redhat.com/dellemc/openstack-cinder-volume-dellemc-rhosp13
```

3. OpenStack デプロイメントに該当するアンダークラウドの IP アドレスを使用して、イメー

ジをタグ付けしてローカルのアンダークラウドレジストリーにプッシュします。

```
$ docker tag registry.connect.redhat.com/dellemc/openstack-cinder-volume-dellemc-rhosp13
192.168.24.1:8787/dellemc/openstack-cinder-volume-dellemc-rhosp13

$ docker push 192.168.24.1:8787/dellemc/openstack-cinder-volume-dellemc-rhosp13
```

4. 以下のパラメーターが含まれる追加の環境ファイルを指定して、オーバークラウドをデプロイします。

```
parameter_defaults:
  DockerCinderVolumeImage: 192.168.24.1:8787/dellemc/openstack-cinder-volume-dellemc-
rhosp13
```

第8章 OPENSTACK コンポーネントの統合と DIRECTOR およびオーバークラウドとの関係

特定の統合ポイントに関する以下の概念を使用して、Red Hat OpenStack Platform (RHOSP) とハードウェアおよびソフトウェアの統合を開始します。

8.1. BARE METAL PROVISIONING (IRONIC)

director 内の OpenStack Bare Metal Provisioning (ironic) コンポーネントを使用して、ノードの電源状態を制御します。director はバックエンドドライバーのセットを使用して、固有のベアメタルの電源コントローラーとやりとりをします。これらのドライバーは、ハードウェアやベンダー固有の拡張や機能を有効化する際に重要です。最も一般的なドライバーは IPMI ドライバーpxe_ipmitool で、Intelligent Platform Management Interface(IPMI)をサポートするサーバーの電源状態を制御します。

Bare Metal Provisioning との統合は、アップストリームの OpenStack コミュニティーから始まります。アップストリームで受け入れられた ironic ドライバーは、コアの RHOSP 製品と director にデフォルトで自動的に含まれます。ただし、認定要件によりサポートされない可能性があります。

機能が継続して確保されるように、ハードウェアドライバーは、常に統合テストを受ける必要があります。サードパーティー製のドライバーのテストおよび適性に関する詳細は、OpenStack コミュニティーページ Ironic/Testing を参照してください。

アップストリームのリポジトリー:

- OpenStack: http://git.openstack.org/cgit/openstack/ironic/
- GitHub: https://github.com/openstack/ironic/

アップストリームのブループリント:

- Launchpad: http://launchpad.net/ironic

Puppet モジュール:

- GitHub: https://github.com/openstack/puppet-ironic

Bugzilla コンポーネント:

- **openstack-ironic**
- **python-ironicclient**
- **python-ironic-oscplugin**
- **openstack-ironic-discoverd**
- **openstack-puppet-modules**
- **openstack-tripleo-heat-templates**

統合メモ:

- アップストリームプロジェクトでは、**ironic/drivers** ディレクトリーにドライバーが含まれます。
- director は、JSON ファイルで定義されたノードをまとめて登録します。os-cloud-config ツール https://github.com/openstack/os-cloud-config/ は、このファイルを解析して、ノード登録の詳細を判断して登録を実行します。これは、**os-cloud-config** ツール、具体的には **nodes.py** ファイルにはドライバーのサポートが必要です。
- director は、Bare Metal Provisioning を使用するように自動的に設定されます。つまり、Puppet 設定では、変更をほぼ加える必要はないということです。ただし、ドライバーが Bare Metal Provisioning に含まれる場合には、お使いのドライバーを **/etc/ironic/ironic.conf** ファイルに追加する必要があります。このファイルを編集して **enabled_drivers** パラメーターを検索してください。

```
enabled_drivers=pxe_ipmitool,pxe_ssh,pxe_drac
```

これにより、Bare Metal Provisioning は drivers ディレクトリーから指定のドライバーを使用できます。

8.2. NETWORKING (NEUTRON)

OpenStack Networking (neutron) は、クラウド環境でネットワークアーキテクチャーを作成する機能を提供します。このプロジェクトは、Software Defined Networking (SDN) ベンダーの統合ポイントを複数提供します。この統合ポイントは通常プラグインまたはエージェントのカテゴリーに分類されます。

プラグインでは、既存の neutron の機能を拡張およびカスタマイズすることができます。ベンダーは、プラグインを記述して、neutron と認定済みのソフトウェアやハードウェアの間で相互運用性を確保することができます。独自のドライバーを統合するためのモジュラーバックエンドを提供する、neutron の Modular Layer 2 (ml2) プラグインのドライバーを開発します。

エージェントでは、固有のネットワーク機能が提供されます。メインの neutron サーバーおよびそのプラグインは、neutron エージェントと通信します。既存の例には、DHCP、Layer 3 のサポート、ブリッジサポートが含まれます。

プラグインとエージェントの両方で、次のいずれかのオプションを選択できます。

- Red Hat OpenStack Platform (RHOSP) ソリューションの一部としてディストリビューションに含める。
- RHOSP のディストリビューションの後にオーバークラウドのイメージに追加する。

認定済みのハードウェアおよびソフトウェアを統合する方法を判断するために、既存のプラグインおよびエージェントの機能を分析します。特に、ml2 プラグインの一部としてドライバーをまず開発することを推奨します。

アップストリームのリポジトリー:

- OpenStack: http://git.openstack.org/cgit/openstack/neutron/

- GitHub: https://github.com/openstack/neutron/

アップストリームのブループリント:

- Launchpad: http://launchpad.net/neutron

Puppet モジュール:

- GitHub: https://github.com/openstack/puppet-neutron

Bugzilla コンポーネント:

- openstack-neutron
- python-neutronclient
- openstack-puppet-modules
- openstack-tripleo-heat-templates

統合メモ:

- アップストリームの neutron プロジェクトには、複数の統合ポイントが含まれます。
  - プラグインは neutron/plugins/ にあります。
  - ml2 プラグインドライバーは neutron/plugins/ml2/drivers/ にあります。

- エージェントは neutron/agents/ にあります。

- OpenStack Liberty リリース以降、ベンダー固有の ml2 プラグインの多くが networking- で始まる独自のリポジトリーに移動されました。たとえば、Cisco 固有のプラグインは https://github.com/openstack/networking-cisco にあります。

- puppet-neutron リポジトリーには、これらの統合ポイントを設定するための個別のディレクトリーも含まれます。
  - プラグイン設定は manifests/plugins/ にあります。
  - ml2 プラグインのドライバー設定は manifests/plugins/ml2/ にあります。
  - エージェントの設定は manifests/agents/ にあります。

- puppet-neutron リポジトリーには、設定関数のライブラリーが別途多数含まれています。たとえば、neutron_plugin_ml2 ライブラリーは、ml2 プラグインの設定ファイルに属性を追加する関数を追加します。

8.3. BLOCK STORAGE (CINDER)

OpenStack Block Storage (cinder) は、Red Hat OpenStack Platform (RHOSP) がボリュームの作成に使用するブロックストレージデバイスと対話する API を提供します。たとえば、Block Storage はインスタンスの仮想ストレージデバイスを提供します。Block Storage は、異なるストレージハードウェアおよびプロトコルをサポートするドライバーのコアセットを提供します。たとえば、コアのドライバーには、NFS、iSCSI、Red Hat Ceph Storage へのサポートを含むものもあります。ベンダーは、追加の認定済みのハードウェアをサポートするためにドライバーを含めることができます。

ベンダーの開発するドライバーおよび設定には、以下に示すように、主に 2 つのオプションがあります。

- RHOSP ソリューションの一部としてディストリビューションに含める。

- RHOSP のディストリビューションの後にオーバークラウドのイメージに追加する。

認定済みのハードウェアおよびソフトウェアを統合する方法を判断するために、既存のドライバーの機能を分析します。

アップストリームのリポジトリー:

- OpenStack: http://git.openstack.org/cgit/openstack/cinder/
- GitHub: https://github.com/openstack/cinder/

アップストリームのブループリント:

- Launchpad: http://launchpad.net/cinder

Puppet モジュール:

- GitHub: https://github.com/openstack/puppet-cinder

Bugzilla コンポーネント:

- openstack-cinder
- python-cinderclient
- openstack-puppet-modules
- openstack-tripleo-heat-templates

統合メモ:

- アップストリームの cinder リポジトリーでは cinder/volume/drivers/ にドライバーが含まれます。

- puppet-cinder リポジトリーには、ドライバー設定の主要なディレクトリーが 2 つ含まれます。

  - manifests/backend ディレクトリーには、ドライバーの設定を行う定義型のセットが含まれます。

  - manifests/volume ディレクトリーには、デフォルトのブロックストレージデバイスを設定するクラスセットが含まれます。

- puppet-cinder リポジトリーには、Cinder 設定ファイルに属性を追加するための cinder_config と呼ばれるライブラリーが含まれます。

8.4. IMAGE STORAGE (GLANCE)

OpenStack Image サービス (glance) は、イメージのストレージを提供するためにストレージ種別と対話する API を提供します。Image サービスは、異なるストレージハードウェアおよびプロトコルをサポートするドライバーのコアセットを提供します。たとえば、コアドライバーには、ファイル、OpenStack Object Storage (swift)、OpenStack Block Storage (cinder)、および Red Hat Ceph Storage のサポートが含まれます。ベンダーは、追加の認定済みのハードウェアをサポートするためにドライバーを含めることができます。

アップストリームのリポジトリー:

- OpenStack:

  - http://git.openstack.org/cgit/openstack/glance/

  - http://git.openstack.org/cgit/openstack/glance_store/

- GitHub:

    - https://github.com/openstack/glance/
    - https://github.com/openstack/glance_store/

アップストリームのブループリント:

- Launchpad: http://launchpad.net/glance

Puppet モジュール:

- GitHub: https://github.com/openstack/puppet-glance

Bugzilla コンポーネント:

- openstack-glance
- python-glanceclient
- openstack-puppet-modules
- openstack-tripleo-heat-templates

統合メモ:

- Image サービスは、統合ポイントを含む Block Storage を使用してイメージストレージを管理できるため、ベンダー固有のドライバーを追加する必要はありません。
- アップストリームの glance_store リポジトリーでは glance_store/_drivers にドライバーが含まれます。

- puppet-glance リポジトリーでは manifests/backend ディレクトリーにドライバー設定が含まれます。

- puppet-glance リポジトリーには、Glance 設定ファイルに属性を追加するための glance_api_config と呼ばれるライブラリーが含まれます。

8.5. SHARED FILE SYSTEMS (MANILA)

OpenStack Shared File System Service (Manila) は、共有および分散型のファイルシステムサービス向けの API を提供します。ベンダーは、認定済みのハードウェアのサポートを追加するためにドライバーを含めることができます。

アップストリームのリポジトリー:

- OpenStack: http://git.openstack.org/cgit/openstack/manila/

- GitHub: https://github.com/openstack/manila/

アップストリームのブループリント:

- Launchpad: http://launchpad.net/manila

Puppet モジュール:

- GitHub: https://github.com/openstack/puppet-manila

Bugzilla コンポーネント:

- openstack-manila

- **python-manilaclient**
- **openstack-puppet-modules**
- **openstack-tripleo-heat-templates**

統合メモ:

- アップストリームの manila リポジトリーでは manila/share/drivers/ にドライバーが含まれます。
- puppet-manila リポジトリーでは manifests/backend ディレクトリーにドライバー設定が含まれます。
- puppet-manila リポジトリーには、manila 設定ファイルに属性を追加するための manila_config と呼ばれるライブラリーが含まれます。

8.6. OPENSHIFT ON OPENSTACK

Red Hat OpenStack Platform (RHOSP) では OpenShift-on-OpenStack のデプロイメントをサポートする方針です。これらのデプロイメントのパートナーインテグレーションの詳細については、Red Hat OpenShift パートナー のページを参照してください。

付録A コンポーザブルサービスのパラメーター

以下のパラメーターは、すべてのコンポーザブルサービスの出力に使用されます。

- service_name
- config_settings
- service_config_settings
- global_config_settings
- step_config
- upgrade_tasks
- upgrade_batch_tasks

以下のパラメーターは、特にコンテナー化されたコンポーザブルサービスの出力に使用されます。

- puppet_config
- kolla_config
- docker_config
- docker_puppet_tasks

- host_prep_tasks
- fast_forward_upgrade_tasks

A.1. すべてのコンポーザブルサービス

以下のパラメーターは、すべてのコンポーザブルサービスに適用されます。

service_name

サービスの名前。このパラメーターを使用して、service_config_settings により、他のコンポーザブルサービスからの設定を適用することができます。

config_settings

作成するサービス用のカスタム hieradata 設定。

service_config_settings

別のサービス用のカスタム hieradata 設定。たとえば、作成するサービスには、OpenStack Identity (keystone) に登録済みのエンドポイントが必要な場合があります。この設定により、1 つのサービスから別のサービスにパラメーターが提供され、サービスが異なるロール上にある場合でも、複数のサービスにまたがった設定が可能になります。

global_config_settings

全ロールに配布されるカスタムの hieradata 設定。

step_config

サービスを設定するための Puppet スニペット。このスニペットは、サービス設定プロセスの各ステップで作成/実行される、統合されたマニフェストに追加されます。ステップは以下のとおりです。

- ステップ 1: ロードバランサーの設定
- ステップ 2: 高可用性および一般のコアサービス (データベース、RabbitMQ、NTP) の設定

- ステップ 3: OpenStack Platform サービスの初期設定 (ストレージ、リングの構築)

- ステップ 4: 一般的な OpenStack Platform サービス

- ステップ 5: サービスのアクティブ化 (Pacemaker) および OpenStack Identity (keystone) のロールとユーザーの作成

参照される Puppet マニフェストでは、step hieradata を使用して (hiera('step') を使用)、デプロイメントプロセスの特定のステップに特定のアクションを定義することができます。

upgrade_tasks

サービスのアップグレードを容易にする Ansible スニペット。スニペットは統合された Playbook に追加されます。それぞれの操作では、以下に示すタグを使用して step を定義します。

- common: すべてのステップに適用される

- step0: 検証

- step1: すべての OpenStack サービスを停止する

- step2: Pacemaker が制御するすべてのサービスを停止する

- step3: パッケージを更新し、新規パッケージをインストールする

- step4: データベースのアップグレードに必要な OpenStack サービスを起動する

- step5: データベースをアップグレードする

upgrade_batch_tasks

upgrade_tasks に類似した機能を持ちますが、リストの順番どおりに Ansible タスクのバッチセットを実施するのみ。デフォルトは 1 ですが、roles_data.yaml ファイルの upgrade_batch_size パラ

メーターを使用して、ロールごとにこの設定を変更することができます。

A.2. コンテナー化されたコンポーザブルサービス

以下のパラメーターは、コンテナー化されたすべてのコンポーザブルサービスに適用されます。

puppet_config

このセクションは、Puppet を使用した設定ファイルの作成をアクティブ化する入れ子状のキーと値のペアのセット。必須のパラメーターは以下のとおりです。

puppet_tags

Puppet を使用して設定ファイルを生成するのに使用される Puppet リソースタグ名。ファイルの生成には、名前の付けられた設定リソースだけが使用されます。タグを指定するすべてのサービスでは、デフォルトのタグ (file、concat、file_line、augeas、cron) が設定に追加されます。例: keystone_config

config_volume

このサービス用に設定ファイルが生成されるボリューム (ディレクトリー) 名。実行中の設定用 Kolla コンテナーにマウントをバインドする場所として、このパラメーターを使用します。

config_image

設定ファイルを生成するのに使用される Docker イメージ名。通常は、ランタイムサービスが使用するコンテナーと同一です。一部のサービスは、共通のベースコンテナーで生成される設定ファイルの共通のセットを共有します。

step_config

この設定により、Puppet を使用して docker 設定ファイルを作成するのに使用されるマニフェストを制御します。このコンテナーの設定ディレクトリー生成には、このマニフェストと共に下記の Puppet タグが使用されます。

kolla_config

コンテナー内の Kolla 設定のマッピングの作成。形式は設定ファイルの絶対パスで始まり、それを以下のサブパラメーターに使用します。

command

コンテナーの起動時に実行するコマンド。

config_files

サービス起動前のサービス設定ファイルの場所 (source) およびコンテナー上の送付先 (dest)。また、コンテナー上でこれらのファイルをマージするか置き換えるか (merge)、ファイルのアクセス権限およびその他のプロパティーを維持するかどうか (preserve_properties) に関するオプションも含まれます。

permissions

コンテナー上の特定ディレクトリーのアクセス権限を設定します。パス、所有者、およびグループが必要です。再帰的にアクセス権限を適用することもできます (recurse)。

以下は、keystone サービス用の kolla_config パラメーターの一例です。

```
kolla_config:
  /var/lib/kolla/config_files/keystone.json:
    command: /usr/sbin/httpd -DFOREGROUND
    config_files:
      - source: "/var/lib/kolla/config_files/src/*"
        dest: "/"
        merge: true
        preserve_properties: true
  /var/lib/kolla/config_files/keystone_cron.json:
    command: /usr/sbin/crond -n
    config_files:
      - source: "/var/lib/kolla/config_files/src/*"
        dest: "/"
        merge: true
        preserve_properties: true
    permissions:
      - path: /var/log/keystone
        owner: keystone:keystone
        recurse: true
```

docker_config

コンテナー設定の各ステップで docker-cmd フックに渡されるデータ

- step_0: Hiera 設定により生成されるコンテナーの設定ファイル
- step_1: ロードバランサーの設定
  a. ベアメタルの設定

b. **コンテナーの設定**

- **step_2: コアサービス (Database/Rabbit/NTP/etc.)**

  a. **ベアメタルの設定**

  b. **コンテナーの設定**

- **step_3: OpenStack サービスの初期設定 (Ringbuilder など)**

  a. **ベアメタルの設定**

  b. **コンテナーの設定**

- **step_4: 一般的な OpenStack サービス**

  a. **ベアメタルの設定**

  b. **コンテナーの設定**

  c. **Keystone コンテナーポストの初期化 (テナント、サービス、エンドポイントの作成)**

- **step_5: サービスのアクティブ化 (Pacemaker)**

  a. **ベアメタルの設定**

  b. **コンテナーの設定**

YAML はパラメーターセットを使用して、各ステップで実行するコンテナーおよび各コンテナーに関

連付けられた docker 設定を定義します。たとえば、以下のようになります。

```
docker_config:
  step_3:
    keystone:
      start_order: 2
      image: *keystone_image
      net: host
      privileged: false
      restart: always
      healthcheck:
        test: /openstack/healthcheck
      volumes: *keystone_volumes
      environment:
        - KOLLA_CONFIG_STRATEGY=COPY_ALWAYS
```

これにより keystone コンテナーが作成され、使用するイメージ、ネットワーク種別、および環境変数などの詳細を定義するための該当パラメーターが使用されます。

docker_puppet_tasks

docker-puppet.py ツールを直接アクティブ化するためのデータを提供します。タスクが実行されるのは、(各ノードではなく) クラスター全体で 1 度だけで、keystone エンドポイントやデータベースユーザーなどの初期化に必要な、さまざまな Puppet スニペットに対して有用です。以下に例を示します。

```
docker_puppet_tasks:
  # Keystone endpoint creation occurs only on single node
  step_3:
    config_volume: 'keystone_init_tasks'
    puppet_tags:
'keystone_config,keystone_domain_config,keystone_endpoint,keystone_identity_provider,keystone_pas
e_ini,keystone_role,keystone_service,keystone_tenant,keystone_user,keystone_user_role,keystone_do
main'
    step_config: 'include ::tripleo::profile::base::keystone'
    config_image: *keystone_config_image
```

host_prep_tasks

これは、コンテナー化されたサービス用にノードホストを準備するためにホスト上で実行する Ansible スニペットです。たとえば、コンテナー作成時に、コンテナーにマウントする特定のディレクトリーを作成しなければならない場合があります。

fast_forward_upgrade_tasks

Fast Forward Upgrade プロセスを容易にする Ansible スニペット。このスニペットは統合された Playbook に追加されます。それぞれの操作では、タグを使用して step および release を定義しま

す。

通常、step は以下のような段階を経ます。

- step=0: 実行中のサービスを確認する
- step=1: サービスを停止する
- step=2: クラスターを停止する
- step=3: リポジトリーを更新する
- step=4: データベースのバックアップ
- step=5: パッケージ更新前コマンド
- step=6: パッケージの更新
- step=7: パッケージ更新後コマンド
- step=8: データベースの更新
- step=9: 検証

tag はリリースに対応します。

- tag=ocata: OpenStack Platform 11

- **tag=pike: OpenStack Platform 12**
- **tag=queens: OpenStack Platform 13**