



# Red Hat OpenStack Platform

## 11

## パートナーのソリューションの統合

---

Red Hat OpenStack Platform 環境における認定済みのサードパーティー製ソフトウェアおよびハードウェアの統合

OpenStack Team



## Red Hat OpenStack Platform 環境における認定済みのサードパーティー製ソフトウェアおよびハードウェアの統合

OpenStack Team  
rhos-docs@redhat.com

## 法律上の通知

Copyright © 2017 Red Hat, Inc.

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, 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 Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

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

All other trademarks are the property of their respective owners.

## 概要

本書は、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や、director を使用したデプロイメントの設定作成など、認定済みのサードパーティー製コンポーネントを Red Hat OpenStack Platform 環境に統合する際の指針を提供します。

## 目次

<b>第1章 はじめに</b>	<b>4</b>
1.1. パートナーのソリューション統合に関する概要	4
1.2. パートナーのソリューション統合の要件	4
<b>第2章 アーキテクチャー</b>	<b>6</b>
2.1. コアコンポーネント	6
2.1.1. IroniC	7
2.1.2. Heat	7
2.1.3. Puppet	9
2.1.4. TripleO および TripleO Heat テンプレート	10
<b>第3章 オーバークラウドイメージ</b>	<b>12</b>
3.1. オーバークラウドイメージの取得	12
3.2. INITRD: 最初の RAMDISK の変更	12
3.3. QCOW: DIRECTOR への VIRT-CUSTOMIZE のインストール	13
3.4. QCOW: オーバークラウドイメージの検査	13
3.5. QCOW: ROOT パスワードの設定	14
3.6. QCOW: イメージの登録	14
3.7. QCOW: サブスクリプションのアタッチと RED HAT リポジトリの有効化	14
3.8. QCOW: カスタムリポジトリファイルのコピー	15
3.9. QCOW: RPM のインストール	15
3.10. QCOW: サブスクリプションプールの消去	16
3.11. QCOW: イメージの登録解除	16
3.12. DIRECTOR へのイメージのアップロード	16
<b>第4章 設定</b>	<b>18</b>
4.1. PUPPET の基礎知識	18
4.1.1. Puppet モジュールの内容の検証	18
4.1.2. サービスのインストール	19
4.1.3. サービスの起動と有効化	20
4.1.4. サービスの設定	20
4.2. OPENSTACK PUPPET モジュールの取得	21
4.3. PUPPET モジュールの設定追加	22
4.4. PUPPET 設定への階層データの追加	23
<b>第5章 オーケストレーション</b>	<b>25</b>
5.1. HEAT テンプレートの基礎知識	25
5.1.1. Heat テンプレートの理解	25
5.1.2. 環境ファイルの理解	26
5.2. デフォルトの DIRECTOR テンプレートの取得	27
5.3. 初回起動での設定のカスタマイズ	28
5.4. オーバークラウドを構成する前の設定のカスタマイズ	29
5.5. オーバークラウド設定後の設定のカスタマイズ	32
5.6. カスタムの PUPPET 設定のオーバークラウドへの適用	33
5.7. オーバークラウドでの HIERA データの変更	35
5.8. 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	36
<b>第6章 コンポーザブルサービス</b>	<b>37</b>
6.1. コンポーザブルサービスアーキテクチャーの考察	37
6.2. ユーザー定義のコンポーザブルサービスの作成	38
6.3. ユーザー定義のコンポーザブルサービスの追加	40
<b>第7章 統合ポイント</b>	<b>42</b>

7.1. BARE METAL PROVISIONING (IRONIC)	42
7.2. NETWORKING (NEUTRON)	43
7.3. BLOCK STORAGE (CINDER)	44
7.4. IMAGE STORAGE (GLANCE)	45
7.5. SHARED FILE SYSTEMS (MANILA)	46
<b>第8章 実例</b> .....	<b>48</b>
8.1. CISCO NEXUS 1000V	48
8.2. NETAPP ストレージ	50



## 第1章 はじめに

本書は、Red Hat OpenStack Platform のパートナーが OpenStack Platform 環境のインストールやデプロイメントライフサイクルの管理に使用するツールとして、Red Hat OpenStack Platform director を使用してソリューションを統合する作業を支援するために執筆されました。director を使用した統合により、パートナーの技術をシームレスに導入することが可能となり、リソースの最適化、デプロイメント所要時間の短縮、ライフサイクル管理コストの削減など、幅広いメリットが得られます。

今後、OpenStack Platform director を使用した統合は、既存のエンタープライズ管理システムおよびプロセスとの充実した統合を提供するための強固な対策の 1 つとなります。director の機能は将来、Red Hat の製品ポートフォリオ内の CloudForms などのツールによって統合/使用されるようになり、サービスデプロイメント管理のより多くのプロセスをグラフィカルユーザーインターフェースで実行できるようになる見通しです。

### 1.1. パートナーのソリューション統合に関する概要

本書は、パートナーがソフトウェアおよびハードウェアソリューションを Red Hat OpenStack Platform に統合するのを支援することを目的としています。この作業は、director でオーバークラウドの一部として設定する方法で行います。手順は、複数のセクションで構成されるワークフロー形式で記載しており、各セクションでは特定の統合タスクの実行方法について説明します。

- ※ **アーキテクチャー:** オーバークラウドの作成や設定を実行する際に director が使用するテクノロジーの一部を検証します。
- ※ **オーバークラウドイメージ:** director はオーバークラウド内の各ノードに、ノード種別の基盤として、ベースのイメージを書き込みます。このセクションでは、デプロイメント前にこれらのイメージを修正してドライバーやソフトウェアを追加できるようにする方法を説明します。これは、アップストリームに寄与する前にドライバーや設定をテストする際などに役立ちます。
- ※ **設定:** director は、主に Puppet モジュールを使用して、オーバークラウド上の各サービスを設定します。このセクションでは、Puppet モジュールがどのように機能し、オーバークラウドの設定にこの Puppet モジュールがどのように使用されるかを説明します。
- ※ **オーケストレーション:** director は、Heat テンプレートセットを使用して、オーバークラウドを作成、設定します。これには、オーバークラウド設定の動作を変更するためのカスタムの環境ファイルや Heat テンプレートなども含まれる場合があります。このセクションでは、そのようなテンプレートを作成してオーバークラウドのカスタムの設定を有効化する方法について重点的に解説します。この操作には、本セクションの 1 つ前の章で説明している Puppet 設定の追加も必要です。
- ※ **統合ポイント:** director がデプロイするイメージには、必須の OpenStack のコンポーネントと設定用の Puppet モジュールセットが含まれます。このセクションでは、コンポーネントドライバーや Puppet モジュールを貢献するためのいくつかのアップストリームプロジェクトについて説明します。アップストリームに貢献されたコンポーネントやモジュールは、Red Hat が検証を行って、今後の Red Hat OpenStack Platform ディストリビューションに追加することができます。
- ※ **実例:** この章では、前の章で得た知識の集大成として、実際の認定済みベンダーが現在 director を使用してオーバークラウドにプロジェクトを統合している方法を実習します。ネットワークおよびストレージの実践的な例もいくつか記載しています。このセクションは、同じようなベンダーが自社製品を Red Hat OpenStack Platform のエコシステムに統合するのに役立ちます。

### 1.2. パートナーのソリューション統合の要件



director を使用して有意義な統合作業を完了させるには、複数の前提条件を満たす必要があります。これらの要件は、技術的な統合に限定されず、さまざまなレベルのパートナーソリューションの文書化も含まれます。これは、統合全体について完全な知識を共有して、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、サポート要員が効率的に作業をサポートできるようにすることが目的です。

最初の要件は、Red Hat OpenStack Platform ソリューションの認定に関連します。パートナーのソリューションを Red Hat OpenStack Platform director を使用して統合するには、まず Red Hat OpenStack Platform で認定される必要があります。

※ [『Red Hat OpenStack Certification Policy Guide』](#)

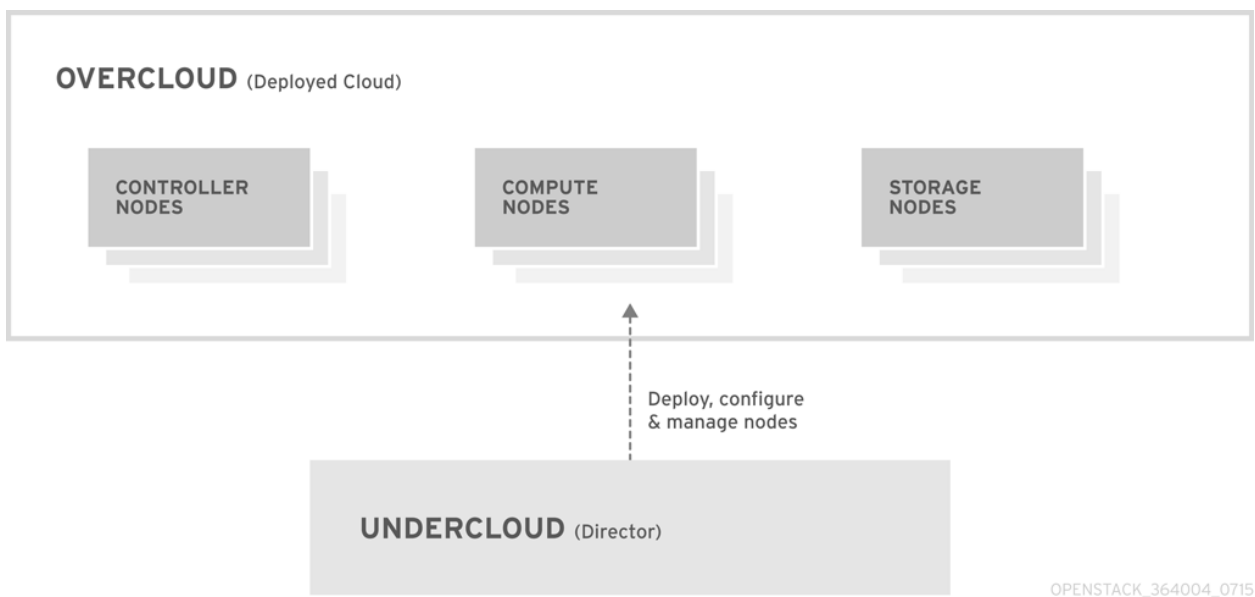
※ [『Red Hat OpenStack Certification Workflow Guide』](#)

## 第2章 アーキテクチャー

director には、OpenStack 環境自体の設定、デプロイ、管理にネイティブの OpenStack API を使用することを推奨します。つまり、director を使用した統合には、それらのネイティブ OpenStack API および補助コンポーネントとの統合が必要となることになります。このような API を使用する主な利点は、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 OpenStack 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。また、OpenStack API を使用すると、director が自動的に OpenStack のコア機能拡張、セキュリティ修正プログラム、バグの修正を自動的に継承することになります。

Red Hat OpenStack Platform director は、完全な OpenStack 環境のインストールおよび管理を行うためのツールセットです。director は、主に OpenStack プロジェクト TripleO (「OpenStack-On-OpenStack」の略語) をベースとしています。このプロジェクトは、OpenStack のコンポーネントを活用して、完全に機能する OpenStack 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をプロビジョニング、制御する新しい OpenStack のコンポーネントが含まれます。

Red Hat OpenStack Platform director は、アンダークラウドとオーバークラウドという 2 つの主要な概念を使用します。この director 自体は、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構成されています。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。オーバークラウドとアンダークラウドに関する情報は、『[director のインストールと使用方法](#)』ガイドを参照してください。



director には、オーバークラウド構成を構築するためのツール、ユーティリティ、テンプレートのサンプルが同梱されています。director は、設定データ、パラメーター、ネットワークポロジの情報を取得し、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。

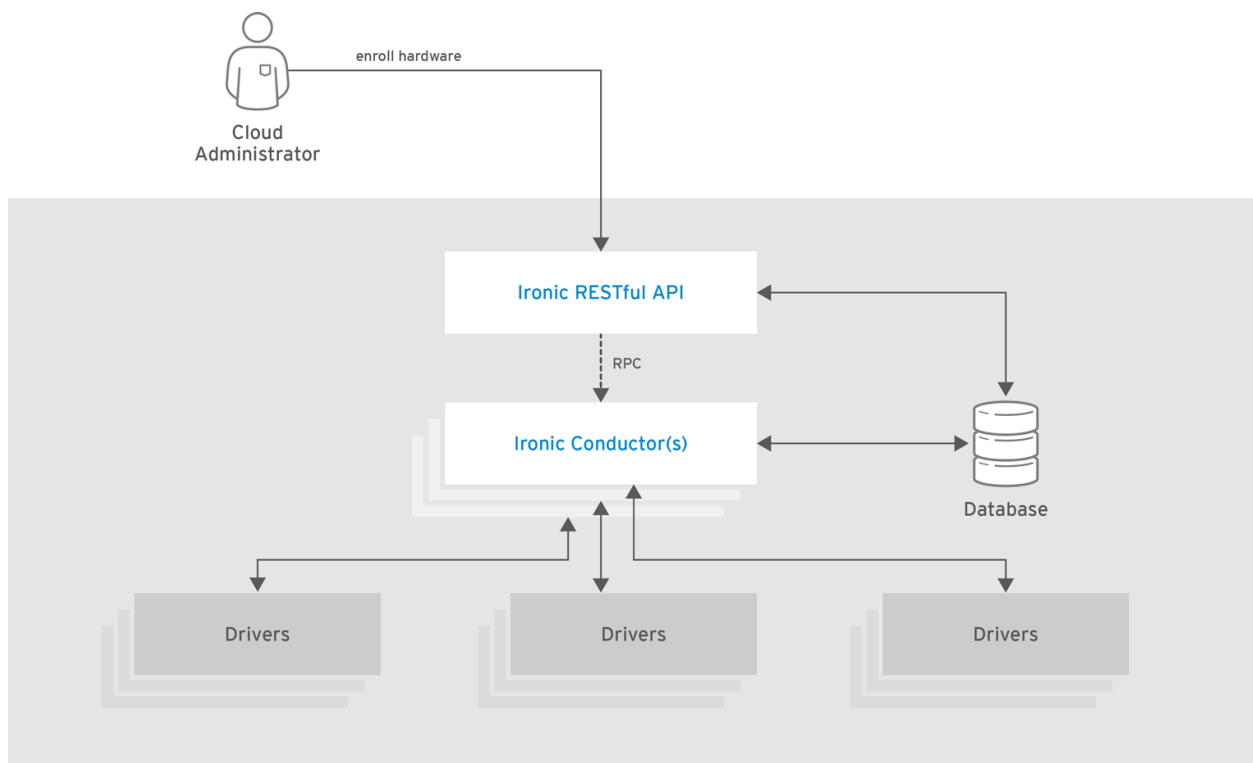
パートナーにはさまざまな異なる要件があります。director のアーキテクチャーを理解すると、統合の際にどのコンポーネントが重要となるかを理解するのに役立ちます。

### 2.1. コアコンポーネント

本セクションでは、Red Hat OpenStack Platform director のコアコンポーネントをいくつか考察して、それらのコンポーネントがオーバークラウドの作成にどのように貢献するのかを説明します。

### 2.1.1. Ironic

Ironic 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を提供します。director は、Ironic 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。Ironic には、ベアメタルノードを定義するネイティブの API があります。director で OpenStack 環境をプロビジョニングする管理者は、特定のドライバーを使用して、Ironic 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が、Ironic 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。Ironic 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取得した情報を使用して、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、ストレージノードとしてプロビジョニングされる可能性が高いです。

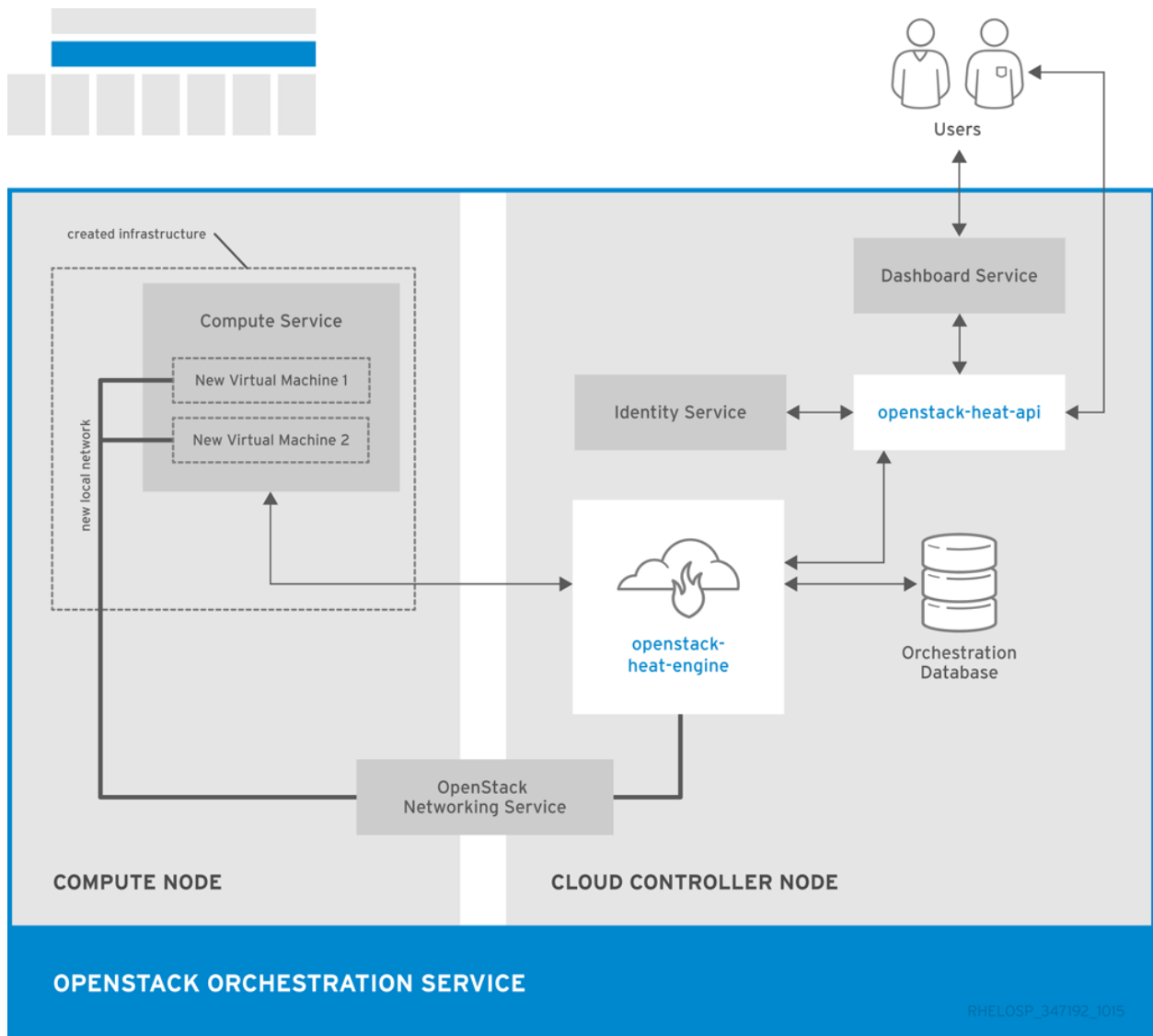


OPENSTACK\_392410\_0216

director でのハードウェアサポートを希望するパートナーは、Ironic のドライバーに対応している必要があります。

### 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として機能します。これにより、組織では、クラウドにデプロイする前に特定のアプリケーションの要素を定義することができます。このプロセスでは、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 は、指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て、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により、アプリケーションスタックは移植可能となり、繰り返し実行して想定通りの結果を得られるようになります。



director は、ネイティブの OpenStack Heat API を使用して、オーバークラウドのデプロイ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。また 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容易に行うためにも 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 Heat ツールを使用して作成中のオーバークラウドのリソース (およびステータス) を表示することができます。たとえば、Heat ツールを使用して、ネストされた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。

Heat には、実稼働向けの OpenStack クラウドを宣言および作成するための幅広く強力な構文がありますが、事前にパートナーのソリューションとの統合に関する知識とスキルが必要です。パートナーのテクノロジーを統合するユースケースにはすべて、Heat のテンプレートが必要です。

### 2.1.3. Puppet

Puppet は、設定管理および実行ツールです。マシンの最終的な状態を記述して、その状態を保持するメカニズムとして使用します。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がローカルで実行され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マニフェストを取得するサーバーモード

管理者は、ノードに新しいマニフェストをアップロードしてローカルで実行する方法と、クライアント/サーバーモデルで Puppet マスターで変更する方法のいずれかを使用して変更を加えます。

Puppet は director のさまざまな場所で使用されています。

- ※ アンダークラウドホストで Puppet をローカルで使用して、**undercloud.conf** に記述された設定に従ってパッケージのインストールや設定を行います。
- ※ ベースのオーバークラウドイメージに **openstack-puppet-modules** パッケージを注入します。これらの Puppet モジュールは、デプロイメント後の設定に使用できます。デフォルトでは、すべての OpenStack サービスが含まれたイメージを作成して、各ノードに使用します。
- ※ Heat 経由で追加の Puppet マニフェストと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、ノードの種別に依存するサービスの有効化や起動、OpenStack の設定の適用などが含まれます。
- ※ 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を検討してください。たとえば、現在の OpenStack Puppet モジュールを取得する方法に関する情報は、「[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

前述したように、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、以下の作業を行う OpenStack サービスセットを統合します。

- ※ オーバークラウドイメージの保存 (Glance)
- ※ オーバークラウドのオーケストレーション (Heat)
- ※ ベアメタルマシンのプロビジョニング (Ironic)

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。また、Heat はこのようなコアの Heat テンプレートに含まれる特定のリソースに対してソフトウェア設定を起動します。このソフトウェア設定は通常、Bash スクリプトか Puppet マニフェストのいずれかです。

通常のソフトウェア設定は、主に 2 つの Heat リソースに依存します。

- ※ 設定を定義するリソース (**OS::Heat::SoftwareConfig**)
- ※ ノードで設定を実装するリソース (**OS::Heat::SoftwareDeployment**)

たとえば、Heat テンプレートコレクションでは、コンピュートノードのデプロイメント後のテンプレート (**puppet/compute-post.yaml**) に以下のセクションが含まれます。

```
resources:

  ComputePuppetConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: puppet
      options:
        enable_debug: {get_param: ConfigDebug}
      outputs:
        - name: result
      config:
        get_file: manifests/overcloud_compute.pp

  ComputePuppetDeployment:
    type: OS::Heat::StructuredDeployments
    properties:
      servers: {get_param: servers}
      config: {get_resource: ComputePuppetConfig}
      input_values:
        update_identifier: {get_param: NodeConfigIdentifiers}
```

**ComputePuppetConfig** リソースは、コンピュートノードの設定が含まれる Puppet マニフェスト (`puppet/manifests/overcloud_compute.pp`) を読み込みます。**ComputePuppetDeployment** リソースは、サービス一覧 (`servers: {get_param: servers}`) に **ComputePuppetConfig** からの設定を適用します。これは Heat の親テンプレートではコンピュートノードとして定義されます。Puppet が正常にマニフェストをすべて適用したかどうかにより、ノードは **ComputePuppetDeployment** の成功/失敗の報告を返します。

このソフトウェアの設定データフローは、director を使用してサードパーティーのソリューションを統合する方法を理解するのに重要です。本ガイドは、このデータフローを使用して、中核となる設定の前後に、オーバークラウドでカスタムを設定を追加する方法を説明します。カスタム設定の実装に使用するソフトウェアの設定データフローの例は、以下を参照してください。

- ※ 「オーバークラウドを構成する前の設定のカスタマイズ」
- ※ 「オーバークラウド設定後の設定のカスタマイズ」

## 第3章 オーバークラウドイメージ

Red Hat OpenStack Platform 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。

本ガイドでは、**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各種アクションについて説明します。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することができます。

### 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/ja/articles/1409973>) に従って、変更を加えていないイメージを使用した状態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際に、複数のディスクが必要です。必要なディスクは以下のとおりです。

- ✦ **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に使用
- ✦ **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に使用
- ✦ **オーバークラウドカーネル、ramdisk、完全なイメージ:** ノードのハードディスクに書き込まれるベースのオーバークラウドシステム

**rhosp-director-images** および **rhosp-director-images-ipa** パッケージからこれらのイメージを取得します。

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

**stack** ユーザーのホーム (`/home/stack/images`) の **images** ディレクトリー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-9.0.tar /usr/share/rhosp-director-images/ironic-python-agent-latest-9.0.tar; do tar -xvf $i; done
```

### 3.2. INITRD: 最初の RAMDISK の変更



場合によっては、内部の 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。以下の手順では、最初の ramdisk を変更する方法を記載しています。オーバークラウドにおいては、これには以下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。

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

```
$ mkdir ipa-tmp
$ cd ipa-tmp
```

**skipcpio** と「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
```

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
```

新しい ramdisk を再作成します。

```
$ find . 2>/dev/null | cpio --quiet -c -o | gzip -8 > ~/images/ironic-
python-agent.initramfs
```

ramdisk に新しいパッケージが存在することを確認します。

```
$ lsinitrd ~/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。**rhel-7-server-rpms** リポジトリから **libguestfs-tools** をインストールします。

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

### 3.4. QCOW: オーバークラウドイメージの検査

**overcloud-full.qcow2** のコンテンツを検査する必要がある場合があります。**qemu-system-x86\_64** コマンドを使用して仮想マシンインスタンスを作成します。

```
$ sudo qemu-system-x86_64 --kernel overcloud-full.vmlinuz --initrd
overcloud-full.initrd -m 1024 --append root=/dev/sda --enable-kvm
overcloud-full.qcow2
```

または、**virt-manager** で以下のブートオプションを使用します。

- ✱ カーネルのパス: /overcloud-full.vmlinuz
- ✱ **initrd** のパス: /overcloud-full.initrd
- ✱ **Kernel** の引数: root=/dev/sda

### 3.5. QCOW: ROOT パスワードの設定

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
```

これにより、管理者レベルのアクセス権限でコンソールを使用してノードにアクセスできるようになります。

### 3.6. QCOW: イメージの登録

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
```

**[username]** および **[password]** は、お客様の Red Hat アカウントの情報に置き換えます。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]
```

このコマンドでは、Red Hat の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。

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

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
```

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
```

**[subscription-pool]** は選択したサブスクリプションプール ID に置き換えるようにしてください。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 attach --pool [subscription-pool]
```

このコマンドではイメージにプールが追加され、以下のコマンドで Red Hat リポジトリを有効化できるようになります。

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

### 3.8. QCOW: カスタムリポジトリファイルのコピー

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、追加のリポジトリが必要です。たとえば、以下は、OpenDaylight リポジトリの内容を使用する設定が含まれたリポジトリファイルの例です。

```
$ cat opendaylight.repo

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

リポジトリ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
```

**--copy-in** オプションは、リポジトリ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 のサポート担当者を確認してください。

### 3.9. QCOW: RPM のインストール

**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** オプションを指定すると、インストールするパッケージを指定することができます。

### 3.10. QCOW: サブスクリプションプールの消去

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
```

これで、イメージからサブスクリプションプールがすべて削除されます。

### 3.11. QCOW: イメージの登録解除

最後に、イメージの登録を解除します。これは、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、各ノードを個別で登録するためです。

```
$ 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. DIRECTOR へのイメージのアップロード

イメージを変更した後は、director にアップロードします。**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てください。

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

これにより、**bm-deploy-kernel**、**bm-deploy-ramdisk**、**overcloud-full**、**overcloud-full-initrd**、**overcloud-full-vmlinuz** のイメージが director にアップロードされます。これらは、デプロイメントとオーバークラウド用のイメージです。このスクリプトにより、director の PXE サーバーにあるイントロスペクションイメージがインストールされます。以下のコマンドを使用して CLI に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 設定

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. PUPPET の基礎知識

次の項では、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 モジュールの内容の検証

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 key-value 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。以下に例を示します。

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

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。されていない場合は、マニフェストにより yum が実行されて、httpd パッケージがインストールされます。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が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一するメソッドとして機能します。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。これらの場所や、パーミッションなどのその他の属性は、マニフェストのファイルのリソース宣言で定義されます。

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

プラグインにより、Puppet のコア機能を拡張することができます。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、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、プラグインのタイプに応じたサブディレクトリーセットが含まれます。以下に例を示します。

- ✳ `/lib/facter`: カスタムファクトの場所
- ✳ `/lib/puppet/type`: 属性の key-value 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せて使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、特定のパッケージの設定を定義するリソース定義を必要とします。

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は、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。これには、以下の内容が含まれるようにマニフェストを編集する必要があります。

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

これにより、以下の操作が実行されます。

- ※ **ensure => running** 属性は、サービスが実行され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

上記の 2 つの手順では、Puppet を使用したサービスのインストールおよび有効化の方法を説明しましたが、サービスにカスタム設定を指定することもできます。本ガイドの例では、ポート 80 に Web ホストを設定するように、すでに `/etc/httpd/conf/httpd.conf` に HTTP サーバーのデフォルト設定が指定されています。このセクションでは、設定を追加して、ユーザー指定のポートに追加の Web ホストを設定します。

そのためには、HTTP 設定ファイルを保存するテンプレートファイルを使用します。これは、ユーザー定義のポートには、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、以下の内容が含まれた **myserver.conf.erb** と呼ばれる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** などです。



**fqdn** が追加されている点に注意してください。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、各該当システムの 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。

このファイルを保存した後は、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",
  }
}
```

これにより、以下の操作が実行されます。

- ※ 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、以前に作成した **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も確認します。
- ※ 2 番目のファイルリソース宣言を追加します。これにより、Web サーバーのディレクトリー (**/var/www/myserver**) が作成されます。
- ※ **notify => Service["httpd"]** 属性を使用して、設定ファイルと httpd サービスの関係も追加します。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、**Github** の **openstack** グループから取得する正式な OpenStack Puppet モジュールを使用します。<https://github.com/openstack> に移動して、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。

この例では、以下のコマンドを使用してクローン作成し、正式な OpenStack Block Storage (**cinder**) を検証します。

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

これにより、Cinder の Puppet モジュールのクローンを作成します。

### 4.3. PUPPET モジュールの設定追加

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が含まれます。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)
}
```

これにより、以下の操作が実行されます。

- ※ **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で定義されます (`$nfs_shares_config = '/etc/cinder/shares.conf'`)。以下は追加の属性です。
- ※ **content** 属性は、`$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;
```

上記の属性により、**cinder.conf** のファイルに以下が追加されます。

```
[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簡単な方法を提供します。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統合ポイントの役割を果たします。

## 4.4. PUPPET 設定への階層データの追加

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 key/value 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。

オーバークラウドを設定するには、Puppet はこの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、NFS 設定の全評価に適用されるように 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することができます。以下に例を示します。

```
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

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多くの場合に、YAML 形式で表現されます。テンプレートの目的は、リソースごとの設定や 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して作成することです。リソースとは、OpenStack のオブジェクトで、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、カスタムリソースが含まれる場合があります。

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 テンプレートの理解

Heat テンプレートの構造には主に 3 つのセクションが含まれます。

##### Parameters

Parameters は Heat に渡される設定のことで、値を指定せずにパラメーターのデフォルト値やスタックをカスタマイズする方法を提供します。これらは、テンプレートの **parameters** セクションで定義されます。

##### Resources

Resources と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のことです。OpenStack には全コンポーネントに対応するコアのリソースセットが含まれています。これらの設定は、テンプレートの **resources** セクションで定義されます。

##### Output

Output 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。これらは、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、**type: OS::Nova::Server** のリソース種別を使用して、特定のフレーバー、イメージ、キーを指定した **my\_instance** と呼ばれる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** の値を返します。

### 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. 環境ファイルの理解

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。

### Parameters

これらは、テンプレートのパラメーターに適用する共通設定で、環境ファイルの **parameters** セクションで定義します。

### Parameter Defaults

これらのパラメーターは、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。これらの設定は、環境ファイルの **parameter\_defaults** セクションで定義します。

### Resource Registry

このセクションでは、カスタムのリソース名と、他の Heat テンプレートへのリンクを定義します。これは実質的に、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方法を提供します。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。

以下に基本的な環境ファイルの例を示します。

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

parameter_defaults:
```

```
NetworkName: my_network

parameters:
  MyIP: 192.168.0.1
```

このファイルにより、**OS::Nova::Server::MyServer** と呼ばれる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。

## 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 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](https://github.com/openstack/tripleo-heat-templates) リポジトリの **Github** にある **openstack** グループから入手できます。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、以下のコマンドを実行します。

```
$ 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には、多数の Heat テンプレートおよび環境ファイルが含まれますが、注意すべき主要なファイルは以下の 3 つです。

### overcloud-without-mergepy.yaml

これはオーバークラウド環境を作成するために使用する主要なテンプレートファイルです。

### overcloud-resource-registry-puppet.yaml

これは、オーバークラウド環境の作成に使用する主要な環境ファイルで、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に Puppet の設定を開始します。

### overcloud-resource-registry.yaml

これは、オーバークラウド環境の作成に使用する標準の環境ファイルです。overcloud-resource-registry-puppet.yaml は、このファイルをベースにしています。このファイルは、お使いの環境をカスタム設定する際に使用します。

director は、最初の 2 つのファイルを使用して、オーバークラウドの作成を開始します。このコレクション内にある他のファイルはすべて **overcloud-resource-registry-puppet.yaml** ファイルと子関係にあるか、独自の環境ファイルに関連する追加の機能を提供して、デプロイメントに追加できるようにします。

### environments

オーバークラウドの作成に使用可能な Heat 環境ファイルが追加で含まれます。これらの環境ファイルは、作成された OpenStack 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。

### extraconfig

追加の機能を有効化するために使用するテンプレート。たとえば、director が提供する **extraconfig/pre\_deploy/rhel-registration** は、ノードの Red Hat Enterprise Linux オペレーティングシステムを Red Hat コンテンツ配信ネットワークまたは Red Hat Satellite サーバーに登録できるようにします。

### firstboot

ノードを最初に作成する際に director が使用する **first\_boot** スクリプトを提供します。

### network

分離ネットワークおよびポートを作成しやすくする Heat テンプレートセット

### puppet

大部分は Puppet を使用した設定によって動作するテンプレート。前述した **overcloud-resource-registry-puppet.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。

### validation-scripts

すべてのデプロイメント設定に有用な検証スクリプトが含まれます。

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。次の複数のセクションでは、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. 初回起動での設定のカスタマイズ

director は、オーバークラウドの初期設定時に全ノードに設定を行うメカニズムを提供し、**cloud-init** でこの設定をアーカイブします。アーカイブした内容は、**OS::TripleO::NodeUserData** リソース種別を使用して呼び出すことが可能です。

以下の例は、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することを目的とします。まず基本的な Heat テンプレート (**nameserver.yaml**) を作成します。このテンプレートは、固有のネームサーバーが指定された各ノードの **resolv.conf** を追加するスクリプトを実行します。**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します。

```
heat_template_version: 2014-10-16

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/resolve.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 (**firstboot.yaml**) を作成します。

```

resource_registry:
  OS::TripleO::NodeUserData: nameserver.yaml

```

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. オーバークラウドを構成する前の設定のカスタマイズ

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、初回のブートが完了してコア設定が開始する前に、カスタム設定を提供するリソースのセットを用意します。これには、以下のリソースが含まれます。

### 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 CephStorage ノードに適用される追加の設定

### OS::TripleO::NodeExtraConfig

Puppet のコア設定前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、変数のネームサーバーが指定された各ノードの `resolv.conf` を追加するスクリプトを実行します。

```
heat_template_version: 2014-10-16

parameters:
  server:
    type: json
  nameserver_ip:
    type: string

resources:
  ExtraPreConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolve.conf
        params:
          _NAMESERVER_IP_: {get_param: nameserver_ip}
  ExtraPreDeployment:
    type: OS::Heat::SoftwareDeployment
    properties:
      config: {get_resource: ExtraPreConfig}
      server: {get_param: server}
      actions: ['CREATE']

outputs:
  deploy_stdout:
    description: Deployment reference, used to trigger post-deploy on
    changes
    value: {get_attr: [ExtraPreDeployment, deploy_stdout]}
```

#### 重要

**servers** パラメーターは、設定を適用するサーバー一覧で、親テンプレートにより提供されます。このパラメーターは、すべての事前設定テンプレートで必須です。

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::Triple0::NodeExtraConfig: nameserver.yaml
parameter_defaults:
  nameserver_ip: 192.168.1.1
```

これにより、以下の操作が実行されます。

1. **OS::Triple0::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、**\*-puppet.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 **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** 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 **ExtraPreConfig** を定義します。**group: script** プロパティ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。このソフトウェア設定ツールは Heat のフックセットで入手できます。この場合は、**script** フックは、**SoftwareConfig** リソースで **config** プロパティ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。これにより、**template** セクションの変数を **params** セクションのパラメータに置き換えることが可能となります。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、その結果、スクリプトは以下ようになります。

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

5. **ExtraPreDeployments** は **ExtraPreConfig** 設定をノードにデプロイします。以下の点に注意してください。
  - ✧ **config** 属性は、Heat がどの設定が適用されるかを理解できるように **ExtraPreConfig** リソースを参照します。
  - ✧ **servers** 属性は、**overcloud-without-mergepy.yaml** が渡すオーバークラウドのノードのマッピングを取得します。
  - ✧ **actions** 属性は、設定を適用するタイミングを定義します。この場合は、オーバークラウドが作成された時にのみ設定を適用します。実行可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND** および **RESUME** です。

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. オーバークラウド設定後の設定のカスタマイズ

オーバークラウドの作成が完了してから、オーバークラウドの初回作成時または更新時に設定を追加する必要となる可能性があります。このような場合

は、**OS::TripleO::NodeExtraConfigPost** リソースを使用して、標準の

**OS::Heat::SoftwareConfig** 種別を使用した設定を適用します。これにより、メインのオーバークラウド設定が完了してから、追加の設定が適用されます。

以下の例では、まず基本的な Heat テンプレート (**nameserver.yaml**) を作成します。このテンプレートは、変数のネームサーバーが指定された各ノードの **resolv.conf** を追加するスクリプトを実行します。

```
heat_template_version: 2014-10-16

parameters:
  servers:
    type: json

  nameserver_ip:
    type: string

resources:

  ExtraConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      config:
        str_replace:
          template: |
            #!/bin/sh
            echo "nameserver _NAMESERVER_IP_" >> /etc/resolve.conf
        params:
          _NAMESERVER_IP_: {get_param: nameserver_ip}

  ExtraDeployments:
    type: OS::Heat::SoftwareDeployments
    properties:
      servers: {get_param: servers}
      config: {get_resource: ExtraConfig}
      actions: ['CREATE']
```

### 重要

**servers** パラメーターは、設定を適用するサーバー一覧で、親テンプレートにより提供されます (**overcloud-without-mergepy.yaml**)。このパラメーターは、すべての **OS::TripleO::NodeExtraConfigPost** テンプレートで必須です。

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 (**post\_config.yaml**) を作成します。

```
resource_registry:
  OS::Triple0::NodeExtraConfigPost: nameserver.yaml
parameter_defaults:
  nameserver_ip: 192.168.1.1
```

これにより、以下の操作が実行されます。

1. **OS::Triple0::NodeExtraConfigPost** 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、**\*-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 **ExtraConfig** を定義します。**group: script** プロパティ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。このソフトウェア設定ツールは Heat のフックセットで入手できます。この場合は、**script** フックは、**SoftwareConfig** リソースで **config** プロパティ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。これにより、**template** セクションの変数を **params** セクションのパラメーターに置き換えることが可能となります。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、その結果、スクリプトは以下ようになります。

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

5. **ExtraDeployments** は **ExtraConfig** 設定をノードにデプロイします。以下の点に注意してください。
  - ※ **config** 属性は、Heat がどの設定が適用されるかを理解できるように **ExtraConfig** リソースを参照します。
  - ※ **servers** 属性は、**overcloud-without-mergepy.yaml** が渡すオーバークラウドのノードのマッピングを取得します。
  - ※ **actions** 属性は、設定を適用するタイミングを定義します。この場合は、オーバークラウドが作成された時にのみ設定を適用します。実行可能なアクションは **CREATE**、**UPDATE**、**DELETE**、**SUSPEND** および **RESUME** です。

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。

## 5.6. カスタムの PUPPET 設定のオーバークラウドへの適用

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。このセクションでは、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した方法に似ていますが、**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** と **SoftwareDeployments** を使用するのに似ていますが、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\_factor** オプションは、**factor** 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。

## 5.7. オーバークラウドでの HIERA データの変更

前述したように、Puppet は Hiera ツールを使用して特定の変数にノード固有の値を指定します。これらのキーと値は通常、`/etc/puppet/hieradata` にあるファイルに保存されます。オーバークラウドでは、このディレクトリーには、カスタムパラメーターを追加する際に使用する、追加の Hiera ファイルのセットが含まれます。

director の Heat テンプレートコレクションにあるパラメーターセットを使用してこの Hiera データを渡します。これらのパラメーターは以下のとおりです。

### ExtraConfig

全ノードに追加する設定

### NovaComputeExtraConfig

コンピュートノードに追加する設定

### controllerExtraConfig

コントローラーノードに追加する設定

### BlockStorageExtraConfig

ブロックストレージノードに追加する設定

### ObjectStorageExtraConfig

オブジェクトストレージノードに追加する設定

### CephStorageExtraConfig

Ceph ストレージ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 に増やすには、以下のように設定します。

```
parameter_defaults:
  NovaComputeExtraConfig:
    nova::compute::reserved_host_memory: 1024
```

これにより、コンピュートノードの `/etc/puppet/hieradata` ディレクトリーにあるカスタムの Hiera ファイルに **nova::compute::reserved\_host\_memory: 1024** が追加されます。

## 5.8. 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。

```
$ 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るようにしてください。

### 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 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ました (『[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)。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。本項では、コンポーザブルサービスアーキテクチャーを考察し、カスタムサービスをコンポーザブルサービスアーキテクチャーに統合する方法の例を記載します。

### 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、**puppet/services** サブディレクトリー内のコンポーザブルサービステンプレートのコレクションが含まれます。これらのサービスは、以下のコマンドで表示することができます。

```
$ ls /usr/share/openstack-tripleo-heat-templates/puppet/services
```

各サービス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、**keystone.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: >
  OpenStack Identity (`keystone`) service configured with Puppet
```

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。たとえば、**keystone.yaml** サービステンプレートは **OS::TripleO::Services::Keystone** リソース種別に登録されます。

```
grep "OS::TripleO::Services::Keystone" /usr/share/openstack-tripleo-heat-templates/overcloud-resource-registry-puppet.j2.yaml
OS::TripleO::Services::Keystone: puppet/services/keystone.yaml
```

**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::Triple0::Services::CACerts
  - OS::Triple0::Services::CephMon
  - OS::Triple0::Services::CephExternal
  - OS::Triple0::Services::CephRgw
  - OS::Triple0::Services::CinderApi
  - OS::Triple0::Services::CinderBackup
  - OS::Triple0::Services::CinderScheduler
  - OS::Triple0::Services::CinderVolume
  - OS::Triple0::Services::Core
  - OS::Triple0::Services::Kernel
  - OS::Triple0::Services::Keystone
  - OS::Triple0::Services::GlanceApi
  - OS::Triple0::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。以下の例は、設定フックを使用するか、「[3章 オーバークラウドイメージ](#)」の手順に従って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ることを前提とします。

独自のサービスを作成する場合には、そのサービスの Heat テンプレートで定義すべき特定の項目があります。

### 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が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。この値は、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。この値は、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。この値は、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。

作成するサービスの必要に応じて、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、ホスト上でサービスの設定を定義します。

- ✳ **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を指定することができます。
- ✳ **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 サービスの設定
  - 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。

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の定義に使用する **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。その方法については、後半で説明します。

## 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

この例では、オーバークラウドのコントローラーノードでのみカスタムの **motd** サービスを設定することを目的としています。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。

まず最初に、**OS::Triple0::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::Triple0::Services::Motd** です。

```
resource_registry:
  OS::Triple0::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る点に注意してください。

デプロイメントに **motd** サービスが追加されましたが、この新規サービスを必要とする各ロールの **roles\_data.yaml** ファイルの **ServicesDefault** リストを更新する必要があります。以下の例では、コントローラーノードのみでこのサービスを設定します。

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
```

このファイルを編集して、**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
```

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 統合ポイント

本章では、director の統合における具体的な統合ポイントを考察し、固有の OpenStack コンポーネントと director またはオーバークラウドの統合とそのコンポーネントの関係について記載します。このセクションに記載する内容は、OpenStack のすべての統合についての包括的な説明ではありませんが、ハードウェアおよびソフトウェアを Red Hat OpenStack Platform に統合する作業を開始するには十分な情報となるはずです。

### 7.1. BARE METAL PROVISIONING (IRONIC)

OpenStack Bare Metal Provisioning (Ironic) のコンポーネントは、director 内で使用され、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。

Ironic 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 Red Hat OpenStack Platform 製品と director にデフォルトで自動的に含まれますが、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持続性に関する情報は、OpenStack コミュニティーページの [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:**

- ✎ 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、**Ironi**c 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、ドライバーに **Ironi**c が含まれる場合には、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。以下に例を示します。

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

これにより、**drivers** ディレクトリーからの指定のドライバーを **Ironi**c が使用できるようになります。

## 7.2. 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。多くのベンダーは、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することを目的にする必要があります。

**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。主な Neutron サーバー (およびプラグイン) 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。

プラグインおよびエージェントは、以下のいずれかが可能です。

- ※ OpenStack Platform 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。
- ※ OpenStack Platform 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できるように、既存のプラグインおよびエージェントの機能を分析することを推奨します。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:**

- ※ 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** リポジトリには、これらの統合の設定用に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。

## 7.3. BLOCK STORAGE (CINDER)

OpenStack Block Storage (Cinder) は、OpenStack 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ための API を提供します。たとえば、Cinder はインスタンスの仮想ストレージデバイスや、異なるストレージハードウェアや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。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、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。

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

- ✳ OpenStack Platform 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。
- ✳ OpenStack Platform 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できるように、既存のドライバーの機能を分析することを推奨します。

**アップストリームのリポジトリ:**

- ✳ 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** リポジトリでは **cinder/volume/drivers/** にドライバーが含まれます。

※ **puppet-cinder** リポジトリには、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。

## 7.4. IMAGE STORAGE (GLANCE)

OpenStack Image Storage (Glance) は、イメージのストレージを提供するストレージ種別と対話するための API を提供します。Glance は、ファイルのサポート、OpenStack Object Storage (Swift)、OpenStack Block Storage (Cinder)、Red Hat Ceph Storage など、異なる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。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:

※ OpenStack:

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

- [http://git.openstack.org/cgit/openstack/glance\\_store/](http://git.openstack.org/cgit/openstack/glance_store/)

※ GitHub:

- <https://github.com/openstack/glance/>

- [https://github.com/openstack/glance\\_store/](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:

- ✳ Glance は、イメージストレージを管理する統合ポイントを含む Cinder を使用できるため、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** リポジトリでは **glance\_store/\_drivers** にドライバーが含まれます。
- ✳ **puppet-glance** リポジトリでは **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** リポジトリには、Cinder 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。

## 7.5. 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:

- ✳ 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** リポジトリでは **manila/share/drivers/** にドライバーが含まれます。
- ※ **puppet-manila** リポジトリでは **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** リポジトリには、Cinder 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。

## 第8章 実例

本章では、Red Hat OpenStack Platform の一部としてベンダーのソリューションを統合する実例を取り上げて説明します。

### 8.1. CISCO NEXUS 1000V

Cisco Nexus 1000V は、仮想マシンアクセス用に設計されたネットワークスイッチです。また、VXLAN、ACL、IGMP スヌーピングを使用した高度なスイッチおよびセキュリティ機能を提供します。Cisco Nexus 1000V の ml2 ドライバーは [networking-cisco](#) リポジトリに含まれており、Neutron サービスと共にインストールできます。

オーバークラウドのイメージには Neutron Puppet モジュール (**puppet-neutron**) が含まれており、このモジュールに、Neutron が Cisco Nexus 1000V を使用するように設定するためのクラス (**neutron::plugins::ml2::cisco::nexus1000v**) が含まれます。このクラスは、モジュールの **manifests/plugins/ml2/cisco/nexus1000v.pp** マニフェストに配置されています。このクラスは、デフォルトのパラメーターセットを使用しますが、このデフォルトパラメーターを上書きして **neutron\_plugin\_ml2** ライブラリーを使用し、m2 プラグインが Cisco Nexus 1000V を使用するように設定することができます。

```
neutron_plugin_ml2 {
  'ml2/extension_drivers'           : value =>
$extension_drivers;
  'ml2_cisco_n1kv/n1kv_vsm_ips'     : value =>
$n1kv_vsm_ip;
  'ml2_cisco_n1kv/username'         : value =>
$n1kv_vsm_username;
  'ml2_cisco_n1kv/password'         : value =>
$n1kv_vsm_password;
  'ml2_cisco_n1kv/default_policy_profile' : value =>
$default_policy_profile;
  'ml2_cisco_n1kv/default_vlan_network_profile' : value =>
$default_vlan_network_profile;
  'ml2_cisco_n1kv/default_vxlan_network_profile' : value =>
$default_vxlan_network_profile;
  'ml2_cisco_n1kv/poll_duration'     : value =>
$poll_duration;
  'ml2_cisco_n1kv/http_pool_size'    : value =>
$http_pool_size;
  'ml2_cisco_n1kv/http_timeout'      : value =>
$http_timeout;
  'ml2_cisco_n1kv/sync_interval'      : value =>
$sync_interval;
  'ml2_cisco_n1kv/max_vsm_retries'    : value =>
$max_vsm_retries;
  'ml2_cisco_n1kv/restrict_policy_profiles' : value =>
$restrict_policy_profiles;
  'ml2_cisco_n1kv/enable_vif_type_n1kv' : value =>
$enable_vif_type_n1kv;
}
```

director の Heat テンプレートコレクションには、Cisco Nexus 1000V の Hiera データを設定するための環境ファイルと登録済みのテンプレートが含まれています。環境ファイルは、**environments/cisco-n1kv-config.yaml** に配置されており、以下のデフォルトの内容が含まれます。

```
resource_registry:
  OS::Triple0::ControllerExtraConfigPre:
    ../puppet/extraconfig/pre_deploy/controller/cisco-n1kv.yaml
  OS::Triple0::ComputeExtraConfigPre:
    ../puppet/extraconfig/pre_deploy/controller/cisco-n1kv.yaml

parameter_defaults:
  N1000vVSMIP: '192.0.2.50'
  N1000vMgmtGatewayIP: '192.0.2.1'
  N1000vVSMDomainID: '100'
  N1000vVSMHostMgmtIntf: 'br-ex'
```

**resource\_registry** は、コントローラーとコンピュートノード (**OS::Triple0::ControllerExtraConfigPre** および **OS::Triple0::ComputeExtraConfigPre**) の事前設定リソースが **puppet/extraconfig/pre\_deploy/controller/cisco-n1kv.yaml** を事前設定用のテンプレートとして使用するように設定します。**parameter\_defaults** のセクションには、これらのリソースに渡すパラメーターの一部が含まれます。

デプロイメントにこの環境ファイルを追加すると Hiera データが定義され、設定中に Puppet が Neutron Puppet モジュールのパラメーターにこのデータを使用します。

Puppet 設定を実際に適用するのは、自動で開始されます。Heat テンプレートコレクションには、コントローラーとコンピュートノードを設定するコアの Puppet マニフェストセットが含まれます。これらのファイルには、Cisco Nexus 1000V Hiera データが設定されていることを検出するためのロジックが含まれます。デプロイメントに **cisco-n1kv.yaml** を含めることで、マニフェストに **neutron::plugins::ml2::cisco::nexus1000v** クラスと Cisco Nexus 1000V の VEM と VSM エージェントが追加されます。

```
if 'cisco_n1kv' in hiera('neutron_mechanism_drivers') {
  include neutron::plugins::ml2::cisco::nexus1000v

  class { 'neutron::agents::n1kv_vem':
    n1kv_source      => hiera('n1kv_vem_source', undef),
    n1kv_version     => hiera('n1kv_vem_version', undef),
  }

  class { 'n1k_vsm':
    n1kv_source      => hiera('n1kv_vsm_source', undef),
    n1kv_version     => hiera('n1kv_vsm_version', undef),
  }
}
```

これは、オーバークラウドが Cisco Nexus 1000V のみを使用するように設定するには以下のステップが必要であるという意味です。

1. **environments/cisco-n1kv-config.yaml** ファイルを編集できるように、ローカルの場所にコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/cisco-n1kv-config.yaml ~/templates/.
```

## 2. **cisco-n1kv-config.yaml** ファイルを編集します。

- ✳ **cisco-n1kv.yaml** を参照する絶対パスを使用するように **resource\_registry** セクションを変更します。
- ✳ Cisco Nexus 1000V パラメーターを追加するように **parameter\_defaults** セクションを変更します。参考として **cisco-n1kv.yaml** を参照してください。

例:

```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /usr/share/openstack-tripleo-heat-templates/puppet/extraconfig/pre_deploy/controller/cisco-n1kv.yaml
  OS::TripleO::ComputeExtraConfigPre: /usr/share/openstack-tripleo-heat-templates/puppet/extraconfig/pre_deploy/controller/cisco-n1kv.yaml

parameter_defaults:
  N1000vVSMIP: '192.0.2.50'
  N1000vMgmtGatewayIP: '192.0.2.1'
  N1000vVSMDomainID: '100'
  N1000vVSMHostMgmtIntf: 'br-ex'
  N1000vVSMUser: admin
  N1000vVSMPassword: p@55w0rd!
```

## 3. デプロイメントに **cisco-n1kv-config.yaml** ファイルを追加します。

```
$ openstack overcloud deploy --templates -e ~/templates/cisco-n1kv-config.yaml
```

これは、オーバークラウドの Hiera データの一部として Cisco Nexus 1000V 設定を定義します。次に、オーバークラウドはこの Hieradata を使用して、コアの設定中に Neutron の Nexus 1000V ml2 ドライバーを設定します。

このセクションでは、director が認定済みのベンダーからのネットワークコンポーネントとオーバークラウドの Neutron サービスを統合する方法について、実例を挙げて説明しました。

## 8.2. NETAPP ストレージ

NetApp は、OpenStack ストレージのコンポーネントに統合するソリューションを複数提供します。以下の例では、ブロックストレージのバックエンドを提供するためにどのように NetApp Storage と Cinder を統合するかを示します。

Cinder のドライバーは、プロジェクト自体に含まれており、GitHub <https://github.com/openstack/cinder> で公開されています。NetApp のドライバーは、リポジトリの **cinder/volume/drivers/netapp/** ディレクトリーに配置されています。これは、ドライバーが Red Hat OpenStack Platform に自動で追加されることを意味します。

NetApp の設定は、オーバークラウドのイメージにも含まれる Cinder の Puppet モジュール (**puppet-cinder**) の中にあります。この設定が含まれる Puppet モジュールのマニフェストは **manifests/backend/netapp.pp** に配置されています。このマニフェストは **cinder\_config** ライブラリーを使用して、netapp の設定を Cinder 設定ファイルに追加します。

```
cinder_config {
  "${name}/nfs_mount_options":      value => $nfs_mount_options;
  "${name}/volume_backend_name":    value =>
$volume_backend_name;
  "${name}/volume_driver":          value =>
'cinder.volume.drivers.netapp.common.NetAppDriver';
  "${name}/netapp_login":           value => $netapp_login;
  "${name}/netapp_password":        value => $netapp_password,
secret => true;
  "${name}/netapp_server_hostname": value =>
$netapp_server_hostname;
  "${name}/netapp_server_port":     value =>
$netapp_server_port;
  "${name}/netapp_size_multiplier": value =>
$netapp_size_multiplier;
  "${name}/netapp_storage_family":  value =>
$netapp_storage_family;
  "${name}/netapp_storage_protocol": value =>
$netapp_storage_protocol;
  "${name}/netapp_transport_type":  value =>
$netapp_transport_type;
  "${name}/netapp_vfiler":          value => $netapp_vfiler;
  "${name}/netapp_volume_list":     value =>
$netapp_volume_list;
  "${name}/netapp_vserver":         value => $netapp_vserver;
  "${name}/netapp_partner_backend_name": value =>
$netapp_partner_backend_name;
  "${name}/expiry_thres_minutes":   value =>
$expiry_thres_minutes;
  "${name}/thres_avl_size_perc_start": value =>
$thres_avl_size_perc_start;
  "${name}/thres_avl_size_perc_stop": value =>
$thres_avl_size_perc_stop;
  "${name}/nfs_shares_config":      value => $nfs_shares_config;
  "${name}/netapp_copyoffload_tool_path": value =>
$netapp_copyoffload_tool_path;
  "${name}/netapp_controller_ips":  value =>
$netapp_controller_ips;
  "${name}/netapp_sa_password":     value =>
$netapp_sa_password, secret => true;
  "${name}/netapp_storage_pools":   value =>
$netapp_storage_pools;
  "${name}/netapp_eseries_host_type": value =>
$netapp_eseries_host_type;
  "${name}/netapp_webservice_path": value =>
$netapp_webservice_path;
}
```

director の Heat テンプレートコレクションには、NetApp Storage バックエンドの Hiera データを設定するための環境ファイルと登録済みのテンプレートが含まれています。環境ファイルは、**environments/cinder-netapp-config.yaml** に配置されており、以下のデフォルトの内容が含まれます。

```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre:
    ../puppet/extraconfig/pre_deploy/controller/cinder-netapp.yaml

parameter_defaults:
  CinderEnableNetappBackend: true
  CinderNetappBackendName: 'tripleo_netapp'
  CinderNetappLogin: ''
  CinderNetappPassword: ''
  CinderNetappServerHostname: ''
  CinderNetappServerPort: '80'
  CinderNetappSizeMultiplier: '1.2'
  CinderNetappStorageFamily: 'ontap_cluster'
  CinderNetappStorageProtocol: 'nfs'
  CinderNetappTransportType: 'http'
  CinderNetappVfiler: ''
  CinderNetappVolumeList: ''
  CinderNetappVserver: ''
  CinderNetappPartnerBackendName: ''
  CinderNetappNfsShares: ''
  CinderNetappNfsSharesConfig: '/etc/cinder/shares.conf'
  CinderNetappNfsMountOptions: ''
  CinderNetappCopyOffloadToolPath: ''
  CinderNetappControllerIps: ''
  CinderNetappSaPassword: ''
  CinderNetappStoragePools: ''
  CinderNetappEseriesHostType: 'linux_dm_mp'
  CinderNetappWebservicePath: '/devmgr/v2'
```

**resource\_registry** は、コントローラーノード (**OS::TripleO::ControllerExtraConfigPre**) の事前設定リソースが **puppet/extraconfig/pre\_deploy/controller/cinder-netapp.yaml** を事前設定用のテンプレートとして使用するように設定します。**parameter\_defaults** のセクションには、これらのリソースに渡すパラメーターの一部が含まれます。

デプロイメントにこの環境ファイルを追加すると Hiera データが定義され、設定中に Puppet が Cinder Puppet モジュールのパラメーターにこのデータを使用します。

**CinderEnableNetappBackend** パラメーターにより、Puppet 設定の適用が実際に開始されるかが決まります。Heat テンプレートコレクションには、コントローラーノードを設定するコアの Puppet マニフェストセットが含まれます。これらのファイルには、**cinder\_enable\_netapp\_backend** Hiera データが設定されているかどうかを検出するロジックが含まれます。Hiera データは、事前設定の **CinderEnableNetappBackend** パラメーターを使用して設定されます。デプロイメントに **cinder-netapp-config.yaml** を追加して **CinderEnableNetappBackend: true** をそのままにすると、コントローラーの Puppet マニフェストには **cinder::backend::netapp** クラスが追加され、環境ファイルからの Hiera データの値を渡します。

```
if hiera('cinder_enable_netapp_backend', false) {
  $cinder_netapp_backend = hiera('cinder::backend::netapp::title')
```



```

cinder_config {
    "${cinder_netapp_backend}/host": value => 'hostgroup';
}

if hiera('cinder::backend::netapp::nfs_shares', undef) {
    $cinder_netapp_nfs_shares =
split(hiera('cinder::backend::netapp::nfs_shares', undef), ',')
}

cinder::backend::netapp { $cinder_netapp_backend :
    netapp_login =>
hiera('cinder::backend::netapp::netapp_login', undef),
    netapp_password =>
hiera('cinder::backend::netapp::netapp_password', undef),
    netapp_server_hostname =>
hiera('cinder::backend::netapp::netapp_server_hostname', undef),
    netapp_server_port =>
hiera('cinder::backend::netapp::netapp_server_port', undef),
    netapp_size_multiplier =>
hiera('cinder::backend::netapp::netapp_size_multiplier', undef),
    netapp_storage_family =>
hiera('cinder::backend::netapp::netapp_storage_family', undef),
    netapp_storage_protocol =>
hiera('cinder::backend::netapp::netapp_storage_protocol', undef),
    netapp_transport_type =>
hiera('cinder::backend::netapp::netapp_transport_type', undef),
    netapp_vfiler =>
hiera('cinder::backend::netapp::netapp_vfiler', undef),
    netapp_volume_list =>
hiera('cinder::backend::netapp::netapp_volume_list', undef),
    netapp_vserver =>
hiera('cinder::backend::netapp::netapp_vserver', undef),
    netapp_partner_backend_name =>
hiera('cinder::backend::netapp::netapp_partner_backend_name', undef),
    nfs_shares => $cinder_netapp_nfs_shares,
    nfs_shares_config =>
hiera('cinder::backend::netapp::nfs_shares_config', undef),
    netapp_copyoffload_tool_path =>
hiera('cinder::backend::netapp::netapp_copyoffload_tool_path', undef),
    netapp_controller_ips =>
hiera('cinder::backend::netapp::netapp_controller_ips', undef),
    netapp_sa_password =>
hiera('cinder::backend::netapp::netapp_sa_password', undef),
    netapp_storage_pools =>
hiera('cinder::backend::netapp::netapp_storage_pools', undef),
    netapp_eseries_host_type =>
hiera('cinder::backend::netapp::netapp_eseries_host_type', undef),
    netapp_webservice_path =>
hiera('cinder::backend::netapp::netapp_webservice_path', undef),
}
}

```

これは、オーバークラウドが NetApp Storage のみを使用するには以下のステップが必要であるという意味です。

1. **environments/cinder-netapp-config.yaml** ファイルを編集できるように、ローカルの場所にコピーします。

```
$ cp /usr/share/openstack-tripleo-heat-templates/environments/cinder-netapp-config.yaml ~/templates/.
```

2. **cinder-netapp-config.yaml** ファイルを編集します。

- ✱ **cinder-netapp.yaml** を参照する絶対パスを使用するように **resource\_registry** セクションを変更します。
- ✱ NetApp パラメーターを追加するように **parameter\_defaults** セクションを変更します。参考として **cinder-netapp.yaml** を参照してください。

例:

```
resource_registry:
  OS::TripleO::ControllerExtraConfigPre: /usr/share/openstack-tripleo-heat-templates/puppet/extraconfig/pre_deploy/controller/cinder-netapp.yaml

parameter_defaults:
  CinderEnableNetappBackend: true
  CinderNetappBackendName: 'tripleo_netapp'
  CinderNetappLogin: 'admin'
  CinderNetappPassword: 'p@55w0rd!'
  CinderNetappServerHostname: 'netapp.example.com'
  CinderNetappServerPort: '80'
  CinderNetappSizeMultiplier: '1.2'
  CinderNetappStorageFamily: 'ontap_cluster'
  CinderNetappStorageProtocol: 'nfs'
  CinderNetappTransportType: 'http'
  CinderNetappNfsShares:
    '192.168.1.200:/storage1,192.168.1.200:/storage2'
  CinderNetappNfsSharesConfig: '/etc/cinder/shares.conf'
  CinderNetappEseriesHostType: 'linux_dm_mp'
  CinderNetappWebservicePath: '/devmgr/v2'
```

**CinderEnableNetappBackend** は **true** 設定のままにしておいてください。

3. デプロイメントに **cinder-netapp-config.yaml** ファイルを追加します。

```
$ openstack overcloud deploy --templates -e ~/templates/cinder-netapp-config.yaml
```

これは、オーバークラウドの Hiera データの一部として NetApp Storage 設定を定義します。次に、オーバークラウドはこの Hieradata を使用して、コアの設定中に Cinder の NetApp バックエンドを設定します。

このセクションでは、director が認定済みのベンダーからのストレージコンポーネントとオーバークラウドの Cinder サービスを統合する方法について、実例を挙げて説明しました。