



Red Hat OpenStack Platform 11

オーバークラウド向けの Red Hat Ceph Storage

オーバークラウドで Red Hat Ceph Storage を使用するための設定

Red Hat OpenStack Platform 11 オーバークラウド向けの Red Hat Ceph Storage

オーバークラウドで Red Hat Ceph Storage を使用するための設定

OpenStack Team

rhos-docs@redhat.com

法律上の通知

Copyright © 2018 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

本ガイドでは、Red Hat Ceph Storage の環境の推奨事項や Ceph Storage ノードでオーバークラウドを実装する方法など、Red Hat OpenStack Platform director を使用して、Red Hat Ceph Storage を使用するオーバークラウドを作成する方法を説明します。

目次

第1章 はじめに	3
1.1. CEPH STORAGE の定義	3
1.2. RED HAT OPENSTACK PLATFORM での RED HAT CEPH STORAGE の使用	3
1.3. 設定要件	3
1.4. シナリオの定義	5
第2章 CEPH STORAGE ノードでのオーバークラウドの作成	6
2.1. STACK ユーザーの初期化	6
2.2. ノードの登録	7
2.3. ノードの手動でのタグ付け	9
2.4. 専用ノード上でのその他の CEPH サービスのデプロイ	10
2.4.1. Ceph MON サービス向けのカスタムロールとフレーバーの作成	11
2.4.2. Ceph MDS サービス向けのカスタムロールとフレーバーの作成	12
2.5. CEPH STORAGE ノードのルートディスクの定義	14
2.6. オーバークラウドでの CEPH STORAGE の有効化	16
2.7. CEPH STORAGE ノードのディスクレイアウトのマッピング	17
2.8. CEPH STORAGE ノードのディスクのクリーニング	18
2.9. CEPH OBJECT GATEWAY のデプロイ	19
2.10. バックアップサービスで CEPH を使用する設定	19
2.11. CEPH ノードごとに複数のボンディングされたインターフェースを設定する方法	20
2.11.1. ボンディングモジュールのディレクティブの設定	23
2.12. CEPH STORAGE CLUSTER のカスタマイズ	24
2.12.1. 異なる Ceph プールへのカスタムの属性の割り当て	25
2.13. ロールへのノードとフレーバーの割り当て	25
2.14. オーバークラウドの作成	26
2.15. オーバークラウドへのアクセス	27
2.16. CEPH STORAGE ノードの監視	28
2.17. 環境のリブート	28
2.18. CEPH クラスターのスケールアップ	29
2.19. CEPH STORAGE ノードのスケールダウンと置き換え	31
2.20. CEPH STORAGE ノードへの OSD ディスクの追加と削除	34
第3章 既存の CEPH STORAGE CLUSTER のオーバークラウドとの統合	35
3.1. 既存の CEPH STORAGE CLUSTER の設定	35
3.2. STACK ユーザーの初期化	37
3.3. ノードの登録	37
3.4. ノードの手動でのタグ付け	39
3.5. 既存の CEPH STORAGE クラスターとの統合	40
3.6. 以前の RED HAT CEPH STORAGE バージョンとの後方互換性	40
3.7. ロールへのノードとフレーバーの割り当て	41
3.8. オーバークラウドの作成	42
3.9. オーバークラウドへのアクセス	42
第4章 まとめ	44
付録A 環境ファイルのサンプル: CEPH クラスターの作成	45
付録B カスタムインターフェーステンプレートの例: 複数のボンディングされたインターフェース	47

第1章 はじめに

Red Hat OpenStack Platform director は、オーバークラウドと呼ばれるクラウド環境を作成します。director を使用して、オーバークラウドの追加機能を設定することができます。これらの追加機能の1つに、director で作成した Ceph Storage Cluster や、既存の Ceph Storage Cluster などの Red Hat Ceph Storage との統合が含まれます。本ガイドには、director を使用してオーバークラウドに Ceph Storage を統合する方法についての説明と設定例を記載します。

1.1. CEPH STORAGE の定義

Red Hat Ceph Storage は、優れたパフォーマンス、信頼性、スケーリングを提供するように設計された、分散型のデータオブジェクトストレージです。非構造化データに対応しており、クライアントが新しいタイプのオブジェクトインターフェースと従来のインターフェースを同時に使用できる分散型のオブジェクトストアは、今後のストレージのあるべき姿です。Ceph デプロイメントはすべて、2 種類のデーモンで構成される **Ceph Storage Cluster** を中心とします。

Ceph OSD (Object Storage Daemon)

Ceph OSD は、Ceph クライアントの代わりにデータを格納します。また、Ceph ノードの CPU とメモリーを使用して、データの複製、リバランス、復旧、監視、レポート作成を実行します。

Ceph モニター

Ceph モニターは、ストレージクラスターの現在のステータスを含む Ceph ストレージクラスターのマッピングのマスターコピーを管理します。

Red Hat Ceph Storage に関する詳しい情報は、『[Red Hat Ceph Storage Architecture Guide](#)』を参照してください。



重要

本ガイドでは、Ceph Block Storage および Ceph Object Gateway (RGW) のみを統合します。Ceph File (CephFS) ストレージの統合については記載していません。

1.2. RED HAT OPENSTACK PLATFORM での RED HAT CEPH STORAGE の使用

Red Hat OpenStack Platform director で Red Hat Ceph Storage をオーバークラウドに統合するには、主に 2 つの方法があります。

Ceph Storage Cluster でのオーバークラウドの作成

director には、オーバークラウドの作成中に Ceph Storage Cluster を作成する機能があります。director は、データの格納に Ceph OSD を使用する Ceph Storage ノードセットを作成します。さらに、director は、オーバークラウドのコントローラーノードに Ceph Monitor サービスをインストールします。このため、組織が高可用性のコントローラーノード 3 台で構成されるオーバークラウドを作成する場合には、Ceph Monitor も高可用性サービスになります。

既存の Ceph Storage のオーバークラウドへの統合

既存の Ceph Storage Cluster がある場合には、オーバークラウドのデプロイメント時に統合できます。これは、オーバークラウドの設定以外のクラスターの管理やスケーリングが可能であることを意味します。

1.3. 設定要件

本ガイドは、『[director のインストールと使用方法](#)』ガイドの補足情報を提供します。これは、「要件」で指定されているのと同じ要件が本ガイドにも適用されることを意味します。必要に応じて、この

要件を実装してください。

Red Hat OpenStack Platform director を使用して Ceph Storage ノードを作成する場合は、それらのノードに対する以下の要件に注意してください。

プロセッサ

Intel 64 または AMD64 CPU 拡張機能のサポートがある 64 ビットの x86 プロセッサ

メモリー

メモリー要件はストレージ容量によって異なります。ハードディスク容量 1 TB あたり最小で 1 GB のメモリーを使用するのが理想的です。

ディスク領域

ストレージ要件はメモリーの容量によって異なります。ハードディスク容量 1 TB あたり最小で 1 GB のメモリーを使用するのが理想的です。

ディスクのレイアウト

推奨される Red Hat Ceph Storage ノードの設定には、以下のようなディスクレイアウトが必要です。

- **/dev/sda:** ルートディスク。director は、主なオーバークラウドイメージをディスクにコピーします。
- **/dev/sdb:** ジャーナルディスク。このディスクは、/dev/sdb1、/dev/sdb2、/dev/sdb3 などのように、Ceph OSD ジャーナル向けにパーティションを分割します。ジャーナルディスクは通常、システムパフォーマンス向上に役立つ Solid State Drive (SSD) です。
- **/dev/sdc 以降:** OSD ディスク。ストレージ要件で必要な数のディスクを使用します。



重要

オーバークラウドのデプロイ前に、ジャーナルおよび OSD の対象となるディスクで既存のパーティションをすべて消去します。さらに、Ceph Storage OSD およびジャーナルディスクは、デプロイメントの一部として設定可能な GPT ディスクラベルが必要です。詳しい情報は「[Ceph Storage ノードのディスクのクリーニング](#)」を参照してください。

ネットワークインターフェースカード

最小で 1 x 1 Gbps ネットワークインターフェースカード (実稼動環境では、最低でも NIC を 2 つ以上使用することを推奨します)。ボンディングされたインターフェース向けの場合や、タグ付けされた VLAN トラフィックを委譲する場合には、追加のネットワークインターフェースを使用します。特に大量のトラフィックにサービスを提供する OpenStack Platform 環境を構築する場合には、ストレージノードには 10 Gbps インターフェースを使用することを推奨します。

Intelligent Platform Management Interface (IPMI)

各 Ceph ノードには、サーバーのマザーボード上に IPMI 機能が必要です。

本ガイドでは、以下の要件も満たす必要があります。

- Red Hat OpenStack Platform director でインストールしたアンダークラウドホスト。「[アンダークラウドのインストール](#)」を参照してください。
- Red Hat Ceph Storage のハードウェアの追加の推奨事項。これらの推奨事項については『[Red Hat Ceph Storage Hardware Guide](#)』を参照してください。



重要

Ceph Monitor サービスは、オーバークラウドのコントローラーノードにインストールされます。これは、パフォーマンスの問題を軽減するために、適切なリソースを提供する必要があるという意味です。お使いの環境のコントローラーノードでは、最低でもメモリー 16 GB を、Ceph Monitor データにはソリッドステートドライブ (SSD) を使用するようになっています。

1.4. シナリオの定義

本ガイドでは、2 つのシナリオを使用します。

- 最初のシナリオでは、**Ceph Storage Cluster** でオーバークラウドを作成します。これは、**director** が **Ceph Storage Cluster** をデプロイすることを意味します。
- 2 番目のシナリオでは、既存の **Ceph Storage Cluster** とオーバークラウドを統合します。これは、オーバークラウドの管理と **Ceph Storage Cluster** の管理を分けることを意味します。

第2章 CEPH STORAGE ノードでのオーバークラウドの作成

本章では、**director** を使用して、独自の **Ceph Storage Cluster** が含まれるオーバークラウドを作成する方法を説明します。オーバークラウドの作成方法や、既存の **Ceph Storage Cluster** との統合方法に関する説明は、「[3章 既存の Ceph Storage Cluster のオーバークラウドとの統合](#)」を参照してください。

本章のシナリオでは、オーバークラウドは 9 台のノードで構成されます。

- 高可用性のコントローラーノード 3 台。各ノードに **Ceph Monitor** サービスが含まれます。
- クラスター内に **Red Hat Ceph Storage** ノード 3 台。これらのノードには、**Ceph OSD** が含まれ、実際のストレージとしての役割を果たします。
- コンピュートノード 3 台

このシナリオのすべてのマシンは、電源管理に **IPMI** を使用したベアメタルマシンです。**director** により **Red Hat Enterprise Linux 7** のイメージが各ノードにコピーされるため、これらのノードではオペレーティングシステムは必要ありません。

director は、イントロスペクションおよびプロビジョニングプロセス中に、プロビジョニングネットワークを使用して各ノードと通信します。すべてのノードは、ネイティブの **VLAN** 経由でネットワークに接続します。この例では、以下の IP アドレスの割り当てで、プロビジョニングサブネットとして **192.0.2.0/24** を使用します。

ノード名	IP アドレス	MAC アドレス	IPMI IP アドレス
director	192.0.2.1	aa:aa:aa:aa:aa:aa	
コントローラー 1	定義済みの DHCP	b1:b1:b1:b1:b1:b1	192.0.2.205
コントローラー 2	定義済みの DHCP	b2:b2:b2:b2:b2:b2	192.0.2.206
コントローラー 3	定義済みの DHCP	b3:b3:b3:b3:b3:b3	192.0.2.207
コンピュート 1	定義済みの DHCP	c1:c1:c1:c1:c1:c1	192.0.2.208
コンピュート 2	定義済みの DHCP	c2:c2:c2:c2:c2:c2	192.0.2.209
コンピュート 3	定義済みの DHCP	c3:c3:c3:c3:c3:c3	192.0.2.210
Ceph 1	定義済みの DHCP	d1:d1:d1:d1:d1:d1	192.0.2.211
Ceph 2	定義済みの DHCP	d2:d2:d2:d2:d2:d2	192.0.2.212
Ceph 3	定義済みの DHCP	d3:d3:d3:d3:d3:d3	192.0.2.213

2.1. STACK ユーザーの初期化

stack ユーザーとして **director** ホストにログインし、以下のコマンドを実行して **director** の設定を初期化します。

■

```
$ source ~/stackrc
```

このコマンドでは、**director** の CLI ツールにアクセスする認証情報が含まれる環境変数を設定します。

2.2. ノードの登録

ノード定義のテンプレート (**instackenv.json**) は JSON ファイル形式で、ノード登録用のハードウェアおよび電源管理の情報が含まれています。以下に例を示します。

```
{
  "nodes": [
    {
      "mac": [
        "b1:b1:b1:b1:b1:b1"
      ],
      "cpu": "4",
      "memory": "6144",
      "disk": "40",
      "arch": "x86_64",
      "pm_type": "pxe_ipmitool",
      "pm_user": "admin",
      "pm_password": "p@55w0rd!",
      "pm_addr": "192.0.2.205"
    },
    {
      "mac": [
        "b2:b2:b2:b2:b2:b2"
      ],
      "cpu": "4",
      "memory": "6144",
      "disk": "40",
      "arch": "x86_64",
      "pm_type": "pxe_ipmitool",
      "pm_user": "admin",
      "pm_password": "p@55w0rd!",
      "pm_addr": "192.0.2.206"
    },
    {
      "mac": [
        "b3:b3:b3:b3:b3:b3"
      ],
      "cpu": "4",
      "memory": "6144",
      "disk": "40",
      "arch": "x86_64",
      "pm_type": "pxe_ipmitool",
      "pm_user": "admin",
      "pm_password": "p@55w0rd!",
      "pm_addr": "192.0.2.207"
    },
    {
      "mac": [
        "c1:c1:c1:c1:c1:c1"
      ],
      "cpu": "4",
```

```

        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.208"
    },
    {
        "mac": [
            "c2:c2:c2:c2:c2:c2"
        ],
        "cpu": "4",
        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.209"
    },
    {
        "mac": [
            "c3:c3:c3:c3:c3:c3"
        ],
        "cpu": "4",
        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.210"
    },
    {
        "mac": [
            "d1:d1:d1:d1:d1:d1"
        ],
        "cpu": "4",
        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.211"
    },
    {
        "mac": [
            "d2:d2:d2:d2:d2:d2"
        ],
        "cpu": "4",
        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",

```

```

        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.212"
    },
    {
        "mac": [
            "d3:d3:d3:d3:d3:d3"
        ],
        "cpu": "4",
        "memory": "6144",
        "disk": "40",
        "arch": "x86_64",
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "p@55w0rd!",
        "pm_addr": "192.0.2.213"
    }
]
}

```

テンプレートの作成後に、**stack** ユーザーのホームディレクトリー (**/home/stack/instackenv.json**) にファイルを保存してから、**director** にインポートします。これには、以下のコマンドを実行します。

```
$ openstack baremetal import --json ~/instackenv.json
```

このコマンドでテンプレートをインポートして、テンプレートから **director** に各ノードを登録します。カーネルと **ramdisk** イメージを全ノードに割り当てます。

```
$ openstack baremetal configure boot
```

director でのノードの登録、設定が完了しました。

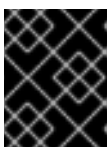
2.3. ノードの手動でのタグ付け

各ノードを登録した後は、ハードウェアを検査して、ノードを特定のプロファイルにタグ付けする必要があります。プロファイルタグにより、ノードがフレーバーに照合され、そのフレーバーはデプロイメントロールに割り当てられます。

新規ノードを検査してタグ付けするには、以下のステップに従います。

1. ハードウェアのイントロスペクションをトリガーして、各ノードのハードウェア属性を取得します。

```
$ openstack baremetal introspection bulk start
```



重要

このプロセスが最後まで実行されて正常に終了したことを確認してください。ベアメタルの場合には、通常 15 分ほどかかります。

2. ノード一覧を取得して **UUID** を識別します。

■

```
$ ironic node-list
```

- 各ノードの **properties/capabilities** パラメーターに **profile** オプションを追加して、ノードを特定のプロファイルに手動でタグ付けします。
たとえば、標準的なデプロイメントは、**control**、**compute**、および **ceph-storage** の 3 つのプロファイルを使用します。以下のコマンドは、3 つのノードを各プロファイルにタグ付けします。

```
$ ironic node-update 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 6faba1a9-e2d8-4b7c-95a2-c7fbdc12129a add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 5e3b2f50-fcd9-4404-b0a2-59d79924b38e add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 484587b2-b3b3-40d5-925b-a26a2fa3036f add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d010460b-38f2-4800-9cc4-d69f0d067efe add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d930e613-3e14-44b9-8240-4f3559801ea6 add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update da0cc61b-4882-45e0-9f43-fab65cf4e52b add
properties/capabilities='profile:ceph-storage,boot_option:local'
$ ironic node-update b9f70722-e124-4650-a9b1-aade8121b5ed add
properties/capabilities='profile:ceph-storage,boot_option:local'
$ ironic node-update 68bf8f29-7731-4148-ba16-efb31ab8d34f add
properties/capabilities='profile:ceph-storage,boot_option:local'
```

ヒント

Ceph MON サービスおよび Ceph MDS サービス用のノードをタグ付けすることのできる新しいカスタムプロファイルを設定することも可能です。詳しくは、「[専用ノード上でのその他の Ceph サービスのデプロイ](#)」を参照してください。

profile オプションを追加すると、適切なプロファイルにノードをタグ付けします。

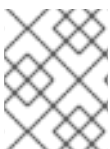


注記

手動でのタグ付けの代わりに、**Automated Health Check (AHC)** ツールを使用し、ベンチマークデータに基づいて、多数のノードに自動でタグ付けします。

2.4. 専用ノード上でのその他の CEPH サービスのデプロイ

デフォルトでは、**director** は **Ceph MON** サービスおよび **Ceph MDS** サービスをコントローラーノードにデプロイします。これは、小型のデプロイメントには適していますが、大型のデプロイメントの場合には、**Ceph** クラスターのパフォーマンスを向上させるために、**Ceph MON** サービスおよび **Ceph MDS** サービスを専用のノードにデプロイすることを推奨します。これは、いずれかのサービス用の **カスタムロール** を作成することによって行うことができます。



注記

カスタムロールについての詳しい情報は、『[オーバークラウドの高度なカスタマイズ](#)』ガイドの「[新規ロールの作成](#)」を参照してください。

director は、全オーバークラウドロールのデフォルトのリファレンスとして以下のファイルを使用します。

/usr/share/openstack-tripleo-heat-templates/roles_data.yaml

このファイルを **/home/stack/templates/** にコピーして、カスタムロールを追加できるようにします。

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

カスタムロールファイル **/home/stack/templates/roles_data_custom.yaml** は後ほどオーバークラウド作成中に呼び出されます(「[オーバークラウドの作成](#)」)。以下のサブセクションでは、**Ceph MON** サービスまたは **Ceph MDS** サービス向けのカスタムロールの設定方法について説明します。

2.4.1. Ceph MON サービス向けのカスタムロールとフレーバーの作成

本項では、**Ceph MON** ロール向けにカスタムロール(名前: **CephMon**) およびフレーバー(名前: **ceph-mon**)を作成する方法について説明します。**director** が全デフォルトロール(「[専用ノード上でのその他の Ceph サービスのデプロイ](#)」に記載)の参照用に使用するファイルはすでにコピー済みのはずです。

1. **/home/stack/templates/roles_data_custom.yaml** ファイルを開きます。
2. **Ceph MON** サービス (**OS::TripleO::Services::CephMon**) をコントローラーノードの下から削除します。

```
[...]
- name: Controller # the 'primary' role goes first
  CountDefault: 1
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMds
  # - OS::TripleO::Services::CephMon // ❶
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephRbdMirror
    - OS::TripleO::Services::CephRgw
    - OS::TripleO::Services::CinderApi
[...]
```

- ❶ この行をコメントアウトします。次のステップでは、これと同じサービスがカスタムロールに追加されます。

3. **roles_data_custom.yaml** の末尾に、**Ceph MON** サービスおよびその他すべての必要なノードサービスを含むカスタムの **CephMon** ロールを追加します。以下に例を示します。

```
- name: CephMon
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephMon
    - OS::TripleO::Services::Kernel
    - OS::TripleO::Services::Ntp
    - OS::TripleO::Services::Timezone
    - OS::TripleO::Services::Snmp
    - OS::TripleO::Services::Sshd
```

```
- OS::TripleO::Services::TripleoPackages
- OS::TripleO::Services::TripleoFirewall
- OS::TripleO::Services::SensuClient
- OS::TripleO::Services::FluentdClient
- OS::TripleO::Services::AuditD
- OS::TripleO::Services::Collectd
```

4. **openstack flavor create** コマンドを使用して、このロール用に **ceph-mon** という名前の新規フレーバーを定義します。

```
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4
ceph-mon
```



注記

このコマンドについての詳しい情報を確認するには、**openstack flavor create --help** を実行してください。

5. このフレーバーを新規プロファイルにマッピングします。このプロファイルも **ceph-mon** という名前です。

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="ceph-mon" ceph-mon
```



注記

このコマンドについての詳しい情報は、**openstack flavor set --help** で確認してください。

これで、ノードを **ceph-mon** プロファイルにタグ付けできるようになりました。そのためには、以下のコマンドを実行します。

```
$ ironic node-update UUID add properties/capabilities='profile:ceph-
mon,boot_option:local'
```

ノードのタグ付けに関する詳しい情報は、「[ノードの手動でのタグ付け](#)」を参照してください。また、カスタムロールプロファイルの関連情報は、「[プロファイルへのノードのタグ付け](#)」も参照してください。

2.4.2. Ceph MDS サービス向けのカスタムロールとフレーバーの作成

本項では、Ceph MDS ロール向けにカスタムロール (名前: **CephMDS**) およびフレーバー (名前: **ceph-mds**) を作成する方法について説明します。**director** が全デフォルトロール ([「専用ノード上でのその他の Ceph サービスのデプロイ」](#)に記載) の参照用に使用するファイルはすでにコピー済みのはずです。

1. **/home/stack/templates/roles_data_custom.yaml** ファイルを開きます。
2. Ceph MDS サービス (**OS::TripleO::Services::CephMds**) をコントローラーノードの下から削除します。

```
[...]
```



```

- name: Controller # the 'primary' role goes first
  CountDefault: 1
  ServicesDefault:
    - OS::Triple0::Services::CACerts
# - OS::Triple0::Services::CephMds // ❶
    - OS::Triple0::Services::CephMon
    - OS::Triple0::Services::CephExternal
    - OS::Triple0::Services::CephRbdMirror
    - OS::Triple0::Services::CephRgw
    - OS::Triple0::Services::CinderApi
[...]
```

- ❶ この行をコメントアウトします。次のステップでは、これと同じサービスがカスタムロールに追加されます。

3. **roles_data_custom.yaml** の末尾に、Ceph MDS サービスおよびその他すべての必要なノードサービスを含むカスタムの **CephMDS** ロールを追加します。以下に例を示します。

```

- name: CephMDS
  ServicesDefault:
    - OS::Triple0::Services::CACerts
    - OS::Triple0::Services::CephMds
    - OS::Triple0::Services::CephClient // ❶
    - OS::Triple0::Services::Kernel
    - OS::Triple0::Services::Ntp
    - OS::Triple0::Services::Timezone
    - OS::Triple0::Services::Snmp
    - OS::Triple0::Services::Sshd
    - OS::Triple0::Services::TripleoPackages
    - OS::Triple0::Services::TripleoFirewall
    - OS::Triple0::Services::SensuClient
    - OS::Triple0::Services::FluentdClient
    - OS::Triple0::Services::AuditD
    - OS::Triple0::Services::Collectd
```

- ❶ Ceph MDS サービスには、Ceph MON サービスまたは Ceph Client サービスによって設定することのできる管理キーリングが必要です。専用ノードに (Ceph MON サービスなしで) Ceph MDS をデプロイするため、ロールには Ceph クライアントサービスも追加してください。

4. **openstack flavor create** コマンドを使用して、このロール用に **ceph-mds** という名前の新規フレーバーを定義します。

```
$ openstack flavor create --id auto --ram 6144 --disk 40 --vcpus 4
ceph-mds
```



注記

このコマンドについての詳しい情報を確認するには、**openstack flavor create --help** を実行してください。

5. このフレーバーを新規プロファイルにマッピングします。このプロファイルも、**ceph-mds** という名前です。

```
$ openstack flavor set --property "cpu_arch"="x86_64" --property
"capabilities:boot_option"="local" --property
"capabilities:profile"="ceph-mds" ceph-mds
```



注記

このコマンドについての詳しい情報は、**openstack flavor set --help** で確認してください。

これで、ノードを **ceph-mds** プロファイルにタグ付けできるようになりました。そのためには、以下のコマンドを実行します。

```
$ ironic node-update UUID add properties/capabilities='profile:ceph-
mds,boot_option:local'
```

ノードのタグ付けに関する詳しい情報は、「[ノードの手動でのタグ付け](#)」を参照してください。また、カスタムロールプロファイルの関連情報は、「[プロファイルへのノードのタグ付け](#)」も参照してください。

2.5. CEPH STORAGE ノードのルートディスクの定義

大半の Ceph Storage ノードは、複数のディスクを使用します。このため、**director** は、Ceph Storage ノードのプロビジョニング時にルートディスクを特定する必要があります。以下のプロパティーを使用すると、ルートディスクの特定に役立ちます。

- **model** (文字列): デバイスの ID
- **vendor** (文字列): デバイスのベンダー
- **serial** (文字列): ディスクのシリアル番号
- **wwn** (文字列): 一意のストレージ ID
- **size** (整数): デバイスのサイズ (GB)

以下の例では、ルートデバイスを特定するディスクのシリアル番号を使用して、オーバークラウドイメージをデプロイするドライブを指定します。

最初に、**director** がイントロスペクションで取得した各ノードのハードウェア情報のコピーを収集します。この情報は、**OpenStack Object Storage (swift)** サーバーに保管されています。この情報を新規ディレクトリーにダウンロードします。

```
$ mkdir swift-data
$ cd swift-data
$ export SWIFT_PASSWORD=`sudo crudini --get /etc/ironic-
inspector/inspector.conf swift password`
$ for node in $(ironic node-list | grep -v UUID | awk '{print $2}'); do
swift -U service:ironic -K $SWIFT_PASSWORD download ironic-inspector
inspector_data-$node; done
```



注記

この例では **crudini** パッケージで入手可能な **crudini** コマンドを使用します。

この操作により、イントロスペクションで取得した各 **inspector_data** オブジェクトからデータがダウンロードされます。全オブジェクトのオブジェクト名の一部には、ノードの **UUID** が使用されます。

```
$ ls -l
inspector_data-15fc0edc-eb8d-4c7f-8dc0-a2a25d5e09e3
inspector_data-46b90a4d-769b-4b26-bb93-50eaefcdb3f4
inspector_data-662376ed-faa8-409c-b8ef-212f9754c9c7
inspector_data-6fc70fe4-92ea-457b-9713-eed499eda206
inspector_data-9238a73a-ec8b-4976-9409-3fcff9a8dca3
inspector_data-9cbfe693-8d55-47c2-a9d5-10e059a14e07
inspector_data-ad31b32d-e607-4495-815c-2b55ee04cdb1
inspector_data-d376f613-bc3e-4c4b-ad21-847c4ec850f8
```

各ノードのディスク情報をチェックします。以下のコマンドを実行すると、各ノードの ID とディスク情報が表示されます。

```
$ for node in $(ironic node-list | grep -v UUID | awk '{print $2}'); do
echo "NODE: $node" ; cat inspector_data-$node | jq '.inventory.disks' ;
echo "-----" ; done
```

たとえば、1つのノードのデータで3つのディスクが表示される場合があります。

```
NODE: 15fc0edc-eb8d-4c7f-8dc0-a2a25d5e09e3
[
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sda",
    "wwn_vendor_extension": "0x1ea4dcc412a9632b",
    "wwn_with_extension": "0x61866da04f3807001ea4dcc412a9632b",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380700",
    "serial": "61866da04f3807001ea4dcc412a9632b"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdb",
    "wwn_vendor_extension": "0x1ea4e13c12e36ad6",
    "wwn_with_extension": "0x61866da04f380d001ea4e13c12e36ad6",
    "model": "PERC H330 Mini",
    "wwn": "0x61866da04f380d00",
    "serial": "61866da04f380d001ea4e13c12e36ad6"
  }
  {
    "size": 299439751168,
    "rotational": true,
    "vendor": "DELL",
    "name": "/dev/sdc",
```

```

    "wnn_vendor_extension": "0x1ea4e31e121cfb45",
    "wnn_with_extension": "0x61866da04f37fc001ea4e31e121cfb45",
    "model": "PERC H330 Mini",
    "wnn": "0x61866da04f37fc00",
    "serial": "61866da04f37fc001ea4e31e121cfb45"
  }
]
-----

```

以下の例では、ルートデバイスを、シリアル番号 **61866da04f37fc001ea4e31e121cfb45** の disk 2 に設定します。そのためには、ノードの定義に **root_device** パラメーターを追加する必要があります。

```

$ ironic node-update 15fc0edc-eb8d-4c7f-8dc0-a2a25d5e09e3 add
properties/root_device='{ "serial": "61866da04f37fc001ea4e31e121cfb45" }'

```

これにより、**director** がルートディスクとして使用する特定のディスクを識別しやすくなります。オーバークラウドの作成の開始時には、**director** はこのノードをプロビジョニングして、オーバークラウドのイメージをこのディスクに書き込みます。その他のディスクは、**Ceph Storage** ノードのマッピングに使用されます。



重要

name でルートディスクを設定しないでください。この値は、ノードのブート時に変更される可能性があります。

2.6. オーバークラウドでの CEPH STORAGE の有効化

オーバークラウドのイメージには、コントローラークラスターに **Ceph OSD** ノードと **Ceph Monitor** の両方を自動的に設定するための **Ceph** サービスと必要な **Puppet** モジュールがすでに含まれています。オーバークラウドの **Heat** テンプレートコレクションには、**Ceph Storage** の設定を有効化するために必要な手順も記載されています。ただし、**director** は **Ceph Storage** を有効化して、対象の設定を渡すための情報を必要とします。この情報を渡すには、**storage-environment.yaml** の環境ファイルを **stack** ユーザーの **templates** ディレクトリーにコピーします。

```

$ cp /usr/share/openstack-tripleo-heat-templates/environments/storage-
environment.yaml ~/templates/

```

storage-environment.yaml のコピーで以下のオプションを変更します。

CinderEnableIscsiBackend

iSCSI バックエンドを有効にするパラメーター。 **false** に設定します。

CinderEnableRbdBackend

Ceph Storage バックエンドを有効にするパラメーター。 **true** に設定します。

CinderEnableNfsBackend

NFS バックエンドを有効にするパラメーター。 **false** に設定します。

NovaEnableRbdBackend

Nova の一時ストレージ用に **Ceph Storage** を有効にするパラメーター。 **true** に設定します。

GlanceBackend

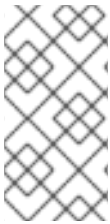
Glance で使用するバックエンドを定義します。イメージに **Ceph Storage** を使用するには、**rbd** に設定します。

parameter_defaults セクションには、以下の行を追加して **neutron** ネットワーク種別に **vxlan** を設定します。

```
NeutronNetworkType: vxlan
```

次に **resource_registry** の各エントリーを各リソースの絶対パスを参照するように変更します。

```
resource_registry:
  OS::TripleO::Services::CephMon: /usr/share/openstack-tripleo-heat-
    templates/puppet/services/ceph-mon.yaml
  OS::TripleO::Services::CephOSD: /usr/share/openstack-tripleo-heat-
    templates/puppet/services/ceph-osd.yaml
  OS::TripleO::Services::CephClient: /usr/share/openstack-tripleo-heat-
    templates/puppet/services/ceph-client.yaml
```



注記

storage-environment.yaml には、Heat を直接使用して Ceph Storage を設定するためのオプションも含まれています。ただし、**director** はこれらのノードを作成して、このシナリオでは、自動的に設定値を定義するので、これらのオプションは必要ありません。

2.7. CEPH STORAGE ノードのディスクレイアウトのマッピング

デフォルトマッピングは、Ceph Storage にルートディスクを使用しますが、実稼動環境の多くは、ストレージやジャーナリング用のパーティションに別個のディスクを複数使用します。このような状況では、以前にコピーした **storage-environment.yaml** ファイルの一部として、ストレージマップを定義します。

storage-environment.yaml ファイルと以下のスニペットを **parameter_defaults** に編集します。

```
ExtraConfig:
  ceph::profile::params::osds:
```

このセクションにより、オーバークラウドに Hiera データが追加されます。Puppet は、設定時にこのデータをカスタムのパラメーターとして使用します。**ceph::profile::params::osds** パラメーターを使用して、適切なディスクとジャーナルパーティションをマッピングします。たとえば、4 つのディスクがある Ceph ノードでは以下のように割り当てられます。

- **/dev/sda**: オーバークラウドのイメージを含むルートディスク
- **/dev/sdb**: ジャーナルのパーティションが含まれるディスク。これは通常、システムパフォーマンスの向上に役立つソリッドステートドライブ (SSD) です。
- **/dev/sdc** および **/dev/sdd**: OSD のディスク

この例では、以下のような内容が含まれるマッピングとなります。

```
ceph::profile::params::osds:
  '/dev/sdc':
    journal: '/dev/sdb'
```

```

'/dev/sdd':
  journal: '/dev/sdb'

```

ジャーナル用に別のディスクを使用しない場合には、OSD ディスクに併置されているジャーナルを使用します。journal パラメーターには空の値を渡します。

```

ceph::profile::params::osds:
  '/dev/sdb': {}
  '/dev/sdc': {}
  '/dev/sdd': {}

```

注記

一部のノードでは、リブート中にディスクパス (例: **/dev/sdb**、**/dev/sdc**) が全く同じブロックデバイスをポイントしない場合があります。そのような **CephStorage** ノードがある場合には、**/dev/disk/by-path/** のシンボリックリンクで、各ディスクを指定してください。以下に例を示します。

```

ceph::profile::params::osds:
  '/dev/disk/by-path/pci-0000:00:17.0-ata-2-part1':
    journal: '/dev/nvme0n1'
  '/dev/disk/by-path/pci-0000:00:17.0-ata-2-part2':
    journal: '/dev/nvme0n1'

```

これにより、ブロックデバイスのマッピングが全デプロイメントで一貫されます。

異なる種別のディスクを混在させた **Ceph** ノードをデプロイすることも可能です (例: 同じ物理ホスト上に SSD と SATA ディスクを混在させるなど)。通常の **Ceph** デプロイメントでは、これは「[Placing Different Pools on Different OSDs](#)」に記載のように、**CRUSH** マップで設定されます。このようなデプロイメントをマッピングする場合には、**storage-environment.yaml** の **ExtraConfig** セクションに以下の行を追加します。

```
ceph::osd_crush_update_on_start: false
```

次に、オーバークラウドのデプロイ時に **Ceph Storage** ノードにディスクマッピングが使用されるように **~/templates/storage-environment.yaml** ファイルを保存します。ストレージで必要な設定が開始されるように、デプロイメント内にこのファイルを追加します。

2.8. CEPH STORAGE ノードのディスクのクリーニング

Ceph Storage OSD およびジャーナルのパーティションには **GPT** ディスクラベルが必要です。これは、**Ceph** OSD サービスをインストールする前に **Ceph Storage** 上の追加のディスクを **GPT** に変換する必要があることを意味します。この変換を行うには、そのディスクから全メタデータを削除して、**director** が **GPT** ラベルを設定できるようにする必要があります。

以下の設定を **/usr/share/instack-undercloud/undercloud.conf** に追加すると、**director** がデフォルトでディスクのメタデータをすべて削除するように設定できます。

```
clean_nodes=true
```

このオプションでは、**Bare Metal Provisioning** サービスが追加のステップを実行してノードをブートし、ノードが **available** に設定されると毎回、ディスクのクリーニングを実行します。これにより、初回のイントロスペクションが終了して、各デプロイメントが開始する前に電源サイクルがもう 1

つ追加されます。Bare Metal Provisioning サービスは **wipefs --force --all** を使用してクリーニングを実行します。



警告

wipefs --force --all により、ディスク上の全データおよびメタデータが削除されますが、**Secure Erase** は実行されません。**Secure Erase** には、はるかに長く時間がかかります。

2.9. CEPH OBJECT GATEWAY のデプロイ

Ceph Object Gateway は、Ceph Storage クラスター内のオブジェクトストレージレイバリティへのインターフェースを使用してアプリケーションを提供します。**Ceph Object Gateway** をデプロイすると、デフォルトの **Object Storage** サービス (**swift**) を Ceph に置き換えることができるようになります。詳しい情報は、『[Object Gateway Guide for Red Hat Enterprise Linux](#)』を参照してください。

デプロイメントで **Ceph Object Gateway** を有効にするには、以下のスニペットを環境ファイル (`~/templates/storage-environment.yaml`) の **resource_registry** に追加します。

```
OS::TripleO::Services::CephRgw: /usr/share/openstack-tripleo-heat-templates/puppet/services/ceph-rgw.yaml
OS::TripleO::Services::SwiftProxy: OS::Heat::None
OS::TripleO::Services::SwiftStorage: OS::Heat::None
OS::TripleO::Services::SwiftRingBuilder: OS::Heat::None
```

このスニペットは、**Ceph Object Gateway** をデプロイするのに加えて、**Object Storage** サービス (**swift**) を無効化します。



注記

これらのリソースも `/usr/share/openstack-tripleo-heat-templates/environments/ceph-radosgw.yaml` にあり、デプロイ中にこの環境ファイルを直接呼び出すことも可能です。本書では、`/home/stack/templates/storage-environment.yaml` でリソースを直接定義して、すべてのリソースとパラメーターを単一の環境ファイルで一元管理します（「[付録A 環境ファイルのサンプル: Ceph クラスターの作成](#)」に記載）。

Ceph Object Gateway で、デフォルトの **Object Storage** サービスを簡単に置き換えることができるため、通常 **swift** を使用するその他のサービスはすべて、**Ceph Object Gateway** を代わりに使用して、追加の設定なしでシームレスに起動することができます。たとえば、**Block Storage Backup** サービス (**cinder-backup**) の設定時には、**Ceph Object Gateway** を使用して **swift** をターゲットバックエンドとして設定します（「[バックアップサービスで Ceph を使用する設定](#)」を参照）。

2.10. バックアップサービスで CEPH を使用する設定

Block Storage Backup サービス (**cinder-backup**) はデフォルトで無効になっています。環境ファイル (`~/templates/storage-environment.yaml`) の **resource_registry** に以下の行を追加して、このサービスを有効化することができます。


```
OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-templates/puppet/services/pacemaker/cinder-backup.yaml
```

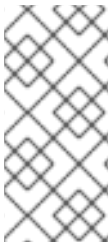


注記

このリソースも `/usr/share/openstack-tripleo-heat-templates/environments/cinder-backup.yaml` で定義され、デプロイ中に直接呼び出すことも可能です。本書では、その代わりに `/home/stack/templates/storage-environment.yaml` でリソースを直接定義して、すべてのリソースとパラメーターを単一の環境ファイルで一元管理します（「[付録A 環境ファイルのサンプル: Ceph クラスターの作成](#)」に記載）。

次に、**cinder-backup** サービスが Ceph にバックアップを保管するように設定します。これには、**cinder-backup** サービスが Ceph Object Storage を使用するように設定する必要があります（「[Ceph Object Gateway のデプロイ](#)」に記載のように、**Ceph Object Gateway** をデプロイすることも前提とします）。そのためには、環境ファイルの `parameter_defaults` に以下の行を追加します。

```
CinderBackupBackend: swift
```



注記

Ceph Object Gateway をデプロイする場合に、**Ceph** ブロックデバイスをバックアップのターゲットとして使用するには、以下の設定を代わりに使用してください。

```
CinderBackupBackend: ceph
```

2.11. CEPH ノードごとに複数のボンディングされたインターフェースを設定する方法

ボンディングされたインターフェースを使用すると、複数の NIC を組み合わせてネットワーク接続に冗長性を追加できます。**Ceph** ノードに十分な NIC がある場合には、ノードごとに **複数のボンディングされたインターフェース** を作成すると、これをさらに一歩進めることができます。

この構成により、ノードが必要とする各ネットワーク接続にボンディングされたインターフェースを使用することができるようになり、冗長性と各ネットワークの専用接続の両方が提供されます。

この構成を最も簡単に実装するには、2つのボンディングを使用して、**Ceph** ノードが使用する各ストレージネットワークに1つずつボンディングを設定します。**Ceph** ノードが使用するストレージネットワークは以下のとおりです。

フロントエンドストレージネットワーク (StorageNet)

Ceph クライアントは、このネットワークを使用して **Ceph** クラスターと対話します。

バックエンドストレージネットワーク (StorageMgmtNet)

Ceph クラスターは、このネットワークを使用して、クラスターの **placement group** ポリシーに従ってデータのバランスを取ります。詳しい情報は、「[Placement Groups \(PG\)](#)」（『[Red Hat Ceph Architecture Guide](#)』）を参照してください。

director は複数のボンディングされた NIC をデプロイするためのテンプレートのサンプルは提供していないので、この設定を行うには、ネットワークインターフェースのテンプレートをカスタマイズする必要があります。ただし、**director** は単一のボンディングされたインターフェースをデプロイするテンプレート

レート (/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml) を提供しているので、このテンプレートで追加の NIC 用のボンディングされたインターフェースを定義して追加することができます。



注記

詳しい手順については、「[カスタムのインターフェーステンプレートの作成](#)」(『[オーバークラウドの高度なカスタマイズ](#)』ガイド)を参照してください。この項では、ブリッジの異なる構成要素とボンディングの定義についても説明しています。

以下のスニペットには、/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml で定義されている単一のボンディングされたインターフェース用のデフォルトの定義が記載されています。

```
type: ovs_bridge // ❶
name: br-bond
members:
-
  type: ovs_bond // ❷
  name: bond1 // ❸
  ovs_options: {get_param: BondInterfaceOvsOptions} ❹
  members: // ❺
  -
    type: interface
    name: nic2
    primary: true
  -
    type: interface
    name: nic3
-
  type: vlan // ❻
  device: bond1 // ❼
  vlan_id: {get_param: StorageNetworkVlanID}
  addresses:
  -
    ip_netmask: {get_param: StorageIpSubnet}
-
  type: vlan
  device: bond1
  vlan_id: {get_param: StorageMgmtNetworkVlanID}
  addresses:
  -
    ip_netmask: {get_param: StorageMgmtIpSubnet}
```

- ❶ **br-bond** という名前の単一のブリッジは、このテンプレートで定義されているボンディングをメンバーとします。この行は、ブリッジの種別 (OVS) を定義しています。
- ❷ **br-bond** ブリッジの最初のメンバーは、ボンディングされたインターフェース自体で、**bond1** という名前が付いています。この行は、**bond1** のボンディングの種別を定義しており、これも OVS に指定されています。
- ❸ デフォルトのボンディングは、この行で定義されているように **bond1** という名前です。

- 4 **ovs_options** のエントリは、**director** が特定の ボンディングモジュールディレクティブのセットを使用するように指示します。これらのディレクティブは、**BondInterfaceOvsOptions** に渡されます。これも同じファイルで設定できます。設定方法についての説明は、「[ボンディングモジュールのディレクティブの設定](#)」を参照してください。
- 5 ボンディングの **members** セクションは、**bond1** でボンディングされるネットワークインターフェースを定義します。この場合は、ボンディングされるインターフェースは **nic2** (プライマリインターフェースとして設定) と **nic3** を使用します。
- 6 **br-bond** ブリッジには、他にも 2 つのメンバーが含まれています。これは、フロントエンドストレージネットワーク (**StorageNetwork**) とバックエンドストレージネットワーク (**StorageMgmtNetwork**) の両方の VLAN です。
- 7 **device** パラメーターは、VLAN が使用する必要のあるデバイスを定義します。この場合は、両方の VLAN がボンディングされたインターフェース **bond1** を使用します。

NIC を少なくとも 2 つ追加すると、ブリッジとボンディングされたインターフェースをもう 1 つ定義できます。これで、VLAN の 1 つを新しいボンディングされたインターフェースに移動できるようになります。これにより、スループットが高くなり、両ストレージネットワーク接続の信頼性が向上します。

`/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml` をこの目的でカスタマイズする場合には、デフォルトの OVS (**type: ovs_bond**) の代わりに Linux ボンディング (**type: linux_bond**) を使用することも推奨します。このボンディングの種別は、エンタープライズレベルの実稼働デプロイメントにより適しています。

以下の編集済みスニペットは、**bond2** という名前の新しい Linux ボンディングをメンバーとする追加の OVS ブリッジ (**br-bond2**) を定義します。**bond2** インターフェースは、2 つの追加の NIC (**nic4** と **nic5**) を使用し、バックエンドストレージネットワークトラフィック専用に使われます。

```
type: ovs_bridge
name: br-bond
members:
-
  type: linux_bond
  name: bond1
  bonding_options: {get_param: BondInterfaceOvsOptions} // 1
  members:
  -
    type: interface
    name: nic2
    primary: true
  -
    type: interface
    name: nic3
-
  type: vlan
  device: bond1
  vlan_id: {get_param: StorageNetworkVlanID}
  addresses:
  -
    ip_netmask: {get_param: StorageIpSubnet}
-
type: ovs_bridge
```

```

name: br-bond2
members:
-
  type: linux_bond
  name: bond2
  bonding_options: {get_param: BondInterfaceOvsOptions}
  members:
  -
    type: interface
    name: nic4
    primary: true
  -
    type: interface
    name: nic5
-
  type: vlan
  device: bond1
  vlan_id: {get_param: StorageMgmtNetworkVlanID}
  addresses:
  -
    ip_netmask: {get_param: StorageMgmtIpSubnet}

```

- 1** **bond1** および **bond2** は両方とも (OVS ではなく) Linux ボンディングで、**ovs_options** の代わりに **bonding_options** を使用してボンディングディレクティブを設定します。関連情報については、「[ボンディングモジュールのディレクティブの設定](#)」を参照してください。

このカスタマイズされたテンプレートの完全な内容は、「[付録B カスタムインターフェーステンプレートの例: 複数のボンディングされたインターフェース](#)」を参照してください。

2.11.1. ボンディングモジュールのディレクティブの設定

ボンディングされたインターフェースの追加/設定が完了した後は、**BondInterfaceOvsOptions** パラメーターを使用して各インターフェースが使用すべきディレクティブを指定します。このパラメーターは、**/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml** の **parameters:** セクションにあります。以下のスニペットには、このパラメーターのデフォルトの定義 (空欄) を示しています。

```

BondInterfaceOvsOptions:
  default: ''
  description: The ovs_options string for the bond interface. Set
               things like lacp=active and/or bond_mode=balance-slb
               using this option.
  type: string

```

default: の行に必要なオプションを定義します。たとえば、**802.3ad** (モード 4) と LACP レート 1 (fast) を使用するには、**'mode=4 lacp_rate=1'** を以下のように設定します。

```

BondInterfaceOvsOptions:
  default: 'mode=4 lacp_rate=1'
  description: The bonding_options string for the bond interface. Set
               things like lacp=active and/or bond_mode=balance-slb
               using this option.
  type: string

```

サポートされているボンディングオプションについては、「[付録C Open vSwitch ボンディングのオプション](#)」(『[オーバークラウドの高度なカスタマイズ](#)』ガイド)を参照してください。カスタマイズした `/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml` テンプレートの完全な内容は、「[付録B カスタムインターフェーステンプレートの例: 複数のボンディングされたインターフェース](#)」を参照してください。

2.12. CEPH STORAGE CLUSTER のカスタマイズ

ExtraConfig フックを使用して Ceph Storage ノードのデフォルト設定パラメーターを上書きして、Puppet 設定を渡すようにデータを定義できます。このデータを渡す方法は 2 種類存在します。

方法 1: Puppet のデフォルト設定の変更

オーバークラウドの設定時に **ceph Puppet** モジュールに渡すパラメーターをカスタマイズします。これらのパラメーターは、`/etc/puppet/modules/ceph/manifests/profile/params.conf` で定義される **ceph::profile::params** Puppet クラスに含まれます。たとえば、以下の環境ファイルのスニペットは、**ceph::profile::params** クラスからのデフォルトの **osd_journal_size** パラメーターをカスタマイズし、デフォルト設定を上書きします。

```
parameter_defaults:
  ExtraConfig:
    ceph::profile::params::osd_journal_size: 2048
```

この内容を環境ファイル (例: **ceph-settings.yaml**) に追加して、「[オーバークラウドの作成](#)」で **openstack overcloud deploy** コマンドを実行する際に追加します。以下に例を示します。

```
$ openstack overcloud deploy --templates -e /home/stack/templates/storage-environment.yaml -e /home/stack/templates/ceph-settings.yaml
```

方法 2: 任意の設定デフォルト

方法 1 で、設定の必要な固有のパラメーターが追加されない場合には、**ceph::conf::args** Puppet クラスを使用して、任意の Ceph Storage パラメーターを渡すことができます。このクラスでは、**stanza/key** 形式とパラメーターの値を定義する **value** を使用してパラメーターの名前を受け入れます。これらの設定値で、各ノード上の **ceph.conf** ファイルが設定されます。たとえば、**ceph.conf** の **global** セクションにある **max_open_files** パラメーターを変更するには、環境ファイルで以下の構造を使用します。

```
parameter_defaults:
  ExtraConfig:
    ceph::conf::args:
      global/max_open_files:
        value: 131072
```

この内容を環境ファイル (例: **ceph-settings.yaml**) に追加して、「[オーバークラウドの作成](#)」で **openstack overcloud deploy** コマンドを実行する際に追加します。以下に例を示します。

```
$ openstack overcloud deploy --templates -e /home/stack/templates/storage-environment.yaml -e /home/stack/templates/ceph-settings.yaml
```

この **ceph.conf** ファイルには最終的に、以下のパラメーターが自動的に入力されるはずです。

```
[global]
max_open_files = 131072
```

2.12.1. 異なる Ceph プールへのカスタムの属性の割り当て

デフォルトでは、**director** で作成した Ceph プールには、同じ配置グループ (**pg_num** および **pgp_num**) とサイズが設定されます。「[Ceph Storage Cluster のカスタマイズ](#)」に記載のいずれかの方法を使用して、これらの設定をグローバルで上書きして、全プールに同じ値を適用することが可能です。

各 Ceph プールに異なる属性を適用することもできます。そのためには、**CephPools** を以下のように使用します。

```
parameter_defaults:
  ExtraConfig:
    CephPools:
      POOL:
        size: 5
        pg_num: 128
        pgp_num: 128
```

POOL は設定するプールの名前に置き換えて、その後に **size**、**pg_num**、**pgp_num** の設定を指定します。

ヒント

一般的な Ceph ユースケースの標準的なプール設定は、「[Ceph Placement Groups \(PGs\) per Pool Calculator](#)」を参照してください。この計算ツールは、通常、Ceph プールを手動で設定するためのコマンドの生成に使用されます。このデプロイメントでは、仕様に基づいて **director** がプールを設定します。

2.13. ロールへのノードとフレーバーの割り当て

オーバークラウドのデプロイメントプランニングでは、各ロールに割り当てるノード数とフレーバーを指定する必要があります。すべての Heat テンプレートのパラメーターと同様に、これらのロールの仕様は環境ファイル(この場合は `~/templates/storage-environment.yaml`)の **parameter_defaults** セクションで宣言する必要があります。

この設定には、以下のパラメーターを使用します。

表2.1 オーバークラウドノードのロールとフレーバー

Heat テンプレートのパラメーター	説明
ControllerCount	スケールアウトするコントローラーノード数
OvercloudControlFlavor	コントローラーノード (control) に使用するフレーバー
ComputeCount	スケールアウトするコンピュートノード数

Heat テンプレートのパラメーター	説明
OvercloudComputeFlavor	コンピュータード (compute) に使用するフレーバー
CephStorageCount	スケールアウトする Ceph Storage (OSD) ノード数
OvercloudCephStorageFlavor	Ceph Storage (OSD) ノード (ceph-storage) に使用するフレーバー
CephMonCount	スケールアウトする専用の Ceph MON ノード数
CephMdsCount	スケールアウトする専用の Ceph MDS ノード数
OvercloudCephMonFlavor	専用の Ceph MON ノード (ceph-mon) に使用するフレーバー
OvercloudCephMdsFlavor	専用の Ceph MDS ノード (ceph-mds) に使用するフレーバー



重要

CephMonCount、**CephMdsCount**、**OvercloudCephMonFlavor**、**OvercloudCephMdsFlavor** のパラメーターは (**ceph-mon** および **ceph-mds** フレーバーと共に)、「[専用ノード上でのその他の Ceph サービスのデプロイ](#)」に記載のように、カスタムの **CephMON** および **CephMds** ロールを作成した場合にのみ有効となります。

たとえば、オーバークラウドが各ロール (Controller、Compute、Ceph-Storage、CephMon) に 3 つずつノードをデプロイするように設定するには、**parameter_defaults** に以下の設定を追加します。

```
parameter_defaults:
  ControllerCount: 3
  OvercloudControlFlavor: control
  ComputeCount: 3
  OvercloudComputeFlavor: compute
  CephStorageCount: 3
  OvercloudCephStorageFlavor: ceph-storage
  CephMonCount: 3
  OvercloudCephMonFlavor: ceph-mon
  CephMdsCount: 3
  OvercloudCephMdsFlavor: ceph-mds
```



注記

Heat テンプレートのパラメーターのより詳細な一覧は、『[director のインストールと使用方法](#)』の「[CLI ツールでのオーバークラウドの作成](#)」を参照してください。

2.14. オーバークラウドの作成

オーバークラウドの作成には、**openstack overcloud deploy** コマンドに追加の引数を指定する必要があります。以下に例を示します。

```
$ openstack overcloud deploy --templates -r
/home/stack/templates/roles_data_custom.yaml -e
/home/stack/templates/storage-environment.yaml --ntp-server pool.ntp.org
```

上記のコマンドは、以下のオプションを使用します。

- **--templates:** デフォルトの Heat テンプレートコレクション (`/usr/share/openstack-tripleo-heat-templates/`) からオーバークラウドを作成します。
- **-r /home/stack/templates/roles_data_custom.yaml:** 「[専用ノード上でのその他の Ceph サービスのデプロイ](#)」でカスタマイズしたロールの定義を指定し、カスタムロールを Ceph MON サービスまたは Ceph MDS サービスに追加します。これらのロールにより、いずれかのサービスを専用のノードにインストールすることができます。
- **-e /home/stack/templates/storage-environment.yaml:** 別の環境ファイルをオーバークラウドデプロイメントに追加します。この場合は、Ceph Storage の設定を含むストレージ環境ファイルです。
- **--ntp-server pool.ntp.org:** NTP サーバーを設定します。

`/home/stack/templates/storage-environment.yaml` で使用する全設定の概要については、「[付録A 環境ファイルのサンプル: Ceph クラスターの作成](#)」を参照してください。

ヒント

オプションの完全な一覧を表示するには、以下を実行します。

```
$ openstack help overcloud deploy
```

詳しい情報は、「[director のインストールと使用方法](#)」ガイドの「[オーバークラウドのパラメーター設定](#)」を参照してください。

オーバークラウドの作成プロセスが開始され、**director** によりノードがプロビジョニングされます。このプロセスは完了するまで多少時間がかかります。オーバークラウドの作成のステータスを確認するには、**stack** ユーザーとして別のターミナルを開き、以下を実行します。

```
$ source ~/stackrc
$ heat stack-list --show-nested
```

2.15. オーバークラウドへのアクセス

director は、**director** ホストがオーバークラウドと対話するための設定と認証を行うスクリプトを生成します。このファイル (**overcloudrc**) は、**stack** ユーザーのホームディレクトリーに保存されます。このファイルを使用するには、以下のコマンドを実行します。

```
$ source ~/overcloudrc
```

これにより、**director** ホストの CLI からオーバークラウドと対話するために必要な環境変数が読み込まれます。**director** ホストとの対話に戻るには、以下のコマンドを実行します。

■


```
$ source ~/stackrc
```

2.16. CEPH STORAGE ノードの監視

オーバークラウドの作成が完了したら、正しく機能していることを確認するため、**Ceph Storage Cluster** のステータスをチェックすることを推奨します。これには、**heat-admin** ユーザーとしてコントローラーノードにログインします。

```
$ nova list
$ ssh heat-admin@192.168.0.25
```

クラスターの正常性を確認します。

```
$ sudo ceph health
```

クラスターに問題がない場合は、上記のコマンドにより、**HEALTH_OK** というレポートが返されます。これは、クラスターを安全に使用できることを意味します。

Ceph モニタークォーラムのステータスを確認します。

```
$ sudo ceph quorum_status
```

これにより、クォーラムに参加するモニターとどれがリーダーであるかが表示されます。

全 Ceph OSD が実行中であるかどうかを確認します。

```
$ ceph osd stat
```

Ceph Storage Cluster の監視に関する詳しい情報は、『**Red Hat Ceph Storage Administration Guide**』の「**Monitoring**」を参照してください。

2.17. 環境のリブート

環境を再起動する必要がある状況が発生する場合があります。たとえば、物理サーバーを変更する必要がある場合や、停電から復旧する必要がある場合などです。このような状況では、**Ceph Storage** ノードが正しく起動されることが重要です。

以下の順序でノードを起動してください。

- **全 Ceph Monitor ノードを最初にブートします。**これにより、高可用性クラスター内の Ceph Monitor サービスを確実にアクティブにします。デフォルトでは、Ceph Monitor サービスはコントローラーノードにインストールされます。Ceph Monitor がカスタムロールで Controller とは別の場合には、このカスタムの Ceph Monitor ロールを必ずアクティブにしてください。
- **Ceph Storage ノードすべてを起動します。**これにより、Ceph OSD クラスターはコントローラーノード上のアクティブな Ceph Monitor クラスターに接続できるようになります。

以下の手順に従って Ceph Storage ノードをリブートします。

1. **Ceph MON** またはコントローラーノードにログインして、Ceph Storage クラスターのリバランスを一時的に無効にします。


```
$ sudo ceph osd set noout
$ sudo ceph osd set norebalance
```

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

3. ノードを再起動します。

```
$ sudo reboot
```

4. ノードが起動するまで待ちます。

5. ノードにログインして、クラスターのステータスを確認します。

```
$ sudo ceph -s
```

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

6. ノードからログアウトして、次のノードを再起動し、ステータスを確認します。全 Ceph Storage ノードが再起動されるまで、このプロセスを繰り返します。

7. 完了したら、Ceph MON またはコントローラーノードにログインして、クラスターのリバランスを再度有効にします。

```
$ sudo ceph osd unset noout
$ sudo ceph osd unset norebalance
```

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

```
$ sudo ceph status
```

オーバークラウドすべてが同時に起動する状況が発生した場合には、Ceph OSD サービスが Ceph Storage ノード上で正しく起動されない場合があります。そのような場合には、Ceph Storage OSD を再起動して、Ceph Monitor サービスに接続できるようにします。各 Ceph Storage ノードで以下のコマンドを実行します。

```
$ sudo systemctl restart 'ceph*'
```

以下のコマンドを使用して、Ceph Storage ノードクラスターの **HEALTH_OK** のステータスを検証します。

```
$ sudo ceph status
```

2.18. CEPH クラスターのスケールアップ

必要な Ceph Storage ノードの数でデプロイメントを再度実行して、オーバークラウド内の Ceph Storage ノードの数をスケールアップすることができます。

この操作を実行する前に、更新するデプロイメント用に十分なノードがあることを確認してください。これらのノードは、**director** に登録済みで、適宜にタグ付けされている必要があります。

新規 Ceph Storage ノードの登録

新しい Ceph Storage ノードを **director** に登録するには、以下のステップを実行します。

1. **director** ホストに **stack** ユーザーとしてログインし、**director** の設定を初期化します。

```
$ source ~/stackrc
```

2. 新規ノードの定義テンプレート (例: **instackenv-scale.json**) で、新しいノードのハードウェアと電源管理の詳細を定義します。

3. このファイルを **OpenStack director** にインポートします。

```
$ openstack baremetal import --json ~/instackenv-scale.json
```

ノードの定義テンプレートをインポートすると、そのテンプレートで定義されている各ノードが **director** に登録されます。

4. カーネルと **ramdisk** イメージを全ノードに割り当てます。

```
$ openstack baremetal configure boot
```



注記

新規ノードの登録に関する詳しい情報は、「[ノードの登録](#)」を参照してください。

新規ノードの手動でのタグ付け

各ノードを登録した後は、ハードウェアを検査して、ノードを特定のプロファイルにタグ付けする必要があります。プロファイルタグにより、ノードがフレーバーに照合され、そのフレーバーはデプロイメントロールに割り当てられます。

新規ノードを検査してタグ付けするには、以下のステップに従います。

1. ハードウェアのイントロスペクションをトリガーして、各ノードのハードウェア属性を取得します。

```
$ openstack baremetal introspection bulk start
```



重要

このプロセスが最後まで実行されて正常に終了したことを確認してください。ベアメタルの場合には、通常 15 分ほどかかります。

2. ノード一覧を取得して **UUID** を識別します。

```
$ ironic node-list
```

3. 各ノードの **properties/capabilities** パラメーターに **profile** オプションを追加して、ノードを特定のプロファイルに手動でタグ付けします。
たとえば、以下のコマンドは、3つの追加のノードを **ceph-storage** プロファイルでタグ付けします。

```
$ ironic node-update 551d81f5-4df2-4e0f-93da-6c5de0b868f7 add
properties/capabilities='profile:ceph-storage,boot_option:local'
$ ironic node-update 5e735154-bd6b-42dd-9cc2-b6195c4196d7 add
properties/capabilities='profile:ceph-storage,boot_option:local'
$ ironic node-update 1a2b090c-299d-4c20-a25d-57dd21a7085b add
properties/capabilities='profile:ceph-storage,boot_option:local'
```

ヒント

タグ付け/登録したノードが複数のディスクを使用している場合には、**director** が各ノードで特定のルートディスクを使用するように設定できます。その手順は、「[Ceph Storage ノードのルートディスクの定義](#)」を参照してください。

Ceph Storage ノードを追加したオーバークラウドの再デプロイ

新規ノードの登録とタグ付けが済んだ後には、オーバークラウドを再デプロイして Ceph Storage ノードの数をスケールアップすることができます。この操作を行う際には、環境ファイル(この場合は `~/templates/storage-environment.yaml`) の **parameter_defaults** にある **CephStorageCount** パラメーターを設定します。「[ロールへのノードとフレーバーの割り当て](#)」では、オーバークラウドは 3 つの Ceph Storage ノードでデプロイするように設定されています。これを 6 ノードにスケールアップするには、以下の設定を使用します。

```
parameter_defaults:
  ControllerCount: 3
  OvercloudControlFlavor: control
  ComputeCount: 3
  OvercloudComputeFlavor: compute
  CephStorageCount: 6
  OvercloudCephStorageFlavor: ceph-storage
  CephMonCount: 3
  OvercloudCephMonFlavor: ceph-mon
```

この設定で再デプロイすると、オーバークラウドの Ceph Storage ノードは 3 つではなく 6 つとなるはずです。

2.19. CEPH STORAGE ノードのスケールダウンと置き換え

Ceph クラスターをスケールダウンしたり、Ceph Storage ノードを置き換える (例: Ceph Storage ノードに問題がある場合など) 必要がある場合があります。いずれの状況でもオーバークラウドから削除する Ceph Storage ノードは無効化し、リバランスして、データの損失が発生しないようにする必要があります。以下の手順では、Ceph Storage ノードを置き換えるステップを説明します。



注記

以下の手順では、『**Red Hat Ceph Storage Administration Guide**』からの手順を使用して、手動で Ceph Storage ノードを削除します。Ceph Storage ノードの手動での削除に関する詳しい情報は、『**Red Hat Ceph Storage Administration Guide**』の「[Removing OSDs \(Manual\)](#)」を参照してください。

heat-admin ユーザーとして、コントローラーノードまたは Ceph Storage ノードにログインします。**director** の **stack** ユーザーには、**heat-admin** ユーザーにアクセスするための SSH キーがあります。

OSD ツリーを一覧表示して、ノードの OSD を検索します。たとえば、削除するノードには、以下の OSD が含まれる場合があります。

```
-2 0.09998      host overcloud-cephstorage-0
0 0.04999      osd.0                      up  1.00000
1.00000
1 0.04999      osd.1                      up  1.00000
1.00000
```

Ceph Storage ノードの OSD を無効化します。今回は、OSD ID は 0 と 1 です。

```
[heat-admin@overcloud-controller-0 ~]$ sudo ceph osd out 0
[heat-admin@overcloud-controller-0 ~]$ sudo ceph osd out 1
```

Ceph Storage Cluster がリバランスを開始します。このプロセスが完了するまで待機してください。以下のコマンドを使用して、ステータスを確認できます。

```
[heat-admin@overcloud-controller-0 ~]$ sudo ceph -w
```

Ceph クラスターのリバランスが完了したら、削除する Ceph Storage ノード (この場合は **overcloud-cephstorage-0**) に **heat-admin** ユーザーとしてログインし、ノードを停止します。

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo systemctl disable ceph-osd@0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo systemctl disable ceph-osd@1
```

overcloud-cephstorage-0 にログインした状態で、CRUSH マップから削除して、データを受信しないようにします。

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd crush remove osd.0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd crush remove osd.1
```

OSD 認証キーを削除します。

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph auth del osd.0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph auth del osd.1
```

クラスターから OSD を削除します。

```
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd rm 0
[heat-admin@overcloud-cephstorage-0 ~]$ sudo ceph osd rm 1
```

ノードからログアウトして、**stack** ユーザーとして **director** ホストに戻ります。

```
[heat-admin@overcloud-cephstorage-0 ~]$ exit
[stack@director ~]$
```

director が再度プロビジョニングしないように、Ceph Storage ノードを無効にします。

```
[stack@director ~]$ ironic node-list
[stack@director ~]$ ironic node-set-maintenance UUID true
```

Ceph Storage ノードを削除するには、ローカルのテンプレートファイルを使用して **overcloud** スタックへの更新が必要です。最初に、オーバークラウドスタックの **UUID** を特定します。

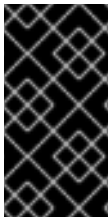
```
$ heat stack-list
```

削除する Ceph Storage ノードの **UUID** を特定します。

```
$ nova list
```

以下のコマンドを実行してスタックからノードを削除し、それに応じてプランを更新します。

```
$ openstack overcloud node delete --stack STACK_UUID --templates -e
ENVIRONMENT_FILE NODE_UUID
```



重要

オーバークラウドの作成時に追加の環境ファイルを渡した場合には、予定外の変更がオーバークラウドに加えられないように、ここで **-e** オプションを使用して環境ファイルを再度渡します。詳しい情報は、『[director のインストールと使用方法](#)』の「[オーバークラウド環境の変更](#)」を参照してください。

stack が更新を完了するまで待機します。**heat stack-list --show-nested** コマンドを使用して、**stack** の更新を監視します。

新規ノードを **director** のノードプールに追加して、Ceph Storage ノードとしてデプロイします。環境ファイル(この場合は `~/templates/storage-environment.yaml`)の **parameter_defaults** セクションの **CephStorageCount** パラメーターを使用して、オーバークラウド内の Ceph Storage ノードの合計数を定義します。以下に例を示します。

```
parameter_defaults:
  ControllerCount: 3
  OvercloudControlFlavor: control
  ComputeCount: 3
  OvercloudComputeFlavor: compute
  CephStorageCount: 3
  OvercloudCephStorageFlavor: ceph-storage
  CephMonCount: 3
  OvercloudCephMonFlavor: ceph-mon
```

ロールごとにノード数を定義する方法については、「[ロールへのノードとフレーバーの割り当て](#)」を参照してください。

環境ファイルを更新したら、通常通りにオーバークラウドを再デプロイします。

```
$ openstack overcloud deploy --templates -e ENVIRONMENT_FILES
```

director は、新しいノードをプロビジョニングして、新しいノードの詳細を用いて **stack** 全体を更新します。

heat-admin ユーザーとしてコントローラーノードにログインして、Ceph Storage ノードのステータスを確認します。以下に例を示します。

```
[heat-admin@overcloud-controller-0 ~]$ sudo ceph status
```

■

osdmap セクションの値が、クラスターに必要なノード数と一致していることを確認します。削除した Ceph Storage ノードは新規ノードに置き換えられました。

2.20. CEPH STORAGE ノードへの OSD ディスクの追加と削除

OSD ディスクで問題が発生して置き換えが必要な場合には、『**Red Hat Ceph Storage Administration Guide**』に記載の通常の手順を使用してください。

- [「Adding an OSD」](#)
- [「Removing an OSD」](#)

第3章 既存の CEPH STORAGE CLUSTER のオーバークラウドとの統合

本章では、オーバークラウドを作成して、既存の Ceph Storage Cluster と統合する方法について説明します。オーバークラウドの作成方法と既存の Ceph Storage Cluster との統合方法に関する説明はいずれも、「[2章 Ceph Storage ノードでのオーバークラウドの作成](#)」を参照してください。

本章のシナリオでは、オーバークラウドは 6 台のノードで構成されます。

- 高可用性のコントローラーノード 3 台
- コンピュートノード 3 台

director は、独自のノードで構成される独立した Ceph Storage Cluster をオーバークラウドに統合します。このクラスターは、オーバークラウドとは別々に管理されます。たとえば、Ceph Storage Cluster は、OpenStack Platform **director** ではなく Ceph 管理ツールを使用してスケールリングします。詳しくは、[Red Hat Ceph](#) のドキュメントを参照してください。

すべての OpenStack マシンは、電源管理に IPMI を使用したベアメタルマシンです。**director** により Red Hat Enterprise Linux 7 のイメージが各ノードにコピーされるため、これらのノードではオペレーティングシステムは必要ありません。

director は、イントロスペクションおよびプロビジョニングプロセス中に、プロビジョニングネットワークを使用してコントローラーノードとコンピュートノードと通信します。すべてのノードは、ネイティブの VLAN 経由でネットワークに接続します。この例では、以下の IP アドレスの割り当てで、プロビジョニングサブネットとして 192.0.2.0/24 を使用します。

ノード名	IP アドレス	MAC アドレス	IPMI IP アドレス
director	192.0.2.1	aa:aa:aa:aa:aa:aa	
コントローラー 1	定義済みの DHCP	b1:b1:b1:b1:b1:b1	192.0.2.205
コントローラー 2	定義済みの DHCP	b2:b2:b2:b2:b2:b2	192.0.2.206
コントローラー 3	定義済みの DHCP	b3:b3:b3:b3:b3:b3	192.0.2.207
コンピュート 1	定義済みの DHCP	c1:c1:c1:c1:c1:c1	192.0.2.208
コンピュート 2	定義済みの DHCP	c2:c2:c2:c2:c2:c2	192.0.2.209
コンピュート 3	定義済みの DHCP	c3:c3:c3:c3:c3:c3	192.0.2.210

3.1. 既存の CEPH STORAGE CLUSTER の設定

1. お使いの環境に適した Ceph クラスターに以下のプールを作成します。

- **volumes:** OpenStack Block Storage (cinder) のストレージ
- **images:** OpenStack Image Storage (glance) のストレージ
- **vms:** インスタンスのストレージ

- **backups:** OpenStack Block Storage Backup (cinder-backup) のストレージ
- **metrics:** OpenStack Telemetry Metrics (gnocchi) のストレージ
以下のコマンドは指標として使用してください。

```
[root@ceph ~]# ceph osd pool create volumes PGNUM
[root@ceph ~]# ceph osd pool create images PGNUM
[root@ceph ~]# ceph osd pool create vms PGNUM
[root@ceph ~]# ceph osd pool create backups PGNUM
[root@ceph ~]# ceph osd pool create metrics PGNUM
```

PGNUM は **配置グループ** の数に置き換えます。1 OSD につき 100 程度を推奨します。たとえば、OSD の合計数を 100 で乗算して、レプリカ数で除算します (**osd pool default size**)。適切な値を判断するには [Ceph Placement Groups \(PGs\) per Pool Calculator](#) を使用することを推奨します。

2. Ceph クラスターに、以下のパーミッティビティーを指定して **client.openstack** ユーザーを作成します。

- **cap_mon:** allow r
- **cap_osd:** allow class-read object_prefix rbd_children, allow rwx pool=volumes, allow rwx pool=vms, allow rwx pool=images, allow rwx pool=backups, allow rwx pool=metrics
以下のコマンドは指標として使用してください。

```
[root@ceph ~]# ceph auth add client.openstack mon 'allow r' osd
'allow class-read object_prefix rbd_children, allow rwx
pool=volumes, allow rwx pool=vms, allow rwx pool=images, allow
rwx pool=backups, allow rwx pool=metrics'
```

3. 次に、**client.openstack** ユーザー向けに作成された **Ceph client key** をメモします。

```
[root@ceph ~]# ceph auth list
...
client.openstack
key: AQLD0h1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==
caps: [mon] allow r
caps: [osd] allow class-read object_prefix rbd_children, allow rwx
pool=volumes, allow rwx pool=vms, allow rwx pool=images, allow rwx
pool=backups, allow rwx pool=metrics
...
```

上記の出力の **key** の値 (**AQLD0h1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==**) が Ceph のクライアントキーです。

4. 最後に、**Ceph Storage Cluster** の **file system ID** をメモします。この値は、クラスターの設定ファイルにある **fsid** の設定で指定されています (**[global]** のセクション下)。

```
[global]
fsid = 4b5c8c0a-ff60-454b-a1b4-9747aa737d19
...
```




注記

Ceph Storage Cluster の設定ファイルに関する詳しい情報は、『[Red Hat Ceph Storage Configuration Guide](#)』の「[Configuration Reference](#)」を参照してください。

Ceph クライアントキーおよびファイルシステム ID はいずれも、後ほど「[既存の Ceph Storage クラスターとの統合](#)」で使います。

3.2. STACK ユーザーの初期化

stack ユーザーとして **director** ホストにログインし、以下のコマンドを実行して **director** の設定を初期化します。

```
$ source ~/stackrc
```

このコマンドでは、**director** の CLI ツールにアクセスする認証情報が含まれる環境変数を設定します。

3.3. ノードの登録

ノード定義のテンプレート (**instackenv.json**) は JSON ファイル形式で、ノード登録用のハードウェアおよび電源管理の情報が含まれています。以下に例を示します。

```
{
  "nodes": [
    {
      "mac": [
        "bb:bb:bb:bb:bb:bb"
      ],
      "cpu": "4",
      "memory": "6144",
      "disk": "40",
      "arch": "x86_64",
      "pm_type": "pxe_ipmitool",
      "pm_user": "admin",
      "pm_password": "p@55w0rd!",
      "pm_addr": "192.0.2.205"
    },
    {
      "mac": [
        "cc:cc:cc:cc:cc:cc"
      ],
      "cpu": "4",
      "memory": "6144",
      "disk": "40",
      "arch": "x86_64",
      "pm_type": "pxe_ipmitool",
      "pm_user": "admin",
      "pm_password": "p@55w0rd!",
      "pm_addr": "192.0.2.206"
    },
    {
      "mac": [
        "dd:dd:dd:dd:dd:dd"
      ],

```

```

    ],
    "cpu": "4",
    "memory": "6144",
    "disk": "40",
    "arch": "x86_64",
    "pm_type": "pxe_ipmitool",
    "pm_user": "admin",
    "pm_password": "p@55w0rd!",
    "pm_addr": "192.0.2.207"
  },
  {
    "mac": [
      "ee:ee:ee:ee:ee:ee"
    ],
    "cpu": "4",
    "memory": "6144",
    "disk": "40",
    "arch": "x86_64",
    "pm_type": "pxe_ipmitool",
    "pm_user": "admin",
    "pm_password": "p@55w0rd!",
    "pm_addr": "192.0.2.208"
  }
]
{
  "mac": [
    "ff:ff:ff:ff:ff:ff"
  ],
  "cpu": "4",
  "memory": "6144",
  "disk": "40",
  "arch": "x86_64",
  "pm_type": "pxe_ipmitool",
  "pm_user": "admin",
  "pm_password": "p@55w0rd!",
  "pm_addr": "192.0.2.209"
}
{
  "mac": [
    "gg:gg:gg:gg:gg:gg"
  ],
  "cpu": "4",
  "memory": "6144",
  "disk": "40",
  "arch": "x86_64",
  "pm_type": "pxe_ipmitool",
  "pm_user": "admin",
  "pm_password": "p@55w0rd!",
  "pm_addr": "192.0.2.210"
}
]
}

```

テンプレートの作成後に、**stack** ユーザーのホームディレクトリー (**/home/stack/instackenv.json**) にファイルを保存してから、**director** にインポートします。これには、以下のコマンドを実行します。

```
$ openstack baremetal import --json ~/instackenv.json
```

このコマンドでテンプレートをインポートして、テンプレートから **director** に各ノードを登録します。
カーネルと **ramdisk** イメージを全ノードに割り当てます。

```
$ openstack baremetal configure boot
```

director でのノードの登録、設定が完了しました。

3.4. ノードの手動でのタグ付け

各ノードを登録した後には、ハードウェアを検査して、ノードを特定のプロファイルにタグ付けする必要があります。プロファイルタグにより、ノードがフレーバーに照合され、そのフレーバーはデプロイメントロールに割り当てられます。

新規ノードを検査してタグ付けするには、以下のステップに従います。

1. ハードウェアのイントロスペクションをトリガーして、各ノードのハードウェア属性を取得します。

```
$ openstack baremetal introspection bulk start
```



重要

このプロセスが最後まで実行されて正常に終了したことを確認してください。ベアメタルの場合には、通常 15 分ほどかかります。

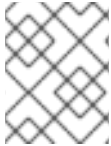
2. ノード一覧を取得して **UUID** を識別します。

```
$ ironic node-list
```

3. 各ノードの **properties/capabilities** パラメーターに **profile** オプションを追加して、ノードを特定のプロファイルに手動でタグ付けします。
たとえば、3つのノードが **control** プロファイルを使用し、別の3つのノードが **compute** プロファイルを使用するようにするには、以下のコマンドを実行します。

```
$ ironic node-update 1a4e30da-b6dc-499d-ba87-0bd8a3819bc0 add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 6faba1a9-e2d8-4b7c-95a2-c7fbdc12129a add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 5e3b2f50-fcd9-4404-b0a2-59d79924b38e add
properties/capabilities='profile:control,boot_option:local'
$ ironic node-update 484587b2-b3b3-40d5-925b-a26a2fa3036f add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d010460b-38f2-4800-9cc4-d69f0d067efe add
properties/capabilities='profile:compute,boot_option:local'
$ ironic node-update d930e613-3e14-44b9-8240-4f3559801ea6 add
properties/capabilities='profile:compute,boot_option:local'
```

profile オプションを追加すると、適切なプロファイルにノードをタグ付けします。



注記

手動でのタグ付けの代わりに、**Automated Health Check (AHC)** ツールを使用し、ベンチマークデータに基づいて、多数のノードに自動でタグ付けします。

3.5. 既存の CEPH STORAGE クラスターとの統合

stack ユーザーのホームディレクトリー内のディレクトリーに `/usr/share/openstack-tripleo-heat-templates/environments/puppet-ceph-external.yaml` のコピーを作成します。

```
[stack@director ~]# mkdir templates
[stack@director ~]# cp /usr/share/openstack-tripleo-heat-templates/environments/puppet-ceph-external.yaml ~/templates/.
```

このファイルを編集して、下記のパラメーターを設定します。

- 絶対パスに **CephExternal** のリソース定義を設定します。

```
OS::TripleO::Services::CephExternal: /usr/share/openstack-tripleo-heat-templates/puppet/services/ceph-external.yaml
```

- **Ceph Storage** 環境の情報を使用して、以下の 3 つのパラメーターを設定します。
 - **CephClientKey:** Ceph Storage Cluster の Ceph クライアントキー。これは、「[既存の Ceph Storage Cluster の設定](#)」で先ほど取得した **key** の値です (例: `AQDL0h1VgEp6FRAAFzT7Zw+Y9V6JJExQAsRnRQ==`)。
 - **CephExternalMonHost:** Ceph Storage Cluster の全 MON ホストの IP をコンマ区切りにしたリスト
 - **CephClusterFSID:** Ceph Storage Cluster のファイルシステム ID。これは、「[既存の Ceph Storage Cluster の設定](#)」で先ほど取得した **Ceph Storage Cluster** の設定ファイルにある **fsid** の値です (例: `4b5c8c0a-ff60-454b-a1b4-9747aa737d19`)。
- **neutron** ネットワーク種別に **vxlan** を設定するには、以下のパラメーターを追加します。
 - **NeutronNetworkType:** `vxlan`
- 必要な場合は、以下のパラメーターと値を使用して、**OpenStack** プールとクライアントユーザーの名前を設定します。
 - **CephClientUserName:** `openstack`
 - **NovaRbdPoolName:** `vms`
 - **CinderRbdPoolName:** `volumes`
 - **GlanceRbdPoolName:** `images`
 - **CinderBackupRbdPoolName:** `backups`
 - **GnocchiRbdPoolName:** `metrics`

3.6. 以前の RED HAT CEPH STORAGE バージョンとの後方互換性

Red Hat OpenStack Platform が、以前のバージョン(Red Hat Ceph Storage 1.3)の外部の Ceph Storage Cluster と統合されている場合には、後方互換性を有効にする必要があります。そのためには、最初に環境ファイル(例: `/home/stack/templates/ceph-backwards-compatibility.yaml`)を作成して、以下の内容を記載します。

```
parameter_defaults:
  ExtraConfig:
    ceph::conf::args:
      client/rbd_default_features:
        value: "1"
```

「[オーバークラウドの作成](#)」に記載のとおり、オーバークラウドのデプロイメントにこのファイルを追加します。

3.7. ロールへのノードとフレーバーの割り当て

オーバークラウドのデプロイメントプランニングでは、各ロールに割り当てるノード数とフレーバーを指定する必要があります。すべての Heat テンプレートのパラメーターと同様に、これらのロールの仕様は環境ファイル(この場合は `~/templates/puppet-ceph.yaml`)の `parameter_defaults` セクションで宣言する必要があります。

この設定には、以下のパラメーターを使用します。

表3.1 オーバークラウドノードのロールとフレーバー

Heat テンプレートのパラメーター	説明
ControllerCount	スケールアウトするコントローラーノード数
OvercloudControlFlavor	コントローラーノード (control) に使用するフレーバー
ComputeCount	スケールアウトするコンピュートノード数
OvercloudComputeFlavor	コンピュートノード (compute) に使用するフレーバー

たとえば、オーバークラウドが各ロール (Controller および Compute) に 3 つずつノードをデプロイするように設定するには、`parameter_defaults` に以下の設定を追加します。

```
parameter_defaults:
  ControllerCount: 3
  ComputeCount: 3
  OvercloudControlFlavor: control
  OvercloudComputeFlavor: compute
```



注記

Heat テンプレートのパラメーターのより詳細な一覧は、『[director のインストールと使用方法](#)』の「[CLI ツールでのオーバークラウドの作成](#)」を参照してください。

3.8. オーバークラウドの作成

オーバークラウドの作成には、**openstack overcloud deploy** コマンドに追加の引数を指定する必要があります。以下に例を示します。

```
$ openstack overcloud deploy --templates -e -e
/home/stack/templates/puppet-ceph-external.yaml --ntp-server pool.ntp.org
```

上記のコマンドは、以下のオプションを使用します。

- **--templates:** デフォルトの Heat テンプレートコレクション (**/usr/share/openstack-tripleo-heat-templates/**) からオーバークラウドを作成します。
- **-e /home/stack/templates/puppet-ceph-external.yaml:** 別の環境ファイルをオーバークラウドデプロイメントに追加します。この場合は、既存の **Ceph Storage Cluster** の設定を含むストレージ環境ファイルです。
- **--ntp-server pool.ntp.org:** NTP サーバーを設定します。

/home/stack/templates/storage-environment.yaml で使用する全設定の概要については、「[付録A 環境ファイルのサンプル: Ceph クラスターの作成](#)」を参照してください。

ヒント

オプションの完全な一覧を表示するには、以下を実行します。

```
$ openstack help overcloud deploy
```

詳しい情報は、「[director のインストールと使用方法](#)」ガイドの「[オーバークラウドのパラメーター設定](#)」を参照してください。

オーバークラウドの作成プロセスが開始され、**director** によりノードがプロビジョニングされます。このプロセスは完了するまで多少時間がかかります。オーバークラウドの作成のステータスを確認するには、**stack** ユーザーとして別のターミナルを開き、以下を実行します。

```
$ source ~/stackrc
$ heat stack-list --show-nested
```

この設定では、オーバークラウドが外部の **Ceph Storage Cluster** を使用するよう設定します。このクラスターは、オーバークラウドから独立して、管理される点に注意してください。たとえば、**Ceph Storage Cluster** は、**OpenStack Platform director** ではなく **Ceph** 管理ツールを使用してスケーリングします。

3.9. オーバークラウドへのアクセス

director は、**director** ホストがオーバークラウドと対話するための設定と認証を行うスクリプトを生成します。このファイル (**overcloudrc**) は、**stack** ユーザーのホームディレクトリーに保存されます。このファイルを使用するには、以下のコマンドを実行します。

```
$ source ~/overcloudrc
```

これにより、**director** ホストの CLI からオーバークラウドと対話するために必要な環境変数が読み込まれます。**director** ホストとの対話に戻るには、以下のコマンドを実行します。

```
$ source ~/stackrc
```

第4章 まとめ

これで、Red Hat Ceph Storage を使用したオーバークラウドの作成および設定が完了しました。オーバークラウド作成後の一般的な機能については、『**director** インストールと使用方法』の「[オーバークラウドの作成後のタスク実行](#)」を参照してください。

付録A 環境ファイルのサンプル: CEPH クラスターの作成

以下のカスタム環境ファイルのサンプルは、「[2章 Ceph Storage ノードでのオーバークラウドの作成](#)」で説明したオプションの多くを使用しています。このサンプルには、コメントアウトされているオプションは含まれません。環境ファイルの概要については、『[オーバークラウドの高度なカスタマイズ](#)』ガイドの「[環境ファイル](#)」を参照してください。

/home/stack/templates/storage-environment.yaml

```
resource_registry: // ❶
  OS::TripleO::Services::CephMon: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-mon.yaml
  OS::TripleO::Services::CephOSD: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-osd.yaml
  OS::TripleO::Services::CephClient: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-client.yaml
  OS::TripleO::Services::CinderBackup: /usr/share/openstack-tripleo-heat-
templates/puppet/services/pacemaker/cinder-backup.yaml // ❷
  OS::TripleO::Services::CephRgw: /usr/share/openstack-tripleo-heat-
templates/puppet/services/ceph-rgw.yaml // ❸
  OS::TripleO::Services::SwiftProxy: OS::Heat::None
  OS::TripleO::Services::SwiftStorage: OS::Heat::None
  OS::TripleO::Services::SwiftRingBuilder: OS::Heat::None

parameter_defaults: // ❹
  CinderEnableIscsiBackend: false
  CinderEnableRbdBackend: true
  CinderEnableNfsBackend: false
  NovaEnableRbdBackend: true
  GlanceBackend: rbd
  CinderBackupBackend: swift // ❺
  ExtraConfig:
    ceph::profile::params::osds: // ❻
      '/dev/sdc':
        journal: '/dev/sdb'
      '/dev/sdd':
        journal: '/dev/sdb'
  CephPools: // ❼
    volumes:
      size: 5
      pg_num: 128
      pgp_num: 128
  ControllerCount: 3 // ❽
  OvercloudControlFlavor: control
  ComputeCount: 3
  OvercloudComputeFlavor: compute
  CephStorageCount: 3
  OvercloudCephStorageFlavor: ceph-storage
  CephMonCount: 3
  OvercloudCephMonFlavor: ceph-mon
  CephMdsCount: 3
  OvercloudCephMdsFlavor: ceph-mds
  NeutronNetworkType: vxlan // ❾
```

- 1 **resource_registry** セクションは、Heat テンプレートにリンクするリソースを定義します。最初の 3 つのエントリー (**CephMon**、**CephOSD**、**CephClient**) は、**Ceph** クラスタ (MON、OSD、クライアント) の異なるコンポーネントを定義するのに使用する Heat テンプレートをリンクします。
- 2 **OS::TripleO::Services::CinderBackup** のエントリーは、**Block Storage Backup** サービスをデプロイするのに必要な Heat テンプレート呼び出しです。バックアップのターゲットは、後で **parameter_defaults** セクションで設定することができます。
- 3 **OS::TripleO::Services::CephRgw** のエントリーは、**Ceph Object Gateway** のデプロイに必要な Heat テンプレート呼び出し、OpenStack で **Ceph Object Storage** を使用するための手段を提供します。デフォルトの **Object Storage** サービス (**swift**) は必要ないので、以下の 3 行で無効化します。詳しくは、「[Ceph Object Gateway のデプロイ](#)」を参照してください。
- 4 **parameter_defaults** セクションは、全テンプレート内のパラメーターのデフォルト値を変更します。ここに記載のエントリーはすべて「[オーバークラウドでの Ceph Storage の有効化](#)」に記載しています。
- 5 **Ceph Object Gateway** をデプロイするので、**Ceph Object Storage** をバックアップのターゲットとして使用することができます。このターゲットを設定するには、**CinderBackupBackend** を **swift** に設定します。詳しくは、「[Ceph Object Gateway のデプロイ](#)」を参照してください。
- 6 「[Ceph Storage ノードのディスクレイアウトのマッピング](#)」の説明にあるように、**ceph::profile::params::osds::** セクションは、カスタムのディスクレイアウトを定義します。
- 7 **CephPools** セクションは、任意の **Ceph** プールのカスタム属性を設定します。この例では、**volumes** プール用にカスタムの **size**、**pg_num**、**pgp_num** の属性を設定します。詳しくは、「[異なる Ceph プールへのカスタムの属性の割り当て](#)」を参照してください。
- 8 各ロールでは ***Count** パラメーターでノード数を割り当て、**Overcloud*Flavor** パラメーターでフレーバーを割り当てます。たとえば、**ControllerCount: 3** は 3 つのノードを **Controller** ロールに割り当て、**OvercloudControlFlavor: control** は各ロールが **control** フレーバーを使用するように設定します。詳しくは、「[ロールへのノードとフレーバーの割り当て](#)」を参照してください。



注記

CephMonCount、**CephMdsCount**、**OvercloudCephMonFlavor**、**OvercloudCephMdsFlavor** のパラメーターは (**ceph-mon** および **ceph-mds** フレーバーと共に)、「[専用ノード上でのその他の Ceph サービスのデプロイ](#)」に記載のように、カスタムの **CephMON** および **CephMds** ロールを作成した場合にのみ有効となります。

- 9 **NeutronNetworkType: neutron** サービスが使用すべきネットワークの種別 (この場合は **vxlan**) を設定します。

付録B カスタムインターフェーステンプレートの例: 複数のボンディングされたインターフェース

以下のテンプレートは、`/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml` をカスタマイズしたバージョンです。バックエンドとフロントエンドのネットワークトラフィックを分離するための複数のボンディングインターフェースと、両方の接続用の冗長性の機能が含まれています(「[Ceph ノードごとに複数のボンディングされたインターフェースを設定する方法](#)」で説明)。また、カスタムのボンディングオプションも使用します('mode=4 lacp_rate=1'。「[ボンディングモジュールのディレクティブの設定](#)」で説明)。

`/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/ceph-storage.yaml` (カスタム)

```
heat_template_version: 2015-04-30

description: >
  Software Config to drive os-net-config with 2 bonded nics on a bridge
  with VLANs attached for the ceph storage role.

parameters:
  ControlPlaneIp:
    default: ''
    description: IP address/subnet on the ctlplane network
    type: string
  ExternalIpSubnet:
    default: ''
    description: IP address/subnet on the external network
    type: string
  InternalApiIpSubnet:
    default: ''
    description: IP address/subnet on the internal API network
    type: string
  StorageIpSubnet:
    default: ''
    description: IP address/subnet on the storage network
    type: string
  StorageMgmtIpSubnet:
    default: ''
    description: IP address/subnet on the storage mgmt network
    type: string
  TenantIpSubnet:
    default: ''
    description: IP address/subnet on the tenant network
    type: string
  ManagementIpSubnet: # Only populated when including
environments/network-management.yaml
    default: ''
    description: IP address/subnet on the management network
    type: string
  BondInterfaceOvsOptions:
    default: 'mode=4 lacp_rate=1'
    description: The bonding_options string for the bond interface. Set
                  things like lacp=active and/or bond_mode=balance-slb
                  using this option.
```

```

    type: string
    constraints:
      - allowed_pattern: "^(?!balance.tcp).*$"
        description: |
          The balance-tcp bond mode is known to cause packet loss and
          should not be used in BondInterfaceOvsOptions.
  ExternalNetworkVlanID:
    default: 10
    description: Vlan ID for the external network traffic.
    type: number
  InternalApiNetworkVlanID:
    default: 20
    description: Vlan ID for the internal_api network traffic.
    type: number
  StorageNetworkVlanID:
    default: 30
    description: Vlan ID for the storage network traffic.
    type: number
  StorageMgmtNetworkVlanID:
    default: 40
    description: Vlan ID for the storage mgmt network traffic.
    type: number
  TenantNetworkVlanID:
    default: 50
    description: Vlan ID for the tenant network traffic.
    type: number
  ManagementNetworkVlanID:
    default: 60
    description: Vlan ID for the management network traffic.
    type: number
  ControlPlaneSubnetCidr: # Override this via parameter_defaults
    default: '24'
    description: The subnet CIDR of the control plane network.
    type: string
  ControlPlaneDefaultRoute: # Override this via parameter_defaults
    description: The default route of the control plane network.
    type: string
  ExternalInterfaceDefaultRoute: # Not used by default in this template
    default: '10.0.0.1'
    description: The default route of the external network.
    type: string
  ManagementInterfaceDefaultRoute: # Commented out by default in this
template
    default: unset
    description: The default route of the management network.
    type: string
  DnsServers: # Override this via parameter_defaults
    default: []
    description: A list of DNS servers (2 max for some implementations)
that will be added to resolv.conf.
    type: comma_delimited_list
  EC2MetadataIp: # Override this via parameter_defaults
    description: The IP address of the EC2 metadata server.
    type: string

resources:

```

```

OsNetConfigImpl:
  type: OS::Heat::StructuredConfig
  properties:
    group: os-apply-config
    config:
      os_net_config:
        network_config:
          -
            type: interface
            name: nic1
            use_dhcp: false
            dns_servers: {get_param: DnsServers}
            addresses:
              -
                ip_netmask:
                  list_join:
                    - '/'
                    - - {get_param: ControlPlaneIp}
                      - {get_param: ControlPlaneSubnetCidr}
            routes:
              -
                ip_netmask: 169.254.169.254/32
                next_hop: {get_param: EC2MetadataIp}
              -
                default: true
                next_hop: {get_param: ControlPlaneDefaultRoute}
          -
            type: ovs_bridge
            name: br-bond
            members:
              -
                type: linux_bond
                name: bond1
                bonding_options: {get_param: BondInterfaceOvsOptions}
                members:
                  -
                    type: interface
                    name: nic2
                    primary: true
                  -
                    type: interface
                    name: nic3
              -
                type: vlan
                device: bond1
                vlan_id: {get_param: StorageNetworkVlanID}
                addresses:
                  -
                    ip_netmask: {get_param: StorageIpSubnet}
          -
            type: ovs_bridge
            name: br-bond2
            members:
              -
                type: linux_bond
                name: bond2

```

```
        bonding_options: {get_param: BondInterfaceOvsOptions}
        members:
          -
            type: interface
            name: nic4
            primary: true
          -
            type: interface
            name: nic5
        -
          type: vlan
          device: bond1
          vlan_id: {get_param: StorageMgmtNetworkVlanID}
          addresses:
            -
              ip_netmask: {get_param: StorageMgmtIpSubnet}

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