



Red Hat OpenStack Platform 13

ハイパーコンバージドインフラストラクチャーガイド

Red Hat OpenStack Platform オーバークラウドにおけるハイパーコンバージドインフラストラクチャーの設定についての理解

Red Hat OpenStack Platform 13 ハイパーコンバージドインフラストラクチャーガイド

Red Hat OpenStack Platform オーバークラウドにおけるハイパーコンバージドインフラストラクチャーの設定についての理解

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

本書では、Red Hat OpenStack Platform のハイパーコンバージェンスの実装について説明します。この実装では、Compute サービスと Ceph Storage サービスが同じホストに配置されます。

目次

第1章 はじめに	3
1.1. 前提条件	3
1.2. 参考資料	3
第2章 プロセスの説明	5
第3章 ハイパーコンバージドノード向けのオーバークラウドロールの準備	6
3.1. COMPUTEHC1 ロール向けのポート割り当ての設定	6
3.2. 新規フレーバーの作成と割り当て	7
第4章 ハイパーコンバージドノード上におけるリソース分離の設定	9
4.1. COMPUTE 用の CPU とメモリーリソースの確保	9
4.1.1. メモリーまたは CPU 割り当てに計算された設定値のオーバーライド	9
4.2. CEPH のバックフィルおよびリカバリーの操作	10
4.3. CEPH 用のメモリーと CPU リソースの確保	11
第5章 ネットワーク設定の最終処理	12
第6章 デプロイメント	14
付録A 付録	17
A.1. スケーリング	17
A.1.1. スケールアップ	17
A.1.2. スケールダウン	17
A.2. COMPUTE の CPU およびメモリーの計算	17
A.2.1. NovaReservedHostMemory	18
A.2.2. cpu_allocation_ratio	18

第1章 はじめに

Red Hat OpenStack Platform のハイパーコンバージドインフラストラクチャー (HCI) の実装では、Red Hat Ceph Storage をストレージプロバイダーとして使用します。このインフラストラクチャーは、Compute サービスと Ceph Storage サービスを同じノードに配置して、リソースの使用率を最適化するように構成されたハイパーコンバージドノードを特長とします。ハイパーコンバージドノードのみのオーバークラウドまたはハイパーコンバージドノードを通常のコンピュータノードおよび Ceph Storage ノードと混在させたオーバークラウドをデプロイすることが可能です。

本書では、他の機能 (例: ネットワーク機能の仮想化など) との統合が可能な形で、オーバークラウド上に上記のいずれかのタイプの HCI をデプロイする方法について説明します。また、ハイパーコンバージドノード上における Compute サービスと Ceph Storage サービスの両方のパフォーマンスを最適な状態にする方法についても記載しています。

1.1. 前提条件

本書では、HCI の完全なデプロイメントの方法を順を追って説明するのではなく、オーバークラウド上にハイパーコンバージドノードをデプロイするのに必要な設定について記載しています。これにより、HCI をオーバークラウドデプロイメントプランにシームレスに統合することができます。

以下のセクションは、次の条件を前提としています。

1. アンダークラウドがすでにデプロイ済みであること。アンダークラウドのデプロイ方法についての説明は、[『director のインストールと使用方法』](#)を参照してください。
2. お使いの環境で、Compute および Ceph Storage の要件を満たすノードをプロビジョニング可能であること。詳しくは、「[オーバークラウドの要件](#)」([『director のインストールと使用方法』](#))を参照してください。
3. 環境内の全ノードの準備がすでに整っていること。これは、ノードで以下の作業が完了していることを意味します。
 - a. 登録 ([「Registering the Nodes」](#)で説明)
 - b. タグ付け ([「Manually Tagging the Nodes」](#)で説明)

詳しくは、[『Deploying an Overcloud with Containerized Red Hat Ceph』](#)を参照してください。

4. [「Cleaning Ceph Storage Node Disks」](#) ([『Deploying an Overcloud with Containerized Red Hat Ceph』](#)) の説明に従って、Compute サービスと Ceph OSD のサービスに使用する予定のノード上のディスクのクリーニングが済んでいること。
5. [「環境ファイルを使用したオーバークラウドの登録」](#) ([『オーバークラウドの高度なカスタマイズ』](#)) に記載の手順に従ってオーバークラウドノードを登録するための環境ファイルの準備が完了していること。

1.2. 参考資料

本書は、Red Hat OpenStack Platform の既存のドキュメントの補足資料としてご利用いただくために提供しています。関連する概念についての詳細情報は、以下のドキュメントを参照してください。

- [『オーバークラウドの高度なカスタマイズ』](#) ガイド: director を使用して OpenStack の高度な機能を設定する方法について記載しています (例: カスタムロールの使用法)。

- [『director のインストールと使用方法』](#) ガイド: アンダークラウドおよびオーバークラウドのエンドツーエンドのデプロイメント情報を提供します。
- [『Deploying an Overcloud with Containerized Red Hat Ceph』](#) ガイド: Red Hat Ceph Storage をストレージプロバイダーとして使用するオーバークラウドのデプロイ方法について記載しています。
- [『ネットワークガイド』](#) : Red Hat OpenStack Platform のネットワークに関する詳しいガイドです。
- [『Hyper-converged Red Hat OpenStack Platform 10 and Red Hat Ceph Storage 2』](#) : 極めて特殊なハードウェアにおける HCI を特長とする環境のデプロイ方法について説明したリファレンスアーキテクチャーです。

第2章 プロセスの説明

大半の Red Hat OpenStack Platform の機能と同様に、ハイパーコンバージェンスは、director で実装するのが最適です。director を使用することにより、既存の Heat テンプレートや環境ファイルを利用してデプロイメントをオーケストレーション することができます。

また、director のインフラストラクチャーは、独自の Heat テンプレートと環境ファイルを使用できるフレームワークも提供します。これは、既存のテンプレートおよび環境ファイルでは特定のユースケースに対応しない場合に役立ちます。

以下のサブセクションでは、デプロイメントプロセスの各ステップについて簡単に説明します。

ハイパーコンバージドノード向けのオーバークラウドロールの準備

ハイパーコンバージドノードを使用するには、そのノードに **ロール** を定義する必要があります。Red Hat OpenStack Platform は、標準的なオーバークラウドノード向けのデフォルトロール (例: Controller、Compute、Ceph Storage) に加えて、ハイパーコンバージドノード向けに事前定義された **ComputeHCI** というロールを提供します。**ComputeHCI** ロールを使用するには、デプロイメントで使用するその他の全ロールとともに、カスタムの **roles_data.yaml** ファイルを生成する必要があります。

リソース分離の設定

HCI をデプロイする際には、Compute サービスと Ceph Storage サービスが、ハイパーコンバージドノードとして相互に認識する必要があります。そうでない場合には、両サービスは、専用のノード上にあるかのごとくリソースを消費します。そのため、リソースの競合が発生して、パフォーマンスが低下してしまう可能性があります。

ネットワーク設定

ハイパーコンバージドノードを使用する場合には、**StorageMgmtNetwork** ポートを適切な NIC にマッピングする必要があります。このステップでは、環境に必要なその他のネットワーク設定を実装することが可能です。

デプロイメント

HCI のデプロイメントプロセスでは、そのデプロイメントに含めるする環境ファイルを指定する必要があります。これには、**ComputeHCI** ロール向けの新規フレーバーを定義し、そのフレーバーをハイパーコンバージドノードにタグ付けしてから、デプロイメント中に (「[3章ハイパーコンバージドノード向けのオーバークラウドロールの準備](#)」で作成した) カスタムの **roles_data.yaml** ファイルを呼び出します。

第3章 ハイパーコンバージドノード向けのオーバークラウドロールの準備

オーバークラウドは通常、コントローラーノード、コンピューットノード、異なるストレージノード種別など、事前定義されたロールのノードで構成されます。これらのデフォルトの各ロールには、director ノード上にあるコアの Heat テンプレートコレクションで定義されているサービスセットが含まれます。ただし、コアの Heat テンプレートのアーキテクチャーは、以下のような設定を行う手段を提供します。

- カスタムロールの作成
- 各ロールへのサービスの追加と削除

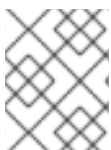
これにより、Compute サービスと Ceph Object Storage Daemon (OSD) サービスの両方で新規ロールを定義することが可能となり、実質的には両サービスが同じ場所に配置され、同じ **ハイパーコンバージドノード** に一緒にデプロイすることができます。

オーバークラウドに使用するロールは、**roles_data.yaml** ファイルで定義されます。director を使用すると、オーバークラウドに使用する全ロールが含まれた、カスタムバージョンのファイルを生成することができます。カスタムバージョンは、「[6章 デプロイメント](#)」の作業中に呼び出すことができます。

Red Hat OpenStack Platform は、ハイパーコンバージドノード専用 **ComputeHCI** という事前定義済みのカスタムロールを提供しています。このロールを使用するには、**ComputeHCI** ロールとオーバークラウドに使用するその他のロールが含まれたカスタムの **roles_data.yaml** ファイルを作成する必要があります。

```
$ openstack overcloud roles generate -o /home/stack/roles_data.yaml
Controller ComputeHCI Compute CephStorage
```

このコマンドにより、カスタムの **roles_data.yaml** ファイルが **/home/stack/roles_data.yaml** に生成されます。このカスタムファイルには、**ComputeHCI** ロールとともに、**Controller**、**Compute**、**CephStorage** のロールが含まれます。オーバークラウドで使用するその他のロールをこのコマンドに追加してください。



注記

カスタムロールに関する詳しい情報は、『[オーバークラウドの高度なカスタマイズ](#)』の「[コンポーザブルサービスとカスタムロール](#)」を参照してください。

3.1. COMPUTEHCI ロール向けのポート割り当ての設定

/usr/share/openstack-tripleo-heat-templates/ にあるデフォルトの Heat テンプレートは、デフォルトロールに必要なネットワーク設定を提供します。この設定には、各ノード上の各サービスに IP アドレスとポートを割り当てる方法が含まれます。

ComputeHCI などのカスタムロールには、必須のポート割り当て用 Heat テンプレートはないので、自分で定義する必要があります。そのためには、**~/templates** に以下の内容を記述した **ports.yaml** という名前の新規テンプレートを作成します。

```
resource_registry:
  OS::TripleO::ComputeHCI::Ports::ExternalPort: /usr/share/openstack-
  tripleo-heat-templates/network/ports/noop.yaml # 1
```

```
OS::Triple0::ComputeHCI::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
OS::Triple0::ComputeHCI::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
OS::Triple0::ComputeHCI::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
OS::Triple0::ComputeHCI::Ports::StorageMgmtPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage_mgmt.yaml # 2
```

- 1** DVR を使用する場合には、この行を以下の設定に置き換えます。

```
OS::Triple0::ComputeHCI::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/external.yaml
```

詳しくは、『[ネットワークガイド](#)』の「[分散仮想ルーター \(DVR\) の設定](#)」を参照してください。

- 2** **ComputeHCI** ロールが IP のプールから選択するようにするには、この行を以下のように置き換えます。

```
OS::Triple0::ComputeHCI::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt_from_pool.yaml
```

環境ファイルで IPv6 アドレスを使用している場合には、この行を以下のように置き換えます。

```
OS::Triple0::ComputeHCI::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt_v6.yaml
```

ComputeHCI ロールが IPv6 アドレスプールから選択するようにするには、以下の設定を使用します。

```
OS::Triple0::ComputeHCI::Ports::StorageMgmtPort:
/usr/share/openstack-tripleo-heat-
templates/network/ports/storage_mgmt_from_pool_v6.yaml
```

その他のストレージ IP およびポートの設定については、**/usr/share/openstack-tripleo-heat-templates/network/ports/** を参考にしてカスタマイズしてください。

関連情報については、「[ネットワークの分離](#)」および「[デプロイするネットワークの選択](#)」(『[オーバークラウドの高度なカスタマイズ](#)』) を参照してください。

3.2. 新規フレーバーの作成と割り当て

「[前提条件](#)」で述べているように、各ノードの登録と、対応するフレーバーとのタグ付けが完了している必要があります。ただし、混合型の HCI のデプロイでは、新しい **ComputeHCI** ロールを定義する必要があります。そのため、そのための新規フレーバーを作成する必要があります。

1. **osdcompute** という名前の新規フレーバーを作成するには、以下のコマンドを実行します。

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

**注記**

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

2. 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、**osdcompute** という名前です。

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

**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。

3. ノードを新しい **osdcompute** プロファイルにタグ付けします。

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

**注記**

ノードのタグ付けに関する詳しい情報は、[「Manually Tagging the Nodes」](#) (『[Deploying an Overcloud with Containerized Red Hat Ceph](#)』) を参照してください。

関連情報については、[「Manually Tagging the Nodes」](#) と [「Assigning Nodes and Flavors to Roles」](#) (『[Deploying an Overcloud with Containerized Red Hat Ceph](#)』) を参照してください。

第4章 ハイパーコンバージドノード上におけるリソース分離の設定

Red Hat OpenStack Platform の HCI 実装では、director は Ceph OSD サービスと Compute サービスを同じ場所に配置してハイパーコンバージドノードを作成します。ただし、この配置をさらに調整しなければ、Ceph サービスと Compute サービスは、同じホスト上でお互いの存在を認識しないため、それらのサービス間で **リソースの競合** が発生するリスクがあります。リソースの競合が発生すると、サービスのパフォーマンスが低下する可能性があり、その場合には、ハイパーコンバージェンスによって提供されるメリットが相殺されてしまいます。

競合が発生しないようにするには、Ceph サービスと Compute サービスの両方にリソースの分離を設定する必要があります。以下のサブセクションでは、その方法について説明します。

4.1. COMPUTE 用の CPU とメモリーリソースの確保

デフォルトでは、Compute サービスのパラメーターは Ceph OSD サービスが同じノード上に配置されていることは考慮に入れません。この問題に対処して、安定を維持し、ホスト可能なインスタンス数を最大化するには、ハイパーコンバージドノードを調整する必要があります。そのためには、ハイパーコンバージドノード上の Compute サービスにリソースの制約を設定する必要があります。これは、**プラン環境ファイル** で設定できます。

プラン環境ファイルは、director が OpenStack Workflow (Mistral) サービスを介して実行可能な **ワークフロー** を定義します。director は、ハイパーコンバージドノード上のリソース制約を設定するための専用のデフォルトのプラン環境ファイルも提供しています。

/usr/share/openstack-tripleo-heat-templates/plan-samples/plan-environment-derived-params.yaml

-p パラメーターを使用して (**openstack overcloud deploy** コマンドに対して)、このプラン環境ファイルをデプロイメント中に呼び出します。このプラン環境ファイルは、OpenStack Workflow に以下の操作を指示します。

1. ハードウェアイントロスペクションデータの取得 (**「ノードのハードウェアの検査」** で収集されるデータ)
2. そのデータに基づいた、ハイパーコンバージドノード上の Compute に最適な CPU とメモリーの制約の算出
3. それらの制約を設定するために必要なパラメーターの自動生成

~/plan-samples/plan-environment-derived-params.yaml プラン環境ファイルは、**hci_profile_config** 下で定義されている CPU およびメモリーの割り当てのワークロード **プロファイル** をいくつか定義します。**hci_profile** パラメーターは、有効化されるワークロードプロファイルを設定します。たとえば、NFV を使用している場合には、**hci_profile: nfv_default** を設定します。

同じ構文を使用して、プラン環境ファイルでカスタムのプロファイルを設定することも可能です。**my_workload** という名前の新規プロファイルを定義する場合の例を以下に示します。

各ワークロードプロファイル内の **average_guest_memory_size_in_mb** および **average_guest_cpu_utilization_percentage** パラメーターは、Compute の **reserved_host_memory** と **cpu_allocation_ratio** の設定値を算出します。これらの値は、Red Hat の推奨値に基づいて計算されます。これは、以前のリリースの手動での計算と同様です。

4.1.1. メモリーまたは CPU 割り当てに計算された設定値のオーバーライド

別の環境ファイルを使用して、OpenStack Workflow によって自動的に定義される Compute の設定をオーバーライドすることができます。これは、**reserved_host_memory** または **cpu_allocation_ratio** のみをオーバーライドして、それ以外は OpenStack Workflow に定義させる場合に役立ちます。以下のスニペットを考慮してください。

```
parameter_defaults:
  ComputeHCIParameters:
    NovaReservedHostMemory: 181000 # 1
  ComputeHCIExtraConfig:
    nova::cpu_allocation_ratio: 8.2 # 2
```

- 1 **NovaReservedHostMemory** パラメーターは、Ceph OSD サービスに確保すべき RAM 容量と、ハイパーコンバージドノード上の 1 ゲストインスタンスあたりのオーバーヘッドを設定します。
- 2 **nova::cpu_allocation_ratio:** パラメーターは、コンピュートノードがインスタンスをデプロイする際に Compute スケジューラーが使用すべき比率を設定します。

ComputeHCIParameters および **ComputeHCIExtraConfig** のフックは、それらのネストされたパラメーターを **ComputeHCI** ロールを使用する全ノード (全ハイパーコンバージドノード) に適用します。**NovaReservedHostMemory** および **nova::cpu_allocation_ratio:** の最適な値を手動で決定する方法に関する詳しい情報は、「[Compute の CPU およびメモリーの計算](#)」を参照してください。

4.2. CEPH のバックフィルおよびリカバリーの操作

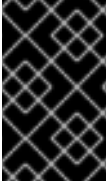
Ceph OSD が削除されると、Ceph は **バックフィル** と **リカバリー** の操作を使用して、クラスターをリバランスします。Ceph は、配置グループポリシーに従って複数のコピーを保管するためにこの操作を実行します。これらの操作は、システムリソースを使用します。Ceph クラスターに負荷がかかっている場合には、リソースがバックフィルとリカバリーに回されるので、パフォーマンスが低下します。

OSD の削除時にこのようなパフォーマンスの影響を軽減するには、バックフィルとリカバリーの操作の優先度を低くすることができます。この方法を使用すると、データのレプリカがより少ない状態が長くなるので、データはより高いリスクにさらされることになります。

バックフィルとリカバリーの操作の優先度を設定するには、以下の内容を記述した **ceph-backfill-recovery.yaml** という名前の環境ファイルを **~/templates** に追加します。

```
parameter_defaults:
  CephAnsibleExtraConfig:
    osd_recovery_op_priority: 3 # 1
    osd_recovery_max_active: 3 # 2
    osd_max_backfills: 1 # 3
```

- 1 **osd_recovery_op_priority** は、OSD クライアント OP の優先度と相対的に、リカバリー操作の優先度を設定します。
- 2 **osd_recovery_max_active** は、1 OSD あたりの 1 回にアクティブなリカバリー要求の件数を設定します。要求がこの値を超えると、リカバリーが加速されますが、それらの要求によりクラスターにかかる負荷が増大します。レイテンシーを低くするには、この値を **1** に設定します。
- 3 **osd_max_backfills** は単一の OSD との間で許容されるバックフィルの最大数を設定します。



重要

サンプルで使用している値は、現在のデフォルト値です。異なる値を使用する予定がなければ、**ceph-backfill-recovery.yaml** デプロイメントに追加する必要はありません。

4.3. CEPH 用のメモリーと CPU リソースの確保

ハイパーコンバージドノードでは、コンテナ化された各 OSD は、`docker run` コマンドの `--memory` と `--cpu-quota` のオプションを使用して、RAM の GB と vCPU を制限する必要があります。 **storage-container-config.yaml** ファイルを編集して、値をオプションに渡すことができます。以下の例は、3 GB の RAM と、1 OSD あたり 1 vCPU を確保します。

```
parameter_defaults:
  CephAnsibleExtraConfig:
    ceph_osd_docker_memory_limit: 3g
    ceph_osd_docker_cpu_limit: 1
```

上記の設定を `/home/stack/templates/storage-container-config.yaml` に保存します。

この例で使用されている値は、平均的なハイパーコンバージドノードに適したデフォルト値です。ハードウェアとワークロードによっては、別の値が適切な場合があります。詳しくは、[『Red Hat Ceph Storage Hardware Guide』](#) を参照してください。

第5章 ネットワーク設定の最終処理

この時点では、HCI ノードでポートを適切に割り当てるのに必要な設定が完了しているはずですが、それらのノードで **StorageMgmtPort** を物理 NIC にマッピングする必要もあります。

1. デフォルトの Heat テンプレートコレクションから、お使いの環境に適した Compute NIC 設定テンプレートを選択します。
 - `/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-vlans/compute.yaml`
 - `/usr/share/openstack-tripleo-heat-templates/network/config/single-nic-linux-bridge-vlans/compute.yaml`
 - `/usr/share/openstack-tripleo-heat-templates/network/config/multiple-nics/compute.yaml`
 - `/usr/share/openstack-tripleo-heat-templates/network/config/bond-with-vlans/compute.yaml`

NIC の設定に関する詳しい情報は、各テンプレートの個々のディレクトリーで **README.md** を参照してください。
2. `~/templates` に **nic-configs** という名前の新規ディレクトリーを作成します。選択したテンプレートを `~/templates/nic-configs/` にコピーして、**compute-hci.yaml** という名前に変更します。
3. 新しい `~/templates/nic-configs/compute-hci.yaml` の **parameters:** セクションには、以下の定義を必ず記述してください。

```
StorageMgmtNetworkVlanID:
  default: 40
  description: Vlan ID for the storage mgmt network traffic.
  type: number
```

この定義が (`.../single-nic-vlans/compute.yaml` 内に) まだ記述されていない場合には、追加してください。

4. 各 HCI ノードで **StorageMgmtNetworkVlanID** を特定の NIC にマッピングします。たとえば、単一の NIC に対してトランク VLAN を選択する場合 (つまり、`.../single-nic-vlans/compute.yaml` をコピーした場合) には、`~/templates/nic-configs/compute.yaml` の **network_config:** セクションに以下のエントリーを追加します。

```
-
  type: vlan
  device: em2
  mtu: 9000 # 1
  use_dhcp: false
  vlan_id: {get_param: StorageMgmtNetworkVlanID}
  addresses:
    -
      ip_netmask: {get_param: StorageMgmtIpSubnet}
```


- 1 NIC を **StorageMgmtNetworkVlanID** にマッピングする際には、**mtu** を **9000** (ジャンボフレーム) に設定することを推奨します。この MTU 設定により、Ceph のパフォーマンスが大幅に向上します。関連情報は、「[director での MTU の設定](#)」(『[ネットワークガイド](#)』) および 「[ジャンボフレームの設定](#)」(『[オーバークラウドの高度なカスタマイズ](#)』) を参照してください。

5. ネットワークの環境ファイル **~/templates/network.yaml** を作成します。このファイルには、以下の内容を記述する必要があります。

```
resource_registry:  
  OS::TripleO::ComputeHCI::Net::SoftwareConfig:  
    /home/stack/templates/nic-configs/compute-hci.yaml
```

このファイルは、後でオーバークラウドのデプロイメント中に (「[6章 デプロイメント](#)」) カスタマイズされた Compute NIC テンプレート (**~/templates/nic-configs/compute-hci.yaml**) を呼び出すのに使用されます。

~/templates/network.yaml を使用してネットワーク関連のパラメーターを定義したり、カスタマイズされたネットワーク用の Heat テンプレートを追加したりすることができます。詳しくは、『[オーバークラウドの高度なカスタマイズ](#)』の「[ネットワーク環境ファイルの作成](#)」を参照してください。

第6章 デプロイメント

この時点で、同じノードに配置されている Compute サービスと Ceph Storage サービスの間のリソースの競合を軽減するのに必要な全設定 (「[4章 ハイパーコンバージドノード上におけるリソース分離の設定](#)」に記載) が完了している必要があります。

開始する前に、以下の点を確認してください。

1. その他すべての Ceph の設定には、別のベース環境ファイルを 1 つ (または複数) 使用していること。いずれのセクションでも、「[Customizing the Storage Service](#)」と「[Sample Environment File: Creating a Ceph Cluster](#)」 (どちらの項も『[Deploying an Overcloud with Containerized Red Hat Ceph](#)』に記載) の `/home/stack/templates/storage-config.yaml` ファイルを使用することを前提とします。
2. 同じ `/home/stack/templates/storage-config.yaml` 環境ファイルは、各ロールに割り当てるノード数も定義します。この設定に関する情報は、「[Assigning Nodes and Flavors to Roles](#)」 (この場合も『[Deploying an Overcloud with Containerized Red Hat Ceph](#)』を参照してください)。

オーバークラウドをデプロイするには、以下のコマンドを実行します。

```
$ openstack overcloud deploy --templates \
  -p /usr/share/openstack-tripleo-heat-templates/plan-samples/plan-
environment-derived-params.yaml \
  -r /home/stack/templates/roles_data.yaml \
  -e /home/stack/templates/ports.yaml
-e /home/stack/templates/environment-rhel-registration.yaml
-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-
ansible/ceph-ansible.yaml \
  -e /home/stack/templates/storage-config.yaml \
  -e /home/stack/templates/storage-container-config.yaml \
  -e /home/stack/templates/network.yaml \
  -e /home/stack/templates/ceph-backfill-recovery.yaml \
  --ntp-server pool.ntp.org
```

ここで、

- **--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。
- **-p /usr/share/openstack-tripleo-heat-templates/plan-samples/plan-environment-derived-params.yaml:** 派生パラメータのワークフローがデプロイメント中に実行されて、ハイパーコンバージドのデプロイメントに確保されるべきメモリーと CPU の値が計算されるように指定します。
- **-r /home/stack/templates/roles_data.yaml:** 「[3章 ハイパーコンバージドノード向けのオーバークラウドロールの準備](#)」で作成した、カスタマイズされたロール定義ファイルを指定します。これには、**ComputeHCI** ロールが含まれます。
- **-e /home/stack/templates/ports.yaml:** 「[ComputeHCI ロール向けのポート割り当ての設定](#)」の環境ファイルを追加します。このファイルは、**ComputeHCI** ロール用のポートを設定します。
- **-e /home/stack/templates/environment-rhel-registration.yaml:** 「[環境ファイルを使用したオーバークラウドの登録](#)」 (『[オーバークラウドの高度なカスタマイズ](#)』) に説明したように、オーバークラウドノードを登録する環境ファイルを追加します。

- **-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible.yaml:** すべてデフォルトの設定で、コンテナ化された Red Hat Ceph クラスタをデプロイするベース環境ファイルを追加します。詳しくは、[『Deploying an Overcloud with Containerized Red Hat Ceph』](#) を参照してください。
- **-e /home/stack/templates/storage-config.yaml:** その他すべての Ceph の設定を定義するカスタムの環境ファイルを追加します。この詳しい例は、[「Sample Environment File: Creating a Ceph Cluster」](#) ([『Deploying an Overcloud with Containerized Red Hat Ceph』](#)) を参照してください。



注記

[「Sample Environment File: Creating a Ceph Cluster」](#) ([『Deploying an Overcloud with Containerized Red Hat Ceph』](#)) では、`/home/stack/templates/storage-config.yaml` ファイルはフレーバーおよび 1 ロールあたりに割り当てるノード数を指定するのにも使用されます。詳しくは、[「Assigning Nodes and Flavors to Roles」](#) を参照してください。

- **/home/stack/templates/storage-container-config.yaml:** 「Section 4.4 Reserving Memory and CPU Resources for Ceph」から、各 Ceph OSD ストレージコンテナ用の CPU とメモリーを確保します。
- **-e /home/stack/templates/network.yaml:** [「5章 ネットワーク設定の最終処理」](#) の環境ファイルを追加します。
- **-e /home/stack/templates/ceph-backfill-recovery.yaml:** [「Ceph のバックフィルおよびリカバリーの操作」](#) の環境ファイルを追加します。
- **--ntp-server pool.ntp.org:** NTP サーバーを設定します。

プランニングしているオーバークラウドのデプロイメントに必要な環境ファイルを追加するには、**-e** フラグを使用します。たとえば、**Single-Root Input/Output Virtualization (SR-IOV)** も有効にするには、それに対応した環境ファイルを追加します。

```
-e /usr/share/openstack-tripleo-heat-templates/environments/neutron-sriov.yaml
```

SR-IOV ネットワークの基本設定を適用するには、それを定義する環境ファイルを追加します。

```
-e /home/stack/templates/network-environment.yaml
```



注記

現在、SR-IOV は HCI でサポートされている唯一の Network Function Virtualization (NFV) 実装です。詳しくは [「仮想ネットワークの SR-IOV サポート」](#) ([『ネットワークガイド』](#)) を参照してください。

デプロイメントオプションの完全な一覧を表示するには、以下のコマンドを実行します。

```
$ openstack help overcloud deploy
```

詳しい情報は、[「CLI ツールを使用したオーバークラウドの作成」](#) ([『director のインストールと使用方法』](#)) を参照してください。

ヒント

応答ファイル を使用してデプロイメントに追加する環境ファイルを指定することも可能です。詳しくは、「[オーバークラウド作成時の環境ファイルの追加](#)」 (『[director のインストールと使用方法](#)』) を参照してください。

付録A 付録

A.1. スケーリング

HCI ノードをスケールアップまたはスケールダウンするには、Compute または Ceph Storage ノードのスケールリングと同じ原則 (および大半はメソッド) が適用されます。以下の点に注意してください。

A.1.1. スケールアップ

単一型の HCI 環境内で (全コンピュータノードがハイパーコンバージドノードである場合) HCI ノードをスケールアップするには、コンピュータノードのスケールアップと同じ方法を使用します。詳しくは、「[ノードのさらなる追加](#)」 (『[director のインストールと使用方法](#)』) を参照してください。

混合型の HCI 環境の HCI ノードのスケールアップにも同じ方法が該当します (オーバークラウドにハイパーコンバージドと通常のコンピュータノードの両方の機能がある場合)。新規ノードをタグ付けする場合には、適切なフレーバー (この場合には **osdcompute**) を使用することを念頭にいらしてください。「[新規フレーバーの作成と割り当て](#)」を参照してください。

A.1.2. スケールダウン

HCI ノードのスケールダウンプロセス (単一型と混合型の両方の HCI 環境) を簡単に説明すると以下のようになります。

1. HCI ノード上の Ceph OSD サービスを無効化して、リバランスします。director は、HCI ノードまたは Ceph Storage ノードの削除時に Red Hat Ceph Storage クラスターを自動でリバランスしないので、このステップが必要となります。
「[Scaling Down and Replacing Ceph Storage Nodes](#)」 (『[Deploying an Overcloud with Containerized Red Hat Ceph](#)』) を参照してください。このガイドに記載のノードの削除の手順には従わないでください。ノード上のインスタンスを移行して、Compute サービスを最初に無効化する必要があります。
2. HCI ノードからインスタンスを移行します。手順については「[コンピュータノードからのインスタンスの移行](#)」を参照してください。
3. ノード上の Compute サービスを無効にして、そのサービスが新規インスタンス起動に使用されるのを防ぎます。
4. オーバークラウドからノードを削除します。

3 番目と 4 番目のステップ (Compute サービスの無効化とノードの削除) については、『[director のインストールと使用方法](#)』の「[コンピュータノードの削除](#)」を参照してください。

A.2. COMPUTE の CPU およびメモリーの計算

本リリースでは、OpenStack Workflow を使用して、ハイパーコンバージドノードに適した CPU とメモリーの割り当て設定値を自動的に設定することができます。ただし、一部のインスタンスでは、OpenStack Workflow による設定は CPU またはメモリーのいずれかに限定して、それ以外は自分で設定することができます。そのためには、通常のとおりオーバーライドすることができます (「[メモリーまたは CPU 割り当てに計算された設定値のオーバーライド](#)」に説明を記載)。

以下のスクリプトを使用すると、ハイパーコンバージドノードに適した **NovaReservedHostMemory** および **cpu_allocation_ratio** の基準値を算出することができます。

[nova_mem_cpu_calc.py](#)

以下のサブセクションでは、両設定について詳しく説明します。

A.2.1. NovaReservedHostMemory

NovaReservedHostMemory パラメーターは、ホストノードに確保するメモリー容量 (MB 単位) を設定します。ハイパーコンバージドノードに適切な値を決定するには、各 OSD が 3 GB のメモリーを消費すると仮定します。メモリーが 256 GB で OSD が 10 の場合には、Ceph に 30 GB のメモリーを割り当てて、Compute に 226 GB 残します。このメモリー容量があるノードでは、たとえば、2 GB のメモリーを使用するインスタンスを 113 ホストすることができます。

ただし、**ハイパーバイザー** には、1 インスタンスあたりの追加のオーバーヘッドを考慮する必要があります。このオーバーヘッドが 0.5 GB と仮定すると、同じノードでは、90 インスタンスしかホストできません。この値は、226 GB を 2.5 GB で除算して割り出します。ホストノードに確保するメモリー容量 (Compute サービスが使用してはならないメモリー) は以下のように算出します。

$$(\text{In} * \text{Ov}) + (\text{Os} * \text{RA})$$

ここで、

- **In** はインスタンス数に置き換えます。
- **Ov** は 1 インスタンスあたりに必要なオーバーヘッドメモリーの容量に置き換えます。
- **Os** はノード上の OSD 数に置き換えます。
- **RA** は、各 OSD に割り当てる必要のある RAM 容量に置き換えます。

90 インスタンスの場合には、 $(90 * 0.5) + (10 * 3) = 75 \text{ GB}$ という計算になります。Compute サービスには、この値を MB 単位で指定します (75000)。

以下の Python コードは、この計算を実装します。

```
left_over_mem = mem - (GB_per_OSD * osds)
number_of_guests = int(left_over_mem /
    (average_guest_size + GB_overhead_per_guest))
nova_reserved_mem_MB = MB_per_GB * (
    (GB_per_OSD * osds) +
    (number_of_guests * GB_overhead_per_guest))
```

A.2.2. cpu_allocation_ratio

Compute スケジューラーは、インスタンスをデプロイする Compute ノードを選択する際に **cpu_allocation_ratio** を使用します。デフォルトでは、この値は **16.0** (16:1) です。1 台のノードに 56 コアある場合には、Compute スケジューラーは 1 台のノードで 896 の仮想 CPU を使用するのに十分なインスタンスをスケジューリングすることになります。この値を超えると、そのノードはそれ以上インスタンスをホストできないと見なされます。

ハイパーコンバージドノードに適切な **cpu_allocation_ratio** を決定するには、各 Ceph OSD が最小で 1 コアを使用すると仮定します (ワークロードが I/O 集中型で、SSD を使用しないノード上にある場合を除く)。56 コア、10 OSD のノードでは、この計算で 46 コアが Compute に確保されます。各インスタンスが割り当てられた CPU の 100 パーセント使用すると、この比率は単にインスタンスの仮想 CPU 数をコア数で除算した値となります ($46 / 56 = 0.8$)。ただし、インスタンスは通常割り当てられた CPU を 100 パーセント使用することはないため、ゲストに必要な仮想 CPU 数を決定する際には、予想される使用率を考慮に入れて、**cpu_allocation_ratio** を高くすることができます。

したがって、インスタンスが仮想 CPU の 10 パーセント (または 0.1) のみを使用すると予想できる場合には、インスタンス用の仮想 CPU は $46 / 0.1 = 460$ の式で示すことができます。この値をコア数 (56) で除算すると、比率は約 8 に増えます。

以下の Python コードは、この計算を実装します。

```
cores_per_OSD = 1.0
average_guest_util = 0.1 # 10%
nonceph_cores = cores - (cores_per_OSD * osds)
guest_vCPUs = nonceph_cores / average_guest_util
cpu_allocation_ratio = guest_vCPUs / cores
```

ヒント

[nova_mem_cpu_calc.py](#) のスクリプトを使用して、**reserved_host_memory** と **cpu_allocation_ratio** の基準値を算出することができます。詳しくは、「[Compute の CPU およびメモリーの計算](#)」を参照してください。