



Red Hat OpenStack Platform 11

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 11 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。Red Hat OpenStack Platform は単一型と混合型のハイパーコンバージドインフラストラクチャーを両方サポートしています。本書には、オーバークラウドでこれらのいずれかを有効化する方法についても記載しています。

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 デプロイメントタイプの選択	6
3.1. 単一型 HCI	6
3.2. 混合型 HCI	6
3.2.1. HCI ノード向けのカスタムロールの作成	7
3.2.2. カスタムロール向けのポート割り当ての設定	8
3.2.3. 新規フレーバーの作成と割り当て	9
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	11
4.1. COMPUTE 用の CPU とメモリーリソースの確保	11
4.2. CEPH NUMA ピニングの設定	13
4.3. CEPH のバックフィルおよびリカバリーの操作	14
第5章 ネットワーク設定の最終処理	16
第6章 デプロイメント	18
6.1. 単一型 HCI のデプロイ	18
6.2. 混合型 HCI のデプロイ	20
付録A 付録	23
A.1. スケーリング	23
A.1.1. スケールアップ	23
A.1.2. スケールダウン	23
A.2. アップストリームのツール	23
A.2.1. Compute の CPU およびメモリーの計算	24
A.2.2. Ceph OSD サービス向けの NUMA ピニングを設定するカスタムスクリプト	25

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。HCI は以下のいずれかをサポートしています。

- 単一型 HCI: オーバークラウドが必要とするコンピュートがすべてハイパーコンバージドノードによって提供される構成
- 混合型 HCI: ハイパーコンバージドノードと通常のコンピュートノードが混在する構成

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 ([「ノードの登録」](#)で説明)
 - b. タグ付け ([「ノードの手動でのタグ付け」](#)で説明)

詳しくは、[『オーバークラウド向けの Red Hat Ceph Storage』](#)を参照してください。

4. [「Ceph Storage ノードのディスクのクリーニング」](#) ([『オーバークラウド向けの Red Hat Ceph Storage』](#)) 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。「Ceph NUMA ピニングの設定」の手順には、director によって提供されないパッケージのインストールを必要とするスクリプトが含まれています。

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』](#) : 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方法)。
- [『director のインストールと使用方法』](#) : 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。
- [『オーバークラウド向けの Red Hat Ceph Storage』](#) : 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。特に、director には、ハイパーコンバージェンスのサポート対象バージョンを実装するためのデフォルトの環境ファイルも含まれています。

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。

デプロイメントタイプの選択

Red Hat OpenStack Platform では、**単一型** と **混合型** の 2 つのタイプの HCI をデプロイすることができます。いずれのタイプでも、Red Hat OpenStack Platform サービスの主要なストレージプロバイダーとして Red Hat Ceph Storage がデプロイされます。

単一型の HCI では、全コンピュートノードに Ceph-OSD サービスを配置してデプロイされます。

混合型の HCI では、通常のコンピュートノードと、Ceph-OSD サービスを同じ場所に配置したコンピュートノードを並行してデプロイすることができます。

HCI のタイプによって実装はかなり異なるので、デプロイする HCI のタイプを決定してから作業を開始してください。

リソース分離の設定

HCI をデプロイする際には、director はハイパーコンバージドノード上の Compute サービスと Ceph Storage サービスが 相互認識するようには設定しません。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。このような問題は、手動で緩和する必要があります。

ネットワーク設定

単一型と混合型のいずれの HCI デプロイメントでも、**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には (単一型、混合型を問わず)、デプロイメントに追加する環境ファイルを指定するステップが含まれます。単一型の HCI では、Compute ロールに実装されるサービスを設定する既存の環境ファイルを指定する必要があります。混合型の HCI では、新規フレーバーを定義してハイパーコンバージドノードにタグ付けし、デプロイメント中にカスタムロールを呼び出す必要があります。

第3章 デプロイメントタイプの選択

単一型の HCI のデプロイでは、Ceph-OSD サービスをコンピュータノードに追加する環境ファイルを呼び出すので、より単純です。混合型の HCI では、HCI ノード向けのカスタムロール、フレーバー、およびネットワーク設定を定義する必要があります。両 HCI タイプは、対照的なデプロイメント方法を提供するので、どちらのタイプが環境により適しているかを判断してください。

以下のサブセクションでは、各デプロイメントタイプの方法を順を追って説明します。いずれのデプロイメントタイプを使用する場合でも、カスタムの環境ファイルと Heat テンプレートを保管するための `/home/stack/templates/` ディレクトリーを作成してください。

3.1. 単一型 HCI

単一型の HCI 構成で Red Hat OpenStack をデプロイするには、以下の環境ファイルを使用します。

`/usr/share/openstack-tripleo-heat-templates/environments/hyperconverged-ceph.yaml`

この環境ファイルは、**CephOSD** サービスを追加して Compute ロールを再定義します。デフォルトでは、**hyperconverged-ceph.yaml** 環境ファイルは分離された **StorageMgmt** ネットワークを使用していることを前提とし、それに応じて Compute のポートを設定します。

分離された **StorageMgmt** ネットワークを使用していない場合は、**StorageMgmtPort** サービスを無効にします。そのためには、以下の手順を実行してください。

1. `~/templates/` に、以下の内容を記述した **hyperconverged-non-isolated.yaml** という名前の新しい環境ファイルを作成します。

```
resource_registry:
    OS::TripleO::Compute::Ports::StorageMgmtPort: OS::Heat::None
```

2. デプロイメントのプロセス中に (「[単一型 HCI のデプロイ](#)」)、**openstack overcloud deploy** を実行して `~/templates/hyperconverged-non-isolated.yaml` を呼び出します。この環境ファイルは、他の設定を正しく上書きするように、最後に呼び出す必要があります。

「[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](#)」に進み、Compute サービスと Ceph Storage サービスの間でのリソースの競合を軽減するための手順に従ってください。

3.2. 混合型 HCI

オーバークラウドは通常、コントローラーノード、コンピュータノード、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 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。



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。

3.2.1. HCI ノード向けのカスタムロールの作成

アンダークラウド上では、以下のファイルでデフォルトのロールが定義されます。

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

自分で作成したカスタムテンプレート用ディレクトリー (`~/templates/`) にこのファイルをコピーします。

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

Compute と Ceph OSD を同じノードに配置する新規ロールを定義するには、**Compute** と **CephStorage** のエントリーの両方を組み合わせたロールを作成します。これは、**OsdCompute** という名前の新規ロールを `~/templates/roles_data_hci.yaml` に追加して、**Compute** ロールサービスを **OsdCompute** にコピーし、Ceph OSD サービスを追加することによって作成します。

```
- name: OsdCompute # 1
  CountDefault: 1 # 2
  disable_upgrade_deployment: True
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
    - OS::TripleO::Services::CephOSD # 3
    - OS::TripleO::Services::Timezone
[...]
```

1 ロールには一意の名前を付ける必要があります。ここでは、**OsdCompute** と定義しています。

2 **CountDefault**: パラメーターは、director がそのロールをデフォルトで適用すべきノードの数を定義します (この場合は 1)。

3 **OS::TripleO::Services::CephOSD** のエントリーは、**Compute** ロール上には存在しない唯一の **CephStorage** ロールです。

Red Hat OpenStack Platform は、ハイパーコンバージドと非ハイパーコンバージドのコンピュータノードの両方の機能を備えたデプロイメントをサポートしています。非ハイパーコンバージドのコンピュータノードをデプロイしない場合には、**Compute** ロールの **CountDefault**: パラメーターを **0** に設定します。

```
- name: Compute
  CountDefault: 0
  disable_upgrade_deployment: True
  ServicesDefault:
    - OS::TripleO::Services::CACerts
    - OS::TripleO::Services::CephClient
    - OS::TripleO::Services::CephExternal
```

```
- OS::TripleO::Services::Timezone
- OS::TripleO::Services::Ntp
[...]
```

「CountDefault:」パラメーターを使用して、各ロールに割り当てるノードの数を定義することができます。ただし、この定義は、本書の後半の「[混合型 HCI のデプロイ](#)」で説明しているように、別の Heat テンプレートで設定することを推奨します。

3.2.2. カスタム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。

カスタムロール（「[HCI ノード向けのカスタムロールの作成](#)」に記載した **OsdCompute** など）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::OsdCompute::Ports::ExternalPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/noop.yaml # 1
  OS::TripleO::OsdCompute::Ports::InternalApiPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/internal_api.yaml
  OS::TripleO::OsdCompute::Ports::StoragePort: /usr/share/openstack-
tripleo-heat-templates/network/ports/storage.yaml
  OS::TripleO::OsdCompute::Ports::TenantPort: /usr/share/openstack-
tripleo-heat-templates/network/ports/tenant.yaml
  OS::TripleO::OsdCompute::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::TripleO::OsdCompute::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 If you want the **OsdCompute** role to select from a pool of IPs, replace this line with:

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

If your environment uses IPv6 addresses, replace this line with:

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

If you want the **OsdCompute** role to select from a pool of IPv6 addresses, use:

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

For any other storage IP and port settings, review the other templates in `/usr/share/openstack-tripleo-heat-templates/network/ports/` for hints on customization.

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。

3.2.3. 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 OsdCompute ロールを定義する必要があるため、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** という名前の新規ロールを作成するには、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、「[ノードの手動でのタグ付け](#)」(『[オーバークラウド向けの Red Hat Ceph Storage](#)』)を参照してください。

関連情報については、「[ノードの手動でのタグ付け](#)」および「[ロールへのノードとフレーバーの割り当て](#)」(『[オーバークラウド向けの Red Hat Ceph Storage](#)』)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

デフォルトの `hyperconverged-ceph.yaml` ファイル (「[単一型 HCI](#)」) またはカスタムの `OsdsCompute` ロール (「[混合型 HCI](#)」) のいずれを使用する場合も、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。本項の計算はを最適なベースラインの指針として使用してから、設定を変更し、決定論とインスタンスのホスティングキャパシティの間の許容可能なトレードオフを特定してください。本書に記載する例では、決定論とアップタイムが優先されます。

以下の Heat テンプレートパラメーターは、Compute サービスがメモリーと CPU リソースをノードで消費する方法を制御します。

`reserved_host_memory`

これは、ホストノードに確保するメモリー容量 (MB 単位) です。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))
```

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この値を使用します。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ル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

「[Compute の CPU およびメモリーの計算](#)」のスクリプトを使用して、`reserved_host_memory` および `cpu_allocation_ratio` の両方のベースライン値を計算することもできます。

使用する値を計算した後は、HCI ノードのデフォルト値として追加します。そのためには、`~/templates` に `compute.yaml` という名前の新しい環境ファイルを作成して、`reserved_host_memory` と `cpu_allocation_ratio` の値を記述します。単一型の HCI デプロイメントの場合には、以下のように記載する必要があります。

```
parameter_defaults:
  NovaComputeExtraConfig: # 1
    nova::compute::reserved_host_memory: 181000
    nova::cpu_allocation_ratio: 8.2
```

- 1 **NovaComputeExtraConfig** の行は、ネストされている全パラメーターをすべての **Compute** ロールに適用します。単一型 HCI デプロイメントの場合には、全コンピュートノードにハイパーコンバージドの設定も適用します。

混合型の HCI の場合には、`~/templates/compute.yaml` に以下の内容を記載する必要があります。


```
parameter_defaults:
```

```
  OsdComputeExtraConfig: # 1
    nova::compute::reserved_host_memory: 181000
    nova::cpu_allocation_ratio: 8.2
```

- 1 **OsdComputeExtraConfig** の行は、ネストされている全設定をカスタムの **OsdCompute** ロールに適用するカスタムリソースです。これは、「[混合型 HCI](#)」で定義しています。

4.2. CEPH NUMA ピニングの設定

NUMA 機能を実装したホストにハイパーコンバージドロールを適用する際には、利用可能な NUMA ソケットの 1 つに Ceph OSD サービスをピンングすることにより、決定論を改善することができます。この設定を行う場合は、ネットワーク IRQ とストレージコントローラーを備えたソケットに Ceph Storage サービスをピンングしてください。これは、Ceph OSD がネットワーク I/O を大量に使用する問題の対処に役立ちます。

これは、ネットワークインターフェースを引数として取り、必要な NUMA 関連の設定をそのインターフェースに適用するシェルスクリプトでオーケストレーションすることができます。このネットワークインターフェースは、Ceph OSD が使用するインターフェースと推定できます。このスクリプト (**numa-systemd-osd.sh**) を **~/templates** に作成してください。

重要

numa-systemd-osd.sh の内容については、より詳細な記述を記載している「[Ceph OSD サービス向けの NUMA ピニングを設定するカスタムスクリプト](#)」を参照してください。

numa-systemd-osd.sh のスクリプトは、NUMA 設定ツールのインストールも試みます。そのため、「[ノードの登録](#)」(『[オーバークラウド向けの Red Hat Ceph Storage](#)』) の説明に従って、オーバークラウドノードを Red Hat に登録しておく必要もあります。

オーバークラウドでこのスクリプトを実行するには、まず最初に **~/templates** に以下の内容を記述した **ceph-numa-pinning-template.yaml** という名前の Heat テンプレートを作成します。

```
heat_template_version: 2014-10-16

parameters:
  servers:
    type: json

resources:
  ExtraConfig:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      inputs:
        - name: OSD_NUMA_INTERFACE
          config: {get_file: numa-systemd-osd.sh} # 1

  ExtraDeployments:
    type: OS::Heat::SoftwareDeployments
    properties:
```

```
servers: {get_param: servers}
config: {get_resource: ExtraConfig}
input_values:
  OSD_NUMA_INTERFACE: 'em2' # 2
actions: ['CREATE'] # 3
```

- 1 **get_file** 関数は、`~/templates/numa-systemd-osd.sh` を呼び出します。このスクリプトは、ネットワークインターフェースを入力として取って (この場合は **OSD_NUMA_INTERFACE**)、そのインターフェースに必要な NUMA 関連の設定を実行することができます。このスクリプトの内容と、それがどう機能するかについての詳しい説明は、「[Ceph OSD サービス向けの NUMA ピニングを設定するカスタムスクリプト](#)」を参照してください。



重要

単一型の HCI デプロイメントでは、`~/templates/numa-systemd-osd.sh` スクリプトの最上位の **IF** ステートメントを編集する必要があります。詳しくは、「[Ceph OSD サービス向けの NUMA ピニングを設定するカスタムスクリプト](#)」を参照してください。

- 2 **OSD_NUMA_INTERFACE** の変数は、Ceph OSD サービスが使用すべきネットワークインターフェース (この例では **em2**) を指定します。`~/templates/numa-systemd-osd.sh` スクリプトは、このインターフェースに必要な NUMA 設定を適用します。
- 3 **actions** には **CREATE** のみを指定するので、スクリプトは初回のデプロイメント時にのみ実行され、更新時には実行されません。

OSD_NUMA_INTERFACE に使用するインターフェースは、**StorageNetwork** 変数または **StorageMgmtNetwork** 変数のいずれかを使用して、全デプロイメントに対して決定することができます。読み取りを大量に行うワークロードには、**StorageNetwork** インターフェースを使用し、書き込みを大量に行うワークロードには、**StorageMgmtNetwork** のインターフェースを使用するとメリットがもたらされます。

Ceph OSD サービスが仮想ネットワークインターフェース (例: ボンディング) を使用する場合には、ボンディング自体ではなく、ボンディングを構成するネットワークデバイスの名前を使用します。たとえば、**bond1** が **em2** と **em4** を使用する場合には、**OSD_NUMA_INTERFACE** を (**bond1** ではなく) **em2** または **em4** に設定します。ボンディングが、同じ NUMA ノードにはない NIC を組み合わせている場合には (**lstopo-no-graphics** で確認)、`numa-systemd-osd.sh` を使用しないでください。

ceph-numa-pinning-template.yaml テンプレートを作成した後は、`~/templates` に以下の内容を記載した **ceph-numa-pinning.yaml** という名前のテンプレートを作成します。

```
resource_registry:
  OS::Triple0::NodeExtraConfigPost: /home/stack/templates/ceph-numa-
    pinning-template.yaml
```

この環境ファイルにより、後ほど、「[6章 デプロイメント](#)」で **ceph-numa-pinning-template.yaml** テンプレートを読み出すことができます。

4.3. 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するため、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:
  ExtraConfig:
    ceph::profile::params::osd_recovery_op_priority: 3 # ①
    ceph::profile::params::osd_recovery_max_active: 3 # ②
    ceph::profile::params::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

このサンプルで使用される値は、現在のデフォルト値です。デフォルトとは異なる値を使用する予定でなければ、**ceph-backfill-recovery.yaml** をデプロイメントに追加する必要はありません。

第5章 ネットワーク設定の最終処理

この時点で、単一型または混合型の HCI デプロイメントのいずれの場合も、HCI ノード上でポートを正しく割り当てするのに必要な設定が完了している必要があります。この設定については、「[単一型 HCI](#)」または「[カスタムロール向けのポート割り当ての設定](#)」で説明しています。

ただし、単一型および混合型のデプロイメントのいずれの場合にも、**StorageMgmtPort** を物理 NIC にマッピングする必要があります。この操作は、以下の手順に従って実行します。

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を作成してから、選択したテンプレートを `~/templates/nic-configs/` にコピーします。
3. 新しい `~/templates/nic-configs/compute.yaml` テンプレートの **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::Compute::Net::SoftwareConfig:  
    /home/stack/templates/nic-configs/compute.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.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。

以下のサブセクションは、単一型または混合型の HCI のデプロイプロセスに対応しています。いずれも、次の条件を前提としています。

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。いずれの項でも、「[オーバークラウドの作成](#)」および「[環境ファイルのサンプル: Ceph クラスターの作成](#)」 (『[オーバークラウド向けの Red Hat Ceph Storage](#)』) に記載した、同じ `/home/stack/templates/storage-environment.yaml` ファイルを使用していることを前提とします。
2. 同じ `/home/stack/templates/storage-environment.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、「[ノードの手動でのタグ付け](#)」 (この場合も『[オーバークラウド向けの Red Hat Ceph Storage](#)』) を参照してください。

デプロイする環境に対応する項に進みます。

- [「単一型 HCI のデプロイ」](#)
- [「混合型 HCI のデプロイ」](#)

6.1. 単一型 HCI のデプロイ

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。たとえば、分離されていない `StorageMgmt` ネットワークを使用してハイパーコンバージドのコンピューティングノードをデプロイすると仮定します。

```
$ openstack overcloud deploy --templates \
  -e /home/stack/templates/environment-rhel-registration.yaml \
  -e /home/stack/templates/storage-environment.yaml \
  -e /usr/share/openstack-tripleo-heat-
templates/environments/hyperconverged-ceph.yaml \
  -e /home/stack/templates/compute.yaml \
  -e /home/stack/templates/network.yaml \
  -e /home/stack/templates/ceph-numa-pinning.yaml \
  -e /home/stack/templates/ceph-backfill-recovery.yaml \
  -e /home/stack/templates/hyperconverged-non-isolated.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。
- `-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](#)」 (from 『[オーバークラウドの高度なカスタマイズ](#)』) に記載しているように、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。「[Ceph NUMA ピニングの設定](#)」の手順には、director が提供しないパッケージのインストールを必要とするスクリプトが含まれています。
- `-e /home/stack/templates/storage-environment.yaml`: その他すべての Ceph の設定を定義す

るベースの環境ファイルを追加します。そのようなファイルの詳しい例は、「Creating an Overcloud with Ceph Storage Nodes」と「Sample Environment File: Creating a Ceph Cluster」(『Red Hat Ceph Storage for the Overcloud』)を参照してください。



注記

「環境ファイルのサンプル: Ceph クラスターの作成」(『オーバークラウド向けの Red Hat Ceph Storage』)では、`/home/stack/templates/storage-environment.yaml` ファイルはフレーバーとロールごと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、「[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。

- **-e /usr/share/openstack-tripleo-heat-templates/environments/hyperconverged-ceph.yaml:** 全コンピュータノードに Ceph サービスを配置してハイパーコンバージド化する環境ファイルを追加します。カスタマイズしたバージョンを別の場所に作成した場合には、パスを適宜に更新してください (例: 分離されていない **StorageMgmt** ネットワークの場合は `~/templates/hyperconverged-ceph.yaml` など)。詳しくは、「[単一型 HCI](#)」を参照してください。
- **-e /home/stack/templates/compute.yaml:** 「[Compute 用の CPU とメモリーリソースの確保](#)」の環境ファイルを追加します。
- **-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-numa-pinning.yaml:** 「[Ceph NUMA ピニングの設定](#)」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。
- **-e /home/stack/templates/hyperconverged-non-isolated.yaml:** 「[単一型 HCI](#)」の環境ファイルを追加します。このファイルは、**StorageMgmtPort** サービスを無効にします。分離されていない **StorageMgmt** ネットワークを使用している場合には、このファイルは必須です。この環境ファイルは、`/usr/share/openstack-tripleo-heat-templates/environments/hyperconverged-ceph.yaml` の設定を上書きするように、最後に呼び出されます。
- **--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、「Configure SR-IOV Support for Virtual Networking」(『Network Functions Virtualization Configuration Guide』) 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。

6.2. 混合型 HCI のデプロイ

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_hci.yaml \
  -e /home/stack/templates/ports.yaml \
  -e /home/stack/templates/environment-rhel-registration.yaml \
  -e /home/stack/templates/storage-environment.yaml \
  -e /home/stack/templates/compute.yaml \
  -e /home/stack/templates/network.yaml \
  -e /home/stack/templates/ceph-numa-pinning.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。
- **-r /home/stack/templates/roles_data_hci.yaml:** 「HCI ノード向けのカスタムロールの作成」のカスタマイズされたロールの定義ファイルを指定します。このファイルは、カスタムの **OsdCompute** ロールを追加します。
- **-e /home/stack/templates/ports.yaml:** 「カスタム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、**OsdCompute** 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」(from 『オーバークラウドの高度なカスタマイズ』) に記載しているように、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。「Ceph NUMA ピニングの設定」の手順には、director が提供しないパッケージのインストールを必要とするスクリプトが含まれています。
- **-e /home/stack/templates/storage-environment.yaml:** その他すべての Ceph の設定を定義す

るベースの環境ファイルを追加します。そのようなファイルの詳しい例は、「Creating an Overcloud with Ceph Storage Nodes」と「Sample Environment File: Creating a Ceph Cluster」(『Red Hat Ceph Storage for the Overcloud』)を参照してください。



注記

「環境ファイルのサンプル: Ceph クラスターの作成」(『オーバークラウド向けの Red Hat Ceph Storage』)では、`/home/stack/templates/storage-environment.yaml` ファイルはフレーバーとロールごと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、「[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。

- `-e /home/stack/templates/compute.yaml`: 「[Compute 用の CPU とメモリーリソースの確保](#)」の環境ファイルを追加します。
- `-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-numa-pinning.yaml`: 「[Ceph NUMA ピニングの設定](#)」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、「[Configure SR-IOV Support for Virtual Networking](#)」(『Network Functions Virtualization Configuration Guide』)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 HCI ノードをスケールアップするには、コンピュート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。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。
「[Ceph Storage ノードのスケールダウンと置き換え](#)」(『[オーバークラウド向けの Red Hat Ceph Storage](#)』)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『[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「[コンピュートノードの削除](#)」を参照してください。

A.2. アップストリームのツール

OpenStack におけるハイパーコンバージェンスに関連した Heat テンプレート、環境ファイル、スクリプト、およびその他のリソースは、以下のアップストリームの Github リポジトリから提供されています。

<https://github.com/RHsyseng/hci/tree/master/custom-templates>

このリポジトリは、「[Compute の CPU およびメモリの計算](#)」および「[Ceph OSD サービス向けの NUMA ピニングを設定するカスタムスクリプト](#)」のスクリプトに対応しています。

これらのスクリプトを使用するには、お使いのアンダークラウドにこのリポジトリを直接クローンします。

```
$ git clone https://github.com/RHsyseng/hci
```

A.2.1. Compute の CPU およびメモリーの計算

「[Compute 用の CPU とメモリーリソースの確保](#)」の項では、ハイパーコンバージドノード上の `reserved_host_memory_mb` および `cpu_allocation_ratio` の適切な値を決定する方法について説明しています。以下の Python スクリプト (このスクリプトも「[アップストリームのツール](#)」のリポジトリから提供されています) を使用して計算することも可能です。

```
#!/usr/bin/env python
# Filename:                nova_mem_cpu_calc.py
# Supported Language(s):   Python 2.7.x
# Time-stamp:              <2017-03-10 20:31:18 jfulton>
# -----
# This program was originally written by Ben England
# -----
# Calculates cpu_allocation_ratio and reserved_host_memory
# for nova.conf based on on the following inputs:
#
# input command line parameters:
# 1 - total host RAM in GB
# 2 - total host cores
# 3 - Ceph OSDs per server
# 4 - average guest size in GB
# 5 - average guest CPU utilization (0.0 to 1.0)
#
# It assumes that we want to allow 3 GB per OSD
# (based on prior Ceph Hammer testing)
# and that we want to allow an extra 1/2 GB per Nova (KVM guest)
# based on test observations that KVM guests' virtual memory footprint
# was actually significantly bigger than the declared guest memory size
# This is more of a factor for small guests than for large guests.
# -----
import sys
from sys import argv

NOTOK = 1 # process exit status signifying failure
MB_per_GB = 1000

GB_per_OSD = 3
GB_overhead_per_guest = 0.5 # based on measurement in test environment
cores_per_OSD = 1.0 # may be a little low in I/O intensive workloads

def usage(msg):
    print msg
    print(
        ("Usage: %s Total-host-RAM-GB Total-host-cores OSDs-per-server " +
         "Avg-guest-size-GB Avg-guest-CPU-util") % sys.argv[0])
    sys.exit(NOTOK)

if len(argv) < 5: usage("Too few command line params")
try:
    mem = int(argv[1])
    cores = int(argv[2])
    osds = int(argv[3])
```

```

    average_guest_size = int(argv[4])
    average_guest_util = float(argv[5])
except ValueError:
    usage("Non-integer input parameter")

average_guest_util_percent = 100 * average_guest_util

# print inputs
print "Inputs:"
print "- Total host RAM in GB: %d" % mem
print "- Total host cores: %d" % cores
print "- Ceph OSDs per host: %d" % osds
print "- Average guest memory size in GB: %d" % average_guest_size
print "- Average guest CPU utilization: %.0f%%" %
average_guest_util_percent

# calculate operating parameters based on memory constraints only
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))
nonceph_cores = cores - (cores_per_OSD * osds)
guest_vCPUs = nonceph_cores / average_guest_util
cpu_allocation_ratio = guest_vCPUs / cores

# display outputs including how to tune Nova reserved mem

print "\nResults:"
print "- number of guests allowed based on memory = %d" % number_of_guests
print "- number of guest vCPUs allowed = %d" % int(guest_vCPUs)
print "- nova.conf reserved_host_memory = %d MB" % nova_reserved_mem_MB
print "- nova.conf cpu_allocation_ratio = %f" % cpu_allocation_ratio

if nova_reserved_mem_MB > (MB_per_GB * mem * 0.8):
    print "ERROR: you do not have enough memory to run hyperconverged!"
    sys.exit(NOTOK)

if cpu_allocation_ratio < 0.5:
    print "WARNING: you may not have enough CPU to run hyperconverged!"

if cpu_allocation_ratio > 16.0:
    print(
        "WARNING: do not increase VCPU overcommit ratio " +
        "beyond OSP8 default of 16:1")
    sys.exit(NOTOK)

print "\nCompare \"guest vCPUs allowed\" to \"guests allowed based on
memory\" for actual guest count"

```

A.2.2. Ceph OSD サービス向けの NUMA ピニングを設定するカスタムスクリプト

「Ceph NUMA ピニングの設定」の項では、Ceph OSD サービスを利用可能な NUMA ソケットにピニングするスクリプトの作成について説明しています。このスクリプト `~/templates/numa-systemd-osd.sh` は、以下の操作を行う必要があります。

- Ceph のネットワークトラフィックに使用されているネットワークインターフェースを取得します。
- **lstopo** を使用して、そのインターフェースの NUMA ソケットを特定します。
- Ceph のネットワークに使用されているネットワークインターフェースのある NUMA ノードを優先する NUMA ポリシーを使用して、**numactl** が OSD サービスを起動するように設定します。
- 各 Ceph OSD デーモンを順次再起動して、サービスが新しい NUMA オプションを使用して実行されるようにします。



重要

numa-systemd-osd.sh のスクリプトは、NUMA 設定ツールのインストールも試みます。そのため、「[ノードの登録](#)」(『[オーバークラウド向けの Red Hat Ceph Storage](#)』)の説明に従って、オーバークラウドノードを Red Hat に登録しておく必要もあります。

以下のスクリプトは、これらすべての操作を混合型 HCI デプロイメントで実行します。各ハイパーコンバージドノードのホスト名には **ceph** または **osd-compute** が使用されていることを前提とします。ハイパーコンバージドのホスト名をカスタマイズする場合には、最上位の **IF** ステートメントを適宜編集します。

```
#!/usr/bin/env bash
{
if [[ `hostname` = *"ceph"* ]] || [[ `hostname` = *"osd-compute"* ]]; then
# 1

    # Verify the passed network interface exists
    if [[ ! $(ip add show $OSD_NUMA_INTERFACE) ]]; then
exit 1
    fi

    # If NUMA related packages are missing, then install them
    # If packages are baked into image, no install attempted
    for PKG in numactl hwloc; do
if [[ ! $(rpm -q $PKG) ]]; then
    yum install -y $PKG
    if [[ ! $? ]]; then
echo "Unable to install $PKG with yum"
exit 1
    fi
fi
done

    if [[ ! $(lstopo-no-graphics | tr -d [:punct:] | egrep
"NUMANode|$OSD_NUMA_INTERFACE") ]];
    then
echo "No NUMANodes found. Exiting."
exit 1
    fi
fi
}
```

```

# Find the NUMA socket of the $OSD_NUMA_INTERFACE
declare -A NUMASOCKET
while read TYPE SOCKET_NUM NIC ; do
if [[ "$TYPE" == "NUMANode" ]]; then
    NUMASOCKET=$(echo $SOCKET_NUM | sed s/L//g);
fi
if [[ "$NIC" == "$OSD_NUMA_INTERFACE" ]]; then
    # because $NIC is the $OSD_NUMA_INTERFACE,
    # the NUMASOCKET has been set correctly above
    break # so stop looking
fi
done < <(lstopo-no-graphics | tr -d [:punct:] | egrep
"NUMANode|$OSD_NUMA_INTERFACE")

if [[ -z $NUMASOCKET ]]; then
echo "No NUMANode found for $OSD_NUMA_INTERFACE. Exiting."
exit 1
fi

UNIT='/usr/lib/systemd/system/ceph-osd@.service'
# Preserve the original ceph-osd start command
CMD=$(crudini --get $UNIT Service ExecStart)

if [[ $(echo $CMD | grep numactl) ]]; then
echo "numactl already in $UNIT. No changes required."
exit 0
fi

# NUMA control options to append in front of $CMD
NUMA="/usr/bin/numactl -N $NUMASOCKET --preferred=$NUMASOCKET"

# Update the unit file to start with numactl
# TODO: why doesn't a copy of $UNIT in /etc/systemd/system work with
numactl?
crudini --verbose --set $UNIT Service ExecStart "$NUMA $CMD"

# Reload so updated file is used
systemctl daemon-reload

# Restart OSDs with NUMA policy (print results for log)
OSD_IDS=$(ls /var/lib/ceph/osd | awk 'BEGIN { FS = "-" } ; { print $2
}')
for OSD_ID in $OSD_IDS; do
echo -e "\nStatus of OSD $OSD_ID before unit file update\n"
systemctl status ceph-osd@$OSD_ID
echo -e "\nRestarting OSD $OSD_ID..."
systemctl restart ceph-osd@$OSD_ID
echo -e "\nStatus of OSD $OSD_ID after unit file update\n"
systemctl status ceph-osd@$OSD_ID
done
fi
} 2>&1 > /root/post_deploy_heat_output.txt

```

- 1 最上位の **IF** ステートメントは、各ハイパーコンバージドノードのホスト名に **ceph** または **osd-compute** が含まれていることを前提としています。ハイパーコンバージドノードのホスト名をカスタマイズする場合には、**IF** ステートメントを適宜編集してください。

また同様に、単一型の HCI デプロイメントでは、全コンピュータノードがハイパーコンバージド化されるので、単一型の HCI デプロイメントでは、最上位の **IF** ステートメントを以下のように変更してください。

```
if [[ `hostname` = *"compute"* ]]; then
```