# Red Hat OpenStack Platform 17.0

## 高可用性デプロイメントと使用方法

Red Hat OpenStack Platform における高可用性のプランニング、デプロイ、および管理

Last Updated: 2024-03-23

# Red Hat OpenStack Platform 17.0 高可用性デプロイメントと使用方法

Red Hat OpenStack Platform における高可用性のプランニング、デプロイ、および管理

OpenStack Team
rhos-docs@redhat.com

## 法律上の通知

Copyright © 2024 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, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

## 概要

OpenStack の環境が効率的に稼働する状態を維持するためには、Red Hat OpenStack Platform director を使用して、OpenStack の主要な全サービスにわたって高可用性および負荷分散を提供する設定を構築します。

# 目次

# 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

# 第1章 RED HAT OPENSTACK PLATFORM の高可用性の概要とプランニング

Red Hat OpenStack Platform (RHOSP) の高可用性 (HA) は、デプロイメントにおけるフェイルオーバーとリカバリーのオーケストレーションを行うサービスのコレクションです。HA デプロイメントを計画する際には、ハードウェア割り当てやネットワーク設定など、環境のさまざまな側面について検討してください。

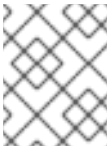
## 1.1. RED HAT OPENSTACK PLATFORM の高可用性サービス

Red Hat OpenStack Platform (RHOSP) は、高可用性 (HA) を実装するのに必要なサービスを提供するための、さまざまなテクノロジーを採用しています。これらのサービスには、Galera、RabbitMQ、Redis、HAProxy、Pacemaker が管理する個別サービス、Systemd、Podman が管理するプレーンコンテナーサービスが含まれます。

### 1.1.1. サービスの種別

**コアコンテナー**

コアコンテナーサービスには、Galera、RabbitMQ、Redis、および HAProxy が含まれます。これらのサービスはすべてのコントローラーノード上で実行され、開始、停止、再起動の各処理に固有の管理と制約が必要です。Pacemaker を使用して、コアコンテナーサービスの起動、管理、およびトラブルシューティングを行います。



**注記**

RHOSP では、MariaDB Galera Cluster を使用してデータベースのレプリケーションを管理します。

**アクティブ/パッシブ**

アクティブ/パッシブのサービスは、1回に1つの Controller ノードでのみ実行され、**openstack-cinder-volume** などのサービスが含まれます。アクティブ/パッシブのサービスを移動するには、Pacemaker を使用して正しい停止/起動シーケンスが実施されるようにする必要があります。

**systemd とプレーンコンテナー**

systemd およびプレーンコンテナーのサービスは独立したサービスで、サービスの中断に対する耐性があります。したがって、Galera 等の高可用性サービスを再起動した場合は、**nova-api** 等の他のサービスを手動で再起動する必要はありません。systemd または Podman を使用して、systemd およびプレーンコンテナーのサービスを直接管理することができます。
HA デプロイメントをオーケストレーションする場合、director はテンプレートおよび Puppet モジュールを使用して、すべてのサービスが正しく設定および起動されるようにします。また、HA の問題のトラブルシューティングを行う場合、**podman** コマンドまたは **systemctl** コマンドを使用して、HA フレームワークのサービスと対話する必要があります。

### 1.1.2. サービスのモード

HA サービスは、以下のいずれかのモードで動作することができます。

**アクティブ/アクティブ**

Pacemaker は同じサービスを複数のコントローラーノードで実行し、HAProxy を使用してトラフィックをノード間に分散するか、1つの IP アドレスにより特定のコントローラーに転送します。HAProxy はラウンドロビンのスケジューリングを使用して、アクティブ/アクティブのサービスにト

ラフィックを分散する場合もあります。コントローラーノードを追加して、パフォーマンスを向上させることができます。

**重要**

アクティブ/アクティブモードは、エッジサイトの分散コンピュートノード (DCN) アーキテクチャーでのみサポートされます。

**アクティブ/パッシブ**

アクティブ/アクティブモードで実行することのできないサービスは、アクティブ/パッシブモードで実行する必要があります。このモードでは、1 度にアクティブにできるサービスのインスタンスは 1 つだけです。たとえば、HAProxy は stick-table オプションを使用して、受信した Galera データベースの接続要求を 1 つのバックエンドサービスに転送します。このモードは、複数の Galera ノードから同じデータに同時に多数の接続が行われるのを防ぐのに役立ちます。

## 1.2. 高可用性ハードウェア割り当てのプランニング

ハードウェアの割り当てを計画する際には、デプロイメントで実行するノード数と、コンピュートノード上で実行する仮想マシン (vm) インスタンス数を考慮してください。

**コントローラーノード**

ストレージ以外のほとんどのサービスは、コントローラーノード上で実行されます。すべてのサービスは 3 つのノードにわたって複製され、アクティブ/アクティブまたはアクティブ/パッシブのサービスとして設定されます。高可用性 (HA) 環境には、最低でも 3 つのノードが必要です。

**Red Hat Ceph Storage ノード**

ストレージサービスはこれらのノード上で実行され、コンピュートノードに Red Hat Ceph Storage 領域プールを提供します。最低でも 3 つのノードが必要です。

**コンピュートノード**

仮想マシン (VM) インスタンスは、コンピュートノード上で実行されます。能力の要件ならびに移行およびリブート操作に必要な数だけ、コンピュートノードをデプロイすることができます。仮想マシンがストレージノード、他のコンピュートノード上の仮想マシン、およびパブリックネットワークにアクセスできるようにするため、コンピュートノードをストレージネットワークおよびプロジェクトネットワークに接続する必要があります。

**STONITH**

高可用性オーバークラウドの Pacemaker クラスターの一部である各ノードには、STONITH デバイスを設定する必要があります。STONITH を使用しない高可用性オーバークラウドのデプロイはサポートの対象外です。STONITH および Pacemaker の詳細は、Red Hat High Availability クラスターでのフェンシングの設定 および RHEL High Availability クラスターのサポートポリシー - フェンシング/STONITH の一般的な要件 を参照してください。

## 1.3. 高可用性ネットワークのプランニング

仮想ネットワークおよび物理ネットワークを計画する際には、プロビジョニングネットワークスイッチの設定および外部ネットワークスイッチの設定を検討してください。

ネットワーク設定に加えて、以下のコンポーネントをデプロイする必要があります。

**プロビジョニングネットワーク用のスイッチ**

- このスイッチは、アンダークラウドをオーバークラウド内のすべての物理コンピューターに接続できる必要があります。

- このスイッチに接続されている各オーバークラウドノードの NIC は、アンダークラウドから PXE ブートできなければなりません。
- **portfast** パラメーターを有効にする必要があります。

**コントローラー/外部ネットワーク用のスイッチ**

- このスイッチは、デプロイメント内の他の VLAN の VLAN タグ付けを行うように設定する必要があります。
- VLAN 100 トラフィックのみを外部ネットワークに許可します。

**ネットワークハードウェアと keystone エンドポイント**

- コントローラーノードのネットワークカードまたはネットワークスイッチの異常がオーバークラウドサービスの可用性を阻害するのを防ぐには、keystone 管理エンドポイントがボンディングされたネットワークカードまたはネットワークハードウェアの冗長性を使用するネットワークに配置されるようにしてください。
  keystone エンドポイントを **internal_api** などの別のネットワークに移動する場合は、アンダークラウドが VLAN またはサブネットに到達できるようにします。詳細は、Red Hat ナレッジベースのソリューション Keystone Admin Endpoint を internal_api network に移行する方法 を参照してください。

## 1.4. 高可用性環境へのアクセス

高可用性 (HA) ノードを調査するには、**stack** ユーザーを使用してオーバークラウドノードにログインし、**openstack server list** コマンドを実行してノードのステータスと詳細を表示します。

**前提条件**

- 高可用性がデプロイされ、動作している。

**手順**

1. 動作中の HA 環境で、アンダークラウドに **stack** ユーザーとしてログインします。
2. オーバークラウドノードの IP アドレスを特定します。

```
$ source ~/stackrc
(undercloud) $ openstack server list
+-------+-----------------------+---+---------------------+---+
| ID    | Name                  |...| Networks            |...|
+-------+-----------------------+---+---------------------+---+
| d1... | overcloud-controller-0 |...| ctlplane=*10.200.0.11* |...|
...
```

3. オーバークラウドノードのいずれかにログインします。

```
(undercloud) $ ssh tripleo-admin@<node_IP>
```

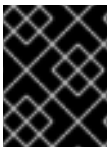
**<node_ip>** は、ログインするノードの IP アドレスに置き換えます。

## 1.5. 関連情報

- *2章 デプロイメントの例: Compute および Ceph サービスを持つ高可用性クラスター*

# 第2章 デプロイメントの例: COMPUTE および CEPH サービスを持つ高可用性クラスター

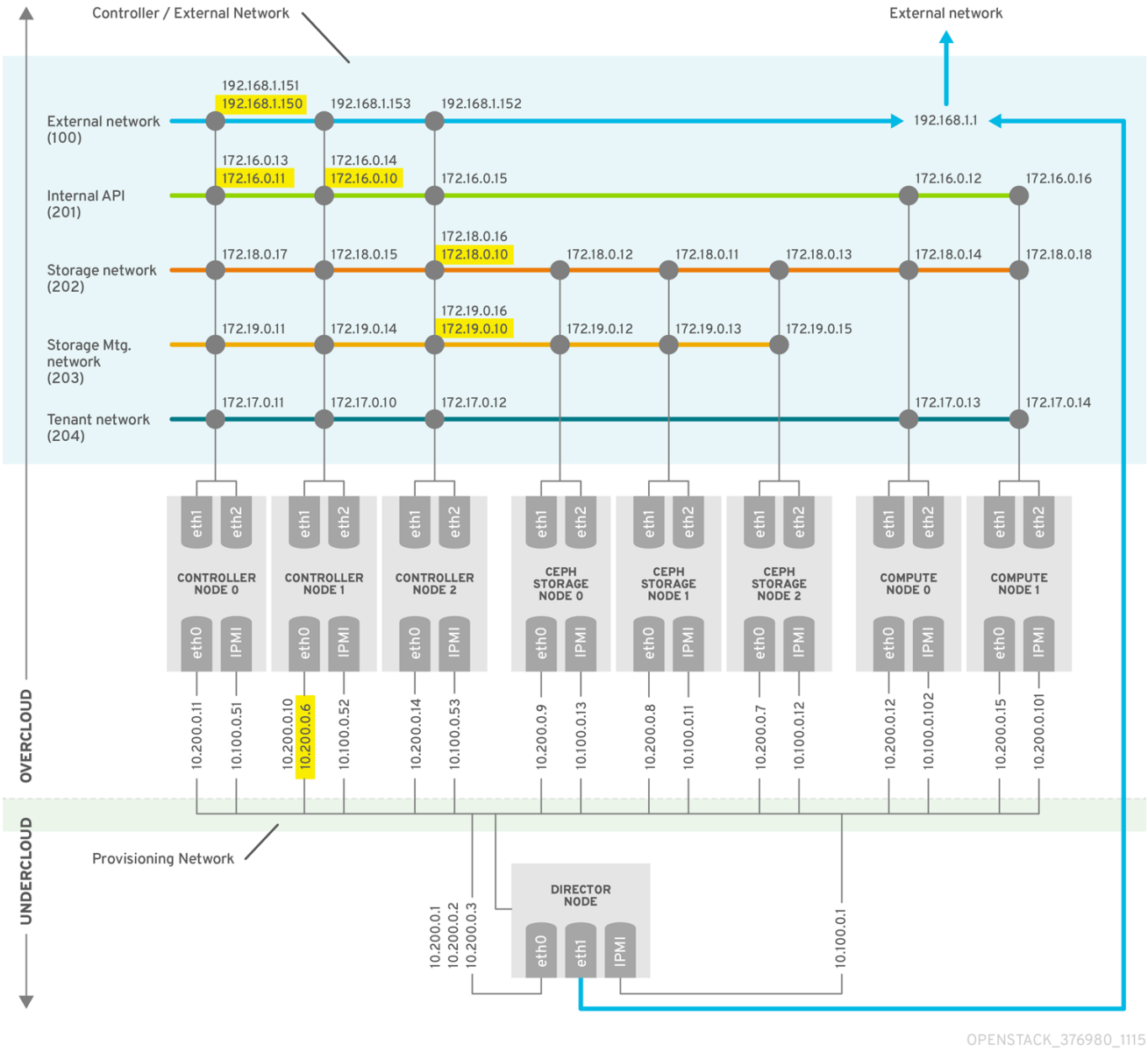
以下のシナリオ例で、OpenStack Compute サービスおよび Red Hat Ceph Storage を持つ高可用性デプロイメントのアーキテクチャー、ハードウェアおよびネットワークの仕様、ならびにアンダークラウドおよびオーバークラウドの設定ファイルについて説明します。



**重要**

このデプロイメントはテスト環境の参照用として使用することを目的としており、実稼働環境用としてはサポートされません。

**図2.1 高可用性デプロイメントアーキテクチャーの例**




## 2.1. 高可用性ハードウェアの仕様の例

HA デプロイメントの例では、特定のハードウェア設定を使用します。ご自分のテストデプロイメントのニーズに応じて、CPU、メモリー、ストレージ、または NIC を調整することができます。

**表2.1 物理コンピューター**

| コンピューターの台数 | 目的 | CPU の数 | メモリー | ディスク容量 | 電源管理 | NIC の数 |
|---|---|---|---|---|---|---|
| 1 | アンダークラウドノード | 4 | 24 GB | 40 GB | IPMI | 2 (外部 x 1、プロビジョニング x 1) + 1 IPMI |
| 3 | コントローラーノード | 4 | 24 GB | 40 GB | IPMI | 3 (オーバークラウド上のボンディング x 2、プロビジョニング x 1) + 1 IPMI |
| 3 | Ceph Storage ノード | 4 | 24 GB | 40 GB | IPMI | 3 (オーバークラウド上のボンディング x 2、プロビジョニング x 1) + 1 IPMI |
| 2 | コンピュートノード (必要に応じて追加する) | 4 | 24 GB | 40 GB | IPMI | 3 (オーバークラウド上のボンディング x 2、プロビジョニング x 1) + 1 IPMI |

## 2.2. 高可用性ネットワークの仕様の例

HA デプロイメントの例では、特定の仮想および物理ネットワーク設定を使用します。ご自分のテストデプロイメントのニーズに応じて、設定を調整することができます。

**注記**

この例には、コントロールプレーンと、オーバークラウドの keystone 管理エンドポイントが設定されたプロビジョニングネットワーク用のハードウェア冗長性は含まれません。高可用性ネットワークを計画する方法は、「高可用性ネットワークのプランニング」 を参照してください。

**表2.2 物理ネットワークおよび仮想ネットワーク**

| 物理 NIC | 目的 | VLAN | 説明 |
|---|---|---|---|

| 物理 NIC | 目的 | VLAN | 説明 |
|---|---|---|---|
| eth0 | プロビジョニングネットワーク (アンダークラウド) | 該当なし | すべてのノードを director (アンダークラウド) から管理します。 |
| eth1 および eth2 | コントローラー/外部 (オーバークラウド) | 該当なし | ボンディングされた NIC および VLAN の組み合わせ |
| | 外部ネットワーク | VLAN 100 | 外部環境からプロジェクトネットワーク、内部 API、および OpenStack Horizon Dashboard へのアクセスを許可します。 |
| | 内部 API | VLAN 201 | コンピュートノードとコントローラーノード間の内部 API へのアクセスを提供します。 |
| | ストレージアクセス | VLAN 202 | コンピュートノードをストレージメディアに接続します。 |
| | ストレージ管理 | VLAN 203 | ストレージメディアを管理します。 |
| | プロジェクトネットワーク | VLAN 204 | RHOSP にプロジェクトネットワークサービスを提供します。 |

## 2.3. 高可用性アンダークラウドの設定ファイルの例

HA デプロイメント例では、アンダークラウドの設定ファイル **instackenv.json**、**undercloud.conf**、および **network-environment.yaml** を使用します。

### instackenv.json

```
{
    "nodes": [
      {
        "pm_password": "testpass",
        "memory": "24",
        "pm_addr": "10.100.0.11",
        "mac": [
          "2c:c2:60:3b:b3:94"
        ],
        "pm_type": "ipmi",
```

```
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "24",
    "pm_addr": "10.100.0.12",
    "mac": [
      "2c:c2:60:51:b7:fb"
    ],
    "pm_type": "ipmi",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "24",
    "pm_addr": "10.100.0.13",
    "mac": [
      "2c:c2:60:76:ce:a5"
    ],
    "pm_type": "ipmi",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "24",
    "pm_addr": "10.100.0.51",
    "mac": [
      "2c:c2:60:08:b1:e2"
    ],
    "pm_type": "ipmi",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
  },
  {
    "pm_password": "testpass",
    "memory": "24",
    "pm_addr": "10.100.0.52",
    "mac": [
      "2c:c2:60:20:a1:9e"
    ],
    "pm_type": "ipmi",
    "disk": "40",
    "arch": "x86_64",
    "cpu": "1",
    "pm_user": "admin"
```

```
        },
        {
          "pm_password": "testpass",
          "memory": "24",
          "pm_addr": "10.100.0.53",
          "mac": [
            "2c:c2:60:58:10:33"
          ],
          "pm_type": "ipmi",
          "disk": "40",
          "arch": "x86_64",
          "cpu": "1",
          "pm_user": "admin"
        },
        {
          "pm_password": "testpass",
          "memory": "24",
          "pm_addr": "10.100.0.101",
          "mac": [
            "2c:c2:60:31:a9:55"
          ],
          "pm_type": "ipmi",
          "disk": "40",
          "arch": "x86_64",
          "cpu": "2",
          "pm_user": "admin"
        },
        {
          "pm_password": "testpass",
          "memory": "24",
          "pm_addr": "10.100.0.102",
          "mac": [
            "2c:c2:60:0d:e7:d1"
          ],
          "pm_type": "ipmi",
          "disk": "40",
          "arch": "x86_64",
          "cpu": "2",
          "pm_user": "admin"
        }
      ],
      "overcloud": {"password": "7adbbbeedc5b7a07ba1917e1b3b228334f9a2d4e",
      "endpoint": "http://192.168.1.150:5000/v2.0/"
                  }
}
```

### undercloud.conf

```
[DEFAULT]
image_path = /home/stack/images
local_ip = 10.200.0.1/24
undercloud_public_vip = 10.200.0.2
undercloud_admin_vip = 10.200.0.3
undercloud_service_certificate = /etc/pki/instack-certs/undercloud.pem
local_interface = eth0
masquerade_network = 10.200.0.0/24
```

```
dhcp_start = 10.200.0.5
dhcp_end = 10.200.0.24
network_cidr = 10.200.0.0/24
network_gateway = 10.200.0.1
#discovery_interface = br-ctlplane
discovery_iprange = 10.200.0.150,10.200.0.200
discovery_runbench = 1
undercloud_admin_password = testpass
...
```

**network-environment.yaml**

```
resource_registry:
  OS::TripleO::BlockStorage::Net::SoftwareConfig: /home/stack/templates/nic-configs/cinder-
storage.yaml
  OS::TripleO::Compute::Net::SoftwareConfig: /home/stack/templates/nic-configs/compute.yaml
  OS::TripleO::Controller::Net::SoftwareConfig: /home/stack/templates/nic-configs/controller.yaml
  OS::TripleO::ObjectStorage::Net::SoftwareConfig: /home/stack/templates/nic-configs/swift-
storage.yaml
  OS::TripleO::CephStorage::Net::SoftwareConfig: /home/stack/templates/nic-configs/ceph-
storage.yaml

parameter_defaults:
  InternalApiNetCidr: 172.16.0.0/24
  TenantNetCidr: 172.17.0.0/24
  StorageNetCidr: 172.18.0.0/24
  StorageMgmtNetCidr: 172.19.0.0/24
  ExternalNetCidr: 192.168.1.0/24
  InternalApiAllocationPools: [{start: 172.16.0.10, end: 172.16.0.200}]
  TenantAllocationPools: [{start: 172.17.0.10, end: 172.17.0.200}]
  StorageAllocationPools: [{start: 172.18.0.10, end: 172.18.0.200}]
  StorageMgmtAllocationPools: [{start: 172.19.0.10, end: 172.19.0.200}]
  # Leave room for floating IPs in the External allocation pool
  ExternalAllocationPools: [{start: 192.168.1.150, end: 192.168.1.199}]
  InternalApiNetworkVlanID: 201
  StorageNetworkVlanID: 202
  StorageMgmtNetworkVlanID: 203
  TenantNetworkVlanID: 204
  ExternalNetworkVlanID: 100
  # Set to the router gateway on the external network
  ExternalInterfaceDefaultRoute: 192.168.1.1
  # Set to "br-ex" if using floating IPs on native VLAN on bridge br-ex
  NeutronExternalNetworkBridge: """"
  # Customize bonding options if required
  BondInterfaceOvsOptions:
    "bond_mode=active-backup lacp=off other_config:bond-miimon-interval=100"
```

## 2.4. 高可用性オーバークラウドの設定ファイルの例

HA デプロイメント例では、オーバークラウドの設定ファイル **haproxy.cfg**、**corosync.cfg**、および **ceph.cfg** が使用されます。

**/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg (コントローラーノード)**

このファイルは、HAProxy が管理するサービスを特定します。これには、HAProxy がモニタリングするサービスの設定が含まれます。このファイルは、すべてのコントローラーノードで同じ内容です。

```
# This file is managed by Puppet
global
  daemon
  group  haproxy
  log  /dev/log local0
  maxconn  20480
  pidfile  /var/run/haproxy.pid
  ssl-default-bind-ciphers
!SSLv2:kEECDH:kRSA:kEDH:kPSK:+3DES:!aNULL:!eNULL:!MD5:!EXP:!RC4:!SEED:!IDEA:!DES
  ssl-default-bind-options  no-sslv3
  stats  socket /var/lib/haproxy/stats mode 600 level user
  stats  timeout 2m
  user  haproxy

defaults
  log  global
  maxconn  4096
  mode  tcp
  retries  3
  timeout  http-request 10s
  timeout  queue 2m
  timeout  connect 10s
  timeout  client 2m
  timeout  server 2m
  timeout  check 10s

listen aodh
  bind 192.168.1.150:8042 transparent
  bind 172.16.0.10:8042 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8042 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8042 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8042 check fall 5 inter 2000 rise 2

listen cinder
  bind 192.168.1.150:8776 transparent
  bind 172.16.0.10:8776 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8776 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8776 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8776 check fall 5 inter 2000 rise 2

listen glance_api
  bind 192.168.1.150:9292 transparent
  bind 172.18.0.10:9292 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
```

```
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk GET /healthcheck
  server overcloud-controller-0.internalapi.localdomain 172.18.0.17:9292 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.18.0.15:9292 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.18.0.16:9292 check fall 5 inter 2000 rise 2

listen gnocchi
  bind 192.168.1.150:8041 transparent
  bind 172.16.0.10:8041 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8041 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8041 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8041 check fall 5 inter 2000 rise 2

listen haproxy.stats
  bind 10.200.0.6:1993 transparent
  mode http
  stats enable
  stats uri /
  stats auth admin:PnDD32EzdVCf73CpjHhFGHZdV

listen heat_api
  bind 192.168.1.150:8004 transparent
  bind 172.16.0.10:8004 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  timeout client 10m
  timeout server 10m
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8004 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8004 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8004 check fall 5 inter 2000 rise 2

listen heat_cfn
  bind 192.168.1.150:8000 transparent
  bind 172.16.0.10:8000 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  timeout client 10m
  timeout server 10m
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8000 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8000 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8000 check fall 5 inter 2000 rise 2

listen horizon
  bind 192.168.1.150:80 transparent
  bind 172.16.0.10:80 transparent
  mode http
  cookie SERVERID insert indirect nocache
  option forwardfor
```

```
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:80 check cookie overcloud-
controller-0 fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:80 check cookie overcloud-
controller-0 fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:80 check cookie overcloud-
controller-0 fall 5 inter 2000 rise 2

listen keystone_admin
  bind 192.168.24.15:35357 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk GET /v3
  server overcloud-controller-0.ctlplane.localdomain 192.168.24.9:35357 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.ctlplane.localdomain 192.168.24.8:35357 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.ctlplane.localdomain 192.168.24.18:35357 check fall 5 inter 2000 rise
2

listen keystone_public
  bind 192.168.1.150:5000 transparent
  bind 172.16.0.10:5000 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk GET /v3
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:5000 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:5000 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:5000 check fall 5 inter 2000 rise 2

listen mysql
  bind 172.16.0.10:3306 transparent
  option tcpka
  option httpchk
  stick on dst
  stick-table type ip size 1000
  timeout client 90m
  timeout server 90m
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:3306 backup check inter 1s on-
marked-down shutdown-sessions port 9200
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:3306 backup check inter 1s on-
marked-down shutdown-sessions port 9200
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:3306 backup check inter 1s on-
marked-down shutdown-sessions port 9200

listen neutron
  bind 192.168.1.150:9696 transparent
  bind 172.16.0.10:9696 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:9696 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:9696 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:9696 check fall 5 inter 2000 rise 2
```

```
listen nova_metadata
  bind 172.16.0.10:8775 transparent
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8775 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8775 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8775 check fall 5 inter 2000 rise 2

listen nova_novncproxy
  bind 192.168.1.150:6080 transparent
  bind 172.16.0.10:6080 transparent
  balance source
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option tcpka
  timeout tunnel 1h
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:6080 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:6080 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:6080 check fall 5 inter 2000 rise 2

listen nova_osapi
  bind 192.168.1.150:8774 transparent
  bind 172.16.0.10:8774 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8774 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8774 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8774 check fall 5 inter 2000 rise 2

listen nova_placement
  bind 192.168.1.150:8778 transparent
  bind 172.16.0.10:8778 transparent
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8778 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8778 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8778 check fall 5 inter 2000 rise 2

listen panko
  bind 192.168.1.150:8977 transparent
  bind 172.16.0.10:8977 transparent
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:8977 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:8977 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:8977 check fall 5 inter 2000 rise 2

listen redis
  bind 172.16.0.13:6379 transparent
  balance first
  option tcp-check
  tcp-check send AUTH\ V2EgUh2pvkr8VzU6yuE4XHsr9\r\n
```

```
  tcp-check send PING\r\n
  tcp-check expect string +PONG
  tcp-check send info\ replication\r\n
  tcp-check expect string role:master
  tcp-check send QUIT\r\n
  tcp-check expect string +OK
  server overcloud-controller-0.internalapi.localdomain 172.16.0.13:6379 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.internalapi.localdomain 172.16.0.14:6379 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.internalapi.localdomain 172.16.0.15:6379 check fall 5 inter 2000 rise 2

listen swift_proxy_server
  bind 192.168.1.150:8080 transparent
  bind 172.18.0.10:8080 transparent
  option httpchk GET /healthcheck
  timeout client 2m
  timeout server 2m
  server overcloud-controller-0.storage.localdomain 172.18.0.17:8080 check fall 5 inter 2000 rise 2
  server overcloud-controller-1.storage.localdomain 172.18.0.15:8080 check fall 5 inter 2000 rise 2
  server overcloud-controller-2.storage.localdomain 172.18.0.16:8080 check fall 5 inter 2000 rise 2
```

### /etc/corosync/corosync.conf ファイル (コントローラーノード)

このファイルは、クラスターのインフラストラクチャーを定義し、すべてのコントローラーノード上で利用することができます。

```
totem {
 version: 2
 cluster_name: tripleo_cluster
 transport: udpu
 token: 10000
}


nodelist {
   node {
       ring0_addr: overcloud-controller-0
       nodeid: 1
   }


   node {
       ring0_addr: overcloud-controller-1
       nodeid: 2
   }


   node {
       ring0_addr: overcloud-controller-2
       nodeid: 3
   }
}

quorum {
   provider: corosync_votequorum
}
```

```
logging {
    to_logfile: yes
   logfile: /var/log/cluster/corosync.log
   to_syslog: yes
}
```

### /etc/ceph/ceph.conf (Ceph ノード)

このファイルには、Ceph の高可用性設定が記載されています。これには、モニタリングするホストのホスト名、IP アドレスが含まれます。

```
[global]
osd_pool_default_pgp_num = 128
osd_pool_default_min_size = 1
auth_service_required = cephx
mon_initial_members = overcloud-controller-0,overcloud-controller-1,overcloud-controller-2
fsid = 8c835acc-6838-11e5-bb96-2cc260178a92
cluster_network = 172.19.0.11/24
auth_supported = cephx
auth_cluster_required = cephx
mon_host = 172.18.0.17,172.18.0.15,172.18.0.16
auth_client_required = cephx
osd_pool_default_size = 3
osd_pool_default_pg_num = 128
public_network = 172.18.0.17/24
```

## 2.5. 関連情報

- director とともに Red Hat Ceph Storage および Red Hat OpenStack Platform をデプロイする
- 1章*Red Hat OpenStack Platform の高可用性の概要とプランニング*

# 第3章 PACEMAKER を使用した高可用性サービスの管理

Pacemaker サービスは、Galera、RabbitMQ、Redis、および HAProxy 等のコアコンテナーのサービスおよびアクティブパッシブのサービスを管理します。Pacemaker を使用して、マネージドサービス、仮想 IP アドレス、電源管理、およびフェンシングについての一般的な情報を表示および管理します。

## 3.1. PACEMAKER リソースバンドルとコンテナー

Pacemaker は、Red Hat OpenStack Platform (RHOSP) のサービスをバンドルセットリソース (バンドル) として管理します。これらのサービスのほとんどはアクティブ/アクティブのサービスで、それぞれのコントローラーノード上で同じように起動し常に動作しています。

ペースメーカーは以下のリソース種別を管理します。

**バンドル**

バンドルリソースはすべてのコントローラーノードで同じコンテナーを設定および複製し、必要なストレージパスをコンテナーディレクトリーにマッピングし、リソース自体に関連する特定の属性を設定します。

**Container**

コンテナーは、HAProxy のような単純な **systemd** サービスから、異なるノード上のサービスの状態を制御および設定する特定のリソースエージェントを必要とする Galera のような複雑なサービスまで、さまざまな種類のリソースを実行することができます。

**重要**

- バンドルまたはコンテナーを管理するのに、**podman** または **systemctl** を使用することはできません。これらのコマンドを使用してサービスのステータスを確認することはできますが、これらのサービスに対してアクションを実行するには Pacemaker を使用する必要があります。
- Pacemaker が制御する Podman コンテナーでは、Podman により **RestartPolicy** が **no** に設定されます。これは、Podman ではなく Pacemaker がコンテナーの起動と停止のアクションを制御するようにするためです。

### 3.1.1. 簡易バンドルセットリソース (簡易バンドル)

簡易バンドルセットリソース (簡易バンドル) はコンテナーのセットで、それぞれのコンテナーには全コントローラーノードにわたってデプロイする同じ Pacemaker サービスが含まれます。

以下の例は、**pcs status** コマンドで出力される簡易バンドルのリストを示しています。

```
Podman container set: haproxy-bundle [192.168.24.1:8787/rhosp-rhel9/openstack-
haproxy:pcmklatest]
  haproxy-bundle-podman-0    (ocf::heartbeat:podman):    Started overcloud-controller-0
  haproxy-bundle-podman-1    (ocf::heartbeat:podman):    Started overcloud-controller-1
  haproxy-bundle-podman-2    (ocf::heartbeat:podman):    Started overcloud-controller-2
```

各バンドルでは、以下の情報を確認することができます。

- Pacemaker がサービスに割り当てる名前
- バンドルに関連付けられたコンテナーへの参照

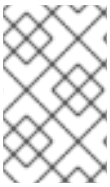
- 異なるコントローラーノードで実行中のレプリカのリストおよびステータス

以下の例は、**haproxy-bundle** 簡易バンドルの設定を示しています。

```
$ sudo pcs resource show haproxy-bundle
Bundle: haproxy-bundle
 Podman: image=192.168.24.1:8787/rhosp-rhel9/openstack-haproxy:pcmklatest network=host
options="--user=root --log-driver=journald -e KOLLA_CONFIG_STRATEGY=COPY_ALWAYS"
replicas=3 run-command="/bin/bash /usr/local/bin/kolla_start"
 Storage Mapping:
  options=ro source-dir=/var/lib/kolla/config_files/haproxy.json target-
dir=/var/lib/kolla/config_files/config.json (haproxy-cfg-files)
  options=ro source-dir=/var/lib/config-data/puppet-generated/haproxy/ target-
dir=/var/lib/kolla/config_files/src (haproxy-cfg-data)
  options=ro source-dir=/etc/hosts target-dir=/etc/hosts (haproxy-hosts)
  options=ro source-dir=/etc/localtime target-dir=/etc/localtime (haproxy-localtime)
  options=ro source-dir=/etc/pki/ca-trust/extracted target-dir=/etc/pki/ca-trust/extracted (haproxy-pki-
extracted)
  options=ro source-dir=/etc/pki/tls/certs/ca-bundle.crt target-dir=/etc/pki/tls/certs/ca-bundle.crt
(haproxy-pki-ca-bundle-crt)
  options=ro source-dir=/etc/pki/tls/certs/ca-bundle.trust.crt target-dir=/etc/pki/tls/certs/ca-
bundle.trust.crt (haproxy-pki-ca-bundle-trust-crt)
  options=ro source-dir=/etc/pki/tls/cert.pem target-dir=/etc/pki/tls/cert.pem (haproxy-pki-cert)
  options=rw source-dir=/dev/log target-dir=/dev/log (haproxy-dev-log)
```

この例では、バンドル内のコンテナーについて以下の情報を示しています。

- **image**: コンテナーによって使用されるイメージ。アンダークラウドのローカルレジストリーを参照します。
- **network**: コンテナーのネットワーク種別。この例では **"host"** です。
- **options**: コンテナーの特定のオプション
- **replicas**: クラスター内で実行する必要のあるコンテナーのコピーの数を示します。各バンドルには 3 つのコンテナーが含まれ、それぞれが各コントローラーノードに対応します。
- **run-command**: コンテナーの起動に使用するシステムコマンド
- Storage Mapping: 各ホスト上のローカルパスからコンテナーへのマッピング。**haproxy** 設定は、**/etc/haproxy/haproxy.cfg** ファイルではなく、**/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** ファイルにあります。



**注記**

HAProxy は、トラフィックの負荷を選択したサービスに分散することによって高可用性サービスを提供しますが、ここでは HAProxy を Pacemaker のバンドルサービスとして管理することによって HAProxy を高可用性サービスに設定します。

### 3.1.2. 複合バンドルセットリソース (複合バンドル)

複合バンドルセットリソース (複合バンドル) は、簡易バンドルに含まれる基本的なコンテナーの設定に加えて、リソース設定を指定する Pacemaker サービスです。

この設定は、マルチステートのリソース (実行するコントローラーノードに応じて異なる状態を取ることができるサービス) を管理するのに必要です。

以下の例には、**pcs status** コマンドで出力される複合バンドルのリストを示しています。

```
Podman container set: rabbitmq-bundle [192.168.24.1:8787/rhosp-rhel9/openstack-
rabbitmq:pcmklatest]
  rabbitmq-bundle-0    (ocf::heartbeat:rabbitmq-cluster):      Started overcloud-controller-0
  rabbitmq-bundle-1    (ocf::heartbeat:rabbitmq-cluster):      Started overcloud-controller-1
  rabbitmq-bundle-2    (ocf::heartbeat:rabbitmq-cluster):      Started overcloud-controller-2
Podman container set: galera-bundle [192.168.24.1:8787/rhosp-rhel9/openstack-mariadb:pcmklatest]
  galera-bundle-0      (ocf::heartbeat:galera):        Master overcloud-controller-0
  galera-bundle-1      (ocf::heartbeat:galera):        Master overcloud-controller-1
  galera-bundle-2      (ocf::heartbeat:galera):        Master overcloud-controller-2
Podman container set: redis-bundle [192.168.24.1:8787/rhosp-rhel9/openstack-redis:pcmklatest]
  redis-bundle-0       (ocf::heartbeat:redis): Master overcloud-controller-0
  redis-bundle-1       (ocf::heartbeat:redis): Slave overcloud-controller-1
  redis-bundle-2       (ocf::heartbeat:redis): Slave overcloud-controller-2
```

この出力は、各複合バンドルについての以下の情報を示しています。

- RabbitMQ: 簡易バンドルと同様に、3 つすべてのコントローラーノードが、サービスのスタンドアロンインスタンスを実行している。
- Galera: 3 つすべてのコントローラーノードが、同じ制約下で Galera マスターとして動作している。
- Redis: **overcloud-controller-0** コンテナーはマスターとして動作し、一方、他の 2 つのコントローラーノードはスレーブとして動作している。それぞれのコンテナー種別は、異なる制約下で動作する可能性があります。

以下の例は、**galera-bundle** 複合バンドルの設定を示しています。

```
[...]
Bundle: galera-bundle
 Podman: image=192.168.24.1:8787/rhosp-rhel9/openstack-mariadb:pcmklatest masters=3
network=host options="--user=root --log-driver=journald -e
KOLLA_CONFIG_STRATEGY=COPY_ALWAYS" replicas=3 run-command="/bin/bash
/usr/local/bin/kolla_start"
 Network: control-port=3123
 Storage Mapping:
  options=ro source-dir=/var/lib/kolla/config_files/mysql.json target-
dir=/var/lib/kolla/config_files/config.json (mysql-cfg-files)
  options=ro source-dir=/var/lib/config-data/puppet-generated/mysql/ target-
dir=/var/lib/kolla/config_files/src (mysql-cfg-data)
  options=ro source-dir=/etc/hosts target-dir=/etc/hosts (mysql-hosts)
  options=ro source-dir=/etc/localtime target-dir=/etc/localtime (mysql-localtime)
  options=rw source-dir=/var/lib/mysql target-dir=/var/lib/mysql (mysql-lib)
  options=rw source-dir=/var/log/mariadb target-dir=/var/log/mariadb (mysql-log-mariadb)
  options=rw source-dir=/dev/log target-dir=/dev/log (mysql-dev-log)
 Resource: galera (class=ocf provider=heartbeat type=galera)
  Attributes: additional_parameters=--open-files-limit=16384 cluster_host_map=overcloud-controller-
0:overcloud-controller-0.internalapi.localdomain;overcloud-controller-1:overcloud-controller-
1.internalapi.localdomain;overcloud-controller-2:overcloud-controller-2.internalapi.localdomain
enable_creation=true wsrep_cluster_address=gcomm://overcloud-controller-
0.internalapi.localdomain,overcloud-controller-1.internalapi.localdomain,overcloud-controller-
2.internalapi.localdomain
  Meta Attrs: container-attribute-target=host master-max=3 ordered=true
  Operations: demote interval=0s timeout=120 (galera-demote-interval-0s)
```

```
            monitor interval=20 timeout=30 (galera-monitor-interval-20)
            monitor interval=10 role=Master timeout=30 (galera-monitor-interval-10)
            monitor interval=30 role=Slave timeout=30 (galera-monitor-interval-30)
            promote interval=0s on-fail=block timeout=300s (galera-promote-interval-0s)
            start interval=0s timeout=120 (galera-start-interval-0s)
            stop interval=0s timeout=120 (galera-stop-interval-0s)
[...]
```

この出力は、簡易バンドルとは異なり、**galera-bundle** リソースには、multi-state リソースのあらゆる側面を決定する明示的なリソース設定が含まれていることを示しています。

**注記**

また、サービスは同時に複数のコントローラーノードで実行される可能性がありますが、コントローラーノード自体は、これらのサービスにアクセスするのに必要な IP アドレスをリッスンしていない場合もあります。サービスの IP アドレスを確認する方法は、「高可用性クラスターでの仮想 IP のリソース情報の表示」 を参照してください。

## 3.2. PACEMAKER クラスターのステータスの確認

Pacemaker が稼働している任意のノードで Pacemaker クラスターのステータスを確認し、アクティブで実行中のリソースの数に関する情報を表示できます。

**前提条件**

- 高可用性がデプロイされ、動作している。

**手順**

1. **tripleo-admin** ユーザーとして任意のコントローラーノードにログインします。

```
$ ssh tripleo-admin@overcloud-controller-0
```

2. **pcs status** コマンドを実行します。

```
[tripleo-admin@overcloud-controller-0 ~]  $ sudo pcs status
```

出力例:

```
Cluster name: tripleo_cluster
Stack: corosync
Current DC: overcloud-controller-1 (version 2.0.1-4.el9-0eb7991564) - partition with quorum

Last updated: Thu Feb  8 14:29:21 2018
Last change: Sat Feb  3 11:37:17 2018 by root via cibadmin on overcloud-controller-2

12 nodes configured
37 resources configured

Online: [ overcloud-controller-0 overcloud-controller-1 overcloud-controller-2 ]
GuestOnline: [ galera-bundle-0@overcloud-controller-0 galera-bundle-1@overcloud-
controller-1 galera-bundle-2@overcloud-controller-2 rabbitmq-bundle-0@overcloud-
controller-0 rabbitmq-bundle-1@overcloud-controller-1 rabbitmq-bundle-2@overcloud-
controller-2 redis-bundle-0@overcloud-controller-0 redis-bundle-1@overcloud-controller-1
```

```
redis-bundle-2@overcloud-controller-2 ]

Full list of resources:
[...]
```

この出力の主要なセクションでは、クラスターに関する以下の情報が表示されます。

- **Cluster name**: クラスターの名前。
- **[NUM] nodes configured**: クラスターを設定するノードの数
- **[NUM] resources configured**: クラスターに設定されているリソースの数
- **Online**: 現在オンライン状態の Controller ノードの名前
- **GuestOnline**: 現在オンライン状態のゲストノードの名前。各ゲストノードは、複合バンドルセットのリソースで設定されています。バンドルセットの詳細は、「Pacemaker リソースバンドルとコンテナー」 を参照してください。

## 3.3. 高可用性クラスターでのバンドルステータスの確認

アンダークラウドノードからバンドルのステータスを確認することも、コントローラーノードのいずれかにログインして直接バンドルのステータスを確認することもできます。

**前提条件**

- 高可用性がデプロイされ、動作している。

**手順**

以下のオプションのいずれかを使用します。

- アンダークラウドノードにログインし、バンドルのステータスを確認します (以下の例では **haproxy-bundle**)。

```
$ sudo podman exec -it haproxy-bundle-podman-0 ps -efww | grep haproxy*
```

出力例:

```
root           7       1  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -Ws
haproxy       11       7  0 06:08 ?        00:00:17 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -
Ws
```

この出力は、コンテナー内で **haproxy** プロセスが実行されていることを示しています。

- コントローラーノードにログインし、バンドルのステータスを確認します (以下の例では **haproxy**)。

```
$ ps -ef | grep haproxy*
```

出力例:

```
root       17774   17729  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg
-Ws
42454      17819   17774  0 06:08 ?        00:00:21 /usr/sbin/haproxy -f
```

```
/etc/haproxy/haproxy.cfg -Ws
root      288508  237714  0 07:04 pts/0    00:00:00 grep --color=auto haproxy*
[root@controller-0 ~]# ps -ef | grep -e 17774 -e 17819
root       17774   17729  0 06:08 ?        00:00:00 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg
-Ws
42454      17819   17774  0 06:08 ?        00:00:22 /usr/sbin/haproxy -f
/etc/haproxy/haproxy.cfg -Ws
root      301950  237714  0 07:07 pts/0    00:00:00 grep --color=auto -e 17774 -e 17819
```

## 3.4. 高可用性クラスターでの仮想 IP のリソース情報の表示

すべての仮想 IP (VIP) または特定の仮想 IP のステータスを確認するには、該当するオプションを指定して **pcs resource show** コマンドを実行します。各 IPaddr2 リソースは、クライアントがサービスへのアクセスを要求するために使用する仮想 IP アドレスを設定します。その IP アドレスが割り当てられたコントローラーノードで異常が発生すると、IPaddr2 リソースは IP アドレスを別のコントローラーノードに再割り当てします。

**前提条件**

- 高可用性がデプロイされ、動作している。

**手順**

1. **tripleo-admin** ユーザーとして任意のコントローラーノードにログインします。

```
$ ssh tripleo-admin@overcloud-controller-0
```

2. 以下のオプションのいずれかを使用します。

- **--full** オプションを指定して **pcs resource show** コマンドを実行し、仮想 IP を使用する全リソースを表示します。

```
$ sudo pcs resource show --full
```

出力例:

```
ip-10.200.0.6 (ocf::heartbeat:IPaddr2): Started overcloud-controller-1
ip-192.168.1.150 (ocf::heartbeat:IPaddr2): Started overcloud-controller-0
ip-172.16.0.10 (ocf::heartbeat:IPaddr2): Started overcloud-controller-1
ip-172.16.0.11 (ocf::heartbeat:IPaddr2): Started overcloud-controller-0
ip-172.18.0.10 (ocf::heartbeat:IPaddr2): Started overcloud-controller-2
ip-172.19.0.10 (ocf::heartbeat:IPaddr2): Started overcloud-controller-2
```

初期状態では、各 IP アドレスは特定のコントローラーノードに接続されます。たとえば、**192.168.1.150** は **overcloud-controller-0** で開始されます。ただし、そのコントローラーノードで異常が発生すると、IP アドレスはクラスター内の他のコントローラーノードに再割り当てされます。

以下の表には、この出力例の IP アドレスと、各 IP アドレスの初期の割り当てをまとめています。

**表3.1 IP アドレスの説明と割り当て元**

| IP アドレス | 説明 | 割り当て元 |
| --- | --- | --- |
| **10.200.0.6** | コントローラーの仮想 IP アドレス | **dhcp_start** および **dhcp_end** の範囲の部分は、**undercloud.conf** ファイルで **10.200.0.5-10.200.0.24** に設定されます。 |
| **192.168.1.150** | パブリック IP アドレス | **network-environment.yaml** ファイルの **ExternalAllocationPools** 属性 |
| **172.16.0.10** | コントローラーノード上の OpenStack API サービスへのアクセスを提供します。 | **network-environment.yaml** ファイルの **InternalApiAllocationPools** |
| **172.16.0.11** | コントローラーノード上の Radis サービスへのアクセスを提供します。 | **network-environment.yaml** ファイルの **InternalApiAllocationPools** |
| **172.18.0.10** | Glance API および Swift プロキシーのサービスへのアクセスを提供するストレージの仮想 IP アドレス | **network-environment.yaml** ファイルの **StorageAllocationPools** 属性 |
| **172.19.0.10** | ストレージ管理へのアクセスを提供します。 | **network-environment.yaml** ファイルの **StorageMgmtAlloctionPools** |

- 特定の仮想 IP を使用するリソースの名前を指定して **pcs resource show** コマンドを実行して、その仮想 IP のアドレスを表示します (以下の例では **ip-192.168.1.150**)。

```
$ sudo pcs resource show ip-192.168.1.150
```

出力例:

```
Resource: ip-192.168.1.150 (class=ocf provider=heartbeat type=IPaddr2)
 Attributes: ip=192.168.1.150 cidr_netmask=32
 Operations: start interval=0s timeout=20s (ip-192.168.1.150-start-timeout-20s)
             stop interval=0s timeout=20s (ip-192.168.1.150-stop-timeout-20s)
             monitor interval=10s timeout=20s (ip-192.168.1.150-monitor-interval-10s)
```

## 3.5. 高可用性クラスターでの仮想 IP のネットワーク情報の表示

特定の仮想 IP (VIP) に割り当てられたコントローラーノードのネットワークインターフェイス情報および特定サービスのポート番号割り当てを表示できます。

**前提条件**

- 高可用性がデプロイされ、動作している。

**手順**

1. 表示する IP アドレスに割り当てられているコントローラーノードにログインし、ネットワークインターフェイスで **ip addr show** コマンドを実行します (以下の例では **vlan100**)。

```
$ ip addr show vlan100
```

出力例:

```
 9: vlan100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN
   link/ether be:ab:aa:37:34:e7 brd ff:ff:ff:ff:ff:ff
   inet *192.168.1.151/24* brd 192.168.1.255 scope global vlan100
      valid_lft forever preferred_lft forever
   inet *192.168.1.150/32* brd 192.168.1.255 scope global vlan100
      valid_lft forever preferred_lft forever
```

2. **netstat** コマンドを実行し、その IP アドレスをリッスンしているすべてのプロセスを表示します (以下の例では **192.168.1.150.haproxy**)。

```
$ sudo netstat -tupln | grep "192.168.1.150.haproxy"
```

出力例:

```
tcp      0      0 192.168.1.150:8778      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:8042      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:9292      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:8080      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:80        0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:8977      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:6080      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:9696      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:8000      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:8004      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:8774      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:5000      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:8776      0.0.0.0:*           LISTEN      61029/haproxy
tcp      0      0 192.168.1.150:8041      0.0.0.0:*           LISTEN      61029/haproxy
```

**注記**

**0.0.0.0** のように、すべてのローカルアドレスをリッスンしているプロセスは、**192.168.1.150** からも利用できます。これらのプロセスには、**sshd**、**mysqld**、**dhclient**、**ntpd** などがあります。

3. HA サービスの設定ファイルを開くことで、デフォルトのポート番号の割り当てとリッスンするサービスを確認します (以下の例では **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg**)。

   - TCP ポート 6080: **nova_novncproxy**
   - TCP ポート 9696: **neutron**
   - TCP ポート 8000: **heat_cfn**
   - TCP ポート 80: **horizon**
   - TCP ポート 8776: **cinder**
     この例では、**haproxy.cfg** ファイルで定義されているサービスの大半は、3 つすべてのコントローラーノードで **192.168.1.150** の IP アドレスをリッスンしています。ただし、**192.168.1.150** の IP アドレスを外部でリッスンしているのは **controller-0** ノードのみです。

     このため、**controller-0** ノードで異常が発生した場合には、HAProxy は **192.168.1.150** を別のコントローラーノードに再割り当てするだけで、他のサービスはすべてフォールバックコントローラーノードですでに実行されている状態となります。

## 3.6. フェンシングエージェントおよび PACEMAKER デーモンスのテータスの確認

Pacemaker が実行されている任意のノードで、フェンシングエージェントと Pacemaker デーモンのステータスを確認し、アクティブで実行中のコントローラーノードの数に関する情報を表示できます。

**前提条件**

- 高可用性がデプロイされ、動作している。

**手順**

1. **tripleo-admin** ユーザーとして任意のコントローラーノードにログインします。

```
$ ssh tripleo-admin@overcloud-controller-0
```

2. **pcs status** コマンドを実行します。

```
[tripleo-admin@overcloud-controller-0 ~]  $ sudo pcs status
```

出力例:

```
 my-ipmilan-for-controller-0 (stonith:fence_ipmilan): Started my-ipmilan-for-controller-0
my-ipmilan-for-controller-1 (stonith:fence_ipmilan): Started my-ipmilan-for-controller-1
my-ipmilan-for-controller-2 (stonith:fence_ipmilan): Started my-ipmilan-for-controller-2

PCSD Status:
 overcloud-controller-0: Online
 overcloud-controller-1: Online
 overcloud-controller-2: Online

Daemon Status:
```

```
  corosync: active/enabled
  pacemaker: active/enabled openstack-cinder-volume        (systemd:openstack-cinder-
volume):      Started overcloud-controller-0
  pcsd: active/enabled
```

この出力には、**pcs status** コマンドの出力の以下のセクションが表示されます。

- **my-ipmilan-for-controller**: 各コントローラーノードのフェンシングの種別 (**stonith:fence_ipmilan**) および IPMI サービスの稼働状態を表示します。
- **PCSD Status**: は、3 つすべてコントローラーノードが現在オンラインであることを示します。
- **Daemon Status**: **corosync**、**pacemaker**、および **pcsd** の 3 つの Pacemaker デーモンのステータスを示します。この例では、3 つすべてのサービスがアクティブかつ有効化されています。

## 3.7. 関連情報

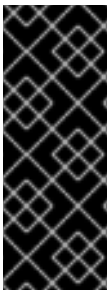
- 高可用性クラスターの設定および管理

# 第4章 STONITH を使用したコントローラーノードのフェンシング

フェンシングとは、クラスターとクラスターリソースを保護するために、異常が発生したノードを分離するプロセスのことです。フェンシングがないと、異常が発生したノードが原因でクラスター内のデータが破損する可能性があります。director は、Pacemaker を使用して、高可用性のコントローラーノードクラスターを提供します。

Pacemaker は、障害の発生したノードをフェンシングするのに STONITH というプロセスを使用します。STONITH は、Shoot the other node in the head の略です。STONITH はデフォルトでは無効化されているため、Pacemaker がクラスター内の各ノードの電源管理を制御できるように手動で設定する必要があります。

コントローラーノードがヘルスチェックに失敗すると、Pacemaker 指定のコーディネーター (DC) として機能するコントローラーノードは、Pacemaker **stonith** サービスを使用して影響を受けるコントローラーノードをフェンシングします。



**重要**

STONITH を使用しない高可用性オーバークラウドのデプロイはサポートの対象外です。高可用性オーバークラウドの Pacemaker クラスターの一部である各ノードには、STONITH デバイスを設定する必要があります。STONITH および Pacemaker の詳細は、Red Hat High Availability クラスターでのフェンシングの設定 および RHEL High Availability クラスターのサポートポリシー - フェンシング/STONITH の一般的な要件 を参照してください。

## 4.1. サポート対象のフェンシングエージェント

フェンシング機能と共に高可用性環境をデプロイする場合、環境のニーズに応じてフェンシングエージェントを選択することができます。フェンシングエージェントを変更するには、**fencing.yaml** ファイルに追加のパラメーターを設定する必要があります。

Red Hat OpenStack Platform (RHOSP) では、以下のフェンシングエージェントがサポートされています。

**Intelligent Platform Management Interface (IPMI)**

Red Hat OpenStack Platform (RHOSP) がフェンシングの管理に使用するデフォルトのフェンシングメカニズム

**STONITH Block Device (SBD)**

SBD (Storage-Based Death) デーモンは、Pacemaker とウォッチドッグデバイスを統合して、フェンシングがトリガーされる際および従来のフェンシングメカニズムが利用できない場合に、ノードを確実にシャットダウンします。

**重要**

- SBD フェンシングは、**pacemaker_remote** を使用するリモートベアメタルまたは仮想マシンノードを持つクラスターではサポートされません。そのため、デプロイメントでインスタンス HA が使用される場合はサポートされません。
- **fence_sbd** および **sbd poison-pill** フェンシングとブロックストレージデバイスの組み合わせはサポートされません。
- SBD フェンシングは、互換性のあるウォッチドッグデバイスでのみサポートされます。詳細は、Support Policies for RHEL High Availability Clusters - sbd and fence_sbd を参照してください。

**fence_kdump**

**kdump** クラッシュリカバリーサービスが設定されたデプロイメントで使用します。このエージェントを選択する場合、ダンプファイルを保存するのに十分なディスク容量を確保してください。
このエージェントは、IPMI、**fence_rhevm**、または Redfish フェンシングエージェントに追加する、セカンダリーメカニズムとして設定することができます。複数のフェンシングエージェントを設定する場合は、2 番目のエージェントが次のタスクを開始する前に、1 番目のエージェントがタスクを完了するのに十分な時間を割り当てるようにしてください。

**重要**

- RHOSP director は **fence_kdump** STONITH エージェントの設定のみをサポートします。フェンスエージェントが依存する完全な **kdump** サービスの設定はサポートされません。**kdump** サービスの設定に関する詳細は、Red Hat Pacemaker クラスターで fence_kdump を設定する方法 の記事を参照してください。
- Pacemaker ネットワークトラフィックインターフェイスが **ovs_bridges** または **ovs_bonds** ネットワークデバイスを使用する場合、**fence_kdump** はサポートされません。**fence_kdump** を有効にするには、ネットワークデバイスを **linux_bond** または **linux_bridge** に変更する必要があります。ネットワークインターフェイスの設定に関する詳細は、ネットワークインターフェイスリファレンス を参照してください。

**Redfish**

DMTF Redfish API をサポートするサーバーが設定されたデプロイメントで使用します。このエージェントを指定するには、**fencing.yaml** ファイルで **agent** パラメーターの値を **fence_redfish** に変更します。Redfish についての詳細は、DTMF のドキュメント を参照してください。

**Red Hat Virtualization (RHV) 用fence_rhevm**

RHV 環境で実行されるコントローラーノードのフェンシングを設定するのに使用します。IPMI フェンシングの場合と同じ様に **fencing.yaml** ファイルを生成することができますが、RHV を使用するには、**nodes.json** ファイルで **pm_type** パラメーターを定義する必要があります。
デフォルトでは、**ssl_insecure** パラメーターは自己署名証明書を受け入れるように設定されます。セキュリティー要件に応じて、パラメーターの値を変更することができます。

**重要**

**UserVMManager** などの RHV で仮想マシンを作成して起動する権限を持つロールを使用するようにしてください。

**マルチレイヤーフェンシング**

複雑なフェンシングのユースケースに対応するために、複数のフェンシングエージェントを設定することができます。たとえば、**fence_kdump** と共に IPMI フェンシングを設定することができます。Pacemaker が各メカニズムをトリガーする順番は、フェンシングエージェントの順序により決まります。

**関連情報**

- 「オーバークラウドへのフェンシングのデプロイ」
- 「オーバークラウドでのフェンシングのテスト」
- 「フェンシングパラメーター」

## 4.2. オーバークラウドへのフェンシングのデプロイ

オーバークラウドにフェンシングをデプロイするには、最初に STONITH および Pacemaker の状態を確認して、**fencing.yaml** ファイルを設定します。続いて、オーバークラウドをデプロイし、追加のパラメーターを設定します。最後に、フェンシングがオーバークラウドに正しくデプロイされていることを確認します。

**前提条件**

- デプロイメントに適したフェンシングエージェントを選択します。サポート対象のフェンシングエージェントの一覧は、「サポート対象のフェンシングエージェント」 を参照してください。
- director にノードを登録した際に作成した **nodes.json** ファイルにアクセスできるようにしてください。このファイルは、デプロイメント中に生成する **fencing.yaml** ファイルに必須なインプットになります。
- **nodes.json** ファイルにノード上のいずれかのネットワークインターフェイス (NIC) の MAC アドレスが含まれている必要があります。詳細は、オーバークラウドのノード登録 を参照してください。
- Red Hat Virtualization (RHV) フェンシングエージェントを使用する場合は、**UserVMManager** などの仮想マシンを管理する権限を持つロールを使用します。

**手順**

1. **tripleo-admin** ユーザーとして各コントローラーノードにログインします。
2. クラスターが動作状態にあることを確認します。

```
$ sudo pcs status
```

出力例:

```
Cluster name: openstackHA
Last updated: Wed Jun 24 12:40:27 2015
Last change: Wed Jun 24 11:36:18 2015
Stack: corosync
Current DC: lb-c1a2 (2) - partition with quorum
```

```
Version: 1.1.12-a14efad
3 Nodes configured
141 Resources configured
```

3. STONITH が無効になっていることを確認します。

```
$ sudo pcs property show
```

出力例:

```
Cluster Properties:
cluster-infrastructure: corosync
cluster-name: openstackHA
dc-version: 1.1.12-a14efad
have-watchdog: false
stonith-enabled: false
```

4. 使用するフェンシングエージェントに応じて、以下のオプションのいずれかを選択します。

- IPMI または RHV のフェンシングエージェントを使用する場合は、**fencing.yaml** 環境ファイルを生成します。

```
(undercloud) $ openstack overcloud generate fencing --output fencing.yaml nodes.json
```

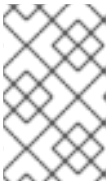
**注記**

このコマンドにより、**ilo** および **drac** の電源管理情報が等価な IPMI 版に変換されます。

- STONITH Block Device (SBD)、**fence_kdump**、あるいは Redfish 等の別のフェンシングエージェントを使用する場合、または事前にプロビジョニングされたノードを使用する場合は、手動で **fencing.yaml** ファイルを作成します。

5. SBD フェンシングのみ: 以下のパラメーターを **fencing.yaml** ファイルに追加します。

```
parameter_defaults:
  ExtraConfig:
    pacemaker::corosync::enable_sbd: true
```



**注記**

このステップは、初回のオーバークラウドデプロイメントにのみ適用されます。既存のオーバークラウドで SBD フェンシングを有効にする方法の詳細は、RHEL 7 と RHEL 8 での sbd フェンシングの有効化 を参照してください。

6. マルチレイヤーフェンシングのみ: 生成された **fencing.yaml** ファイルにレベル固有のパラメーターを追加します。

```
parameter_defaults:
  EnableFencing: true
  FencingConfig:
    devices:
      level1:
```

```
    - agent: [VALUE]
      host_mac: aa:bb:cc:dd:ee:ff
      params:
        <parameter>: <value>
    level2:
    - agent: fence_agent2
      host_mac: aa:bb:cc:dd:ee:ff
      params:
        <parameter>: <value>
```

**<parameter>** および **<value>** を、フェンシングエージェントが必要とする実際のパラメーターおよび値に置き換えてください。

7. **fencing.yaml** ファイルおよびデプロイメントに該当するその他の環境ファイルを指定して、**overcloud deploy** コマンドを実行します。

```
openstack overcloud deploy --templates \
-e /usr/share/openstack-tripleo-heat-templates/environments/network-isolation.yaml \
-e ~/templates/network-environment.yaml \
-e ~/templates/storage-environment.yaml --ntp-server pool.ntp.org --neutron-network-type
vxlan --neutron-tunnel-types vxlan \
-e fencing.yaml
```

8. SBD フェンシングのみ: ウォッチドッグタイマーデバイスの間隔を設定し、間隔が正しく設定されていることを確認します。

```
# pcs property set stonith-watchdog-timeout=<interval>
# pcs property show
```

**検証**

1. **heat-admin** ユーザーとしてオーバークラウドにログインし、Pacemaker がリソースマネージャーとして設定されていることを確認します。

```
$ source stackrc
$ openstack server list | grep controller
$ ssh tripleo-admin@<controller-x_ip>
$ sudo pcs status | grep fence
stonith-overcloud-controller-x (stonith:fence_ipmilan): Started overcloud-controller-y
```

この例では、Pacemaker は、**fencing.yaml** ファイルで指定された各コントローラーノードに STONITH リソースを使用するように設定されています。

**注記**

制御するのと同じノードに **fence-resource** プロセスを設定することはできません。

2. フェンシングリソースの属性を確認します。STONITH 属性の値は、**fencing.yaml** ファイルの値と一致している必要があります。

```
$ sudo pcs stonith show <stonith-resource-controller-x>
```

**関連情報**

- 「オーバークラウドでのフェンシングのテスト」
- 「フェンシングパラメーター」
- RHEL High Availability のコンポーネントの探索 - sbd および fence_sbd

## 4.3. オーバークラウドでのフェンシングのテスト

フェンシングが正しく機能することをテストするには、コントローラーノード上の全ポートを閉じ、サーバーを再起動してフェンシングをトリガーします。

**重要**

以下の手順では、コントローラーノードへの接続を意図的にすべて切断するので、ノードが再起動します。

**前提条件**

- フェンシングがオーバークラウドにデプロイされ、実行されている。フェンシングのデプロイ方法は、「オーバークラウドへのフェンシングのデプロイ」 を参照してください。
- 再起動にコントローラーノードを使用することができる。

**手順**

1. **stack** ユーザーとしてコントローラーノードにログインし、source コマンドで認証情報ファイルを読み込みます。

```
$ source stackrc
$ openstack server list | grep controller
$ ssh tripleo-admin@<controller-x_ip>
```

2. **root** ユーザーに切り替え、コントローラーノードへの接続をすべて閉じます。

```
$ sudo -i
iptables -A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT &&
iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 22 -j ACCEPT &&
iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 5016 -j ACCEPT &&
iptables -A INPUT -p udp -m state --state NEW -m udp --dport 5016 -j ACCEPT &&
iptables -A INPUT ! -i lo -j REJECT --reject-with icmp-host-prohibited &&
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT &&
iptables -A OUTPUT -p tcp --sport 5016 -j ACCEPT &&
iptables -A OUTPUT -p udp --sport 5016 -j ACCEPT &&
iptables -A OUTPUT ! -o lo -j REJECT --reject-with icmp-host-prohibited
```

3. 別のコントローラーノードから、Pacemaker のログファイルでフェンシングイベントの有無を確認します。

```
$ ssh tripleo-admin@<controller-x_ip>
$ less /var/log/cluster/corosync.log
(less): /fenc*
```

STONITH サービスがコントローラーでフェンシングアクションを実行していれば、ログファイルにフェンシングイベントが記録されます。

4. 数分間待ってから **pcs status** コマンドを実行して、再起動したコントローラーノードがクラスター内で再度実行されていることを確認します。再起動したコントローラーノードが出力に表示される場合には、フェンシングが正しく機能しています。

## 4.4. STONITH デバイス情報の表示

STONITH がフェンシングデバイスをどのように設定するかを確認するには、オーバークラウドから **pcs stonith show --full** コマンドを実行します。

**前提条件**

- フェンシングがオーバークラウドにデプロイされ、実行されている。フェンシングのデプロイ方法は、「オーバークラウドへのフェンシングのデプロイ」 を参照してください。

**手順**

- コントローラーノードのリストと、その STONITH デバイスのステータスを表示します。

```
$ sudo pcs stonith show --full
 Resource: my-ipmilan-for-controller-0 (class=stonith type=fence_ipmilan)
  Attributes: pcmk_host_list=overcloud-controller-0 ipaddr=10.100.0.51 login=admin
passwd=abc lanplus=1 cipher=3
  Operations: monitor interval=60s (my-ipmilan-for-controller-0-monitor-interval-60s)
 Resource: my-ipmilan-for-controller-1 (class=stonith type=fence_ipmilan)
  Attributes: pcmk_host_list=overcloud-controller-1 ipaddr=10.100.0.52 login=admin
passwd=abc lanplus=1 cipher=3
  Operations: monitor interval=60s (my-ipmilan-for-controller-1-monitor-interval-60s)
 Resource: my-ipmilan-for-controller-2 (class=stonith type=fence_ipmilan)
  Attributes: pcmk_host_list=overcloud-controller-2 ipaddr=10.100.0.53 login=admin
passwd=abc lanplus=1 cipher=3
  Operations: monitor interval=60s (my-ipmilan-for-controller-2-monitor-interval-60s)
```

この出力には、リソースごとに以下の情報が表示されます。

- フェンシングデバイスが必要に応じてマシンの電源をオン/オフするのに使用する IPMI 電源管理サービス (例: **fence_ipmilan**)
- IPMI インターフェイスの IP アドレス (例: **10.100.0.51**)
- ログインに使用するユーザー名 (例: **admin**)
- ノードにログインするのに使用するパスワード (例: **abc**)
- それぞれのホストをモニターする間隔 (例: **60** 秒)

## 4.5. フェンシングパラメーター

オーバークラウドにフェンシングをデプロイする際には、フェンシングを設定するのに必要なパラメーターを定義して **fencing.yaml** ファイルを生成します。

**fencing.yaml** 環境ファイルの設定例を以下に示します。

```
parameter_defaults:
  EnableFencing: true
  FencingConfig:
    devices:
    - agent: fence_ipmilan
      host_mac: 11:11:11:11:11:11
      params:
        ipaddr: 10.0.0.101
        lanplus: true
        login: admin
        passwd: InsertComplexPasswordHere
        pcmk_host_list: host04
        privlvl: administrator
```

このファイルには以下のパラメーターが含まれます。

**EnableFencing**

Pacemaker の管理するノードのフェンシング機能を有効にします。

**FencingConfig**

フェンシングデバイスおよび各デバイスのパラメーターをリスト表示します。

- **agent**: フェンシングエージェント名。
- **host_mac**: サーバー上のプロビジョニングインターフェイスまたはその他のネットワークインターフェイスの小文字の MAC アドレス。これをフェンシングデバイスの一意の識別子として使用できます。

**重要**

IPMI インターフェイスの MAC アドレスは使用しないでください。

- **params**: フェンスデバイスパラメーターのリスト。

**フェンシングデバイスのパラメーター**

フェンシングデバイスのパラメーターのリストを表示します。IPMI フェンシングエージェントのパラメーターを以下の例に示します。

- **auth**: IPMI 認証種別 (**md5**、**password**、または none)
- **ipaddr**: IPMI IP アドレス
- **ipport**: IPMI ポート
- **login**: IPMI デバイス用のユーザー名
- **passwd**: IPMI デバイス用のパスワード
- **lanplus**: lanplus を使用して、接続のセキュリティーを向上させます。
- **privlvl**: IPMI デバイスの権限レベル
- **pcmk_host_list**: Pacemaker ホストのリスト

**関連情報**

- 「オーバークラウドへのフェンシングのデプロイ」
- 「サポート対象のフェンシングエージェント」

## 4.6. 関連情報

- Red Hat High Availability クラスターでのフェンシングの設定

# 第5章 HAPROXY を使用したトラフィック負荷の分散

HAProxy サービスは、トラフィックの負荷を高可用性クラスター内のコントローラーノードに分散する機能に加えて、ロギングおよびサンプル設定を提供します。**haproxy** パッケージに含まれる **haproxy** デーモンは、同じ名前の **systemd** サービスに対応します。Pacemaker は、HAProxy サービスを **haproxy-bundle** と呼ばれる高可用性サービスとして管理します。

## 5.1. HAPROXY の仕組み

director は、ほとんどの Red Hat OpenStack Platform サービスを HAProxy サービスを使用するように設定することができます。Director は、これらのサービスを **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** ファイルで設定します。このファイルは、HAProxy が各オーバークラウドノードの専用のコンテナーで実行されるように指示します。

HAProxy が管理するサービスのリストを以下の表に示します。

**表5.1 HAProxy が管理するサービス**

| | | | |
|---|---|---|---|
| aodh | cinder | glance_api | gnocchi |
| haproxy.stats | heat_api | heat_cfn | horizon |
| keystone_admin | keystone_public | mysql | neutron |
| nova_metadata | nova_novncproxy | nova_osapi | nova_placement |

**haproxy.cfg** ファイル内の各サービスで、以下の属性が設定されます。

- **listen**: 要求をリッスンするサービスの名前
- **bind**: サービスがリッスンする IP アドレスおよび TCP ポート番号
- **server**: HAProxy を使用する各コントローラーノードサーバーの名前、IP アドレスおよびリッスンするポート、ならびにサーバーに関する追加情報

以下の例は、**haproxy.cfg** ファイル内の OpenStack Block Storage (cinder) サービスの設定を示しています。

```
listen cinder
  bind 172.16.0.10:8776
  bind 192.168.1.150:8776
  mode http
  http-request set-header X-Forwarded-Proto https if { ssl_fc }
  http-request set-header X-Forwarded-Proto http if !{ ssl_fc }
  option httpchk
  server overcloud-controller-0 172.16.0.13:8777 check fall 5 inter 2000 rise 2
  server overcloud-controller-1 172.16.0.14:8777 check fall 5 inter 2000 rise 2
  server overcloud-controller-2 172.16.0.15:8777 check fall 5 inter 2000 rise 2
```

この出力例は、OpenStack Block Storage (cinder) サービスに関する以下の情報を示しています。

- **172.16.0.10:8776**: オーバークラウド内で使用する内部 API ネットワーク (VLAN201) 上の仮想 IP アドレスおよびポート

- **192.168.1.150:8776**: オーバークラウド外から API ネットワークへのアクセスを提供する外部ネットワーク (VLAN100) 上の仮想 IP アドレスおよびポート
- **8776**: OpenStack Block Storage (cinder) サービスがリッスンしているポート番号
- **server**: コントローラーノード名および IP アドレス。HAProxy は、これらの IP アドレスに送信された要求を **server** の出力にリスト表示されるコントローラーノードのいずれかに転送することができます。
- **httpchk**: コントローラーノードサーバーでのヘルスチェックを有効にします。
- **fall 5**: サービスがオフラインであると判断されるヘルスチェックの失敗回数
- **inter 2000**: 連続する 2 回のヘルスチェックの間隔 (ミリ秒単位)
- **rise 2**: サービスが動作中であると判断されるヘルスチェックの成功回数

**haproxy.cfg** ファイルで使用できる設定の詳細は、**haproxy** パッケージがインストールされている任意のノードの **/usr/share/doc/haproxy-[VERSION]/configuration.txt** ファイルを参照してください。

## 5.2. HAPROXY STATS の表示

すべての HA デプロイメントにおいて、director によりデフォルトで HAProxy Stats (統計) も有効になります。この機能により、データ転送、接続、およびサーバーの状態についての詳細情報を HAProxy Stats のページで確認することができます。

director は、HAProxy Stats ページにアクセスするのに使用する **IP:Port** アドレスも設定し、その情報を **haproxy.cfg** ファイルに保存します。

**前提条件**

- 高可用性がデプロイされ、動作している。

**手順**

1. HAProxy がインストールされている任意のコントローラーノードで **/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg** ファイルを開きます。
2. **listen haproxy.stats** セクションを探します。

```
listen haproxy.stats
  bind 10.200.0.6:1993
  mode http
  stats enable
  stats uri /
  stats auth admin:<haproxy-stats-password>
```

3. Web ブラウザーで 10.200.0.6:1993 に移動し、**stats auth** 行の認証情報を入力して HAProxy Stats ページを表示します。

## 5.3. 関連情報

- haproxy 1.8 のドキュメント

- haproxy.cfg が OpenStack のサービスをロードバランシングできるよう正しく設定されているかどうか、確認する方法はありますか？

# 第6章 GALERA を使用したデータベースレプリケーションの管理

Red Hat OpenStack Platform では、データベースレプリケーションの管理に MariaDB Galera Cluster を使用します。Pacemaker は、データベースのマスター/スレーブのステータスを管理するバンドルセットリソースとして、Galera サービスを実行します。Galera を使用して、ホスト名の解決、クラスターの整合性、ノードの整合性、データベースレプリケーションのパフォーマンス等、データベースクラスターのさまざまな要素をテストおよび検証することができます。

データベースクラスターの整合性を詳しく調べる際には、各ノードは以下の条件を満たしている必要があります。

- ノードが正しいクラスターの一部である。
- ノードがクラスターに書き込み可能である。
- ノードがクラスターからのクエリーおよび書き込みコマンドを受け取ることができる。
- ノードがクラスター内の他のノードに接続されている。
- ノードが Write Set をローカルデータベースのテーブルにレプリケーションしている。

## 6.1. MARIADB クラスターでのホスト名の解決の確認

MariaDB Galera クラスターのトラブルシューティングを行うには、まずホスト名解決の問題を取り除き、続いて各コントローラーノードのデータベースで Write Set のレプリケーションステータスを確認します。MySQL データベースにアクセスするには、オーバークラウドのデプロイメント時に director が設定したパスワードを使用します。

デフォルトでは、director は Galera リソースを IP アドレスにではなくホスト名にバインドします。そのため、ホスト名の解決を妨げる問題 (例: DNS の設定不良や異常など) が原因で、Pacemaker による Galera リソースの管理が不適切になる場合があります。

**手順**

1. コントローラーノードから **hiera** コマンドを実行し、MariaDB データベースの root パスワードを取得します。

```
$ sudo hiera -c /etc/puppet/hiera.yaml "mysql::server::root_password"
*[MYSQL-HIERA-PASSWORD]*
```

2. ノードで実行される MariaDB コンテナーの名前を取得します。

```
$ sudo podman ps | grep -i galera
a403d96c5026  undercloud.ctlplane.localdomain:8787/rhosp-rhel9/openstack-mariadb:16.0-
106           /bin/bash /usr/lo...  3 hours ago  Up 3 hours ago         galera-bundle-podman-0
```

3. 各ノードの MariaDB データベースから、Write Set のレプリケーション情報を取得します。

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-
PASSWORD]" -e "SHOW GLOBAL STATUS LIKE 'wsrep_%';"
    +----------------------------+----------+
    | Variable_name              | Value    |
    +----------------------------+----------+
    | wsrep_applier_thread_count | 1        |
```

```
| wsrep_apply_oooe           | 0.018672 |
| wsrep_apply_oool           | 0.000630 |
| wsrep_apply_window         | 1.021942 |
| ...                        | ...      |
+----------------------------+----------+
```

関連する変数はそれぞれ **wsrep** の接頭辞を使用します。

4. クラスターが正しいノード数を報告することを確認して、MariaDB Galera Cluster の健全性および整合性を検証します。

## 6.2. MARIADB クラスターの整合性の確認

MariaDB Galera Cluster の問題を調べるには、各コントローラーノードで特定の **wsrep** データベース変数を調べて、クラスター全体の整合性を確認します。

**手順**

- 以下のコマンドを実行します。その際、**<variable>** を確認する **wsrep** データベース変数に置き換えます。

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW GLOBAL STATUS LIKE <variable>;"
```

以下の例で、ノードのクラスター状態の UUID を表示する方法を説明します。

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW GLOBAL STATUS LIKE 'wsrep_cluster_state_uuid';"

+--------------------------+--------------------------------------+
| Variable_name            | Value                                |
+--------------------------+--------------------------------------+
| wsrep_cluster_state_uuid | e2c9a15e-5485-11e0-0800-6bbb637e7211 |
+--------------------------+--------------------------------------+
```

以下の表には、クラスターの整合性を確認するのに使用できる **wsrep** データベース変数をまとめています。

**表6.1 クラスターの整合性を確認するためのデータベース変数**

| 変数 | 概要 | 説明 |
|---|---|---|
| **wsrep_cluster_state_uuid** | クラスターの状態の UUID | ノードが属するクラスターの ID。全ノードに同一のクラスター ID が割り当てられている必要があります。異なる ID が割り当てられたノードは、クラスターに接続できません。 |

| 変数 | 概要 | 説明 |
| --- | --- | --- |
| **wsrep_cluster_size** | クラスター内のノード数 | 任意のノードで確認することができます。値が実際のノード数を下回る場合には、いずれかのノードで異常が発生したか、接続が失われたことを意味します。 |
| **wsrep_cluster_conf_id** | クラスターの全変更回数 | クラスターが複数のコンポーネント (パーティション) に分割されたかどうかを判断します。通常、ネットワーク障害が原因でパーティションが発生します。全ノードで値が同一でなければなりません。<br><br>異なる **wsrep_cluster_conf_id** を報告するノードがある場合には、**wsrep_cluster_status** の値をチェックして、ノードがクラスターにまだ書き込み可能かどうかを確認します (**Primary**)。 |
| **wsrep_cluster_status** | Primary コンポーネントのステータス | ノードがクラスターに書き込み可能かどうかを判断します。ノードがクラスターに書き込み可能であれば、**wsrep_cluster_status** の値は **Primary** になります。それ以外の値の場合には、ノードが稼働していないパーティションの一部であることを示します。 |

## 6.3. MARIADB クラスターでのデータベースノードの整合性の確認

MariaDB Galera Cluster の特定のコントローラーノードに関する問題を調査するには、特定の **wsrep** データベース変数を確認してノードの整合性を確認します。

**手順**

- 以下のコマンドを実行します。その際、**<variable>** を確認する **wsrep** データベース変数に置き換えます。

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW GLOBAL STATUS LIKE <variable>;"
```

以下の表には、ノードの整合性を確認するのに使用できる **wsrep** データベース変数をまとめています。

**表6.2 ノードの整合性を確認するためのデータベース変数**

| 変数 | 概要 | 説明 |
| --- | --- | --- |
| **wsrep_ready** | ノードがクエリーを受け入れる能力 | ノードがクラスターから Write Set を受け入れることができるかどうかを示します。その場合には、**wsrep_ready** は **ON** になります。 |
| **wsrep_connected** | ノードのネットワーク接続性 | ノードがネットワーク上の他のノードに接続できるかどうかを示します。その場合には、**wsrep_connected** は **ON** になります。 |
| **wsrep_local_state_comment** | ノードの状態 | ノードの状態の概要を示します。ノードがクラスターに書き込み可能であれば、**wsrep_local_state_comment** の一般的な値は **Joining**、**Waiting on SST**、**Joined**、**Synced**、または **Donor** です。<br><br>ノードが稼働していないコンポーネントの一部の場合には、**wsrep_local_state_comment** の値は **Initialized** になります。 |

**注記**

- ノードがクラスター内のノードのサブセットにしか接続されている場合でも、**wsrep_connected** の値は **ON** になります。たとえば、クラスターのパーティションの場合には、そのノードは、クラスターに書き込みができないコンポーネントの一部となっている可能性があります。クラスターの整合性の確認に関する詳細は、「MariaDB クラスターの整合性の確認」 を参照してください。
- **wsrep_connected** 値が **OFF** の場合、ノードはどのクラスターコンポーネントにも接続されていません。

## 6.4. MARIADB クラスターでのデータベースレプリケーションのパフォーマンスのテスト

MariaDB Galera Cluster のパフォーマンスを確認するには、特定の **wsrep** データベース変数をチェックして、クラスターのレプリケーションスループットでベンチマークテストを実行します。

これらの変数の 1 つのクエリーを行うたびに、**FLUSH STATUS** コマンドは変数の値をリセットします。ベンチマークテストを行うには、複数のクエリーを実行し、差異を分析する必要があります。この差異は、Flow Control がクラスターのパフォーマンスにどの程度影響を及ぼしているかを判断するのに役立ちます。

Flow Control は、クラスターがレプリケーションを制御するのに使用するメカニズムです。ローカルの

受信キューが一定のしきい値を超えると、キューのサイズが減少するまで、Flow Control はレプリケーションを一時停止します。Flow Control の詳細は、Galera Cluster の Web サイトで Flow Control を参照してください。

**手順**

- 以下のコマンドを実行します。その際、**<variable>** を確認する **wsrep** データベース変数に置き換えます。

```
$ sudo podman exec galera-bundle-podman-0 sudo mysql -B --password="[MYSQL-HIERA-PASSWORD]" -e "SHOW STATUS LIKE <variable>;"
```

以下の表には、データベースレプリケーションのパフォーマンスをテストするのに使用できるさまざまな **wsrep** データベース変数をまとめています。

**表6.3 データベースレプリケーションのパフォーマンスを確認するためのデータベース変数**

| 変数 | 概要 | 使用方法 |
|---|---|---|
| **wsrep_local_recv_queue_avg** | 最後のクエリー後のローカル受信 Write Set キューの平均サイズ | 値が 0.0 を超えていれば、ノードが受信速度に対応して Write Set を適用できないことが分かります。この場合、レプリケーションのスロットリングがトリガーされます。このベンチマークの詳しい情報は、**wsrep_local_recv_queue_min** と **wsrep_local_recv_queue_max** を確認してください。 |
| **wsrep_local_send_queue_avg** | 最後のクエリー後の送信キューの平均長さ | 値が 0.0 を超えていれば、レプリケーションのスロットルおよびネットワークスループットの問題の可能性が高いことが分かります。 |
| **wsrep_local_recv_queue_min** および **wsrep_local_recv_queue_max** | 最後のクエリー後のローカル受信キューの最小/最大サイズ | **wsrep_local_recv_queue_avg** の値が 0.0 を超える場合、これらの変数を確認してキューサイズの範囲を判断することができます。 |

| 変数 | 概要 | 使用方法 |
| --- | --- | --- |
| **wsrep_flow_control_paused** | 最後のクエリー以降、Flow Control がノードを一時停止した時間の割合 | 値が 0.0 を超えていれば、Flow Control がノードを一時停止したことが分かります。一時停止の時間を把握するには、**wsrep_flow_control_paused** の値をクエリー間の秒数で乗算します。可能な限り 0.0 に近い値が最適値です。<br><br>以下に例を示します。<br><br>● 最後のクエリーから 1 分後の **wsrep_flow_control_paused** の値が 0.50 の場合、Flow Control は 30 秒間ノードを一時停止しています。<br><br>● 最後のクエリーから 1 分後に **wsrep_flow_control_paused** の値が 1.0 の場合、Flow Control はその 1 分間にわたってノードを一時停止します。 |
| **wsrep_cert_deps_distance** | 並行して適用することのできるシーケンス番号 (**seqno**) の最小値と最大値の差の平均 | スロットリングおよび一時停止の場合、この変数は並行して適用することのできる Write Set 数の平均を示します。この値を **wsrep_slave_threads** 変数と比較して、実際に同時に適用することのできる Write Set の数を確認します。 |

| 変数 | 概要 | 使用方法 |
| --- | --- | --- |
| **wsrep_slave_threads** | 同時に適用することのできるスレッドの数 | この変数の値を増やして、より多くのスレッドを同時に適用することができます。これにより、**wsrep_cert_deps_distance** の値も増加します。**wsrep_slave_threads** の値は、ノードの CPU コア数よりも大きくすることはできません。<br><br>たとえば、**wsrep_cert_deps_distance** の値が **20** の場合は、**wsrep_slave_threads** の値を **2** から **4** の値を増やして、ノードを適用することができる write set の量を増やすことができます。<br><br>問題のあるノードの **wsrep_slave_threads** の値がすでに最適値である場合、接続性の問題を調査する際に、ノードをクラスターから除外することができます。 |

## 6.5. 関連情報

- MariaDB Galera クラスターとは何ですか?

# 第7章 高可用性リソースに関するトラブルシューティング

リソースに異常が発生した場合は、問題の原因と場所を調査し、異常が発生したリソースを修正し、必要に応じてリソースをクリーンアップする必要があります。デプロイメントによって、リソース異常にはさまざまな原因が考えられ、リソースを調査して問題の修正方法を決定する必要があります。

たとえば、リソースの制約を確認することで、リソースが相互に障害とならないようにし、また互いに接続できるようにすることができます。他のコントローラーノードよりも頻繁にフェンシングされるコントローラーノードを調査し、通信の問題を識別できる場合もあります。

リソースの問題の場所に応じて、以下のオプションのいずれかを選択します。

**コントローラーノードの問題**

コントローラーノードのヘルスチェックに失敗した場合は、コントローラーノード間の通信に問題があることを示しています。問題を調査するには、コントローラーノードにログインして、サービスが正常に起動できるかどうかを確認します。

**個別のリソースの問題**

コントローラーのほとんどのサービスが正しく動作している場合は、**pcs status** コマンドを実行し、出力で特定の Pacemaner リソース障害に関する情報がないか、あるいは、**systemctl** コマンドを実行し、Pacemaker 以外のリソースのエラーを調べます。

## 7.1. 高可用性クラスターでのリソースの制約の表示

リソースの問題を調査する前に、サービスがどのように起動されるかに関する制約を表示することができます。これには、各リソースが配置される場所、リソースが起動される順序、別のリソースと共に配置する必要があるかどうか、などの制約が含まれます。

**手順**

- 以下のオプションのいずれかを使用します。
  - リソースの制約をすべて表示するには、任意のコントローラーノードにログインして **pcs constraint show** コマンドを実行します。

```
$ sudo pcs constraint show
```

以下の例には、Controller ノードで **pcs constraint show** コマンドを実行した場合の出力を示しており、途中省略しています。

```
Location Constraints:
  Resource: galera-bundle
    Constraint: location-galera-bundle (resource-discovery=exclusive)
      Rule: score=0
        Expression: galera-role eq true
  [...]
  Resource: ip-192.168.24.15
    Constraint: location-ip-192.168.24.15 (resource-discovery=exclusive)
      Rule: score=0
        Expression: haproxy-role eq true
  [...]
  Resource: my-ipmilan-for-controller-0
    Disabled on: overcloud-controller-0 (score:-INFINITY)
  Resource: my-ipmilan-for-controller-1
    Disabled on: overcloud-controller-1 (score:-INFINITY)
```

```
  Resource: my-ipmilan-for-controller-2
    Disabled on: overcloud-controller-2 (score:-INFINITY)
Ordering Constraints:
  start ip-172.16.0.10 then start haproxy-bundle (kind:Optional)
  start ip-10.200.0.6 then start haproxy-bundle (kind:Optional)
  start ip-172.19.0.10 then start haproxy-bundle (kind:Optional)
  start ip-192.168.1.150 then start haproxy-bundle (kind:Optional)
  start ip-172.16.0.11 then start haproxy-bundle (kind:Optional)
  start ip-172.18.0.10 then start haproxy-bundle (kind:Optional)
Colocation Constraints:
  ip-172.16.0.10 with haproxy-bundle (score:INFINITY)
  ip-172.18.0.10 with haproxy-bundle (score:INFINITY)
  ip-10.200.0.6 with haproxy-bundle (score:INFINITY)
  ip-172.19.0.10 with haproxy-bundle (score:INFINITY)
  ip-172.16.0.11 with haproxy-bundle (score:INFINITY)
  ip-192.168.1.150 with haproxy-bundle (score:INFINITY)
```

この出力には、以下の主要な制約種別が表示されています。

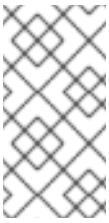
Location Constraints

リソースを割り当てることのできる場所をリスト表示します。

- 最初の制約は、**galera-role** 属性が **true** に設定されたノードで実行する **galera-bundle** リソースを設定するルールを定義します。
- 場所に関する 2 番目の制約は、IP リソース **ip-192.168.24.15** は **haproxy-role** 属性が **true** に設定されたノードでのみ実行されることを指定します。これは、クラスターが IP アドレスを **haproxy** サービスに関連付けることを意味し、サービスを到達可能にするために必要です。
- 場所に関する 3 番目の制約は、**ipmilan** リソースが各コントローラーノードで無効化されることを意味します。

Ordering Constraints

リソースを起動することのできる順序をリスト表示します。この例は、仮想 IP アドレスリソース **IPaddr2** は HAProxy サービスより先に起動しなければならない、という制約を示しています。



**注記**

順序に関する制約は、IP アドレスリソースおよび HAProxy にのみ適用されます。Compute などのサービスは、Galera などの依存関係にあるサービスの中断に対する耐性があると予想されるため、その他すべてのリソースは systemd により管理されます。

Colocation Constraints

共に配置する必要があるリソースをリスト表示します。すべての仮想 IP アドレスは **haproxy-bundle** リソースにリンクされています。

- 特定のリソースの制約を表示するには、任意のコントローラーノードにログインして **pcs property show** コマンドを実行します。

```
$ sudo pcs property show
```

出力例:

```
Cluster Properties:
 cluster-infrastructure: corosync
 cluster-name: tripleo_cluster
 dc-version: 2.0.1-4.el9-0eb7991564
 have-watchdog: false
 redis_REPL_INFO: overcloud-controller-0
 stonith-enabled: false
Node Attributes:
 overcloud-controller-0: cinder-volume-role=true galera-role=true haproxy-role=true
rabbitmq-role=true redis-role=true rmq-node-attr-last-known-
rabbitmq=rabbit@overcloud-controller-0
 overcloud-controller-1: cinder-volume-role=true galera-role=true haproxy-role=true
rabbitmq-role=true redis-role=true rmq-node-attr-last-known-
rabbitmq=rabbit@overcloud-controller-1
 overcloud-controller-2: cinder-volume-role=true galera-role=true haproxy-role=true
rabbitmq-role=true redis-role=true rmq-node-attr-last-known-
rabbitmq=rabbit@overcloud-controller-2
```

この出力で、リソースの制約が正しく設定されていることを確認できます。たとえば、すべてのコントローラーノードで **galera-role** 属性は **true** であり、**galera-bundle** リソースはこれらのノードでのみ実行されることを意味します。

## 7.2. PACEMAKER リソースの問題の調査

Pacemaker が管理するリソースの異常を調査するには、リソースに異常が発生しているコントローラーノードにログインして、リソースのステータスおよびログイベントを確認します。たとえば、**openstack-cinder-volume** リソースのステータスおよびログイベントを調べます。

**前提条件**

- Pacemaker サービスが含まれるコントローラーノード
- ログイベントを表示するための root ユーザーのアクセス権限

**手順**

1. リソースに異常が発生しているコントローラーノードにログインします。
2. **pcs status** コマンドに **grep** オプションを付けて実行して、サービスのステータスを確認します。

```
# sudo pcs status | grep cinder
  Podman container: openstack-cinder-volume [192.168.24.1:8787/rh-osbs/rhosp161-
openstack-cinder-volume:pcmklatest]
    openstack-cinder-volume-podman-0     (ocf::heartbeat:podman):        Started controller-1
```

3. リソースのログイベントを表示します。

```
# sudo less /var/log/containers/stdouts/openstack-cinder-volume.log
[...]
2021-04-12T12:32:17.607179705+00:00 stderr F ++ cat /run_command
2021-04-12T12:32:17.609648533+00:00 stderr F + CMD='/usr/bin/cinder-volume --config-file
```

```
/usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf'
2021-04-12T12:32:17.609648533+00:00 stderr F + ARGS=
2021-04-12T12:32:17.609648533+00:00 stderr F + [[ ! -n '' ]]
2021-04-12T12:32:17.609648533+00:00 stderr F + . kolla_extend_start
2021-04-12T12:32:17.611214130+00:00 stderr F +++ stat -c %U:%G /var/lib/cinder
2021-04-12T12:32:17.616637578+00:00 stderr F ++ [[ cinder:kolla != \c\i\n\d\e\r\:\k\o\l\l\a ]]
2021-04-12T12:32:17.616722778+00:00 stderr F + echo 'Running command:
'\''/usr/bin/cinder-volume --config-file /usr/share/cinder/cinder-dist.conf --config-file
/etc/cinder/cinder.conf'\''
2021-04-12T12:32:17.616751172+00:00 stdout F Running command: '/usr/bin/cinder-volume
--config-file /usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf'
2021-04-12T12:32:17.616775368+00:00 stderr F + exec /usr/bin/cinder-volume --config-file
/usr/share/cinder/cinder-dist.conf --config-file /etc/cinder/cinder.conf
```

4. 出力およびログからの情報に基づいて、異常が発生したリソースを修正します。

5. **pcs resource cleanup** コマンドを実行し、リソースのステータスおよび異常回数をリセットします。

```
$ sudo pcs resource cleanup openstack-cinder-volume
  Resource: openstack-cinder-volume successfully cleaned up
```

## 7.3. SYSTEMD リソースの問題の調査

systemd が管理するリソースの異常を調査するには、リソースに異常が発生しているコントローラーノードにログインして、リソースのステータスおよびログイベントを確認します。たとえば、**tripleo_nova_conductor** リソースのステータスおよびログイベントを調べます。

**前提条件**

- systemd サービスが含まれるコントローラーノード
- ログイベントを表示するための root ユーザーのアクセス権限

**手順**

1. **systemctl status** コマンドを実行し、リソースのステータスおよび最近のログイベントを表示します。

```
[tripleo-admin@controller-0 ~]$ sudo systemctl status tripleo_nova_conductor
● tripleo_nova_conductor.service - nova_conductor container
   Loaded: loaded (/etc/systemd/system/tripleo_nova_conductor.service; enabled; vendor
preset: disabled)
   Active: active (running) since Mon 2021-04-12 10:54:46 UTC; 1h 38min ago
 Main PID: 5125 (conmon)
    Tasks: 2 (limit: 126564)
   Memory: 1.2M
   CGroup: /system.slice/tripleo_nova_conductor.service
           └─5125 /usr/bin/conmon --api-version 1 -c
cc3c63b54e0864c94ac54a5789be96aea1dd60b2f3216b37c3e020c76e7887d4 -u
cc3c63b54e0864c94ac54a5789be96aea1dd60b2f3216b37c3e020c76e7887d4 -r
/usr/bin/runc -b /var/lib/containers/storage/overlay-
containers/cc3c63b54e0864c94ac54a5789be96aea1dd60b2f3216b37c3e02>
```

```
Apr 12 10:54:42 controller-0.redhat.local systemd[1]: Starting nova_conductor container...
Apr 12 10:54:46 controller-0.redhat.local podman[2855]: nova_conductor
Apr 12 10:54:46 controller-0.redhat.local systemd[1]: Started nova_conductor container.
```

2. リソースのログイベントを表示します。

```
# sudo less /var/log/containers/tripleo_nova_conductor.log
```

3. 出力およびログからの情報に基づいて、異常が発生したリソースを修正します。

4. リソースを再起動して、サービスのステータスを確認します。

```
# systemctl restart tripleo_nova_conductor
# systemctl status tripleo_nova_conductor
● tripleo_nova_conductor.service - nova_conductor container
   Loaded: loaded (/etc/systemd/system/tripleo_nova_conductor.service; enabled; vendor
preset: disabled)
   Active: active (running) since Thu 2021-04-22 14:28:35 UTC; 7s ago
  Process: 518937 ExecStopPost=/usr/bin/podman stop -t 10 nova_conductor (code=exited,
status=0/SUCCESS)
  Process: 518653 ExecStop=/usr/bin/podman stop -t 10 nova_conductor (code=exited,
status=0/SUCCESS)
  Process: 519063 ExecStart=/usr/bin/podman start nova_conductor (code=exited,
status=0/SUCCESS)
 Main PID: 519198 (conmon)
    Tasks: 2 (limit: 126564)
   Memory: 1.1M
   CGroup: /system.slice/tripleo_nova_conductor.service
           └─519198 /usr/bin/conmon --api-version 1 -c
0d6583beb20508e6bacccd5fea169a2fe949471207cb7d4650fec5f3638c2ce6 -u
0d6583beb20508e6bacccd5fea169a2fe949471207cb7d4650fec5f3638c2ce6 -r /usr/bin/runc
-b /var/lib/containe>

Apr 22 14:28:34 controller-0.redhat.local systemd[1]: Starting nova_conductor container...
Apr 22 14:28:35 controller-0.redhat.local podman[519063]: nova_conductor
Apr 22 14:28:35 controller-0.redhat.local systemd[1]: Started nova_conductor container.
```

# 第8章 高可用性 RED HAT CEPH STORAGE クラスターのモニタリング

Red Hat Ceph Storage とともにオーバークラウドをデプロイする場合、Red Hat OpenStack Platform は **ceph-mon** モニターデーモンを使用して Ceph クラスターを管理します。director はデーモンをすべてのコントローラーノードにデプロイします。

## 8.1. RED HAT CEPH 監視サービスのステータスの確認

Red Hat Ceph Storage の監視サービスのステータスを確認するには、コントローラーノードにログインして **service ceph status** コマンドを実行します。

**手順**

- コントローラーノードにログインし、Ceph Monitoring サービスが動作していることを確認します。

```
$ sudo service ceph status
=== mon.overcloud-controller-0 ===
mon.overcloud-controller-0: running {"version":"0.94.1"}
```

## 8.2. RED HAT CEPH の監視設定の確認

Red Hat Ceph Storage の監視サービスの設定を確認するには、コントローラーノードまたは Red Hat Ceph ノードにログインして **/etc/ceph/ceph.conf** ファイルを開きます。

**手順**

- コントローラーノードまたは Ceph ノードにログインし、**/etc/ceph/ceph.conf** ファイルを開き、モニタリング設定のパラメーターを表示します。

```
[global]
osd_pool_default_pgp_num = 128
osd_pool_default_min_size = 1
auth_service_required = cephx
mon_initial_members = overcloud-controller-0,overcloud-controller-1,overcloud-controller-2
fsid = 8c835acc-6838-11e5-bb96-2cc260178a92
cluster_network = 172.19.0.11/24
auth_supported = cephx
auth_cluster_required = cephx
mon_host = 172.18.0.17,172.18.0.15,172.18.0.16
auth_client_required = cephx
osd_pool_default_size = 3
osd_pool_default_pg_num = 128
public_network = 172.18.0.17/24
```

この例は、以下の情報を示しています。

  - **mon_initial_members** パラメーターにより、3 つすべてのコントローラーノードは、Red Hat Ceph Storage クラスターをモニターするように設定されています。
  - コントローラーノードと Red Hat Ceph Storage ノード間の通信パスを提供するために、**172.19.0.11/24** ネットワークが設定されています。

- Red Hat Ceph Storage ノードはコントローラーノードとは別のネットワークに割り当てられ、モニタリングするコントローラーノードの IP アドレスは **172.18.0.15**、**172.18.0.16**、および **172.18.0.17** です。

## 8.3. RED HAT CEPH ノードのステータスの確認

特定の Red Hat Ceph Storage ノードのステータスを確認するには、ノードにログインして **ceph -s** コマンドを実行します。

**手順**

- Ceph ノードにログインし、**ceph -s** コマンドを実行します。

```
# ceph -s
    cluster 8c835acc-6838-11e5-bb96-2cc260178a92
     health HEALTH_OK
     monmap e1: 3 mons at {overcloud-controller-0=172.18.0.17:6789/0,overcloud-controller-1=172.18.0.15:6789/0,overcloud-controller-2=172.18.0.16:6789/0}
            election epoch 152, quorum 0,1,2 overcloud-controller-1,overcloud-controller-2,overcloud-controller-0
     osdmap e543: 6 osds: 6 up, 6 in
      pgmap v1736: 256 pgs, 4 pools, 0 bytes data, 0 objects
            267 MB used, 119 GB / 119 GB avail
                256 active+clean
```

この出力例からは、**health** パラメーターの値が **HEALTH_OK** であることが分かります。これは、Ceph ノードがアクティブで正常であることを示します。この出力には、3 つの **overcloud-controller** ノード上で実行されている 3 つの Ceph Monitoring サービス、ならびにこれらのサービスの IP アドレスおよびポートも示されています。

## 8.4. 関連情報

- Red Hat Ceph の製品ページ