Red Hat Enterprise Linux 7

High Availability Add-On の管理

Red Hat High Availability デプロイメントの設定

Last Updated: 2023-03-25

Red Hat Enterprise Linux 7 High Availability Add-On の管理

Red Hat High Availability デプロイメントの設定

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

High Availability Add-On の管理 では、Red Hat Enterprise Linux 7 向けに High Availability Add-On を使用するサンプルのクラスター設定を紹介します。

目次

第1章 PACEMAKER を使用した RED HAT HIGH AVAILABILITY クラスターの作成

本章では、**pcs** コマンドを使用して 2 ノードの Red Hat High Availability クラスターを作成する手順を説明します。クラスターの作成後、必要なリソースやリソースグループを設定できます。

本章で説明しているクラスターを設定する場合には次のコンポーネントが必要になります。

- クラスターを作成するのに使用する 2 つのノード。この例では、使用されるノードは **z1.example.com** および **z2.example.com** です。
- プライベートネットワーク用のネットワークスイッチ、クラスター同士の通信およびネットワーク電源スイッチやファイバーチャンネルスイッチなどのクラスターハードウェアとの通信に必要になります。
- 各ノード用の電源フェンスデバイス、ここでは APC 電源スイッチの 2 ポートを使用しています。この例では、APC 電源スイッチの 2 ポートを使用します。ホスト名は **zapc.example.com** です。

本章は 3 つの項に分かれています。

- 「クラスターソフトウェアのインストール」 では、クラスターソフトウェアのインストール手順を説明します。
- 「クラスターの作成」 では、2 ノードクラスターの設定手順を説明します。
- 「排他処理の設定」 では、クラスターの各ノードにフェンスデバイスを設定する手順を説明します。

1.1. クラスターソフトウェアのインストール

クラスターのインストールおよび設定手順を以下に示します。

1. クラスターの各ノードに、Red Hat High Availability Add-On ソフトウェアパッケージと、使用可能なすべてのフェンスエージェントを、High Availability チャンネルからインストールします。

```
# yum install pcs pacemaker fence-agents-all
```

2. **firewalld** デーモンを実行している場合は、以下のコマンドを実行して Red Hat High Availability Add-On が必要とするポートを有効にします。

注記

firewalld デーモンがシステムにインストールされているかどうかを確認するには、**rpm -q firewalld** コマンドを実行します。**firewalld** デーモンがインストールされている場合は、**firewall-cmd --state** コマンドを使用して実行されているかどうかを確認できます。

```
# firewall-cmd --permanent --add-service=high-availability
# firewall-cmd --add-service=high-availability
```

3. **pcs** を使用してクラスターの設定やノード間の通信を行うため、**pcs** の管理アカウントとなるユーザー ID **hacluster** のパスワードを各ノードに設定する必要があります。**hacluster** ユーザーのパスワードは、各ノードで同じにすることが推奨されます。

```
# passwd hacluster
Changing password for user hacluster.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

4. クラスターを設定する前に、各ノードで起動時にブートするよう **pcsd** デーモンが起動および有効化されている必要があります。このデーモンは **pcs** コマンドで動作し、クラスターのノード全体で設定を管理します。

   クラスターの各ノードで次のコマンドを実行して、システムの起動時に **pcsd** サービスが起動し、**pcsd** が有効になるように設定します。

```
# systemctl start pcsd.service
# systemctl enable pcsd.service
```

5. **pcs** を実行するノードでクラスター内の各ノードの **pcs** ユーザー **hacluster** を認証します。

   次のコマンドは、**z1.example.com** と **z2.example.com** で設定される 2 ノードクラスターの両ノードに対して、**z1.example.com** の **hacluster** ユーザーを認証します。

```
[root@z1 ~]# pcs cluster auth z1.example.com z2.example.com
Username: hacluster
Password:
z1.example.com: Authorized
z2.example.com: Authorized
```

1.2. クラスターの作成

この手順では、**z1.example.com** ノードおよび **z2.example.com** ノードで設定される Red Hat High Availability Add-On クラスターを作成します。

1. **z1.example.com** で以下のコマンドを実行し、2 つのノード **z1.example.com** と **z2.example.com** で設定される 2 ノードクラスター **my_cluster** を作成します。これにより、クラスター設定ファイルが、クラスターの両ノードに伝搬されます。このコマンドには **--start** オプションが含まれます。このオプションを使用すると、クラスターの両ノードでクラスターサービスが起動します。

```
[root@z1 ~]# pcs cluster setup --start --name my_cluster \
z1.example.com z2.example.com
z1.example.com: Succeeded
z1.example.com: Starting Cluster...
z2.example.com: Succeeded
z2.example.com: Starting Cluster...
```

2. クラスターサービスを有効にし、ノードの起動時にクラスターの各ノードでクラスターサービスが実行するようにします。

注記

使用している環境でクラスターサービスを無効のままにしておきたい場合などは、この手順を省略できます。この手順を行うことで、ノードがダウンした場合にクラスターやリソース関連の問題をすべて解決してから、そのノードをクラスターに戻すことができます。クラスターサービスを無効にしている場合には、ノードを再起動する時に **pcs cluster start** コマンドを使って手作業でサービスを起動しなければならないので注意してください。

```
[root@z1 ~]# pcs cluster enable --all
```

pcs cluster status コマンドを使用するとクラスターの現在の状態を表示できます。**pcs cluster setup** コマンドで **--start** オプションを使用してクラスターサービスを起動した場合は、クラスターが稼働するのに時間が少しかかる可能性があるため、クラスターとその設定で後続の動作を実行する前に、クラスターが稼働していることを確認する必要があります。

```
[root@z1 ~]# pcs cluster status
Cluster Status:
 Last updated: Thu Jul 25 13:01:26 2013
 Last change: Thu Jul 25 13:04:45 2013 via crmd on z2.example.com
 Stack: corosync
 Current DC: z2.example.com (2) - partition with quorum
 Version: 1.1.10-5.el7-9abe687
 2 Nodes configured
 0 Resources configured
```

1.3. 排他処理の設定

クラスターの各ノードにフェンスデバイスを設定する必要があります。フェンス設定コマンドの説明やオプションは、『Red Hat Enterprise Linux 7 High Availability Add-On リファレンス 』 を参照してください。排他処理 (フェンシング) やその重要性は、Fencing in a Red Hat High Availability Cluster を参照してください。

注記

フェンスデバイスを設定する場合は、そのデバイスが、クラスター内のノードまたはデバイスと電源を共有しているかどうかに注意する必要があります。ノードとそのフェンスデバイスが電源を共有していると、その電源をフェンスできず、フェンスデバイスが失われた場合は、クラスターがそのノードをフェンスできない可能性があります。このようなクラスターには、フェンスデバイスおよびノードに冗長電源を提供するか、または電源を共有しない冗長フェンスデバイスが存在する必要があります。SBD やストレージフェンシングなど、その他のフェンシング方法でも、分離した電源供給の停止時に冗長性を得られます。

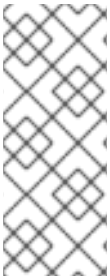
ここでは、ホスト名が **zapc.example.com** の APC 電源スイッチを使用してノードをフェンスし、**fence_apc_snmp** フェンスエージェントを使用します。ノードはすべて同じフェンスエージェントで排他処理されるため、**pcmk_host_map** と **pcmk_host_list** のオプションを使ってすべてのフェンスデバイスを一つのリソースとして設定できます。

pcs stonith create コマンドを使って **stonith** リソースとしてデバイスを設定し、フェンスデバイスを作成します。以下のコマンドは、**z1.example.com** ノードおよび **z2.example.com** ノードの **fence_apc_snmp** フェンスエージェントを使用する、**stonith** リソース **myapc** を設定しま

す。**pcmk_host_map** オプションにより、**z1.example.com** がポート 1 にマップされ、**z2.example.com** がポート 2 にマップされます。APC デバイスのログイン値とパスワードはいずれも **apc** です。デフォルトでは、このデバイスは各ノードに対して、60 秒間隔で監視を行います。

また、ノードのホスト名を指定する際に、IP アドレスを使用できます。

```
[root@z1 ~]# pcs stonith create myapc fence_apc_snmp \
ipaddr="zapc.example.com" pcmk_host_map="z1.example.com:1;z2.example.com:2" \
pcmk_host_check="static-list" pcmk_host_list="z1.example.com,z2.example.com" \
login="apc" passwd="apc"
```



注記

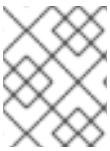
fence_apc_snmp stonith デバイスを作成するときに次のような警告メッセージが表示されることがありますがこのメッセージは無視して構いません。

```
Warning: missing required option(s): 'port, action' for resource type:
stonith:fence_apc_snmp
```

次のコマンドは、既存の STONITH デバイスのパラメーターを表示します。

```
[root@rh7-1 ~]# pcs stonith show myapc
 Resource: myapc (class=stonith type=fence_apc_snmp)
  Attributes: ipaddr=zapc.example.com pcmk_host_map=z1.example.com:1;z2.example.com:2
pcmk_host_check=static-list pcmk_host_list=z1.example.com,z2.example.com login=apc
passwd=apc
  Operations: monitor interval=60s (myapc-monitor-interval-60s)
```

フェンスデバイスの設定後に、デバイスをテストする必要があります。フェンスデバイスのテストの説明は、『High Availability Add-On リファレンス』 の フェンス機能: STONITH の設定



注記

ネットワークインターフェイスを無効にしてフェンスデバイスのテストを実行しないでください。フェンシングが適切にテストされなくなります。

注記

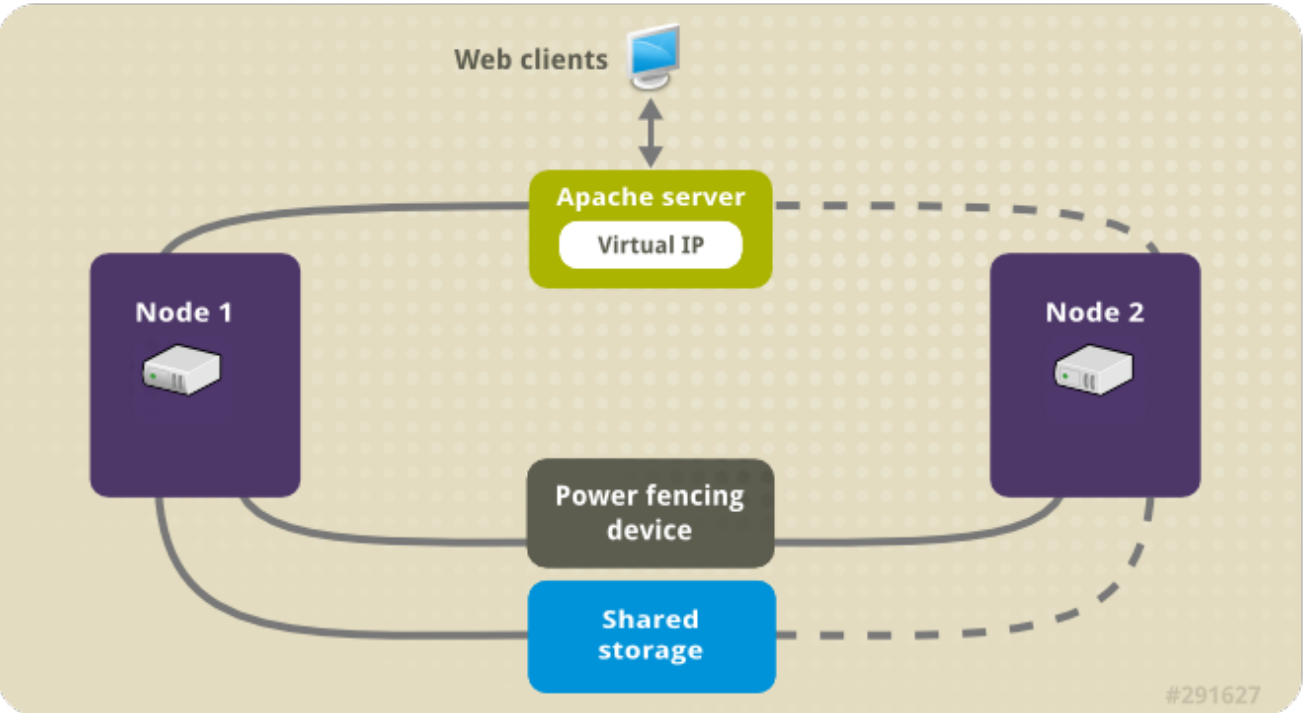
フェンシングを設定してクラスターが起動すると、タイムアウトに到達していなくても、ネットワークの再起動時に、ネットワークを再起動するノードのフェンシングが発生します。このため、ノードで意図しないフェンシングが発生しないように、クラスターサービスの実行中はネットワークサービスを再起動しないでください。

第2章 RED HAT HIGH AVAILABILITY クラスターのアクティブ/パッシブ APACHE HTTP サーバー

本章では、**pcs** コマンドを使用してクラスターリソースを設定し、2 ノードの Red Hat Enterprise Linux High Availability Add-On クラスターでアクティブ/パッシブな Apache HTTP サーバーを設定する方法を説明します。このユースケースでは、クライアントはフローティング IP アドレスを使用して Apache HTTP サーバーにアクセスします。Web サーバーは、クラスターにある 2 つのノードのいずれかで実行します。Web サーバーが実行しているノードが正常に動作しなくなると、Web サーバーはクラスターの 2 番目のノードで再起動し、サービスの中断は最小限に抑えられます。

図2.1「2 ノードの Red Hat High Availability クラスターの Apache」 クラスターのハイレベルの概要を表示します。クラスターはネットワーク電源スイッチおよび共有ストレージで設定される 2 ノードの Red Hat High Availability クラスターです。クライアントは仮想 IP を使用して Apache HTTP サーバーにアクセスするため、クラスターノードはパブリックネットワークに接続されます。Apache サーバーは、ノード 1 またはノード 2 のいずれかで実行します。いずれのノードも、Apache のデータが保持されるストレージにアクセスできます。

図2.1 2 ノードの Red Hat High Availability クラスターの Apache



[D]

このユースケースでは、システムに以下のコンポーネントが必要です。

- 各ノードに電源フェンスが設定されている 2 ノードの Red Hat High Availability クラスター。この手順では、1章*Pacemaker を使用した Red Hat High Availability クラスターの作成* で提供されるクラスターの例を使用します。
- Apache に必要なパブリック仮想 IP アドレス。
- iSCSI、ファイバーチャネル、またはその他の共有ネットワークブロックデバイスを使用する、クラスターのノードに対する共有ストレージ。

クラスターは、Web サーバーで必要な LVM リソース、ファイルシステムリソース、IP アドレスリソース、Web サーバーリソースなどのクラスターコンポーネントを含む Apache リソースグループで設定されます。このリソースグループは、クラスター内のあるノードから別のノードへのフェールオーバーが

可能なため、いずれのノードでも Web サーバーを実行できます。クラスターにリソースグループを作成する前に次の手順を行います。

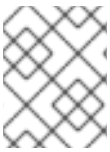
1. 「LVM ボリュームを ext4 ファイルシステムで設定」 の説明に従い **my_lv** 論理ボリュームに **ext4** ファイルシステムを設定します。
2. 「Web サーバーの設定」 の説明に従い web サーバーを設定します。
3. 「ボリュームグループのアクティブ化をクラスター内に限定」 の説明に従い、**my_lv** を含むボリュームグループの作動はクラスターでしか行えないよう限定し、またボリュームグループが起動時にクラスター以外の場所で作動しないようにします。

上記の手順をすべて完了したら、「pcs コマンドを使用したリソースおよびリソースグループの作成」の説明に従いリソースグループおよびそのグループに含ませるリソースを作成します。

2.1. LVM ボリュームを EXT4 ファイルシステムで設定

このユースケースでは、クラスターのノード間で共有されるストレージに、LVM 論理ボリュームを作成する必要があります。

次の手順に従い LVM 論理ボリュームを作成しその論理ボリューム上に **ext4** ファイルシステムを作成します。この例では、LVM 論理ボリュームを作成する LVM 物理ボリュームを保管するのに、共有パーティション **/dev/sdb1** が使用されます。



注記

LVM ボリュームと、クラスターノードで使用するパーティションおよびデバイスは、クラスターノード以外には接続しないでください。

/dev/sdb1 パーティションは共有させるストレージとなるため、この手順は一つのノードでのみ行います。

1. パーティション **/dev/sdb1** に LVM 物理ボリュームを作成します。

```
# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
```

2. 物理ボリューム **/dev/sdb1** で設定されるボリュームグループ **my_vg** を作成します。

```
# vgcreate my_vg /dev/sdb1
  Volume group "my_vg" successfully created
```

3. ボリュームグループ **my_vg** を使用して、論理ボリュームを作成します。

```
# lvcreate -L450 -n my_lv my_vg
  Rounding up size to full physical extent 452.00 MiB
  Logical volume "my_lv" created
```

lvs コマンドを使って論理ボリュームを表示してみます。

```
# lvs
  LV      VG      Attr      LSize   Pool Origin Data%  Move Log Copy%  Convert
  my_lv   my_vg   -wi-a---- 452.00m
```

```
  ...
```

4. **ext4** ファイルシステムを **my_lv** 論理ボリューム上に作成します。

```
# mkfs.ext4 /dev/my_vg/my_lv
mke2fs 1.42.7 (21-Jan-2013)
Filesystem label=
OS type: Linux
...
```

2.2. WEB サーバーの設定

次の手順に従って Apache HTTP サーバーを設定します。

1. クラスターの各ノードに、Apache HTTP サーバーがインストールされていることを確認します。Apache HTTP サーバーのステータスを確認するには、クラスターに **wget** ツールがインストールされている必要があります。

   各ノードで、以下のコマンドを実行します。

```
# yum install -y httpd wget
```

2. Apache リソースエージェントが Apache HTTP サーバーの状態を取得できるようにするため、クラスターの各ノードの **/etc/httpd/conf/httpd.conf** ファイルに以下のテキストが含まれ、コメントアウトされていないことを確認してください。これが記載されていない場合は、ファイルの末尾に追加します。

```
<Location /server-status>
   SetHandler server-status
   Require local
</Location>
```

3. **apache** リソースエージェントを使用して Apache を管理する場合は **systemd** が使用されません。そのため、Apache のリロードに **systemctl** が使用されないようにするため、Apache によって提供される **logrotate** スクリプトを編集する必要があります。

   クラスター内の各ノード上で、**/etc/logrotate.d/httpd** ファイルから以下の行を削除します。

```
/bin/systemctl reload httpd.service > /dev/null 2>/dev/null || true
```

   削除した行を以下の 3 行に置き換えます。

```
/usr/bin/test -f /run/httpd.pid >/dev/null 2>/dev/null &&
/usr/bin/ps -q $(/usr/bin/cat /run/httpd.pid) >/dev/null 2>/dev/null &&
/usr/sbin/httpd -f /etc/httpd/conf/httpd.conf -c "PidFile /run/httpd.pid" -k graceful > /dev/null
2>/dev/null || true
```

4. Apache で提供する Web ページを作成します。クラスター内のいずれかのノードに 「LVM ボリュームを ext4 ファイルシステムで設定」 で作成したファイルシステムをマウントし、そのファイルシステム上で **index.html** ファイルを作成したら再びファイルシステムをアンマウントします。

```
# mount /dev/my_vg/my_lv /var/www/
# mkdir /var/www/html
# mkdir /var/www/cgi-bin
# mkdir /var/www/error
# restorecon -R /var/www
# cat <<-END >/var/www/html/index.html
<html>
<body>Hello</body>
</html>
END
# umount /var/www
```

2.3. ボリュームグループのアクティブ化をクラスター内に限定

次の手順でボリュームグループを設定すると、クラスターでしかボリュームグループを作動することができなくなり、またボリュームグループは起動時にクラスター以外の場所では作動しなくなります。ボリュームグループがクラスター外部のシステムによってアクティブ化されると、ボリュームグループのメタデータが破損することがあります。

この手順では **/etc/lvm/lvm.conf** 設定ファイル内の **volume_list** のエントリーを編集します。**volume_list** のエントリーに記載されているボリュームグループはクラスターマネージャーの管轄外となるローカルノードでの自動作動が許可されます。ノードのローカルな root ディレクトリーやホームディレクトリーに関連するボリュームグループはこのリストに含ませてください。クラスターマネージャーで管理するボリュームグループは **volume_list** のエントリーには入れないでください。ここでの手順に **clvmd** を使用する必要はありません。

クラスター内の各ノードで以下の手順を実行します。

1. 次のコマンドを実行して、**/etc/lvm/lvm.conf** ファイルで **locking_type** が 1 に設定されていることと **use_lvmetad** が 0 に設定されていることを確認します。また、このコマンドを実行すると、すべての **lvmetad** プロセスがすぐに無効になり、停止します。

   ```
   # lvmconf --enable-halvm --services --startstopservices
   ```

2. 以下のコマンドを使用して、ローカルストレージに現在設定されているボリュームグループを確認します。これにより、現在設定されているボリュームグループの一覧が出力されます。このノードの root とホームディレクトリーに、別のボリュームグループの領域を割り当てると、この例のように以下のボリュームが出力に表示されます。

   ```
   # vgs --noheadings -o vg_name
     my_vg
     rhel_home
     rhel_root
   ```

3. **my_vg** 以外のボリュームグループ (クラスターに定義したボリュームグループ) をエントリーとして **/etc/lvm/lvm.conf** という設定ファイルの **volume_list** に追加します。

   例えば、root ディレクトリー用のボリュームグループ、ホームディレクトリー用のボリュームグループを別々に用意している場合は、**lvm.conf** ファイルの **volume_list** の行のコメントを外して以下のように root ディレクトリー用、ホームディレクトリー用の各ボリュームグループを **volume_list** のエントリーとして追加します。クラスターに対してだけ定義したボリュームグループ (この例では **my_vg**) は、この一覧は含まれない点に注意してください。

   ```
   volume_list = [ "rhel_root", "rhel_home" ]
   ```

-

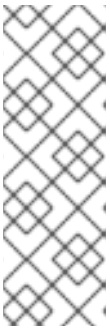
注記

クラスターマネージャーの管轄外で作動させるローカルボリュームグループがノードにない場合でも **volume_list** のエントリーは **volume_list = []** と指定して初期化する必要があります。

4. **initramfs** ブートイメージを再構築して、クラスターが制御するボリュームグループがブートイメージによりアクティベートされないようにします。以下のコマンドを使用して、**initramfs** デバイスを更新します。このコマンドが完了するまで最大 1 分かかる場合があります。

```
# dracut -H -f /boot/initramfs-$(uname -r).img $(uname -r)
```

5. ノードを再起動します。



注記

ブートイメージを作成したノードを起動してから、新しい Linux カーネルをインストールした場合は、新しい **initrd** イメージは、作成時に実行していたカーネル用で、ノードの再起動時に実行している新しいカーネル用ではありません。再起動の前後で **uname -r** コマンドを使って実行しているカーネルリリースを確認し必ず正しい **initrd** デバイスを使用するよう注意してください。リリースが同じでない場合には、新規カーネルで再起動した後に **initrd** ファイルを更新して、ノードを再起動します。

6. ノードが再起動したら **pcs cluster status** コマンドを実行し、クラスターサービスがそのノードで再度開始されたかどうかを確認します。**Error: cluster is not running on this node** というメッセージが表示される場合は、以下のコマンドを入力します。

```
# pcs cluster start
```

または、クラスター内の各ノードの再起動が完了するのを待ってから次のコマンドで各ノードでのクラスターサービスの起動を行います。

```
# pcs cluster start --all
```

2.4. PCS コマンドを使用したリソースおよびリソースグループの作成

このユースケースでは、クラスターリソースを 4 つ作成する必要があります。すべてのリソースが必ず同じノードで実行するように、このリソースを、リソースグループ **apachegroup** に追加します。作成するリソースは以下のとおりで、開始する順に記載されています。

1. 「LVM ボリュームを ext4 ファイルシステムで設定」 で作成した LVM ボリュームグループを使用する **my_lvm** という名前の **LVM** リソース。

2. 「LVM ボリュームを ext4 ファイルシステムで設定」 の手順で作成したファイルシステムデバイス **/dev/my_vg/my_lv** を使用する、**my_fs** という名前の **Filesystem** リソース。

3. **apachegroup** リソースグループのフローティング IP アドレスである **IPaddr2** リソース。物理ノードに関連付けられている IP アドレスは使用できません。**IPaddr2** リソースの NIC デバイスが指定されていない場合、クラスターノードによって使用される静的に割り当てられた IP アドレスと同じネットワーク上にフローティング IP が存在しないと、フローティング IP アドレスを割り当てる NIC デバイスが適切に検出されません。

4. **Website** と言う名前の **apache** リソース、「Web サーバーの設定」 の手順で定義した **index.html** ファイルと Apache 設定を使用します。

以下の手順で、**apachegroup** リソースグループと、このグループに追加するリソースを作成します。リソースは、グループに追加された順序で起動し、その逆の順序で停止します。この手順は、クラスター内のいずれかのノードで実行してください。

1. 次のコマンドでは **my_lvm** LVM リソースを作成しています。このコマンドは、**exclusive=true** パラメーターを指定し、クラスターのみが LVM 論理ボリュームをアクティブ化できるようにします。リソースグループ **apachegroup** は存在しないため、このコマンドによりリソースグループが作成されます。

```
[root@z1 ~]# pcs resource create my_lvm LVM volgrpname=my_vg \
exclusive=true --group apachegroup
```

リソースを作成すると、そのリソースは自動的に起動します。以下のコマンドを使用すると、リソースが作成され、起動していることを確認できます。

```
# pcs resource show
 Resource Group: apachegroup
     my_lvm (ocf::heartbeat:LVM): Started
```

pcs resource disable と **pcs resource enable** のコマンドを使用すると手作業によるリソースの停止と起動をリソースごと個別に行うことができます。

2. 以下のコマンドでは、設定に必要な残りのリソースを作成し、作成したリソースを既存の **apachegroup** リソースグループに追加します。

```
[root@z1 ~]# pcs resource create my_fs Filesystem \
device="/dev/my_vg/my_lv" directory="/var/www" fstype="ext4" --group \
apachegroup

[root@z1 ~]# pcs resource create VirtualIP IPaddr2 ip=198.51.100.3 \
cidr_netmask=24 --group apachegroup

[root@z1 ~]# pcs resource create Website apache \
configfile="/etc/httpd/conf/httpd.conf" \
statusurl="http://127.0.0.1/server-status" --group apachegroup
```

3. リソースと、そのリソースを含むリソースグループの作成が完了したら、クラスターのステータスを確認します。4 つのリソースがすべて同じノードで実行していることに注意してください。

```
[root@z1 ~]# pcs status
Cluster name: my_cluster
Last updated: Wed Jul 31 16:38:51 2013
Last change: Wed Jul 31 16:42:14 2013 via crm_attribute on z1.example.com
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
Version: 1.1.10-5.el7-9abe687
2 Nodes configured
6 Resources configured


Online: [ z1.example.com z2.example.com ]
```

```
Full list of resources:
 myapc (stonith:fence_apc_snmp): Started z1.example.com
 Resource Group: apachegroup
     my_lvm (ocf::heartbeat:LVM): Started z1.example.com
     my_fs (ocf::heartbeat:Filesystem): Started z1.example.com
     VirtualIP (ocf::heartbeat:IPaddr2): Started z1.example.com
     Website (ocf::heartbeat:apache): Started z1.example.com
```

「排他処理の設定」 の手順でクラスターにフェンスデバイスを設定していないとリソースはデフォルトでは起動しないので注意してください。

4. クラスターが稼働したら、ブラウザーで、**IPaddr2** リソースとして定義した IP アドレスを指定して、Hello と単語が表示されるサンプル表示を確認します。

```
Hello
```

設定したリソースが実行されていない場合は、**pcs resource debug-start *resource*** コマンドを実行してリソースの設定をテストできます。**pcs resource debug-start** コマンドの詳細は『High Availability Add-On リファレンス』 を参照してください。

2.5. リソース設定のテスト

「pcs コマンドを使用したリソースおよびリソースグループの作成」 で示すようにクラスターの状態表示では全リソースが **z1.example.com** ノードで実行しています。以下の手順に従い、1 番目のノードを **スタンバイ** モードにし、リソースグループが **z2.example.com** ノードにフェールオーバーするかどうかをテストします。1 番目のノードをスタンバイモードにすると、このノードはリソースをホストできなくなります。

1. 以下のコマンドは、**z1.example.com** ノードを **スタンバイ** モードにします。

```
root@z1 ~]# pcs node standby z1.example.com
```

2. **z1** をスタンバイモードにしたらクラスターの状態を確認します。リソースはすべて **z2** で実行しているはずです。

```
[root@z1 ~]# pcs status
Cluster name: my_cluster
Last updated: Wed Jul 31 17:16:17 2013
Last change: Wed Jul 31 17:18:34 2013 via crm_attribute on z1.example.com
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
Version: 1.1.10-5.el7-9abe687
2 Nodes configured
6 Resources configured


Node z1.example.com (1): standby
Online: [ z2.example.com ]

Full list of resources:

 myapc (stonith:fence_apc_snmp): Started z1.example.com
 Resource Group: apachegroup
```

```
    my_lvm (ocf::heartbeat:LVM): Started z2.example.com
    my_fs (ocf::heartbeat:Filesystem): Started z2.example.com
    VirtualIP (ocf::heartbeat:IPaddr2): Started z2.example.com
    Website (ocf::heartbeat:apache): Started z2.example.com
```

定義している IP アドレスの Web サイトは、中断せず表示されているはずです。

3. **スタンバイ** モードから **z1** を削除するには、以下のコマンドを実行します。

```
root@z1 ~]# pcs node unstandby z1.example.com
```

注記

ノードを **スタンバイ** モードから削除しても、リソースはそのノードにフェイルオーバーしません。これは、リソースの **resource-stickiness** 値により異なります。**resource-stickiness** メタ属性の詳細は、『High Availability Add-On リファレンス』の現在のノードを優先させるリソースの設定を参照してください。

第3章 RED HAT HIGH AVAILABILITY クラスターのアクティブ/パッシブな NFS サーバー

本章では、共有ストレージを使用して 2 ノードの Red Hat Enterprise Linux High Availability Add-On クラスターで高可用性アクティブ/パッシブ NFS サーバーを設定する方法について説明します。この手順では、Pacemaker クラスターリソースの設定に **pcs** を使用します。このユースケースでは、クライアントが、フローティング IP アドレスから NFS ファイルシステムにアクセスします。NFS サービスは、クラスターにある 2 つのノードのいずれかで実行します。NFS サーバーが実行しているノードが正常に動作しなくなると、NFS サーバーはクラスターの 2 番目のノードで再起動し、サービスの中断が最小限に抑えられます。

このユースケースでは、システムに以下のコンポーネントが必要です。

- Apache HTTP サーバーを実行するクラスターを作成するために使用される 2 つのノード。この例では、使用されるノードは **z1.example.com** および **z2.example.com** です。
- 各ノード用の電源フェンスデバイス、ここでは APC 電源スイッチの 2 ポートを使用しています。この例では、APC 電源スイッチの 2 ポートを使用します。ホスト名は **zapc.example.com** です。
- NFS サーバーに必要なパブリック仮想 IP アドレス。
- iSCSI、ファイバーチャネル、またはその他の共有ネットワークデバイスを使用する、クラスター内のノードの共有ストレージ。

2 ノード Red Hat Enterprise Linux で高可用性アクティブ/パッシブ NFS サーバーを設定するには、以下の手順を実行する必要があります。

1. 「NFS クラスターの作成」 の説明に従って、NFS サーバーを実行するクラスターを作成し、クラスターの各ノードにフェンシングを設定します。
2. 「LVM ボリュームを ext4 ファイルシステムで設定」 の説明に従って、クラスターのノードに対する共有ストレージの LVM 論理ボリューム **my_lv** にマウントされた **ext4** ファイルシステムを設定します。
3. 「NFS 共有の設定」 の説明に従って、LVM 論理ボリュームの共有ストレージで NFS 共有を設定します。
4. 「ボリュームグループのアクティブ化をクラスター内に限定」 の説明に従って、論理ボリューム **my_lv** が含まれる LVM ボリュームグループをクラスターのみがアクティブ化できるようにし、ボリュームグループが起動時にクラスターの外部でアクティブ化されないようにします。
5. 「クラスターリソースの設定」 の説明に従って、クラスターリソースを作成します。
6. 「リソース設定のテスト」 に従って、設定した NFS サーバーをテストします。

3.1. NFS クラスターの作成

以下の手順に従って、NFS クラスターをインストールおよび作成します。

1. 「クラスターソフトウェアのインストール」 の手順に従って、**z1.example.com** および **z2.example.com** ノードにクラスターソフトウェアをインストールします。
2. 「クラスターの作成」 で説明されている手順を使用して、**z1.example.com** および **z2.example.com** で設定される 2 ノードクラスターを作成します。この手順の例と同様に、クラスターには **my_cluster** という名前が付けられます。

3. 「排他処理の設定」 の説明に従って、クラスターの各ノードにフェンスデバイスを設定します。この例では、ホスト名が **zapc.example.com** という APC 電源スイッチの 2 つのポートを使用してフェンシングが設定されます。

3.2. LVM ボリュームを EXT4 ファイルシステムで設定

このユースケースでは、クラスターのノード間で共有されるストレージに、LVM 論理ボリュームを作成する必要があります。

次の手順に従い LVM 論理ボリュームを作成しその論理ボリューム上に **ext4** ファイルシステムを作成します。この例では、LVM 論理ボリュームを作成する LVM 物理ボリュームを保管するのに、共有パーティション **/dev/sdb1** が使用されます。

注記

LVM ボリュームと、クラスターノードで使用するパーティションおよびデバイスは、クラスターノード以外には接続しないでください。

/dev/sdb1 パーティションは共有させるストレージとなるため、この手順は一つのノードでのみ行います。

1. パーティション **/dev/sdb1** に LVM 物理ボリュームを作成します。

```
[root@z1 ~]# pvcreate /dev/sdb1
  Physical volume "/dev/sdb1" successfully created
```

2. 物理ボリューム **/dev/sdb1** で設定されるボリュームグループ **my_vg** を作成します。

```
[root@z1 ~]# vgcreate my_vg /dev/sdb1
  Volume group "my_vg" successfully created
```

3. ボリュームグループ **my_vg** を使用して、論理ボリュームを作成します。

```
[root@z1 ~]# lvcreate -L450 -n my_lv my_vg
  Rounding up size to full physical extent 452.00 MiB
  Logical volume "my_lv" created
```

lvs コマンドを使って論理ボリュームを表示してみます。

```
[root@z1 ~]# lvs
  LV      VG      Attr      LSize   Pool Origin Data%  Move Log Copy%  Convert
  my_lv   my_vg   -wi-a---- 452.00m
  ...
```

4. **ext4** ファイルシステムを **my_lv** 論理ボリューム上に作成します。

```
[root@z1 ~]# mkfs.ext4 /dev/my_vg/my_lv
mke2fs 1.42.7 (21-Jan-2013)
Filesystem label=
OS type: Linux
...
```

3.3. NFS 共有の設定

以下の手順では、NFS デーモンフェールオーバーに対して NFS 共有を設定します。この手順は、クラスターの 1 つのノードのみで行う必要があります。

1. **/nfsshare** ディレクトリーを作成します。

```
[root@z1 ~]# mkdir /nfsshare
```

2. 「LVM ボリュームを ext4 ファイルシステムで設定」 で作成した **ext4** ファイルシステムを **/nfsshare** ディレクトリーにマウントします。

```
[root@z1 ~]# mount /dev/my_vg/my_lv /nfsshare
```

3. **/nfsshare** ディレクトリー内に **exports** ディレクトリーツリーを作成します。

```
[root@z1 ~]# mkdir -p /nfsshare/exports
[root@z1 ~]# mkdir -p /nfsshare/exports/export1
[root@z1 ~]# mkdir -p /nfsshare/exports/export2
```

4. NFS クライアントがアクセスするファイルを、**exports** ディレクトリーに置きます。この例では、**clientdatafile1** および **clientdatafile2** という名前のテストファイルを作成します。

```
[root@z1 ~]# touch /nfsshare/exports/export1/clientdatafile1
[root@z1 ~]# touch /nfsshare/exports/export2/clientdatafile2
```

5. ext4 ファイルシステムをアンマウントし、LVM ボリュームグループを非アクティブにします。

```
[root@z1 ~]# umount /dev/my_vg/my_lv
[root@z1 ~]# vgchange -an my_vg
```

3.4. ボリュームグループのアクティブ化をクラスター内に限定

次の手順では、LVM ボリュームグループを設定して、クラスターのみがボリュームグループをアクティブ化でき、ボリュームグループが起動時にクラスターの外部でアクティブ化されないようにします。ボリュームグループがクラスター外部のシステムによってアクティブ化されると、ボリュームグループのメタデータが破損することがあります。

この手順では **/etc/lvm/lvm.conf** 設定ファイル内の **volume_list** のエントリーを編集します。**volume_list** のエントリーに記載されているボリュームグループはクラスターマネージャーの管轄外となるローカルノードでの自動作動が許可されます。ノードのローカルな root ディレクトリーやホームディレクトリーに関連するボリュームグループはこのリストに含ませてください。クラスターマネージャーで管理するボリュームグループは **volume_list** のエントリーには入れないでください。ここでの手順に **clvmd** を使用する必要はありません。

クラスター内の各ノードで以下の手順を実行します。

1. 次のコマンドを実行して、**/etc/lvm/lvm.conf** ファイルで **locking_type** が 1 に設定されていることと **use_lvmetad** が 0 に設定されていることを確認します。また、このコマンドを実行すると、すべての **lvmetad** プロセスがすぐに無効になり、停止します。

```
# lvmconf --enable-halvm --services --startstopservices
```

2. 以下のコマンドを使用して、ローカルストレージに現在設定されているボリュームグループを確認します。これにより、現在設定されているボリュームグループの一覧が出力されます。このノードの root とホームディレクトリーに、別のボリュームグループの領域を割り当てると、この例のように以下のボリュームが出力に表示されます。

```
# vgs --noheadings -o vg_name
  my_vg
  rhel_home
  rhel_root
```

3. **my_vg** 以外のボリュームグループ (クラスターに定義したボリュームグループ) をエントリーとして **/etc/lvm/lvm.conf** という設定ファイルの **volume_list** に追加します。例えば、root ディレクトリー用のボリュームグループ、ホームディレクトリー用のボリュームグループを別々に用意している場合は、**lvm.conf** ファイルの **volume_list** の行のコメントを外して以下のように root ディレクトリー用、ホームディレクトリー用の各ボリュームグループを **volume_list** のエントリーとして追加します。

```
volume_list = [ "rhel_root", "rhel_home" ]
```

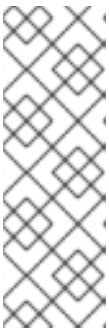
注記

クラスターマネージャーの管轄外で作動させるローカルボリュームグループがノードにない場合でも **volume_list** のエントリーは **volume_list = []** と指定して初期化する必要があります。

4. **initramfs** ブートイメージを再構築して、クラスターが制御するボリュームグループがブートイメージによりアクティベートされないようにします。以下のコマンドを使用して、**initramfs** デバイスを更新します。このコマンドが完了するまで最大 1 分かかる場合があります。

```
# dracut -H -f /boot/initramfs-$(uname -r).img $(uname -r)
```

5. ノードを再起動します。



注記

ブートイメージを作成したノードを起動してから、新しい Linux カーネルをインストールした場合は、新しい **initrd** イメージは、作成時に実行していたカーネル用で、ノードの再起動時に実行している新しいカーネル用ではありません。再起動の前後で **uname -r** コマンドを使って実行しているカーネルリリースを確認し必ず正しい **initrd** デバイスを使用するよう注意してください。リリースが同じでない場合には、新規カーネルで再起動した後に **initrd** ファイルを更新して、ノードを再起動します。

6. ノードが再起動したら **pcs cluster status** コマンドを実行し、クラスターサービスがそのノードで再度開始されたかどうかを確認します。**Error: cluster is not running on this node** というメッセージが表示される場合は、以下のコマンドを入力します。

```
# pcs cluster start
```

または、クラスターの各ノードを再起動して、クラスターの全ノードでクラスターサービスを開始するまで待機するには、次のコマンドを使用します。

```
# pcs cluster start --all
```

3.5. クラスターリソースの設定

このセクションでは、このユースケースで、クラスターリソースを設定する手順を説明します。

注記

pcs resource create コマンドを使用してクラスターリソースを作成する場合、作成直後に **pcs status** コマンドを実行してリソースが稼働していることを検証することが推奨されます。「排他処理の設定」 の手順でクラスターにフェンスデバイスを設定していないとリソースはデフォルトでは起動しないので注意してください。

設定したリソースが実行されていない場合は、**pcs resource debug-start *resource*** コマンドを実行してリソースの設定をテストできます。このコマンドは、クラスターの制御や認識の範囲外でサービスを起動します。設定したリソースが再度実行されたら、**pcs resource cleanup *resource*** コマンドを実行してクラスターが更新を認識するようにします。**pcs resource debug-start** コマンドの詳細は 『High Availability Add-On リファレンス』 を参照してください。

以下の手順では、システムリソースを設定します。これらのリソースがすべて同じノードで実行するように、これらのリソースはリソースグループ **nfsgroup** に含まれます。リソースは、グループに追加された順序で起動し、その逆の順序で停止します。この手順は、クラスター内のいずれかのノードで実行してください。

1. 以下のコマンドは **my_lvm** という名前の LVM リソースを作成します。このコマンドは、**exclusive=true** パラメーターを指定し、クラスターのみが LVM 論理ボリュームをアクティブ化できるようにします。リソースグループ **my_lvm** は存在しないため、このコマンドによりリソースグループが作成されます。

```
[root@z1 ~]# pcs resource create my_lvm LVM volgrpname=my_vg \
exclusive=true --group nfsgroup
```

クラスターのステータスを確認し、リソースが実行していることを確認します。

```
root@z1 ~]#  pcs status
Cluster name: my_cluster
Last updated: Thu Jan  8 11:13:17 2015
Last change: Thu Jan  8 11:13:08 2015
Stack: corosync
Current DC: z2.example.com (2) - partition with quorum
Version: 1.1.12-a14efad
2 Nodes configured
3 Resources configured

Online: [ z1.example.com z2.example.com ]

Full list of resources:
 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM):   Started z1.example.com

PCSD Status:
  z1.example.com: Online
```

```
  z2.example.com: Online

Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

2. クラスターに **Filesystem** リソースを設定します。

注記

options=*options* パラメーターを使用すると、**Filesystem** リソースのリソース設定の一部としてマウントオプションを指定できます。すべての設定オプションを確認する場合は、**pcs resource describe Filesystem** コマンドを実行します。

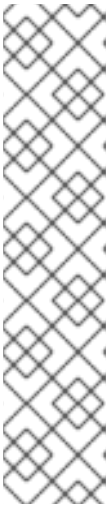
以下のコマンドは、ext4 の **Filesystem** リソース **nfsshare** を、**nfsgroup** リソースグループに追加します。このファイルシステムは、「LVM ボリュームを ext4 ファイルシステムで設定」で作成された LVM ボリュームグループと ext4 ファイルシステムを使用します。このファイルシステムは 「NFS 共有の設定」 で作成された **/nfsshare** ディレクトリーにマウントされます。

```
[root@z1 ~]# pcs resource create nfsshare Filesystem \
device=/dev/my_vg/my_lv directory=/nfsshare \
fstype=ext4 --group nfsgroup
```

my_lvm リソースおよび **nfsshare** リソースが実行していることを確認します。

```
[root@z1 ~]# pcs status
...
Full list of resources:
 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM):   Started z1.example.com
     nfsshare   (ocf::heartbeat:Filesystem):    Started z1.example.com
...
```

3. **nfsgroup** リソースグループの一部である **nfs-daemon** という名前の **nfsserver** リソースを作成します。



注記

nfsserver リソースを使用して、**nfs_shared_infodir** パラメーターを指定できます。これは、NFS デーモンが NFS 関連のステートフル情報の格納に使用するディレクトリーです。この属性は、このエクスポートのコレクションで作成した **Filesystem** リソースのいずれかのサブディレクトリーに設定することが推奨されます。これにより、NFS デーモンは、このリソースグループを再配置する必要がある場合に別のノードで使用できるデバイスに、ステートフル情報を保存します。これにより、NFS デーモンは、このリソースグループを再度移動する必要が生じた場合に、別のノードで利用可能になるステートフル情報をデバイスに保存します。この例では、**/nfsshare** は **Filesystem** リソースで管理される共有ストレージディレクトリーで、**/nfsshare/exports/export1** および **/nfsshare/exports/export2** はエクスポートディレクトリーです。**/nfsshare/nfsinfo** は、**nfsserver** リソースの共有情報ディレクトリーです。

```
[root@z1 ~]# pcs resource create nfs-daemon nfsserver \
nfs_shared_infodir=/nfsshare/nfsinfo nfs_no_notify=true \
--group nfsgroup
[root@z1 ~]# pcs status
...
```

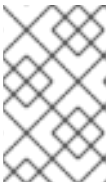
4. **exportfs** リソースを追加して **/nfsshare/exports** ディレクトリーをエクスポートします。このリソースは、**nfsgroup** リソースグループに含まれます。これにより、NFSv4 クライアントの仮想ディレクトリーが構築されます。このエクスポートには、NFSv3 クライアントもアクセスできます。

```
[root@z1 ~]# pcs resource create nfs-root exportfs \
clientspec=192.168.122.0/255.255.255.0 \
options=rw,sync,no_root_squash \
directory=/nfsshare/exports \
fsid=0 --group nfsgroup

[root@z1 ~]# # pcs resource create nfs-export1 exportfs \
clientspec=192.168.122.0/255.255.255.0 \
options=rw,sync,no_root_squash directory=/nfsshare/exports/export1 \
fsid=1 --group nfsgroup

[root@z1 ~]# # pcs resource create nfs-export2 exportfs \
clientspec=192.168.122.0/255.255.255.0 \
options=rw,sync,no_root_squash directory=/nfsshare/exports/export2 \
fsid=2 --group nfsgroup
```

5. NFS 共有にアクセスするために、NFS クライアントが使用するフローティング IP アドレスリソースを追加します。指定するフローティング IP アドレスには DNS リバースルックアップが必要になりますが、クラスターのすべてのノードで **/etc/hosts** にフローティング IP アドレスを指定して対処することもできます。このリソースは、リソースグループ **nfsgroup** に含まれます。このデプロイメント例では、192.168.122.200 をフローティング IP アドレスとして使用します。

```
[root@z1 ~]# pcs resource create nfs_ip IPaddr2 \
ip=192.168.122.200 cidr_netmask=24 --group nfsgroup
```

6. NFS デプロイメント全体が初期化されたら、NFSv3 の再起動通知を送信する **nfsnotify** リソースを追加します。このリソースは、リソースグループ **nfsgroup** に含まれます。



注記

NFS の通知が適切に処理されるようにするには、フローティング IP アドレスにホスト名が関連付けられており、それが NFS サーバーと NFS クライアントで同じである必要があります。

```
[root@z1 ~]# pcs resource create nfs-notify nfsnotify \
source_host=192.168.122.200 --group nfsgroup
```

リソースとリソースの制約を作成したら、クラスターのステータスを確認できます。すべてのリソースが同じノードで実行していることに注意してください。

```
[root@z1 ~]# pcs status
```

```
...
Full list of resources:
 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM):   Started z1.example.com
     nfsshare   (ocf::heartbeat:Filesystem):    Started z1.example.com
     nfs-daemon (ocf::heartbeat:nfsserver):     Started z1.example.com
     nfs-root   (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs-export1        (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs-export2        (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs_ip     (ocf::heartbeat:IPaddr2):       Started  z1.example.com
     nfs-notify (ocf::heartbeat:nfsnotify):     Started z1.example.com
...
```

3.6. リソース設定のテスト

以下の手順を使用するとシステムの設定を検証できます。NFSv3 または NFSv4 のいずれかで、エクスポートされたファイルシステムをマウントできるはずです。

1. デプロイメントと同じネットワークにあるクラスター外部のノードで NFS 共有をマウントして、NFS 共有が表示されることを確認します。この例では、192.168.122.0/24 ネットワークを使用します。

   ```
   # showmount -e 192.168.122.200
   Export list for 192.168.122.200:
   /nfsshare/exports/export1 192.168.122.0/255.255.255.0
   /nfsshare/exports         192.168.122.0/255.255.255.0
   /nfsshare/exports/export2 192.168.122.0/255.255.255.0
   ```

2. NFSv4 で NFS 共有をマウントできることを確認する場合は、クライアントノードのディレクトリーに NFS 共有をマウントします。マウントしたら、エクスポートディレクトリーの内容が表示されることを確認します。テスト後に共有をアンマウントします。

   ```
   # mkdir nfsshare
   # mount -o "vers=4" 192.168.122.200:export1 nfsshare
   # ls nfsshare
   clientdatafile1
   # umount nfsshare
   ```

3. NFSv3 で NFS 共有をマウントできることを確認します。マウント後、テストファイル **clientdatafile1** が表示されることを確認します。NFSv4 とは異なり NFSv3 は仮想ファイルシステムを使用しないため、特定のエクスポートをマウントする必要があります。テスト後に共有をアンマウントします。

   ```
   # mkdir nfsshare
   # mount -o "vers=3" 192.168.122.200:/nfsshare/exports/export2 nfsshare
   # ls nfsshare
       clientdatafile2
   # umount nfsshare
   ```

4. フェイルオーバーをテストするには、以下の手順を実行します。

   a. クラスター外部のノードに NFS 共有をマウントし、「NFS 共有の設定」 で作成した **clientdatafile1** にアクセスできることを確認します。

```
# mkdir nfsshare
# mount -o "vers=4" 192.168.122.200:export1 nfsshare
# ls nfsshare
clientdatafile1
```

b. クラスター内で、**nfsgroup** を実行しているノードを確認します。この例では、**nfsgroup** が **z1.example.com** で実行しています。

```
[root@z1 ~]# pcs status
...
Full list of resources:
 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM):   Started z1.example.com
     nfsshare   (ocf::heartbeat:Filesystem):    Started z1.example.com
     nfs-daemon (ocf::heartbeat:nfsserver):     Started z1.example.com
     nfs-root   (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs-export1        (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs-export2        (ocf::heartbeat:exportfs):      Started z1.example.com
     nfs_ip     (ocf::heartbeat:IPaddr2):       Started  z1.example.com
     nfs-notify (ocf::heartbeat:nfsnotify):     Started z1.example.com
...
```

c. クラスター内のノードから、**nfsgroup** を実行しているノードをスタンバイモードにします。

```
[root@z1 ~]# pcs node standby z1.example.com
```

d. **nfsgroup** が、別のクラスターノードで正常に起動することを確認します。

```
[root@z1 ~]# pcs status
...
Full list of resources:
 Resource Group: nfsgroup
     my_lvm     (ocf::heartbeat:LVM):   Started z2.example.com
     nfsshare   (ocf::heartbeat:Filesystem):    Started z2.example.com
     nfs-daemon (ocf::heartbeat:nfsserver):     Started z2.example.com
     nfs-root   (ocf::heartbeat:exportfs):      Started z2.example.com
     nfs-export1        (ocf::heartbeat:exportfs):      Started z2.example.com
     nfs-export2        (ocf::heartbeat:exportfs):      Started z2.example.com
     nfs_ip     (ocf::heartbeat:IPaddr2):       Started  z2.example.com
     nfs-notify (ocf::heartbeat:nfsnotify):     Started z2.example.com
...
```

e. NFS 共有をマウントしたクラスターの外部のノードから、この外部ノードが NFS マウント内のテストファイルに引き続きアクセスできることを確認します。

```
# ls nfsshare
clientdatafile1
```

ファイルオーバー中、クライアントに対するサービスは一時的に失われますが、クライアントはユーザーが介入しなくても回復するはずです。デフォルトでは、NFSv4 を使用するクライアントの場合は、マウントの復旧に最大 90 秒かかることがあります。この 90 秒

は、システムの起動時にサーバーが監視する NFSv4 ファイルのリースの猶予期間です。NFSv3 クライアントでは、数秒でマウントへのアクセスが回復します。

f. クラスター内で、最初に **nfsgroup** を実行していたノードをスタンバイモードから削除します。ただし、スタンバイモードから回復しただけでは、クラスターリソースがこのノードに戻りません。

```
[root@z1 ~]# pcs node unstandby z1.example.com
```

注記

ノードを **スタンバイ** モードから削除しても、リソースはそのノードにフェイルオーバーしません。これは、リソースの **resource-stickiness** 値により異なります。**resource-stickiness** メタ属性の詳細は、『High Availability Add-On リファレンス』の現在のノードを優先させるリソースの設定を参照してください。

第4章 RED HAT HIGH AVAILABILITY クラスター (RED HAT ENTERPRISE LINUX 7.4 以降) のアクティブ/アクティブ SAMBA サーバー

Red Hat Enterprise Linux 7.4 リリースでは、Red Hat の Resilient Storage Add-On は、Pacemaker を使用してアクティブ/アクティブクラスター設定で Samba を実行するサポートを提供します。Red Hat の Resilient Storage Add-On には High Availability Add-On が含まれます。

注記

Samba のサポートポリシーの詳細は、Red Hat カスタマーポータルの Support Policies for RHEL Resilient Storage - ctdb General Policies および Support Policies for RHEL Resilient Storage - Exporting gfs2 contents via other protocols を参照してください。

本章では、共有ストレージを使用した 2 ノードの Red Hat Enterprise Linux High Availability Add-On クラスターでアクティブ/アクティブ Samba サーバーを設定する方法を説明します。この手順では、Pacemaker クラスターリソースの設定に **pcs** を使用します。

このユースケースでは、システムに以下のコンポーネントが必要です。

- Clustered Samba を実行しているクラスターの作成に使用する 2 つのノード。この例で使用するノードは **z1.example.com** と **z2.example.com** で、それぞれの IP アドレスは **192.168.1.151** と **192.168.1.152** です。
- 各ノード用の電源フェンスデバイス、ここでは APC 電源スイッチの 2 ポートを使用しています。この例では、APC 電源スイッチの 2 ポートを使用します。ホスト名は **zapc.example.com** です。
- iSCSI またはファイバーチャネルを使用する、クラスターのノードに対する共有ストレージ。

2 ノード Red Hat Enterprise Linux High Availability Add-On クラスターで高可用性アクティブ/アクティブ NFS サーバーを設定するには、以下のステップを実行する必要があります。

1. 「クラスターの作成」 の説明に従い、Samba 共有をエクスポートして、クラスターの各ノードに対するフェンシングを設定します。
2. 「GFS2 ファイルシステムでのクラスター化 LVM ボリュームの設定」 の説明に従い、クラスターのノードに対する共有ストレージ上のクラスター化された LVM 論理ボリューム **my_clv** にマウントした **gfs2** ファイルシステムを設定します。
3. 「Samba の設定」 を参照してクラスターの各ノードで Samba を設定します。
4. 「Samba クラスターリソースの設定」 の説明に従って、Samba クラスターリソースを作成します。
5. 「リソース設定のテスト」 に従って、設定した Samba 共有をテストします。

4.1. クラスターの作成

次の手順に従って、Samba サービスに使用するクラスターのインストールと作成を行います。

1. 「クラスターソフトウェアのインストール」 の手順に従って、**z1.example.com** および **z2.example.com** ノードにクラスターソフトウェアをインストールします。

2. 「クラスターの作成」 で説明されている手順を使用して、**z1.example.com** および **z2.example.com** で設定される 2 ノードクラスターを作成します。この手順の例と同様に、クラスターには **my_cluster** という名前が付けられます。

3. 「排他処理の設定」 の説明に従って、クラスターの各ノードにフェンスデバイスを設定します。この例では、ホスト名が **zapc.example.com** という APC 電源スイッチの 2 つのポートを使用してフェンシングが設定されます。

4.2. GFS2 ファイルシステムでのクラスター化 LVM ボリュームの設定

このユースケースでは、クラスターのノード間で共有しているストレージにクラスター化 LVM 論理ボリュームを作成する必要があります。

本項では、クラスター化 LVM 論理ボリュームを GFS2 ファイルシステムでそのボリューム上に作成します。この例では **/dev/vdb** 共有パーティションを使って LVM 論理ボリュームの作成元となる LVM 物理ボリュームを格納します。

注記

LVM ボリュームと、クラスターノードで使用するパーティションおよびデバイスは、クラスターノード以外には接続しないでください。

この手順を始める前に、Resilient Storage チャンネルの **lvm2-cluster** と **gfs2-utils** パッケージをクラスターの両方のノードにインストールします。

```
# yum install lvm2-cluster gfs2-utils
```

/dev/vdb パーティションは共有させるストレージとなるため、この手順は 1 つのノードでのみ行います。

1. グローバル Pacemaker パラメーター **no_quorum_policy** を **freeze** に設定します。この設定により、クォーラムが失われるたびにクラスター全体にフェンシングが発生しなくなります。このポリシーの設定についての詳細は、『Global File System 2』 を参照してください。

   ```
   [root@z1 ~]# pcs property set no-quorum-policy=freeze
   ```

2. **dlm** リソースをセットアップします。これは、**clvmd** サービスと GFS2 ファイルシステムに必要な依存関係です。

   ```
   [root@z1 ~]# pcs resource create dlm ocf:pacemaker:controld op monitor interval=30s on-fail=fence clone interleave=true ordered=true
   ```

3. **clvmd** をクラスターリソースとしてセットアップします。

   ```
   [root@z1 ~]# pcs resource create clvmd ocf:heartbeat:clvm op monitor interval=30s on-fail=fence clone interleave=true ordered=true
   ```

   この開始手順の一環として、**ocf:heartbeat:clvm** リソースエージェントにより、**/etc/lvm/lvm.conf** ファイルの **locking_type** パラメーターが **3** に変更され、**lvmetad** デーモンが無効化されることに注意してください。

4. **clvmd** および **dlm** の依存関係をセットアップし、順番に起動します。**clvmd** リソースは **dlm** の後に起動し、**dlm** と同じノードで実行する必要があります。

```
[root@z1 ~]# pcs constraint order start dlm-clone then clvmd-clone
Adding dlm-clone clvmd-clone (kind: Mandatory) (Options: first-action=start then-action=start)
[root@z1 ~]# pcs constraint colocation add clvmd-clone with dlm-clone
```

5. **dlm** および **clvmd** リソースが全てのノードで実行されていることを確認します。

```
[root@z1 ~]# pcs status
...
Full list of resources:
...
 Clone Set: dlm-clone [dlm]
     Started: [ z1 z2 ]
 Clone Set: clvmd-clone [clvmd]
     Started: [ z1 z2 ]
```

6. クラスター化論理ボリュームの作成

```
[root@z1 ~]# pvcreate /dev/vdb
[root@z1 ~]# vgcreate -Ay -cy cluster_vg /dev/vdb
[root@z1 ~]# lvcreate -L4G -n cluster_lv cluster_vg
```

7. ボリュームが正しく作成されているかどうかを確認するには、論理ボリュームを表示する **lvs** コマンドを使用します。

```
[root@z1 ~]# lvs
  LV         VG         Attr       LSize ...
  cluster_lv cluster_vg -wi-ao---- 4.00g
  ...
```

8. GFS2 でボリュームをフォーマットします。この例では、**my_cluster** がクラスター名です。また、2 つのジャーナルを示すために **-j 2** を指定しています。これは、ジャーナルの数が、クラスターのノードの数に一致する必要があるためです。

```
[root@z1 ~]# mkfs.gfs2 -p lock_dlm -j 2 -t my_cluster:samba /dev/cluster_vg/cluster_lv
```

9. ファイルシステムのマウントと管理を行うように Pacemaker を設定するための **Filesystem** リソースを作成します。この例では、**fs** という **Filesystem** リソースを作成し、両方のクラスターで **/mnt/gfs2share** を作成します。

```
[root@z1 ~]# pcs resource create fs ocf:heartbeat:Filesystem
device="/dev/cluster_vg/cluster_lv" directory="/mnt/gfs2share" fstype="gfs2" --clone
```

10. GFS2 ファイルシステムと **clvmd** のサービスとの依存関係を設定し、順番に起動します。GFS2 は **clvmd** の後に起動し、**clvmd** と同じノードで実行する必要があります。

```
[root@z1 ~]# pcs constraint order start clvmd-clone then fs-clone
Adding clvmd-clone fs-clone (kind: Mandatory) (Options: first-action=start then-action=start)
[root@z1 ~]# pcs constraint colocation add fs-clone with clvmd-clone
```

11. 想定通りに GFS2 ファイルがマウントされていることを確認します。

```
[root@z1 ~]# mount |grep /mnt/gfs2share
/dev/mapper/cluster_vg-cluster_lv on /mnt/gfs2share type gfs2 (rw,noatime,seclabel)
```

4.3. SAMBA の設定

以下の手順では、Samba 環境を初期化し、クラスターノードで Samba を設定します。

1. クラスターの両方のノードで、以下の手順を実行します。

   a. **samba**、**ctdb**、**cifs-utils** をインストールします。

   ```
   # yum install samba ctdb cifs-utils
   ```

   b. **firewalld** デーモンを実行している場合は、以下のコマンドを実行して、 **ctdb** と **samba** サービスに必要なポートを有効にします。

   ```
   # firewall-cmd --add-service=ctdb --permanent
   # firewall-cmd --add-service=samba --permanent
   # firewall-cmd --reload
   ```

   c. 以下のコマンドを実行して、ctdb や samba サービスが動作しておらず、ブート時に起動しないようにします。これら 2 つのサービスがお使いのシステムに存在して動作しているわけではないことに注意してください。

   ```
   # systemctl disable ctdb
   # systemctl disable smb
   # systemctl disable nmb
   # systemctl disable winbind
   # systemctl stop ctdb
   # systemctl stop smb
   # systemctl stop nmb
   # systemctl stop winbind
   ```

   d. **/etc/samba/smb.conf** ファイルでは、Samba サーバーを設定して **[public]** 共有定義を設定します。以下に例を示します。

   ```
   # cat << END > /etc/samba/smb.conf
   [global]
   netbios name = linuxserver
   workgroup = WORKGROUP
   server string = Public File Server
   security = user
   map to guest = bad user
   guest account = smbguest
   clustering = yes
   ctdbd socket = /tmp/ctdb.socket
   [public]
   path = /mnt/gfs2share/public
   guest ok = yes
   read only = no
   END
   ```

この例で見られるように Samba をスタンドアローンサーバーとして設定する方法や、**testparm** ユーティリティーで **smb.conf** ファイルを検証する方法は、『システム管理者のガイド』の ファイルとプリントサーバーを参照してください。

e. クラスターノードの IP アドレスを **/etc/ctdb/nodes** ファイルに追加します。

```
# cat << END > /etc/ctdb/nodes
192.168.1.151
192.168.1.152
END
```

f. クラスターのノード間における負荷分散は、このクラスターによってエクスポートされた Samba 共有へのアクセスに使用できる 2 つ以上の IP アドレスを **/etc/ctdb/public_addresses** ファイルに追加できます。この IP は、Samba サーバーの名前の DNS で設定する必要があるアドレスで、SMB クライアントが接続するアドレスです。複数の IP アドレスで 1 つのタイプ A の DNS レコードとして Samba サーバーの名前を設定し、ラウンドロビンがクラスターのノードにわたりクライアントを分散できるようにします。

この例では、DNS エントリー **linuxserver.example.com** が、**/etc/ctdb/public_addresses** ファイル下にリストされている両方のアドレスで定義されています。これにより、DNS によって、ラウンドロビン方式でクラスターノードにわたり Samba クライアントが分散されます。この操作を行う際、DNS エントリーがニーズに一致する必要があります。

このクラスターによってエクスポートされた Samba 共有へのアクセスに使用できる IP アドレスを **/etc/ctdb/public_addresses** ファイルに追加します。

```
# cat << END > /etc/ctdb/public_addresses
192.168.1.201/24 eth0
192.168.1.202/24 eth0
END
```

g. Samba グループを作成し、パブリックテスト共有ディレクトリーのローカルユーザーを追加して、以前に作成したグループをプライマリーグループとして設定します。

```
# groupadd smbguest
# adduser smbguest -g smbguest
```

h. CTDB 関連のディレクトリーで SELinux コンテキストが 正しいことを確認してください。

```
# mkdir /var/ctdb/
# chcon -Rv -u system_u -r object_r -t ctdbd_var_lib_t /var/ctdb/
changing security context of ‘/var/ctdb/’
# chcon -Rv -u system_u -r object_r -t ctdbd_var_lib_t /var/lib/ctdb/
changing security context of ‘/var/lib/ctdb/’
```

2. クラスターの 1 つのノードで、以下の手順に従います。

a. CTDB ロックファイルとパブリック共有のディレクトリーを設定します。

```
[root@z1 ~]# mkdir -p /mnt/gfs2share/ctdb/
[root@z1 ~]# mkdir -p /mnt/gfs2share/public/
```

b. GFS2 共有上の SELinux コンテキストを更新します。

```
[root@z1 ~]# chown smbguest:smbguest /mnt/gfs2share/public/
[root@z1 ~]# chmod 755 /mnt/gfs2share/public/
[root@z1 ~]# chcon -Rv -t ctdbd_var_run_t /mnt/gfs2share/ctdb/
changing security context of ‘/mnt/gfs2share/ctdb/’
[root@z1 ~]# chcon -Rv -u system_u -r object_r -t samba_share_t /mnt/gfs2share/public/
changing security context of ‘/mnt/gfs2share/public’
```

4.4. SAMBA クラスターリソースの設定

本項では、このユースケースで Samba クラスターリソースを設定する手順について説明します。

次の手順では、**samba.cib** というクラスターの **cib** のスナップショットを作成し、実行しているクラスターで直接リソースを設定するのではなく、リソースをテストファイルに追加します。リソースと制約を設定すると、この手順により、実行しているクラスター設定に **samba.cib** がプッシュされます。

クラスターの 1 ノードで、以下の手順を行います。

1. クラスター設定ファイルの **cib** ファイルのスナップショットを作成します。

```
[root@z1 ~]# pcs cluster cib samba.cib
```

2. Samba が使用する CTDB リソースを作成します。このリソースは、両方のクラスターノードで実行できるようにクローンリソースとして作成してください。

```
[root@z1 ~]# pcs -f samba.cib resource create ctdb ocf:heartbeat:CTDB \
ctdb_recovery_lock="/mnt/gfs2share/ctdb/ctdb.lock" \
ctdb_dbdir=/var/ctdb ctdb_socket=/tmp/ctdb.socket \
ctdb_logfile=/var/log/ctdb.log \
op monitor interval=10 timeout=30 op start timeout=90 \
op stop timeout=100 --clone
```

3. クローン Samba サーバーを作成します。

```
[root@z1 ~]# pcs -f samba.cib resource create samba systemd:smb --clone
```

4. クラスターリソースのコロケーションと順序の制約を作成します。起動順序は、Filesystem リソース、CTDB リソース、Samba リソースです。

```
[root@z1 ~]# pcs -f samba.cib constraint order fs-clone then ctdb-clone
Adding fs-clone ctdb-clone (kind: Mandatory) (Options: first-action=start then-action=start)
[root@z1 ~]# pcs -f samba.cib constraint order ctdb-clone then samba-clone
Adding ctdb-clone samba-clone (kind: Mandatory) (Options: first-action=start then-
action=start)
[root@z1 ~]# pcs -f samba.cib constraint colocation add ctdb-clone with fs-clone
[root@z1 ~]# pcs -f samba.cib constraint colocation add samba-clone with ctdb-clone
```

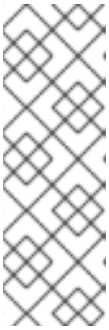
5. **cib** スナップショットのコンテンツをクラスターにプッシュします。

```
[root@z1 ~]# pcs cluster cib-push samba.cib
CIB updated
```

6. クラスターのステータスを確認し、リソースが実行していることを確認します。

Red Hat Enterprise Linux 7.4 では、CTDB による Samba の起動、共有のエクスポート、安定化にしばらく時間がかかることがあります。このプロセスの前にクラスターステータスを確認すると、CTDB ステータスの呼び出しが失敗したというメッセージが表示されることがあります。このプロセスが完了すれば、**pcs resource cleanup ctdb-clone** コマンドを使用して表示されたメッセージを消去できます。

```
[root@z1 ~]# pcs status
Cluster name: my_cluster
Stack: corosync
Current DC: z1.example.com (version 1.1.16-12.el7_4.2-94ff4df) - partition with quorum
Last updated: Thu Oct 19 18:17:07 2017
Last change: Thu Oct 19 18:16:50 2017 by hacluster via crmd on z1.example.com

2 nodes configured
11 resources configured

Online: [ z1.example.com z2.example.com ]

Full list of resources:

 myapc  (stonith:fence_apc_snmp):       Started z1.example.com
 Clone Set: dlm-clone [dlm]
     Started: [ z1.example.com z2.example.com ]
 Clone Set: clvmd-clone [clvmd]
     Started: [ z1.example.com z2.example.com ]
 Clone Set: fs-clone [fs]
     Started: [ z1.example.com z2.example.com ]
 Clone Set: ctdb-clone [ctdb]
     Started: [ z1.example.com z2.example.com ]
 Clone Set: samba-clone [samba]
     Started: [ z1.example.com z2.example.com ]
```



注記

設定したリソースが実行されていない場合は、**pcs resource debug-start *resource*** コマンドを実行してリソースの設定をテストできます。このコマンドは、クラスターの制御や認識の範囲外でサービスを起動します。設定したリソースが再度実行された場合は、**pcs resource cleanup *resource*** コマンドを実行してクラスターが更新を認識するようにします。**pcs resource debug-start** コマンドの詳細は、『High Availability Add-On リファレンス』 の クラスターリソースの有効化、無効化、および禁止 の項を参照してください。

4.5. リソース設定のテスト

Samba の設定に成功した場合は、クラスターのノードで Samba 共有をマウントできます。以下の例の手順では、Samba 共有をマウントしています。

1. クラスターノードの既存のユーザーを **smbpasswd** ファイルに追加してパスワードを割り当てます。以下の例では、既存のユーザー **smbuser** を追加しています。

```
[root@z1 ~]# smbpasswd -a smbuser
New SMB password:
Retype new SMB password:
Added user smbuser
```

2. Samba 共有をマウントします。

```
[root@z1 ~]# mkdir /mnt/sambashare
[root@z1 ~]# mount -t cifs -o user=smbuser //198.162.1.151/public /mnt/sambashare
Password for smbuser@//198.162.1.151/public:  ********
```

3. ファイルシステムがマウントされているかどうかを確認します。

```
[root@z1 ~]# mount | grep /mnt/sambashare
//198.162.1.151/public on /mnt/sambashare type cifs
(rw,relatime,vers=1.0,cache=strict,username=smbuser,domain=LINUXSERVER,uid=0,noforceu
id,gid=0,noforcegid,addr=10.37.167.205,unix,posixpaths,serverino,mapposix,acl,rsize=1048576
wsize=65536,echo_interval=60,actimeo=1)
```

Samba の復元を確認するには、以下の手順を行います。

1. 以下のコマンドを使用して CTDB リソースを手動で停止します。

```
[root@z1 ~]# pcs resource debug-stop ctdb
```

2. このリソースを停止すると、システムによってサービスが復元されます。**pcs status** コマンドを使用してクラスターのステータスを確認します。**ctdb-clone** リソースが開始したことがわかりますが、**ctdb_monitor** がエラーしたこともわかります。

```
[root@z1 ~]# pcs status
...
Clone Set: ctdb-clone [ctdb]
     Started: [ z1.example.com z2.example.com ]
...
Failed Actions:
* ctdb_monitor_10000 on z1.example.com 'unknown error' (1): call=126, status=complete,
exitreason='CTDB status call failed: connect() failed, errno=111',
    last-rc-change='Thu Oct 19 18:39:51 2017', queued=0ms, exec=0ms
...
```

このステータスのエラーを消去するには、クラスターノードの 1 つで以下のコマンドを実行します。

```
[root@z1 ~]# pcs resource cleanup ctdb-clone
```

付録A 改訂履歴

改訂 6-1 7.7 GA 公開用ドキュメントの準備	Wed Aug 7 2019	Steven Levine
改訂 5-2 7.6 GA 公開用ドキュメントの準備	Thu Oct 4 2018	Steven Levine
改訂 4-2 7.5 GA 公開用ドキュメントの準備	Wed Mar 14 2018	Steven Levine
改訂 4-1 7.5 ベータ版公開用ドキュメントの準備	Thu Dec 14 2017	Steven Levine
改訂 3-4 7.4 のバージョンを更新	Wed Aug 16 2017	Steven Levine
改訂 3-3 7.4 GA 公開用ドキュメントバージョン	Wed Jul 19 2017	Steven Levine
改訂 3-1 7.4 ベータ版公開用ドキュメントの準備	Wed May 10 2017	Steven Levine
改訂 2-6 7.3 の更新	Mon Apr 17 2017	Steven Levine
改訂 2-4 7.3 GA リリースのバージョン	Mon Oct 17 2016	Steven Levine
改訂 2-3 7.3 Beta 公開用ドキュメントの準備	Fri Aug 12 2016	Steven Levine
改訂 1.2-3 7.2 GA 公開用ドキュメントの準備。	Mon Nov 9 2015	Steven Levine
改訂 1.2-2 7.2 ベータ公開用ドキュメントの準備	Tue Aug 18 2015	Steven Levine
改訂 1.1-19 7.1 GA リリース向けのバージョン	Mon Feb 16 2015	Steven Levine
改訂 1.1-10 7.1 ベータリリース向けバージョン	Thu Dec 11 2014	Steven Levine
改訂 0.1-33 7.0 GA リリース向けバージョン	Mon Jun 2 2014	Steven Levine