PowerGres Plus SR & Pacemaker pgsql RA

動作検証レポート

1.0.1 版
2014 年 8 月 13 日

SRA OSS, Inc. 日本支社
〒170-0022 東京都豊島区南池袋 2-32-8 8F
Tel. 03-5979-2701 Fax. 03-5979-2702
http://www.sraoss.co.jp/

目次

1. はじめに

SRA OSS, Inc 日本支社では、PostgreSQL に信頼性とセキュリティをプラスした PowerGres Plus というデータベース製品を販売しております。PostgreSQL にはまだ実装されていない、トランザクションログの二重化やデータ暗号化機能を管理ツールから簡単に利用することが可能です。

本ドキュメントは、PowerGres Plus に搭載されているデータ暗号化機能と PostgreSQL のストリーミングレプリケーションについての概要や構築方法、Pacemaker という HA クラスタソフトウェアとの連携について解説し、PowerGres Plus を利用する際の助けとなることを目的としています。

2. 概要

PowerGres Plus データ暗号化機能と PostgreSQL ストリーミングレプリケーション、Pacemaker によるクラスタ構成について、それぞれどのようなものか説明します。

2.1. データ暗号化機能

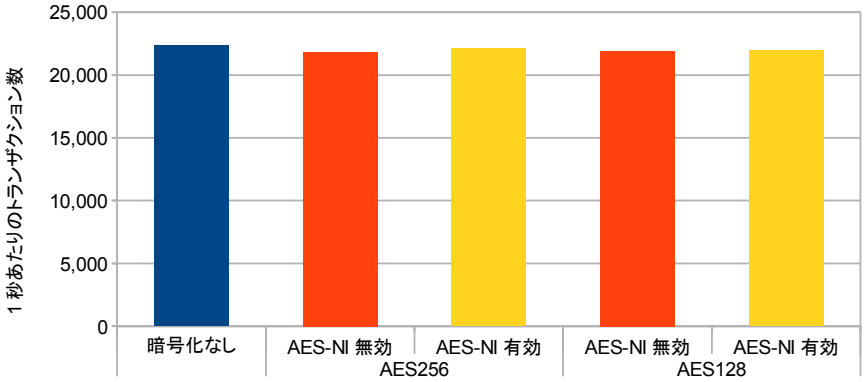
PowerGres Plus V9.1 では、業界標準の強力な暗号化アルゴリズム AES を用いて、データベースに格納するデータはもちろん、バックアップデータやトランザクションログ、一時ファイルまで暗号化できます。

PostgreSQL の場合、格納されているデータが平文のままなため直接ファイルをのぞくとデータが見えてしまいますが、暗号化されている場合は、直接ファイルをのぞかれてもデータを読むことはできません。これによりセキュリティ基準として広がりつつあるクレジットカード会社 5 社によって策定された PCI DSS (Payment Card Industry Data Security Standard)の求めるデータの暗号化基準を満たすことができます。

データの暗号化、復号は自動的に行われるため、アプリケーションが暗号化を意識する必要はありません。また、暗号化、復号はディスク I/O が発生するときにのみ行われ、バッファ・キャッシュ内のデータアクセスでは行われないため、全体としてのオーバーヘッドが少ないという特長があります。

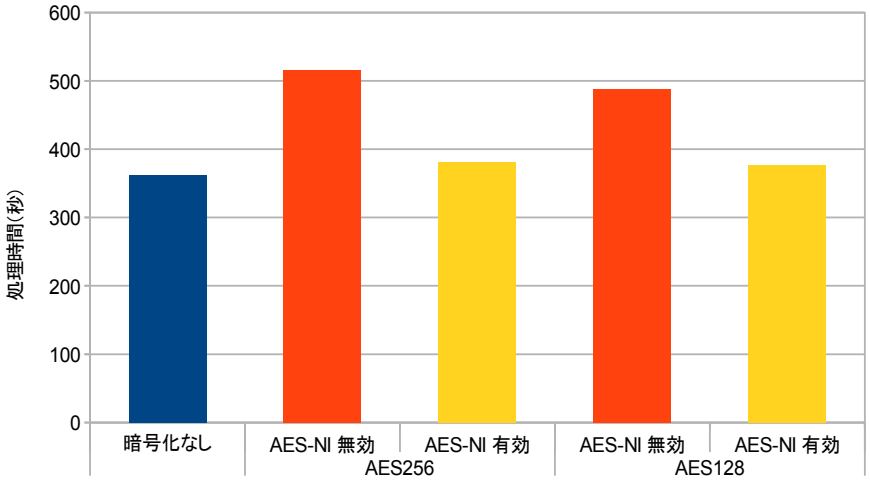
次のグラフは、暗号化なし、暗号化あり(AES128)、暗号化あり(AES256)の 3 パターンにての OLTP 性能を測定した際の TPS をグラフ化したものです。ご覧のように、暗号化機能の利用有無によるオーバーヘッドはほ

とんどないことがわかります。



測定結果（１）　*OLTP* 性能

次のグラフは、バッチ処理の処理時間を暗号化の有無で比較したものです。ディスク I/O が連続的となる処理では 3 割程度のオーバーヘッドが確認できます。



測定結果（２）　バッチ処理性能

測定結果グラフの左から 3 番目と 5 番目は Intel Xeon プロセッサの 5600 番台以降に搭載されている「AES-NI」を利用して測定したものです。「AES-NI」を利用した場合では、暗号化と復号によるオーバヘッドがほとんど無くなっています。

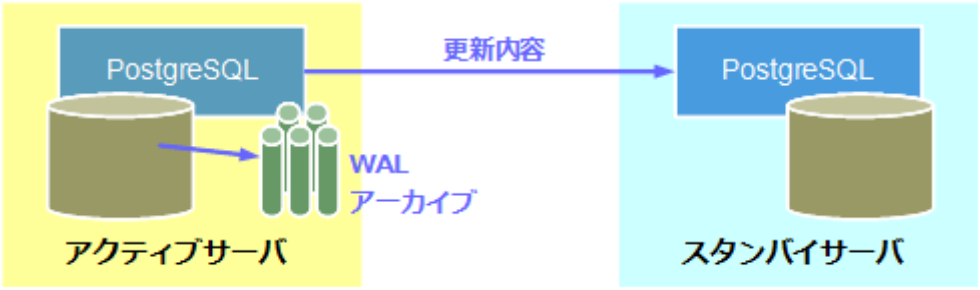
測定環境は右の通りです。

ハードウェア	FUJITSU PRIMERGY RX300 S7 CPU: Intel Xeon E5-2690 2.90GHz(8コア)×2 DRAM: 160GB ストレージ: SSD(PCI Express)
ソフトウェア	OS: Red Hat Enterprise Linux 6.2(64ビット) RDBMS: PowerGres Plus V9.1(64ビット)

2.2. ストリーミングレプリケーション

ストリーミングレプリケーションは、PostgreSQL9.0 以降のバージョンで利用できる、PostgreSQL 本体のレプリケーション機能です。参照/更新が可能なアクティブサーバのデータベースへの更新内容を、スタンバイサーバのデータベースへ転送することで、データベースを複製し続けることができます。

非同期レプリケーションは、待機系データベースに更新内容が反映されるまでには若干の遅延がありますが、比較的遅延は少なく、稼働系データベースへの負荷が小さいレプリケーション方式です。



また、アーカイブモードを有効にしておくことで、更新内容が書かれた WAL ファイルを指定したコマンドにてアーカイブし続けることができ、スタンバイサーバにて極端に同期遅延が発生した場合でも、WAL アーカイブを利用して最新の状態まで自動的に追いつくことができます。

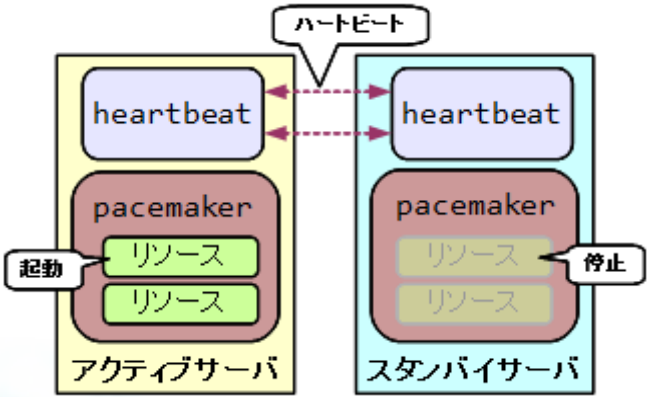
ホットスタンバイの機能を有効にした場合には、スタンバイサーバにて参照クエリを受け付けることが可能です。ただし、自動的にスタンバイサーバに参照クエリを割り振る機能は持っていないため、クライアント側で用意する必要があります。

PostgreSQL のバージョン 9.1 以降では、更新内容(WAL)をスタンバイサーバにも書き込んでから、クライアントに応答を返す、同期レプリケーションも利用できるようになっています。PowerGres Plus V9.1 は PostgreSQL9.1 をベースにしていますので、同期レプリケーションの利用が可能です。

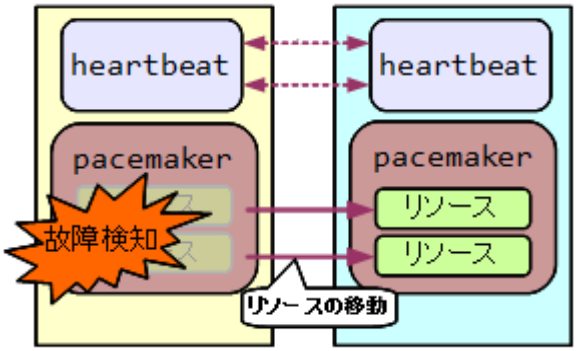
2.3. Pacemaker と pgsql RA

本検証では、HA クラスタのノードを監視する部分に heartbeat というソフトウェアを利用し、リソースを制御する部分に pacemaker というソフトウェアを利用します。

heartbeat は、「ハートビート」と呼ぶパケットを他ノードに向けて定期的に送信し死活監視を行います。一定以上応答のないノードは、ダウンしたものとみなします。本検証では 2 台のサーバにて行いますので、2 台のノードが互いに死活監視を行うこととなります。



pacemaker は、HA クラスタにおいてサービスを提供するために必要な構成要素であるリソースの管理を行います。本検証では、PostgreSQL の起動、停止、監視などを行う pgsql リソースと、仮想 IP の起動、停止、監視を行う仮想 IP リソースを登録します。pacemaker は pgsql や稼働 IP の状態に応じて、リソースの再起動を試みたり、フェイルオーバを行ったりします。

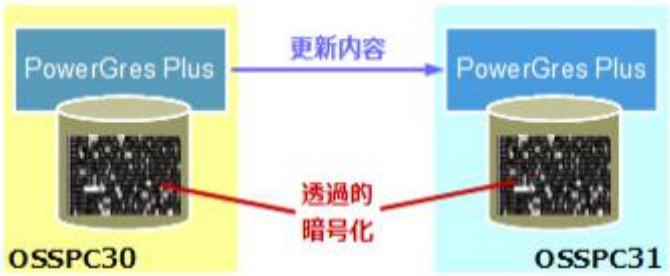


リソースの起動、停止、監視などの操作は、リソースエージェント（RA）と呼ばれる実行ファイルにて行います。

3. 暗号化レプリケーションのセットアップ

3.1. 検証環境について

3.1.1. 構成



検証環境として「HP ProLiant MicroServer Gen8 」を 2 台利用します。
ホスト名は osspc30 と osspc31 であるものとします。

```
[osspc30 ~]# cat /etc/redhat-release
CentOS release 6.4 (Final)
[osspc30 ~]# uname -a
Linux osspc30 2.6.32-358.el6.x86_64 #1 SMP Fri Feb 22 00:31:26 UTC
2013 x86_64 x86_64 x86_64 GNU/Linux
```

root ユーザにてソフトウェアのインストールを行い、Pacemaker の操作は root にて、PowerGres Plus の管理ツール（PowerGres マネージャ）の操作や、データベースの操作は postgres ユーザにて行う前提で記載します。

3.1.2. OS 側の準備

両サーバにて、あらかじめ、postgresql、pacemaker、heartbeat のパッケージがインストールされている場合は削除します。

コマンド例や作業手順の右上の「Active」「Standby」という記載は、アクティブ側のサーバ、スタンバイ側のサーバのどちらでこの操作を行うかをあらわします。両方書いてある場合には、両サーバで行います。

Active
Standby

```
[osspc30 ~]#rpm -qa |grep postgresql
[osspc30 ~]#rpm -qa |grep pacemaker
[osspc30 ~]#rpm -qa |grep heartbeat
```

SELinux とファイアウォールは無効にして検証を行っています。

3.2. ソフトウェア入手

PowerGres Plus Linux 版の評価版を以下のＵＲＬからダウンロードします。執筆時点(2014 年 7 月 22 日)での最新バージョンは 9.1 update3 ですが、メジャーバージョンが 9.1 であれば別バージョンでも同じようにインストールおよび動作させることが可能です。

```
http://powergres.sraoss.co.jp/s/ja/download/download.php
```

製品を購入されている場合は、インストールメディアを利用してインストールします。

3.3. PowerGres Plus

3.3.1. インストール

Active
Standby

osspc30, osspc31 の両サーバにて、ダウンロードした製品を解凍しインストールします。

```
[osspc30]# tar zxf powergresplus-9.1update3.tar.gz
[osspc30]# cd powergresplus-9.1update3
[osspc30]# sh install.sh
本ソフトウェアプログラムは、以下の「使用許諾契約書」の条件にしたがって
使用許諾されます。したがって、必ず本製品をインストールする前に「使用許
      :
Do you agree to the license terms (yes/no): yes

Please enter the license key (XXXX-XXXX-XXXX-XXXX): Evaluation
```

使用許諾契約書の内容に同意し、ライセンスキーを入力します。評価版を利用する場合は "Evaluation" と入力すると、60 日間無料で利用できます。なお、カーネルパラメータの設定値が推奨値よりも少ない場合は、増やすかどうか尋ねられますので、yes と入力し増加させます。

```
Installing packages...
準備中...                ########################################## [100%]
  1:powergresplus91-libs   ########################################## [  9%]
  2:powergresplus91        ########################################## [ 18%]
      :
```

```
 PowerGres Plus V9.1 installation completed.
[osspc30]#
```

PowerGres Plus は以下のディレクトリにインストールされます。

```
[osspc30]# ls /opt/powergresplus91/
bin  etc  include  lib  libexec  share
```

3.3.2. 管理ツールの起動

Active
Standby

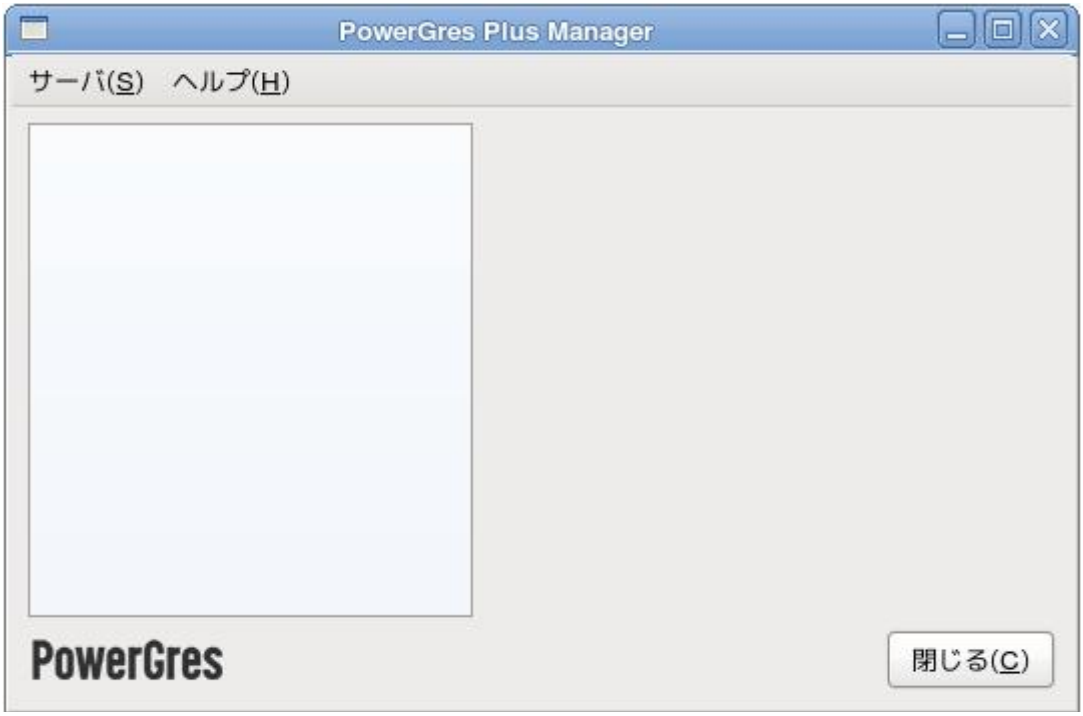
PowerGres Plus の管理ツール（PowerGres マネージャ）の操作や、データベースを管理するユーザは、このドキュメントでは postgres ユーザにて行うこととし、postgres ユーザとして GUI ログインします。

postgres ユーザの ~/.bash_profile に以下の行を追加します。

```
PATH=/opt/powergresplus91/bin:$PATH
MANPATH=/opt/powergresplus91/share/man:$MANPATH
LD_LIBRARY_PATH=/opt/powergresplus91/lib
export PATH MANPATH LD_LIBRARY_PATH
```

環境変数を適用後、それぞれのサーバにて管理ツールを起動します。

```
[osspc30]$ . .bash_profile
[osspc30]$ powergres-mgr &
```



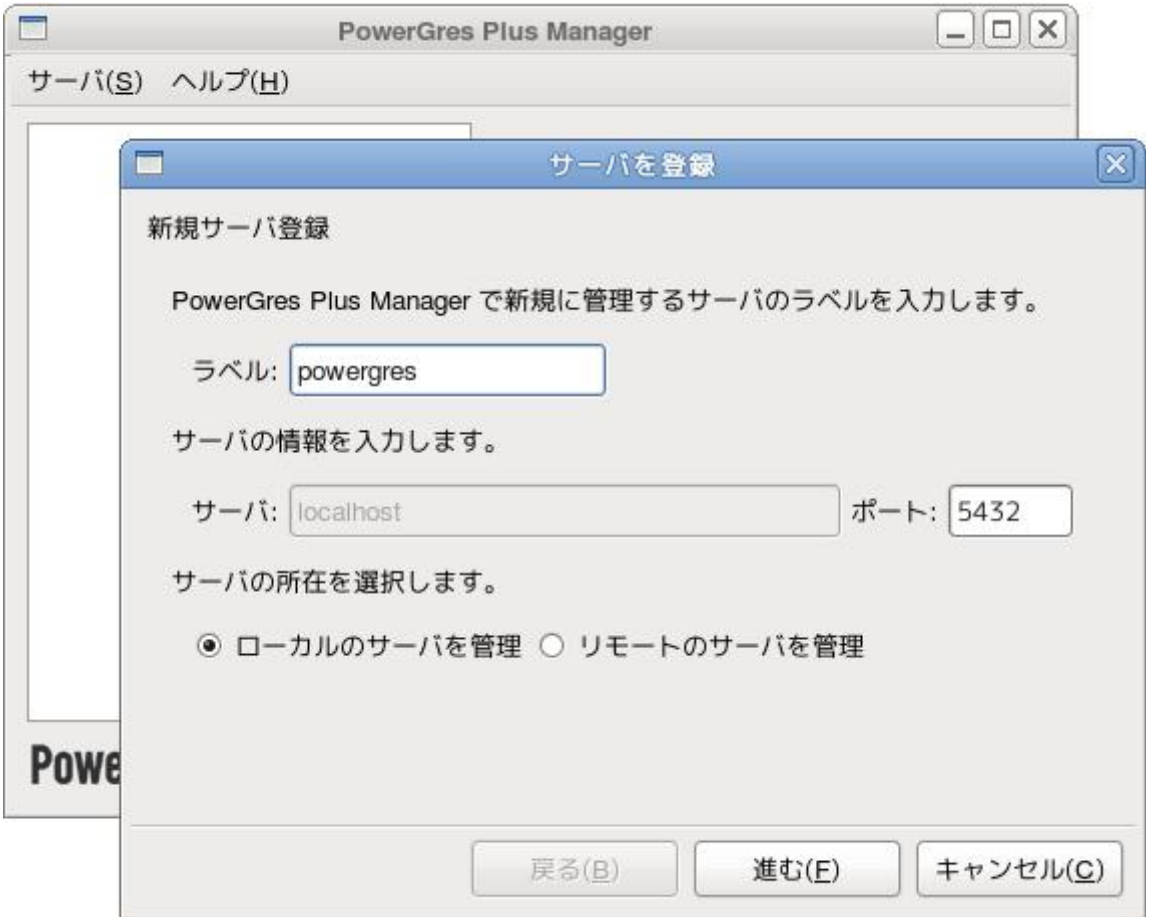
Active

3.3.3. 新規サーバの登録

PostgreSQL および PowerGres では、導入時には必ずデータベースクラスタと呼ばれる、データを格納する領域の初期化を行います。通常、挿入されたデータはデータベースクラスタ内に保存されますが、本検証で利用する PowerGres Plus の暗号化機能は、暗号化して保存するデータは別途、保存する領域を指定します。

この操作は片側のノードのみで行います。

「サーバ(S)」メニューから「サーバを登録(A)...」を選択し、新規サーバを登録します。



「ラベル」には、管理ツールでサーバを識別するための任意の名前を指定してください。ここでは「powergres」としています。「ポート」はデフォルトで「5432」が指定されていますが必要に応じて変更します。

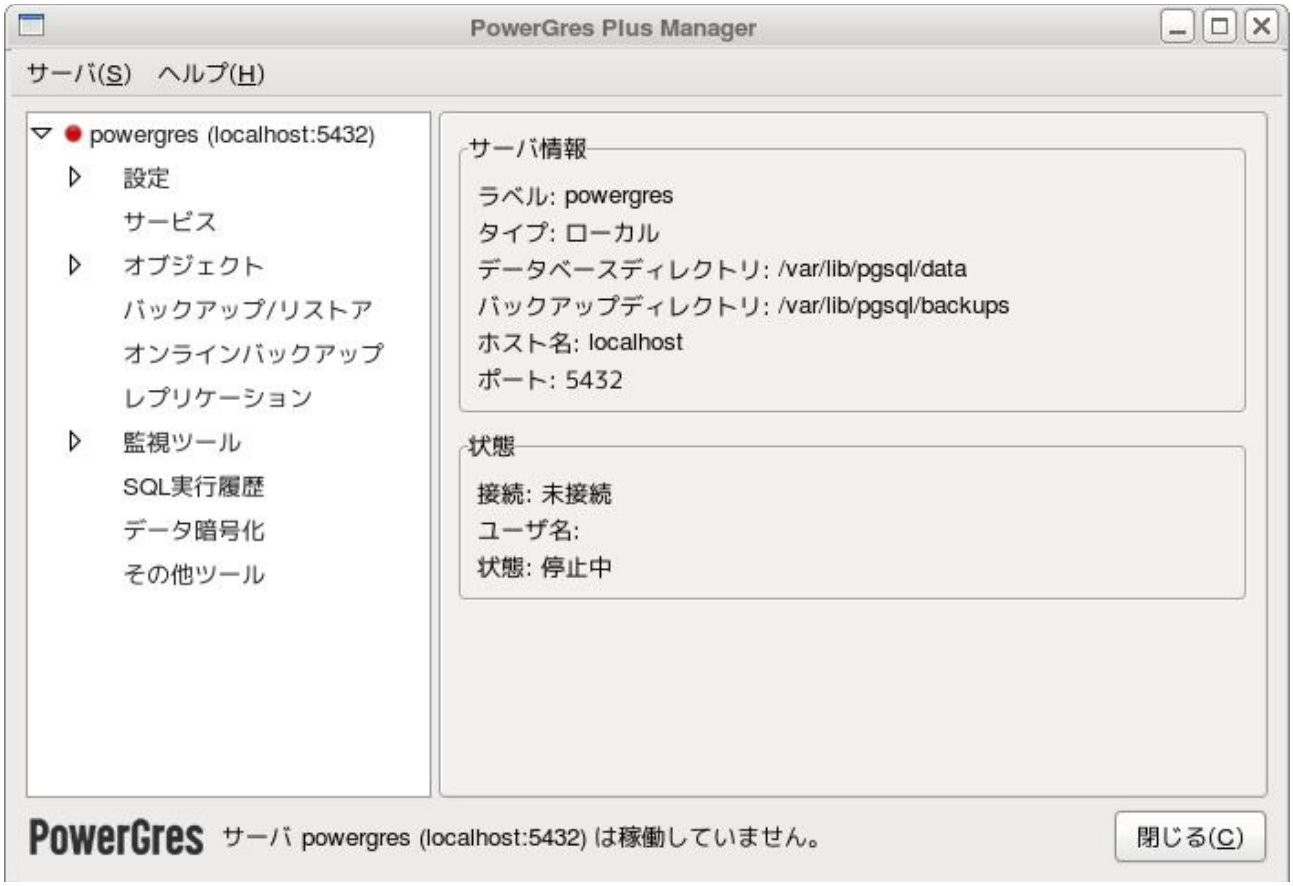
「データベースディレクトリ」で指定した領域は PowerGres のデータベースクラスタ（データ格納領域）として初期化されます。「バックアップディレクトリ」には、信頼性向上のため 2 重化されたトランザクションログが格納されます。また、オンラインバックアップ機能利用時のバックアップ先として利用されます。

ここでは、それぞれ「/var/lib/pgsql/data」「/var/lib/pgsql/backups」を指定します。

サーバの種類は「新規データベースクラスタを作成」を選択してください。

「文字エンコーディング」にはデフォルトで利用する文字エンコーディングを、「スーパーユーザ名」「パスワード」には PowerGres を操作するスーパーユーザとパスワードを指定してください。

新規サーバ登録を完了すると、下記のように作成したサーバ情報が登録されます。

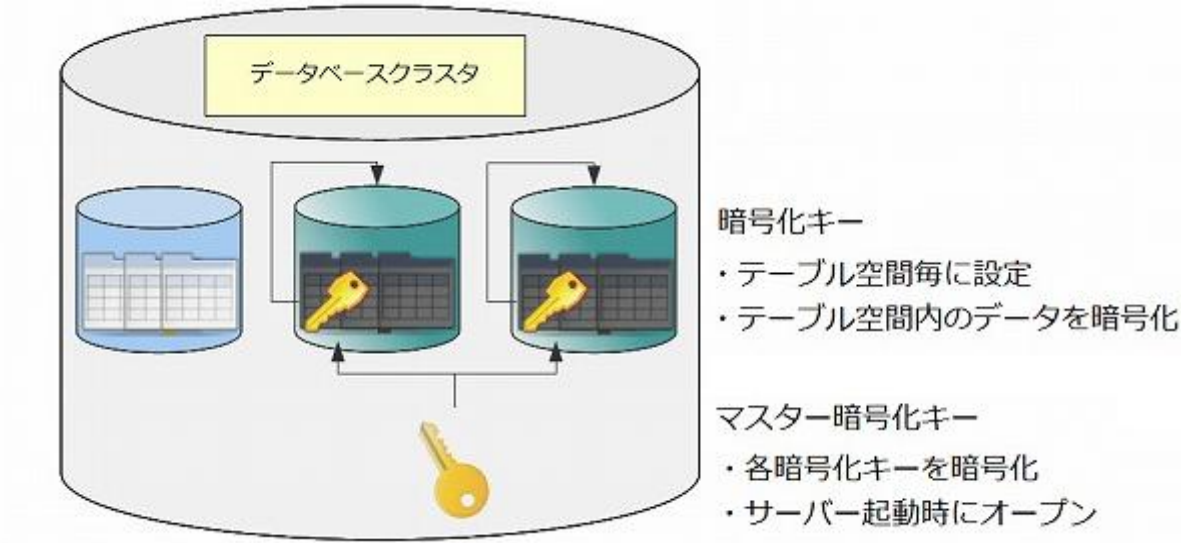


うまく登録できない場合は、下記ページ PowerGres Plus 管理ツールマニュアルの「作成できない場合のヒント」をご覧ください。

http://powergres.sraoss.co.jp/manual/Plus/V91/admin/server.html

サービスの起動は、サブメニュー「サービス」より「サービスを開始」をクリックします。

3.4. 透過的データ暗号化



透過的データ暗号化では、二層のキーストアファイルによってデータを保護しています。暗号化されたテーブル空間には、その中の全てのデータを暗号化 / 復号するキーストアファイルがあり、更にこのキーストアファイルはデータベースクラスタに 1 つだけ作成されるマスター暗号化キーによって暗号化されます。

マスター暗号化キーは指定したパスフレーズに基づいて暗号化され、キーストアファイルに保存されます。

3.4.1. マスター暗号化キーの作成

Active

まず、暗号化キーを格納するディレクトリを設定し、マスター暗号化キーを作成します。

PowerGres マネージャのサブメニュー「データ暗号化」より、「キーストア格納ディレクトリ」を設定します。ここでは、「/var/lib/pgsql/keystore」としています。

サブメニュー「サービス」の「サービスを再起動」にて、設定の反映を行い、「マスター暗号化キーを作成...」をクリックしてマスター暗号化キーを作成します。

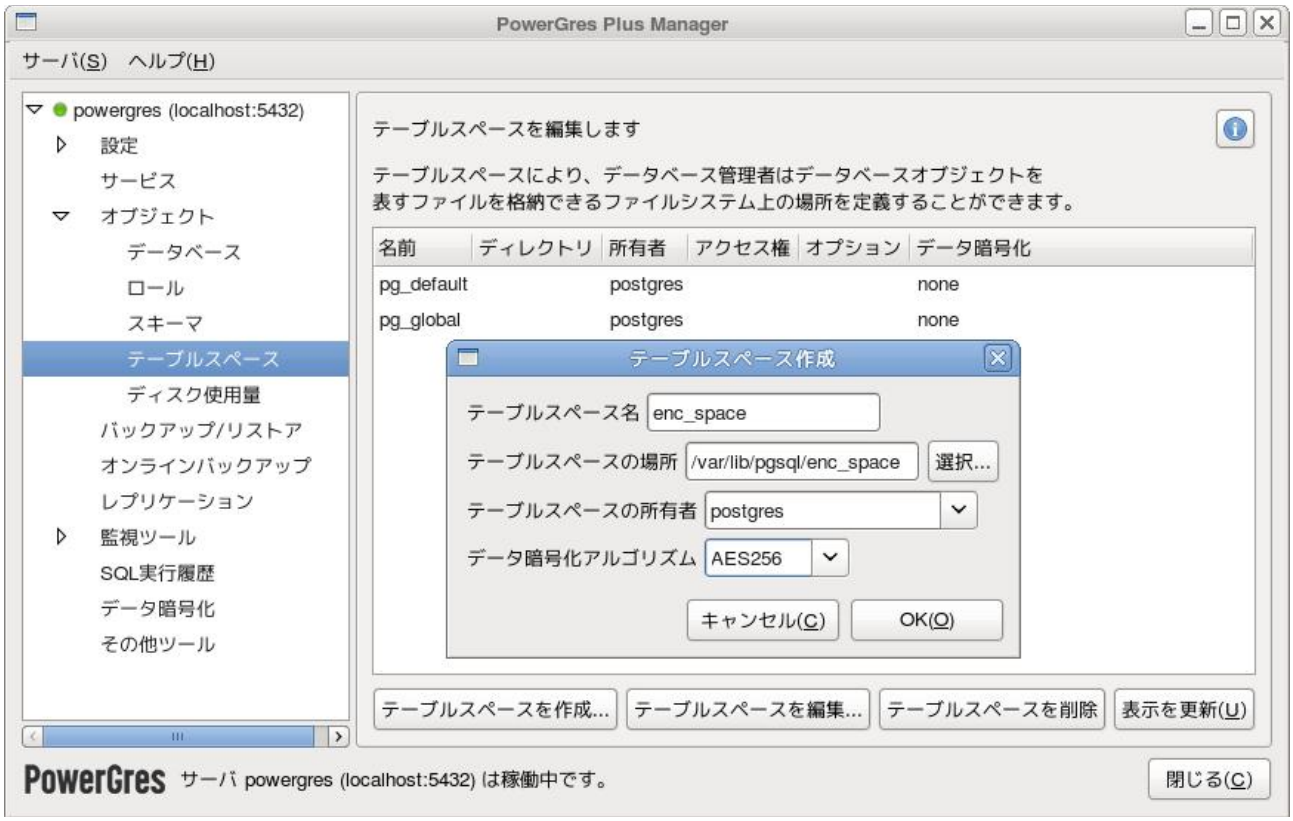
マスター暗号化キーは、このパスフレーズで保護されますので、推測されにくい文字列を指定してください。また、パスフレーズを忘れると暗号化されたデータには 2 度とアクセスできなくなりますのでご注意ください。

次に「キーストアの自動オープンを有効にする...」をクリックし、サービスの起動に合わせて自動的にキーストアがオープンされるようにしておきます。

3.4.2. 暗号化テーブルスペースの作成

Active

PowerGres マネージャのサブメニュー「オブジェクト」の左側にある三角の矢印をクリックし「テーブルスペース」を選択し、「テーブルスペースを作成...」をクリックします。

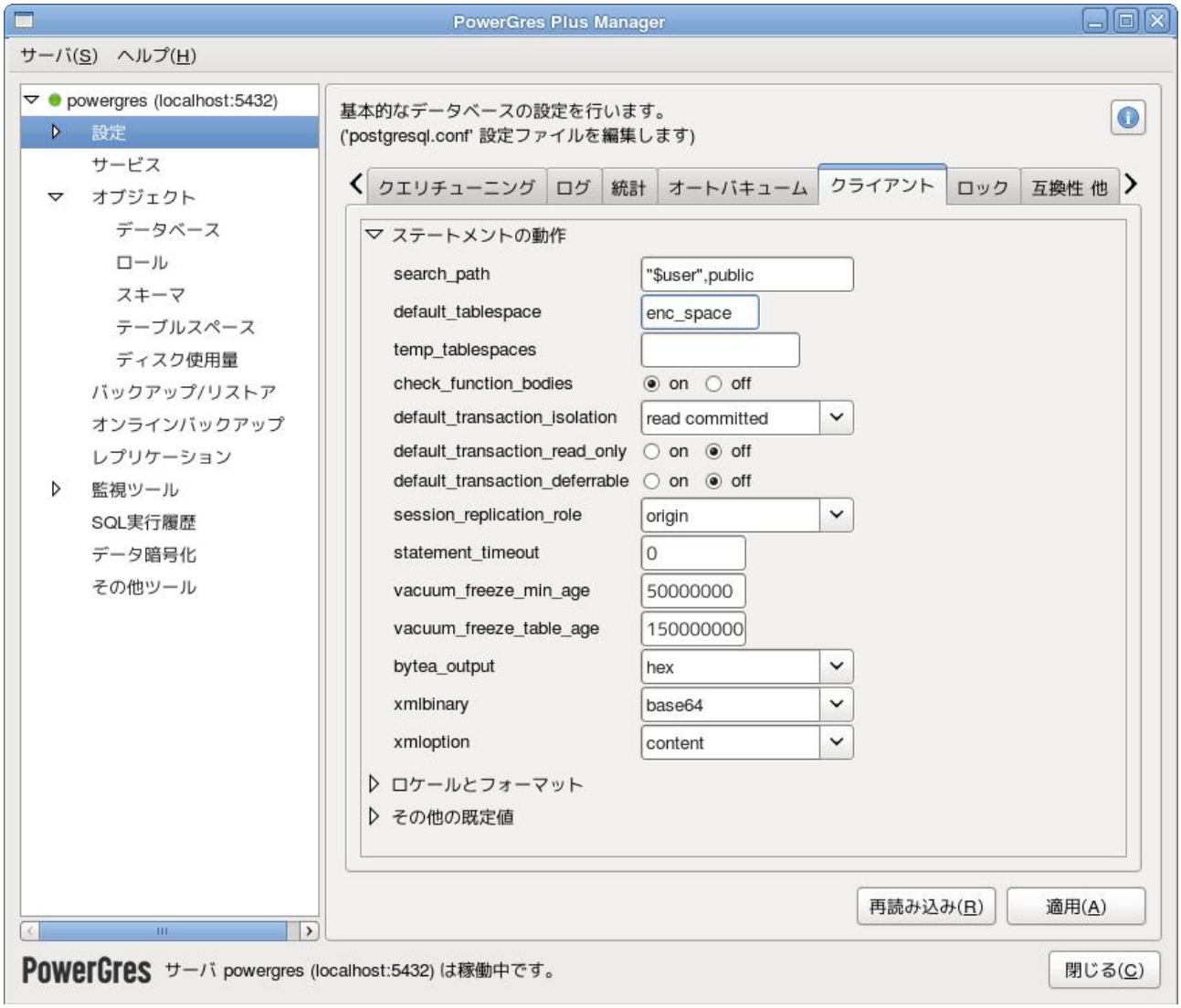


ここでは、テーブルスペース名は「enc_space」とし、場所を「/var/lib/pgsql/enc_space」としています。データベースで利用する全てのデータを透過的に暗号化する場合は、この領域に全てのデータを格納することになります。

「データ暗号化アルゴリズム」は「AES128」「AES256」から選択します。

次に、今後作成する全てのテーブルが暗号化されたテーブルスペースに作成されるよう、デフォルトで利用するテーブルスペースの設定を変更します。

管理ツールのサブメニュー「設定」より「クライアント」タブの「ステートメントの動作」を選択し、「default_tablespace」という項目に作成したテーブルスペースの名称を入力して「適用」をクリックします。



設定の適用後は、サブメニュー「サービス」より「設定を再読み込み(R)」を行ってください。

3.5. ストリーミングレプリケーション

3.5.1. 稼働系サーバの設定

Active

PowerGres マネージャの「サーバ(S)」メニューから「ホットスタンバイを構築(H)...」をクリック、ホットスタンバイ構築ウィザードを起動し、「稼働系サーバの設定」を選択します。

「待機系サーバの数」はデフォルトで「3」が指定されていますが、待機サーバの最大数を指定します。

「レプリケーション用のスーパーユーザ」は、待機系サーバから稼動系サーバにレプリケーション接続する際に利用するユーザで、PowerGres では「repli」というユーザで固定されており、新しく作成されます。指定したパスワードは、待機系サーバ構築時に必要になります。

「IP アドレスの範囲」には、待機系サーバが存在している IP アドレス範囲を指定してください。

3.5.2. 待機系サーバの設定

Standby

待機系サーバに postgres ユーザにて GUI ログインし、稼働系と同じ要領で PowerGres マネージャを起動します。「サーバ(S)」メニューから「ホットスタンバイを構築(H)...」をクリック、ホットスタンバイ構築ウィザードを起動し、「待機系サーバの設定」を選択します。

待機系サーバの「ラベル」には任意の名前を指定します。ここでは「powergres」としておきます。

「ポート」は通常は「5432」を指定します。データベースディレクトリは、稼働系サーバと同じ「/var/lib/pgsql/data」を指定します。

稼働系サーバへの設定には、稼働系サーバへ接続するための設定を入力します。

「ホスト」には稼働系サーバの IP アドレスまたはホスト名を指定してください。「ポート」には稼働系サーバで PowerGres Plus が動作しているポート番号を指定します。「パスワード」には、稼働系サーバ構築時に指定した、レプリケーション接続ユーザ「repli」に設定したパスワードを設定します。

設定が完了すると、上記のようにサーバが登録されますので、稼働系サーバと同様に、サブメニュー「データ暗号化」より「キーストアの自動オープンを有効にする...」を選択し、サービス起動に合わせて自動的にキーストアがオープンされるように設定後、サブメニュー「サービス」より「スタンバイ開始」をクリックします。

3.5.3. 暗号化レプリケーションの確認

暗号化されたテーブルへの更新が待機系サーバへレプリケーションされ、待機系サーバにて更新されたデータが参照できることを確認します。

稼働系サーバと待機系サーバそれぞれにて、PowerGres マネージャを起動し、サブメニュー「その他ツール」より「psql コマンドラインツールの実行」ボタンをクリックするか、bash から postgres ユーザにて psql コマンドを実行し psql 端末を起動します。

SHOW 文にてデフォルトのテーブルスペースが「enc_space」になっていることを確認し、任意にテーブルを作成後データを INSERT してみます。

Active

```
psql (9.1.12)
"help" でヘルプを表示します.

postgres=# SHOW default_tablespace;
 default_tablespace
--------------------
 enc_space
(1 行)

postgres=# CREATE TABLE t1 (name text);
CREATE TABLE
postgres=# INSERT INTO t1 VALUES ('hoge');
INSERT 0 1
postgres=#
```

INSERT したデータが待機系にレプリケーションされていることを確認します。

Standby

```
postgres=# SELECT * FROM t1;
 name
------
 hoge
(1 行)
```

試しに待機サーバのキーストアの自動オープンを無効にした場合、データの参照ができないことを確認してみます。

PowerGres マネージャのサブメニュー「データ暗号化」より、「キーストアの自動オープンを無効にする...」をクリックし、サブメニュー「サービス」より「サービスを停止」、「スタンバイ開始」にて再起動します。

この状態で待機サーバにて先ほどと同じクエリを発行すると、以下のようにデータの複号ができない旨が表示され、ERROR となることが確認できました。

Standby

```
postgres=# SELECT * FROM t1;
ERROR:  could not encrypt or decrypt data because the keystore is not open
HINT:  Open the existing keystore, or set the master encryption key to
create and open a new keystore
postgres=#
```

4. Pacemaker

4.1. インストール

当検証で必要最低限の設定を行います。特に Pacemaker の各リソース設定についての細かい解説はしていません。

4.1.1. リポジトリパッケージの入手

Active
Standby

Pacemaker は Linux-HA JAPAN にて公開されている Pacemaker リポジトリパッケージを使用します。検証環境は 64bit 版の CentOS6.4 のため、下記 URL より「pacemaker-1.0.13-1.2.el6.x86_64.repo.tar.gz」をダウンロードしています。

```
http://linux-ha.sourceforge.jp/wp/dl/packages
```

インストールは、稼働系サーバ、待機系サーバ共に行い、ここから先の操作は root ユーザにて行います。

4.1.2. yum インストール

Active
Standby

外部ネットワークが有効な場合は、ダウンロードしたリポジトリパッケージ内のリポジトリファイルを利用して Pacemaker と関連パッケージの yum インストールを行います。yum により必要パッケージが自動的にインストールされるため便利です。

まず、「/etc/yum.repos.d/CentOS-Base.repo」ファイルの「base」と「updates」に、以下の行を追加し CentOS 同梱版の Pacemaker やその関連パッケージを管理対象からはずします。

```
exclude=pacemaker pacemaker-libs corosync cluster-glue heartbeat resource-agents
```

次に、ダウンロードしたリポジトリパッケージを展開し、pacemaker リポジトリを設置し yum コマンドにてインストールを行います。

```
[osspc30]# cd /tmp
[osspc30]# tar zxf ~/pacemaker-1.0.13-1.2.el6.x86_64.repo.tar.gz
[osspc30]# cd pacemaker-1.0.13-1.2.el6.x86_64.repo/
[osspc30]# cp pacemaker.repo /etc/yum.repos.d/
[osspc30]# yum install pacemaker.x86_64 heartbeat.x86_64 pm_extras.x86_64
```

インストールされる heartbeat、pacemaker、pm_extras、cluster-glue、corosync、libesmtp、resorce-agents 等のパッケージが下記の例のように pacemaker リポジトリからインストールされることを確認してください。

```
================================================================================================================
 Package                    Arch            Version                   Repository           Size
================================================================================================================
Installing:
 heartbeat                  x86_64          3.0.5-1.1.el6             pacemaker           162 k
 pacemaker                  x86_64          1.0.13-1.el6              pacemaker           5.6 M
 pm_extras                  x86_64          1.3-1.el6                 pacemaker            24 k
Installing for dependencies:
 PyXML                      x86_64          0.8.4-19.el6              base                892 k
 cluster-glue               x86_64          1.0.11-1.el6              pacemaker           261 k
 cluster-glue-libs          x86_64          1.0.11-1.el6              pacemaker           110 k
 corosync                   x86_64          1.4.5-1.el6               pacemaker           164 k
 corosynclib                x86_64          1.4.5-1.el6               pacemaker           143 k
 heartbeat-libs             x86_64          3.0.5-1.1.el6             pacemaker           263 k
 libesmtp                   x86_64          1.0.4-16.el6              pacemaker            57 k
 pacemaker-libs             x86_64          1.0.13-1.el6              pacemaker           262 k
 resource-agents            x86_64          3.9.5-1.el6               pacemaker           459 k

Transaction Summary
================================================================================================================
Install      12 Package(s)

Total download size: 8.3 M
Installed size: 18 M
Is this ok [y/N]:
```

また、上記例の PyXML のように、依存関係により OpenIPMI-libs、net-snmp-libs、lm_sensors-libs、openhpi-libs、perl-TimeDate、PyXML、openssl、uuid などのパッケージがインストールされていない場合は、base や updates リポジトリなどからインストールされます。

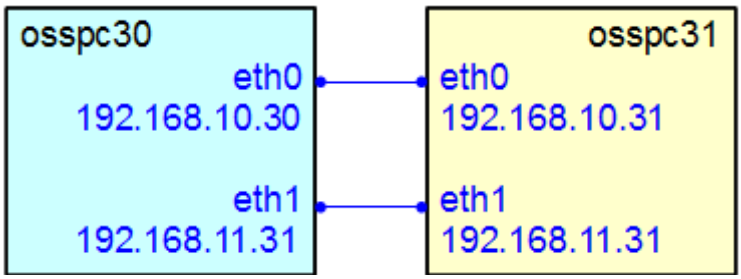
外部ネットワークに繋がっていない場合は、依存関係により必要になるパッケージを手動でインストールし、pacemaker リポジトリのみを指定して yum インストールを行います。

4.2. Heartbeat の設定

Active
Standby

主に稼働系サーバと待機系サーバで相互に死活監視を行うための「ハートビート通信」の設定を行います。

4.2.1. ネットワークについて



ここでは、192.168.10.0/24 というサービス用のネットワークと 192.168.11.0/24 というレプリケーション用のネットワークを用意し、それぞれのサーバの eth0、eth1 にて上記の IP アドレスが設定されている想定で記載します。ハートビート通信は eth0、eth1 両方を利用します。

```
[osspc30 ~]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

192.168.10.30  osspc30
192.168.10.31  osspc31
192.168.10.10  service_vip
192.168.11.30  osspc30
192.168.11.31  osspc31
192.168.11.10  replication_vip
```

/etc/hosts には osspc30,osspc31 の eth0 の IP アドレスと、後述する仮想 IP アドレスのエントリを追加しておきます。クライアントが接続するためのサービス用仮想 IP として「service_vip」、レプリケーション用の仮想 IP として、「replication_vip」を用意します。

4.2.2. authkeys ファイル

相手が同じ HA クラスタのノードかを識別するため、同じ認証キー（powergresplus_osspc3031）が設定されたファイルを稼働系、待機系両方に配置し、パーミッションを 0600 に変更します。

```
[osspc30 ~]# cat /etc/ha.d/authkeys
auth 1
1 sha1 powergresplus_osspc3031
[osspc31 ~]# cat /etc/ha.d/authkeys
auth 1
1 sha1 powergresplus_osspc3031
```

4.2.3. ha.cf（設定ファイル）

heartbeat の設定は以下の項目を追記しています。

```
pacemaker respawn                       # Pacemaker の利用を宣言
node osspc30                            # クラスタに参加するノード
node osspc31
bcast eth0 eth1                         # ハートビート通信を行うデバイスを
                                # ブロードキャストで指定
respawn root /usr/lib64/heartbeat/ifcheckd
                                # crm_mon の出力にノード情報を付加
```

設定ファイルは全く同じものを待機サーバへもコピーして配置し、サービスを起動します。また、自動起動の設定は、この検証では停止しておくこととします。

```
[osspc30]# scp /etc/ha.d/ha.cf osspc31:/etc/ha.d/
[osspc30]# service heartbeat start
[osspc30]# ssh osspc31 service heartbeat start
[osspc30]# chkconfig heartbeat off
[osspc30]# ssh osspc31 chkconfig heartbeat off
```

crm_mon コマンドによりハートビート通信の状態が確認できます。「-A」は「Node Attributes」を表示、「-i 1」は 1 秒間隔で監視します。

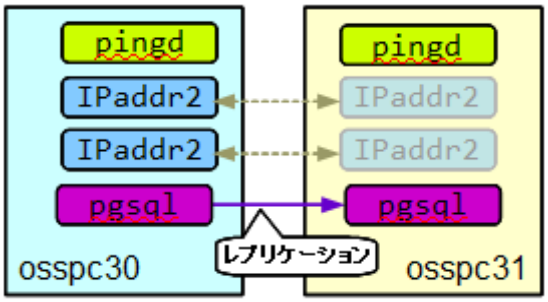
```
[osspc30]# crm_mon -A -i 1
 （省略）
Online: [ osspc30 osspc31 ]


Node Attributes:
* Node osspc30:
  + osspc31-eth0                    : up
  + osspc31-eth1                    : up
* Node osspc31:
  + osspc30-eth0                    : up
  + osspc30-eth1                    : up
```

上記のように表示されれば、eth0、eth1 それぞれにて互いに通信ができている状態といえます。

4.3. Pacemaker の設定

主にリソースの設定を行います。当レポートでは、検証を行うための必要最低限のリソースを設定します。



pgsql リソースは、PostgreSQL の起動、停止、監視を行うリソースです。PowerGres Plus 9.1 むけにも使用できます。ストリーミングレプリケーションを行っているため、マスターとスレーブの 2 つの状態がある、ms リソースとして登録します。

IPaddr2 リソースは仮想 IP リソースです。サービス用の仮想 IP とレプリケーション用の仮想 IP の 2 つのリソースを作成し、稼働系のノードで仮想 IP が起動するよう設定します。

ping リソースは、pingd リソースはネットワークの外部到達性チェックを行うためのリソースです。外部ネットワークに到達できないノードでは、マスターにならないよう制約を設定します。

4.3.1. リソース定義

リソースの設定は、全ての設定内容をファイルに記述し crm コマンドにてリソース設定を読み込みます。

```
primitive r_service_vip ocf:heartbeat:IPaddr2 \
  params nic="eth0" ip="192.168.10.10" cidr_netmask="24" \
  meta target-role="Started" is-managed="true"
primitive r_replication_vip ocf:heartbeat:IPaddr2 \
  params nic="eth1" ip="192.168.11.10" cidr_netmask="24" \
  meta migration-threshold="0"
group g_service r_service_vip r_replication_vip \
    meta target-role="Started"
```

上記は、仮想 IP リソースの設定です。eth0 にはサービス向けの仮想 IP を作成し、eth1 にはレプリケーション用の仮想 IP を作成します。2 つのリソースは必ず同じノードで起動するよう、グループ制約を設定します。

次に、pgsql リソースの設定例を記載しています。

```
primitive r_pgsql ocf:heartbeat:pgsql \
  params \
    pgctl="/opt/powergresplus91/bin/pg_ctl" \
    psql="/opt/powergresplus91/bin/psql" \
    pgdata="/var/lib/pgsql/data" \
    repuser="repli" rep_mode="sync" restore_command="false" \
    monitor_user="postgres" monitor_password="postgres" \
    start_opt="-p 5432" \
    node_list="osspc30 osspc31" \
    primary_conninfo_opt="keepalives_idle=60 \
      keepalives_interval=5 keepalives_count=5" \
    master_ip="192.168.11.10" \
    stop_escalate="0" \
  op start interval="0s" timeout="120s" on-fail="restart" \
  op stop interval="0s" timeout="120s" on-fail="block" \
  op monitor interval="7s" role="Master" timeout="30s" on-fail="restart" \
  op monitor interval="30s" timeout="30s" on-fail="restart" \
  op promote interval="0s" timeout="120s" on-fail="restart" \
  op demote interval="0s" timeout="120s" on-fail="block" \
  op notify interval="0s" timeout="90s"
```

pg_ctl や psql のコマンドパスは、PowerGres Plus のインストール先を指定します。データベースクラスタ位置を表す pgdata は、PowerGres Plus の管理ツールにて指定したディレクトリを設定します。

PowerGres Plus の管理ツールを利用してストリーミングレプリケーションを構築した場合には、レプリケーションユーザは repli という名前で自動作成されるため、repuser には repli を指定します。rep_mode は同期レプリケーション(sync)を設定し、WAL アーカイブの設定はしないこととします。

monitor_user、monitor_password は Pacemaker が PostgreSQL（PowerGres）の死活監視を行う際に利用するデータベースユーザを指定します。template1 データベースに接続可能なユーザを作成し設定します。ここでは、postgres ユーザを設定しています。

primary_conninfo_opt には待機系ノードの recovery.conf の primary_conninfo パラメータのオプションを指定します。ここでは、TCP キープアライブの idle 時間や、応答がない場合の再送間隔を設定しています。

master_ip は、レプリケーションの稼働系ノードの IP を指定しますが、ここではレプリケーション用に作成する仮想 IP を指定します。

stop_escalate には PostgreSQL（PowerGres）を停止する際、immediate（強制終了）モードを利用するまでの再試行回数を指定します。ここでは、直ちに停止できるよう"0"を指定しています。

op で始まる項目は、各監視オペレーションの設定を行っています。

```
ms ms_pgsql r_pgsql \
  meta \
    master-max="1" master-node-max="1" \
    clone-max="2" clone-node-max="1" notify="true" is-managed="true"
```

pgsql リソースは、両ノードで起動させ、稼働系の状態と待機系の状態を持つ、マスター／スレーブリソースとして登録します。

```
primitive r_pingd ocf:pacemaker:pingd \
  params name="default_ping_set" multiplier="100" \
    host_list="192.168.10.100"
clone c_pingd r_pingd
```

次に pingd というリソースを設定します。host_list で指定した IP アドレスに ping を送信し、成功すると multiplier 値（100）のスコアをセットする設定をしています。

pingd リソースは、両ノードにて起動するリソースのため作成したリソースをクローン化します。

```
location rsc_location-1 ms_pgsql \
  rule $role="master" 200: #uname eq osspc30 \
  rule $role="master" 100: #uname eq osspc31 \
  rule $role="master" -inf: defined fail-count-r_service_vip \
  rule -inf: not_defined default_ping_set or default_ping_set lt 100
```

上記では、ロケーション制約を設定しています。最初の２つでは、両ノードが正常な状態で全てのリソースの起動が片側のノード（osspc30）に集まるよう、osspc30 のスコアを大きくしています。

３つ目の制約は、サービス用の仮想 IP にて故障を検出した場合、そのノードがマスタになるスコアについて、マイナス無限大を設定しています。

４つ目の制約は、pingd リソースの結果が外部到達成功を表す 100 になっていない場合や、結果が見当たらない場合は、そのノードがマスタになることを避けるため、スコアをマイナス無限大に設定しています。

```
colocation rsc_colocation-1 inf: ms_pgsql c_pingd
colocation rsc_colocation-2 inf: g_service ms_pgsql:Master
order rsc_order-1 0: c_pingd ms_pgsql
order rsc_order-2 0: \
  ms_pgsql:promote g_service:start symmetrical=false
order rsc_order-3 0: \
  ms_pgsql:demote g_service:stop symmetrical=false
```

コロケーション制約は、リソースが同一ノードで起動することに対して、スコアを設定するものです。ここでは、ms_pgsql と c_pingd が同一ノードで起動すること、仮想 IP グループと pgsql のマスタノードが同一ノードで起動するよう制約をつけています。オーダー制約は、リソースの起動順序に制約をつけるものです。

```
property no-quorum-policy="ignore" stonith-enabled="false"
rsc_defaults resource-stickiness="INFINITY" migration-threshold="1"
```

全体設定として 2 つ設定します。no-quorum-policy は 2 台構成では"ignore"を、STONITH は利用しません。

rsc_defaults は各リソースのデフォルト値を設定します。 migration-threshold はフェイルオーバする際の閾値として利用され、"1"は 1 回の故障検出で、フェイルオーバすることを意味します。

4.3.2. リソース設定の適用

前項の全てのリソース設定を 1 枚のファイルに記述し、crm コマンドにて設定をロードします。設定前の状態としまして、両ノードの PowerGres は停止しておきます。これは、PowerGres Plus の管理ツールから停止しても良いですし、pg_ctl コマンドにて停止しても結構です。

Heartbeat は稼働系ノードだけで起動し、待機系は停止しておきます。

Active
```
[osspc30]# service heartbeat start
[osspc31]# service heartbeat stop
```
Standby

crm_mon コマンドにて osspc30 のみが Online になっていることを確認します。

Active
```
[osspc30]# crm_mon -i 1
(省略)
Online: [ osspc30 ]
```

crm コマンドにて設定をロードします。

Active
```
[osspc30]# crm configure load update crm_conf.txt
```

リソースの監視設定について WARNING が出力されることがあります。今回はリソース設定を簡素にするため、pgsql リソース以外は、監視オペレーションの設定をしていませんのでデフォルト値が採用されていますが、実際の環境では、各リソースについて op ではじまる監視設定を行ってください。

pingd が osspc30 で起動することで、osspc30 の PowerGres が起動します。一定時間後に pgsql リソースのステータスがプライマリに移行すると、仮想 IP リソースも起動します。

```
[osspc30]# crm_mon -Afr -i 1
(省略)
Online: [ osspc30 ]

Full list of resources:

 Resource Group: g_service
   r_service_vip       (ocf::heartbeat:IPaddr2):       Started osspc30
```

```
    r_replication_vip  (ocf::heartbeat:IPaddr2):       Started osspc30
 Master/Slave Set: ms_pgsql
    Masters: [ osspc30 ]
    Stopped: [ r_pgsql:1 ]
 Clone Set: c_pingd
    Started: [ osspc30 ]

Node Attributes:
* Node osspc30:
    + default_ping_set                  : 100
    + master-r_pgsql:0                  : 1000
    + r_pgsql-data-status               : LATEST
    + r_pgsql-master-baseline           : 000000002B0148D8
    + r_pgsql-status                    : PRI

Migration summary:
* Node osspc30:
```

pgsql-status はレプリケーション処理の動作状態をあらわします。当初は、「HS:alone」として起動しますが、一定時間経過することで、promote され「PRI」（プライマリ）に移行します。

pgsql-data-status はノードが保持しているデータの状態を示します。LATEST はマスタサーバになった時に設定され、HA クラスタ内で最も新しいデータを持っていることを示します。

上記の状態となりましたら、待機系の heartbeat を起動します。

Standby

```
[osspc31]# service heartbeat start
```

crm_mon の出力に osspc31 のノード情報が追加されます。

```
[osspc30]# crm_mon -Afr -i 1
(省略)
Online: [ osspc30 osspc31 ]

Full list of resources:
```

```
 Resource Group: g_service
    r_service_vip      (ocf::heartbeat:IPaddr2):       Started osspc30
    r_replication_vip  (ocf::heartbeat:IPaddr2):       Started osspc30
 Master/Slave Set: ms_pgsql
    Masters: [ osspc30 ]
    Slaves: [ osspc31 ]
 Clone Set: c_pingd
    Started: [ osspc30 osspc31 ]

Node Attributes:
* Node osspc30:
    + default_ping_set                  : 100
    + master-r_pgsql:0                  : 1000
    + osspc31-eth0                      : up
    + osspc31-eth1                      : up
    + r_pgsql-data-status               : LATEST
    + r_pgsql-master-baseline           : 000000002B014988
    + r_pgsql-status                    : PRI
* Node osspc31:
    + default_ping_set                  : 100
    + master-r_pgsql:1                  : 100
    + osspc30-eth0                      : up
    + osspc30-eth1                      : up
    + r_pgsql-data-status               : STREAMING|SYNC
    + r_pgsql-status                    : HS:sync

Migration summary:
* Node osspc30:
* Node osspc31:
```

osspc31 の pgsql-status は、単独状態を表す「HS:alone」、非同期レプリケーション状態を表す「HS:async」を経由して、同期レプリケーション状態を表す「HS:sync」となります。

4.4. フェイルオーバ確認

Active

各ノードの PowerGres に問題が生じた場合の動作を確認します。まず、稼働系の PowerGres に障害が発生したと仮定し、kill コマンドで postgres プロセスを強制終了します。

```
[osspc30]# crm_mon -Afr -i 1
(省略)
Full list of resources:

 Resource Group: g_service
    r_service_vip       (ocf::heartbeat:IPaddr2):       Started osspc31
    r_replication_vip   (ocf::heartbeat:IPaddr2):       Started osspc31
 Master/Slave Set: ms_pgsql
    Masters: [ osspc31 ]
    Stopped: [ r_pgsql:0 ]
 Clone Set: c_pingd
    Started: [ osspc30 osspc31 ]

Node Attributes:
* Node osspc30:
    + default_ping_set                  : 100
    + master-r_pgsql:0                  : -INFINITY
    + osspc31-eth0                      : up
    + osspc31-eth1                      : up
    + r_pgsql-data-status               : DISCONNECT
    + r_pgsql-status                    : STOP
* Node osspc31:
    + default_ping_set                  : 100
    + master-r_pgsql:1                  : 1000
    + osspc30-eth0                      : up
    + osspc30-eth1                      : up
    + r_pgsql-data-status               : LATEST
    + r_pgsql-master-baseline           : 000000002E000198
    + r_pgsql-status                    : PRI
```

```
 Migration summary:
 * Node osspc30:
   r_pgsql:0: migration-threshold=1 fail-count=1
 * Node osspc31:

 Failed actions:
    r_pgsql:0_monitor_7000 (node=osspc30, call=14, rc=7,
status=complete): not running
```

pgsql リソースのモニター監視に失敗し pgsql リソースに fail-count が「＋1」されます。これを受けて 、osspc30 の仮想 IP リソースが停止します。一定時間後、待機系である osspc31 の pgsql が promote され 、pgsql-status が「PRI」となり、仮想 IP が osspc31 にて「started」となることが確認できます。

HA クラスタの状態を元に戻すには、以下の手順で行います。

1. ロックファイルのクリア

heartbeat のリソースエージェント(RA)が管理するロックファイルがあります。フェイルオーバが行われると、/var/lib/pgsql/tmp/PGSQL.lock に作成されますので、削除します。

2. PGDATA データの再同期

PowerGres （PostgreSQL も）9.1 では、昇格したノードの待機系として、旧稼働系のデータベースクラスタを利用することができないため、稼働系ノードから、待機系ノードへデータベースクラスタの物理コピーを取得します。

```
[osspc30]$ rm -fr /var/lib/pgsql/data/*
[osspc30]$ rm -fr /var/lib/pgsql/enc_space/*
[osspc30]$ pg_basebackup -h 192.168.11.31 -D ./data -U repli -P
```

-h には稼働系ノードのレプリケーション用の IP アドレスまたは仮想 IP アドレスを指定します。

3. リソースのエラー状態のクリア

過去のリソース異常は、記録されたまま残っているため、エラー情報をクリーンアップします。

```
[osspc30]# crm resource cleanup ms_pgsql
```

5. 本構成の懸念点

透過的暗号化と HA クラスタを組み合わせる場合、「キーストアの自動オープンを有効にする」という設定を適用する必要があります。自動オープンせずに、起動時に手動でパスフレーズを入れる方がセキュリティとしては頑健です。しかしながら、障害時に自動的に切り替わることを実現するためには、自動オープンとせざるを得ません。この点は、ＨＡクラスタを構成する以上は、避けられないトレードオフとなります。

6. まとめ

本ドキュメントでは PowerGres Plus V9.1 の透過的暗号化機能の紹介、ストリーミングレプリケーションの紹介、Heartbeat / Pacemaker による HA クラスタ構築の紹介をして、これらを組み合わせたクラスタ構成について動作検証内容を報告しました。

検証の結果、PowerGres Plus V9.1 の透過的暗号化機能と ストリーミングレプリケーションによるデータ同期、Pacemaker によるＨＡクラスタ構成の組み合わせは可能であるということがいえます。

7. 免責事項

本ドキュメントは SRA OSS, Inc. 日本支社が作成したものですが、SRA OSS, Inc. 日本支社は、本ドキュメントに対して、正確性、有用性、その他を保証するものではありません。本ドキュメントの内容を利用する場合は、利用者の責任において実施いただくものとなります。