



EDB Postgres High Availability and Horizontal Read Scaling Architecture

リリース **1.1**

2020 年 01 月 29 日

目次

第 1 章	アーキテクチャの概要	1
1.1	フェールオーバーマネージャーの概要	2
1.2	PgPool-II の概要	3
1.2.1	PCP の概要	3
1.2.2	Pgpool Watchdog	3
第 2 章	Architecture	5
第 3 章	PgPool による高可用性の実装	10
3.1	フェールオーバーマネージャーの構成	11
3.2	Pgpool の構成	12
3.3	pgpool_backend.sh	13
第 4 章	オプションのコンポーネント	14
4.1	仮想 IP アドレス	14
4.2	Pgpool Watchdog	14
第 5 章	付録	16
5.1	付録 A: サポートされるフェールオーバーシナリオ	16
5.2	付録 B: 統合スクリプト	18
5.2.1	load_balancer_detach.sh	18
5.2.2	load_balancer_attach.sh	19
5.2.3	follow_master.sh	20
5.2.4	pgpool_backend.sh	21

第 6 章	結論	27
索引		29

第 1

アーキテクチャの概要

高可用性と読み取りスケーラビリティは EDB Postgres Advanced Server のコア機能セットの一部ではないため、Advanced Server は外部ツールに依存してこの機能を提供します。このドキュメントでは、EDB Failover Manager および Pgpool-II が提供する機能に焦点を当て、これらのツールを中心に形成された高可用性アーキテクチャの影響について説明します。Advanced Server に提供される利点を活用するために、Failover Manager と Pgpool を最適に構成する方法を示します。アーキテクチャで説明されているリファレンスアーキテクチャを使用する セクションでは、自動フェールオーバーメカニズム（Failover Manager を使用）を実装することで高可用性を実現する方法を学習できます。同時に、大規模なワークロードと読み取り集中型または混合ワークロードを備えた多数の同時クライアントにシステムをスケーリングして、水平スケーリング/読み取りを実現します。スケーラビリティ（Pgpool を使用）。

このドキュメントで説明されているアーキテクチャは、EFM 3.8、EDB pgPool 4.0、および Advanced Server 12 向けに開発およびテストされています。

Advanced Server および Failover Manager のドキュメントは、次の EnterpriseDB から入手できます。

<https://www.enterprisedb.com/resources/product-documentation>

pgPool-II のドキュメントは次の場所にあります。

<http://www.pgpool.net/docs/latest/en/html>

1.1 フェールオーバーマネージャーの概要

フェールオーバーマネージャーは、Postgres ストリーミングレプリケーションクラスターの状態を監視し、障害を迅速に検証する高可用性モジュールです。データベースの障害が発生すると、フェールオーバーマネージャーはストリーミングレプリケーションスタンバイノードを書き込み可能なマスターノードに自動的に昇格させ、継続的なパフォーマンスを確保し、最小限のサービス中断でデータ損失を防ぎます。

基本的な EFM アーキテクチャの用語

フェールオーバーマネージャークラスターは、ネットワーク上の次のホスト:

- ****マスターノード****は、データベースクライアントにサービスを提供するプライマリデータベースサーバーです。
- 1 つ以上の****スタンバイノード****は、マスターノードに関連付けられたストリーミングレプリケーションサーバーです。
- ****監視ノード****は、フェールオーバーシナリオでマスターまたはスタンバイのいずれかのアサーションを確認します。クラスターに 3 つ以上のノードが含まれる場合、クラスターには専用の監視ノードは必要ありません。データベースホストである 3 番目のクラスターメンバーがない場合は、専用の監視ノードを使用できます。クラスターには複数の監視ノードが含まれる場合があります。

1.2 PgPool-II の概要

Pgpool-II (Pgpool) は、EPAS およびコミュニティ Postgres クラスターの複数のスタンバイで SELECT クエリの水平スケーラビリティのための接続プーリングと負荷分散を提供するオープンソースアプリケーションです。backend_weight パラメーターを使用して、読み取りトラフィックがマスターノードに送信されるのを防ぐように Pgpool を設定できます。このような場合、データ変更言語 (DML) クエリ (つまり、INSERT、UPDATE、DELETE) は常にマスターノードに送信されますが、読み取りクエリはスタンバイ、混合および読み取り集中型のワークロードでスケーラビリティを提供します。

EnterpriseDB は、次の Pgpool 機能をサポートしています。

- 負荷分散
- 接続プーリング
- 高可用性
- 接続制限

1.2.1 PCP の概要

Pgpool は、Pgpool のステータスの取得や Pgpool プロセスのリモート終了などの管理操作を実行する管理者向けの PCP と呼ばれるインターフェイスを提供します。PCP コマンドは、ネットワーク経由で Pgpool を操作する UNIX コマンドです。

1.2.2 Pgpool Watchdog

watchdog は、高可用性機能を提供する Pgpool のオプションのサブプロセスです。watchdog によって追加される機能は次のとおりです。

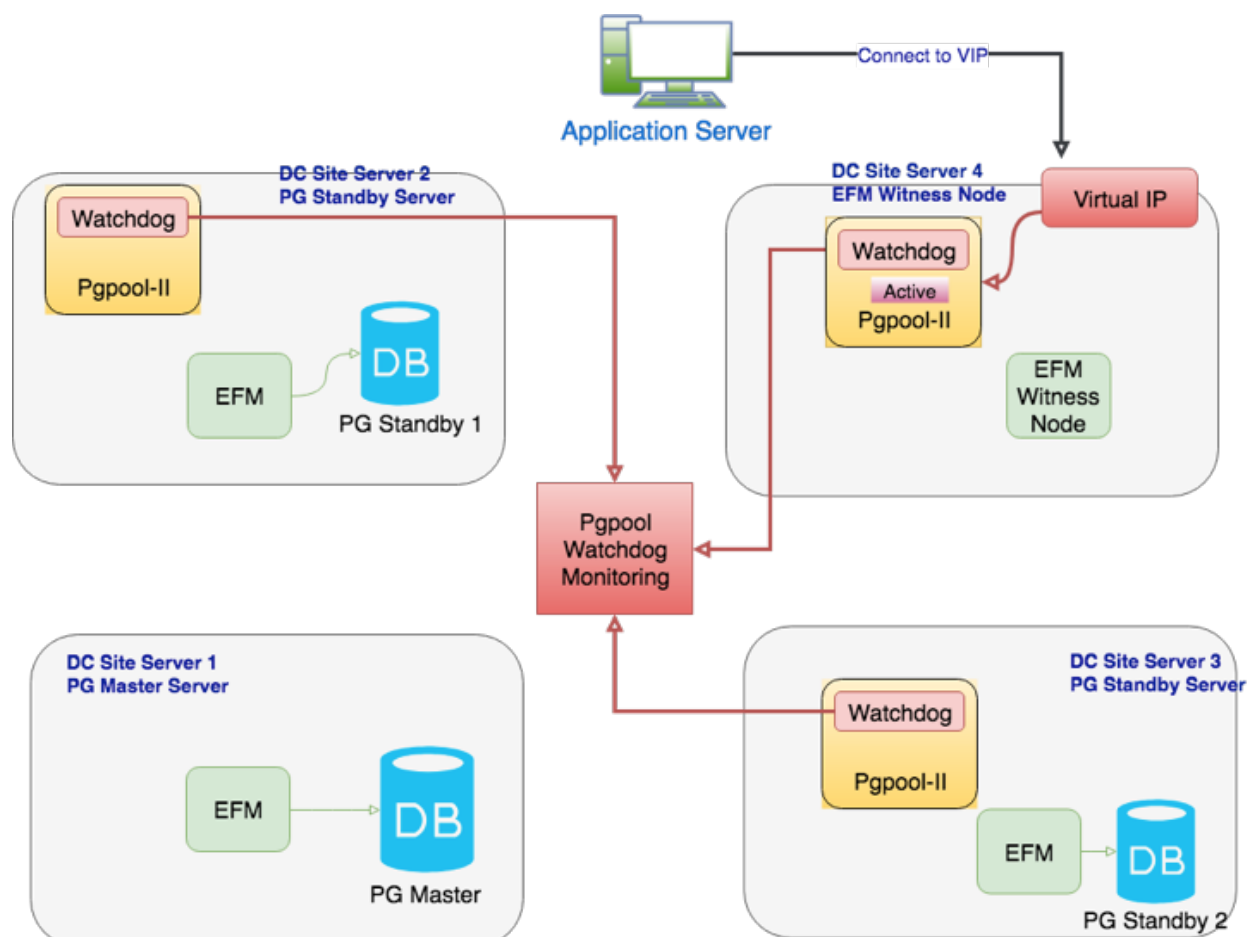
- pgpool サービスのヘルスチェック
- 他のウォッチドッグプロセスの相互監視
- 特定の障害が検出された場合のアクティブ/スタンバイ状態の変更
- サーバーの切り替えに同期して割り当てる自動仮想 IP アドレス
- 回復中のサーバーのスタンバイとしての自動登録

Pgpool watchdog コンポーネントの詳細については、次を参照してください。

<http://www.pgpool.net/docs/latest/en/html/tutorial-watchdog.html>

第 2

Architecture



サンプルのアーキテクチャー図には、次の表で説明する 4 つのノードが示されています。

シナリオ	コンポーネント
サーバー 1	Advanced Server と Failover Manager を Postgres Streaming Replication で実行しているマスターノード。アプリケーション/クライアントは、サーバー 4 の Pgpool ポート（または、使用する場合は仮想 IP）を介して接続し、書き込み操作（つまり、INSERT、UPDATE、DELETE）を実行します。
サーバー 2 およびサーバー 3	フェールオーバーマネージャーを実行するスタンバイノード（Pgpool-II はオプション）。これは、ストリーミングレプリケーションのスタンバイノードです。アプリケーション/クライアントは、サーバー 4 の Pgpool ポート（または、使用されている場合は仮想 IP）を介してこのデータベースに接続し、読み取り操作（SELECT など）を実行します。必要に応じて、ウォッチドッグを実行して、オプションのスタンバイ Pgpool インスタンスをここでセットアップできます。
サーバー 4	Pgpool-II および Failover Manager を実行するオプションの監視ノード。このサーバーは、アクティブなデータベースではなく、フェールオーバーマネージャーと Pgpool でセットアップされます。すべてのアプリケーション/クライアントは、このサーバーの IP アドレスまたはこのサーバーを指す仮想 IP を介して、このサーバーを介して他のデータベースに接続します。少なくとも 3 つの他の EFM ノードが存在する場合、監視ノードは必要ないことに注意してください。このサンプルアーキテクチャの監視ノードは、デモ用に提供されています。

このアーキテクチャ：

- マスターノードに障害が発生した場合に 2 つのスタンバイを提供することにより、最大の可用性を実現します。
- 負荷分散のための複数のスタンバイで読み取りスケーラビリティを向上させることにより、混合および読み取り集中型のワークロードで最大のパフォーマンスを実現します。
- マスターで Pgpool を実行せずに負荷分散を実行することにより、マスターノードの負荷を軽減します。
- watchdog を使用して Pgpool を高可用性モードで構成することにより、Pgpool の単一障害点を回避します。
- 最も負荷の少ないノード（監視ノード）で Pgpool マスター/アクティブインスタンスを実行し、フェールオーバーマネージャーとリソースを共有しながらパフォーマンスを向上させます（TCO を削減します）。

1 つまたは複数のスタンバイが同期レプリケーションで構成されている場合、ユーザーは障害イベントでほぼゼロのデータ損失を達成できます。

このアーキテクチャでは、次の動作が期待できます。

シナリオ	HA への影響	読み取りスケーラビリティへの影響
スイッチオーバー/スイッチバック: これは、一部の OS / DB レベルのアクティビティ、およびダウンタイム中にマスターとして利用可能なスタンバイのいずれかを昇格するときにかかる計画的なダウンタイムです。	**HA への影響なし**(ロール変更中の数秒の妨害を除く)。EFM / PgPool クラスター内のノードの数はそのままです。スイッチオーバーは、EFM によって (EFM コマンドを介して) 行われます。使用可能なスタンバイの 1 つが、ダウンタイム中に新しいマスターとして昇格されます。古いマスターは、手動セットアップなしで EFM によって新しいスタンバイとして再構成され、HA セットアップのノードの総数を維持します。	**読み取りスケーラビリティへの影響なし** (ロール変更中の数秒の妨害を除く)。スイッチオーバー後、クラスター内のスタンバイの総数は同じままであるため、負荷分散/読み取りスケーラビリティへの影響はありません。EFM による切り替えが完了すると、変更後の PgPool を更新するためにポストプロモーションスクリプトが呼び出されます。したがって、PgPool はすべてのクラスターノードの役割を変更します。
フェイルオーバー: これは、発生する可能性のある計画外のダウンタイムであり、アプリケーションがマスターデータベースにアクセスできなくなります (プライマリ DB がダウンします)。	HA への影響はありません。ただし、このようなインシデント (フェイルオーバー) は、EFM / PgPool クラスターに 1 つのスタンバイのみを残します。常に最大可用性 (1 マスター、2 スタンバイ) を維持するには、古い/ダウンしたマスターを新しいスタンバイとして (pg_basebackup または pg_rewind を使用して) 再構築し、EFM クラスターに接続する必要があります (または新しいマシンを次のように導入する必要があります) スタンバイ)。DBA による手動の介入が必要です。フェイルオーバーは EFM	読み取りのスケーラビリティが影響を受けます。フェールオーバー後、古い/ダウンしたマスターが新しいスタンバイとして再構築され、EFM クラスターに接続されるまで、読み取りスケーラビリティ/ロードバランシングに使用できるスタンバイは 1 つだけです。ノードの総数 (この場合は 3 つのノード) が復元されると、EFM 接続スクリプトはノードを Pgpool クラスターに接続します。完了後、両方のスタンバイがロードバランシングに使用可能になります。
	によって自動的に実行されます。	

PgPool による高可用性の実装

フェールオーバーマネージャーは、Postgres ノードの状態を監視します。マスターノードに障害が発生した場合、フェールオーバーマネージャーはスタンバイノードへの自動フェールオーバーを実行します。Pgpool はバックエンドノードの状態を監視せず、スタンバイノードへのフェールオーバーを実行しないことに注意してください。

バージョン 3.2 以降、フェールオーバーマネージャーエージェントは、Pgpool の PCP インターフェイスを使用して、スタンバイノードのフェールオーバーの実行後に障害が発生したノードを Pgpool の負荷分散から切り離すように構成できます。必要な構成ファイルの変更と関連するスクリプトの詳細については、以降のセクションで説明します。

3.1 フェールオーバーマネージャーの構成

Failover Manager は、障害が発生したデータベースノードを Pgpool ロードバランシングから削除する機能を提供します。フェールオーバーマネージャーは、フェールオーバーマネージャークラスターに戻ったときにノードを Pgpool に再接続することもできます。この動作を設定するには、次のパラメーターの `efm.properties` ファイルでロードバランサーの `* attach` および `detach *` スクリプトを識別する必要があります。

- `script.load.balancer.attach=/path/to/load_balancer_attach.sh %h`
- `script.load.balancer.detach=/path/to/load_balancer_detach.sh %h`

`load.balancer.detach` で参照されるスクリプトは、フェールオーバーマネージャーがデータベースノードに障害が発生したと判断したときに呼び出されます。スクリプトは、PCP インターフェース呼び出しを発行することにより、Pgpool からノードを切り離します。Pgpool ポートに接続された `psql` セッションで `SHOW NODES` を呼び出すことで、`load.balancer.detach` スクリプトが正常に実行されたことを確認できます。`SHOW NODES` を呼び出すと、ノードが `down` としてマークされていることが返されます。Pgpool は、ダウンしたノードにクエリを送信しません。

`load.balancer.attach` で参照されるスクリプトは、`resume` コマンドが `efm` コマンドラインインターフェイスに発行され、ダウンしたノードをフェールオーバーマネージャークラスターに追加し直すときに呼び出されます。ノードがクラスターに再参加すると、`load.balancer.attach` で参照されるスクリプトが呼び出され、PCP インターフェイスコールが発行され、ノードが Pgpool クラスターに追加されます。Pgpool ポートに接続された `psql` セッションで `SHOW NODES` を呼び出すことで、`load.balancer.attach` スクリプトが正常に実行されたことを確認できます。コマンドは、ノードが `up` としてマークされていることを返す必要があります。この時点で、Pgpool はこのノードの負荷分散候補としての使用を再開します。これらの各パラメーターのサンプルスクリプトは、付録 B に記載されています。

3.2 Pgpool の構成

Pgpool ホスト上の `pgpool.conf` ファイルにある以下の設定パラメーターの値を提供する必要があります:

```
follow_master_command = '/path/to/follow_master.sh %d %P'
load_balance_mode = on
master_slave_mode = on
master_slave_sub_mode = 'stream'
fail_over_on_backend_error = off
health_check_period = 0
failover_if_affected_tuples_mismatch = off
failover_command = ''
failback_command = ''
search_primary_node_timeout = 3
```

フェールオーバーマネージャー以外のセットアップでプライマリ/マスターノードが（フェールオーバーまたは手動昇格によって）Pgpool で変更されると、Pgpool はすべてのスタンバイノードを自身から切り離し、各スタンバイノードに対して `follow_master_command` を実行します。新しいマスターノードに従います。フェールオーバーマネージャーは、ポストプロモーションスクリプトを実行する前にスタンバイノードを再設定するため（フェールオーバーマネージャーの設定に一致するようにスタンバイが Pgpool でプライマリに昇格される）、`follow_master_command` はスタンバイノードを Pgpool に再接続するだけです。

スタンバイノードに読み取りトラフィックを分散することで読み取りのスケラビリティを確保するために、負荷分散がオンになっていることに注意してください。

また、フェールオーバーマネージャーがヘルスチェックの実行とフェールオーバーのトリガーを担当するため、ヘルスチェックとエラートリガーバックエンドフェールオーバーがオフになっていることにも注意してください。この場合、Pgpool がヘルスチェックを実行することはお勧めできません。これにより、Failover Manager との競合が発生したり、早期にフェールオーバーが実行されたりすることはありません。

最後に、`search_primary_node_timeout` は低い値に設定されており、Failover Manager がトリガーするフェールオーバー時に Pgpool サービスの迅速な回復を保証します。

3.3 pgpool_backend.sh

アタッチおよびデタッチスクリプトを正常に呼び出すには、pgpool_backend.sh スクリプトを指定する必要があります。pgpool_backend.sh は、Pgpool で実際の PCP インターフェイスコマンドを発行するためのヘルパースクリプトです。Failover Manager のノードは IP アドレスで識別されますが、PCP コマンドはノード ID を参照します。pgpool_backend.sh は、IP アドレスからノード ID へのマッピングを透過的に実行するための抽象化レイヤーを提供します。

オプションのコンポーネント

4.1 仮想 IP アドレス

Pgpool-II と Failover Manager はどちらも、シームレスなフェイルオーバーのために仮想 IP を使用する機能を提供します。両方ともこの機能を提供しますが、Pgpool が仮想 IP を現在アクティブな Pgpool ホストに（マルチ Pgpool ウォッチドッグセットアップで）関連付ける間、Failover Manager は仮想 IP をマスターデータベースノードに関連付けることに注意する必要があります。

Pgpool のアクティブなインスタンス（サンプルアーキテクチャではサーバー 4）がダウンした場合、利用可能なスタンバイ Pgpool インスタンス（ウォッチドッグの優先度による）がアクティブな Pgpool インスタンスとして処理されることに注意してください。

4.2 Pgpool Watchdog

Watchdog は、Pgpool ノードの高可用性を提供します。このセクションでは、各 Pgpool ノードでウォッチドッグを構成するために必要な構成パラメーターをリストします。

すべての Pgpool ノードに共通の Watchdog 設定パラメーター

```
use_watchdog = on # enable watchdog
wd_port = 9000 # watchdog port, can be changed
delegate_IP = 'Virtual IP address'
wd_lifecycle_method = 'heartbeat'
wd_interval = 10 # we can lower this value for quick detection
wd_life_point = 3
# virtual IP control
if_cmdconfig_path = '/sbin' # ifconfig command path
if_up_cmd = 'ifconfig eth0:0 inet $_IP_$ netmask 255.255.255.0'
# startup delegate IP command
if_down_cmd = 'ifconfig eth0:0 down' # shutdown DIP
arping_path = '/usr/sbin' # arping command path
arping_cmd = 'arping -U $_IP_$ -w 1' # arping command
```

各 Pgpool ノードのカスタムウォッチドッグ設定パラメーター

各 Pgpool ノードで次の構成パラメータを設定する必要があります。間隔と再試行の値は、要件とテスト結果に応じて調整できます。

```
other_pgpool_hostname0 = '<server# IP/hostname>'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
other_pgpool_hostname1 = '<server# IP/hostname>'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
wd_priority = <any integer>
```

マスターウォッチドッグノードを選択するための選択で、wd_priority を使用してローカルウォッチドッグノードの優先度を上げることができます。クラスターの起動時または古いマスターウォッチドッグノードに障害が発生した場合にクラスターが新しいマスターノードを選択すると、wd_priority 値の高いノードがマスターウォッチドッグノードとして選択されます。

5.1 付録 A: サポートされるフェールオーバーシナリオ

サポートされているフェールオーバーシナリオの概要を以下に示します。リストは包括的なものではないことに注意してください。Failover Manager が各フェールオーバー/障害シナリオを処理する方法の詳細については、Failover Manager のドキュメントを参照してください。

シナリオ	フェールオーバーマネージャー/ pgPool 応答
マスターデータベースがダウンしています	ほとんどの場合、フェールオーバーマネージャーはフェールオーバーを実行し、使用可能なスタンバイの 1 つをマスターノードに昇格させます。仮想 IP アドレス（構成されている場合）が再割り当てされます。
マスターエージェントがクラッシュするか、マスターノードが失敗する	時期尚早のフェールオーバー/プロモーションを防ぐために、フェールオーバーマネージャーは最初にマスターデータベースがまだ稼働中であるかどうかを確認します（つまり、マシン全体ではなく、マスターサーバーの EFM エージェントのみがダウンしています）。必要に応じて、EFM は、利用可能なスタンバイの 1 つをマスターノードに昇格させることにより、その後フェールオーバーを実行します。仮想 IP アドレス（構成されている場合）が再割り当てされます。
スタンバイエージェントが終了するか、スタンバイノードに障害が発生する	フェールオーバーマネージャーは管理者に通知し、 <code>load_balancer_detach.sh</code> スクリプトを呼び出します（ <code>efm.properties</code> で適切に構成されている場合）。詳細については、 load_balancer_detach.sh を参照してください 付録 B の
専用の監視エージェントが終了するか、ノードに障害が発生する	EFM:管理者に Pgpool が通知されます:Pgpool はフェールオーバーを実行し、既存のスタンバイ PgPool インスタンスがアクティブインスタンスとして昇格されます。仮想 IP（構成されている場合）は、新しいアクティブなインスタンスに再割り当てされます。
スタンバイがクラスターに追加されます	スタンバイノードが復旧すると、 <code>efm resume <clustername></code> コマンドを使用してフェールオーバーマネージャークラスターに追加されます。 <code>load_balancer_attach.sh</code> スクリプトは（適切に設定されている場合）その後呼び出され、PCP インターフェイスを介して Pgpool クラスターを更新します。詳細については、 load_balancer_attach.sh を参照してください 付録 B の

5.2 付録 B:統合スクリプト

5.2.1 load_balancer_detach.sh

```
#!/bin/bash
#%h host name
output_file=/tmp/efm-scripts/pp_log
pool_backend=/tmp/efm-scripts/pgpool/pgpool_backend.sh
node_address=$1
current_date_time=`date +%Y-%m-%d %H:%M:%S`;
echo $current_date_time >>$output_file
echo "node address to detach = $node_address". >>$output_file
$pool_backend detach $node_address >>$output_file
echo "-----".>>$output_file
exit 0
```

5.2.2 load_balancer_attach.sh

```
#!/bin/bash
#%h host name
output_file=/tmp/efm-scripts/pp_log
pool_backend=/tmp/efm-scripts/pgpool/pgpool_backend.sh
node_address=$1
current_date_time=`date +"%Y-%m-%d %H:%M:%S"`;
echo $current_date_time >>$output_file
echo "node address to attach = $node_address". >>$output_file
$pool_backend attach $1 >>$output_file
echo "-----".>>$output_file
exit 0
```

5.2.3 follow_master.sh

```
#!/bin/sh
PCP_USER= # PCP user name
PCP_PORT= # PCP port number as in pgpool.conf
PCP_HOST= # hostname of Pgpool-II
PGPOOL_PATH= # Pgpool-II installed path
export PCPPASSFILE= # Path to PCPPASS file
# Execute command by failover.
# special values: %d = node id
# %h = host name
# %p = port number
# %D = database cluster path
# %m = new master node id
# %M = old master node id
# %H = new master node host name
# %P = old primary node id
# %R = new master database cluster path
# %r = new master port number
# %% = '%' character
detached_node_id=$1
old_master_id=$2
echo detached_node_id $1
echo old_master_id $2
## If $detached_node_id is equal to $old_master_id,
## then do nothing, since the old master is no longer
## supposed to be part of the cluster.
if [ $detached_node_id -ne $old_master_id ]; then
sleep 10
$PGPOOL_PATH/pcp_attach_node -w -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT $detached_node_id
fi
```

5.2.4 pgpool_backend.sh

```
#!/bin/bash
#
# pgpool-II backend node configuration driver.
#
# usage: promote_standby.sh hostname [port]
#
# set the following variables according to your setup
PCP_USER= # PCP user name
PCP_PORT= # PCP port number as in pgpool.conf
PCP_HOST= # hostname of Pgpool-II
PGPOOL_PATH= # Pgpool-II installed path
export PCPPASSFILE= # Path to PCPPASS file
# function returns the Pgpool-II backend node-id for the given
hostname
# and port number, And if the node-id is not found 255 is returned
# Arguments:
# 1- Hostname
# 2- Port (optional) if not provided, node-id of first matching
# hostname will be returned
#
function get_pgpool_nodeid_from_host {
if [ -z "$1" ]; then
echo "hostname not provided"
return 255
fi
#Now get the total number of nodes configured in Pgpool-II
node_count=`$PGPOOL_PATH/pcp_node_count -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT -w\`
echo searching node-id for $1:$2 from $node_count configured backends
i=0
while [ $i -lt $node_count ];
do
nodeinfo=`$PGPOOL_PATH/pcp_node_info -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT -w $i\`
hostname=`echo $nodeinfo \| awk -v N=1 '{print $N}'\`
port=`echo $nodeinfo \| awk -v N=2 '{print $N}'\`
#if port number is <= 0 we just need to compare hostname
if [ "$hostname" == $1 ] && ( [ -z "$2" ] \| \| [ $port -eq $2 ] );
then
echo "$1:$2 has backend node-id = $i in Pgpool-II"
return $i
```

(次のページに続く)

(前のページからの続き)

```

fi
let i=i+1
done
return 255
}
# function returns 1 if Pgpool-II backend node for the given hostname
# and port number is the primary node in Pgpool-II
# returns 0 for the standby node and 255 if no node exist for the
hostname
# Arguments:
# 1- Hostname
# 2- Port (optional) if not provided, node-id of first matching
# hostname will be returned
#
function is_host_is_primary_pgpool_node {
if [ -z "$1" ]; then
echo "hostname not provided"
return 255
fi
#Now get the total number of nodes configured in Pgpool-II
node_count=`$PGPOOL_PATH/pcp_node_count -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT -w\`
echo searching node-id for $1:$2 from $node_count configured backends
i=0
while [ $i -lt $node_count ];
do
nodeinfo=`$PGPOOL_PATH/pcp_node_info -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT -w $i\`
hostname=`echo $nodeinfo \| awk -v N=1 '{print $N}'\`
port=`echo $nodeinfo \| awk -v N=2 '{print $N}'\`
role=`echo $nodeinfo \| awk -v N=6 '{print $N}'\`
#if port numbner is <= 0 we just need to compare hostname
if [ "$hostname" == $1 ] && ( [ -z "$2" ] \|\| [ $port -eq $2 ] );
then
echo "$1:$2 has backend node-id = $i in Pgpool-II"
# check if the node role is primary
if [ "$role" == "primary" ]; then
return 1
else
return 0
fi
fi
let i=i+1

```

(次のページに続く)

(前のページからの続き)

```
done
return 255
}
# Function promotes the node-id to the new master node
# Arguments:
# 1- node-id: Pgpool-II backend node-id of node to be promoted to
master
function promote_node_id_to_master {
if [ -z "$1" ]; then
echo "node-id not provided"
return 255
fi
$PGPOOL_PATH/pcp_promote_node -w -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT $1
return $?
}
# Function attach the node-id to the Pgpool-II
# Arguments
# 1- node-id: Pgpool-II backend node-id to be attached
function attach_node_id {
if [ -z "$1" ]; then
echo "node-id not provided"
return 255
fi
$PGPOOL_PATH/pcp_attach_node -w -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT $1
return $?
}
# Function detach the node-id from the Pgpool-II
# Arguments
# 1- node-id: Pgpool-II backend node-id to be detached
function detach_node_id {
if [ -z "$1" ]; then
echo "node-id not provided"
return 255
fi
$PGPOOL_PATH/pcp_detach_node -w -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT $1
return $?
}
# function promotes the standby node identified by hostname:port
# to the master node in Pgpool-II
# Arguments:
```

(次のページに続く)

(前のページからの続き)

```
# 1- Hostname
# 2- Port (optional) if not provided, node-id of first matching
# hostname will be promoted
#
function promote_standby_to_master {
get_pgpool_nodeid_from_host $1 $2
node_id=$?
if [ $node_id -eq 255 ]; then
echo unable to find Pgpool-II backend node id for $1:$2
return 255
else
echo promoting node-id: $node_id to master
promote_node_id_to_master $node_id
return $?
fi
}
# function attaches the backend node identified by hostname:port
# to Pgpool-II
# Arguments:
# 1- Hostname
# 2- Port (optional) if not provided, node-id of first matching
# hostname will be promoted
#
function attach_node {
get_pgpool_nodeid_from_host $1 $2
node_id=$?
if [ $node_id -eq 255 ]; then
echo unable to find Pgpool-II backend node id for $1:$2
return 255
else
echo attaching node-id: $node_id to Pgpool-II
attach_node_id $node_id
return $?
fi
}
# function detaches the backend node identified by hostname:port
# from Pgpool-II
# Arguments:
# 1- Hostname
# 2- Port (optional) if not provided, node-id of first matching
# hostname will be promoted
#
function detach_node {
```

(次のページに続く)

(前のページからの続き)

```

get_pgpool_nodeid_from_host $1 $2
node_id=$?
if [ $node_id -eq 255 ]; then
echo unable to find Pgpool-II backend node id for $1:$2
return 255
else
echo detaching node-id: $node_id from Pgpool-II
detach_node_id $node_id
return $?
fi
}
function print_usage {
echo "usage:"
echo " $(basename $0) operation hostname [port]".
echo " operations:".
echo " check_primary: check if node has a primary role".
echo " promote: promote node".
echo " attach: attach node".
echo " detach: detach node".
}
# script entry point
if [ -z "$1" ] \|\| [ -z "$2" ]; then
echo "ERROR: operation not provided"
print_usage
exit 1
fi
shopt -s nocasematch
case "$1" in
"check_primary" )
is_host_is_primary_pgpool_node $2 $3
;;
"promote" ) echo "promote"
promote_standby_to_master $2 $3
;;
"attach" ) echo "attach"
attach_node $2 $3;;
"detach" ) echo "detach"
detach_node $2 $3;;
"watchdog" ) echo "detach"
$PGPOOL_PATH/pcp_watchdog_info -w -U $PCP_USER -h $PCP_HOST -p
$PCP_PORT -v;;
\*) echo "invalid operation $1".
print_usage;;

```

(次のページに続く)

(前のページからの続き)

```
esac  
exit $?
```

第 6

結論

EDB Postgres 高可用性および水平読み取りスケーリングアーキテクチャガイド

Copyright©2018-2020 EnterpriseDB Corporation。無断複写・転載を禁じます。

EnterpriseDB®Corporation 34 Crosby Drive, Suite 201, Bedford, MA 01730, USA

T +1 781 357 3390

F +1 978 467 1307 E

info@enterprisedb.com

www.enterprisedb.com

- EDB は、存在する場合、EDB 製品のログオン UI の設計、コーディングのベストプラクティスの確立、レビュー、および入力検証の検証を行います。EDB は追加の入力コンポーネントに対して同じアプローチに従いますが、製品の性質上、限定された検証が可能な信頼できるユーザーによって入力および送信されるフリーフォーム SQL、WMI またはその他の文字列を受け入れる必要がある場合があります。このような場合、ユーザーが誤った入力やその他の危険な入力を防ぐことはできません。
- EDB は、将来、認証された信頼できるユーザーからのフリーフォーム SQL、WMI またはその他の潜在的に危険な入力を受け入れる製品に機能を追加する権利を留保しますが、そのような機能

はすべて、可能な限り最小のリスクを提供するように設計およびテストされます可能であれば、スーパーユーザーまたは同等の特権が必要です。

- EDB は、すべての潜在的な脅威を予測できることを保証しません。したがって、すべての潜在的な脆弱性が対処または検討されたことをプロセスが完全に保証することはできません。

索引

Architecture, 6

EFM overview, 2

Failover Manager configuration file, 11
follow_master.sh, 20

integration scripts, 18

load_balancer_attach.sh, 19
load_balancer_detach.sh, 18

Pgpool configuration file, 12
Pgpool overview, 3
Pgpool watchdog, 14
pgpool_backend.sh, 13, 21
PgPool による高可用性の実装, 10

supported failover scenarios, 16

アーキテクチャの概要, 1

オプションのコンポーネント, 14

結論, 27
付録, 16