



EDB Failover Manager

リリース **3.10**

EnterpriseDB

2020 年 06 月 28 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 ウォッチドッグ	3
第 2 章	アーキテクチャ	5
第 3 章	PgPool による高可用性の実装	9
3.1	フェールオーバーマネージャーの構成	10
3.2	Pgpool の構成	11
3.3	pgpool_backend.sh	12
第 4 章	オプションコンポーネント	13
4.1	仮想 IP アドレス	13
4.2	Pgpool ウォッチドッグ	13
第 5 章	付録	15
5.1	付録 A:サポートされるフェールオーバーシナリオ	15
5.2	付録 B:統合スクリプト	17
5.2.1	load_balancer_detach.sh	17
5.2.2	load_balancer_attach.sh	18
5.2.3	follow_master.sh	19
5.2.4	pgpool_backend.sh	20
第 6 章	結論	25
	索引	27

第 1 章

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 EDBPostgresAdvancedServer のコア機能セットの一部ではないため、AdvancedServer はこの機能を提供するために外部ツールに依存しています。このドキュメントでは、EDB-FailoverManager と Pgpool-II によって提供され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。AdvancedServer に提供する利点を活用するために、FailoverManager と Pgpool を最適に構成する方法を示します。:ref: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<architecture>‘セクションでは、自動フェイルオーバーメカニズム（フェイルオーバーマネージャーを使用）を実装することで高可用性を実現する方法を学びながら、システムをより大きなワークロードと多数の同時クライアントで読み取り集中型または混合ワークロードにスケーリングし、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するアーキテクチャは、EFM3.10、EDBpgPool4.0、および AdvancedServer12 用に開発およびテストされています。

AdvancedServer および FailoverManager のドキュメントは、EnterpriseDB の次の場所から入手できます。

<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

FailoverManager は、Postgres ストリーミングレプリケーションクラスタの状態を監視し、障害をすばやく検証する高可用性モジュールです。データベース障害が発生した場合、フェイ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の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て、継続的なパフォーマンスを確保し、サービスの中断を最小限に抑えてデータの損失を防ぎます。

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****が、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、フェイルオーバーシナリオでのマスターまたはスタンバイ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ている場合、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の 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ロードバランシングを提供するオープンソースアプリケーションです。pgpool は、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構成できます。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での終了などの管理操作を実行する PCP と呼ばれるインターフェイスを管理者に提供します。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 ウォッチドッグ

watchdog is an optional sub process of Pgpool that provides a high availability feature. Features added by watchdog include:

- 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
- サーバー切り替えに同期して割り当てられる自動仮想 IP アドレス
- リカバリ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、以下を参照してください:

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

第 2 章

アーキテクチャ

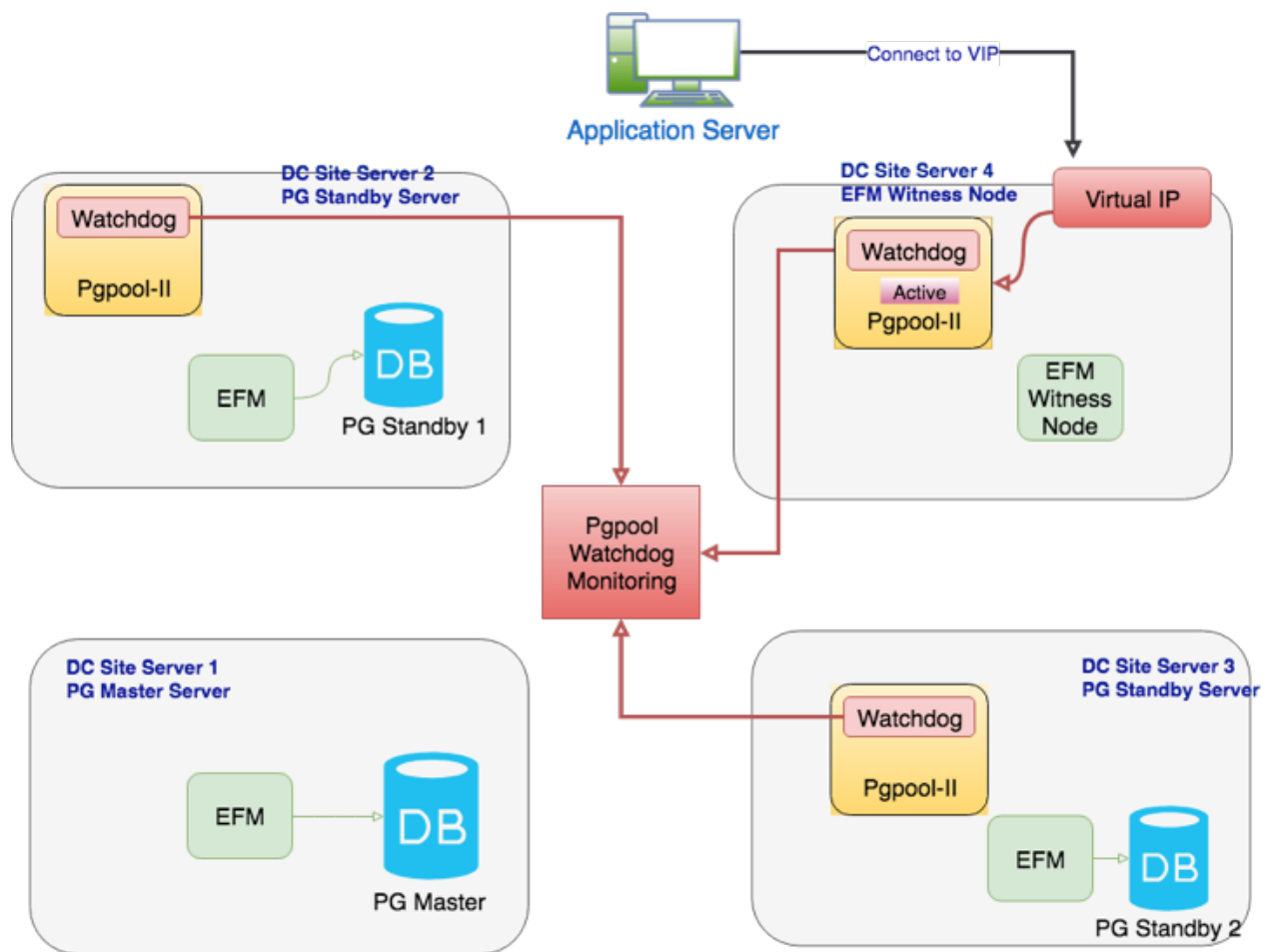


図 1 典型的な EFM および PgPool 構成

サンプルアーキテクチャ図には、次の表に示す 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	AdvancedServer および FailoverManager と PostgresStreamingReplication を実行するマスターノード。アプリケーション/クライアントは、サーバー 4 の Pgpool ポート経由で（または、使用されている場合は仮想 IP 経由で）接続して、書き込み操作（つまり、INSERT、UPDATE、DELETE）を実行します。
サーバー 2 とサーバー 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（つまり、SELECT）を実行します。必要に応じて、オプションのスタンバイ Pgpool インスタンスをここで設定し、watchdog を実行できます。
サーバー 4	Pgpool-II および FailoverManager 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を使用せずに、フェールオーバーマネージャーと Pgpool を使用してセットアップされています。すべてのアプリケーション/クライアントは、このサーバーを介して、このサーバーの IP アドレスを介して、またはこのサーバーを指す仮想 IP を介して、他のデータベースに接続します。他に 3 つ以上の EFM ノードが存在する場合、監視ノードは不要である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。
- ロードバランシング用に複数のスタンバイを使用して読み取りスケーラビリティを向上させることにより、読み取り集中型の混合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 を実行せずにロードバランシング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 を使用して高可用性モードで Pgpool を構成することにより、Pgpool の単一障害点を回避します。
- 最も負担の少ないノード（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て、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（TCO を削減するために）パフォーマンスを向上させます。

1 つ以上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データ損失をほぼゼロにすることができます。

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のために計画されたダウンタイムであり、ダウンタイム中に使用可能なスタンバイをマスターとしてプロモートするときに行われます。	**HA への影響なし** (役割の変更中の数秒間の妨害を除く)。EFM/PgPool クラスタ内のノードの数はそのままです。切り替えは、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の 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はありません** (役割変更中の数秒間の妨害を除く)。スイッチオーバー後も、クラスター内のスタンバイの総数は変わらないため、負荷分散/読み取りのスケーラビリティには影響しません。EFM によってスイッチオーバーが行われると、ポストプロモーションスクリプトが呼び出され、PgPool が変更されて更新されます。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: これは予期しないダウンタイムであり、アプリケーションから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 ダウン)。	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 (フェールオーバー) では、EFM/PgPool クラスタにスタンバイが 1 つだけ残ります。常に最大の可用性 (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 (pg_basebackup または pg_rewind を介して) 新しいスタンバイとして再構築し、EFM クラスタ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 による手動介入が必要です。フェイルオーバーは EFM によって自動的に実行されます。	読み取りスケーラビリティが影響を受けません。フェイルオーバー後、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 クラスタに接続されるまで、読み取りスケーラビリティ/負荷分散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 接続スクリプトがノードを Pgpool クラスタに接続します。完了すると、両方のスタンバイがロードバランシングに使用できるようになります。

第 3 章

PgPool による高可用性の実装

FailoverManager 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、フェイ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イ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イ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、フェイルオーバーマネージャーエージェントは、Pgpool の PCP インターフェースを使用して、スタンバイノードのフェイルオーバーを実行した後、障害が発生したノードを Pgpool 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、以下のセクションで説明します。

3.1 フェイルオーバーマネージャーの構成

FailoverManager は、Pgpool ロードバランシングから障害のあるデータベースノードを削除する機能を提供します。FailoverManager は、FailoverManager 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、以下のパラメーターの `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、フェイ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 は停止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、`efm` コマンドラインインターフェースに `resume` コマンドが発行されて、ダウンしたノードをフェイルオーバーマネージャークラスターに戻すときに呼び出されます。ノードがクラスターに再び参加すると、`load.balancer.attach` で参照されるスクリプトが呼び出され、PCP インターフェー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 はこのノードを負荷分散の候補として使用を再開します。これらの各パラメータのサンプルスクリプトは、付録 B にあります。

3.2 Pgpool の構成

Pgpool ホストの `pgpool.conf` ファイルで以下の設定パラメータ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
```

When the primary/master node is changed in Pgpool (either by failover or by manual promotion) in a non-Failover Manager setup, Pgpool detaches all standby nodes from itself, and executes the `follow_master_command` for each standby node, making them follow the new master node. Since Failover Manager reconfigures the standby nodes *before* executing the post-promotion script (where a standby is promoted to primary in Pgpool to match the Failover Manager configuration), the `follow_master_command` merely needs to reattach standby nodes to Pgpool.

読み込みトラフィックをスタンバイノードに分散することで読み込みスケー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、ヘルスチェックの実行とフェイルオーバーのトリガーは `FailoverManager` が担当するため、ヘルスチェックとエラーによってトリガーされるバックエンドフェイ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して、フェイルオーバーマネージャーとの競合を引き起こしたり、フェイルオーバーを早期に実行したりしないようにすることはお勧めできません。

最後に、`search_primary_node_timeout` は低い値に設定されており、フェイルオーバーマネージャーによってトリガーされるフェイルオーバー時に Pgpool サービスの迅速なリカバリを保証します。

3.3 pgpool_backend.sh

In order for the attach and detach scripts to be successfully called, a `pgpool_backend.sh` script must be provided. `pgpool_backend.sh` is a helper script for issuing the actual PCP interface commands on Pgpool. Nodes in Failover Manager are identified by IP addresses, while PCP commands refer to a node ID. `pgpool_backend.sh` provides a layer of abstraction to perform the IP address to node ID mapping transparently.

第 4 章

オプションコンポーネント

4.1 仮想 IP アドレス

Pgpool-II と FailoverManager はどちらも、シームレスなフェイルオーバーに仮想 IP を使用する機能を提供します。どちらもこの機能を提供しますが、フェイルオーバーマネージャーは仮想 IP をマスターデータベースノードに関連付け、Pgpool は仮想 IP を現在アクティブな Pgpool ホストに関連付けます（マルチ Pgpool ウォッチドッグ設定で）。

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、使用可能なスタンバイ Pgpool インスタンス（watchdog の優先順位に応じて）がアクティブな Pgpool インスタンスとして機能することに注意してください。

4.2 Pgpool ウォッチドッグ

Watchdog は、Pgpool ノードに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 ノードに共通のウォッチドッグ構成パラメーター

```
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
```

(次のページに続く)

(前のページからの続き)

```
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構成パラメーター

以下の構成パラメーターは、各 Pgpool ノードで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>
```

Note that `wd_priority` can be used to elevate the local watchdog node priority in the elections to select master watchdog node. The node with the higher `wd_priority` value will get selected as master watchdog node when cluster will be electing its new master node at the time of cluster startup or in the event of old master watchdog node failure.

第 5 章

付録

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。FailoverManager 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、FailoverManager のドキュメントを参照してください。

シナリオ	フェイルオーバーマネージャー/pgPool レスポンス
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、フェイルオーバーマネージャーはフェイ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、マスターノードで障害が発生する	早期のフェイルオーバー/プロモーションを防ぐために、フェイ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、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ることにより、フェイ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、スタンバイノードで障害が発生する	FailoverManager は管理者に通知し、load_balancer_detach.sh スクリプトを呼び出します (efm.properties で適切に構成されている場合)。詳細については、 :ref:load_balancer_detach.sh を参照してください <load_balancer_detach.sh> 付録 B の
専用監視エージェントが終了するか、ノードが失敗する	EFM:管理者は Pgpool に通知されます:Pgpool はフェイ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、efm resume <clustername> 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。続いて load_balancer_attach.sh スクリプトが呼び出され（適切に構成されている場合）、PCP インターフェースを介して Pgpool クラスターを更新します。詳細については、 :ref:load_balancer_attach.sh を参照してください <load_balancer_attach.sh> 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
fi
let i=i+1
done
return 255
}
```

(次のページに続く)

(前のページからの続き)

```

# 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
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"

```

(次のページに続く)

(前のページからの続き)

```
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:
# 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
}
```

(次のページに続く)

(前のページからの続き)

```

}
# 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 {
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".
}

```

(次のページに続く)

(前のページからの続き)

```
# 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;;
esac
exit $?
```

第 6 章

結論

EDBPostgresHighAvailabilityandHorizontalReadScalingArchitectureGuide

Copyright©2018-2020EnterpriseDBCorporation。全著作権所有。

EnterpriseDB®Corporation34CrosbyDrive、Suite201、Bedford、MA01730、USA

電話番号 +17813573390

F+19784671307E

info@enterprisedb.com

www.enterprisedb.com

- EnterpriseDB および PostgresEnterpriseManager は、EnterpriseDBCorporation の登録商標です。EDB および EDBPostgres は EnterpriseDBCorporation の商標です。Oracle は Oracle、Inc. の登録商標です。その他の商標は、各所有者の商標です。
- EDB は、EDB 製品が存在する場合、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、および検証を行います。EDB は、追加の入力コンポーネントについても同じアプローチに従いますが、製品の性質上、制限付きの検証が可能な信頼できるユーザーが入力および送信するフリーフォーム SQL、WMI、またはその他の文字列を受け入れることが必要になる場合があります。このような場合、ユーザーが誤った入力や危険な入力をするのを防ぐことはできません。
- EDB は、将来的に認証済みの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、そのようなすべての機能が可能な限り最小限のリスクを提供するように設計およびテストされるようにします。可能、スーパーユーザーまたは同等の権限が必要です。
- EDB は、すべての潜在的な脅威を予測できること、または予測できることを保証するものではありません。したがって、私たちのプロセスでは、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完全に保証できません。

索引

EFM overview, 2

Failover Manager configuration file, 10
follow_master.sh, 19

integration scripts, 17

load_balancer_attach.sh, 18
load_balancer_detach.sh, 17

Pgpool configuration file, 11
Pgpool overview, 3
Pgpool watchdog, 13
pgpool_backend.sh, 12, 20
PgPool による高可用性の実装, 9

supported failover scenarios, 15

アーキテクチャ, 5
アーキテクチャの概要, 1

オプションコンポーネント, 13

結論, 25
付録, 15