



# **EDB Postgres High Availability and Horizontal Read Scaling Architecture**

リリース **4.1**

2020 年 12 月 20 日



# 目次

|       |                                                          |    |
|-------|----------------------------------------------------------|----|
| 第 1 章 | Architecture Overview                                    | 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 | PgpoolWatchdog . . . . .                                 | 3  |
| 第 2 章 | Architecture                                             | 5  |
| 第 3 章 | Implementing High Availability with Pgpool               | 9  |
| 3.1   | フェールオーバーマネージャーの構成 . . . . .                              | 9  |
| 3.2   | Pgpool の構成 . . . . .                                     | 10 |
| 3.3   | 仮想 IP アドレス . . . . .                                     | 11 |
| 3.4   | Pgpool-IIWatchdog の構成 . . . . .                          | 12 |
| 第 4 章 | EFM Pgpool Integration Using Azure Network Load Balancer | 15 |
| 第 5 章 | Configuration for Number of Connections and Pooling      | 21 |
| 第 6 章 | Conclusion                                               | 23 |
|       | 索引                                                       | 25 |



## 第 1 章

# Architecture Overview

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外部ツールに依存しています。このドキュメントでは、EDBFailoverManager 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意味について説明します。AdvancedServer に提供され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。「アーキテクチャ」セクション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と、自動フェールオーバーメカニズム（FailoverManager を使用）を実装することにより高可用性を実現する方法を学ぶことができます。（Pgpool を使用して）水平スケーリング/読み取りスケーリングを実現します。

このドキュメントで説明するアーキテクチャは、EFM4.1、EDBpgPool、および AdvancedServer13 向け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 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プライマリ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。

### Basic EFM Architecture Terminology

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にある EFM プロセスで構成されます。

- **Primary** 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**\*\*Standby nodes\*\*** は、プライマリ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。
- **Witness node** は、フェールオーバーシナリオでプライマリ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。障害が発生した状態で、プライマリが半分以上のノードを持つパーティション内で検出されると、プライマリのままになります。そのため、EFM は、偶数のエージェントを含むクラスターでの実行をサポートしています。

## 1.2 Pgpool-II の概要

Pgpool-II (Pgpool) は、EPAS およびコミュニティ Postgres クラスター内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。すべてのバックエンドで、`backend_weight` パラメーターは、バックエンドノードに向けられる読み取りトラフィックの比率を設定できます。プライマリノードの読み取りトラフィックを防ぐために、`backend_weight` パラメーターを 0 に設定できます。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、読み取り中にプライマリノードに送信されます。クエリはスタンバイに対して負荷分散され、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 PgpoolWatchdog

watchdog は Pgpool のオプションのサブプロセスであり、高可用性機能を提供します。watchdog によって追加された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リーダー/スタンバイ状態の変更
- サーバーの切り替えに同期した自動仮想 IP アドレスの割り当て
- 回復中のスタンバイとしてのサーバーの自動登録

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 章

# Architecture

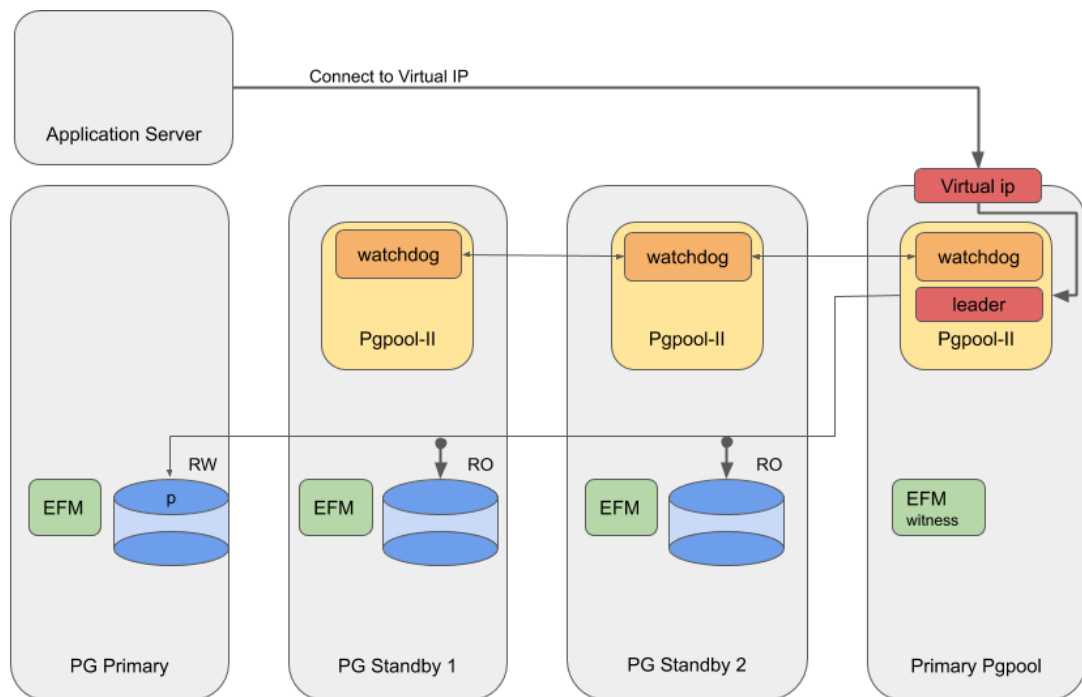


図 1 典型的な EFM および Pgpool 構成

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

| Systems                    | Components                                                                                                                                                                                                                                                                                                                                                                                                         |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| プライマリ Pgpool/EFM 監視<br>ノード | プライマリ Pgpool ノードは、Pgpool と EFMwitness のみを実行します。そのため、Pgpool で利用可能なリソースをできるだけ多く残します。通常の実行モード（Pgpool フェイルオーバーなし）では、プライマリ Pgpool ノードが仮想 IP アドレスを接続し、すべてのアプリケーションが仮想 IP アドレスを介して Pgpool に接続します。Pgpool はすべての書き込みトラフィックをプライマリデータベースノードに転送し、すべてのスタンバイノード間ですべての読み取りのバランスをとります。プライマリ Pgpool ノードでは、EFM 監視プロセスにより、データベースノードの 1 つでも 3 つの EFM エージェントの最小クォータが利用可能になります。失敗します。いくつかの例は、メンテナンスまたは障害のためにノードがすでに使用不可であり、別の障害が発生した場合です。 |
| プライマリデータベースノード             | プライマリデータベースノードは、Postgres（プライマリ）と EFM のみを実行し、すべてのリソースを Postgres に残します。読み取り/書き込みトラフィック（つまり、INSERT、UPDATE、DELETE）は、プライマリ Pgpool ノードによってこのノードに転送されます。                                                                                                                                                                                                                                                                  |
| スタンバイノード                   | スタンバイノードは、Postgres（スタンバイ）、EFM、および非アクティブな Pgpool プロセスを実行しています。プライマリデータベースに障害が発生した場合、EFM はこれらのスタンバイノードのいずれかで Postgres を昇格させ、読み書きトラフィックを処理します。プライマリ Pgpool に障害が発生した場合、Pgpool ウォッチドッグは、VIP を接続するスタンバイノードの 1 つで Pgpool をアクティブにし、データベースノードへのアプリケーション接続の転送を処理します。二重障害状態（プライマリ Pgpool ノードとプライマリデータベースノードの両方に障害が発生している）では、これらのプライマリプロセスの両方が同じノードで終了する可能性があることに注意してください。                                                     |

このアーキテクチャ:

- プライマリ Postgres ノードに障害が発生した場合に昇格できる 2 つのスタンバイを提供することにより、高可用性を実現します。
- ウォッチドッグ構成で少なくとも 3 つの Pgpool プロセスを提供することにより、高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パフォーマンスが向上します。
- 読み取り専用トラフィックをプライマリ pgpool ノードにリダイレクトすることにより、プライマリデータベースノードの負荷を軽減します。
- プライマリデータベースノード上の Pgpool と Postgres 間のリソースの競合を防ぎます。プライマリデータベースノードで Pgpool を実行しないことにより、プライマリ Postgres プロセスはできるだけ多くのリソースを利用できます。

- プライマリ Pgpool ノード上の pgpool と Postgres 間のリソースの競合を防ぎます。プライマリ Pgpool ノードでスタンバイデータベースを実行しないことにより、Pgpool はできるだけ多くのリソースを利用できます。
- オプションで、同期イベントを設定して、障害発生時にほぼゼロのデータ損失を実現できます。

注釈: このアーキテクチャにより、Postgres を実行している 3 つの仮想マシンと Pgpool を実行している 3 つの仮想マシンを完全に分離することもできます。この種類のセットアップには 2 つの追加の仮想マシンが必要ですが、フェールオーバーシナリオで Pgpool と Postgres の間のリソースの競合を防ぎたい場合は、より良い選択です。このセットアップでは、EFMWitnessProcess を実行する追加の 7 番目のノードなしでアーキテクチャを実行できます。障害の解決を向上させるために、efmwitness エージェントを Pgpool サーバーに展開できます。

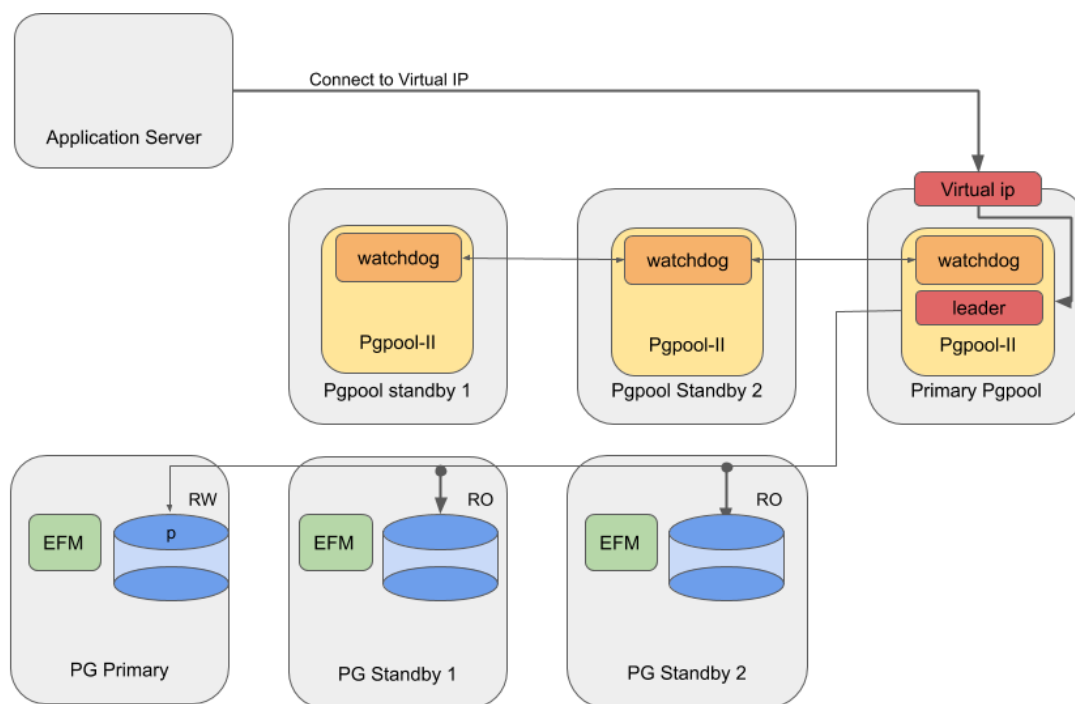


図 2 個別の仮想マシンへの EFM と Pgpool の展開



## 第 3 章

# Implementing High Availability with Pgpool

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。データベース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。

### 3.1 フェールオーバーマネージャーの構成

フェールオーバーマネージャーは、障害が発生したデータベースノードを Pgpool の負荷分散から削除する機能を提供します。また、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。Pgpool を使用して高可用性のために EFM を構成するには、クラスタープロパティファイルで次のプロパティを設定する必要があります。

`pgpool.enable=<true/false>`

`'pcp.user'=<PCP コマンドを呼び出すユーザー>`

`'pcp.host'=<pgpool が使用する仮想 IP。pgpool パラメーター「delegate_IP」と同じ>`

`'pcp.port'=<pgpool が pcp コマンドをリッスンするポート>`

`'pcp.pass.file'=<PCPPASSFILE の絶対パス>`

`'pgpool.bin'=<pgpoolbin ディレクトリの絶対パス>`

## 3.2 Pgpool の構成

このセクションでは、Pgpool-II と EFM を統合するための pgpool.conf ファイル内のいくつかの重要なパラメーターの構成をリストします。

### Backend node setting

3 つの PostgreSQL バックエンドノード、1 つのプライマリノードと 2 つのスタンバイノードがあります。pgpool.conf の backend\_\* 構成パラメーターを使用して構成し、すべてのノードに等しいバックエンドの重みを使用します。これにより、読み取りクエリがすべてのノードに均等に分散されます。

```
backend_hostname0 = 'server1_IP'
backend_port0 = 5444
backend_weight0 = 1
backend_flag0 = 'DISALLOW_TO_FAILOVER'

backend_hostname1 = 'server2_IP'
backend_port1 = 5444
backend_weight1 = 1
backend_flag1 = 'DISALLOW_TO_FAILOVER'

backend_hostname2 = 'server3_IP'
backend_port2 = 5444
backend_weight2 = 1
backend_flag2 = 'DISALLOW_TO_FAILOVER'
```

### Enable Load-balancing and streaming replication mode

pgpool.conf ファイルで次の構成パラメータを設定して、負荷分散とストリーミング複製モードを有効にします

```
master_slave_mode = on
master_slave_sub_mode = 'stream'
load_balance_mode = on
```

### Disable health-checking and failover

ヘルスチェックとフェイルオーバーは EFM で処理する必要があるため、Pgpool-II 側でこれらを無効にする必要があります。pgpool-II 側でヘルスチェックとフェイルオーバーを無効にするには、次の値を割り当てます。

```
health_check_period = 0
fail_over_on_backend_error = off
failover_if_affected_tuples_mismatch = off
failover_command = ""
failback_command = ""
```

pgpool.conf ファイルの値を設定する際には、以下を確認してください。

- `pgpool.conf` の `wd_priority` の値は、各ノードで異なるようにしてください。最高の値を持つノードが最高の優先度を取得します。
- プロパティ `backend_hostname0`、`backend_hostname1`、`backend_hostname2` などは（EFM の用語では）共有プロパティであり、`pgpool.conf` ファイル内のすべてのノードに対して同じ値を保持する必要があります。
- `if_*` の正しいインターフェース値を更新し、`pgpool.conf` ファイルの `cmdprops` を `arping` します。
- 各ノードの `pgpool.conf` ファイルのノード数に応じて、プロパティ `heartbeat_destination0`、`heartbeat_destination1`、`heartbeat_destination2` などを追加します。ここで、`heartbeat_destination0` はローカルノードの `ip/hostname` である必要があります。

### Setting up PCP

スクリプトは PCP インターフェイスを使用するため、パスワードプロンプトなしで PCP 接続を許可するには、PCP および PCPPASS ファイルを設定する必要があります。

PCP のセットアップ:<http://www.pgpool.net/docs/latest/en/html/configuring-pcp-conf.html>

PCPPASS のセットアップ:<https://www.pgpool.net/docs/latest/en/html/pcp-commands.html>

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。

フェールオーバーマネージャーは、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はオフになっています。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、FailoverManager 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、フェールオーバーマネージャーによってトリガーされた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 仮想 IP アドレス

Pgpool-II と FailoverManager はどちらも、シームレスなフェー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-II リーダーは仮想 IP を介してアプリケーション接続を受け取るプロセスです。この設計と同様に、このような仮想 IP 管理は Pgpool-II ウォッチドッグシステムによって実行されます。EFMVIP はこの設計には有益な効果がなく、無効にする必要があります。

Pgpool のアクティブインスタンス（サンプルアーキテクチャのプライマリ Pgpool サーバー）の障害状況では、次に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従って）がアクティブ化され、リーダー Pgpool インスタンスとして機能します。

## 3.4 Pgpool-II Watchdog の構成

Watchdog は、Pgpool-II 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-II ノードでウォッチドッグに必要な構成をリストします。

### Common watchdog configurations on all Pgpool nodes

以下の構成パラメーターは、ウォッチドッグを有効にして構成します。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。

```
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
ifconfig_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 delegate IP command
arping_path = '/usr/sbin' # arping command path
```

---

**注釈:** eth0 の値をシステムのネットワークインターフェイスに置き換えます。接続数の調整、およびプール設定については、[第 5 章](#) を参照してください。

---

### Watchdog configurations on server 2

```
other_pgpool_hostname0 = 'server 3 IP/hostname'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
other_pgpool_hostname1 = 'server 4 IP/hostname'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
wd_priority = 1
```

### Watchdog configurations on server 3

```
other_pgpool_hostname0 = 'server 2 IP/hostname'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
other_pgpool_hostname1 = 'server 4 IP/hostname'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
wd_priority = 3
```

#### Watchdog configurations on server 4

```
other_pgpool_hostname0 = 'server 2 IP/hostname'
other_pgpool_port0 = 9999
other_wd_port0 = 9000
other_pgpool_hostname1 = 'server 3 IP/hostname'
other_pgpool_port1 = 9999
other_wd_port1 = 9000
wd_priority = 5 # use high watchdog priority on server 4
```



## 第 4 章

# EFM Pgpool Integration Using Azure Network Load Balancer

このセクションでは、データベース、EFM、および Pgpool が Azure の CentOS8 仮想マシンにインストールされる、EFMPgpool 統合の特定のユースケースについて説明します。この特定のユースケースでは、AzureLoadBalancer (LNB) を使用して、PgpoolVIP を使用してトラフィックを誘導する代わりに、すべてのアクティブな Pgpool インスタンスにトラフィックを分散しています。

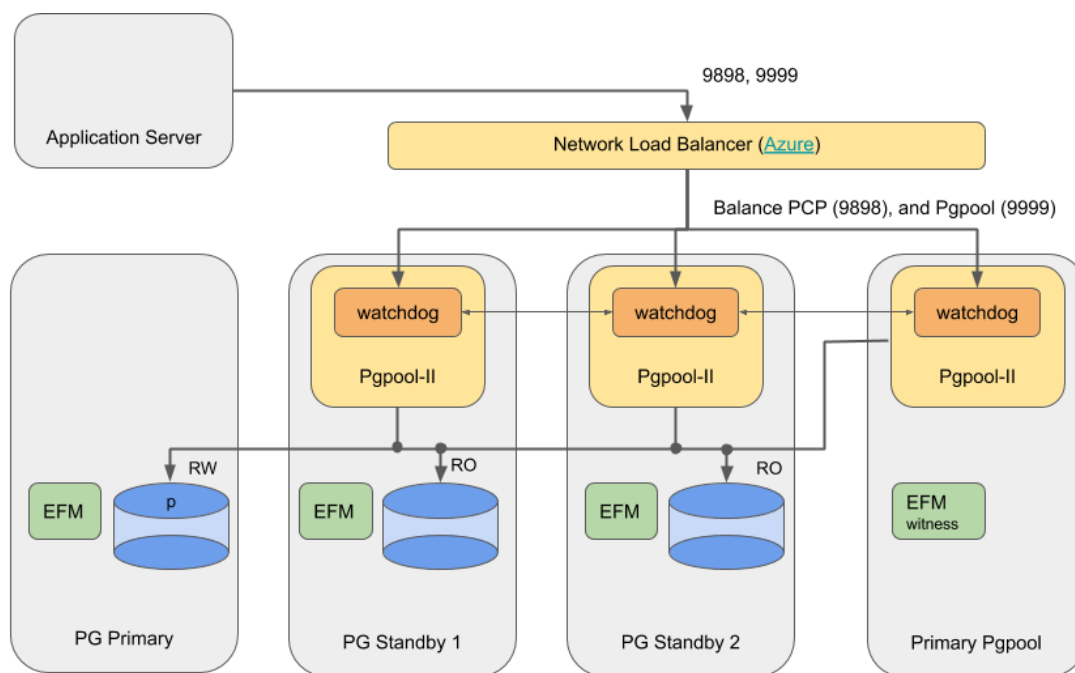


図 1 AzureLoadBalancer を使用した EFM と Pgpool の統合のアーキテクチャ図

### Step 1 (Installation) :

次のように、AzureVirtualMachines に AdvancedServer データベース、EFM、および Pgpool をインストールして

構成します。

| Systems   | Components                                                         |
|-----------|--------------------------------------------------------------------|
| Primary   | AdvancedServer13 および FailoverManager4.1 を実行しているプライマリノード            |
| Standby 1 | AdvancedServer13、FailoverManager4.1、および Pgpool4.1 を実行しているスタンバイノード。 |
| Standby 2 | AdvancedServer13、FailoverManager4.1、および Pgpool4.1 を実行しているスタンバイノード。 |
| Witness   | フェールオーバーマネージャー 4.1 および Pgpool4.1 を実行している監視ノード。                     |

### Step 2 (Pgpool configuration) :

第 3 章に記載されている手順に従って Pgpool を構成します (delegate\_ip は除きます。このアーキテクチャでは空のままにしてください)。

### Step 3 (Azure Load Balancer configuration) :

AzureNLB を使用するには、次の構成を行う必要があります。

**Networking** :NetworkLoadBalancer および各仮想マシンに対して、次の設定を確認する必要があります:パブリック IP とプライベート IP を NLB に割り当て、プライベート IP のみを仮想マシンに割り当てます。アプリケーションサーバーはパブリック IP 経由で NLB に接続し、NLB はプライベート IP 経由で仮想マシンに接続する必要があります。

現在のシナリオでは、各コンポーネントに割り当てられた IP アドレスは次のとおりです。

- NLB のパブリック IP:40.76.240.33 (pcp.host)
- Primarydb のプライベート IP:172.16.1.3 (これはロードバランサーのバックエンドプールの一部ではないことに注意してください)
- Standby 1 のプライベート IP:172.16.1.4
- Standby 2 のプライベート IP:172.16.1.5
- 監視ノードのプライベート IP:172.16.1.6

データベース、EFM、および Pgpool の実行に必要なポートが通信用に開いていることを確認します。以下は、これらの各コンポーネントのデフォルトポートのリストです (環境に合わせてポートをカスタマイズできます)。

- データベース:5444
- EFM:7800 (bind.address)
- Pgpool:9000、9694、9898、9999

**Backend pool** :Pgpool インスタンスを実行する 3 つの仮想マシンすべてで構成されるバックエンドプールを作成します。仮想マシンのプライベート IP を使用して、バックエンドプールを作成します。

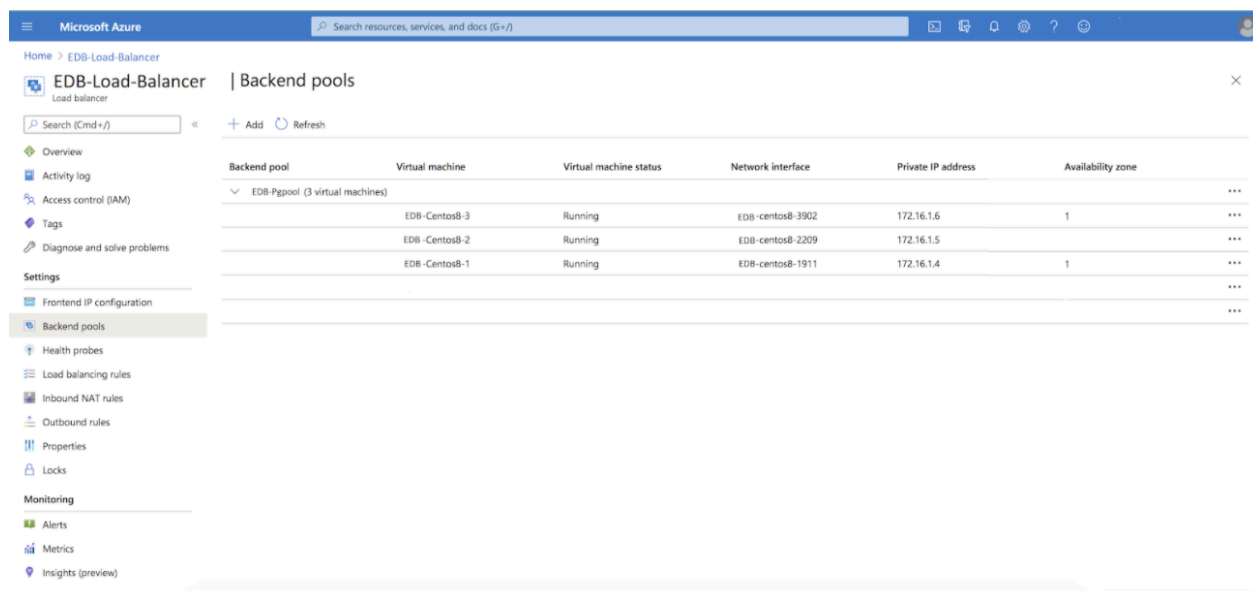


図 2 Azure コンソールのバックエンドプール

**Health Probe** :ヘルスプローブを追加して、仮想マシンで Pgpool インスタンスが使用可能かどうかを確認します。ヘルスプローブは、ポート 9999 でバックエンドプールの仮想マシンに定期的に ping を実行します。どの仮想マシンからも応答を受信しない場合、Pgpool インスタンスが利用できないと見なし、その特定のマシンへのトラフィック送信を停止します。

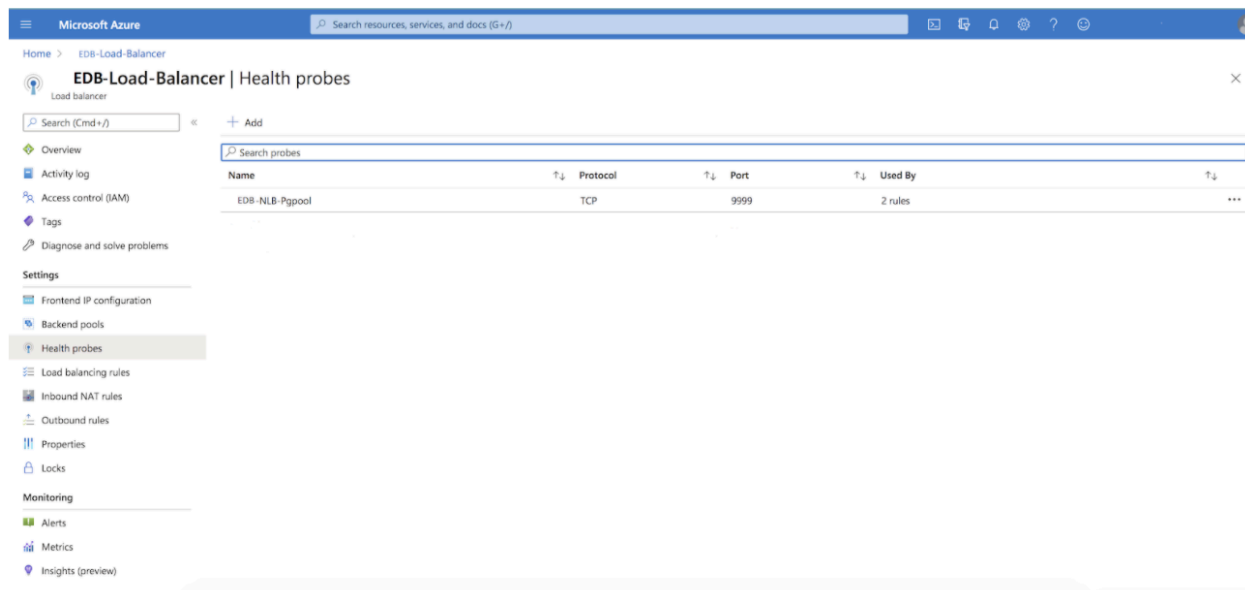


図 3 Azure コンソールのヘルスプローブ

**Load balancing rules** :2 つの負荷分散ルールを追加します-ポート 9898 とポート 9999 にそれぞれ 1 つ。これらの

ルールは、特定のポートに向かうネットワークトラフィックがバックエンドプールに存在するすべての仮想マシンに均等に分散されるようにします。

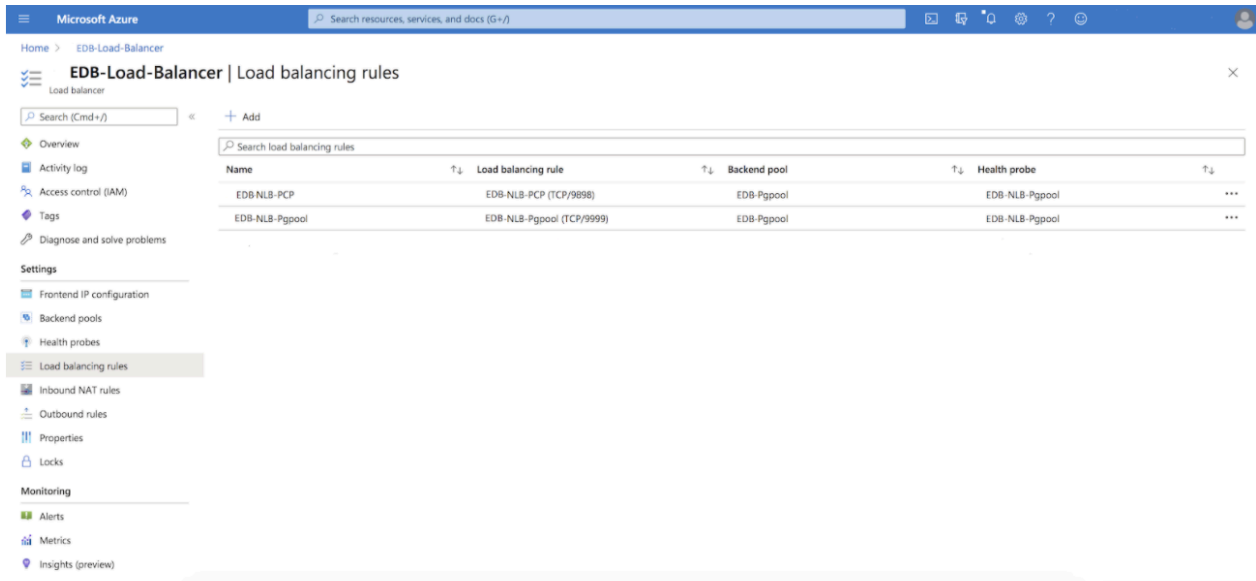


図 4 Azure コンソールの負荷分散ルール

### 1. ポート 9999 用に作成されたルール（つまり、PCP ポート）

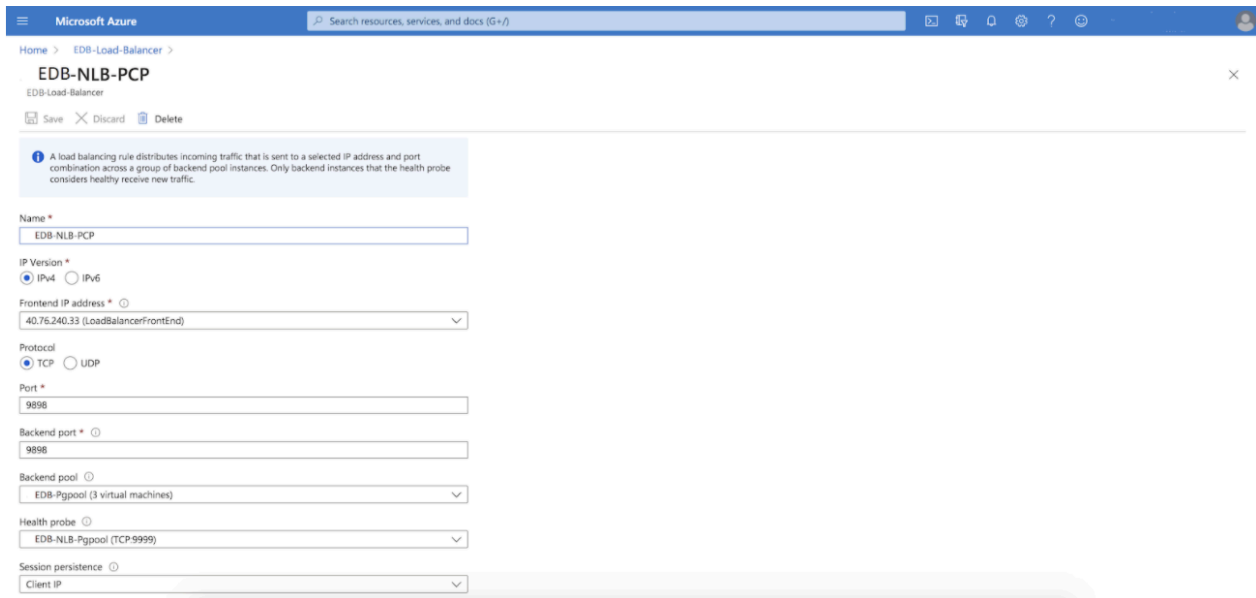


図 5 ポート 9999 のロードバランシングルール

### 2. ポート 9999 用に作成されたルール（つまり、Pgpool ポート）

上記のセットアップの構成後、ポート 9999 でネットワークロードバランサーの IP アドレスで Postgres に接続できます。プライマリデータベースサーバーで障害が発生した場合、EFM は新しいプライマリを昇格し、トラフィック

The screenshot shows the 'New Load Balancing Rule' configuration page in the Microsoft Azure portal. The rule is named 'EDB-NLB-Pgpool' and is associated with the 'EDB-Load-Balancer' resource. The configuration includes the following fields:

- Name \***: EDB-NLB-Pgpool
- IP Version \***: ☒ IPv4 ☐ IPv6
- Frontend IP address \***: 40.76.240.33 (LoadBalancerFrontEnd)
- Protocol**: ☒ TCP ☐ UDP
- Port \***: 9999
- Backend port \***: 9999
- Backend pool**: EDB-Pgpool (3 virtual machines)
- Health probe**: EDB-NLB-Pgpool (TCP-9999)
- Session persistence**: Client IP

A blue information box at the top states: 'A load balancing rule distributes incoming traffic that is sent to a selected IP address and port combination across a group of backend pool instances. Only backend instances that the health probe considers healthy receive new traffic.'

図 6 ポート 9999 のロードバランシングルール

クを再配布するように Pgpool を再構成します。Pgpool プロセスのいずれかがトラフィックを受け入れることができない場合、ネットワークロードバランサーは残りの 2 つの Pgpool プロセスにすべてのトラフィックを再分配します。listen\_backlog\_multiplier が、フェイルオーバーの場合により多くの接続を補償するように調整されていることを確認してください。



## 第 5 章

# Configuration for Number of Connections and Pooling

Pgpool には、プーリングと接続処理を調整するための構成がいくつかあります。この構成に応じて、`max_connections` の Postgres 構成も設定して、必要に応じてすべての接続が受け入れられるようにする必要があります。さらに、CloudArchitecture はアクティブ/アクティブインスタンスで動作するため、`num_init_children` をすべての Pgpool インスタンスに分散する必要があることに注意してください（通常使用される値をアクティブインスタンスの数で除算します）。以下のテキストでは、構成を変更した場合の影響について説明し、オンプレミスとクラウドアーキテクチャの両方の値を推奨しています。

**max\_pool** :通常、`max_pool` を 1 に設定することをお勧めします。また、再接続が多いアプリケーションの場合、`max_pool` をユーザー、データベース、およびアプリケーション接続の接続オプションの異なる組み合わせの数に設定できます。プール内の 1 つを除くすべての接続は古い接続になり、パフォーマンスを向上させることなく Postgres からの接続スロットを消費します。したがって、アクティブな接続と古い接続の健全な比率を維持するために、`max_pool` を 4 を超えて構成しないことをお勧めします。例として、常に再接続し、それぞれ独自のデータベースに接続する 2 人の異なるユーザーを使用するアプリケーションの場合、2 に設定します。両方のユーザーが両方のデータベースに接続できる場合は、4 に設定します。Pgpool で `num_init_children` を下げるか、Postgres で `max_connections` を調整します。

**num\_init\_children** :`num_init_children` をアクティブに並行して実行できる接続の数に設定することをお勧めしますが、この値はアクティブな Pgpool-II インスタンスの数（オンプレミスアーキテクチャのインスタンスとすべてのインスタンスクラウドアーキテクチャ用）。例:3 つの Pgpool インスタンスを持つアーキテクチャで、アプリケーションが 100 個のアクティブな接続を並行して使用できるようにするには、オンプレミスアーキテクチャの `num_init_children` を 100 に設定し、クラウドアーキテクチャの `num_init_children` を 33 に設定します。`num_init_children` を増やすには、通常、Postgres で `max_connections` を調整する必要があることに注意してください。

**listen\_backlog\_multiplier**:アクティブな接続の数（`num_init_children`）で開いている接続の数（アプリケーションが認識する）を乗算するように設定できます。例として、オンプレミスアーキテクチャで、アプリケーションが 500 の接続を開き、そのうち 100 が並行してアクティブになる場合、`num_init_children` を 100 に設定し、`listen_backlog_multiplier` を 4 に設定する必要があります。並列接続、および別の 400（`listen_backlog_multiplier*`

num\_init\_children) 接続は、接続がブロックされる前にキューに入れられます。アプリケーションは合計 500 のオープン接続を認識し、Postgres は常に最大 100 接続の負荷を処理します。listen\_backlog\_multiplier を増やすと、アプリケーションはより多くの接続を認識するようになりますが、並列アクティブ接続の数は増加しません (num\_init\_children によって決定されます)。

**max\_connections:**Postgres の max\_connections を [アクティブな pgpool インスタンスの数]\*[max\_pool]\*[num\_init\_children]+[superuser\_reserved\_connections] (Postgres) よりも高く設定することをお勧めします。例:3 つのインスタンスがアクティブ/パッシブ、max\_pool が 2 に設定され、num\_init\_children が 100 に設定され、superuser\_reserved\_connections (Postgres) が 5 に設定されたオンプレミスセットアップで、Postgresmax\_connections は  $[1*2*100+5]$ 。これは 205 接続以上です。クラウドセットアップの同様のセットアップは、3 つのアクティブインスタンス、max\_pool を 2 に設定、num\_init\_children を 33 に設定、superuser\_reserved\_connections (Postgres) を 5 に設定して実行します。この場合、Postgresmax\_connections は  $[3*2*33+5]$ 203 以上です。推奨設定以下に設定すると、新しい接続を開く際に問題が発生する可能性があり、max\_pool と組み合わせて予期しない動作を引き起こす可能性があることに注意してください num\_init\_children、max\_pool、max\_connections の関係については、この背景情報を参照してください。

## 第 6 章

# Conclusion

### EDB Postgres High Availability and Horizontal Read Scaling Architecture Guide

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

EnterpriseDB®Corporation34CrosbyDrive、Suite201、Bedford、MA01730、USA

T+17813573390

F+19784671307E

[info@enterprisedb.com](mailto:info@enterprisedb.com)

[www.enterprisedb.com](http://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、それぞれの所有者の商標です。
- 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危険な入力を禁止することはできません。
- 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に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, 5  
Architecture Overview, 1  
  
Conclusion, 23  
Configuration for Number of Connections and  
    Pooling, 21  
  
EFM overview, 2  
EFM Pgpool Integration Using Azure Network Load  
    Balancer, 15  
  
Failover Manager configuration file, 9  
  
Implementing High Availability with Pgpool, 9  
  
Pgpool configuration file, 10  
Pgpool overview, 3