

High Availability & Scalability Guide 4.2

EnterpriseDB

High Availability & Scalability Guide

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パートではないため、Advanced Server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、EDB Failover Managerと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。

序文アーキテクチャcomponents_ha_pgpool appendix_a appendix_b結論

</div>

Architecture Overview

このガイドでは、Failover ManagerとPgpoolを構成してAdvanced Serverに提供する利点を最大限に活用する方法を説明します。アーキテクチャのセクションで説明したリファレンス・アーキテクチャを使用すると、（フェイルオーバーマネージャーとの）自動フェイルオーバーメカニズム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大規模なワークロードのためのシステムおよび読み取り中心または混在ワークロードとの同時クライアント数の増加をスケーリングしながら、（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されているアーキテクチャは、EFM 4.2、EDB pgPool 4.2、およびAdvanced Server 13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、EnterpriseDBから入手できます。

<https://www.enterprisedb.com/docs/>

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>

フェイルオーバーマネージャーの概要

フェイ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。データベース障害が発生すると、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プロモートさせ、継続的なパフォーマンスを保証し、最小限のサービス中断でデータロスを防ぎます。

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は、ネットワーク上の次のホストに存在するEFMプロセスで構成されます。

- プライマリノードは、データベースクライアントにサービスを提供しているプライマリデータベースサーバです。
- 1つ以上のスタンバイノードは、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です。
- ウィットネスノードは、フェイルオーバーシナリオでプライマリまたはスタンバイのアクションを確認します。障害が発生したシチュエーションで、プライマリが半分以上のノードを持つパーティションで検出さ自分自身と、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。

Pgpool-IIの概要

Pgpool-II (Pgpool) は、EPASおよびコミュニティPostgresクラスターのマルチプルのスタンバイでのSELECTクエリの水平スケーラビリティのためのコネクション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について、`backend_weight`パラメータは、読み取りトラフィックの比率をバックエンドノードに向けるように設定できます。プライマリノードでの読み取りトラフィックを防ぐために、`backend_weight`パラメータを0に設定できます。そのような場合、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るため、読み取り集中型のワークロードが混在する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

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と呼ばれるインタフェースを提供します。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。

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。

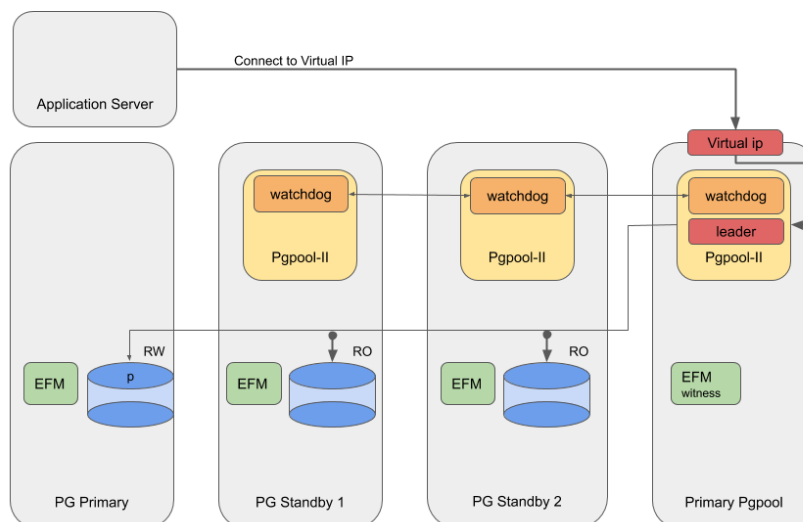
- `pgpool`サービスのヘルスチェック
- 他のウォッチドッグプロセスの相互モニタリング
- 特定の障害が検出された場合のリーダー/スタンバイ状態の変更

- サーバーの切り替えに同期的した自動仮想IPアドレスの割り当て
- 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、次を参照してください。

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

Architecture



A typical EFM and Pgpool configuration

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。

Systems	Components
Primary Pgpool/EFM witness node	プライマリPgpoolノードは、PgpoolとEFM witnessのみを実行します。そのため、できるだけ多くのリソースをPgpoolで使用できます。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、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。

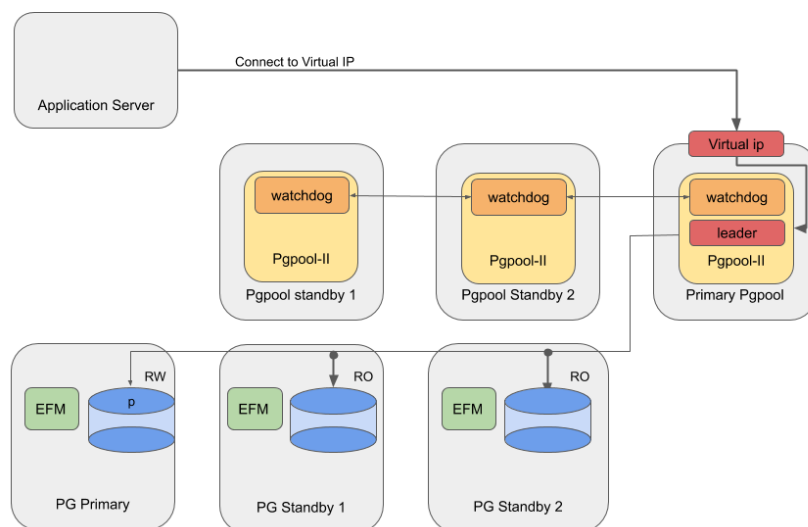
Systems	Components
Primary Database node	プライマリデータベースノードは、Postgres（プライマリ）とEFMのみを実行し、すべてのリソースをPostgresに残します。読み取り/書き込みトラフィック（つまり、INSERT、UPDATE、DELETE）は、プライマリPgpoolノードによってこのノードに転送されます。
Standby nodes	スタンバイノードは、Postgres（スタンバイ）、EFM、および非アクティブなPgpoolプロセスを実行しています。プライマリデータベースに障害が発生した場合、EFMはこれらのスタンバイノードのいずれかでPostgresをプロモート、読み書きトラフィックをハンドルします。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ハンドルします。二重障害シチュエーション（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）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、混在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。
- プライマリpgpoolノードで読み取り専用トラフィックを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ロードします。
- 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。
- 必要に応じて、同期レプリケーションイベントをセットアップして、障害発生時にほぼゼロのデータロスを実現できます。

Note * このアーキテクチャにより、Postgresを実行する3つの仮想マシンをPgpoolを実行する3つの仮想マシンから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、

EFM Witness Process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、efm witnessエージェントをPgpoolサーバーに展開できます。



Deployment of EFM and Pgpool on separate virtual machines

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イベントに、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イルオーバーを行います。Pgpoolはバックエンドノードの状態をモニタせず、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。

フェールオーバーマネージャーの設定

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EFMを高可用性用に構成するには、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 = User that would be invoking PCP commands`

`pcp.host =`

Virtual IP that would be used by pgpool. Same as pgpool parameter 'delegate_IP'

`pcp.port = The port on which pgpool listens for pcp commands`

`pcp.pass.file = Absolute path of PCPPASSFILE`

`pgpool.bin = Absolute path of pgpool bin directory`

Pgpoolの構成

このセクションでは、`edg_tran_1`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て、Pgpool-IIをEFMと統合します。

バックエンドノードの設定

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`make`、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'
```

負荷分散およびストリーミングレプリケーションモードを有効化

`pgpool.conf` ファイルで次の設定パラメータして、負荷分散とストリーミングレプリケーションモードを有効にしモード。

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

ヘルスチェックとフェイルオーバーを無効にします

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、次のことを確認してください。

- pgpool.confのwd_priorityの値を各ノードで異なるようにします。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 _ *の正しいインタフェース値を更新し、pgpool.confファイルのcmd props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。

** PCPのセットアップ**

スクリプトはPCPインタフェースを使用するため、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。

セットアップPCP：<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保証に、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イ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、Failover Managerとの競合が発生したり、早期にフェイ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、フェイルオーバーマネージャー・トリガ・フェイルオーバー時にはpgpoolサービスのプロンプトリカバリを確実にする保証に低い値に設定されています。

仮想IPアドレス

Pgpool-IIとFailover Managerの両方が、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-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。このデザインのように、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。EFM VIPはこのデザインには有益な効果がなく、無効にする必要があります。

Pgpoolのアクティブなインスタンス（サンプルアーキテクチャのプライマリPgpoolサーバー）の障害シチュエーションでは、次に使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順位に従って）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。

Pgpool-II Watchdogの設定

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。

すべてのPgpoolノードでの一般的なウォッチドッグ構成

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。インターバルとリトライの値は、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
```

Note * eth0の値をシステムのネットワークインタフェースに置き換えます。接続数のチューニング、およびプール構成については、[Chapter 5](#)を参照してください。

サーバー2のウォッチドッグ構成

```
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
```

サーバー3のウォッチドッグ構成

```
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
```

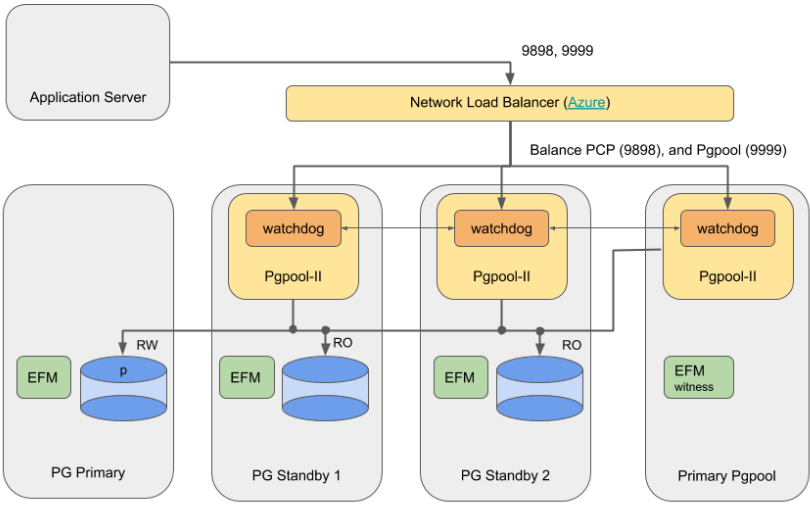
サーバー4のウォッチドッグ構成

```
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
```

EFM Pgpool Integration Using Azure Network Load Balancer

</ div>

このセクションでは、データベース、EFM、およびPgpoolがAzureのCentOS 8仮想マシンにインストールされる、EFM Pgpool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、Azure Load Balancer (LNB) を使用して、Pgpool VIP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。



Architecture diagram for EFM and Pgpool integration using Azure Load Balancer

ステップ1（インストール）：

次のように、Azure Virtual MachinesにAdvanced Server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	Advanced Server 13およびFailover Manager 4.2を実行しているプライマリノード

Systems	Components
Standby 1	Advanced Server 13、Failover Manager 4.2、およびPgpool 4.2を実行しているスタンバイノード。
Standby 2	Advanced Server 13、Failover Manager 4.2、およびPgpool 4.2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4.2およびPgpool 4.2を実行している監視ノード。

ステップ2（Pgpool設定）：

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）。

ステップ3（Azureロードバランサーの構成）：

Azure NLBを使用するには、次の構成を行う必要があります。

ネットワーキング：割り当て公衆IPだけでなく、NLBにプライベートIP、および仮想マシンにのみ、プライベートIP：あなたは、ネットワーク負荷バランサ用と仮想マシンのそれぞれについて、以下の設定を保証する必要があります。アプリケーションサーバーはパブリックIPを介してNLBに接続し、NLBはプライベートIPを介して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の公衆IP：40.76.240.33（PCPホスト。）
- 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パートではないことに注意してノート）
- スタンバイ1のプライベートIP：172.16.1.4
- スタンバイ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します）。

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

バックエンドプール：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。

Backend pool	Virtual machine	Virtual machine status	Network interface	Private IP address	Availability zone
EDB-Pgpool (3 virtual machines)	EDB-CentOS-3	Running	EDB-centos8-3902	172.16.1.6	1
	EDB-CentOS-2	Running	EDB-centos8-2209	172.16.1.5	1
	EDB-CentOS-1	Running	EDB-centos8-1911	172.16.1.4	1

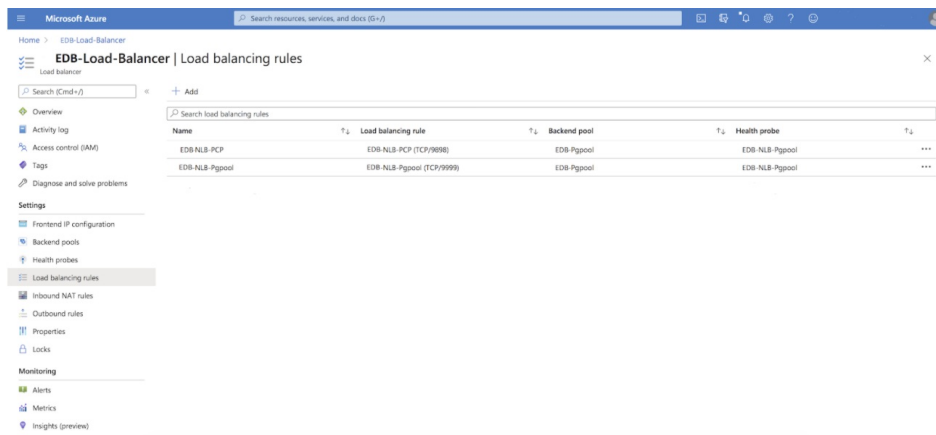
Backend pool in Azure console

ヘルスプローブ：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。

Name	Protocol	Port	Used By
EDB-NLB-Pgpool	TCP	9999	2 rules

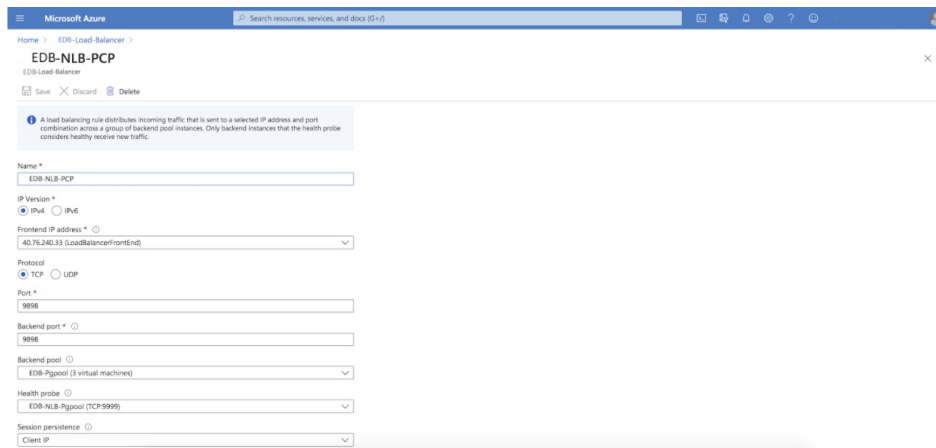
Health probes in Azure console

負荷分散ルールは：2つのロードバランシングのルールを追加します- 1 ずつポート9898とポート9999これらの規則のためのネットワークトラフィックが特定のポートに向かってくる仮想マシンがバックエンドプール内に存在するすべてに均等に分散されることを保証必要があります。



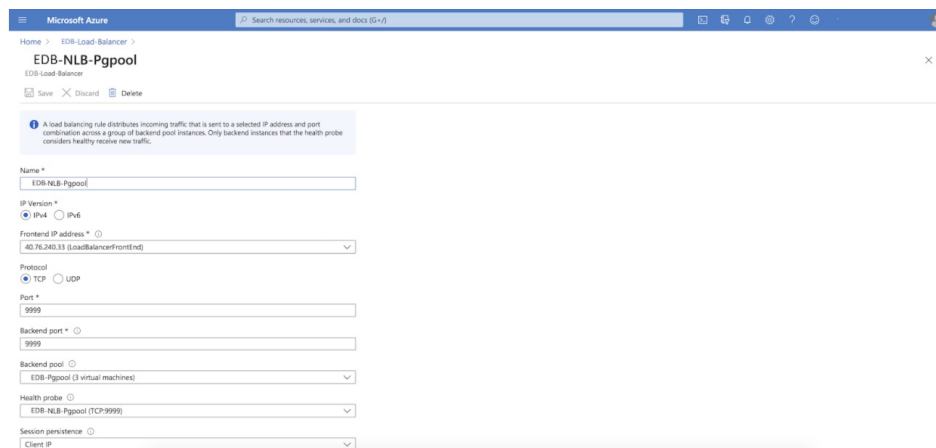
Load balancing rules in Azure console

1.ポート 9999用に作成されたルール（つまり、PCPポート）



Load balancing rule for port 9999

1.ポート 9999用に作成されたルール（つまり、Pgpoolポート）



Load balancing rule for port 9999

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で障害が発生した場合、EFMは新し

いブライマリをプロモート、トラフィッ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。フェイルオーバーの場合、listen_backlog_multiplierがより多くの接続数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。

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、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。また、クラウドのアーキテクチャは、すべてのpgpoolのインスタンス（アクティブなインスタンスの数で通常使用される値を分割）上に拡散 `num_init_children` に必要とする、アクティブ/アクティブ・インスタンスで動作することに留意されノート。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、多くの再接続を行うアプリケーションの場合、`max_pool` は、アプリケーション接続のユーザー、データベース、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からのコネクション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、4を超える `max_pool` を構成しないことをお勧めします。例として、絶えず再接続し、両方とも自分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 ****：並行してアクティブに実行できる接続の数に `num_init_children` を設定することをお勧めしますが、値をアクティブなPgpool-IIインスタンスの数で割る必要があります（オンプレミスアーキテクチャの場合、クラウドアーキテクチャのすべてのインスタンス）。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行えるようにするには、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。通常、`num_init_children` を増やすには、Postgresで `max_connections` を調整する必要があります。

**** listen_backlog_multiplier ****：（アプリケーションによって認識される）オープン接続の数とアクティブな接続の数（`num_init_children`）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、100が並行してアクティブになる500接続をアプリケーションが開く場合、`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。並行して、接続がブロックされる前に別の400（`listen_backlog_multiplier*num_init_children`）接続が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ロード。`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しますが、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 $[\text{number of active pgpool instances}] * [\text{max_pool}] * [\text{num_init_children}] + [\text{superuser_reserved_connections}]$ (Postgres) より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、**num_init_children** が100、**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は、Postgres **max_connections** は $[1 * 2 * 100 + 5]$ 以上であり、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に設定して実行されます。この場合、Postgres **max_connections** は203以上の $[3 * 2 * 33 + 5]$ 以上に設定する必要があります。推奨設定以下で設定すると、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** との組み合わせで予期しない動作が発生する可能性があります（アクティブな接続が少ないかアクティブではありませんが、Postgresの接続スロットを使用した古いプール接続による接続の問題が引き続き発生する可能性があります）。**num_init_children**、**max_pool**、および **max_connections** のリレーションについては、この背景情報を参照してください。

[page]