



# High Availability Routing for Postgres (HARP)

リリース 2.0.3

EDB

2022 年 04 月 07 日



# 目次

|              |                                    |           |
|--------------|------------------------------------|-----------|
| <b>第 1 章</b> | <b>Release Notes</b>               | <b>3</b>  |
| 1.1          | リリース 2.0.3 (2022-3-31)             | 3         |
| 1.1.1        | 機能強化                               | 3         |
| 1.1.2        | バグ修正                               | 3         |
| 1.2          | リリース 2.0.2 (2022-2-24)             | 3         |
| 1.2.1        | 機能強化                               | 3         |
| 1.2.2        | バグ修正                               | 4         |
| 1.2.3        | 既知の問題                              | 5         |
| 1.3          | リリース 2.0.1 (2022-1-31)             | 5         |
| 1.3.1        | 機能強化                               | 5         |
| 1.4          | リリース 2.0.0 (2021-12-01)            | 5         |
| 1.4.1        | エンジン                               | 5         |
| 1.4.2        | 設定                                 | 6         |
| 1.4.3        | 機能強化                               | 6         |
| 1.5          | リリース 1.0.1 (2021-06-23)            | 6         |
| 1.5.1        | ドキュメント                             | 6         |
| 1.5.2        | バグ修正                               | 7         |
| 1.6          | リリース 1.0 (2021-06-15)              | 7         |
| 1.6.1        | 機能強化                               | 7         |
| 1.6.2        | バグ修正                               | 7         |
| 1.7          | リリース 0.2 (2021-02-23)              | 7         |
| 1.8          | リリース 0.1 (2020-08-13)              | 7         |
| <b>第 2 章</b> | <b>HARP Functionality Overview</b> | <b>9</b>  |
| 2.1          | クォーラムの重要性                          | 9         |
| 2.2          | 書き込みターゲットの削減                       | 9         |
| 2.3          | 基本アーキテクチャ                          | 10        |
| 2.4          | 仕組み                                | 11        |
| 2.4.1        | リーダーの決定                            | 13        |
| 2.4.2        | 接続ルーティング                           | 13        |
| 2.4.3        | コロケーション                            | 14        |
| 2.5          | 推奨されるアーキテクチャと使用                    | 14        |
| 2.5.1        | ロケーション設定                           | 16        |
| 2.5.2        | BDR 3.7 互換性                        | 16        |
| <b>第 3 章</b> | <b>Installation</b>                | <b>17</b> |
| 3.1          | ソフトウェアバージョン                        | 17        |
| 3.2          | TPAExec                            | 17        |
| 3.3          | パッケージのインストール                       | 18        |
| 3.3.1        | etcd パッケージ                         | 18        |

|              |                                                |           |
|--------------|------------------------------------------------|-----------|
| 3.4          | コンセンサスレイヤー                                     | 18        |
| 3.4.1        | etcd                                           | 18        |
| 3.4.2        | BDR                                            | 19        |
| <b>第 4 章</b> | <b>Configuring HARP for Cluster Management</b> | <b>21</b> |
| 4.1          | 標準構成                                           | 21        |
| 4.1.1        | クラスター名                                         | 22        |
| 4.1.2        | DCS 設定                                         | 22        |
| 4.1.3        | 例                                              | 23        |
| 4.1.4        | HARP マネージャー固有                                  | 23        |
| 4.1.5        | HARP プロキシ固有                                    | 24        |
| 4.1.6        | 例                                              | 25        |
| 4.2          | 実行時ディレクティブ                                     | 25        |
| 4.2.1        | クラスター全体                                        | 25        |
| 4.2.2        | ノードディレクティブ                                     | 26        |
| 4.2.3        | プロキシディレクティブ                                    | 28        |
| <b>第 5 章</b> | <b>Cluster Bootstrapping</b>                   | <b>31</b> |
| 5.1          | クラスター全体のブートストラップ                               | 31        |
| 5.2          | 場所のブートストラップ                                    | 32        |
| 5.3          | ノードのブートストラップ                                   | 32        |
| 5.4          | プロキシブートストラップ                                   | 33        |
| <b>第 6 章</b> | <b>HARP Manager</b>                            | <b>35</b> |
| 6.1          | 仕組み                                            | 35        |
| 6.2          | 設定                                             | 36        |
| 6.3          | 使用法                                            | 36        |
| 6.4          | Postgres の HARP Manager 制御の無効化と再有効化            | 37        |
| <b>第 7 章</b> | <b>HARP Proxy</b>                              | <b>39</b> |
| 7.1          | 仕組み                                            | 39        |
| 7.2          | 設定                                             | 39        |
| 7.3          | 使用法                                            | 40        |
| 7.4          | PgBouncer 構成ファイル                               | 40        |
| 7.5          | HARP プロキシノード管理の無効化と再有効化                        | 41        |
| 7.6          | パススルーユーザー認証                                    | 41        |
| 7.7          | 仕組み                                            | 42        |
| 7.8          | 設定                                             | 42        |
| 7.9          | 使用法                                            | 43        |
| <b>第 8 章</b> | <b>harpctl Command-line Tool</b>               | <b>45</b> |
| 8.1          | 概要                                             | 45        |
| 8.2          | 設定                                             | 46        |
| 8.3          | 実行                                             | 46        |
| 8.4          | harpctl apply                                  | 46        |
| 8.5          | harpctl fence                                  | 47        |
| 8.6          | harpctl get                                    | 48        |
| 8.6.1        | harpctl get cluster                            | 48        |

|               |                                       |           |
|---------------|---------------------------------------|-----------|
| 8.6.2         | harpctl get leader . . . . .          | 48        |
| 8.6.3         | harpctl get location . . . . .        | 49        |
| 8.6.4         | harpctl get locations . . . . .       | 49        |
| 8.6.5         | harpctl get node . . . . .            | 49        |
| 8.6.6         | harpctl get nodes . . . . .           | 50        |
| 8.6.7         | harpctl get proxy . . . . .           | 50        |
| 8.6.8         | harpctl get proxies . . . . .         | 50        |
| 8.7           | harpctl manage . . . . .              | 51        |
| 8.8           | harpctl promote . . . . .             | 51        |
| 8.9           | harpctl set . . . . .                 | 52        |
| 8.9.1         | harpctl set cluster . . . . .         | 52        |
| 8.9.2         | harpctl set node . . . . .            | 52        |
| 8.9.3         | harpctl set proxy . . . . .           | 52        |
| 8.10          | harpctl unfence . . . . .             | 53        |
| 8.11          | harpctl unmanage . . . . .            | 53        |
| <b>第 9 章</b>  | <b>Consensus Layer Considerations</b> | <b>55</b> |
| 9.1           | BDR ドライバーの互換性 . . . . .               | 55        |
| 9.2           | クォーラムの維持 . . . . .                    | 55        |
| 9.2.1         | マルチコンセンサスバリエント . . . . .              | 56        |
| 9.2.2         | TPAexec とコンセンサス . . . . .             | 57        |
| <b>第 10 章</b> | <b>Security and Roles</b>             | <b>59</b> |
| 10.1          | Postgres の許可 . . . . .                | 59        |
| 10.2          | BDR のアクセス許可 . . . . .                 | 59        |
| 10.3          | BDR コンセンサス権限 . . . . .                | 60        |



Postgres の高可用性ルーティング (HARP) は、*Bi-directional Replication (BDR)* クラスター用のクラスター管理ツール。このツールのコアデザインは、単一のデータセンターまたはリージョン内のすべてのアプリケーショントラフィックを一度に 1 つのノードのみにルーティングすることです。リードマスターと呼ばれるこのノードは、データの競合の可能性を減らすための主要な書き込みターゲットとして機能します。

HARP は、分散コンセンサスモデルを活用して、クラスター内の BDR ノードの可用性を判断します。Lead Master が失敗または使用不可になると、HARP は新しい Lead Master を選択し、それに応じてアプリケーショントラフィックをリダイレクトします。

BDR のコア機能と一緒に、アプリケーショントラフィックをリードマスターノードにルーティングするこのメカニズムにより、データロスリスクなしにファストフェイルオーバーとスイッチオーバーが可能になります。

HARP には、BDR バージョン 3.6 以降が必要です。



# 第1章 Release Notes

## 1.1 リリース 2.0.3 (2022-3-31)

### 1.1.1 機能強化

- **HARP プロキシは、読み取り専用のユーザ専用 TLS 証明書をサポート** DCS 読み取り専用ユーザの HARP プロキシ DSN で TLS 証明書とキーを指定できます。(78516、HNG-522)

### 1.1.2 バグ修正

- **HARP プロキシは、50 秒後に終了する代わりに、DCS への接続を試行し続けます。**(75406、HNG-548、HNG-560、HNG-561)

## 1.2 リリース 2.0.2 (2022-2-24)

### 1.2.1 機能強化

- **ビルトインプロキシの接続調整** builtin プロキシが使用できる接続の最大数を指定できるようになりました。プロキシは、計算されたシステムリソース制限内に収まるように接続数を下方に調整します。(75406、79250、HNG-489、HNG-498、HNG-503、HNG-508)
- **BDRDCS の CAMO は無効** HARP は、データベースへの接続で CAMO を無効にして、Distributed Consensus System (DCS) で BDR を使用する際のパフォーマンスの低下を回避します。(HNG-438)
- **HARP プロキシのセキュリティの向上** builtin プロキシの読み取り専用 DCS 権限を持つユーザを指定できます。(75406、HNG-452)
- **Postgres を管理するための開始、停止、ステータスフック** postgres の起動、postgres の停止、および postgres のステータスの取得のために、HARP マネージャ構成でスタート、stop、および status コマンドを提供できます。コマンドを提供しない場合、デフォルトで systemd が使用されます。(HNG-492)
- **HARPProxyrpm および deb パッケージソフトの依存関係として、Pgouncer が削除されました。** harp\_proxy\_mode として pgbouncer を選択しない限り、Pgouncer はインストールされません。(HNG-511)
- **HARPManager は BDRDCS との通信パフォーマンスを改善しました** HARP Manager は、ローカル UNIX ドメインソケットを使用して BDR DCS とのみ通信します。(78516、HNG-494)

- ビルトインプロキシがデフォルトのプロキシになりました `pgbouncer` が以前のリリースでデフォルトで使用されていた場合、`harp_proxy_mode` を `pgbouncer` として指定して、アップグレード時にプロキシとして続行する必要があります。(HNG-511)
- バイナリがサービス名とマッチするようになりました `harp_manager` バイナリの名前付けは `harp-manager` になりました。同様に、`harp_proxy` バイナリの名前付けは `harp-proxy` になりました。以前の名前のシンボリックリンクが提供されます。(HNG-514)
- **HarpManager のデフォルトの改善** ラグ構成のデフォルトがデフォルトでオフに設定されるようになりました。リーダーリースの期間のデフォルト設定は 6 秒になりました。リーダーリースの更新設定がデフォルトで 2 秒になりました。(HNG-520)

### 1.2.2 バグ修正

- HARP Manager は、終了時にデータベースを停止するようになりました。(HNG-497)
- `builtin` プロキシの使用時に HARP プロキシが接続をリークしなくなりました。(75406、HNG-445)
- `builtin` プロキシの接続が閉じられたときに、HARP プロキシが誤って「スプライス：同等なによる接続のリセット」を報告しなくなりました。(75406、HNG-445、)
- `builtin` または `pgbouncer` プロキシステータスのクエリ時に `Harpctl` が返るようになりました。(HNG-499)
- `harpctl get cluster` の出力には、リーダーおよび以前のリーダーフィールドが含まれなくなりました。(HNG-483)
- `Harpctl` は、プロキシ関連のコマンドを実行するときにプロキシ名前を検証するようになりました。(HNG-471)
- `Harpctl` は、リーダーの正しいルーティングステータスを報告するようになりました。(HNG-441)
- HARP 構成ファイルには、対応するプロキシ `builtin` または `pgbouncer` の正しい構成パラメーターが含まれるようになりました。(78516、HNG-456)
- `TPAExec` は、重複したユーザを持つ DCS エンドポイントの紛らわしい DSN を作成しなくなりました。(78516、HNG-495)
- `request_timeout` 設定パラメータには、ミリ秒単位の単位を指定する必要はありません。(78363、HNG-504)
- `listen_port` および `listen_host` の設定は、`TPAExec` を使用してプロキシインスタンスごとに構成できるようになりました。(78848、HNG-456) \*リーダーノードになれないサブスクライバーのみのノードは、リーダーシップの対象から除外されます。(78516、HNG-411)

### 1.2.3 既知の問題

以前に隔離されたシャドウノードがアクティブなクラスターノードとして戻ると、いかに選挙とリーダーシップの変更がトリガーされます。ノードがプロモートされる場合があります。適格なノード間でリーダーシップを競います。最初の適格なノードがリーダーになります。プロモートオプションを Promote コマンドで使用して、目的のノードをリーダーにします。\* HARP Proxy が BDR DCS の読み取り専用ユーザアクセスで構成されている場合、Harpctl は HARP プロキシバージョンを結果することができません。バージョン情報は、読み取り専用権限を持つユーザが保存することはできません。これにより、harpctl を使用してバージョン情報をクエリーするときにプロキシのバージョン情報が失われます。データベースの停止オプションでデータベースをフェンシングした後、HARP マネージャーが再起動され、BDR DCS が設定されている場合、データベースは再起動されますが、フェンス状態になります。EDB Postgres Advanced Server を展開する場合、use\_unix\_sock は機能しません。EDB Postgres Advanced Server のデフォルトの UNIX ソケットディレクトリは正しく決定されていません。

## 1.3 リリース 2.0.1 (2022-1-31)

### 1.3.1 機能強化

- BDR コンセンサスが一般公開されました

HARP は、分散コンセンサスサービス (DCS) ソースに etcd と BDR のマルチプルのオプションを提供します。BDR コンセンサスオプションは、etcd が存在しない展開で使用できます。BDR コンセンサスオプションの使用はベータ版とは見なされなくなり、稼働環境での使用がサポートされるようになりました。

\*トランスポートレイヤープロキシが一般公開

HARP は、クライアントアプリケーションとデータベース間の接続をルーティングするために、アプリケーションレイヤー (L7) とトランスポートレイヤー (L4) のマルチプルのプロキシオプションを提供します。ネットワークレイヤー 4 またはトランスポートレイヤープロキシは、ネットワークパケットを単に転送しますが、レイヤー 7 はネットワークトラフィックを終了します。以前はシンプルプロキシと呼ばれていたトランスポートレイヤープロキシは、ベータ版とは見なされなくなり、稼働環境での使用がサポートされるようになりました。

## 1.4 リリース 2.0.0 (2021-12-01)

### 1.4.1 エンジン

すべての操作を最適化するための `golang` のシステムの完全な書き換えクラスタの状態を YAML 経由でブートストラップまたは修正できるようになりました

### 1.4.2 設定

- YAML で書き直されました\*構成ファイルを `harp.ini` から `config.yml` に変更

### 1.4.3 機能強化

- HARP プロキシは、サポートされているアーキテクチャでの HAProxy の必要性を非推奨にします。

HARP ルーターを使用して DCS コンテンツを HTTP ベースの URI 要求の適切なオンラインまたはオフライン状態に変換することは、リードマスターを決定するためにロードまたは HAProxy が必要であることを意味しました。HARP プロキシは、定期的な反復ステータスチェックなしでこれを自動的に行います。

- DCS キーサブスクリプションを使用して、状態の変化に直接応答します。

関連するクラスターの状態が変化すると、クラスターはすぐに応答するため、フェイルオーバーとスイッチオーバーの時間が改善されます。

- etcd SSL 設定との互換性。

SSL 暗号化を介して etcd と通信できるようになりました。

\*スイッチオーバー時のトランザクションラグはゼロ。

レプリケートされたすべてのトランザクションが再生されるまで、新しいリードノードにはトランザクションがルーティングされないため、競合の可能性が減少します。

\*実験的 BDR コンセンサスレイヤー

分散コンセンサスサービス (DCS) として BDR コンセンサスを使用すると、実装に必要な変更の量が削減されます。

\*実験プロキシ

セッション制御を強化するためのプロキシ実装。

## 1.5 リリース 1.0.1 (2021-06-23)

### 1.5.1 ドキュメント

- HARP の頭字語の解像度を標準化する

## 1.5.2 バグ修正

- CAMO ラグチェックを修正して、`maximum_camo_lag` が 0 に設定されている場合に対応

## 1.6 リリース 1.0 (2021-06-15)

### 1.6.1 機能強化

- `--dry-run` オプションが `harpctl leader set` に追加されました\*最小構成値が適用されます
- `lock_interval` パラメータは秒の小数部として指定できます**ロギングと出力の改善**並列適用をハンドルするためにレプリケーションクエリーが更新されました

### 1.6.2 バグ修正

- `leader set` が失敗すると、`harpctl` はエラーコードを返します**ノードの進行状況が返されない場合のコーナーケースエラーを防止**潜在的に空のノードレコードをハンドル\*リードノードの候補を決定するときに未処理の例外をキャッチする

## 1.7 リリース 0.2 (2021-02-23)

これは、次の変更を含むメンテナンスリリースです。

- EnterpriseDB カスタマーポータルから入手可能な文書\*リードマスタ以外のノードの非可用性をレポート
- BDR を DCS レイヤーとして使用する場合、潜在的な障害状況を修正します。BDR ノードが実行されていません
- RPM パッケージの問題を修正し、新規インストールでの新規スタートを防止

## 1.8 リリース 0.1 (2020-08-13)

これは、BDR の HARP サポートを提供する最初のベータリリースです。

**コンセンサスレイヤーとしてのネイティブ BDR (3.6.21 以降) の使用**コンセンサスレイヤーとしての `etcd` の使用

BDR がコンセンサスレイヤーとして使用される場合、現在 HARP は物理的ストリーミングレプリカでのオペレーションをサポートしていないことに注意してください。



## 第2章 HARP Functionality Overview

HARP は、BDR クラスターの高可用性への新しいアプローチです。コンセンサス主導のクォーラムを活用して、半排他的な方法で正しい接続エンドポイントを決定し、アプリケーションからの意図しないマルチノード書き込みを防ぎます。

### 2.1 クォーラムの重要性

HARP の主な目的は、管理する Postgres クラスターにフルクォーラムを適用することです。定足数は、一般的に投票機関に適用される用語であり、特定の最小限の出席者が決定を make ことを義務付けています。または、おそらくもっと簡単に：マジョリティールール。

投票が同オーダー以外の結果で終わるためには、奇数個のノードが完全なクラスターメンバーシップを構成する必要があります。ただし、定足数はこの制約を厳密に要求しません。シンプル過半数で十分です。これは、N ノードのクラスターでは、クォーラムが意味のある投票を行うために最低  $N/2 + 1$  ノードを必要とすることを意味します。

このすべてにより、クラスターがどのノードを「担当」するかに関して常に合意することが保証されます。マルチプルのノードで構成される BDR クラスターの場合、これにより、どのノードがプライマリ書き込みターゲットであるかが決まります。HARP は、このノードをリードマスターとして指定します。

### 2.2 書き込みターゲットの削減

クォーラムの概念を無視するか、それを不十分に適用すると、「正しい」書き込みターゲットがあいまいまたは不明なスプリットブレインシナリオが発生する可能性があります。標準の Postgres クラスターでは、1 つのノードのみが書き込み可能で、残りのノードにレプリケーショントラフィックを送信することが重要です。

BDR などのマルチマスター対応のアプローチでも、必要な競合管理の量を減らして、クラスター全体で同一のデータを取得することは有益です。物理的場所または地域ごとにマルチプルの BDR ノードで構成されるクラスターでは、これは通常、単一の BDR ノードが「リーダー」として機能し、残りのノードが「シャドウ」であることを意味します。これらの Shadow ノードは引き続き書き込み可能ですが、どうしても必要な場合を除き、書き込みはお勧めできません。

Quorum を活用することで、すべてのノードがどの Postgres ノードがクラスター全体またはローカル BDR リージョンを表すかを正確に一致させることができます。定義、クォーラムの残りの部分との接続を失ったノード、またはそれによって無効にされたノードは、クラスターリーダーになることはできません。

これにより、書き込みが意図せずに 2 つの Postgres ノードに到達するスプリットブレインの状況を防ぎます。VPN、プロキシ、ロード、DNS などのテクノロジーとは異なり、構成の誤りやネットワークパーティショ

ンによって定足数から派生したコンセンサスを回避することはできません。コンセンサスレイヤーに連絡して HARP によって維持されるクォラムの状態を判断できる限り、有効なターゲットは 1 つだけです。

## 2.3 基本アーキテクチャ

HARP のデザインは、基本的にマネージャーとプロキシからなる 2 つの部分で構成されています。次の図は、これらが単一の Postgres インスタンスとどのように相互作用するかを説明しています。

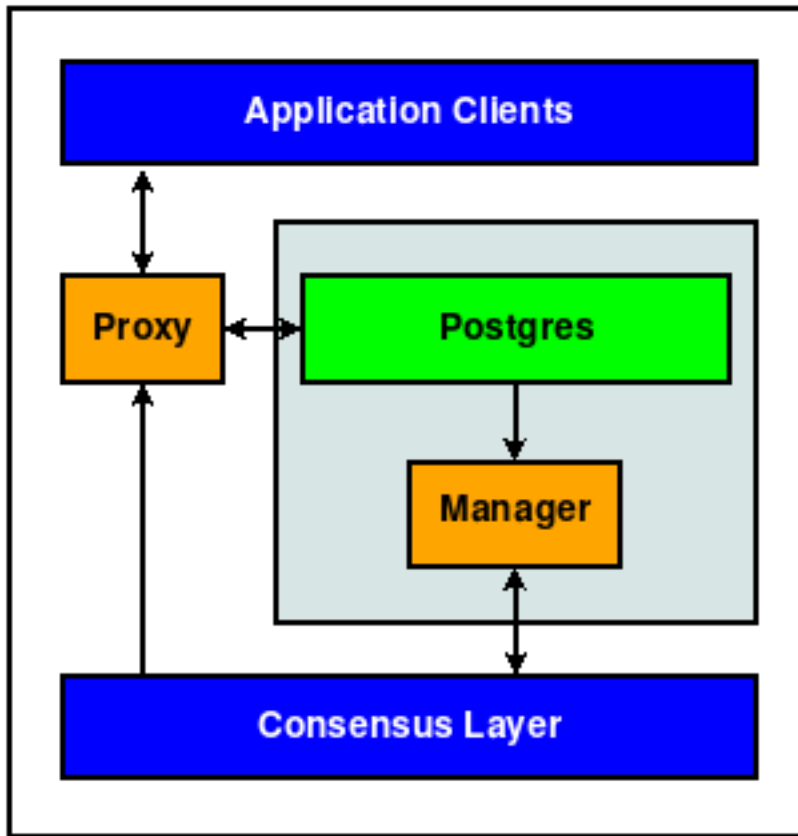


図 1: HARP Unit

コンセンサスレイヤーは、Harp Manager が割り当てられた Postgres ノードについて学習した情報を維持する外部エンティティであり、HARP プロキシはこの情報を有効な Postgres ノードターゲットに変換します。プロキシはコンセンサスレイヤーからノードターゲットを取得するため、このようなインスタンスがいくつか独立して存在する場合があります。

BDR 自分自身をコンセンサスレイヤーとして使用している間、各サーバーノードは代わりにこのバリエーションに似ています。

いずれの場合も、各ユニットは次の要素で構成されます。

- Postgres または EDB インスタンス
- Postgres のさまざまな属性を追跡するためのコンセンサスレイヤーリソースインスタンス
- Postgres ノードの状態を伝える HARP Manager プロセス コンセンサスレイヤー\*トラフィックを適切なリードマスターノードに転送する HARP プロキシサービスコンセンサスレイヤーから派生

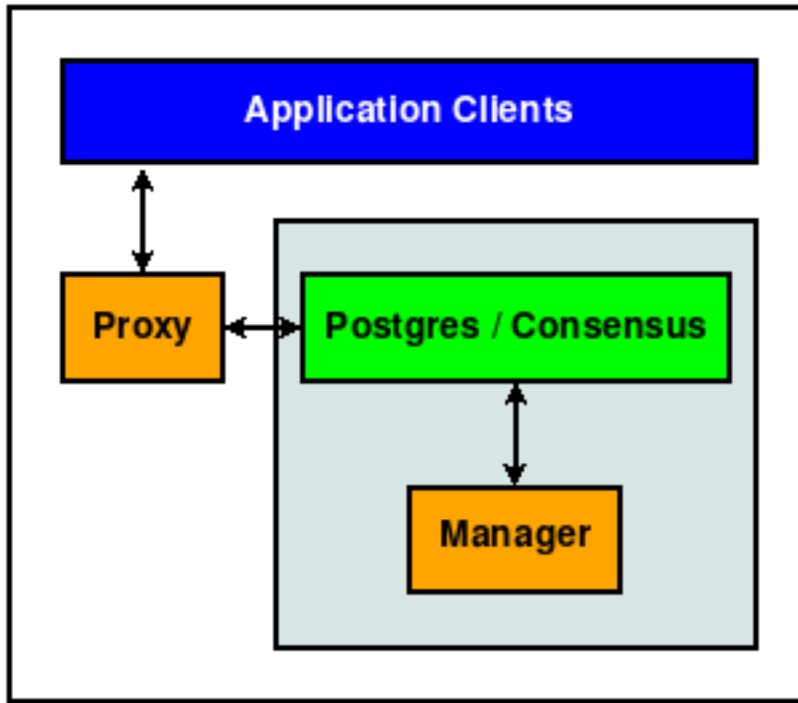


図 2: HARP Unit w/BDR Consensus

すべてのアプリケーションスタックがプロキシコンポーネント専用の追加ノードリソースにアクセスできるわけではないため、アプリケーションサーバーと組み合わせてスタック自分自身を簡素化できます。

これは、リードマスター/シャドウマスター構成で編成された単一のデータセンターで2つの BDR ノードを使用する一般的なデザインです。

BDR 自分自身を HARP コンセンサスレイヤーとして使用する場合、クォーラムマジョリティを保証には、少なくとも3つの完全修飾 BDR ノードが存在する必要があることに注意してください。

(上の図には、BDR ノード間の接続は示されていません。)

## 2.4 仕組み

BDR クラスターを管理する場合、HARP は定義されたロケーションごとに最大1つの「リーダー」ノードを維持します。正式には、これはリードマスターと呼ばれます。この位置を取るのに適格な他の BDR ノードは、それらがリーダーのロールを引き受けるときまでシャドウマスター状態です。

アプリケーションは、プロキシサービスを介してのみ現在のリーダーに連絡できます。コンセンサスレイヤーはリーダーの状態を伝える前にクォーラム契約を必要とするため、すべてのプロキシサービスがトラフィックをそのノードに転送します。

高レベルでは、これが最終的にマルチプルノードとのアプリケーションの相互作用を同時に妨げるものです。

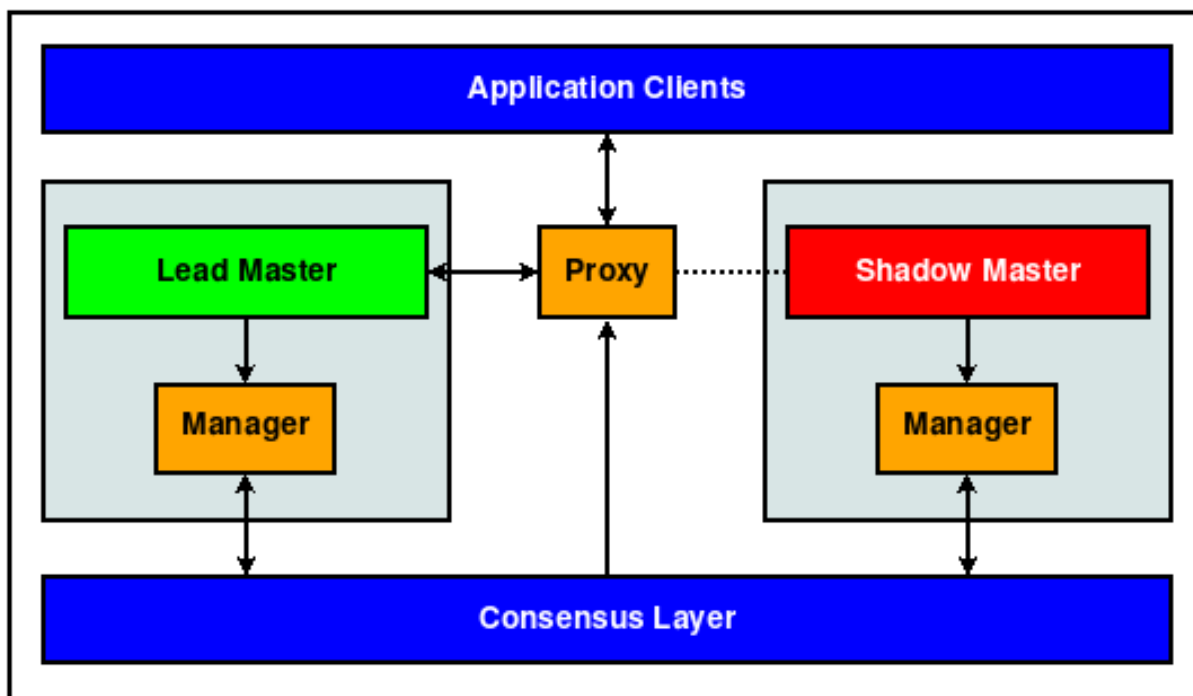


図 3: HARP Cluster

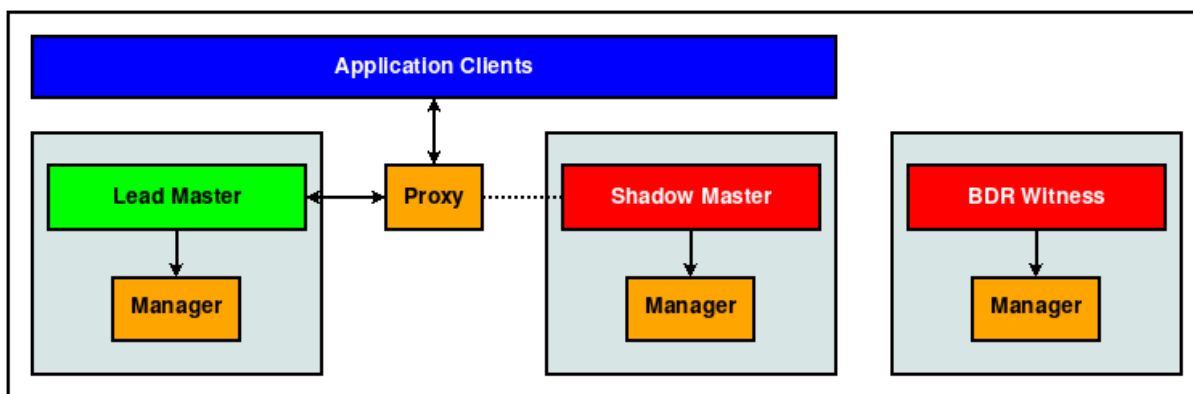


図 4: HARP Cluster w/BDR Consensus

## 2.4.1 リーダーの決定

例として、単一のデータセンター内に存在する可能性がある、ローカルに細分化された BDR Always-On グループ内のリードマスターのロールを検討します。Postgres または Manager リソースが開始され、構成可能なリフレッシュインターバルの後、以下が発生する必要があります。

1. Manager は、割り当てられた Postgres リソースのステータスを確認します。Postgres が実行されていない場合は、設定可能なタイムアウト後に再試行してください。Postgres が実行されている場合、続行します。
2. Manager は、コンセンサスレイヤーでリーダーリースのステータスを確認します。-リースが請求されていない場合、リースを取得し、この Manager に割り当てられた Postgres インスタンスの ID を割り当てます。このリース期間は設定可能ですが、長すぎると予期しないリーダーシップの移行が発生する可能性があります。-リースが既に請求されている場合は、リース TTL を更新します。-それ以外の場合は何もしません。

ここでは明らかにもっと多くのことが起こりますが、この単純化されたバージョンは何が起こっているのかを説明するはずです。リーダーリースは 1 つのノードでのみ保持でき、他の場所で保持されている場合、HARP マネージャーはあきらめて後で再試行します。

---

**注釈:** 選択したコンセンサスレイヤーに応じて、リーダーリースのステータスを確認するために繰り返しループするのではなく、HARP は代わりに通知をサブスクライブします。この場合、ポーリングではなく、リースの状態が変わるとすぐに応答できます。現在、この機能は etcd コンセンサスレイヤーに制限されています。

---

これは、HARP 自分自身が選挙を行わず、クォーラムを管理しないことを意味します。これはコンセンサスレイヤーに委任されます。リースを取得する行為は、コンセンサスレイヤーのクォーラムによって承認される必要があるため、要求が成功した場合、そのノードはそのロケーションのクラスターをリードします。

## 2.4.2 接続ルーティング

リードマスターのロールが確立されると、HARP プロキシに反映されるのと同様の確定的な結果で接続が処理されます。HARP プロキシが特定のバックエンドリソースの接続ターゲットを決定するニーズがある場合を考えます。

1. HARP プロキシは、構成された場所にある現在のリードマスターのコンセンサスレイヤーに問い合わせます。
2. これが未設定または移行中の場合; Postgres への新しいクライアント接続は禁止されていますが、クライアントは蓄積され、リードマスターが表示されるまで一時停止状態になります。-既存のクライアント接続は現在のトランザクションを完了することができ、新しい接続と同様の保留状態に戻ります。
3. クライアント接続がリードマスターに転送されます。

この場合に示される相互作用は、HARP Manager または Postgres との相互作用を必要としないことに注意してください。コンセンサスレイヤー自分自身は、プロキシの観点からのすべての真実のソースです。

### 2.4.3 コロケーション

ワークユニットの配置は、組織がこれらの原則に従うことを要求されるようなものです。

1. Manager と Postgres ユニットは、同じノード内に同時に存在する必要があります。
2. コンセンサスレイヤーの内容は、すべての運用ワークユニットの規範ロールを決定します。

これにより、クラスターフォラムの責任がコンセンサスレイヤー自身自身に委任されますが、HARP は重要なロールの割り当てとキー/値のストレージにそれを活用します。コンセンサスレイヤーが動作不能または到達不能である場合、ストレージも取得も成功せず、したがって、不正な Postgres ノードが接続を受け入れることを防ぎます。

その結果、コンセンサスレイヤーは、安全性を最大限に高めるために、一般に HARP または HARP 管理対象ノードの外部に存在する必要があります。リファレンス図は、このような分離を促進オーダーにこれを反映していますが、必須ではありません。

---

**注釈:** クラスター状態を操作および管理するオーダーに、BDR には Raft Consensus モデルの独自の実装が含まれています。HARP は、この同じレイヤーを活用して外部依存への依存を減らし、サーバーリソースを保持するように構成できます。ただし、このアプローチには特定の欠点があり、[コンセンサスレイヤー](#)のセクションでさらに詳しく説明します。

---

## 2.5 推奨されるアーキテクチャと使用

HARP は、主に 2 つ（またはそれ以上）のデータセンター内にあり、少なくとも 5 つの BDR ノードで構成される BDR Always-On アーキテクチャを表すように設計されました。これは、ロジカルスタンバイノードをカウントしません。

現在のおよび標準の表現形式は、次の図で見ることができます。

この図では、HARP Manager は BDR ノード 1~4 に存在します。クラスターの初期状態は、BDR ノード 1 が DC A のリードマスターであり、BDR ノード 3 が DC B のリードマスターであるということです。

これにより、DC A の HARP プロキシリソースが BDR ノード 1 に接続され、同様に DC B の HARP プロキシリソースが BDR ノード 3 に接続されます。

---

**注釈:** この図は DC ごとに 1 つの HARP プロキシのみを示していますが、これは単なる例示であり、単一障害点と見なされるべきではありません。任意の数の HARP プロキシノードが存在する場合があります、それらはすべてアプリケーショントラフィックを同じノードに向けます。

---

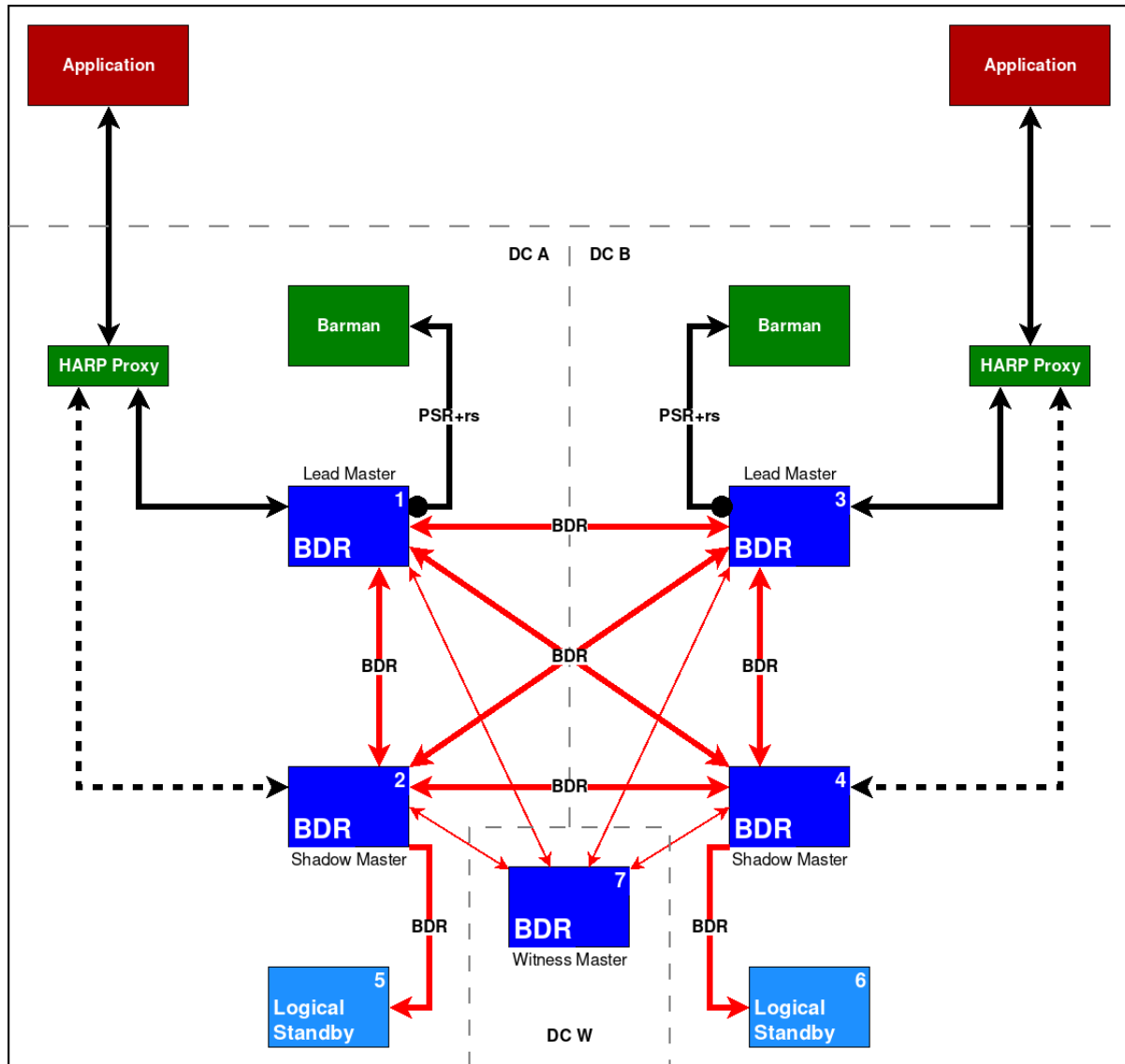


図 5: BDR Always-On Reference Architecture

### 2.5.1 ロケーション設定

マルチプルの BDR ノードがロケーションでリードマスターロックを取得できるようにオーダーには、`config.yml` 構成ファイル内でロケーションを定義する必要があります。

上記の図を再現するには、BDR ノード 1 および 2 の `config.yml` 構成に次の行を追加します。

```
location: dca
```

BDR ノード 3 および 4 の場合：

```
location: dcb
```

これは、それぞれのデータセンターで指定されている HARP プロキシノードにも適用されます。

### 2.5.2 BDR 3.7 互換性

BDR 3.7 以降では、BDR ノード自分自身にロケーションを割り当てることにより、より直接的なロケーション定義を提供します。これは、BDR ノードに接続しているときに次の SQL API ファンクションを呼び出すことで実行されます。したがって、BDR ノード 1 および 2 の場合、次のようにします。

```
SELECT bdr.set_node_location(dca);
```

BDR ノード 3 および 4 の場合：

```
SELECT bdr.set_node_location(dcb);
```

その後、HARP Manager の将来のバージョンは、BDR 自分自身から直接 `location` フィールドを取得します。この HARP 機能はまだ利用できないため、HARP がこの BDR API メソッドとの互換性を報告するまで、これと `config.yml` の設定を使用することをお勧めします。

## 第3章 Installation

HARP の標準インストールには、2つのシステムサービスが含まれます。

管理対象ノード上の *HARP マネージャー* (``harp-manager``) 他の場所での HARP プロキシ (`harp-proxy`)

通常、これらのサービスをインストールして構成し、適切な Quorum ベースの接続ルーティングのために Postgres を管理するには2つの方法があります。

### 3.1 ソフトウェアバージョン

HARP は外部ソフトウェアに依存しています。これらは、ここにリストされている最小バージョンに適合する必要があります。

| Software  | Min Ver |
|-----------|---------|
| etcd      | 3.4     |
| PgBouncer | 1.14    |

### 3.2 TPAExec

HARP をインストールして構成する最も簡単な方法は、クラスターの展開と管理に EDB の TPAexec ユーティリティを使用することです。このソフトウェアの詳細については、TPAexec product page を参照してください。

---

**注釈:** TPAExec は現在、BDR クラスター展開専用の EULA を通じてのみ利用可能です。上記の URL にアクセスできない場合は、詳細について販売担当者またはアカウント担当者にお問い合わせください。

---

TPA config.yml ファイルに次の属性が含まれていることを確認して、HARP を介してクラスタールーティングを管理自分自身必要があることを認識するように、TPAexec を構成する必要があります。

```
cluster_vars:
  failover_manager: harp
```

---

**注釈:** 21.1 より前のバージョンの TPAexec では、若干異なるアプローチが必要です。

---

```
cluster_vars:  
  enable_harp: true
```

この後、HARP は、クラスターを変更するための通常の `tpaexec` コマンドを呼び出すことによりインストールされます。

```
tpaexec provision ${CLUSTER_DIR}  
tpaexec deploy  ${CLUSTER_DIR}
```

クラスター固有の考慮事項がない限り、他の変更は必要ありません。

### 3.3 パッケージのインストール

現在、CentOS / RHEL パッケージソフトは、EDB パッケージインフラストラクチャを介して提供されています。詳細については、[HARP product page](#) を参照してください。

#### 3.3.1 etcd パッケージ

現在、多くの一般的な Linux ディストリビューションの `etcd` パッケージソフトは、標準のパブリックリポジトリからは入手できません。したがって、EDB は、RHEL および CentOS バージョン 7 および 8、Debian、および Ubuntu LTS などのバリエーション用に `etcd` をパッケージ化しました。繰り返しますが、これらのライブラリを使用するには、HARP パッケージリポジトリへのアクセスが必要です。

### 3.4 コンセンサスレイヤー

HARP が動作するには、分散コンセンサスレイヤーが必要です。現在、これは `bdr` または `etcd` でなければなりません。3 つ未満の BDR ノードを使用する場合、`etcd` に依存する必要がある場合があります。そうしないと、BDR サービスの停止によりコンセンサスレイヤーが単一ノードに縮小され、ノードコンセンサスが妨げられ、Postgres ルーティングが無効になります。

#### 3.4.1 etcd

`etcd` をコンセンサスレイヤーとして使用する場合、`etcd` は Postgres ノードに直接インストールするか、アクセス可能な別の場所にインストールする必要があります。

`etcd` をコンセンサスレイヤーとして設定するには、これを `HARP config.yml` 構成ファイルに含めます。

```
dcs:  
  driver: etcd  
  endpoints:  
    - host1:2379
```

(次のページに続く)

(前のページからの続き)

- host2:2379
- host3:2379

TPAExec を使用する場合、設定済みの etcd エンドポイントはすべてここに自動的に入力されます。

### 3.4.2 BDR

bdr ネイティブコンセンサスレイヤーは、BDR 3.6.21 および 3.7.3 から利用できます。このコンセンサスレイヤーモデルでは、BDR クラスターのルーティングを管理する際に補助ソフトウェアは必要ありません。

前述のように、クラスターでクォーラムを使用できるように保証には、ノードのメンテナンスまたは停止中に BDR のコンセンサスレイヤーが応答し続けるように、常に 3 つ以上のノードを使用します。

BDR をコンセンサスレイヤーとして設定するには、これを `config.yml` 構成ファイルに含めます。

```
dcg:  
  driver: bdr  
  endpoints:  
    - host=host1 dbname=bdrdb user=harp_user  
    - host=host2 dbname=bdrdb user=harp_user  
    - host=host3 dbname=bdrdb user=harp_user
```

ここに見られるように、BDR コンセンサスレイヤーのエンドポイントは、標準の Postgres DSN 接続フォーマットに従います。



## 第4章 Configuring HARP for Cluster Management

HARP 構成ファイルは、読みやすいように簡素化された標準の YAML スタイルのフォーマットに従います。このファイルは、デフォルトで/etc/harp ディレクトリにあり、`config.yml` 名前付けです。

`-f/--config` 引数を使用して、構成ファイルの場所をすべての HARP 実行可能ファイルに明示的に提供できます。

### 4.1 標準構成

HARP は基本的に 3 つのコンポーネントとして動作します。

- HARP マネージャー
- HARP プロキシ
- `harpctl`

これらはそれぞれ、同じ標準の `config.yml` 構成フォーマットを使用します。これには、常に次のセクションが含まれます。

- `cluster.name`-すべての操作の対象となるクラスターの名前。
- `dcs`-すべてのエンドポイントの DCS ドライバと接続構成。

基本的に、これは標準プリアンブルが常に HARP 操作に含まれ、これに似ていることを意味します。

```
cluster:  
  name: mycluster  
  
dcs:  
  ...
```

他のセクションは名前付けた HARP コンポーネントにオプションか特定であると考えられるべきです。

### 4.1.1 クラスター名

HARP との相互作用には、`cluster` 見出しの下での ``name`` エントリーが必要です。各 HARP クラスターには、明確化の目的と、特定のクラスターの DCS 内のデータのラベル付けの両方の名前があります。

HARP マネージャーは、HARP プロキシと `harpcctl` が使用するクラスターに関する情報をここに書き込みます。HARP プロキシサービスは、このクラスター内のノードにトラフィックを誘導します。`harpcctl` 管理ツールは、このクラスターと対話します。

### 4.1.2 DCS 設定

コンセンサスレイヤーの構成は、HARP 機能のキーとなります。DCS がなければ、HARP にはクラスターメタデータを格納する場所がなく、リーダーシップ選挙を行うことができません。したがって、構成のこの部分は必須ですが、特定の要素はオプション。

すべての要素は、`dc`s 名前付けのセクションの下に、ここで説明するマルチプルの補足エントリとともに指定する必要があります。

- ``driver`` : 使用する必須コンセンサスレイヤーのタイプ。現在は `etcd` または `bdr` になります。コンセンサスレイヤーとしての `bdr` のサポートは実験的。コンセンサスレイヤーとして `bdr` を使用すると、コンセンサスストレージ用の追加ソフトウェア、ただし最低 3 つが必要データベースのメンテナンス中にクォーラムを維持するための完全な BDR メンバノード。
- ``endpoints`` : DCS に接続するために必要な接続文字列のリスト。可能であれば、DCS のすべてのノードをここにリストする必要があります。これにより、HARP DCS の大部分がまだ残っている限りファンクションし続けますネットワーク経由で操作可能で到達可能。

コンセンサスレイヤーとして `etcd` を使用する場合の形式は次のとおりです。

```
dc:
  endpoints:
    - host1:2379
    - host2:2379
    - host3:2379
```

実験的な `bdr` コンセンサスレイヤーを使用する場合の形式は次のとおりです。

`edb_notranlate_3` 現在、`bdr` コンセンサスレイヤーでは、ローカルの `postgres` インスタンスを指す最初のエンドポイントが必要です。

- ``request\_timeout`` : リクエストが失敗したとみなす時間（ミリ秒）。HARP が DCS に要求を行い、この時間内に応答を受信しない場合期間は、オペレーションが失敗したと見なす必要があります。これにより問題が発生する可能性がありますリクエストの性質に応じて、エラーとして記録されるか、再試行されます。デフォルト：250。

次の DCS SSL 設定は、ドライバ：`etcd` が構成ファイルで設定されている場合にのみ適用されます。

- ``ssl`` : `on` または `off` のいずれかで、DCS との SSL 通信を有効にします。デフォルト：`off`
- ``ssl\_ca\_file`` : クライアント SSL 認証局（CA）ファイル。

- `ssl_cert_file` : クライアント SSL 証明書ファイル。
- `ssl_key_file` : クライアント SSL キーファイル。

### 4.1.3 例

次に、3つのノードで構成される etcd DCS に接続するように HARP を構成する方法の例を示します。

```

dcs:
  driver: etcd
  endpoints:
    - host1:2379
    - host2:2379
    - host3:2379

```

### 4.1.4 HARP マネージャー固有

すべての HARP コンポーネントに必要な汎用サービスオプションに加えて、Manager には少なくとも 1 つ以上の設定が必要です。

- `log_level` : DEBUG、INFO、WARNING、ERROR、または CRITICAL のいずれか HARP サービスからのログ出力の量を変更する可能性があります。
- `name` : このマネージャーが表す Postgres ノードの必須名前。Manager は特定のノードのみを表すことができるため、そのノードにはここで名前が名前付け、このマネージャーに名前を付ける役割も果たします。これが BDR ノードである場合、マッチする必要があります実行時にノード作成時に使用される値 `bdr.create_node(node_name, ...)` ファンクションおよび `bdr.local_node_summary.node_name` ビュー列。英数字およびアンダースコアのみ。
- `start_command` : これは、DCS の情報の代わりに使用できます監視するデータベースを起動します。これは、bdr を使用する場合に必要ですコンセンサスレイヤー。
- `status_command` : これは、DCS の情報の代わりに使用できます Harp Manager を使用して、データベースが実行されているかどうかを判断します。これはコンセンサスレイヤーとして bdr を使用する場合に必要です。
- `stop_command` : これは、DCS の情報の代わりに使用できますデータベースを停止します。

したがって、HARP Manager の完全な構成例は次のようになります。

```

cluster:
  name: mycluster

dcs:
  driver: etcd
  endpoints:
    - host1:2379

```

(次のページに続く)

- host2:2379
- host3:2379

**manager:**

**name:** node1  
**log\_level:** INFO

これは基本的に DCS 連絡先情報、関連するサービスのカスタマイズ、クラスター自分自身の名前、およびノードの名前であることに注意してください。他のすべての設定はノード自分自身に関連付けられ、DCS 内に保存されます。

特定のノード設定および HARP Manager で管理されるノードの初期化の詳細については、[ノードのブートストラップ](#) のセクションをお読みください。

### 4.1.5 HARP プロキシ固有

一部の構成オプションは、HARP プロキシに固有です。これらはデーモン自分自身の動作に影響を与えるため、現在 `config.yml` ファイル自分自身に配置されています。

プロキシベースの設定は `proxy` 見出しの下で指定でき、次のものが含まれます。

- `location` : ロケーション HARP プロキシの必須名前が表す必要があります。HARP プロキシノードは、次のように、実行されている場所に直接関連付けられます。それらは常にトラフィックを現在の Lead Master ノードに向けます。これは定義されたプロキシに指定されます。
- `log_level` : DEBUG、INFO、WARNING、ERROR、または CRITICAL のいずれか HARP サービスからのログ出力の量を変更する可能性があります。

\*デフォルト : INFO

- `name` : この特定のプロキシの名前。各プロキシノードには、関連する統計または動作を保証名前が名前付けます状態は、ステータスチェックおよびその他のインタラクティブイベントで使用できます。
- `type` : `pgbouncer` または実験的なビルトイン通過地点プロキシを使用するかどうかを指定します。すべてのプロキシは同じプロキシタイプを使用する必要があります。実験的な BDR DCS と組み合わせて、シンプルプロキシのみで実験することをお勧めします。 `pgbouncer` または `builtin` の場合があります。

\*デフォルト : `pgbouncer`

- `pgbouncer_bin_dir` : PgBouncer バイナリが置かれているディレクトリ。HARP は PgBouncer バイナリを利用するため、これらがどこにあるかを知るニーズがあります。これはプラットフォームまたはディストリビューションに依存する可能性があるため、デフォルト仮定は、適切なバイナリがそれ以外の場合は、環境の `PATH` 変数。

#### 4.1.6 例

HARP プロキシには、クラスター名前、DCS 接続設定、場所、およびオペレーション中のプロキシの名前が必要です。以下に例を示します。

```
cluster:
  name: mycluster

dcs:
  driver: etcd
  endpoints:
    - host1:2379
    - host2:2379
    - host3:2379

proxy:
  name: proxy1
  location: dc1
  pgbouncer_bin_dir: /usr/sbin
```

他のすべての属性は、プロキシのスタートアップに DCS から取得されます。

## 4.2 実行時ディレクティブ

config.yml ファイルで最小 YAML を使用して HARP Manager、HARP Proxy、または harpctl を構成することは可能ですが、一部のカスタマイズは DCS 内で保持され自分自身。これらの値は、ブートストラップで初期化するか、特に harpctl set 指示子で設定する必要があります。

このセクションでは、これらの概要と、それらの指定方法について説明します。

### 4.2.1 クラスター全体

ここでの設定は、ブートストラップ中に cluster YAML 見出しの下で設定するか、harpctl set cluster コマンドで変更する必要があります。

- `event_sync_interval`：同期を待機する時間（ミリ秒）。HARP 内でイベントが発生すると、クラスター全体で非同期的に発生します。HARP Manager は、メタデータの変更を検出するとすぐに動作をスタートし、また、HARP プロキシはトラフィックを一時停止し、エンドポイントの再構成をスタートする場合があります。これはの最大量にほぼ近似することを意図した安全インターバルすべての HARP コンポーネント間に存在する可能性があるイベント時間のスキュー。

例、ノード A がオフラインになり、ノード B の HARP マネージャーがノード C の 5 ミリ秒前にこのイベントを受信するとします。保証の HARP マネージャーサービスが処理を開始する前にイベントを受信するには、少なくとも 5 ミリ秒の設定が必要です。

これは、HARP プロキシにも適用されます。

## 4.2.2 ノードディレクティブ

ノード指向の設定のほとんどは、HARP マネージャーがアクティブなときに変更して適用できます。これらの項目は、初期ブートストラップ後に DCS に保持されるため、構成ファイルを変更せずに変更できます。

ここでの設定は、ブートストラップ中に `node` YAML 見出しの下で設定するか、`harpctl set node` コマンドで変更する必要があります。

- ```camo_enforcement``` : CAMO キューの状態を厳密に実施するかどうか。strict に設定すると、HARP は BDR へのスイッチオーバーまたはフェイルオーバーを許可しません。CAMO パートナーノードは、CAMO キュー全体に完全に追いついていない限り移行の時間。lag\_only に設定すると、標準ラグのみ maximum\_camo\_lag などのしきい値が適用されます。
- ```dcs_reconnect_interval``` : 切断されたノードが DCS への再接続を試行するインターバル（ミリ秒単位）。

\*デフォルト 1000

- ```dsn``` : 管理対象 Postgres ノードへの完全な接続文字列が必要です。このパラメータはすべての HARP サービスに等しく適用され、有効にしますコンテナごとに 1 つのサービスのみを実行するマイクロアーキテクチャ。

---

**注釈:** HARP はデフォルトで ```sslmode``` 引数を ```require``` に設定し、SSL を必要としないサーバーへの接続を防ぎます。この動作を無効にするには、このパラメータを ```disable```、```allow```、```prefer``` などのより許容度の高い値に明示的に設定します。

---

- ```db_data_dir``` : 必須の Postgres データディレクトリ。これは、HARP Manager が Postgres をスタート、停止、またはリロードするために必要です。サービスまた、構成ファイルのデフォルトの場所でもあります。ストリーミングレプリカのプロモーションを制御するために後で使用される。
- ```db_conf_dir``` : Postgres 構成ファイルの場所。一部のプラットフォームでは、Postgres 構成ファイルを Postgres データディレクトリ自分自身。これらの場合、これはそれに設定する必要があります予想される場所。
- ```db_log_file``` : Postgres ログファイルの場所。

\*デフォルト /tmp/pg\_ctl.out

- **イベント** : HARP が DCS に到達できない場合、いくつかの準備キーとリーダーシップリース自分自身がエキスパイアになります。これにより、ノードがルーティングを考慮することを暗黙的に防ぎます。ただし、そのようなノードは公式にフェンスされておらず、`stop_database_when_fenced` が `false` に設定されている場合、マネージャはデータベースのモニタリングを停止しません。

\*デフォルト : False

- ```leader_lease_duration``` : リードマスターの秒単位の時間リースは更新されない場合は持続します。これにより、HARP マネージャーは特定の有効期限が別のノードに許可される前に、ロックをリフレッシュする猶予期間代わりにリードマスターロックを取得します。

\*デフォルト : 30

- `lease_refresh_interval` : ミリ秒単位の時間リードマスターリースの更新。これは本質的に時間を制御します HARP Manager がその割り当てに対して実行する各シリーズのチェックの間 Postgres ノード、およびコンセンサスでノードのステータスが更新されたときレイヤー

\*デフォルト: 5000 - `max_dcs_failures` : `fence_node_on_dcs_failure` に従ってフェンスされたノードとしてマーキングする前の DCS 要求の失敗の量。これにより、一時的な通信の中断によるデータベースノードのシャットダウンが防止されます。

\*デフォルト: 10

- `maximum_lag` : 最後の最大許容分散 (バイト単位) 許可される前に、以前のリードマスターとこのノードの LSN を記録したリードマスターロックを取ります。これにより、ノードがターミナル的な量を経験することを防ぎますリードマスターロックの取得からの遅延。このチェックを無効にするには、-1 に設定します。

\*デフォルト: 1048576 (1MB)

- `maximum_camo_lag` : 最後の最大許容分散 (バイト単位) このノードとその CAMO パートナー間で LSN を受信し、LSN を適用しました。これは、CAMO が使用可能で有効になっているクラスターにのみ適用する必要があります。したがって、これは `pg2q.enable_camo` が設定されている BDR EE クラスターにのみ適用されます。特に厳しい CAMO 適用キュー制限のあるクラスターを設定する必要がありますこれは非常に低いか、適用されていない CAMO トランザクションを回避するために 0 までです。に設定このチェックを無効にするには-1。

\*デフォルト: 1048576 (1MB)

- `ready_status_duration` : ノードの準備ができていない時間 (秒) 更新されない場合、ステータスは持続します。これはフェイルセーフであり、HARP マネージャーが担当している場合、HARP プロキシがノードに連絡しない動作を停止します。

\*デフォルト: 30

- `db_bin_dir` : Postgres バイナリが置かれているディレクトリ。HARP は `pg_ctl` などの Postgres バイナリを利用するため、どこでそれを知るニーズがありますこれらがあります。これはプラットフォームまたはディストリビューションに依存する可能性があるため、デフォルト假定は、適切なバイナリがそれ以外の場合は、環境の `PATH` 変数。
- `priority` : 任意の数値。このオプションが-1 に設定されているノードは、`harpctl` を使用してリードマスターを明示的に設定しようとしても、リードマスターロールを取得できません。

\*デフォルト: 100

- `stop_database_when_fenced` : 考えられるすべてのルーティングからノードを単に削除するのではなく、ノードがフェンスされたときにデータベースを停止します。これは、HARP プロキシ以外のソースからのデータがデータベースに到達しないようにするための、またはプロキシが他の何らかの理由でクライアントを切断できない場合の追加の安全対策です。

\*デフォルト: False

- `consensus_timeout` : 読み取りを中止するまでのミリ秒数またはコンセンサスレイヤーに書き込みます。コンセンサスレイヤーが失われたイベントクォラムまたは到達不能になった場合、無限タイムアウト。これにより、このような場合のブロック動作が防止されます。注: `bdr` をコンセンサスレイヤーとして使用する場合、認識される最高のタイムアウト 1000ms です。

\*デフォルト：250

- ```use_unix_socket```：HARP Manager の使用を優先することを指定しますデータベースに接続するための UNIX ソケット。

\*デフォルト：False

これらの実行時指示子はすべて、`harpctl` を介して変更できます。`node1` で `lease_refresh_interval` を 100 ミリ秒に減らしたいかどうかを検討します。

```
harpctl set node node1 lease_refresh_interval=100
```

### 4.2.3 プロキシディレクティブ

サービスがアクティブなときにプロキシの特定の設定を変更できます。これらの項目は、初期ブートストラップ後に DCS に保持されるため、構成ファイルを変更せずに変更できます。これらの設定の多くは、同等の PgBouncer への直接マッピングであり、関連する場合はこれらにノートします。

ここでの設定は、ブートストラップ中に `proxies YAML` 見出しの下で設定するか、`harpctl set proxy` コマンドで変更する必要があります。`harpctl set proxy` を介して設定されたプロパティは、プロキシのリスタートが必要です。

- ```auth_file```：PgBouncer スタイルの `userlist.txt` ファイルへのフルパス。HARP プロキシはこのファイルを使用して、`pgbouncer` ユーザを保存します。PgBouncer の管理データベースへのアクセス。このファイルは他のユーザーに使用される場合があります同様に。プロキシはこのファイルを変更して、パスワードを追加および変更します `pgbouncer` ユーザ。

\*デフォルト `/etc/harp/userlist.txt`

- ```auth_type```：パスワードに使用する Postgres 認証の種類マッチングこれは実際には PgBouncer 設定であり、完全には互換性がありません Postgres `pg_hba.conf` 機能を使用します。`md5`、`pam` の使用をお勧めします `cert`、または `scram-sha-256`。

\*デフォルト `md5`

- ```auth_query```：Postgres でユーザーのパスワードを確認するクエリ。`pg_shadow` に直接アクセスするには、管理者権限が必要です。を使用することをお勧めします代わりに `SECURITY DEFINER` ファンクションを呼び出す非スーパーユーザー。使用する場合クラスターを作成する `TPAexec`、`pgbouncer_get_auth` 名前付けのファンクションはこれを満たすために、`pg_catalog` 名前空間内のすべてのデータベースにインストールされます目的。
- ```auth_user```：`auth_user` が設定されているユーザ、`auth_file` は、`pg_shadow` からの `auth_query` クエリーを介してクエリされます `auth_user` を使用したデータベース。`auth_user` のパスワードは `auth_file` から取得。
- ```client_tls_ca_file```：クライアントを検証するためのルート証明書ファイル 証明書。`client_tls_sslmode` を設定する必要があります。
- ```client_tls_cert_file```：プライベートキーの証明書。クライアントはできます それを検証します。`client_tls_sslmode` を設定する必要があります。

- `client_tls_key_file` : PgBouncer がクライアントを受け入れるための秘密キー 接続。  
`client_tls_sslmode` を設定する必要があります。
- `client_tls_protocols` : 使用できる TLS プロトコルバージョンクライアント接続。許可される値 :  
`tlsv1.0`、`tlsv1.1`、`tlsv1.2`、`tlsv1.3`。ショートカット : `all` (`tlsv1.0`、`tlsv1.1`、`tlsv1.2`、`tlsv1.3`)、  
`secure` (`tlsv1.2`、`tlsv1.3`)、`legacy` (すべて)。

\*デフォルト `secure`

- `client_tls_sslmode` : クライアント SSL 機能を有効にするかどうか。 `disable` `allow` `prefer` `require` `verify-ca` `verify-full` のいずれかです。

\*デフォルト `disable`

- `database_name` : データベースクライアントを表す必須の名前 HARP プロキシに接続するとき  
に使用します。これは安定 (stable) エンドポイントであり、変更せず、現在のノード、データベース  
名前、ポートなどを指します。リードマスターに接続するために必要です。グローバル値\*を使用で  
きますここで、データベースに関係なく、すべての接続がこのターゲットに向けられます名前
- `default_pool_size` : 許可するアクティブな接続の最大量データベース/ユーザの組み合わせごと。  
これはコネクションプーリングを目的としていますが、しかし、セッションプーリングモードでは何  
もしません。これは PgBouncer 設定です。

\*デフォルト 25

- `ignore_startup_parameters` : デフォルトでは、PgBouncer はスタートアップパケットで追跡できる  
パラメーター : `client_encoding`、`datestyle`、`timezone`、および `standard_conforming_strings`。  
他のすべてパラメータはエラーを発生させます。他のパラメーターを許可するには、PgBouncer が管  
理者によって処理されていることを認識できるように、ここで指定しますそれらは無視できます。多  
くの場合、これを設定する必要があります Java アプリケーションが正しくファンクションするため  
の `extra_float_digits`。

\*デフォルト `extra_float_digits`

- `listen_address` : プロキシが監視するする IP アドレス 接続。 `pgbouncer` および組み込みプロキシ  
によって使用されます。

\*デフォルト 0.0.0.0

- `listen_port` : プロキシが接続を監視するするシステムポート。 `pgbouncer` および組み込みプロキ  
シによって使用されます。

\*デフォルト 6432

- `max_client_conn` : アクティブなクライアントの合計最大量プロキシで許可される接続。これは、  
多くの注文になりますこれらはすべて `default_pool_size` より大きい接続であるため、セッション  
がまだ割り当てられていない、または使用するためにセッションをリリースしている別のクライアン  
ト接続。これは PgBouncer 設定です。

\*デフォルト 100

- `monitor_interval` : PgBouncer のプロキシチェック間の秒単位の時間。 HARP Proxy は実際の接  
続管理として PgBouncer を管理するためレイヤー、さまざまなステータスと統計を定期的にチェッ  
クして確認するニーズがありますまだ運用中です。この情報の一部はログに記録されるか、DCS に  
登録されています。

\*デフォルト 5

- `server_tls_protocols` : 使用できる TLS プロトコルバージョンサーバー接続。許可される値: `tlsv1.0`, `tlsv1.1`, `tlsv1.2`, `tlsv1.3`。ショートカット: `all` (`tlsv1.0`, `tlsv1.1`, `tlsv1.2`, `tlsv1.3`)、`secure` (`tlsv1.2`, `tlsv1.3`)、`legacy` (すべて)。

\*デフォルト `secure`

- `server_tls_sslmode` : サーバー SSL 機能を有効にするかどうか。 `disable` `allow` `prefer` `require` `verify-ca` `verify-full` のいずれかです。

\*デフォルト `disable`

- `session_transfer_mode` : セッションを転送する方法。 `fast` `wait` `reconnect` のいずれかです。

\*デフォルト `wait`

- `server_transfer_timeout` : Harp プロキシが PAUSE をあきらめて KILL コマンドを発行するまで待機する秒数。

\*デフォルト 30

次の 2 つのオプションは、ビルトインプロキシを使用する場合にのみ適用されます。

- `keepalive` : ビルトインプロキシがキープアライブメッセージをアイドルリーダー接続に送信するまで待機する秒数。

\*デフォルト 5

- `timeout` : リーダーへの接続を断念するまでにビルトインプロキシが待機する秒数。

\*デフォルト 1

`harpctl` を使用してすべてのプロキシのこれらの設定を変更する場合、プロキシ名前の代わりに `global` キーワードを使用します。例:

```
harpctl set proxy global max_client_conn=1000
```

## 第5章 Cluster Bootstrapping

HARP を使用オーダーには、最小限のメタデータが DCS に存在する必要があります。クラスターを「ブートストラップ」するプロセスとは、本質的に、ノード、場所、およびその他のランタイム構成を一度に、またはリソースごとに初期化することを意味します。

このプロセス全体は、`harpctl apply` コマンドによって管理されます。詳細については、`harpctl docs` を確認してください。

このセクションでは、DCS がセットアップされ機能していることを前提としています。

---

**重要:** このドキュメントの例は、セクションごとにブートストラップを行う必要があることを示していますが、そうではありません。これらのいずれかまたはすべてを 1 つの YAML ドキュメントに結合し、一度にすべて適用することができます。説明と簡略化のために、これらのセクションを単純に分割します。

---

### 5.1 クラスター全体のブートストラップ

一部の設定はクラスター全体に適用され、ブートストラップ中に指定できます。現在、これは `event_sync_interval` ランタイムディレクティブにのみ適用されますが、他のものは後で追加される可能性があります。

このフォーマットは次のとおりです。

```
cluster:
  name: mycluster
  event_sync_interval: 100
```

そのファイルの名前付けが `cluster.yml` であると仮定すると、次のように適用します。

```
harpctl apply cluster.yml
```

クラスター名前が DCS 内でまだ定義されていない場合、これもその値を初期化します。

---

**重要:** ここで指定されたクラスター名前パラメータは、`config.yml` で指定されたクラスター名前を常にオーバーライドします。前提は、ブートストラップファイルがクラスターまたはその大規模構成の一部をブートストラップするために必要なすべての要素を提供することです。`config.yml` ファイルの主な目的は、HARP マネージャー、HARP プロキシ、または `harpctl` の実行を具体的に制御することです。

---

## 5.2 場所のブートストラップ

すべての HARP ノードは、最大で 1 つのロケーションに関連付けられます。このロケーションは、単一のデータセンター、マルチプルの基礎となるサーバーで構成されるグループ化されたリージョン、Amazon アベイラビリティゾーンなどです。これは、HARP がノードをグループ化して、そのロケーションのノードをリードマスターとして表すことができるようにするロジカル構造にすぎません。

したがって、1 つ以上の場所を初期化する必要があります。このフォーマットは次のとおりです。

```
cluster:
  name: mycluster

locations:
  - location: dc1
  - location: dc2
```

そのファイルの名前付けが `locations.yml` であると仮定すると、次のように適用します。

```
harpctl apply locations.yml
```

クラスターの操作を実行するときは、名前をプリアンブルとして含めて、変更が適切な場所送到られるようにしてください。

ロケーションがブートストラップされると、簡単な検査で表示されます。

```
> harpctl get locations

Cluster   Location Leader Previous Leader Target Leader Lease Renewals
-----
mycluster dc1                                <nil>
mycluster dc2                                <nil>
```

ここに見られるように、両方の場所は HARP によって認識され、ノードとプロキシの割り当てに使用できます。

## 5.3 ノードのブートストラップ

HARP ノードは名前付けクラスター内に存在し、指定された名前を持つ必要があります。これを超えて、他のすべての設定は動的であり、HARP がそれらと対話する方法に影響を与える可能性があるため、DCS 自分自身内に保持されます。このため、各ノードは、[設定](#) の文書で説明されている 1 つ以上のランタイム指示子を使用してブートストラップする必要があります。

ノードのブートストラップ中に、いくつかの必須フィールドがあります。

- name
- location
- dsn

- `pg_data_dir`

その他はすべてオプションであり、クラスター自身自身に依存する場合があります。一度にマルチプルのノードをブートストラップすることが可能であるため、一般的にこのフォーマットは次の構造に適合します。

```
cluster:
  name: mycluster

nodes:
  - name: node1
    location: dc1
    dsn: host=node1 dbname=bdrdb user=postgres
    pg_data_dir: /db/pgdata
    leader_lease_duration: 10
    priority: 500
```

そのファイルの名前付けが `node1.yml` であると仮定すると、次のように適用します。

```
harpctl apply node1.yml
```

ノードがブートストラップされると、簡単な検査で表示されます。

```
> harpctl get nodes
```

| Cluster         | Name  | Location | Ready | Fenced | Allow | Routing | Routing | Status | Role    | Type   |
|-----------------|-------|----------|-------|--------|-------|---------|---------|--------|---------|--------|
| → Lock Duration |       |          |       |        |       |         |         |        |         |        |
| -----           |       |          |       |        |       |         |         |        |         |        |
| → -----         |       |          |       |        |       |         |         |        |         |        |
| mycluster       | bdra1 | dc1      | true  | false  | true  |         | ok      |        | primary | bdr 30 |

## 5.4 プロキシブートストラップ

ロケーションやノードとは異なり、プロキシは、ロケーション内のすべてのプロキシに適用される構成テンプレートを提供することもできます。これらは、`global` の指定で DCS に保存されます。各プロキシにはインスタンスとして存在する名前も必要ですが、特定の設定が特定のオーバーライドをニーズとしない限り、それ以上のカスタマイズは必要ありません。

これは、クラスターのデフォルトの構成設定が同じであるプロキシがマルチプル存在する可能性が高いため、すべてのプロキシに対してこれらの値を繰り返す必要はないためです。

さらに、プロキシテンプレートをブートストラップする場合、接続の割り当て用に少なくとも 1 つのデータベースを定義する必要があります。これらの注意事項を念頭に置いて、このフォーマットは次のとおりです。

```
cluster:
  name: mycluster
```

(次のページに続く)

(前のページからの続き)

```
proxies:
  monitor_interval: 5
  default_pool_size: 20
  max_client_conn: 1000
  database_name: bdrdb
  instances:
    - name: proxy1
    - name: proxy2
      default_pool_size: 50
```

これにより、2 つのプロキシ proxy1 と proxy2 の HARP が構成され、proxy2 のみがカスタム default\_pool\_size を持ち、それ以外の場合はグローバル設定を使用します。

そのファイルの名前付けが proxy.yml であると仮定すると、次のように適用します。

```
harpctl apply proxy.yml
```

プロキシテンプレートがブートストラップされると、簡単な検査で表示されます。

```
> harpctl get proxies
```

| Cluster   | Name   | Pool Mode | Auth Type | Max Client Conn | Default Pool Size |
|-----------|--------|-----------|-----------|-----------------|-------------------|
| mycluster | global | session   | md5       | 1000            | 20                |
| mycluster | proxy1 | session   | md5       | 1000            | 20                |
| mycluster | proxy2 | session   | md5       | 1000            | 50                |

## 第6章 HARP Manager

HARP Manager は、ローカルの PostgreSQL / BDR ノードと対話し、その状態に関する情報をコンセンサスレイヤーに保存するデーモンです。マネージャーは、各ノードのリーダーステータスを現在保持しているノードを特定し、構成（ラグ、CAMO ラグなど）のコンストレインを適用して、資格のないノードがリーダーの検討対象とならないようにします。

クラスター内のすべての Postgres ノードには、関連する HARP マネージャーが必要です。他のノードが存在する場合がありますが、リードまたはシャドウマスターの役割、または将来的に HARP がサポートする他の機能として参加することはできません。

### 6.1 仕組み

スタート時に、HARP Manager は `pg_ctl` を使用して Postgres がまだ実行されていない場合は起動します。この後、定期的に `node.lease_refresh_interval` 設定で定義されたサーバーをチェックします。HARP Manager は、以下を含む Postgres に関するさまざまなデータを収集します。

**ノードの現在の LSN** Postgres が実行され、接続を受け入れている場合。この特定のデータポイント定期的に更新する必要があるリースと見なされます。有効期限が切れた場合、HARP プロキシは、既存のルーティングからノードを削除します。**すべてのアップストリーム BDR 同等な現在の適用 LSN 位置。** CAMO が有効になっている場合： - CAMO パートナーの名前 - ピア CAMO 状態 (`is_ready`) - LMO 位置を受信および適用した CAMO キューノードが **BDR か通常の Postgres かなどのノードタイプ**。ノードの現在のロール（読み取り/書き込み、物理的ストリーミングレプリカなど）ロジカルスタンバイなど。\* BDR ノードの状態。限られた場合を除き、ACTIVE である必要があります。**他のメタデータ収集用の BDR ノード ID。** その他の追跡値。

---

**重要:** HARP 内の BDR ノードに名前を付ける場合、BDR ノード名前は ``node.name`` 構成属性で表されるノード名前とマッチする必要があります。これは、ブートストラッププロセスで既に行われている必要があります。

---

ここで収集されたデータは、他の HARP Manager プロセスで完全に利用でき、遅延、パートナーの準備状況、およびスイッチオーバーとフェイルオーバーの動作を指示するその他の基準を評価するために使用されます。

ノードメタデータを更新した後、HARP マネージャーは、ローカルノードが既に保持している場合はリードマスターリースをリフレッシュするか、リースの期限が切れている場合はリースを取得シークとします。すべてのノードの現在の状態は他のすべてのノードに既知であるため、前のリードマスターであったノードには、存在する場合は自動優先ランキングが与えられます。そうでない場合、他のすべてのノードは、LSN ラグ、ノードの優先度、およびその他の基準ごとにリストされ、最も適格なノードがリードマスターリースを捕捉します。

このプロシージャは、ノードが存在するすべての定義済みの場所で発生します。したがって、ロケーション DC1 および DC2 の場合、それぞれにリードマスターノードがあり、両方に個別のリースおよび選択プロセスがあります。

HARP マネージャーは、これらの Postgres ステータス保証、リース更新、および選択を繰り返して、クラスターが HARP プロキシからの接続に対して常にリードマスターターゲットを持つようにします。

## 6.2 設定

HARP Manager は、`dc`、`cluster`、および `manager` 構成スタンザを想定しています。以下は機能的な例です。

```
cluster:
  name: mycluster

dcs:
  driver: etcd
  endpoints:
    - host1:2379
    - host2:2379
    - host3:2379

manager:
  name: node1
  postgres_bin_dir: /usr/lib/postgresql/13/bin
```

構成ファイルへの変更（デフォルト：`/etc/harp/config.yml`）は、実行中のインスタンスに `SIGHUP` を発行するか、サービスレベルのリロードを呼び出すことで適用できます。

詳細については、[設定](#) を参照してください。

## 6.3 使用法

これは、HARP Manager の基本的な使用法です。

```
Usage of ./harp-manager:
  -f string
      Optional path to config file (shorthand)
  --config string
      Optional path to config file
```

フォークされたデーモンとして `harp-manager` を起動する引数がないことに注意してください。このソフトウェアは、`systemd` を介して、または最上位プロセスとしてコンテナ内で起動するように設計されています。これは、`journald` または接続されたコンテナターミナルを介したキャプチャおよびアクセスのために、出力が `STDOUT` および `STDERR` に向けられることも意味します。

## 6.4 Postgres の HARP Manager 制御の無効化と再有効化

Postgres の HARP Manager 制御を一時的に一時停止することが可能です。これにより、デーモンの実行は継続されますが、クラスターの既存の動作に影響を与える可能性のある操作は実行されません。管理を再度有効にすると、オペレーションが再開されます。

ノード管理を一時的に無効にする例は次のとおりです。

```
harpctl unmanage node node1
```

詳細については、[harpctl Command-line Tool](#) の文書を参照してください。

HARP Manager によるノード管理はデフォルトで有効になっています。



## 第7章 HARP Proxy

HARP プロキシは、クライアントアプリケーションと Postgres 間の抽象化レイヤーとして機能するデーモンです。コンセンサスレイヤーとインターフェイスして、現在のリードマスターノードの ID を取得し、その場所にトラフィックを転送します。計画的なスイッチオーバーまたは計画外のフェイルオーバーが発生したイベント、DCS の指示に従って、新しいリードマスターノードに自動的にリダイレクトされます。

HARP プロキシ用に pgbouncer またはビルトインを選択できます。pgbouncer を使用する場合、HARP プロキシは DCS と PgBouncer 間のインタフェースレイヤーです。そのオーダー、HARP プロキシがアクティビティを完全に管理するには、PgBouncer が前提条件であり、さらにインストールする必要があります。

組み込みプロキシには、追加のソフトウェアは必要ありません。組み込みを使用する場合、HARP プロキシはレベル 4 パススループロキシとして機能します。

### 7.1 仕組み

HARP プロキシは、起動時に PgBouncer をまだ実行していない場合は起動し、クライアント接続を一時停止状態のままにします。その後、DCS に接続してリードマスターの ID を特定し、これをデータベース接続のターゲットとして使用するよう PgBouncer を構成し、接続アクティビティを再開します。すべてのアプリケーションクライアントトラフィックは、PgBouncer を通過して、このプロキシが動作している場所の現在のリードマスターノードに到達します。

PgBouncer の実行中、HARP プロキシは DCS 内の `monitor_interval` 構成設定に基づいてそのステータスを確認し、モニタリングのために DCS に保存します。これにより、`harpctl` を使用して、構成されているすべてのプロキシ、または特定のプロキシのステータスを取得できます。

Lead Master リースが設定されていないイベント、HARP Proxy は、新しい Lead Master が確立されるまで、すべての接続トラフィックを一時停止します。これは、`harpctl promote` を使用して、新しいリードマスターへの計画的な移行を呼び出す場合にも当てはまります。このために PgBouncer PAUSE コマンドを使用するため、既存のセッションは、停滞する前に保留中のトランザクションを完了できます。

### 7.2 設定

HARP プロキシは、`dc`s、`cluster`、および `proxy` 構成スタンザを想定しています。以下は機能的な例です。

```
cluster:
  name: mycluster

dcs:
```

(次のページに続く)

```
driver: etcd
endpoints:
  - host1:2379
  - host2:2379
  - host3:2379

proxy:
  name: proxy1
```

## 7.3 使用法

これは、HARP プロキシの基本的な使用法です。

```
Usage of ./harp-proxy:
  -f string
      Optional path to config file (shorthand)
  --config string
      Optional path to config file
```

フォークされたデーモンとして `harp-proxy` を起動する引数がないことに注意してください。このソフトウェアは、`systemd` を介して、または最上位プロセスとしてコンテナ内で起動するように設計されています。これは、`journald` または接続されたコンテナターミナルを介したキャプチャおよびアクセスのために、出力が `STDOUT` および `STDERR` に向けられることも意味します。

## 7.4 PgBouncer 構成ファイル

HARP プロキシは現在、接続管理とリダイレクションに PgBouncer を使用しているため、`pgbouncer.ini` ファイルが存在する必要があります。HARP Manager は、[プロキシディレクティブ](#) の文書で定義されているさまざまな実行時指示子に基づいてこのファイルを構築します。

このファイルは、HARP プロキシが使用する `config.yml` と同じフォルダーに配置されます。HARP プロキシによって起動された PgBouncer プロセスは、この構成ファイルを使用し、デバッグまたは情報提供の目的で使用できます。この自動生成された `pgbouncer.ini` ファイルへの変更は、HARP プロキシが再起動されるたびに失われるため、代わりに `harpctl set proxy` を使用してこれらの設定を変更します。`harpctl set proxy` を呼び出しても、プロキシが再起動されるまで `pgbouncer.ini` ファイルは更新されません。

## 7.5 HARP プロキシノード管理の無効化と再有効化

PgBouncer の HARP プロキシ制御を一時的に一時停止することができます。これにより、デーモンの実行は継続されますが、クラスターの既存の動作に影響を与える可能性のある操作は実行されません。管理を再度有効にすると、オペレーションが再開されます。

特定のプロキシの管理を一時的に無効にする例は次のとおりです。

```
harpctl unmanage proxy proxy1
```

詳細については、[harpctl Command-line Tool](#) の文書を参照してください。

プロキシノード管理はデフォルトで有効になっています。

## 7.6 パススルーユーザー認証

`auth_user` および `auth_query` ランタイム指示子を使用するように HARP プロキシを構成することを強くお勧めします。これらが設定されていない場合、PgBouncer `userlist.txt` ファイルには、PgBouncer が Postgres に代わって認証するニーズがあるすべてのユーザのユーザー名とパスワードのハッシュの組み合わせを含める必要があります。

これは、基になる PgBouncer サービスを操作するオーダーに管理レベルのユーザとして HARP プロキシによって利用されるため、`pgbouncer` ユーザ自分自身ではないはずです。

TPAexec によって管理されるクラスターでは、プロビジョニング中にファンクションが作成され、`template1` データベースの `pg_catalog` スキーマにインストールされます。これは、その後に作成されるデータベースにもファンクションが含まれ、ユーザがどのデータベースに接続しようとしているかに関係なく、PgBouncer で利用できることを意味します。

TPAexec を使用しない場合でも、次のファンクション定義をお勧めします。

```
CREATE OR REPLACE FUNCTION pg_catalog.pgbouncer_get_auth(p_username TEXT)
RETURNS TABLE(username TEXT, password TEXT) AS $$
BEGIN
    RETURN QUERY
    SELECT username::TEXT, passwd::TEXT FROM pg_catalog.pg_shadow
    WHERE username = p_username;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER

REVOKE ALL ON FUNCTION pg_catalog.pgbouncer_get_auth(p_username TEXT)
FROM PUBLIC

GRANT EXECUTE ON FUNCTION pg_catalog.pgbouncer_get_auth(p_username TEXT)
TO <auth_user>;
```

HARP プロキシに提供される `auth_user` フィールドを `<auth_user>` に置き換えることを忘れないでください。

次に、Bootstrap ファイルで、次の設定を完了します。

```
cluster:
  name: mycluster

proxies:
  monitor_interval: 5
  default_pool_size: 20
  max_client_conn: 1000
  auth_user: pgb_auth
  auth_query: "SELECT * FROM pg_catalog.pg_bouncer_get_auth($1)"
  database_name: bdrdb
  instances:
    - name: proxy1
    - name: proxy2
```

harpcctl set proxy を使用してこれらのフィールドを定義することもできます。

```
harpcctl set proxy global auth_user=pgb_auth
```

---

**注釈:** これは、HARP プロキシを起動する `postgres` または `enterprisedb` OS ユーザーが、`auth_user` が *Postgres* に対して認証できるように、`pgpass` ファイルが必要であることを意味します。

---

## 7.7 仕組み

HARP プロキシは、起動時に、プロキシインスタンスごとにブートストラップファイルで指定されたリスニングアドレスとリスニングポートで着信接続を監視します。すべてのアプリケーションクライアントトラフィックは、ビルトインプロキシを介して、このプロキシが動作している場所の現在のリードマスターノードに渡されます。

Lead Master リースが設定されていないイベント、HARP Proxy は、新しい Lead Master が確立されるまで、すべての接続トラフィックを切断します。これは、`harpcctl promote` を使用して、新しいリードマスターへの計画的な移行を呼び出す場合にも当てはまります。すぐに切断されます。

## 7.8 設定

HARP プロキシは、`dcs`、`cluster`、および `proxy` 構成スタanzas を想定しています。以下は機能的な例です。

```
cluster:
  name: mycluster

dcs:
```

(次のページに続く)

(前のページからの続き)

```
driver: etcd
endpoints:
  - host1:2379
  - host2:2379
  - host3:2379

proxy:
  name: proxy1
```

## 7.9 使用法

これは、HARP プロキシの基本的な使用法です。

```
Usage of ./harp-proxy:
-f string
    Optional path to config file (shorthand)
--config string
    Optional path to config file
```

フォークされたデーモンとして **harp-proxy** を起動する引数がないことに注意してください。このソフトウェアは、systemd を介して、または最上位プロセスとしてコンテナ内で起動するように設計されています。これは、journald または接続されたコンテナターミナルを介したキャプチャおよびアクセスのために、出力が STDOUT および STDERR に向けられることも意味します。



## 第8章 harpctl Command-line Tool

harpctl は、コンセンサスレイヤーのコンテンツを直接操作して、目的のクラスタージオメトリに合わせるためのコマンドラインツール。たとえば、ノードのステータスを調べたり、ノードを Lead Master に「プロモート」したり、クラスター管理を無効/有効にしたり、クラスター設定をブートストラップしたりすることができます。

### 8.1 概要

```
$ harpctl --help

Usage:
  harpctl [command]

Available Commands:
  apply      Command to set up a cluster
  completion generate the autocompletion script for the specified shell
  fence      Fence specified node
  get        Command to get resources
  help       Help about any command
  manage     Manage Cluster
  promote    Promotes the sepcified node to be primary,
  set        Command to set resources.
  unfence    unfence specified node
  unmanage   Unmanage Cluster
  version    Command to get version information

Flags:
  -c, --cluster string      name of cluster.
  -f, --config-file string  config file (default is /etc/harp/config.yml)
      --dcs stringArray     Address of dcs endpoints. ie: 127.0.0.1:2379
  -h, --help                help for harpctl
  -o, --output string       json, yaml.
  -t, --toggle              Help message for toggle

Use "harpctl [command] --help" for more information about a command.
```

この基本的な概要に加えて、使用可能な各コマンドには、独自のシリーズの使用可能なサブコマンドとフラグがあります。

## 8.2 設定

harpctl が動作するには、コンセンサスレイヤーと対話する必要があることに注意することが重要です。つまり、正常に実行するには、`config.yml` に特定の最小量の設定を定義する必要があります。これには以下が含まれます：

- `dc.driver`
- `dc.endpoints`
- `cluster.name`

etcd を使用した例：

```
cluster:
  name: mycluster
dc:
  driver: etcd
  endpoints:
    - host1:2379
    - host2:2379
    - host3:2379
```

詳細については、[設定](#) を参照してください。

## 8.3 実行

次のように harpctl を実行します。

```
harpctl command [flags]
```

各コマンドには、独自のシリーズのサブコマンドとフラグがあります。次の方法でコマンドを実行すると、これらのヘルプが利用できます。

```
harpctl command --help
```

## 8.4 harpctl apply

`apply` コマンドを使用して、目的のクラスターのさまざまな属性を定義するファイルを使用して HARP クラスターを「ブートストラップ」する必要があります。

`apply` コマンドは次のように実行する必要があります。

```
harpctl apply <filename>
```

これにより、基本的にすべての初期クラスターメタデータ、デフォルトまたはカスタム管理設定などが作成されます。これは、DCS がクラスターを管理するための究極の真実のソースとして使用されるため、ここで行われます。これにより、これらの設定を動的に変更できるようになります。

これは、クラスター全体をブートストラップするために 1 回、オブジェクトのタイプごとに 1 回、または単純化のためにノードごとに 1 回行うことができます。

これは、単一ノードのブートストラップファイルの例です。

```
cluster:
  name: mycluster
nodes:
  - name: firstnode
    dsn: host=host1 dbname=bdrdb user=harp_user
    location: dc1
    priority: 100
    db_data_dir: /db/pgdata
```

ここで見られるように、複数の環境が同じ環境で動作している場合、すべての変更が正しい HARP クラスターをターゲットと保証に、常にクラスター名前のプリアンブルを含めることをお勧めします。

`apply` がエラーなしで完了すると、ノードはクラスターの残りの部分と統合されます。

---

**注釈:** 定義されたすべてのセクションが同時に適用されるため、このコマンドを使用してクラスター全体を一度にブートストラップすることもできます。ただし、初期定義時にクラスターの各部分を検証するのが難しくなるため、テスト以外の用途にはこの使用をお勧めしません。

---

## 8.5 harpctl fence

ローカルまたは指定されたノードを **fenced** としてマークします。このステータスのノードは、本質的にクラスターから完全に除外されます。HARP プロキシはトラフィックを送信せず、その代表的な HARP マネージャーはリードマスターリースを要求せず、さらなる手順も実行されます。実行中の場合、HARP Manager はノード上の Postgres も停止します。

`fence` コマンドは次のように実行する必要があります。

```
harpctl fence (<node-name>)
```

ノード名自分自身はオプション。省略すると、`harpctl` はローカルに構成されたノードの名前を使用します。

## 8.6 harpctl get

クラスターのさまざまな要素のコンセンサスレイヤーに格納されている情報を取得します。これには、ノード、場所、クラスター自身などが含まれます。完全なリストには以下が含まれます。

- **cluster**-クラスターの状態を返します
- **leader**-現在または指定された位置リーダーを返します
- **location**-現在または指定された位置情報を返す
- **locations**-すべての場所のリストを返します
- **node**-指定された Postgres ノードを返します
- **nodes**-すべての Postgres ノードのリストを返します
- **proxy**-現在または指定されたプロキシ情報を返します
- **proxies**-すべてのプロキシノードのリストを返します

### 8.6.1 harpctl get cluster

現在のクラスターのコンセンサスレイヤーに保存されている情報を取得します。

```
> harpctl get cluster
```

| Name      | Enabled |
|-----------|---------|
| ----      | -----   |
| mycluster | true    |

### 8.6.2 harpctl get leader

指定した場所の DCS に保存されている現在のリードマスターのノード情報を取得します。ロケーションが渡されない場合、harpctl は、実行された現在のノードのロケーションに基づいて、それを導き出そうとします。

例：

```
> harpctl get leader dc1
```

| Cluster   | Name   | Ready | Role    | Type  | Location | Fenced | Lock  | Duration |
|-----------|--------|-------|---------|-------|----------|--------|-------|----------|
| -----     | ----   | ----- | -----   | ----- | -----    | -----  | ----- | -----    |
| mycluster | mynode | true  | primary | bdr   | dc1      | false  | 30    |          |

### 8.6.3 harpctl get location

指定された場所の場所情報を取得します。ロケーションが渡されない場合、harpctl は、実行された現在のノードのロケーションに基づいて、それを導き出そうとします。

例：

```
> harpctl get location dc1
```

| Cluster   | Location | Leader | Previous Leader | Target Leader | Lease | Renewals |
|-----------|----------|--------|-----------------|---------------|-------|----------|
| mycluster | dc1      | mynode | mynode          |               |       | <nil>    |

### 8.6.4 harpctl get locations

DCS に現在存在するすべての場所の情報を取得します。

例：

```
> harpctl get locations
```

| Cluster   | Location | Leader   | Previous Leader | Target Leader | Lease | Renewals |
|-----------|----------|----------|-----------------|---------------|-------|----------|
| mycluster | dc1      | mynode   | mynode          |               |       | <nil>    |
| mycluster | dc2      | thatnode | thatnode        |               |       | <nil>    |

### 8.6.5 harpctl get node

指定されたノードの DCS に格納されているノード情報を取得します。

例：

```
> harpctl get node mynode
```

| Cluster   | Name   | Location | Ready | Fenced | Allow Routing | Routing | Status | Role    | Type   |
|-----------|--------|----------|-------|--------|---------------|---------|--------|---------|--------|
| mycluster | mynode | dc1      | true  | false  | true          | ok      |        | primary | bdr 30 |

### 8.6.6 harpctl get nodes

クラスター内のすべてのノードの DCS に格納されているノード情報を取得します。

例：

```
> harpctl get nodes
```

| Cluster    | Name  | Location | Ready | Fenced | Allow Routing | Routing | Routing Status | Role    | Type | Lock |
|------------|-------|----------|-------|--------|---------------|---------|----------------|---------|------|------|
| ↪Duration  |       |          |       |        |               |         |                |         |      |      |
| -----      |       |          |       |        |               |         |                |         |      |      |
| ↪-----     |       |          |       |        |               |         |                |         |      |      |
| myclusters | bdra1 | dc1      | true  | false  | true          |         | ok             | primary | bdr  | 30   |
| myclusters | bdra2 | dc1      | true  | false  | false         |         | N/A            | primary | bdr  | 30   |
| myclusters | bdra3 | dc1      | true  | false  | false         |         | N/A            | primary | bdr  | 30   |

### 8.6.7 harpctl get proxy

指定したプロキシの DCS に保存されているプロキシ情報を取得します。このクラスターのプロキシのデフォルトを表示するには、`global` を指定します。

例：

```
> harpctl get proxy proxy1
```

| Cluster   | Name   | Pool Mode | Auth Type | Max Client Conn | Default Pool Size |
|-----------|--------|-----------|-----------|-----------------|-------------------|
| -----     |        |           |           |                 |                   |
| mycluster | proxy1 | session   | md5       | 1000            | 20                |

### 8.6.8 harpctl get proxies

クラスター内のすべてのプロキシの DCS に格納されているプロキシ情報を取得します。さらに、デフォルトのプロキシ設定の `global` 擬似プロキシをリストします。

例：

```
> harpctl get proxies
```

| Cluster   | Name   | Pool Mode | Auth Type | Max Client Conn | Default Pool Size |
|-----------|--------|-----------|-----------|-----------------|-------------------|
| -----     |        |           |           |                 |                   |
| mycluster | global | session   | md5       | 500             | 25                |
| mycluster | proxy1 | session   | md5       | 1000            | 20                |
| mycluster | proxy2 | session   | md5       | 1500            | 30                |

## 8.7 harpctl manage

クラスターが管理状態にないイベント、Postgres のモニタリングとコンセンサスレイヤーの更新を再開するようにすべての HARP Manager サービスに指示します。これは、HARP ソフトウェアの更新またはクラスター全体に影響を与える可能性のあるその他の重要な変更が続いて、メンテナンスが完了した後に行う必要があります。

manage コマンドは次のように実行する必要があります。

```
harpctl manage cluster
```

**注釈:** 現在、`cluster` レベルでのみクラスター管理を有効または無効にできます。それ以降のバージョンでは、個々のノードまたはプロキシに対してこれを `make` ことも可能になります。

## 8.8 harpctl promote

リーダーシップ要件を満たす次の利用可能なノードを現在の場所のリードマスターに昇格させます。これは要求されたイベントであるため、次の場合にスムーズなハンドオーバーを呼び出します。

1. 次の場合、既存のリードマスターはリードマスターリースをリリースします。-CAMO が有効になっている場合、昇格されたノードは最新であり、キューの準備ができていない必要がより小さい。-古いリードマスターとプロモートノード間のレプリケーションラグは `node.maximum_lag` より小さい。
2. プロモートノードは、リードマスターリースを取得する唯一の有効な候補であり、現在のホルダーによってリリースされるとすぐに実行されます。他のすべてのノードは、未設定のリードマスターリースを無視します。-CAMO が有効になっている場合、昇格したノードは、リードマスターリースを保持していても、CAMO キューが完全に適用されるまで一時的にクライアントトラフィックを無効にします。
3. pgbouncer を使用している場合、HARP プロキシは PAUSE 接続を使用して、進行中のトランザクションを完了します。リードマスターリースが昇格されたノードによって要求されると、新しい接続ターゲット用に PgBouncer を再構成し、データベーストラフィックを再開します。HARP プロキシが組み込みプロキシを使用している場合、既存の接続を終了し、クライアントから新しい接続が要求されると、リードマスターへの新しい接続を作成します。

promote コマンドは次のように実行する必要があります。

```
harpctl promote (<node-name>)
```

`--force` オプションは、リードマスタになるための基準を満たさない場合でも、ノードをリードマスターに強制的に設定するために提供できます。これにより、CAMO ステータスまたはレプリケーションラグの検証が回避され、昇格したノードへの即時移行が発生します。これは、昇格のために正確なノードを指定する唯一の方法です。

これを成功させるには、ノードがオンラインで動作している必要があることに注意してください。このオプションは注意して使用する必要があります。

## 8.9 harpctl set

クラスター内の特定の属性を指定された値に設定します。これは、`apply` を使用するのではなく、特定のノード、プロキシ、場所、またはクラスター自身自身の構成設定を調整するために使用されます。これは、次のオブジェクトタイプに使用できます。

- `cluster`-クラスター関連の属性を設定します。
- `location`-特定のロケーション属性を設定します。
- `node`-特定のノード属性を設定します。
- `proxy`-特定のプロキシ属性を設定します。

### 8.9.1 harpctl set cluster

クラスター関連の属性のみを設定します。現時点ではこれらのうちの 1 つしかありませんが、HARP の将来のバージョンではさらに追加される可能性があります。

例：

```
harpctl set cluster event_sync_interval=200
```

### 8.9.2 harpctl set node

名前付けノードのノード関連属性を設定します。設定 文書の「Node Directives」セクションに記載されているオプションはすべてここで有効です。

例：

```
harpctl set node mynode priority=500
```

### 8.9.3 harpctl set proxy

名前付けプロキシのプロキシ関連属性を設定します。設定 文書の「Proxy

Directives」セクションに記載されているオプションはすべてここで有効です。この方法で設定されたプロパティでは、新しい値を有効にする前にプロキシをリスタートする必要があります。

例：

```
harpctl set proxy proxy1 max_client_conn=750
```

クラスタ全体のプロキシのデフォルトには `global` を使用します。

```
harpctl set proxy global default_pool_size=10
```

## 8.10 harpctl unfence

ローカルまたは指定されたノードから **fenced** 属性を削除します。これにより、以前に適用されたすべてのクラスター除外がノードから削除され、再びトラフィックを受信したり、リードマスターリースを保持したりできるようになります。Postgres も実行されていない場合は起動されます。

**unfence** コマンドは次のように実行する必要があります。

```
harpctl unfence (<node-name>)
```

ノード名自分自身はオプション。省略すると、**harpctl** はローカルに構成されたノードの名前を使用します。

## 8.11 harpctl unmanage

クラスター内のすべての HARP Manager サービスに実行を継続するように指示しますが、Postgres をアクティブにモニタことはなく、コンセンサスレイヤーの内容を変更することもできません。これは、ノードの停止などの通常のフェイルオーバーイベントが、リーダーシップの移行をもたらさないことを意味します。これは、システムまたは HARP のメンテナンスを目的としています HARP ソフトウェアに変更を加えるか、クラスターにその他の重要な変更を加える前に行う必要があります。

**unmanage** コマンドは次のように実行する必要があります。

```
harpctl unmanage cluster
```

---

**注釈:** 現在、`cluster` レベルでのみクラスター管理を有効または無効にできます。それ以降のバージョンでは、個々のノードまたはプロキシに対してこれを **make** ことも可能になります。

---



## 第9章 Consensus Layer Considerations

HARP は、分散制御システム (DCS) としても知られるコンセンサスレイヤーのさまざまな実装で動作できるように設計されています。

現在、次の DCS 実装がサポートされています。

-etcd- BDR

このセクションでは、サポートされている DCS 実装と HARP の相互作用に固有の情報を提供します。

### 9.1 BDR ドライバーの互換性

bdr ネイティブコンセンサスレイヤーは、BDR バージョン 3.6.21 および 3.6.21 からそれぞれ利用できます。

投票クォーラムを維持するために、BDR 論理スタンバイノードは BDR クラスター内のコンセンサス通信に参加しないことに注意してください。DCS クォーラム要件を満たすために、これらを合計ノードリストでカウントしないでください。

### 9.2 クォーラムの維持

任意のアーキテクチャのクラスターは、投票定足数を介してコンセンサスを維持するために少なくとも  $n/2 + 1$  ノードを必要とします。したがって、3 ノードのクラスターは単一ノードの停止に耐えることができ、5 ノードのクラスターは2 ノードの停止に耐えることができます。コンセンサスが失われた場合、DCS が特定のロケーション内のどのノードがリードマスターであるかを決定論的に識別することを妨げるため、HARP は動作不能になります。

その結果、どちらの DCS が選択されても、ノードの半分以上が常に `_cluster-wide_` で利用可能でなければなりません。これは、2 つ以上のデータセンター間で DCS ノードを配布するときに重要な要素になる可能性があります。ネットワークパーティションは、多数決を維持できない場所のクォーラムを妨げるため、HARP は運用を停止します。

したがって、Consensus Layer 自分自身を構築するときは、奇数のノード（最小 3）が重要です。理想的なケースは、ノードを少なくとも 3 つの独立した場所に分散して、単一のネットワークパーティションがコンセンサスを混乱させないようにすることです。

構成例の 1 つは、プライマリ BDR ノードと一致する 2 つのデータセンター内の 2 つの DCS ノードと、別の場所にある 5 番目の DCS ノード（BDR ウィットネスなど）を指定することです。そのようなデザインを使用して、2 つの BDR データセンターの間のネットワークパーティションは独立しているノードのおかげでコンセンサスを混乱させないでしょう。

## 9.2.1 マルチコンセンサスバリエーション

HARP 自分自身は、設定されたロケーションごとに 1 つのリードマスターを想定しています。通常、各ロケーションは、`location` 構成設定を使用して HARP 内で指定されます。ロケーションごとに別個の DCS クラスターを作成することにより、HARP とは無関係にこの動作をエミュレートすることが可能になります。

これを行うには、`config.yml` で HARP を設定して、目的の場所ごとに異なる DCS 接続ターゲットを使用する必要があります。

DC-A の HARP ノードは次のようなものを使用します。

```
location: dca
dcs:
  driver: etcd
  endpoints:
    - dcs-a1:2379
    - dcs-a2:2379
    - dcs-a3:2379
```

DC-B は、正規の場所にあるノードに対応する異なるホスト名を使用しますが、

```
location: dcb
dcs:
  driver: etcd
  endpoints:
    - dcs-a1:2379
    - dcs-a2:2379
    - dcs-a3:2379
```

このデザインには異なる Data Centers の間の DCS 通信が全くありません、そして、したがって、それらの間の Network Partition は HARP オペレーションにインパクトないでしょう。この結果、HARP は他のロケーションのノードを完全に認識せず、各ロケーションは本質的に別個の HARP クラスターとして動作します。

BDR は DCS として使用する場合、BDR がすべての参加者ノードにわたってコンセンサスレイヤーを維持するため、これは不可能です。

このアプローチの潜在的な欠点は、`harpctl` が現在の場所の外部のノードと対話できないことです。ノード情報の取得、リードマスターの取得または設定、または他のロケーションをターゲットとするその他のオペレーションは不可能です。基本的に、この組織は `--location` パラメータを `harpctl` に使用不可にします。

### 9.2.2 TPAexec とコンセンサス

上記の考慮事項も TPAexec に統合されています。etcd を使用してクラスターを展開すると、ロケーションごとに個別の DCS クラスターが自動的にコンストラクト、厳密な一貫性を維持しながら高可用性が促進されます。

したがって、この構成例：

```
cluster_vars:
  failover_manager: harp
  harp_consensus_protocol: etcd

locations:
  - Name: first
  - Name: second
```

first ロケーションに割り当てられた DCS ノードをグループ化すると、second ロケーションは別のクラスターになります。この動作をオーバーライドするには、特定のグループ化を強制するように `harp_location` を暗黙的に構成します。

したがって、この例では、すべての etcd ノードを単一の凝集 DCS レイヤーに結果ます。

```
cluster_vars:
  failover_manager: harp
  harp_consensus_protocol: etcd

locations:
  - Name: first
  - Name: second
  - Name: all_dcs

instance_defaults:
  vars:
    harp_location: all_dcs
```

`harp_location` オーバーライドは、従来のデータセンターを優先してリージョン内のアベイラビリティーゾーンを優先する Amazon などのクラウドプロバイダーを使用する場合、特定のノードグループを優先するためにも必要になる場合があります。



## 第10章 Security and Roles

基本的なパッケージのインストールと構成を超えて、HARP 自分自身が動作するには Postgres の権限が必要です。これらにより、必要に応じて Postgres または BDR に関する情報を収集し、コンセンサスレイヤー内のノードステータスを維持できます。

### 10.1 Postgres の許可

`node.dsn` パラメータで指定されたロールは、次のいずれかの方法で作成されている必要があります。

- `CREATE USER ...`
- `CREATE ROLE ... WITH LOGIN`

これにより、ロールがデータベースにログインして診断情報を収集できるオーダーになります。

同様に、このロールの `pg_hba.conf` にエントリーが存在する必要があります。これにはさまざまな方法があります。例として、すべてのデータベースホストが `10.10.*` のどこかにある VPN サブネットを考えます。そのような場合、最も簡単なアプローチは特定の行を追加することです：

| #       | TYPE | DATABASE | USER      | ADDRESS      | METHOD        |
|---------|------|----------|-----------|--------------|---------------|
| hostssl |      | all      | harp_user | 10.10.1.1/16 | scram-sha-256 |

---

**注釈：** この場合、廃止された ``md5`` ではなく、より新しい ``scram-sha-256`` 認証を使用しました。また、``hostssl`` を指定して SSL 認証を要求することも選択しました。

---

### 10.2 BDR のアクセス許可

BDR ノードには、特定のロールが HARP 対応ユーザに付与されている場合にのみ可視されるメタデータとビューがあります。この場合、HARP ユーザには以下が必要です。

```
GRANT bdr_monitor TO harp_user;
```

`bdr_monitor` BDR ロールは、クラスターオペレーションに関する進行中の情報を維持するためのステータスモニタリングツールを対象としているため、HARP に最適です。

## 10.3 BDR コンセンサス権限

`dc.s.driver` 設定パラメータが `bdr` に設定されている場合、HARP はコンセンサスレイヤーとして BDR 自分自身を利用します。そのため、現在 `bdr_superuser` ロールでのみ使用可能な API メソッドへのアクセスが必要です。これは、HARP 対応ユーザが以下を必要とすることを意味します。

```
GRANT bdr_superuser TO foobar;
```

これは BDR の将来のバージョンで変わるかもしれませんが、現在 BDR コンセンサスモデルへのアクセスはスーパーユーザ同等のパーミッションを必要とします。

---

**重要:** BDR スーパーユーザーは Postgres スーパーユーザーではありません\*。`bdr\_superuser` ロールには、制限された機能、テーブル、ビュー、およびその他のオブジェクトへのアクセスなど、BDR 内での昇格された特権が付与されるだけ自分自身。

---