Pgpool-II概要から導入・運用ノウハウまで徹底解説

2024/02/27

SRA OSS LLC
彭 博 (ペン ボ)

自己紹介

ペン ボ

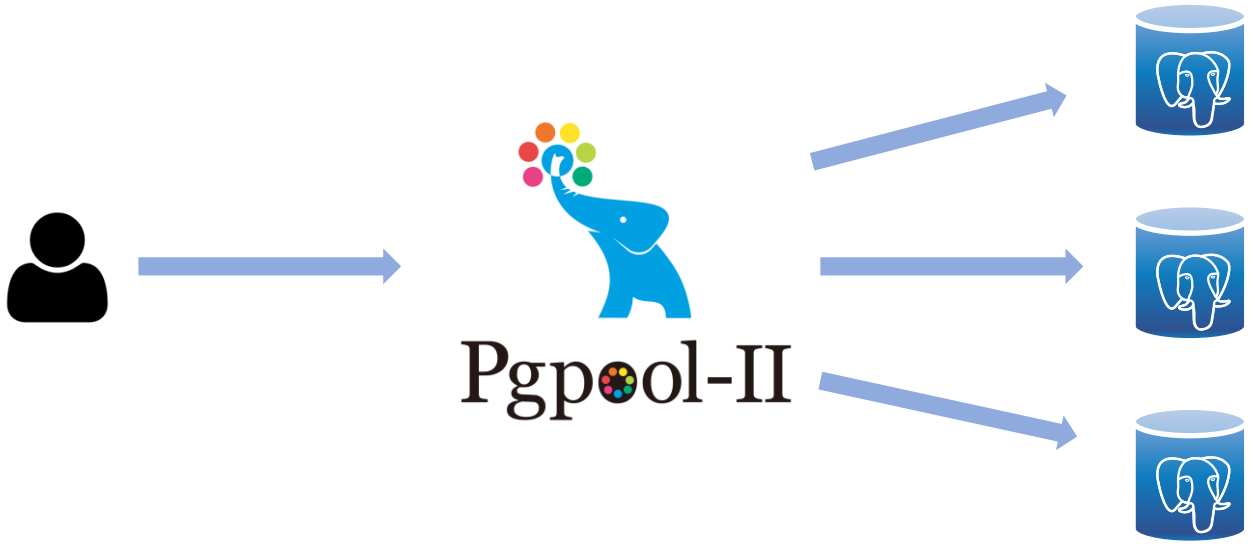
- 名前： 彭 博 (Bo Peng)
  pengbo@sraoss.co.jp
- 所属：SRA OSS LLC
  基盤技術グループ
- 職務：
  - OSS技術サポート、ミドルウェア構築
  - PostgreSQLクラスタ管理ツールPgpool-II開発者

本セッションの流れ

- Pgpool-IIの概要および機能紹介
- Pgpool-IIの設定のポイントおよび運用のノウハウ
  - 構成の紹介
  - 安全運用のための設定
  - 障害時の対応など

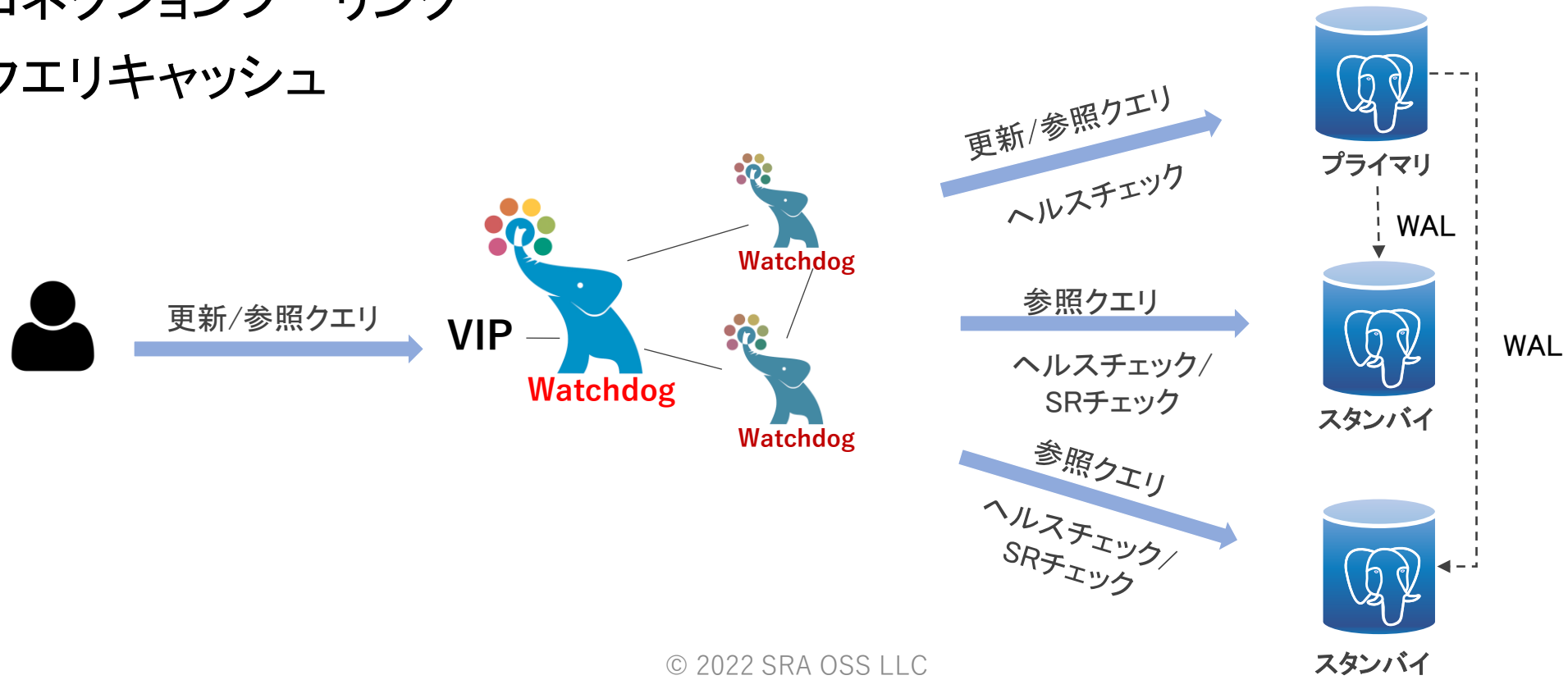
Pgpool-IIとは

- クライアントとPostgreSQLの間で動作するミドルウェア
- Pgpool Global Development Groupによって開発・メンテナンスされているOSS
- PostgreSQL単体では実現できない自動フェイルオーバー、負荷分散、コネクションプーリングなどの機能を提供
- ユーザは複数PostgreSQLサーバを意識せず、1台のように見える



Pgpool-IIの主な機能

- 参照クエリの負荷分散
- 自動フェイルオーバー
- Watchdog
- コネクションプーリング
- クエリキャッシュ



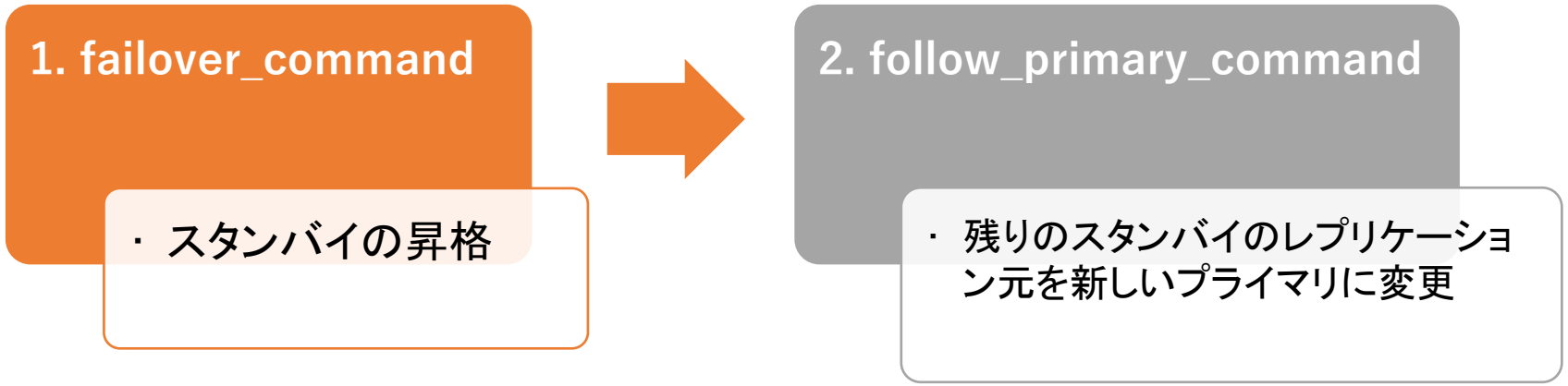
自動フェイルオーバ

- Pgpool-IIは定期的に各PostgreSQLの状態を監視する
- PostgreSQLの障害を検知すると、フェイルオーバーを実行する

自動フェイルオーバー

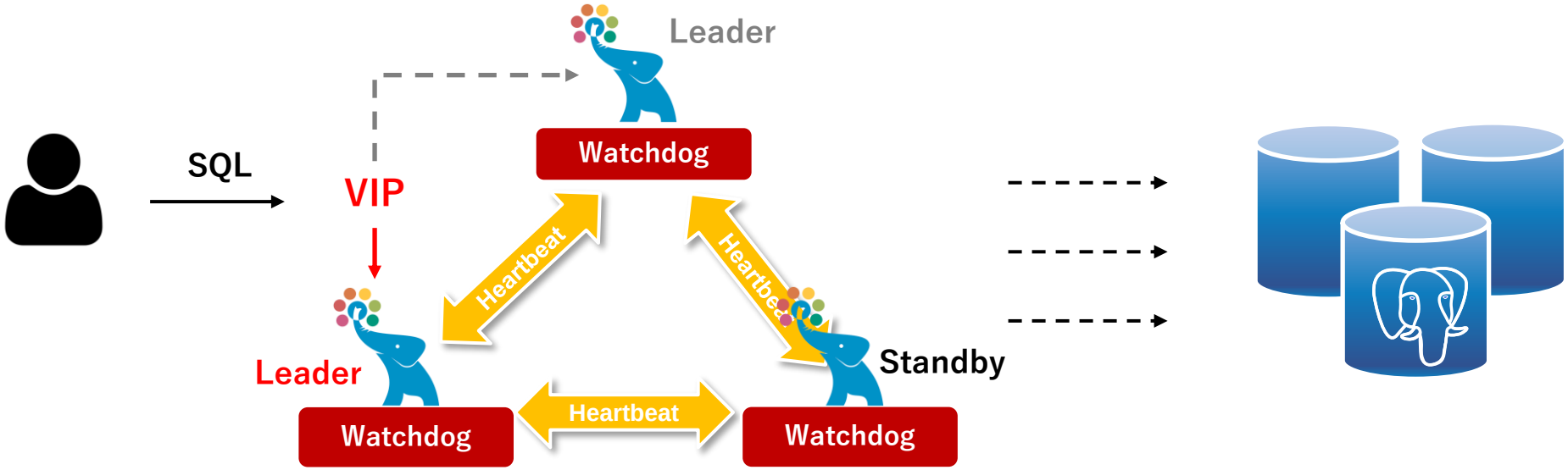
フェイルオーバー処理

- プライマリがダウンした場合
  - ダウンしたノードの状態を変更し（up->down）、以下のパラメータに設定されているスクリプトを実行



Pgpool-IIの高可用性（Watchdog）

- Pgpool-IIの単一障害点を回避
- 複数のPgpool-IIがお互いに監視することで、Pgpool-IIを冗長化するための機能
- 定期的に他のPgpool-IIノードにハートビート信号を送信
- Pgpool-IIノードの障害が検出された際に、Watchdogは投票によって新しいリーダーを決定し、切り替える
- 仮想IPの自動切り替えにより、クライアントは常に同じIPでPgpool-IIにアクセスできる
- バックエンドノードのフェイルオーバーの動作を制御

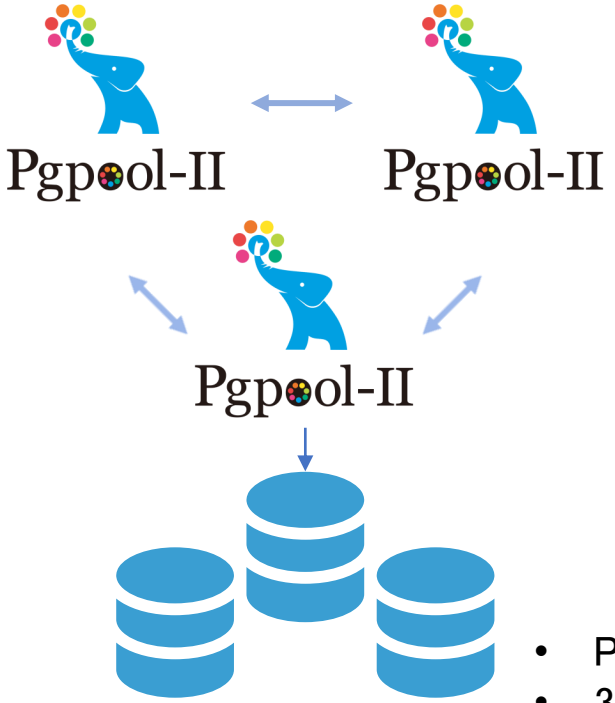


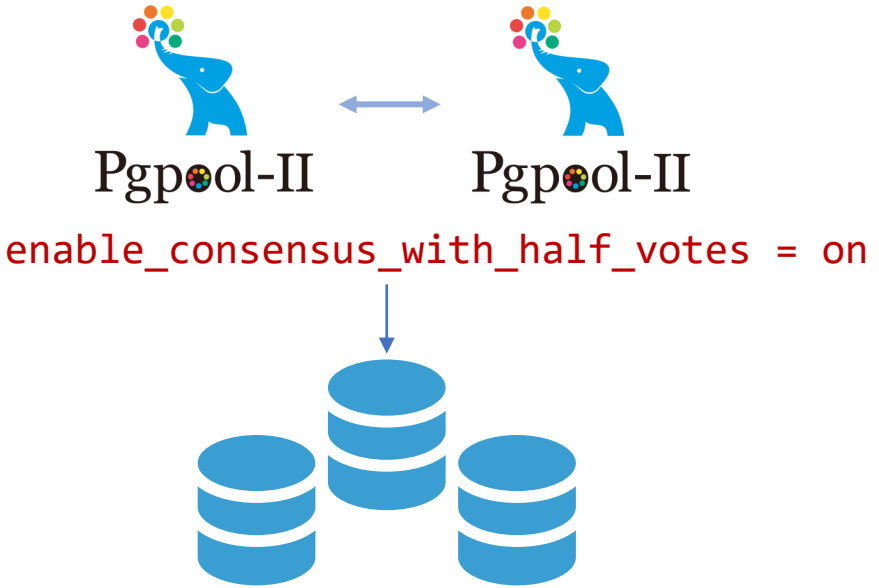
Pgpool-IIを安定稼働させるための設定・運用のポイント

Pgpool-II/PostgreSQL構成

どのような構成にするか?

- Pgpool-IIは3台構成推奨だが、2台でも動作する
  - Pgpool-II 2台構成の場合、enable_consensus_with_half_votes = onにする必要がある
  - Pgpool-II 2台構成ではスプリットブレインのリスクがあるので、物理的に安定したネットワークでの利用を推奨
- PostgreSQLは2台でも、3台以上でも可
  - 3台以上の場合は構成が多少複雑だが、性能・可用性ともに向上
- Pgpool-IIとPostgreSQLは同一サーバ上でも、それぞれの別々のサーバ上でも構成可





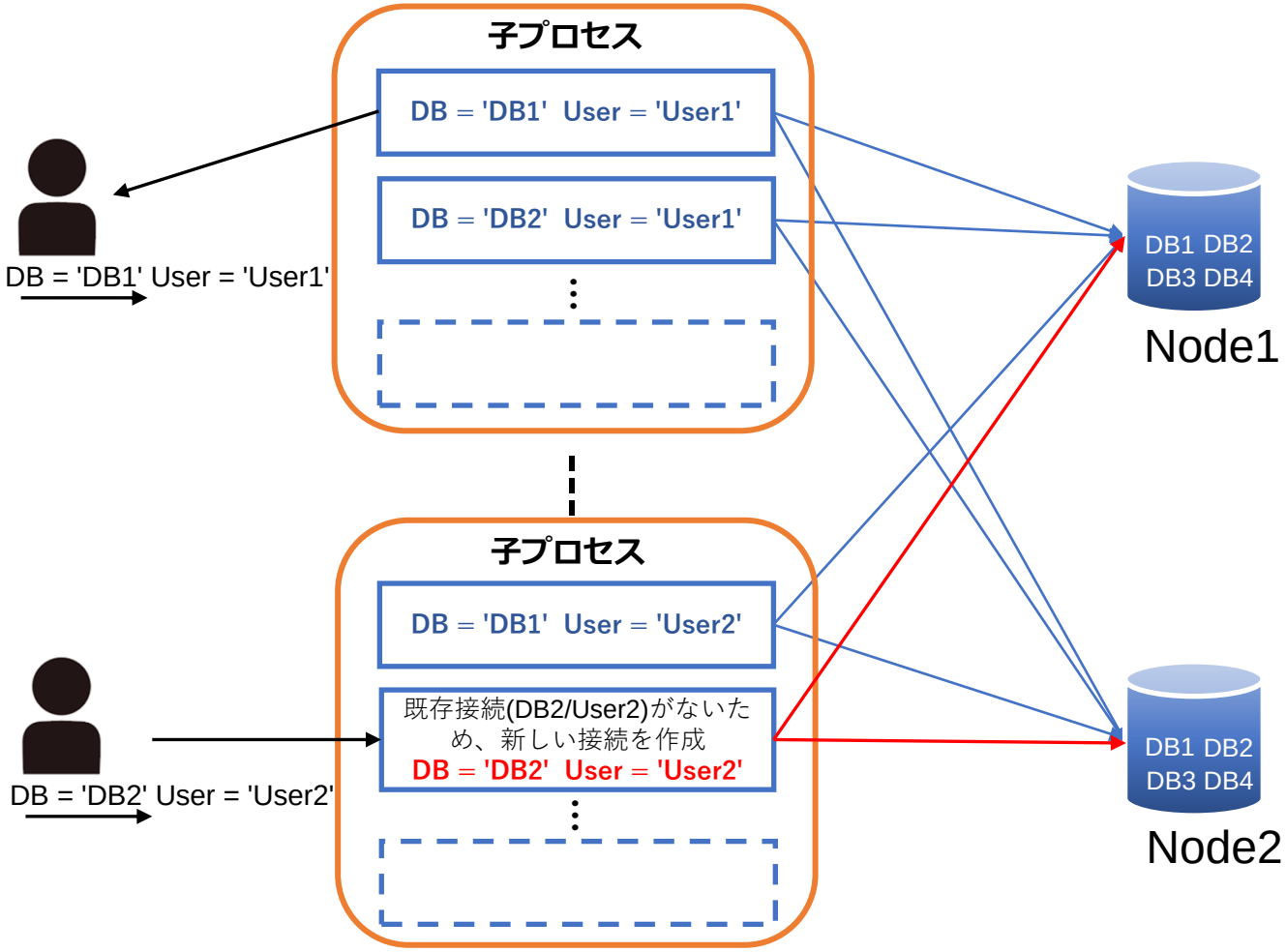
- PostgreSQLは2台でも、3台以上でもよい
- 3台以上の場合、follow_primary_commandの設定が必要

最大接続数・コネクションプール関連の設定

最大同時接続数

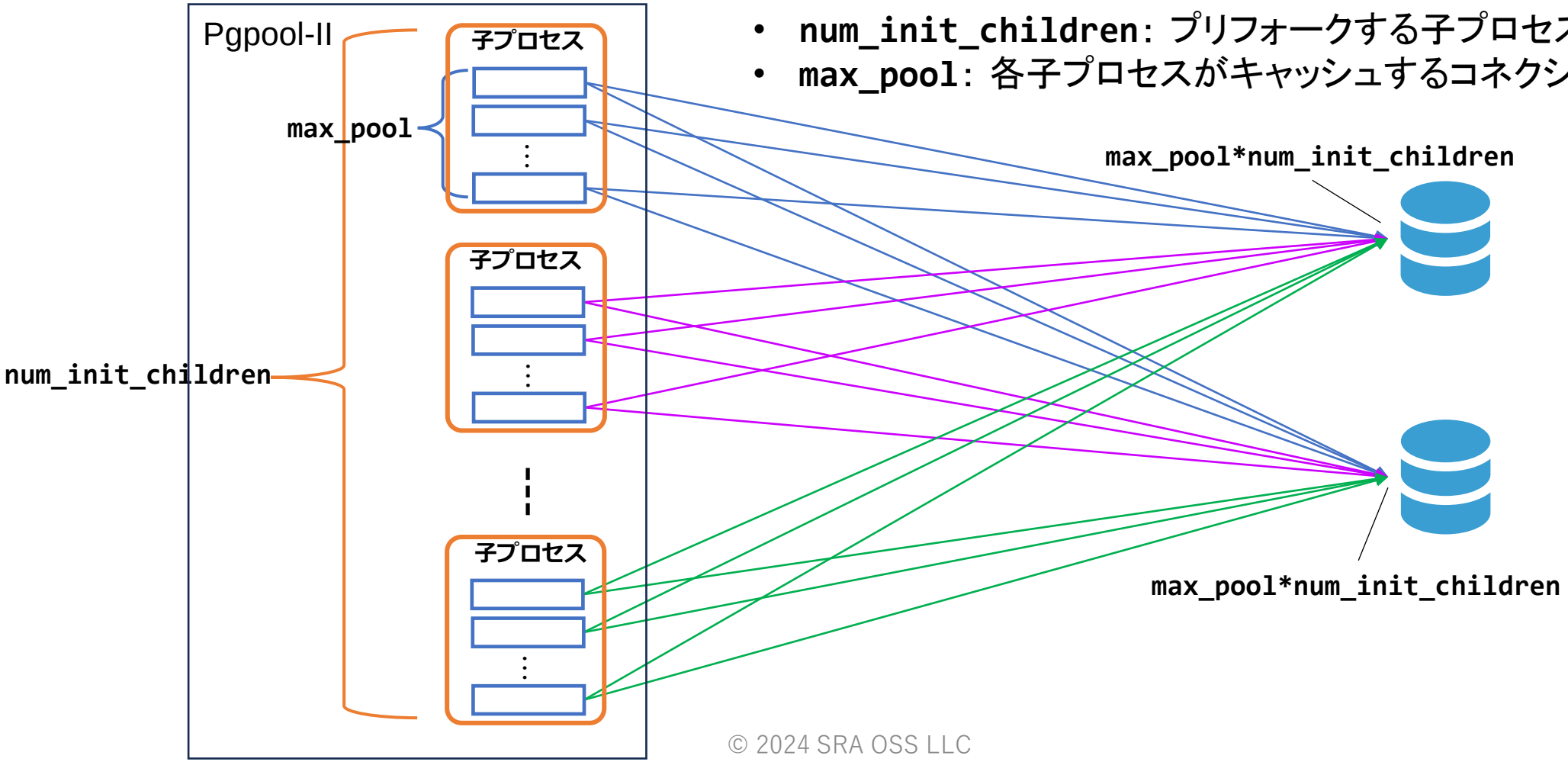
Pgpool-IIコネクションプーリングの仕組み

1. 起動時に、Pgpool-IIはnum_init_children個の子プロセスをプリフォークし、各子プロセスはmax_poolの設定値まで接続をキャッシュする
2. Pgpool-IIはクライアントからの接続要求を待ち受ける
3. クライアントからの接続要求が来ると、1つの子プロセスが接続を受け付ける
4. この子プロセスは、保持している接続内に、接続情報（データベース/ユーザ）の一致するものがあるかどうかを探す。見つかったら再利用する。
5. 見つからなかった場合、新しい接続を作成し、それをプールに登録する。プールに空きスロットがない場合、最も古い接続をクローズし、新しい接続を作成する。
6. クライアントが接続を終了する。Pgpool-IIはクライアントへの接続をクローズするが、PostgreSQLへの接続を保持する



最大同時接続数

- Pgpool-IIはすべてのバックエンドに接続する
- 各バックエンドへの最大接続数は **max_pool * num_init_children**



- **num_init_children**: プリフォークする子プロセス数
- **max_pool**: 各子プロセスがキャッシュするコネクションの最大数

最大同時接続数

他に考慮すべき点

- PostgreSQLのスーパーユーザによる接続のために予約されている接続の数
- クエリのキャンセルを行うと、通常の接続とは別に新たな接続が張られる
- 旧Watchdogリーダーが PostgreSQLへの接続を終了する前に、新リーダーがPostgreSQLへ接続するケースもあり得る

クエリのキャンセルを考慮しない場合

```
max_pool * num_init_children * 2 <=
           (max_connections - superuser_reserved_connections)
```

クエリのキャンセルを考慮する場合

```
max_pool * num_init_children * 2 * 2 <=
           (max_connections - superuser_reserved_connections)
```

コネクションプール関連設定

アプリケーション側でもコネクションプールを利用する場合要注意

- アプリケーションがコネクションプール（例えば、Tomcat JDBCコネクションプール）を使用している場合、Pgpool-IIへアイドル状態の接続が残る
- クライアントが前回のクエリからアイドル状態のまま、**client_idle_limit**以上経過した場合、Pgpool-IIはそのセッションを切断する
- client_idle_limitの設定によりPgpool-IIが接続を切断した場合、次回アプリケーションがPgpool-IIにリクエストを送信しようとしたときに、エラーが発生する

対策(Tomcat JDBCコネクションプールの場合)

アイドル接続の生存期間を設定

minIdle="0"

timeBetweenEvictionRunsMillis="5000" (デフォルト)

minEvictableIdleTimeMillis="60000" (デフォルト)

接続を検証

timeBetweenEvictionRunsMillis="5000" (デフォルト)

testOnBorrow="true"

testwhileIdle="true"

validationQuery="SELECT 1"

コネクションプール関連設定

DBサーバのメモリ使用率の上昇

- PostgreSQLの接続ごとの子プロセスは、基本的に切断までOSにメモリを返却しない
- Pgpool-IIコネクションプーリングを利用する場合、クライアントが終了してもPostgreSQLへの接続が保持され、そのプロセスのメモリはOSに返却されない
- 長く存在し続けているプロセスがあると、メモリが増え続けていることがあり得る

対策：定期的にプロセスを再起動することでメモリを OS に返却

- **child_life_time**：Pgpool-II子プロセスがアイドル状態のままでいる時間の上限
- **child_max_connections**：Pgpool-II子プロセスが受付可能なクライアント接続の数の上限

Pgpool-II子プロセスがアイドル状態になる時間がchild_life_timeより短い場合には、Pgpool-II子プロセスがPostgreSQLへの接続を終了する機会はないので、**child_max_connections**も設定しておく方がより確実に終了できる

想定外のフェイルオーバーの回避策

想定外のフェイルオーバーの回避策

フェイルオーバーの契機①

- ヘルスチェックでダウンと判定された場合
  - ヘルスチェックとは、各PostgreSQLの状態を監視するプロセス

    一時的なネットワーク障害によって、ヘルスチェックが失敗し、その PostgreSQLがダウンしていると判定し、フェイルオーバーしてしまう

回避策

ヘルスチェックリトライの設定
health_check_max_retries
health_check_retry_delay

想定外のフェイルオーバーの回避策

フェイルオーバーの契機②

- PostgreSQLへの接続時、および接続後にネットワーク通信エラーが発生し、かつfailover_on_backend_error = onの場合
    - 即時にフェイルオーバー開始（リトライ不可）
    - PostgreSQLが起動しているにもかかわらず、一時的なネットワークエラーによって、フェイルオーバーしてしまう

回避策

ネットワークエラーが頻繁に発生する環境では、
failover_on_backend_error = off

想定外のフェイルオーバーの回避策

フェイルオーバーの契機③

- pg_terminate_backend()またはSIGTERMを使ってセッションを切断し、かつfailover_on_backend_shutdown = onの場合
    - 管理者によるシャットダウン操作なのか、単にセッションが終了されたかを判断できない

回避策

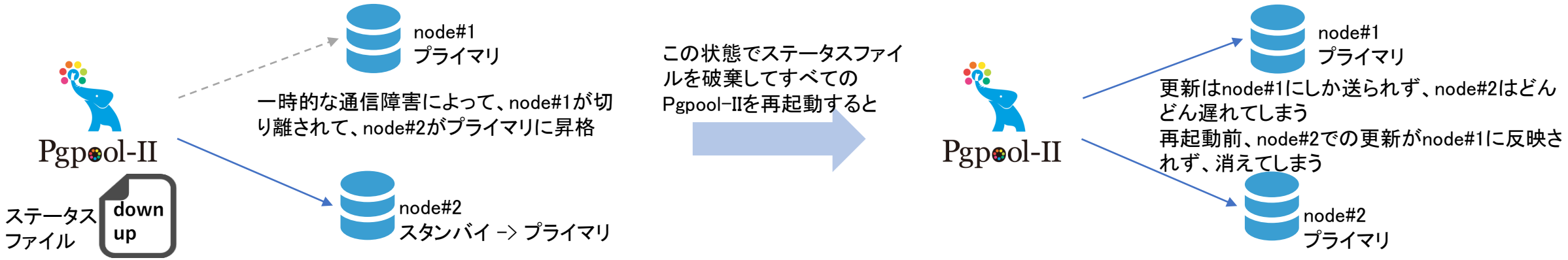
failover_on_backend_shutdown = off（デフォルト）

安全運用のための設定

安全運用のための設定

ステータスファイル（pgpool_status）でPostgreSQLの状態を管理
本番環境では、-D/--discard-status起動オプションを利用しない

- ステータスファイル
  - PostgreSQLの状態up/downを管理するファイル
  - PostgreSQLがダウン/復帰した場合に、動的にステータスファイルを更新
  - ステータスファイルが存在すれば、Pgpool-II起動時にこのファイルからPostgreSQLの状態を取得
- -D/--discard-statusオプション
  - Pgpool-II起動時に、ステータスファイルを破棄し、以前の状態を引き継がないように
  - テスト以外の用途にはお勧めしない



-D/--discard-statusオプションをつけてすべてのPgpool-IIを再起動すると、複数のプライマリが存在することがあり得る

安全運用のための設定

PostgreSQLレプリケーション構成の障害の回避はPgpool-IIで対応

- 同期レプリケーション構成の場合、スタンバイサーバが停止するとプライマリへの更新処理が止まってしまう
- レプリケーションスロットを利用する場合、スタンバイが停止し、しばらく復帰させる予定がない場合には、WALが溜まり続けてディスクフルになる可能性がある

Pgpool-IIのフェイルオーバースクリプトで対応

安全運用のための設定

PostgreSQLのレプリケーション構成の障害の回避はPgpool-IIで対応

スクリプト設定例

①
failover_command

```
if [ スタンバイがダウンした場合 ]; then
    # ダウンしたスタンバイに対応するレプリケーションスロットを削除
    SELECT pg_drop_replication_slot('レプリケーションスロット名')

    # 現プライマリでupノード数に合わせて以下を実行
    ALTER SYSTEM SET synchronous_standby_names TO 'FIRST <upノード数> (node#1,node#2,node#3)';
    SELECT pg_reload_conf();

if [ プライマリがダウンした場合 ]; then
    # 次のマスタ候補のスタンバイを昇格させた後、新プライマリで一旦同期スタンバイを空にする
    # 同期スタンバイの設定はfollow_primary_commandで行う
    ALTER SYSTEM SET synchronous_standby_names TO ''
    SELECT pg_reload_conf();
```

②
follow_primary_command
(各PostgreSQLノードで実行)

```
# このスタンバイに対応するレプリケーションスロットを作成
SELECT pg_create_physical_replication_slot('レプリケーションスロット名')

# 新プライマリのスタンバイとして再構成した後、新プライマリでupノード数に合わせて以下を実行
ALTER SYSTEM SET synchronous_standby_names TO 'FIRST <upノード数> (node#1,node#2,node#3)';
SELECT pg_reload_conf();
```

障害時の対応

障害発生時の確認ポイント

SHOW POOL_NODESやpcp_node_infoを用いてPostgreSQLの状態を確認

```
node_id|status|pg_status| role   | pg_role |replication_delay|replication_state|replication_sync_state
-------+------+---------+-------+---------+-----------------+-----------------+----------------------
0      | up   | up      |primary| primary | 0               |                 |
1      | up   | up      |standby| standby | 0               | streaming       | sync
```

- statusとpg_statusがup状態であるか
- roleとpg_roleが一致しているか
- replication_delayに大幅な遅延が発生していないか
- レプリケーションが正常に動作しているか
  - スタンバイのreplication_stateが「streaming」であるか
  - スタンバイの「replication_sync_state（sync/async）」が設定通りになっているか

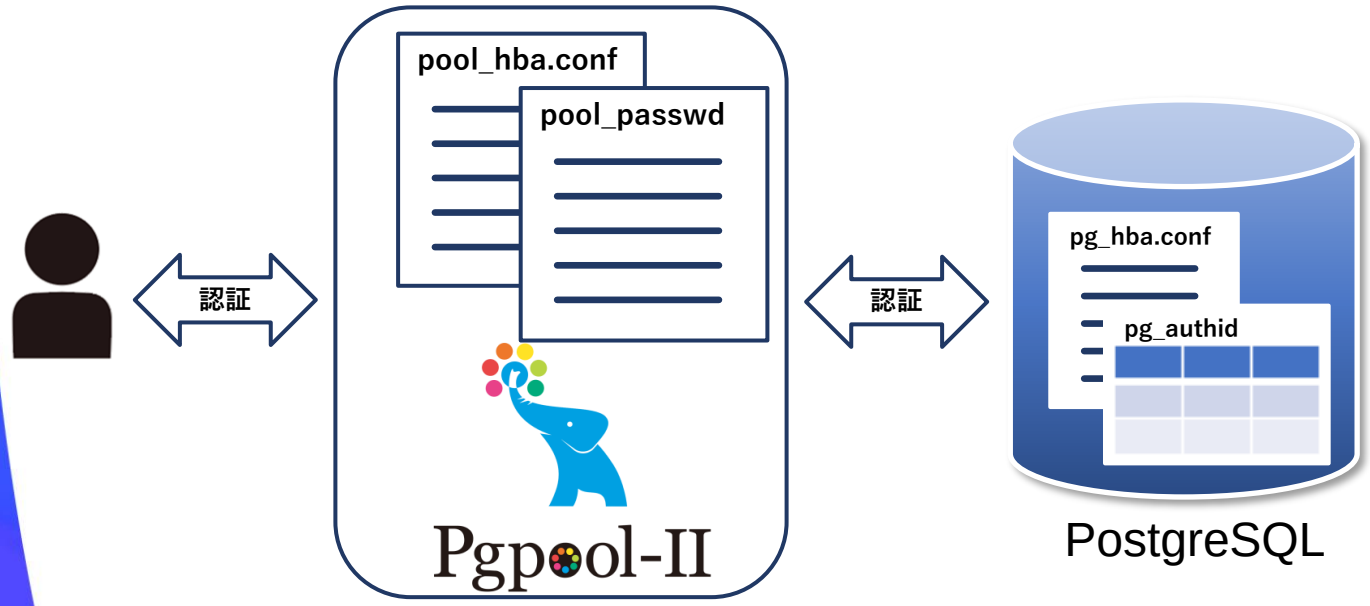
障害ノードの復旧

- ダウンしたノードをスタンバイとしてストリーミングレプリケーション構成に復帰させ、Pgpool-II管理下に戻す方法
  - Pgpool-IIオンラインリカバリ機能を使う
  - PostgreSQLベースバックアップを使う
    - ストリーミングレプリケーション構成に復旧させるだけでは、Pgpool-II管理下には戻せないため、pcp_attach_nodeを使用してPgpool-II管理下に復帰させる必要がある
  - auto_failback機能を使う
    - 一時的な通信障害によって切り離されたスタンバイを自動的に復帰させる方法
    - ストリーミングレプリケーションが正常に動作している場合に限り
- 複数プライマリが存在する、あるいはすべてのDBがダウンした場合
  - どちらかのPostgreSQLを正として、そちらのノードを基準にストリーミングレプリケーション構成を再構築

パスワードファイルの管理

Pgpool-IIクライアント認証の仕組み

- クライアントからPostgreSQLに接続するときに
  1. クライアントとPgpool-II間の認証
  2. Pgpool-IIとPostgreSQL間の認証
- PostgreSQLは実際のクライアントではなくPgpool-IIのIPアドレスしか見えないので、クライアント認証を制御するために、Pgpool-IIではPostgreSQLのpg_hba.confと同様の仕組み（**pool_hba.conf**）を持っている
- あらかじめパスワードをパスワードファイル（pool_passwd）に登録しておく必要がある



1. クライアントがPgpool-IIに接続する
2. enable_pool_hba = onの場合は、Pgpool-IIはユーザの認証方式をpool_hba.confから取得する。
   enable_pool_hba = offの場合は、Pgpool-IIはユーザの認証方式をPostgreSQLから取得する
3. Pgpool-IIは、pool_passwdファイルからユーザのパスワードを抽出する
4. ユーザはパスワードの入力を求められる
5. Pgpool-IIは、ユーザーが入力したパスワードを検証する。pool_passwdに保存されているパスワードと一致する場合、このパスワードを用いて各PostgreSQLと認証を行う

パスワードファイルを管理しない方法

- コンテナなどの利用で、パスワードファイルの管理が難しい場合には、password認証を利用することで、パスワードファイルを作成しなくて良い
- password認証を利用する場合は、パスワードは平文のまま転送されるので、**SSL暗号化通信**の利用を強くお勧めする

pgpool.conf

```
enable_pool_hba = off
ssl = on
ssl_key = '/path/to/file'
ssl_cert = '/path/to/file'
```

pool_hba.conf

```
hostssl    all    all    all    password
```

まとめ

- スプリットブレイン対策の観点から、Pgpool-II3台構成を推奨だが、2台でも動作する
- アプリケーション最大同時接続数 ≠ max_connections
- 安全運用のためのポイント
  - ステータスファイルでPostgreSQLの状態を管理
  - SHOW POOL_NODESやpcp_node_infoコマンドを使ってクラスタを監視
  - フェイルオーバースクリプトをカスタマイズすることで、機能を拡張可能

ご清聴ありがとうございました。