PostgreSQL/Pgpool-II クラスタ構築実践入門

2020-04-24

SRA OSS, Inc. 日本支社 高塚 遥

講演の概要:

- Pgpool-IIで PostgreSQLの

  高可用性
  ＋
  参照負荷分散

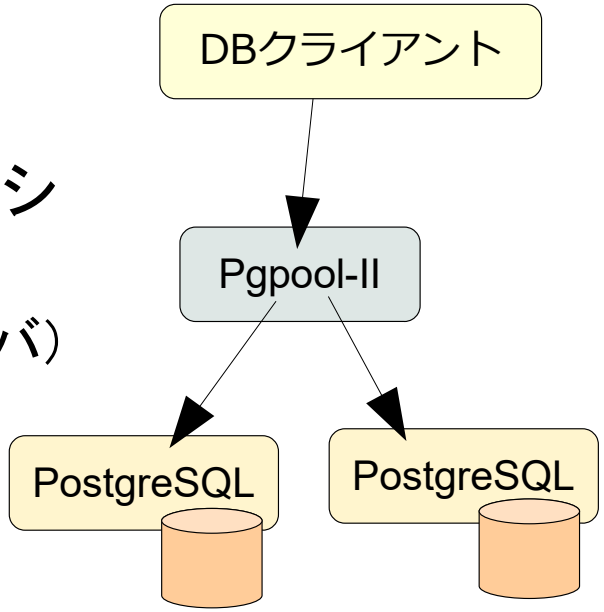
  クラスタを構築する方法を解説

講演者:

- 高塚 遥
- 普段は ヘルプデスク、コンサルティングなど、PostgreSQLの支援業務を各種手掛ける
- (特非)日本PostgreSQLユーザ会 理事

PostgreSQL/Pgpool-IIとは

- PostgreSQLとは
  - 代表的なオープンソース RDBMS
- Pgpool-II
  - PostgreSQL用のOSS多機能プロキシ
    - コネクションプール
    - ノード制御（PostgreSQLのフェイルオーバ）
    - クエリベースレプリケーション
    - ロードバランス
    - クエリキャッシュ
    - 自身の高可用性クラスタ動作



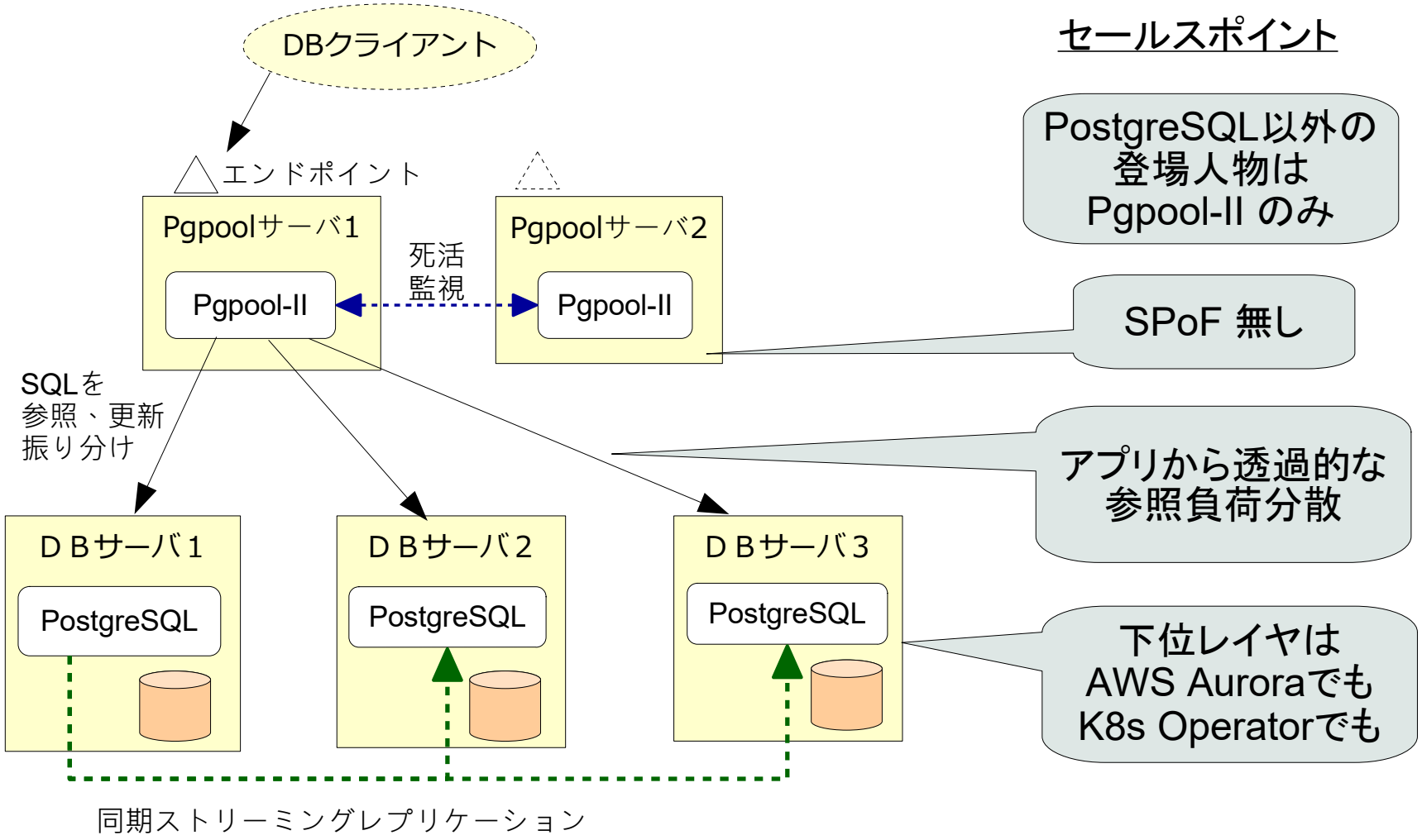
Pgpool-II 不要？

- 今時のPostgreSQL配備は、クラウドサービスを使ったり、コンテナプラットフォームに載せて、任せとけばよいのでは？
- L4ロードバランサで十分では？

⇒ 透過的な参照/更新振り分けができるソフトウェアは、他に無い。

現代的な各種要素とも組み合わせ可能。

こういうものを作る方法を解説します



ソフトウェア入手

- 今回の環境: CentOS 7
- PostgreSQL
  - RHEL/CentOSなら PGDG yumリポジトリから
    https://yum.postgresql.org/repopackages.php
- Pgpool-II
  - PGDGにも含まれるが推奨は pgpool.net のリポジトリ
    - ◎小刻みなバグ修正リリース ⇔ × 複数バージョン系列共存

    https://www.pgpool.net/mediawiki/index.php/Yum_Repository

導入先ホストの下準備

- 鍵認証ssh
  - Pgpoolサーバ → DBサーバ
  - DBサーバ間　　（場合によっては）
  - Pgpoolサーバ間　（場合によっては）
  - 通常は postgresユーザ で
- Firewall
  - 登場するポートを空ける
    - wd_heartbeat_port は UDP、それ以外は TCP
- SELinux
  - 無効、または、テストして各種許可ポリシー追加

PostgreSQL同期レプリケーション構築

- 今回バージョン: PostgreSQL 12.x

◆各サーバで

```
[root]# yum install postgresql12-server
[root]# su - postgres
[postgres]$ vi .bash_profile
    export PATH=/usr/pgsql-12/bin:$PATH
    export LD_LIBRARY_PATH=/usr/pgsql-12/lib:$PATH
    export PGDATA=/var/lib/pgsql/12/data
```

◆プライマリで

```
[postgres]$ initdb --no-locale -E UTF-8
[postgres]$ vi $PGDATA/postgresql.conf
    listen_addresses = '*'
    synchronous_commit = remote_apply
[postgres]$ vi $PGDATA/pg_hba.conf
    host  replication  all  samenet  trust
    host  all          all  samenet  trust
[postgres]$ pg_ctl start
```

「local」なら
非同期モード

DBサーバ間で
replication接続および
DB接続を許可し、
Pgpoolからの DB接続 を許可

◆各スタンバイで

```
[postgres]$ pg_basebackup -h host1 --write-recovery-conf -D $PGDATA
[postgres]$ vi $PGDATA/postgresql.auto.conf
    primary_conninfo = 'user=postgres passfile=''/var/lib/pgsql/.pgpass''
    host=host1 port=5432 sslmode=prefer sslcompression=0 gssencmode=prefer
    krbsrvname=postgres target_session_attrs=any application_name=host2'

[postgres]$ pg_ctl start
```

ANY を使って
シンプルに記述

◆プライマリで

```
[postgres]$ psql -t
=# ALTER SYSTEM SET synchronous_standby_names TO 'ANY 2 (host1,host2,host3)';
=# SELECT pg_reload_conf();
=# SELECT * FROM pg_stat_replication;
25282 |       10 | postgres | host2           | 10.10.10.151 |
 |       42980 | 2020-02-06 13:00:38.774745+09 |              | streaming | 0/50
00148 | 0/5000148 | 0/5000148 | 0/5000148  |           |           |
|            1 | quorum     | 2020-02-06 13:23:45.055784+09
 25297 |       10 | postgres | host3           | 10.10.10.152 |
 |       49996 | 2020-02-06 13:01:11.60794+09  |              | streaming | 0/50
00148 | 0/5000148 | 0/5000148 | 0/5000148  |           |           |
|            1 | quorum     | 2020-02-06 13:23:45.787998+09
```

Pgpool-II構築 - まず単体で

- 今回のバージョン: Pgpool-II 4.1.x
- やるべきこと
  - インストール
  - 設定ファイル記述
  - pcpユーザ作成
  - フェイルオーバコマンド配置　（バックエンド障害時の
  - フォローマスタコマンド配置　　構成固有の追加処理）
  - オンラインリカバリ設定　　（バックエンド復旧を簡単に）

◆pgpoolサーバで
```
[root]# yum install postgresql12-server
[root]# yum install pgpool-II-pg12
```

/etc/pgpool-II/pgpool.conf

- ベースは pgpool.conf.sample-stream

```
listen_addresses = '*'
backend_hostname0 = 'host1'
backend_port0 = 5432
backend_weight0 = 1
backend_data_directory0 = '/var/lib/pgsql/12/data'
backend_flag0 = 'ALLOW_TO_FAILOVER'
backend_application_name0 = 'host1'
 {バックエンド1、バックエンド2 についても上記と同様に記述}
log_destination = 'syslog'
log_line_prefix = ''
follow_master_command = '/etc/pgpool-II/follow_master_command.sh %d %h %p %D %m
 %H %M %P %r %R %N %S'
failover_command = '/etc/pgpool-II/failover_command.sh %d %h %p %D %m %H %M %P
 %r %R %N %S'
health_check_period = 20
```

当初の最小限の
変更すべき箇所

/etc/pgpool-II/pcp.conf

- pcp.conf は pg_md5コマンドで生成

```
[postgres]$ echo "postgres:$(/usr/bin/pg_md5 {パスワード})" > /etc/pgpool-II/pcp.conf
[postgres]$ chmod 0600 /etc/pgpool-II/pcp.conf
```

  - pcp_で始まる Pgpool-II操作コマンドを使うのに必要

pgpool 起動

- 死活確認用の DBユーザを作っておく

◆プライマリDBサーバで実行

```
[postgres]$ psql
=# CREATE ROLE nobody LOGIN;
```

- 起動

◆pgpoolサーバで実行

```
[postgres]$ pgpool
[postgres]$ psql -t -p 9999 -h localhost -U postgres -d postgres
=# SHOW pool_nodes;
 0       | host1     | 5432 | up     | 0.333333  | primary | 0          | false
     | 0                 |                   |                     | 2020-02-06 17:58:09
 1       | host2     | 5432 | up     | 0.333333  | standby | 0          | true
     | 0                 |                   |                     | 2020-02-06 17:58:09
 2       | host3     | 5432 | up     | 0.333333  | standby | 0          | false
     | 0                 |                   |                     | 2020-02-06 17:58:09
```

md5、scram-sha-256 接続の設定

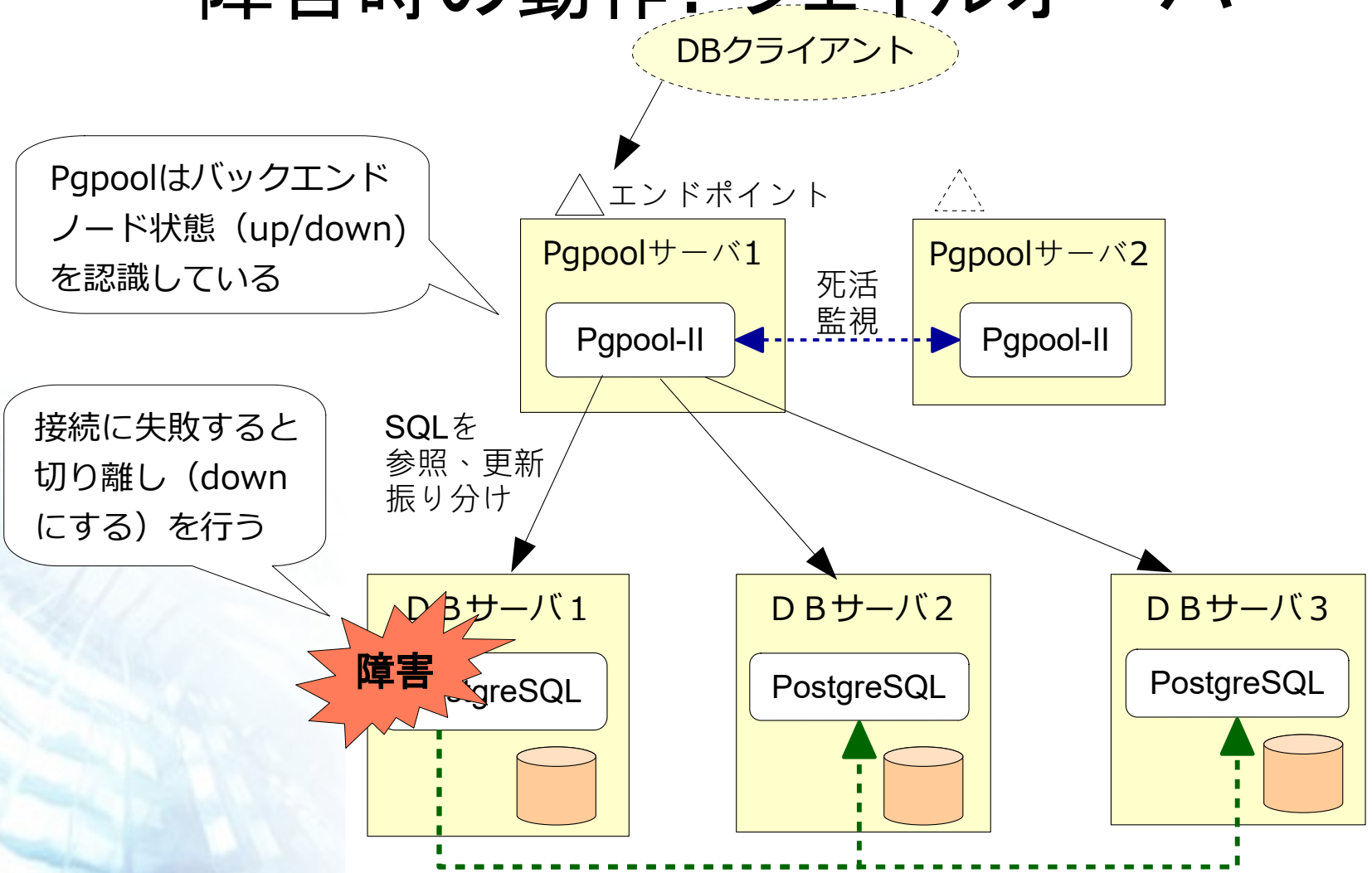
- md5認証には Pgpool-II 自体でハッシュの保持が必要
  - Scram-sha-256 の場合 パスワードそのものの保持が必要

◆pgpoolサーバで

```
[postgres]$ vi /etc/pgpool-II/pgpool.conf
    enable_pool_hba = on
    pool_passwd = 'pool_passwd'

[postgres]$ pg_md5 --md5auth --username=postgres pass
[postgres]$ cat /etc/pgpool-II/pool_passwd
postgres:md5d2743fc4ae70b03845da881b3d77b963

[postgres]$ vi /etc/pgpool-II/pool_hba.conf
  host    all       all       127.0.0.1/32       md5

[postgres]$ pgpool reload
[postgres]$ psql -t -h 127.0.0.1 -p 9999 -U postgres
ユーザ postgres のパスワード:
```

Pgpool-II で認証が行われるのと合わせて、PostgreSQL側の md5認証にそのまま引き渡される。

障害時の動作: フェイルオーバ



フェイルオーバコマンド

- フェイルオーバ時に呼ばれるスクリプト
  - なくてもノード切り離し (down状態移行) は行われる
- サンプル failover.sh.sample が付属
  - 非同期レプリケーションならこのまま使える
    - PostgreSQLバージョン、ssh鍵ファイル名などを調整
    - レプリケーションスロットの処理も含まれる
- 本構成で記述すべき、基本の処理内容:
  - スタンバイがダウンなら、何もしない
  - プライマリがダウンなら、次マスタ候補の DBサーバに昇格コマンドを投げる

同期モードのフェイルオーバコマンド

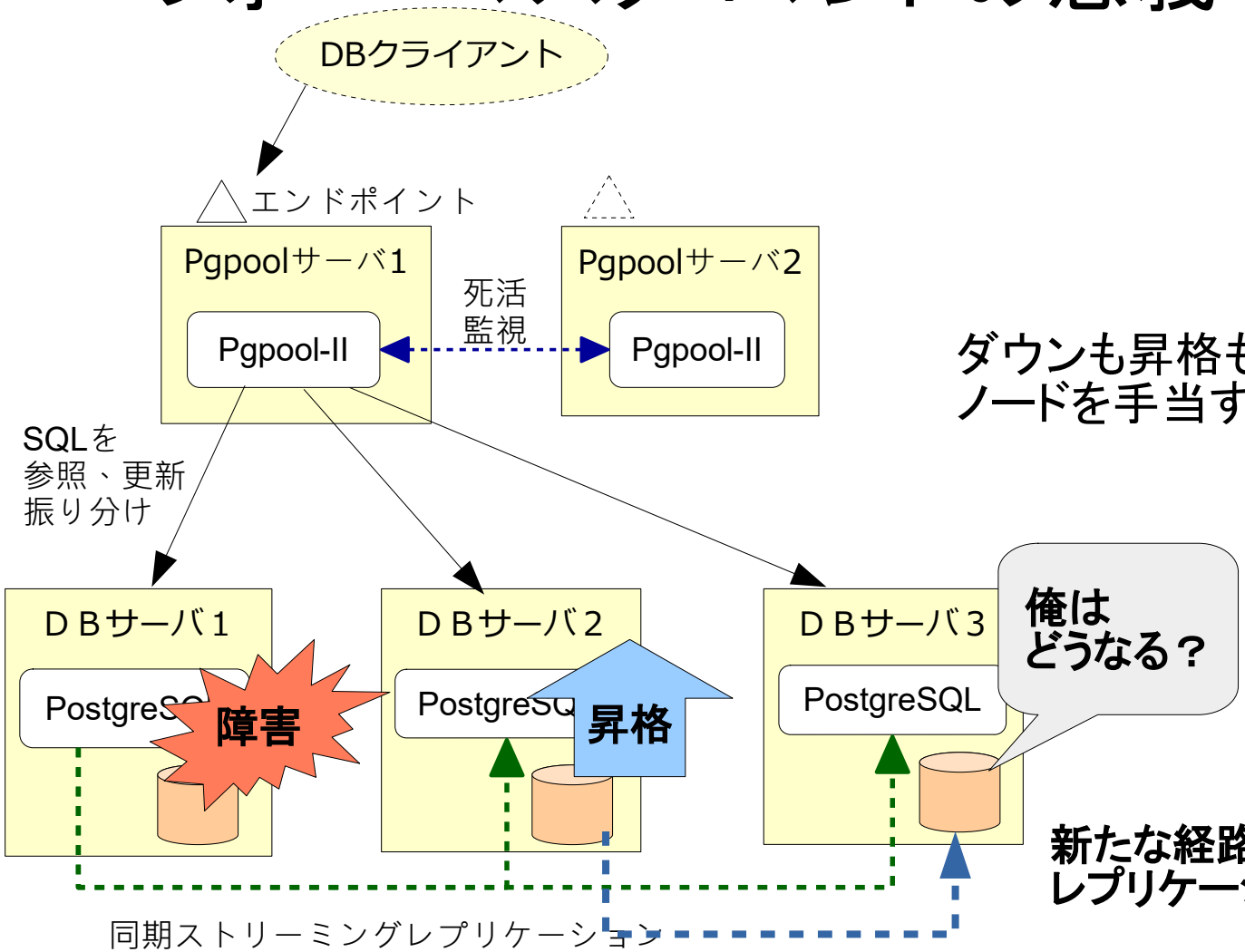
- スタンバイがダウン:
  - 現プライマリDBでupノード数に合わせて以下SQLを実行

```
ALTER SYSTEM SET synchronous_standby_names TO 'ANY 1 (host1,host2,host3)';
SELECT pg_reload_conf();
```

- プライマリがダウン:
  - 次マスタ候補のDBサーバでPostgreSQLを昇格させる
  - 昇格した新プライマリDBで同様にupノード数に合わせて synchronous_standby_name を変えるSQLを実行

ANY 2 ⇒ ANY 1 ⇒ 空欄

フォローマスタコマンドの意義



ダウンも昇格もしなかった
ノードを手当する

新たな経路で
レプリケーションが必要

フォローマスタコマンド

- 設定があると、プライマリダウンのフェイルオーバ時、新プライマリ以外のノードが全て一度downになり、各ノード毎に指定したスクリプトが実行される
  - 設定しなければ ノードが down になることも無い
- サンプル follow_master.sh.sample が付属
  - たくさんの処理が盛り込まれている
- 本構成で記述すべき、基本の処理内容:
  - 当該DBサーバのレプリケーション元を新プライマリに変更
  - ノードをアタッチする(upにする)コマンドを実行する

follow_master.sh.sample

- 当該スタンバイPostgreSQLが稼働中なら:
  - そのスタンバイPostgreSQLについて:
    - PostgreSQLを停止
    - 設定を書き換えてレプリケーション元を新プライマリに変更
    - pg_rewindでデータのWAL位置を新プライマリに合わせる
    - pg_rewind失敗なら、pg_basebackupで初期コピー再作成
    - PostgreSQLを起動
  - pcp_attach_nodeで当該ノードをアタッチ（upにする）
  - このほかレプリケーションスロットの処理も含まれる
- PostgreSQLが停止中なら何もせず down のまま
  - 障害ノードは自動復旧させないという趣旨

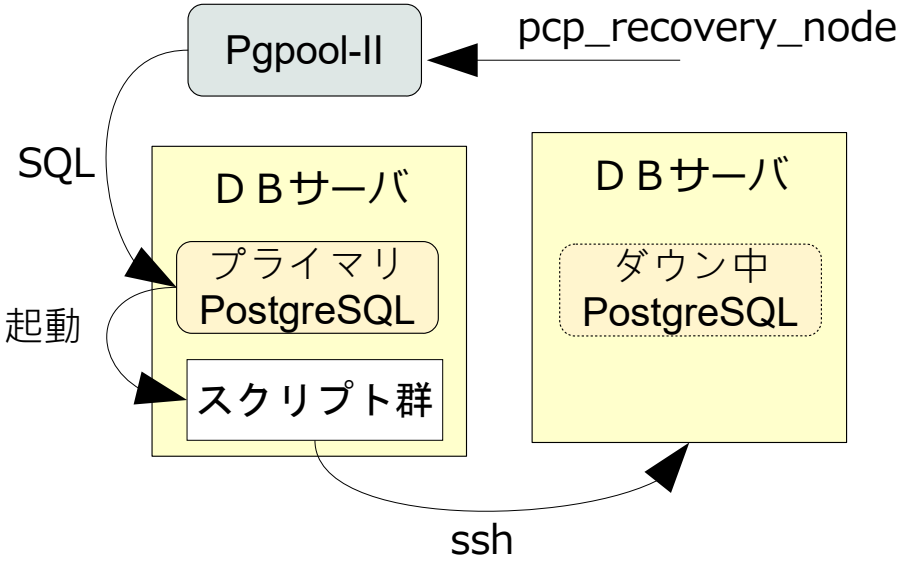
同期モードのフォローマスタコマンド

- フォローマスタコマンド設定があると、フェイルオーバ時に upノード数は 1つ（新プライマリ）だけになる
- フォローできたなら、upノード数が増えるので、プライマリsynchronous_standby_namesを変更
  - 3ノード構成なら、あり得るのは「ANY 1」のみ
  - 4ノード以上や、障害ノードを自動復旧するなら、残upノード数に応じた指定が必要

```
ALTER SYSTEM SET synchronous_standby_names TO 'ANY 1 (host1,host2,host3)' ;
SELECT pg_reload_conf();
```

オンラインリカバリについて

- Pgpoolサーバに対する pcp_recovery_node でバックエンドノード（PostgreSQL）を復旧
  - pg_basebackup で再初期同期して、pcp_attach_node でアタッチする
  - 設定箇所が多い
    - pgpool_recovery拡張
    - $PGDATAにスクリプト
  - GUI（pgpoolAdmin）の「リカバリ」ボタンを使いたいなら



Pgpool-II を高可用性化

- Pgpool-IIのwatchdog機能を使う
  - pgpoolサーバを追加して、pgpool.conf に設定
  - Pgpool-II自体の死活監視、フェイルオーバのみならず、バックエンドノードの制御について協調動作する
- 他の方法も:
  - 一般的な各種HAクラスタソフトで保護
  - サーバダウンに対しては保護されている仮想マシン上でサービスの監視と自動再起動設定
  - コンテナに載せて、K8s等で保護

```
use_watchdog = on
wd_hostname = 'poolhost1'
delegate_IP = '10.10.10.155'
if_up_cmd = '/usr/bin/sudo /sbin/ip addr add $_IP_$/24 dev eth0 label eth0:0'
if_down_cmd = '/usr/bin/sudo /sbin/ip addr del $_IP_$/24 dev eth0'
arping_cmd = '/usr/bin/sudo /usr/sbin/arping -U $_IP_$ -w 1 -I eth0'
heartbeat_destination0 = 'poolhost2'
other_pgpool_hostname0 = 'poolhost2'
other_pgpool_port0 = 9999
enable_consensus_with_half_votes = on
```

watchdog設定:
最小限の変更すべき箇所
（ホストごとに設定が異なる）

```
use_watchdog = on
wd_hostname = 'poolhost2'
delegate_IP = '10.10.10.155'
if_up_cmd = '/usr/bin/sudo /sbin/ip addr add $_IP_$/24 dev eth0 label eth0:0'
if_down_cmd = '/usr/bin/sudo /sbin/ip addr del $_IP_$/24 dev eth0'
arping_cmd = '/usr/bin/sudo /usr/sbin/arping -U $_IP_$ -w 1 -I eth0'
heartbeat_destination0 = 'poolhost1'
other_pgpool_hostname0 = 'poolhost1'
other_pgpool_port0 = 9999
enable_consensus_with_half_votes = on
```

watchdog構成では
「ヘルスチェック有効」も重要
health_check_period = 20

エンドポイントの切り替え

- 仮想IP
- 仮想IPが利用できないクラウド上では:
  - Elastic IP の類（ドキュメントに AWSむけサンプル有）
  - ルートテーブル（AWS VPC の場合）
  - L4ロードバランサ
  - プライベートむけDNS
  - いずれも if_up_cmd、if_down_cmd にスクリプト指定
- 各WebAPサーバ毎に pgpool-II を配置
  - watchdog連携する／しない

pgpool.conf に

```
if_up_cmd =   'sudo ifupdown.sh up   172.31.25.241 20 eni-045049b95775cd5dc eth0 us-west-2'
if_down_cmd = 'sudo ifupdown.sh down 172.31.25.241 20 eni-05271115f9a4aff67 eth0 us-west-2'
arping_cmd = 'true'
```

AWS の プライベートIP を使った一例
※AZ内なので実運用むけではない

ifupdown.shスクリプト:

```
#!/bin/bash
CMD=$1   ;  ADDR=$2   ; MSK=$3   ;
ENI=$4   ;  DEV=$5    ; RGN=$6
if [ "_$CMD" = "_up" ]
then
  /usr/bin/aws --region $RGN ec2 assign-private-ip-addresses ¥
    --network-interface-id $ENI --private-ip-addresses $ADDR
  /sbin/ip addr add ${ADDR}/${MSK} dev $DEV
elif ["_$CMD" = "_down" ]
then
  /sbin/ip addr del ${ADDR}/${MSK} dev $DEV
  /usr/bin/aws --region $RGN ec2 unassign-private-ip-addresses ¥
    --network-interface-id $ENI --private-ip-addresses $ADD
fi
```

fence/STONITH の類

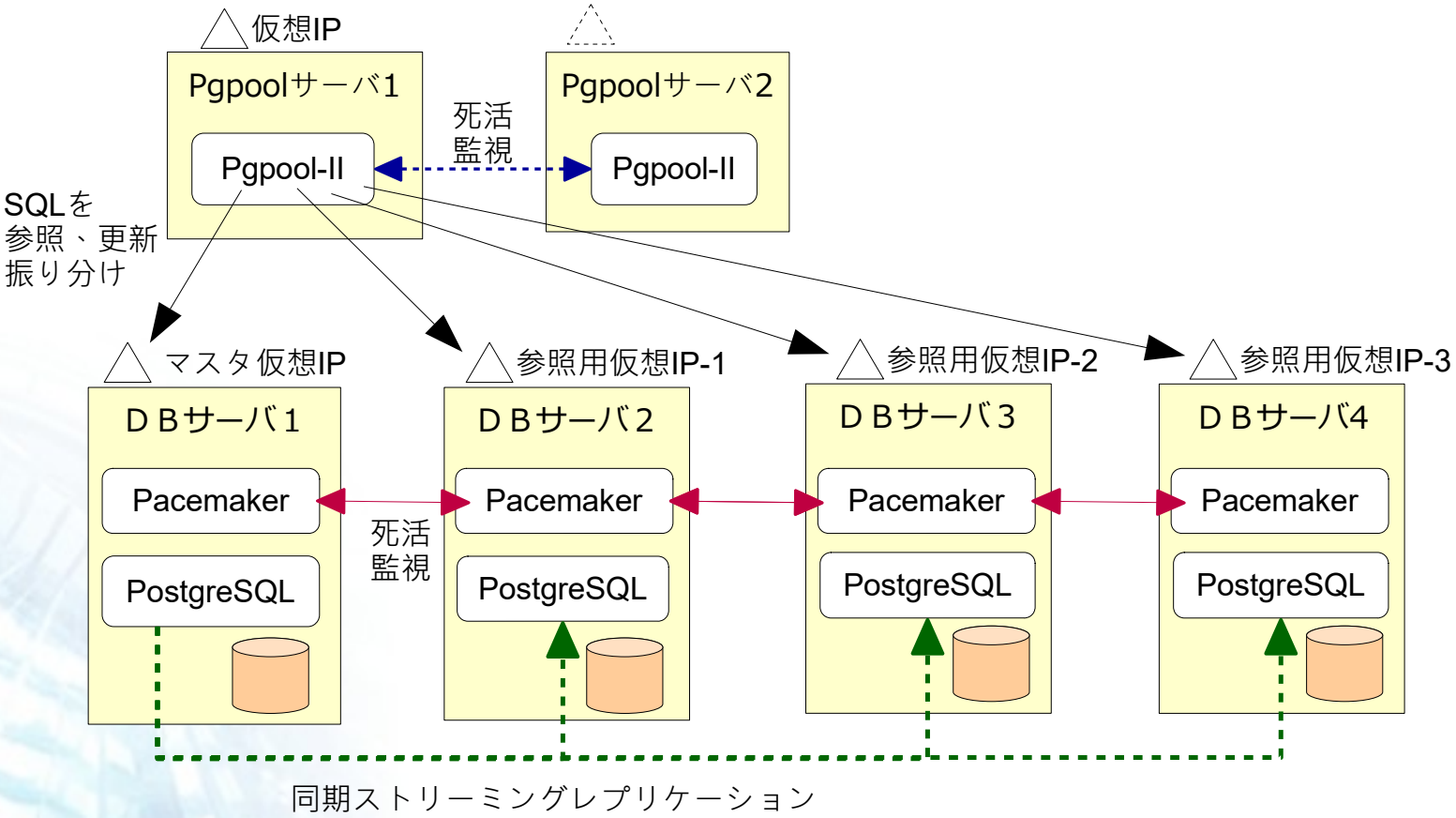
- ダウンした pgpool を確実に無効化したい
  - サーバは生きていて仮想IPが残っている場合など
- wd_escalation_command にスクリプトを設定
  - sshでダウンしたサーバに行って仮想IPを解除
  - ダウンしたサーバをシャットダウン
  - ssh不能ならそれでOK、
    あるいは、ipmi や、仮想化基盤へのAPIを使用
  - pgpool が 3ノード以上の場合にダウンしたものを
    把握するには pcp_watchdog_info を使用

下位レイヤに耐障害性がある場合

- AWS Aurora などクラウドサービスのPostgreSQL
- K8s 各種の賢いPostgreSQLオペレータ
- Pacemaker + PostgreSQLマスタスレーブ構成

- Pgpool-II の設定
  - フェイルオーバ禁止 → フェイルオーバコマンド等も不要
  - プライマリ固定
  - バックエンドの各種検査処理を省略 （構成によっては）
  - ドキュメントに AWS Aurora むけ設定の言及がある

構成例: Pacemakerクラスタと組み合わせ



```
backend_hostname0 = '《read_write_endpoint》'
backend_port0 = 5432
backend_weight0 = 1
backend_data_directory0 = '/DUMMY/data'
backend_flag0 = 'ALWAYS_MASTER|DISALLOW_TO_FAILOVER'
backend_application_name0 = ''
backend_hostname1 = '《readonly_endpoint_1》'
backend_port1 = 5432
backend_weight1 = 1
backend_data_directory1 = '/DUMMY/data'
backend_flag1 = 'DISALLOW_TO_FAILOVER'
backend_application_name1 = ''
   {バックエンド2以降についても上記と同様に記述、
    Auroraのようにロードバランス機能も提供されるなら バックエンド 0 と 1 だけで良い}
sr_check_period = 0
health_check_period = 0
failover_on_backend_error = off
```

自己回復できる
バックエンドPostgreSQL
に対する pgpool.conf設定例

まとめ

- Pgpool-II は引き続き採用されています
  - 透過的SQL参照更新振り分けができるのが優位点です
- Pgpool-II ＋ PostgreSQL で、
  高可用性 ＋ 参照負荷分散 のクラスタが組めます
- Pgpool-II のクラスタ構成は分かりにくい所があるので解説しました
- クラウドサービスやコンテナといった、現代的なコンポーネントとも組み合わせできます