PostgreSQL 17
増分バックアップの実践活用と運用上の考慮点

株式会社 SRA OSS
越野 太基

自己紹介

- 名前： 越野 太基
  koshino@sraoss.co.jp
- 所属：株式会社SRA OSS
  技術部 データベース技術グループ
- 職務：
  - PostgreSQL技術サポート
  - PostgreSQLクラスタ管理ツールPgpool-II開発者

目次

- PostgreSQL の増分バックアップの有用性
- PostgreSQL 標準機能による増分バックアップ手順
- リストア手順
- まとめ

PostgreSQLの
増分バックアップの有用性

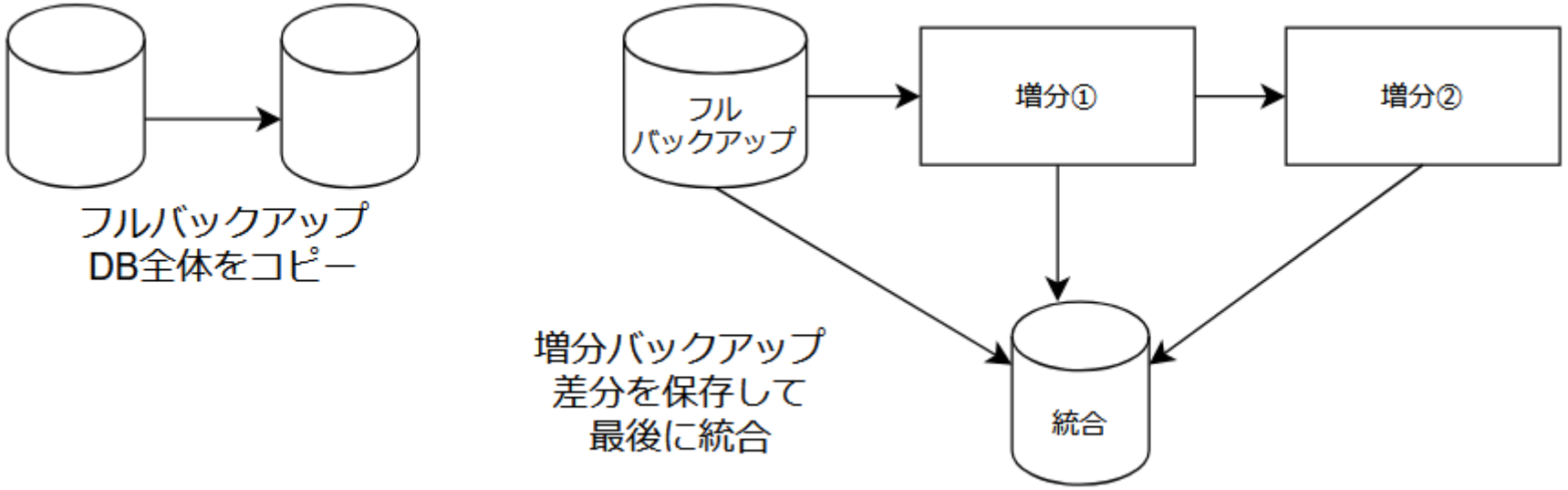
PostgreSQLのバックアップ方法の種類

- 論理バックアップ
  - SQL ステートメントとして保存、pg_dump などのツールで実行
    - テーブル単位の復元が可能
    - 異なるバージョン間の移行に適している
- 物理バックアップ
  - データベースファイルをそのままコピー、pg_basebackup 等実行
    - フルバックアップ
    - 増分バックアップ
  - PITR に対応している

★今回は物理バックアップに注目

フルバックアップと増分バックアップ

- フルバックアップ
  - データディレクトリ全体をコピーする方法
- 増分バックアップ
  - 前回のバックアップ以降で変更・追加された部分だけを保存する方法



フルバックアップ vs 増分バックアップ

	ストレージコスト (例)1 TB DB を 30 日分管理する場合	バックアップ時間 (例)10 % の変更	サーバーへの負荷
フルバックアップ	各バックアップで フルサイズ (例) 1 TB × 30 = 30 TB	全ファイルをバックアップする時間がかかる	取得時間が長いため 比較的高い
増分バックアップ	初回フルバックアップ ＋増分のみ (例) 1TB + 増分サイズのみ	増分の取得時間のみ (例) 10 % の変更で より短時間で取得	取得時間が短いため 比較的負荷は低い

- 複数世代を管理するのに非常に効率的
  - ※手順の誤りが起きやすい点には注意
    - 統合前に最新のフルバックアップファイルを消してしまうなど

PostgreSQL 標準機能による増分バックアップ手順

PostgreSQL17の新機能

増分バックアップが公式機能に

- pg_basebackup コマンドの拡張
  - --incremental オプションが追加、増分のみ取得可能に
- pg_combinebackup
  - フルバックアップと増分バックアップを統合
- 新パラメータの追加
  - summarize_wal, wal_summary_keep_time などのパラメータ追加
- Windows 環境で利用可能
  - pg_rman などは非対応だった Windows 環境でもバックアップ可能
    - pg_rman: 増分バックアップを可能にするサードパーティモジュール

pg_basebackup による増分バックアップ

- 実行環境(例)

領域	パス	備考
データベースクラスタ	data	
WAL領域	data/pg_wal	
バックアップ格納先	backup	フルバックアップ、 増分バックアップを保存

- パラメータ

パラメータ名	設定値	備考
summarize_wal	on(off がデフォルト)	onでないと 増分バックアップ不可
wal_level	replica(デフォルト) 以上	summarize_wal が on の時 は replica 以上が必須

フルバックアップ取得前のテーブル状態

- full_backup というテーブルがあるだけの状態
  - 増分バックアップを取得前にテーブルを追加し
    任意の時点に戻れるか検証する

```
$ psql
psql (17.6)
"help"でヘルプを表示します。
postgres=# ¥d
             リレーション一覧
 スキーマ |    名前     | タイプ | 所有者
----------+-------------+----------+----------
 public   | full_backup | テーブル | postgres
(1 行)
```

pg_basebackup による増分バックアップ

フルバックアップ取得→増分バックアップ取得→統合で実施

- フルバックアップを取得

```
# -P オプションでサイズや進捗を確認可能
$ pg_basebackup  -D backup/full -P
23818/23818 kB (100%), 1/1 テーブル空間
```

- テストテーブル inc1 を作成

```
postgres=# create table inc1 (i int);
CREATE TABLE
postgres=# ¥d
        リレーション一覧
 スキーマ | 名前 |  タイプ  |  所有者
----------+------+----------+----------
 public   | full_backup | テーブル | postgres
 public   | inc1 | テーブル | postgres
```

pg_basebackup による増分バックアップ

- 一回目の増分バックアップを取得

```
$ pg_basebackup --incremental=backup/fullbackup_manifest -D backup/inc1
```

  - --incrementalで直前のバックアップマニフェストファイルを参照する
- inc2 テーブルを作成後二回目の増分バックアップを取得

```
postgres=# create table inc2 (i int);
CREATE TABLE
$ pg_basebackup --incremental=backup/inc1backup_manifest -D backup/inc2
```

  - 注意点：前回バックアップの backup_manifest を参照する

増分バックアップの base ディレクトリ

- 増分バックアップの中身

```
# トップディレクトリは通常のバックアップと変わりなし
$ ls backup/inc1/
PG_VERSION          pg_commit_ts    pg_replslot         pg_twophase
backup_label        pg_dynshmem     pg_serial           pg_wal
backup_manifest     pg_hba.conf     pg_snapshots        pg_xact
base                pg_ident.conf   pg_stat             postgresql.auto.conf
current_logfiles    pg_logical      pg_stat_tmp         postgresql.conf
global              pg_multixact    pg_subtrans
log                 pg_notify       pg_tblspc

# base 以下に増分バックアップ特有のファイルがあることから増分と判断できる
$ ls backup/inc1/base/1/
1247_fsm   4163            INCREMENTAL.2652   INCREMENTAL.3394
1249_fsm   4165            INCREMENTAL.2653   INCREMENTAL.3394_vm
1255_fsm   4167            INCREMENTAL.2654   INCREMENTAL.3395
1259_fsm   4169            INCREMENTAL.2655   INCREMENTAL.3431
```

増分バックアップの統合とリストア

- pg_combinebackup でフルバックアップ～任意の増分地点まで統合
  - pg_verifybackup で検証もセットで行うこと

```
# フルバックアップからinc1(inc1 テーブルがある地点)まで
$ pg_combinebackup -o backup/combine_full_inc1 backup/full/ backup/inc1

# フルバックアップからinc2(inc1, inc2 テーブルがある地点)まで
$ pg_combinebackup -o backup/combine_full_inc2 backup/full/ backup/inc1 backup/inc2/

# 各統合バックアップを pg_verifybackup で検証
$ pg_verifybackup backup/combine_full_inc1
バックアップが正常に検証されました
$ pg_verifybackup backup/combine_full_inc2
バックアップが正常に検証されました
```

バックアップ内容

- 各地点でのバックアップが取れていることを確認

```
# postgresql.conf の port 番号をずらして各クラスタを起動
$ pg_ctl start -D backup/combine_full_inc1/
$ psql –p 5433
postgres=# ¥d
          リレーション一覧
 スキーマ｜   名前    ｜ タイプ ｜ 所有者
----------+-------------+----------+----------
 public   | full_backup | テーブル | postgres
 public   | inc1        | テーブル | postgres

$ pg_ctl start -D backup/combine_full_inc2/
$ psql –p 5434
postgres=# ¥d
          リレーション一覧
 スキーマ｜   名前    ｜ タイプ ｜ 所有者
----------+-------------+----------+----------
 public   | full_backup | テーブル | postgres
 public   | inc1        | テーブル | postgres
 public   | inc2        | テーブル | postgres
```

ディレクトリサイズと実行時間

- フルバックアップに比べてサイズは小さい

```
$ du -h  --max-depth=1 backup/
888M    backup/full
6M    backup/inc1
4M    backup/inc2
```

- 実行時間もフルバックアップに比べて短い

```
# 1000 行持つ inc3 テーブル作成後に実施
# 実行されているファイルサイズが小さく完了している
$ time pg_basebackup -D backup/inc_3 --incremental=backup/inc_2backup_manifest -c fast -P
  4045/888986 kB (100%), 1/1 テーブル空間
real    0m4.391s
$ time pg_basebackup -D backup/full_3 -c fast -P
888986/888986 kB (100%), 1/1 テーブル空間
real    0m32.777s
```

増分バックアップ利用時の注意点①

- summarize_walをonにすること(デフォルトで off)
  - $PGDATA/pg_wal/summaries/ 以下のファイルが作成されるようになる

```
# summarize_wal を on にしてテーブル作成
postgres=# create table summary_test(i int);
CREATE TABLE

# サマリファイルが作成される
$ ls -l data/pg_wal/summaries/
合計 8
-rw-------. 1 postgres postgres 480 10月 21 14:07
000000010000000010000060000000001001B0E8.summary

# summarize_wal が off で取得したフルバックアップを対象とするとエラー
$ pg_basebackup --incremental=backup/fullbackup_manifest -D backup/inc1
pg_basebackup: エラー: ベースバックアップを開始できませんでした:
ERROR:  WAL集約が有効でなければ差分バックアップは取得できません
```

- wal_level を replica 以上とすること
  - summarize_wal が on の時は wal_level を replica 以上が必須

増分バックアップ利用時の注意点②

- wal_summary_keep_timeを増分バックアップ取得間隔より長くすること
  - summary ファイルを保存する期間を制御するパラメータ
  - この期間を過ぎると増分バックアップは失敗する

```
#wal_summary_keep_time = ‘1d’ として 1 日ごとに古い summary を削除するよう設定
$ ls –l  data/pg_wal/summaries/
-rw-------. 1 postgres postgres 32 10月 30 16:52
000000010000001CF000028000000001D0000028.summary

# 2 日開けて増分バックアップを取得しようとすると失敗
$ pg_basebackup -D backup/inc_1 --incremental=backup/fullbackup_manifest -P -c fast
pg_basebackup: エラー: ベースバックアップを開始できませんでした:
ERROR:  WAL集計がタイムライン1上の1/CD000060から1/CF000028まで必要ですが、そのタイムライン上のそのLSN範囲での集計は不完全です

DETAIL:  この範囲で集計されていない最初のLSNは1/CD000060です。
```

増分バックアップ利用時の注意点③

- フルバックアップ～最新増分バックアップはすべて保管すること
  - pg_combinebackup 時にフルバックアップと
    初回増分バックアップ～指定したい時点までの増分バックアップが
    すべてないとエラー

```
# 最初の引数にフルバックアップを指定せずに実行
$ pg_combinebackup -o backup/combine_test backup/inc1 backup/inc2
pg_combinebackup: エラー: “backup/inc1"のバックアップは差分バックアップですが、最初のバックアップはフルバックアップである必要があります

# backup/inc_1 を引数に渡さず実行
# LSN 位置に飛びが見られエラーとなる
$ pg_combinebackup -o backup/combine_1 backup/full/ backup/inc_2
pg_combinebackup: エラー: “backup/full/"のバックアップはLSN 1/1A000060で始まっていますが、1/1C000028を期待していました
```

増分バックアップ利用時の注意点④

- 統合したバックアップを作成してから古いバックアップを削除する
  - (従来)定期的にフルバックアップを取得しそれ以前のバックアップを削除
  - pg_combinebackup を実行してそれ以前のバックアップを削除する
    - pg_combinebackup したバックアップはフルバックアップとして利用可能
    - 統合後も pg_verifybackup で検証することを忘れずに

```
# combine_full_inc1 が full ~ inc_1 まで統合したバックアップ
$ ls -l backup/
合計 16
drwx------. 20 postgres postgres 4096 10月 21 16:44 combine_full_inc1
drwx------. 20 postgres postgres 4096 10月 21 16:43 full
drwx------. 20 postgres postgres 4096 10月 21 16:43 inc_1
drwx------. 20 postgres postgres 4096 10月 21 16:44 inc_2

# pg_combinebackup したらそれ以前のバックアップは削除してよい
$ pg_combinebackup -o backup/combine_full_inc2 backup/combine_full_inc1/ backup/inc_2

$ pg_verifybackup backup/combine_full_inc2
バックアップが正常に検証されました
```

pg_combinebackup オプション①

- 高速化オプション

オプション	説明	環境	メリット
--copy	デフォルト ファイルをコピー		
--copy-file-range	copy_file_rangeを利用した ファイル部分コピー	FreeBSDか カーネル4.5+の Linux	カーネル空間で 完結するため高速
--clone	Reflink機能での ファイルクローンコピー	カーネル 4.5+の Btrfs、Reflinkが有効なXFS macOS の APFS	Inodeと ブロックマップ コピーのみのため高速

- 注意: --copy-file-range, --clone は入出力ファイルが同一ファイルシステムである必要有

```
$ pg_combinebackup -o /mnt2/combine_1 /mnt1/full/ /mn1/inc_1 --clone

pg_combinebackup: エラー: "/mnt1/inc_1/pg_wal/000000010000000100000B9"の
"mnt2/combine3/pg_wal/000000010000000100000B9"へのファイル範囲のコピー中のエラー:
無効なクロスデバイスリンクです
```

pg_combinebackup オプション②

- バックアップ統合対象のオプション(-N, --no-sync は無効である必要有)

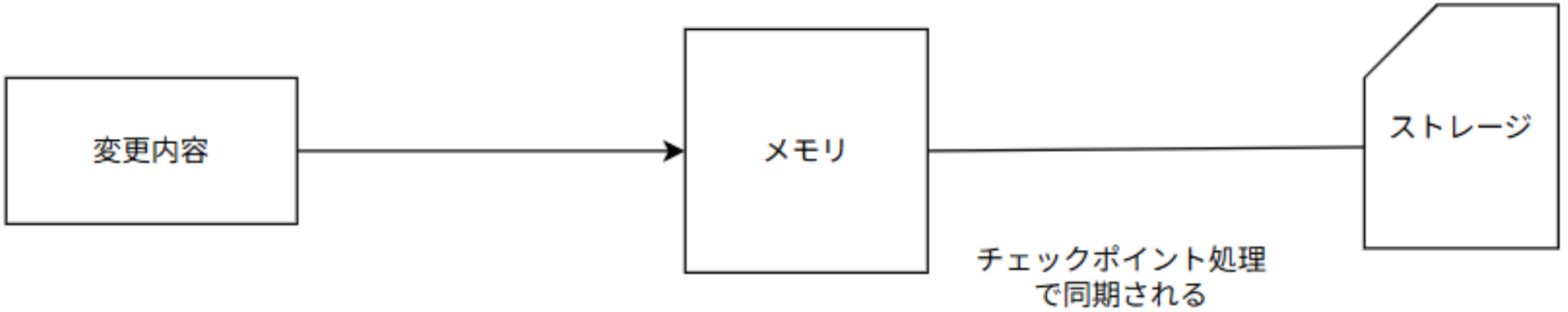
オプション	バックアップ統合の対象	メリット
--fsync	デフォルト バックアップディレクトリ内全ファイル	最小限の バックアップで済む
--plain	WALファイルと テーブル空間全体の シンボリックリンクまで	左記を バックアップに含められる
--syncfs	バックアップディレクトリを含む ファイルシステム全体	ファイル一つ一つを 開かないため統合が早い

  - --syncfs はファイルシステムに他アプリの書き込みがあるときは注意
    - PostgreSQL 以外の書き込みもバックアップに含まれてしまう
- -N, --no-sync: ディスク書き込み前にオペレーションを返す
  - 処理は早いが同期中にクラッシュするとデータを失う恐れがある
- -k, --link オプション(v18 以降)
  - コピーはせずハードリンクを作成する
  - 実コピーが伴わないため高速

リストア手順

その前にPostgreSQLのデータ保存の仕組みをおさらい

- データの変更内容はストレージではなくメモリに書き込む
  - ストレージに書き込む処理は高価であるため

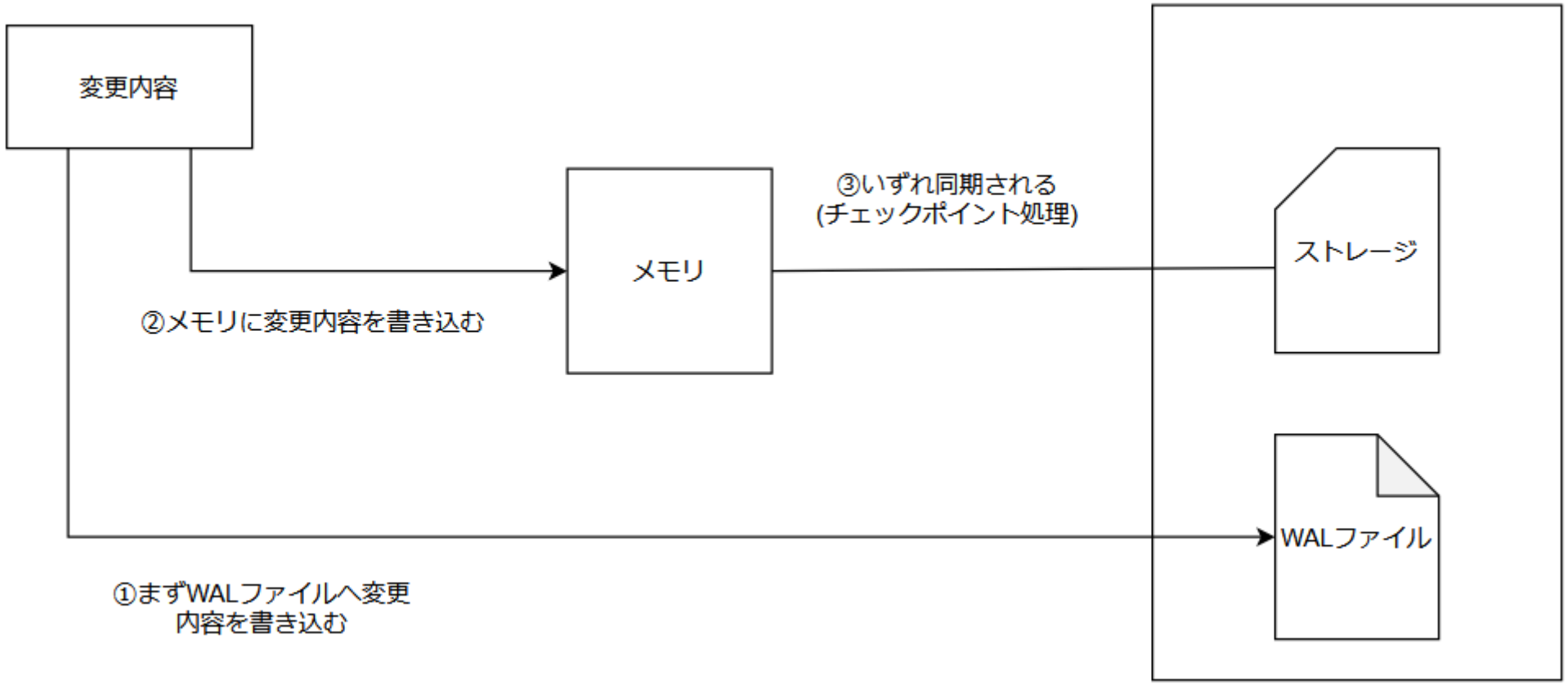


→同期前に障害が起きるとメモリの変更内容が消えてしまう

→メモリ内容が消えてもデータを損失しない仕組みが必要

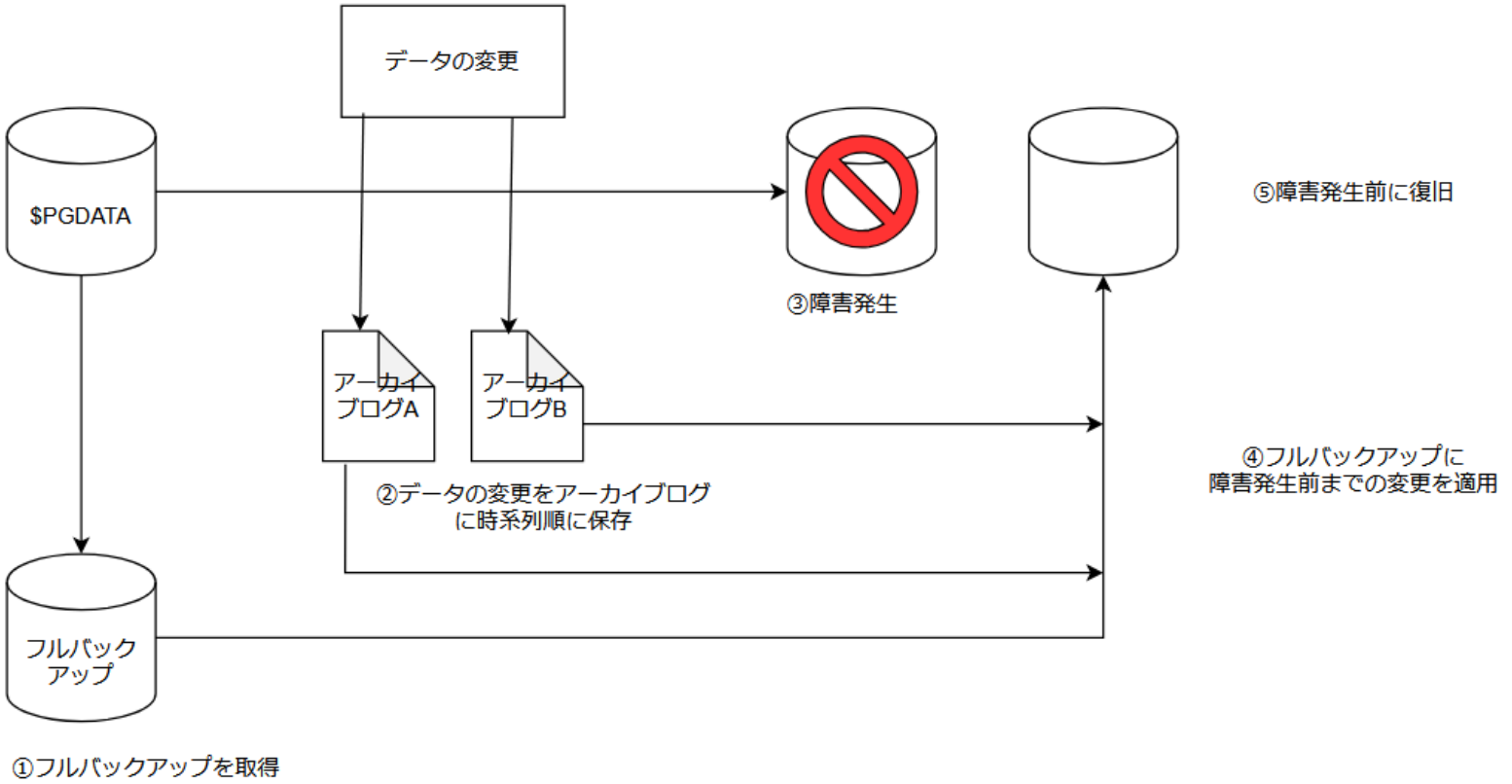
WALファイル

- メモリに書き込む前にWALファイルへ保存
  - 障害が起きたらWALファイルの変更内容を読んで復旧



Point In Time Recovery

- フルバックアップとWALファイルのアーカイブログを掛け合わせて任意の時点の状態に復旧できる仕組み



PITR と増分バックアップの違い

- PITR に必要なもの
  - フルバックアップ
  - WALアーカイブログ
    - フルバックアップとリカバリしたい地点までの WAL を適用してリストア

- 増分バックアップに必要なもの
  - フルバックアップ
  - 初回フルバックアップからの増分バックアップファイル
    - フルバックアップ～任意の時点までの増分ファイルを統合してリストア
    - WALを管理したい場合は別途設定が必要

PITR と増分バックアップは両立可

- PITR と 増分バックアップの両立
  - archive_mode, archive_command を設定して
    WALアーカイブログを補完する仕組みを設ければよい

- メリット
  - 増分バックアップタイミング以外へ任意の時刻にリカバリできる
    - PITR 機能の強み
  - PITRのみよりリカバリ時間を短縮できる
    - 増分バックアップの強み
      フルバックアップ～増分バックアップによる高速なリストア

増分バックアップにおけるPITR設定

- バックアップ元のPGDATAで postgresql.conf を設定

```
#wal をアーカイブする設定
#%p は保管するWALファイルの相対パス, %f がWALのファイル名に置き換わる
archive_mode = on
archive_command = ‘cp “%p” “/pg_arc/%f”’

#restore を行う設定
restore_command = ‘cp “/pg_arc/%f” “%p” ’
```

- 増分バックアップを取得

```
# フルバックアップ～バックアップの取得・統合、検証まで実施
$ pg_basebackup  -D backup/full
$ pg_basebackup --incremental=backup/fullbackup_manifest -D backup/inc1
$ pg_combinebackup -o backup/combine_full_inc1 backup/full/ backup/inc1
$ pg_verifybackup backup/combine_full_inc1
```

障害から復旧

- 従来の PITR と同様に復旧可能

```
# データを移動して障害を疑似的に起こす
$ mv data data_crash

# 統合したバックアップを元ディレクトリにコピー
$ cp –r databackup/combine_full_inc1 data

# リカバリモードで起動するよう指示するrecovery.signal を作成して起動
$ touch data/recovery.signal

# 必要に応じて postgresql.conf を設定し復旧タイミングを指定
recovery_target_time = ‘2025-10-20 14:05:00'

$ pg_ctl start

# ログに以下が出力されていれば復旧完了
LOG: starting archive recovery
LOG: archive recovery complete
```

まとめ

実行例

【実行例】

- バックアップ設計
  - 例:週一回のバックアップの統合＋毎日の増分バックアップ
  - PITR も利用してより子細なタイミングへのリストアを可能に
- バックアップ取得の自動化
  - 初回フルバックアップ以降の増分バックアップを自動化
  - cronやsystemdタイマー等で
    pg_basebackup --incremental コマンドを定期実行
- リストア検証
  - 例:月一回は実際にリストアテストを実施
  - pg_combinebackupとpg_verifybackupで検証

増分バックアップのメリットと注意点

- メリット
  - ストレージコストの削減
  - バックアップ・リストア時間の短縮
  - 運用負荷の軽減
- 注意点
  - summarize_wal , wal_keep_timeの有効化
    archive_mode の replica 以上等の設定
  - pg_verifybackup による定期検証
  - 増分バックアップの保管
    - pg_combinebackup した以前のバックアップのみ削除するようにする
  - 統合バックアップ・古いバックアップ両方を
    保存できるストレージサイズが必要
    - 統合直後は一時的に古いバックアップもストレージに残る

会社紹介

株式会社SRA OSS

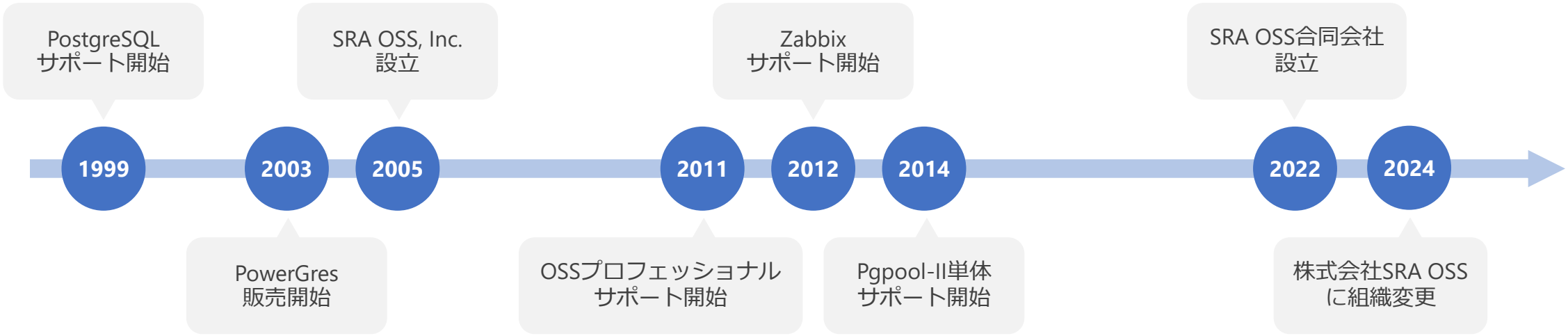
所 在 地: 東京都豊島区南池袋2-32-8
設 立 日: 2022年6月17日
株　　主: 株式会社SRA
　　株式会社NTTデータ
資 本 金: 7,000万円
社　　長: 稲葉 香理

事業内容

- オープンソースソフトウェア (OSS) 関連のサポート、製品開発・販売、構築・コンサル
- OSSの教育、開発、コミュニティ運営支援
- ソフトウェアの研究開発

顧　　問: 石井達夫

技術顧問: 増永 良文 (お茶の水女子大学名誉教授)



ご清聴ありがとうございました。