PostgreSQL バックアップ実践と バックアップ管理ツールの紹介

2021/11/12 PostgreSQL Conference Japan 2021
NTT OSSセンタ 黒田 佳祐

本資料の情報は、Linux環境におけるPostgreSQL 13系での検証結果をもとにしています

自己紹介

- NTT OSSセンタ 黒田佳祐(クロダケイスケ)
- PostgreSQL 技術検証・開発・サポート業務に従事
- はじめてさわったPostgreSQL バージョン8.3
- 好きなPostgreSQLの機能 ストリーミングレプリケーション

バックアップ業務の全体像

- バックアップ設計
  - 取得タイミング、保管期間、保管容量、手法などの決定
- **バックアップ実践**
  - **バックアップ実行手順の作成と実行**
- バックアップ試験・運用
  - 正常かつ目標時間内に取得・リカバリが可能か

本セッションのスコープ

セッション概要

- PostgreSQLにおけるバックアップ対象
- 論理バックアップとリストアの実践
- 物理バックアップとリストア、リカバリの実践
- バックアップ管理ツールの紹介

セッション目標

- PostgreSQLのバックアップ対象がわかる
- 論理バックアップ、物理バックアップから復旧する手順が理解できる
- どのようなバックアップ管理ツールがあるか知る

論理バックアップと
物理バックアップ

論理バックアップ

- データベースのデータ定義およびデータをバックアップ
- リストア可能なのはバックアップ取得時点のみ
- pg_dump/pg_dumpall で取得、オンラインで実行可能

物理バックアップ

- データベースを構成するディレクトリ・ファイルをバックアップ
- リストア後にWALを用いてリカバリするため、
  バックアップ時点以降のWALがあれば任意の時点に復旧可能
- pg_basebackup で取得、オンラインで実行可能

論理と物理の使い分け

	論理バックアップ	物理バックアップ
起動中の取得	可能	可能
バックアップ取得時間 および復旧時間	物理バックアップより長い傾向 データ量の増加により長時間化	論理バックアップより短い傾向 ファイル数・サイズの増加により長時間化
取得手順と復旧手順	シンプル	複雑
復旧可能地点	バックアップ取得時点のみ	バックアップ取得時点以降で、 適用可能なWALがあれば 任意の時点に復旧可能
異なるバージョンへの移行	可能 (データ加工が必要な場合あり)	不可

バックアップ対象

- データベースがすべて失われた時、バックアップから
  復旧できることを目標として必要なバックアップ対象を整理

物理バックアップの対象

- データベースクラスタを構成するすべてのファイルとアーカイブWAL

データベースクラスタとは

- 同一クライアント接続先(インスタンス)で管理される
  データベースのデータの格納場所(ディレクトリ)
- このディレクトリ配下に、各種データファイルが格納される
- PostgreSQLは、ひとつのデータベースクラスタ内に
  複数のデータベースを持つ

データベースクラスタとは

- デフォルトのディレクトリは /var/lib/pgsql/data
- 各種コマンドの -D オプションでディレクトリを指定する
- -D オプション省略時は、環境変数 PGDATA の情報が利用される
- 本資料では /home/postgres/pg13/pgdata1 を指定

データベースクラスタ全体像

```
/home/postgres/pg13/pgdata1
|-- PG_VERSION
|-- base
|-- current_logfiles
|-- global
|-- pg_commit_ts
|-- pg_dynshmem
|-- pg_hba.conf
|-- pg_ident.conf
|-- pg_logical
|-- pg_multixact
|-- pg_notify
|-- pg_replslot
|-- pg_serial
|-- pg_snapshots
|-- pg_stat
|-- pg_stat_tmp
|-- pg_subtrans
|-- pg_tblspc
|-- pg_twophase
|-- pg_wal
|-- pg_xact
|-- postgresql.auto.conf
|-- postgresql.conf
|-- postmaster.opts
`-- postmaster.pid
```

- これらのディレクトリ・ファイルがすべて
- 一部ディレクトリは
  シンボリックリンクにより実体を
  別領域に配置することも可能、
  テーブルスペースは
  この仕組みを利用している

データベースクラスタ全体像

```
pg_tblspc
`-- 16400 -> /home/postgres/pg13/ts1
```

- pg_tblspc ディレクトリは
  テーブルスペース先への
  シンボリックリンクを持つ

- テーブルスペース内に作成された
  オブジェクトファイルの実体は
  /home/postgres/pg13/ts1 に格納される

データベースクラスタ全体像

```
/home/postgres/pg13/pgdata1
|-- PG_VERSION
|-- base
|-- current_logfiles
|-- global
|-- pg_commit_ts
|-- pg_dynshmem
|-- pg_hba.conf
|-- pg_ident.conf
|-- pg_logical
|-- pg_multixact
|-- pg_notify
|-- pg_replslot
|-- pg_serial
|-- pg_snapshots
|-- pg_stat
|-- pg_stat_tmp
|-- pg_subtrans
|-- pg_tblspc
|-- pg_twophase
|-- pg_wal -> /home/postgres/pg13/pg_wal1
|-- pg_xact
|-- postgresql.auto.conf
|-- postgresql.conf
|-- postmaster.opts
`-- postmaster.pid
```

- pg_walディレクトリも、initdbによるデータベースクラスタ作成時にWAL格納ディレクトリを指定すると、シンボリックリンクになりWALの実体を異なる領域に格納する
- **/home/postgres/pg13/pg_wal1** に格納

アーカイブWAL

```
/home/postgres/pg13/pgdata1   -- データベースクラスタ
/home/postgres/pg13/pg_wal1   -- WALディレクトリ
/home/postgres/pg13/ts1   -- テーブルスペースディレクトリ
/home/postgres/pg13/pgarch1   -- アーカイブWALディレクトリ
```

アーカイブWAL

データベースクラスタ

/home/postgres/pg13/pgdata1

アーカイブWALディレクトリ

/home/postgres/pg13/pgarch1

- アーカイブWALは、データベースクラスタとは異なるディレクトリに保存
- アーカイブWALも、物理バックアップからの復旧に必須
- ここでは同一サーバ上に保存しているが、実際のバックアップ運用では、PostgreSQL本体とは異なるサーバ・ストレージに保存することを推奨

論理バックアップの対象

- データベースクラスタ全体で共有するデータ
  - Role(User)
  - Tablespace
- 各データベース定義および各データベースが保持する
  オブジェクト定義とデータ
  - Database
  - Table
  - View(定義のみ)
  - Index
  - Schema
  - Extension
  - Sequence
  - Function

論理バックアップの対象

- 以下の設定ファイルは論理バックアップに含まれないため、
  別途ファイルベースでバックアップしておく必要あり
  - postgresql.auto.conf
  - postgresql.conf
  - pg_hba.conf

- その他、論理バックアップに含まれず、復旧時にリストアできない情報が存在するため注意が必要
- 論理バックアップの対象に、リストアに必要な情報が含まれているかどうかは環境や要件に応じて確認が必須
  - レプリケーションスロットの情報は含まれないため手動復旧が必要
  - データベースクラスタを再作成した場合、
    各オブジェクトに割り当てられるOIDをそのままリストアすることはできない

バックアップ実践

利用アプリケーション・手順

- PostgreSQL 本体付属のクライアントアプリケーションを利用
- 論理バックアップ は **pg_dump・pg_dumpall**
- 論理バックアップからのリストア は **pg_restore**
- 物理バックアップ は **pg_basebackup**
- 物理バックアップからのリストア・リカバリ は OSコマンド

利用アプリケーション・手順

- 論理バックアップは SQLでのデータ抽出、
  物理バックアップは rsync コマンドと組み合わせた
  バックアップといった、異なる手法も複数存在する

- 今回は最も汎用的なアプリケーション・手順での実践例を紹介

検証環境

OS	Rocky Linux 8.4
PostgreSQLバージョン	PostgreSQL 13.4
データベースクラスタ(PGDATA)	/home/postgres/pg13/pgdata1
テーブルスペース	/home/postgres/pg13/ts1
WAL	/home/postgres/pg13/pg_wal1
アーカイブWAL	/home/postgres/pg13/pgarch1

表記

- [postgres]$ -- OS の postgres ユーザで実行
- [root]#   -- OS の root ユーザで実行

論理バックアップシナリオ

- pg_dumpall で各データベースのデータも取得可能だが、
  データベースの単位データはpg_dumpで取得した方が
  データベース個別のリストアなど柔軟に対応可能

- pg_dumpall では、データベースクラスタ全体で共有の情報、
  つまりロールとテーブルスペース情報のみ収集するのが定番

- データベースがすべて破損した時、
  論理バックアップから復旧する手順を想定

論理バックアップシナリオ

1. 設定ファイルのバックアップ
2. pg_dumpall によるバックアップ
3. pg_dump によるバックアップ
4. データ全損、ディレクトリの復旧
5. データベースクラスタ・設定ファイルの復旧と起動
6. pg_dumpall バックアップのリストア
7. pg_restore によるリストア
8. 統計情報収集

1. 設定ファイルのバックアップ

```
[postgres]$
mkdir /home/postgres/pg13/pgdata1_bk

cd /home/postgres/pg13
cp -p pgdata1/postgresql.conf pgdata1_bk/postgresql.conf
cp -p pgdata1/postgresql.auto.conf pgdata1_bk/postgresql.auto.conf
cp -p pgdata1/pg_hba.conf pgdata1_bk/pg_hba.conf
```

- 設定ファイルはファイルベースでバックアップ取得
  - サーバ破損に備えて別サーバに保管すべき、設定変更時は必ずバックアップ

2. pg_dumpall によるバックアップ

```
[postgres]$
pg_dumpall -U postgres -h localhost -p 5432 ¥
  --globals-only --file=pg_dumpall_dbcluster1.dmp

--globals-only -- データベースクラスタ全体で共有する
                  ロール情報およびテーブルスペース情報のみダンプ
-U postgres    -- 論理バックアップはスーパーユーザであるpostgresユーザで取得
-h localhost   -- 接続先ホスト、今回はローカルホスト上のPostgreSQLに接続
-p 5432        -- 接続先ポート、PostgreSQLのデフォルトポートは5432
```

- ロール情報、テーブルスペース情報の取得

3. pg_dump によるバックアップ

```
[postgres]$
pg_dump -U postgres -h localhost -p 5432 ¥
  --format=custom -d postgres --file=pg_dump_postgres.dmp

pg_dump -U postgres -h localhost -p 5432 ¥
  --format=custom -d db1 --file=pg_dump_db1.dmp
```

format=custom -- カスタム形式アーカイブのダンプファイルを出力。デフォルトで圧縮され、
pg_restoreによるテーブル単位のリストアなど柔軟な操作が可能な形式

- postgres データベースと、db1 データベースのバックアップを個別に取得

4. データ全損、ディレクトリ再作成

```
[postgres]$
rm -f -r /home/postgres/pg13/pgdata1
rm -f -r /home/postgres/pg13/pg_wal1
rm -f -r /home/postgres/pg13/ts1
rm -f -r /home/postgres/pg13/pgarch1

mkdir -p /home/postgres/pg13/pgdata1 /home/postgres/pg13/pgarch1 ¥
  /home/postgres/pg13/pg_wal1 /home/postgres/pg13/ts1

chmod 700 /home/postgres/pg13/pgdata1 /home/postgres/pg13/pgarch1 ¥
  /home/postgres/pg13/pg_wal1 /home/postgres/pg13/ts1
```

- データベースがすべて破損したことを想定しデータを削除後、各ディレクトリを再作成

5. データベースクラスタ・設定ファイルの復旧と起動

```
[postgres]$
initdb -D /home/postgres/pg13/pgdata1 -X /home/postgres/pg13/pg_wal1 ¥
  --encoding=UTF8 --no-locale --data-checksums

cd /home/postgres/pg13
cp -p pgdata1_bk/postgresql.conf pgdata1/postgresql.conf
cp -p pgdata1_bk/postgresql.auto.conf pgdata1/postgresql.auto.conf
cp -p pgdata1_bk/pg_hba.conf pgdata1/pg_hba.conf

pg_ctl start
```

- データベースクラスタを initdb により生成した後、
  設定ファイルをリストアして PostgreSQL を起動

6. pg_dumpall バックアップのリストア

```
[postgres]$
psql -U postgres -h localhost -p 5432 -d postgres -f pg_dumpall_dbcluster1.dmp
```

postgresロールは、initdb時点で作成されているため、リストア時に
「postgresロールが既に存在する」というERRORが発生するが、無視して問題ないため対処不要

```
psql:pg_dumpall_dbcluster1.dmp:16: ERROR:  role "postgres" already exists
```

- pg_dumpall で取得したロール情報とテーブルスペース情報のリストア

7. pg_restore によるリストア

```
[postgres]$
pg_restore -U postgres -h localhost -p 5432 --format=custom ¥
  --clean --create --if-exists -d template1 pg_dump_postgres.dmp

pg_restore -U postgres -h localhost -p 5432 --format=custom ¥
  --clean --create --if-exists -d template1 pg_dump_db1.dmp
```

clean,create,if-exists -- これらのオプションを同時に指定すると、リストア処理開始時に、
リストア対象データベースが存在する場合は削除後再作成、存在しない場合は作成する。
postgresデータベースは再作成、db1データベースは新規作成されることになる。

-d template1 -- 上記オプションにより、リストア対象データベースを削除後再作成するためには、
pg_restoreを開始する時点での接続先には、リストア対象とは異なるデータベースを
指定する必要がある。デフォルトで用意されているtemplate1を指定。
template1は接続先となるだけで、リストア自体はダンプが持つリストア対象データベースである、
postgresおよびdb1に対して実行される。

8. 統計情報収集

```
[postgres]$
psql -c "ANALYZE" postgres
psql -c "ANALYZE" db1
```

- データリストア後は、データベース全体に対してANALYZEを手動実行して、すみやかに統計情報を最新化しておくことを推奨
- データの正常性確認は実際の各テーブルの状況を確認する必要あり

pg_basebackup

- 物理バックアップを取得するためのクライアントアプリケーション
- データベースクラスタにくわえて、
  バックアップ中に生成されたWALの収集が可能
- バックアップ時点への復旧には、pg_basebackupで取得した
  データベースクラスタと、バックアップ中に生成されたWALが必須

pg_basebackup

- 故障発生時点直前までリカバリしたい場合、
  pg_basebackup 取得時点から故障発生時点直前までのWALが必須。
  故障が発生したデータベースクラスタのアーカイブWALと、
  データベースクラスタに残った未アーカイブのWALが必要となる

pg_basebackup

- デフォルトで --xlog-method=stream オプションにより、
  バックアップ中に生成されたWALをバックアップ取得と並行収集する

- --xlog-method=fetch を指定した場合、バックアップ完了後に
  WALの収集を開始するため、バックアップ中に必要なWALが
  リサイクルされてしまうと、pg_basebackup は失敗する

- --xlog-method=fetch は、9.6以前の -x(--xlog)オプション指定時と
  同じ動作のため、9.6以前の pg_basebackup では、
  WAL収集失敗によるエラー発生事例が多い

pg_basebackupのバックアップ

pg_basebackup は、
データベースクラスタにくわえて
バックアップ中の更新データを含む
WALを収集

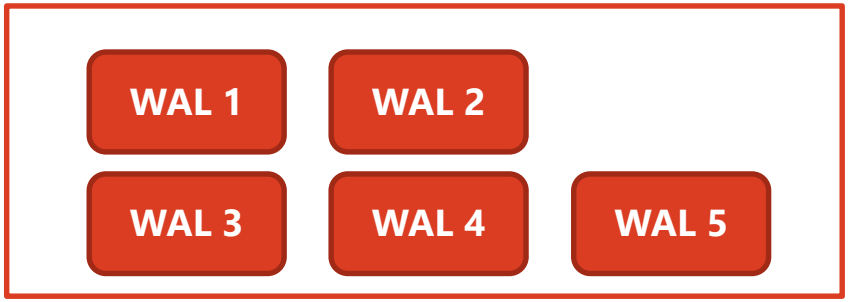
これにより、バックアップ時点であれば
このバックアップのみで復旧できる
スタンドアロンバックアップとなっている

pg_basebackupとアーカイブWAL

稼働中のPostgreSQL

PostgreSQLは、データ更新時にWALに更新内容を書き込み、書き込み終わったWALが順次アーカイブされる

pg_basebackup



アーカイブWAL

pg_basebackupのリストア・リカバリ

稼働中PostgreSQLの
WAL(未アーカイブ)

稼働中のデータベースクラスタが破損した場合、
・pg_basebackup
・アーカイブWAL
・稼働中の未アーカイブWAL
を組み合わせることで、
WAL 8 時点までリカバリ可能

pg_basebackup

アーカイブWAL

物理バックアップシナリオ 1

- WALをふくめてすべてのデータが破損した時、
  物理バックアップ取得時点に復旧するシナリオ

物理バックアップシナリオ 1

1. pg_basebackup によるバックアップ
2. データ全損、ディレクトリの復旧
3. バックアップの展開
4. 起動および統計情報収集

1. pg_basebackup によるバックアップ

```
[postgres]$
pg_basebackup -U postgres -h localhost -p 5432 ¥
  -D /home/postgres/pg13/pg_basebackup --format=tar --gzip

-D /home/postgres/pg13/pg_basebackup -- pg_basebackupによるバックアップ取得先
--format=tar -- tar形式での取得。テーブルスペースやWALを個別にtar形式で取得する
--gzip       -- tarファイルをgzip形式で圧縮
```

- pg_basebackup による物理バックアップの取得
- テーブルスペースを含むデータベースクラスタに対して、
  同じサーバ上でpg_basebackupを実行すると、
  稼働中のテーブルスペースディレクトリにバックアップを
  取得しようとして競合するため、tar形式で取得する必要がある

1. pg_basebackup によるバックアップ

```
[postgres]$
ls -l /home/postgres/pg13/pg_basebackup
total 4280
-rw-------. 1 postgres postgres     484 Oct 19 10:11 16387.tar.gz
-rw-------. 1 postgres postgres  178364 Oct 19 10:11 backup_manifest
-rw-------. 1 postgres postgres 4177222 Oct 19 10:11 base.tar.gz
-rw-------. 1 postgres postgres   17064 Oct 19 10:11 pg_wal.tar.gz

16387.tar.gz  -- テーブルスペースディレクトリのバックアップ(自動的に割り振られるOID名)
base.tar.gz   -- データベースクラスタのバックアップ
pg_wal.tar.gz -- 同時に収集したWALのバックアップ

backup_manifest -- バックアップデータの検証用ファイル
```

2. データ全損、ディレクトリ再作成

```
[postgres]$
rm -f -r /home/postgres/pg13/pgdata1
rm -f -r /home/postgres/pg13/pg_wal1
rm -f -r /home/postgres/pg13/ts1
rm -f -r /home/postgres/pg13/pgarch1

mkdir -p /home/postgres/pg13/pgdata1 /home/postgres/pg13/pgarch1 ¥
  /home/postgres/pg13/pg_wal1 /home/postgres/pg13/ts1

chmod 700 /home/postgres/pg13/pgdata1 /home/postgres/pg13/pgarch1 ¥
  /home/postgres/pg13/pg_wal1 /home/postgres/pg13/ts1
```

- データベースがすべて破損したことを想定しデータを削除後、各ディレクトリを再作成

3. バックアップの展開

```
[postgres]$
tar xvf base.tar.gz -C /home/postgres/pg13/pgdata1
tar xvf pg_wal.tar.gz -C /home/postgres/pg13/pg_wal1
tar xvf 16387.tar.gz -C /home/postgres/pg13/ts1

rm -r /home/postgres/pg13/pgdata1/pg_wal
ln -s /home/postgres/pg13/pg_wal1 /home/postgres/pg13/pgdata1/pg_wal
```

- 取得したバックアップをそれぞれのディレクトリに展開する
- pg_wal ディレクトリはシンボリックリンクとしてはリストアされず、手動でシンボリックリンクを再作成する必要がある

4. 起動および統計情報収集

```
[postgres]$
pg_ctl start

psql -c "ANALYZE" postgres
psql -c "ANALYZE" db1
```

- バックアップ取得時点であれば、リカバリ設定等は不要で、
  リストア後そのまま起動することが可能

- 物理バックアップからの復旧時も手動での統計情報収集を推奨

物理バックアップシナリオ 2

- 未アーカイブのWALを用いて、故障発生時点の直前まで
  リカバリするシナリオ

- データベースクラスタが破損しても、
  pg_walに残った未アーカイブWALが正常な状態で残っていれば、

  ・データベースクラスタのバックアップ
  ・アーカイブWAL
  ・pg_walに残った未アーカイブのWAL

  を用いることで、未アーカイブのWALが持つ更新情報まで復旧可能

物理バックアップシナリオ 2

1. pg_basebackup によるバックアップ
2. データ全損、ディレクトリの復旧
3. 未アーカイブWALの退避
4. バックアップの展開
5. 未アーカイブWALのリストア
6. アーカイブWALの確認
7. リストア設定追加、リカバリシグナルファイル作成
8. 起動および統計情報収集

1. pg_basebackup によるバックアップ

```
[postgres]$
pg_basebackup -U postgres -h localhost -p 5432 ¥
  -D /home/postgres/pg13/pg_basebackup --format=tar --gzip
```

1. pg_basebackup によるバックアップ

```
[postgres]$
ls -l /home/postgres/pg13/pg_basebackup
total 4280
-rw-------. 1 postgres postgres     484 Oct 19 10:11 16387.tar.gz
-rw-------. 1 postgres postgres  178364 Oct 19 10:11 backup_manifest
-rw-------. 1 postgres postgres 4177222 Oct 19 10:11 base.tar.gz
-rw-------. 1 postgres postgres   17064 Oct 19 10:11 pg_wal.tar.gz
```

2. データ全損、ディレクトリの復旧

```
[postgres]$
rm -f -r /home/postgres/pg13/pgdata1
rm -f -r /home/postgres/pg13/ts1

mkdir -p /home/postgres/pg13/pgdata1 /home/postgres/pg13/ts1
chmod 700 /home/postgres/pg13/pgdata1 /home/postgres/pg13/ts1
```

- データベースクラスタおよびテーブルスペース領域のデータは破損したとして削除しディレクトリを再作成

3. 未アーカイブWALの退避

```
[postgres]$
ll /home/postgres/pg13/pg_wal1
total 65540
-rw-------. 1 postgres postgres 16777216 Nov  1 15:58 000000010000000000000005
-rw-------. 1 postgres postgres      337 Nov  1 15:58 000000010000000000000005.00000028.backup
-rw-------. 1 postgres postgres 16777216 Nov  1 15:59 000000010000000000000007
-rw-------. 1 postgres postgres 16777216 Nov  1 15:58 000000010000000000000008
-rw-------. 1 postgres postgres 16777216 Nov  1 15:59 000000010000000000000009
drwx------. 2 postgres postgres       59 Nov  1 15:59 archive_status

mkdir /home/postgres/pg13/pg_wal1_tmp
mv /home/postgres/pg13/pg_wal1/* /home/postgres/pg13/pg_wal1_tmp
```

- pg_wal1 ディレクトリに残った、未アーカイブのWALは
  リカバリ時に使用するため退避しておく

4. バックアップの展開

```
[postgres]$
tar xvf base.tar.gz -C /home/postgres/pg13/pgdata1
tar xvf pg_wal.tar.gz -C /home/postgres/pg13/pg_wal1
tar xvf 16387.tar.gz -C /home/postgres/pg13/ts1

rm -r /home/postgres/pg13/pgdata1/pg_wal
ln -s /home/postgres/pg13/pg_wal1 /home/postgres/pg13/pgdata1/pg_wal
```

- 取得したバックアップをそれぞれのディレクトリに展開する
- pg_wal ディレクトリはシンボリックリンクとしてはリストアされず、手動でシンボリックリンクを再作成する必要がある

5. 未アーカイブWALのリストア

```
[postgres]$
cp -p -r /home/postgres/pg13/pg_wal1_tmp/* /home/postgres/pg13/pg_wal1/
```

- 退避しておいた未アーカイブのWALを読み込めるようにリストア

6. アーカイブWALの確認

```
[postgres]$
ll /home/postgres/pg13/pgarch1/
-rw-------. 1 postgres postgres 16777216 Nov  1 15:54 000000010000000000000001
-rw-------. 1 postgres postgres 16777216 Nov  1 15:56 000000010000000000000002
-rw-------. 1 postgres postgres 16777216 Nov  1 15:56 000000010000000000000003
-rw-------. 1 postgres postgres      337 Nov  1 15:56 000000010000000000000003.00000060.backup
-rw-------. 1 postgres postgres 16777216 Nov  1 15:58 000000010000000000000004
-rw-------. 1 postgres postgres 16777216 Nov  1 15:58 000000010000000000000005
-rw-------. 1 postgres postgres      337 Nov  1 15:58 000000010000000000000005.00000028.backup
-rw-------. 1 postgres postgres 16777216 Nov  1 15:59 000000010000000000000006
```

- リカバリ時に使用するアーカイブWALの確認
- 他サーバに保管している場合は、PostgreSQLが読み込めるディレクトリにアーカイブWALを展開しておく必要あり

7. リストア設定追加、リカバリシグナルファイル作成

```
[postgres]$
echo "restore_command = 'cp /home/postgres/pg13/pgarch1/%f "%p"'" ¥
  >> $PGDATA/postgresql.conf

touch $PGDATA/recovery.signal
```

- restore_command に アーカイブWALディレクトリを指定し、
  起動時のリカバリ処理でアーカイブWALを読み込ませる

- $PGDATAにrecovery.signal ファイルを作成してから起動すると、
  PostgreSQLはリカバリモードで起動する

8. 起動および統計情報収集

```
[postgres]$
pg_ctl start

psql -c "ANALYZE" postgres
psql -c "ANALYZE" db1
```

未アーカイブWALによる故障直前までの復旧

データベースクラスタ

稼働中PostgreSQLの
WAL(未アーカイブ)

データベースクラスタのコピー
(WAL 5 の途中まで適用済)に
アーカイブWAL 5,6
未アーカイブWAL 7
を適用することで
WAL 7 の更新状況まで復旧

データベースクラスタのコピー

pg_basebackup

アーカイブWAL

参考、主な起動時ログ確認ポイント

NTT

-- データベースクラスタが中断した時刻、つまりバックアップ取得時刻
"database system was interrupted; last known up at 2021-11-01 15:58:05 JST"

-- WALの適用により復旧できた、最も最近完了した、トランザクション完了時の時刻
"last completed transaction was at log time 2021-11-01 15:59:29.265732+09"

-- リカバリ処理完了後、タイムラインIDと呼ばれる分岐点を管理するIDが変更されたことを示す
"selected new timeline ID: 2"

-- リカバリ処理完了を示す
"archive recovery complete"

-- PostgreSQLが正常に起動し、接続を受け入れられる状況になったことを示す
"database system is ready to accept connections"

バックアップ管理ツール

バックアップの管理

- バックアップをスケジュール起動したり管理する仕組みを
  PostgreSQL本体は持っていない
- pg_basebackup の起動や、不要となったバックアップおよび
  アーカイブWALの削除などをスクリプト化して運用する必要がある
- 多くのデータベースクラスタのバックアップを
  管理する必要がある場合は煩雑

バックアップの管理

- バックアップ管理ツールを使うことで管理負荷を軽減できる
- 管理ツールであっても、物理バックアップによるリストア、リカバリを実施することに変わりはない
- 物理バックアップの仕組みを理解したうえで、さらにバックアップ管理ツール自体の習熟が必要になる点は注意が必要

バックアップ管理ツール

- 2020,2021年 に PostgreSQL News の Related Open Source タグで リリース情報が展開されたバックアップ管理ツールを紹介
- 上記に加えて、NTT OSSセンタが主要コントリビュータである pg_rmanも紹介
- いずれも、物理バックアップおよびアーカイブWALの管理が可能

バックアップ管理ツール

	pg_rman	Barman	pgBackRest	WAL-G
最新バージョン	1.3.13(2021/08/30)	2.14(2021/09/23)	2.35(2021/08/23)	1.1(2021/08/12)
対応PostgreSQL	9.6～13	8.3～13	8.3～14	明記なし
RPM	あり(Github)	あり (EDB REPOSITORY/ POSTGRESQL COMMON REPOSITORY)	あり (Crunchy REPOSITORY/ POSTGRESQL COMMON REPOSITORY)	なし
言語	C	Python	C	Go
ライセンス	The PostgreSQL License	GPLv3	The PostgreSQL License	Apache License 2.0

バックアップ管理ツール

- デファクトスタンダードとなっているツールは現状ない
- 増分バックアップやリソース制御機能を備えたツールもあるため、要件に合わせて選定するのが望ましい
- インストール、環境設定、バックアップリストアの手順を紹介

pg_rman

pg_rman インストール・環境設定

https://github.com/ossc-db/pg_rman/releases からRPMを入手

```
[root]#
rpm ivh pg_rman-1.3.13-1.pg13.rhel8.x86_64.rpm

[postgres]$
mkdir /home/postgres/pg13/pg_rman -- バックアップカタログ用ディレクトリ

-- バックアップカタログの初期化
pg_rman init -B /home/postgres/pg13/pg_rman ¥
 -D /home/postgres/pg13/pgdata1 -A /home/postgres/pg13/pgarch1
```

pg_rman バックアップ・リストア

```
[postgres]$
export BACKUP_PATH=/home/postgres/pg13/pg_rman
pg_rman backup --backup-mode=full --progress  -- バックアップ
pg_rman show      -- バックアップ一覧確認
pg_rman validate -- バックアップ検証(実行しないとリストア不可)

pg_ctl stop -m immediate

pg_rman restore -- リストア
pg_ctl start
```

- 既存データベースクラスタの状態確認や、
  recovery関連設定、recovery.signalの作成は自動的に行われる

Barman

Barman インストール・環境設定

NTT

```
https://ftp.postgresql.org/pub/repos/yum/common/redhat/rhel-8-x86_64/からRPMを入手

[root]#
rpm ivh barman-2.14-1.rhel8.noarch.rpm
rpm ivh barman-cli-2.14-1.rhel8.noarch.rpm
rpm ivh python3-barman-2.14-1.rhel8.noarch.rpm

mkdir /var/log/pg_barman /home/postgres/barman
chown postgres:postgres /var/log/pg_barman /home/postgres/barman

cat <<EOF >> /etc/barman.conf -- Barman全体設定ファイル
barman_user = postgres
barman_home = /home/postgres/barman -- Barmanバックアップ格納ディレクトリ
log_file = /var/log/pg_barman/barman.log
EOF
```

Barman 環境設定

```
[root]#
cat <<EOF >> /etc/barman.d/pg.conf -- Barmanバックアップ取得対象DB設定ファイル
[pg]
description =  "Our main PostgreSQL server"
conninfo = host=localhost user=barman dbname=postgres
backup_method = postgres
streaming_conninfo = host=localhost user=streaming_barman dbname=postgres
streaming_archiver = on
slot_name = barman
create_slot = auto
EOF
```

Barman 環境設定、WALバックアップ

```
[postgres]#
createuser -s -P barman
createuser -P --replication streaming_barman

cat <<EOF >> $PGDATA/postgresql.conf
archive_mode = off -- Barmanがアーカイブするためoff
EOF

barman receive-wal pg &>/dev/null & -- BarmanがWALをストリーミング収集・アーカイブ
barman switch-wal --force --archive pg
barman check pg
```

- WALをストリーミング収集する機能が備わっている

Barman バックアップ・リストア

```
[postgres]#
barman backup pg       -- バックアップ
barman list-backup pg -- バックアップ一覧確認

pg_ctl stop -m immediate
barman recover pg 20211019T112204 /home/postgres/pg13/pgdata1 -- リストア
pg_ctl start
```

pgBackRest

pgBackRest インストール・環境設定

https://ftp.postgresql.org/pub/repos/yum/common/redhat/rhel-8-x86_64/からRPMを入手

```
[root]#
rpm ivh pgbackrest-2.35-1.rhel8.x86_64.rpm

mkdir /home/postgres/pgbackrest
chmod 750 /home/postgres/pgbackrest

cat <<EOF > /etc/pgbackrest.conf
[global]
repo1-path=/home/postgres/pgbackrest -- リポジトリディレクトリ

[main]
pg1-path=/home/postgres/pg13/pgdata1 -- バックアップ対象データベースクラスタ
EOF
```

pgBackRest 環境設定

```
[postgres]$
-- archive_command で pgBackRest を利用
cat <<EOF >> $PGDATA/postgresql.conf
archive_command = 'pgbackrest --stanza=main archive-push %p'
EOF
pg_ctl restart

-- バックアップカタログ作成
pgbackrest --stanza=main --log-level-console=info stanza-create

-- バックアップカタログ確認
pgbackrest --stanza=main --log-level-console=info check
```

- アーカイブ処理もpgBackRestのコマンドを利用する必要がある

pgBackRest バックアップ・リストア

```
[postgres]$
pgbackrest --stanza=main --log-level-console=info backup -- バックアップ

pg_ctl stop -m immediate

rm -f -r /home/postgres/pg13/pgdata1
rm -f -r /home/postgres/pg13/ts1

pgbackrest --stanza=main restore -- リストア
pg_ctl start
```

WAL-G

WAL-G インストール

```
[root]#
mkdir /root/.go
export GOPATH=$HOME/.go

go get github.com/wal-g/wal-g
cd $GOPATH/src/github.com/wal-g/wal-g
make install
make deps
make pg_build

cp -p /root/.go/src/github.com/wal-g/wal-g/main/pg/wal-g /usr/bin
```

- RPMパッケージがないため、ビルドする必要あり

WAL-G 事前設定

```
[postgres]$
mkdir -p /home/postgres/wal-g/backup

cat <<EOF >> ~/.bash_profile
export WALG_FILE_PREFIX=/home/postgres/wal-g/backup -- WAL-Gバックアップディレクトリ
EOF

cat <<EOF >> $PGDATA/postgresql.conf
archive_command = '/usr/bin/wal-g wal-push %p' -- archive_command で WAL-G を利用
EOF

pg_ctl restart
```

WAL-G バックアップ・リストア

```
[postgres]$
wal-g backup-push /home/postgres/pg13/pgdata1 -- バックアップ
wal-g backup-list -- バックアップ一覧確認

pg_ctl stop -m immediate
rm -f -r /home/postgres/pg13/pgdata1
rm -f -r /home/postgres/pg13/ts1

wal-g backup-fetch /home/postgres/pg13/pgdata1 LATEST -- リストア

cat <<EOF >> $PGDATA/postgresql.conf
restore_command = '/usr/bin/wal-g wal-fetch %f %p' -- restore_command で WAL-G を利用
EOF
touch $PGDATA/recovery.signal
pg_ctl start
```

まとめ

- 論理バックアップの対象は、
  データベースクラスタ共有の情報(ロール、テーブルスペース)および
  各データベースの情報

- 物理バックアップの対象は、
  データベースクラスタのファイルとアーカイブWAL

まとめ

- 論理バックアップからの復旧はシンプルだが、
  バックアップ時点にしか復旧できない
- 物理バックアップからの復旧は複雑、
  WALをどのように適用してリカバリするかによって手順も異なる
- 多くのデータベースクラスタのバックアップ管理が必要であったり、
  特定の機能が必要な場合は、バックアップ管理ツールの利用を検討