What’s Neon?
～クラウドネイティブを実現するOSS～

2023年11月24日

PostgreSQL Conference Japan 2023

はじめに

- 名前
  - ✓ 清野 裕貴
- 主な業務
  - ✓ PostgreSQL や周辺ツールの調査・検証・サポート
  - ✓ クラウドネイティブDBMS関連の調査・検証

※ 本資料の会社名、製品名は各社の商標または登録商標です

本講演について

- 講演資料に記載の情報は 2023.10.23 時点の情報です

- 本発表に掲載している検証結果は、以下の環境で取得したものです
  - Ubuntu 2204 (Windows11-WSL2)
  - docker 24.0.6 , containerd 1.6.24
  - docker-compose 1.29.2
  - Neon 3953 (PostgreSQL 16)

- 環境や条件により、結果が異なる場合があります

クラウドネイティブなDBMS

クラウドネイティブなDBMS、どのようなものをイメージしますか？

- PostgreSQL on K8s?
- NewSQL?
- マネージドサービス（マネージドなDBMS）？

例えば、代表的なメガクラウドのクラウドネイティブDBMS

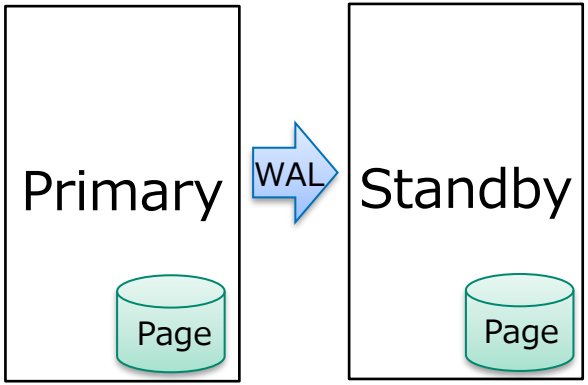
- Amazon Aurora
- AlloyDB for PostgreSQL (GCP)
- Azure SQL Database (SQL DB) - Hyperscale サービスレベル　※ SQL Server互換

アーキテクチャの共通点

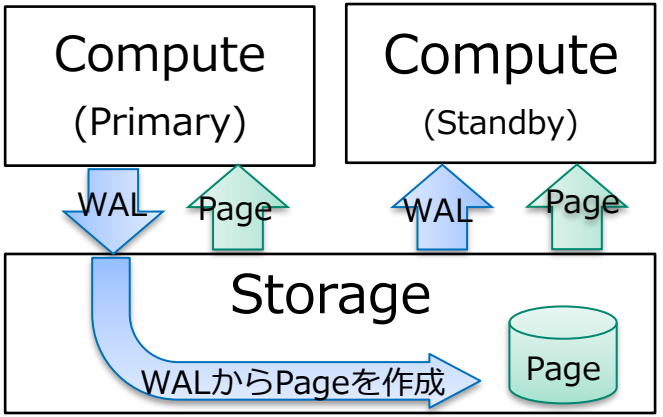
Compute/Storageレイヤを分離したアーキテクチャを採用している点

- 必要なリソース（Compute or Storage）だけをより細かい粒度で柔軟にスケールできる
- Computeの処理量（ex. Checkpoint）を減らして、より高スループットが実現できる　等のメリット

PostgreSQLなど既存RDBMS



クラウド向けアーキテクチャ



※ 細かい違いは無視してかなりざっくりと記載しています

（参考）Amazon Aurora

Amazon Web Services発表資料より抜粋（Amazon Aurora アーキテクチャ概要）

スケールアウト, 分散, マルチテナントデザイン

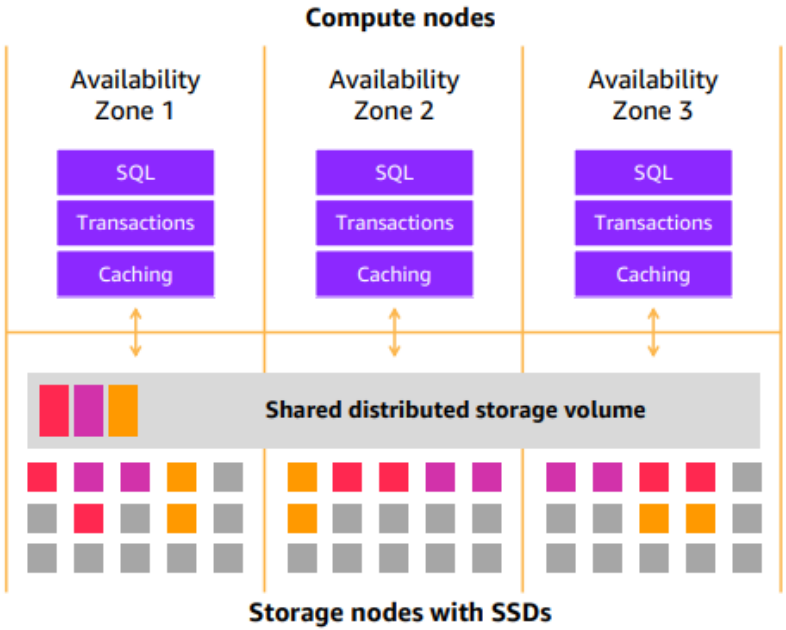
Aurora scale-out, distributed architecture

データベース用に設計された
専用の log-structured 分散ストレージシステム

3つの異なるアベイラビリティゾーンに分散された
数百のストレージノードにストライピングされた
ストレージボリューム

データは10GBのプロテクショングループの単位で
書き込まれる。自動的に最大64TBまで拡張

AZ+1 の障害から保護するためにデータを
各アベイラビリティーゾーンに2つのコピー、
リージョン内で計6つのコピー



アジェンダ

- What’s Neon ?
- Neon Architecture
- Install

What’s Neon?

Neon の概要

- **Neon**：AWS Aurora PostgreSQL（Serverless）などの代替を目指すOSS
  - ✓ 主な開発元はNeonという会社で、PostgreSQL Committerの Heikki さんが共同創業者

#	項目	詳細（2023/10/23時点）	備考
1	主な開発元	Neon社	Neonを用いた DaaS も提供し始めている
2	開発状況	活発。毎日Commitされている	-
3	PJ開始時期	2021年3月	-
4	リリースバージョン	Release 4053	定期的にDockerイメージが更新されており、ユーザも利用することが可能（すべてのリリースバージョンごとに更新はされていない）
5	開発言語	C言語/Rust	Compute周り: C言語 ⇒ PostgreSQLコードやExtensionなど Storage周り: Rust ⇒ マルチテナントを想定しており、プロセス毎に制御できるデータを限定
6	ライセンス	Apache License 2	PostgreSQL Licenseではない

(※) https://neon.tech/team/ (※) https://github.com/neondatabase/neon

PostgreSQL ベース

Compute部分は **PostgteSQLベース** で実装（Pageserverも一部PostgreSQLの処理が流用されている）

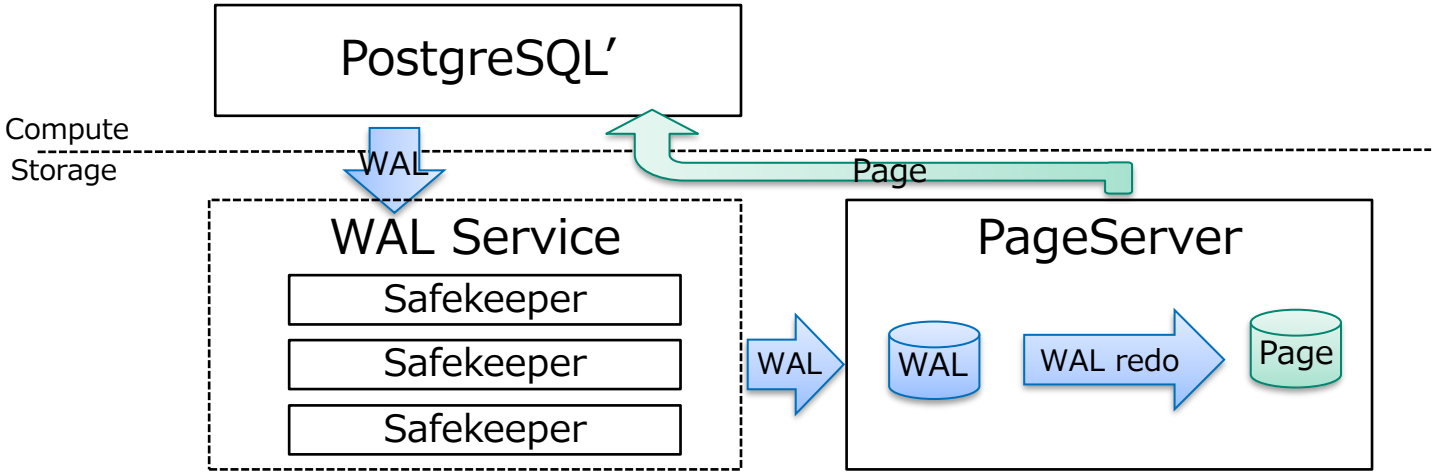
最終的には、PostgreSQL の拡張機能としての実装を目指している

- PostgreSQL からの変更は v16で3,000行程度
  - プランナ、エグズキュータの変更はなく、MVCCはPostgreSQLの実装が利用できる
- データの read/write 処理に変更を加えている
  - read/write のシステムコールの近くを hook している
- 現時点でも PostgreSQL との互換性は高い
  - 全てのインデックスタイプをサポート
  - PostgreSQL の拡張機能が使用できる（一部使用できない拡張機能もある *）

* uuid-ossp, sslinfo, pg_cron, file_fdw (https://neon.tech/docs/extensions/pg-extensions)
pgroonga (https://community.neon.tech/t/support-for-pgroonga-extension/1331)

基本的なアーキテクチャ

- Neonは、Auroraなどと同じく、**Compute/Storage レイヤを分離**したアーキテクチャを採用
  - ✓ Compute = PostgreSQL のNeonカスタム版
  - ✓ Storage = Neonストレージシステム (一部、PostgreSQLのコードが流用されています。)



※ 一部省略しています
- メタデータ管理のため、
Storage Brokerも存在する

Neonの特徴

Compute & Storge 分離

- Compute 側の処理削減 (write is a no-op)
- マルチテナンシー
- 安価なストレージ
- 粒度の高いスケール

サーバレス

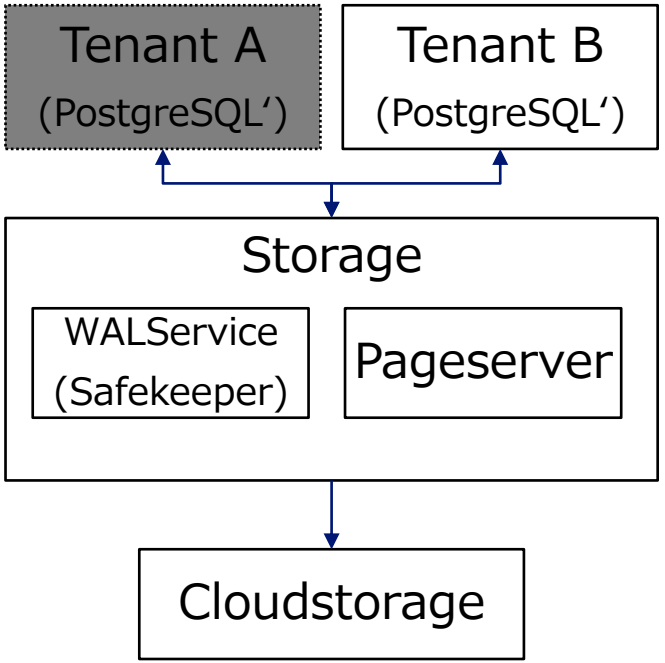
- 運用コスト
- 運用オペレーション

ブランチ

- 複数DB環境の管理

Compute & Storage分離

- Compute(PostgreSQL) が **write 関連の処理から解放**される
  - WALレコード を Safekeeper に転送するだけ
  - 共有メモリ内のページはディスクに書き込まず破棄される
- マルチテナンシー（ストレージレイヤの共有）
  - 複数テナントで同じ Storage を利用可能
  - レプリカも同じ Storage (のプライマリと同じデータ)を利用可能
- 粒度の高いスケール
  - コンポーネント単位でスケールが可能
    - Sagekeeper はスケールアウトも可能
- 安価なストレージ
  - 古いデータ（頻繁にアクセスされないデータ）は **Cloud Storage（S3などのクラウドストレージ） に塩漬け**できる
    - 高価なローカルSSDに置く必要がない

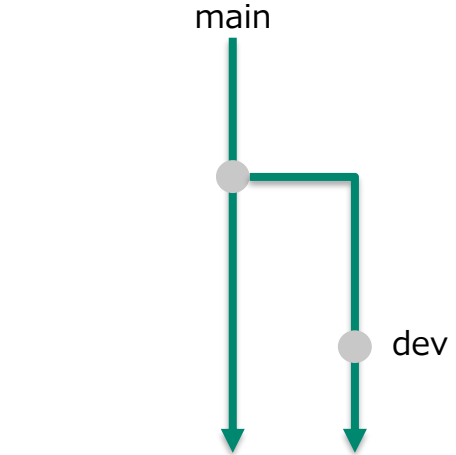


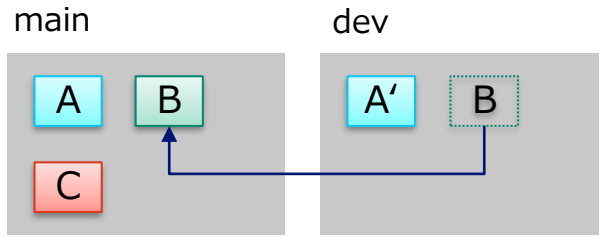
サーバレス

- 運用コスト
  - ワークロードに応じて**オンデマンドでComputeを起動**可能
- 運用オペレーション
  - コンポーネントダウン時の**リカバリOPは再起動のみ（いわゆるステートレス）**
  - 起動が速い
    - › クラッシュ時のリカバリにおいても、PostgreSQL のようなWALリプレイは発生せず、即時にクエリを受け入れできる
    - › ただし、初回データアクセスは Pageserver へのアクセスが生じる

ブランチ

- 複数DB環境の管理
  - 複数の環境ごとにブランチ管理（分岐・マージ）することが可能
    - 例１：開発用・テスト用・ステージング用にブランチ
    - 例２：開発者ごとにブランチ
  - ブランチは即座に切られる（**copy-on-write**）
  - ブランチ内で Page が見つからなければ、親のブランチの Page を検索しに行く仕組み

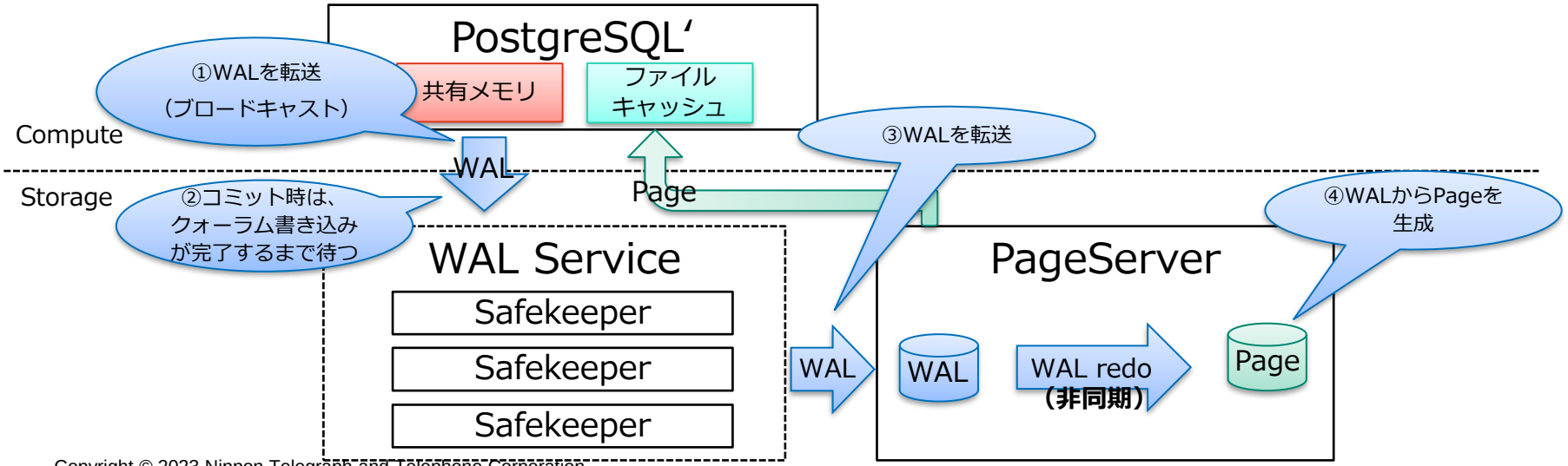




Neon Architecture

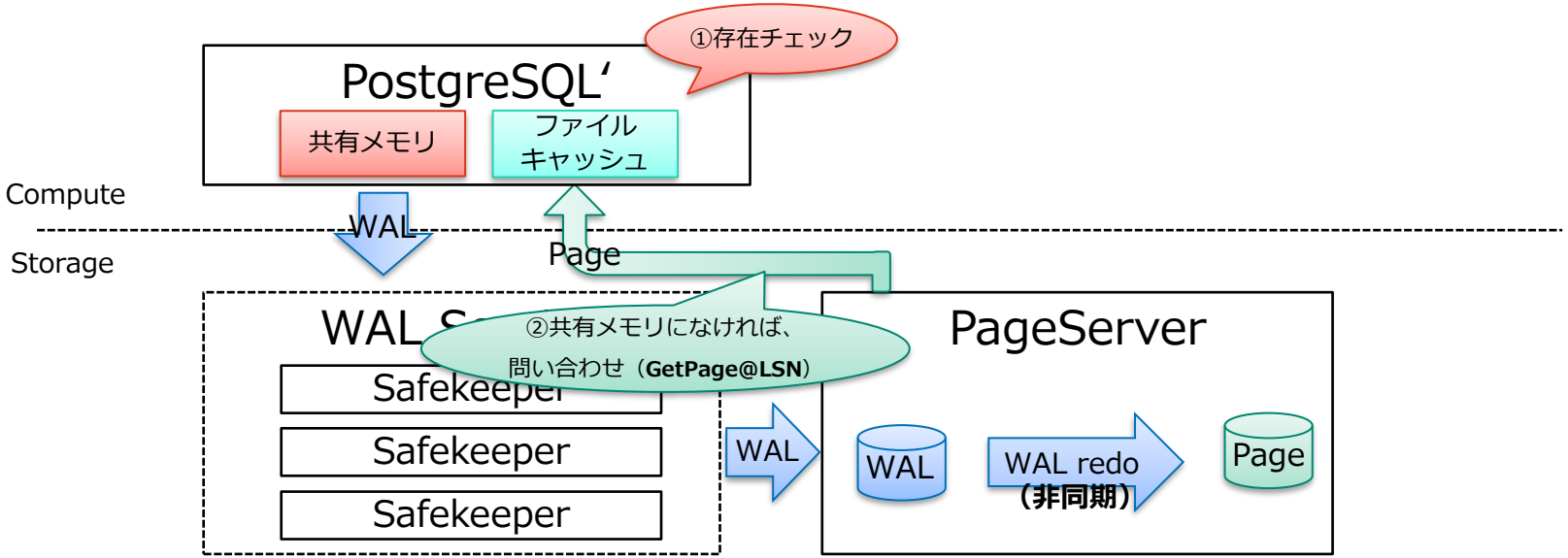
データの書き込みフロー

- 通常のPostgreSQLとの違いは、生成したWALを、常時WAL Serviceに転送している点
  - ✓ クエリ処理(パース、リライト、プラン生成)は同じ。書き込み時に共有メモリ or ファイルキャッシュにデータがなければ、PageServerに問い合わせ
  - ✓ 独自のWAL転送プロセス(wal proposer)によって、常時WALレコードを転送。**コミット時には、クォーラムが確保できるまで待つ**。（**コンセンサスアルゴリズム→同期ストリーミングレプリケーションのクォーラムコミットのイメージが近い**）



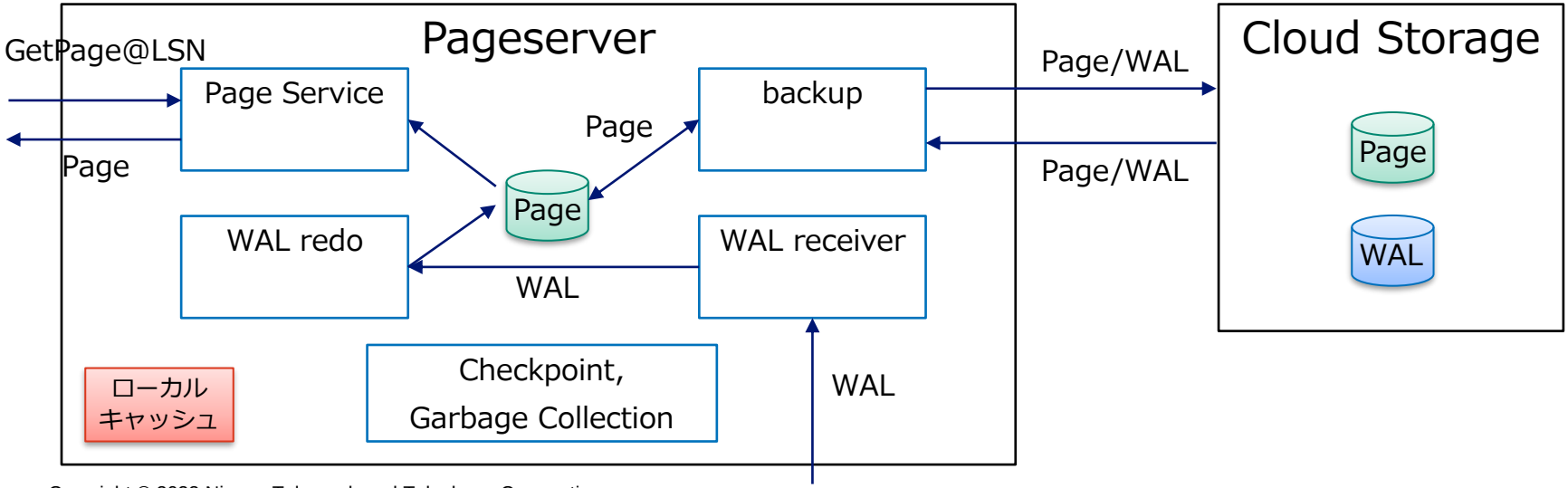
データの読み込みフロー

- 通常のPostgreSQLにおけるディスク読み込みが、PageServerへの問い合わせに変わる
  - ✓ 通常のPostgreSQLと同じで、共有メモリでも管理している。**共有メモリ or ファイルキャッシュに存在しなければ、PageServerに問い合わせる**
  - ✓ PageServerへの問い合わせには、**最新のLSN（LogSequenceNumber）も含める**のが特徴的。



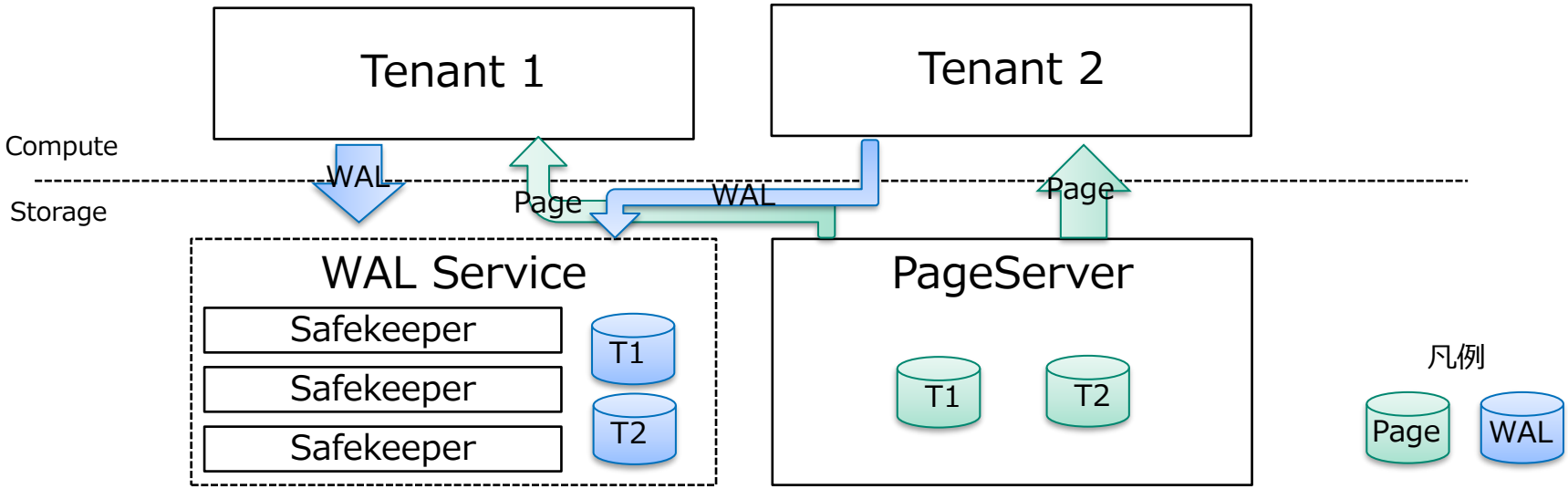
Pageserver の処理

- Pageserver は **既存の Page から WAL Redo により新しい Page を作成**する
- 必要なタイミング(Computeから要求されたタイミング)で Cloud Storage から**必要な Page/WAL をダウンロード**する
- Computeからの応答高速化のために**一定量の Page をローカルキャッシュにも格納**



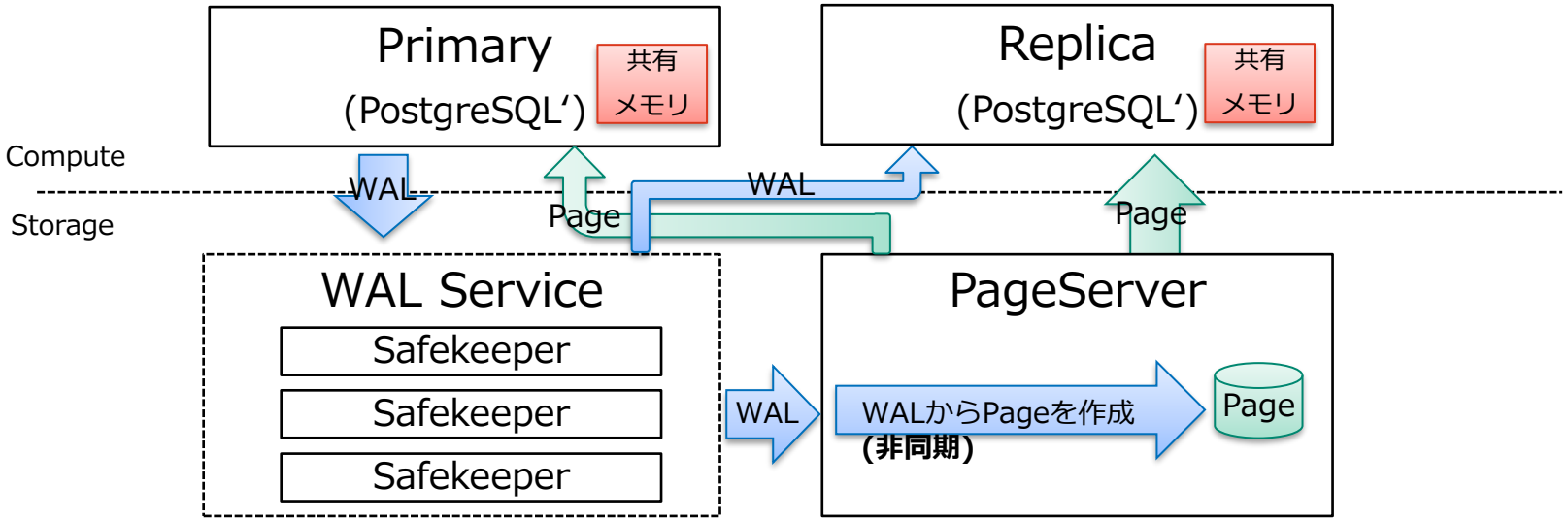
マルチテナンシー

- Neon の Storage レイヤは**マルチテナンシー**をサポート
- **TenantID (+TimelineID)** をキーとして、テナント毎のデータを分別管理



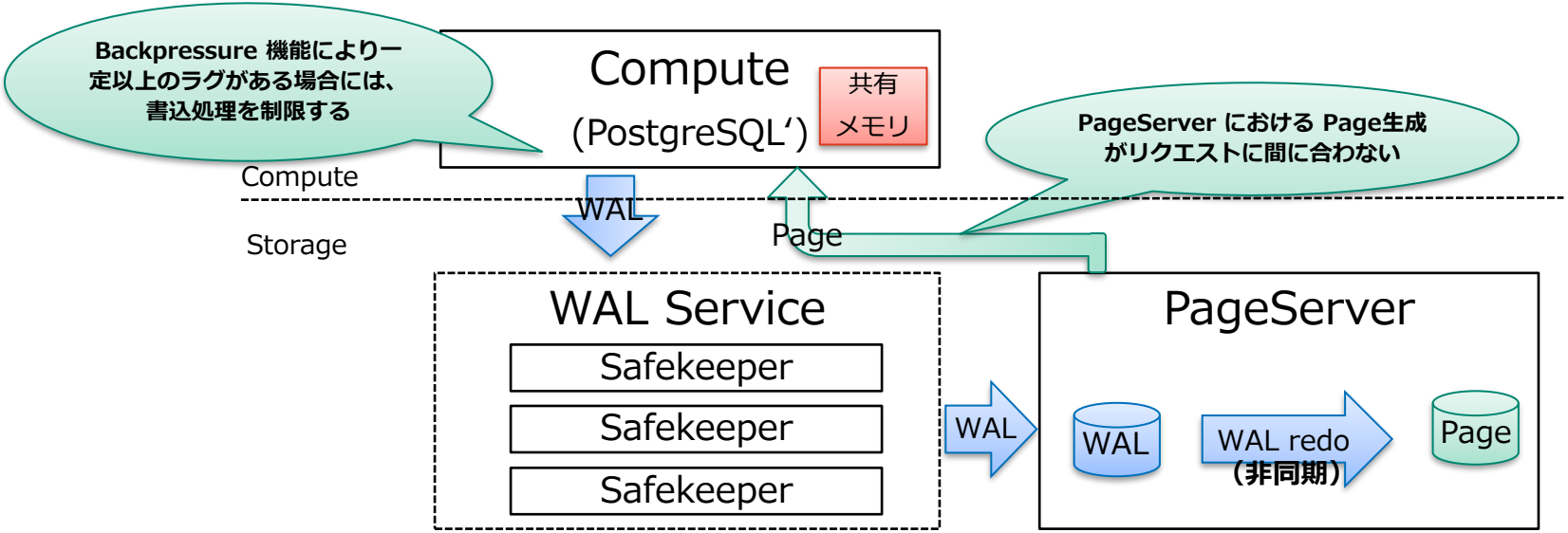
参照用Replica

- 参照用の **Replica もサポート**している
- 通常のPostgreSQLと異なり、**WAL Service から WALをストリーミング**する



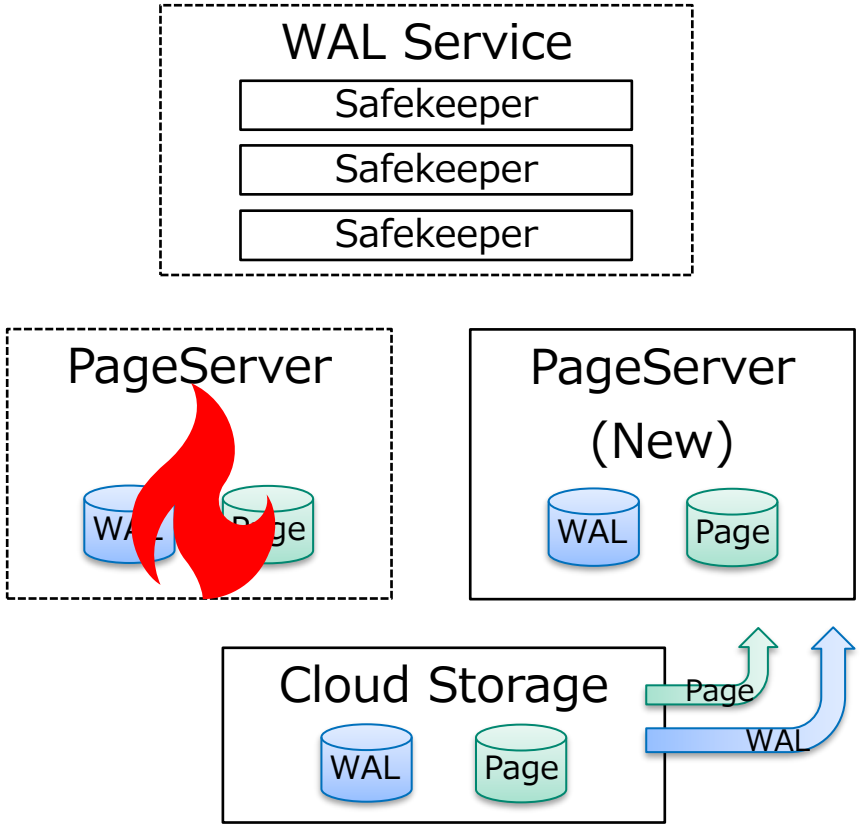
Page生成遅延への対応

- Compute が Page をリクエストした際に、まだ対応するページが生成されておらず、タイムアウトエラーとなる可能性がある。このラグを一定期間に抑えるための**書込みに制限をかける Backpressure 機能**が用意されている



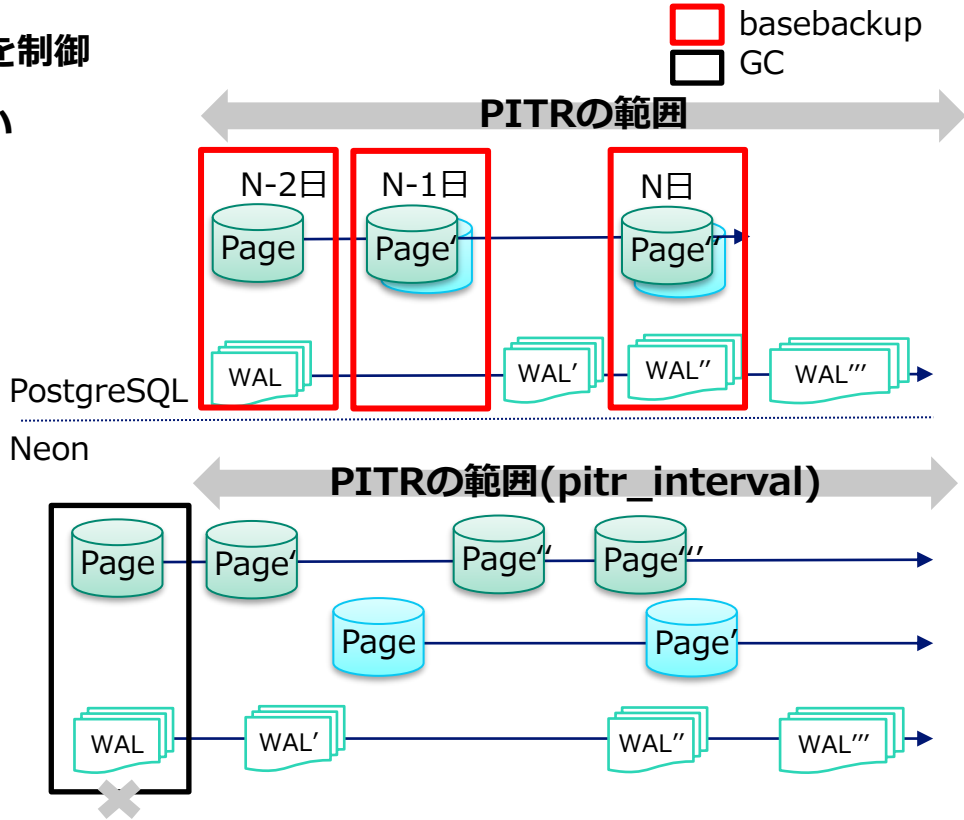
耐久性

- **耐久性は2つの要素で担保**
  - 直近のWALを補完する **Safekeeper**
  - Cloud Storage
- **Pageserver は失っても問題ない**
  - 新しい Pageserver を起動して、必要なものを **Cloud Storage からダウンロード**するだけ



ポイントインタイムリカバリ

- **パラメータ(pitr_interval)によりPITRの範囲を制御**
- **Neon にはベースバックアップという概念がない**
  1. リストアポイントの直前の Page を検索
  2. リストアポイントまでの WAL Redo を行う
- **バックアップ、リカバリの運用**
  - › PostgreSQL の一般的なバックアップ運用
    1. ベースバックアップ
    2. 定常的にWALをアーカイブ
  - › Neon のバックアップ運用
    1. 独自の形式で Page, WAL を**定常的**に CloudStorage に保存
    2. **pitr_interval** を超過すると **Gabage Collection**を行う。(古い Page, WAL の削除)



ストレージエンジン

Neon の ストレージエンジンは以下のような特徴を持つ。

- **KVS(Key-value store)**
  - Key：relationid + block number + LSN
  - Value：**8KBのPage** or **WALレコード**
- **LSM（Log-Structured Merge）ツリーを参考に実装されている**
- **イミュータブルなファイル**
  - 既存のファイルは決して修正されない
  - 古いファイルはマージと圧縮が行われ、新しいファイルを作成し、古いファイルを削除する
- **全てのファイルは Cloud Storage にアップロード**
  - データが存在しない場合は、オンデマンドでダウンロードしてデータを戻す。

ストレージフォーマット

- イミュータブルなファイルはレイヤーファイルと呼ばれ、2種類存在する

デルタレイヤーファイル

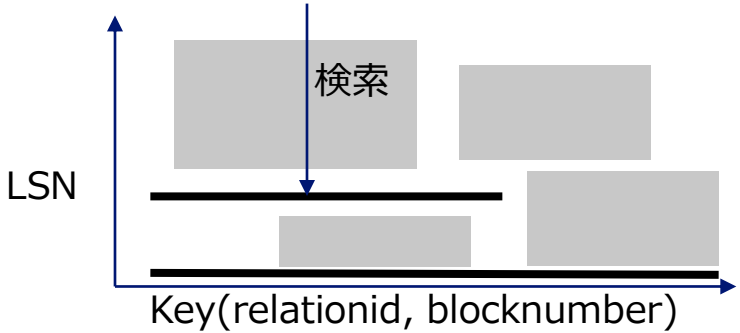
» 特定キー範囲（relationid, blocknumber）の**複数LSN範囲**の **WALレコード** を保持

イメージレイヤーファイル

» 特定キー範囲（relationid, blocknumber）の**単一LSN** の **Page** を保持

» 蓄積された WALレコード 量の一定の閾値に基づき作成される

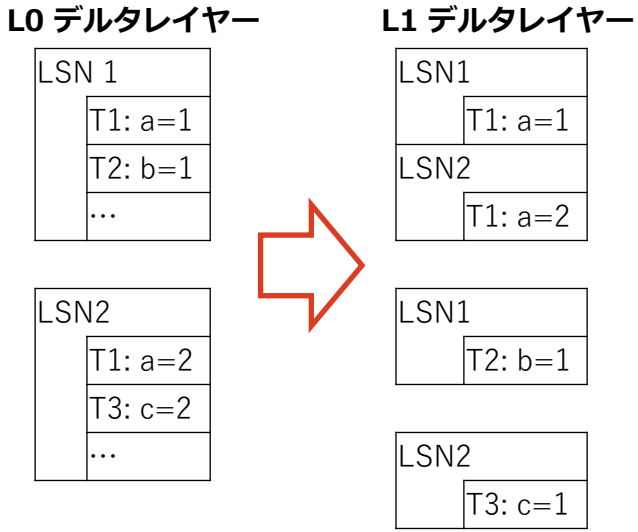
» Page の検索効率に寄与する

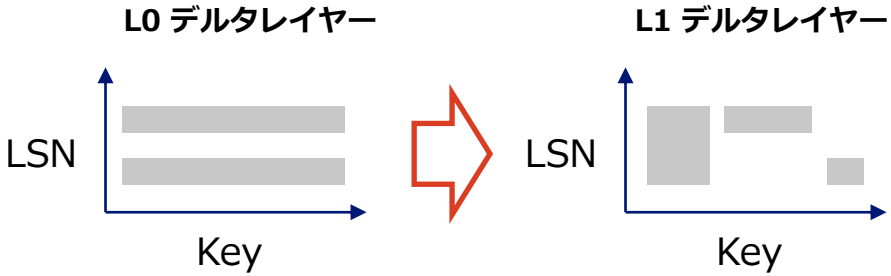


※ シンプルに抽象化しており
実際の構造とは異なります。

メンテナンス

- Compaction
  - Pageserver のファイルをマージ・圧縮する処理。ファイル内のWALレコードの対象のキーを狭く、LSNの範囲を広く取るよう変換する。

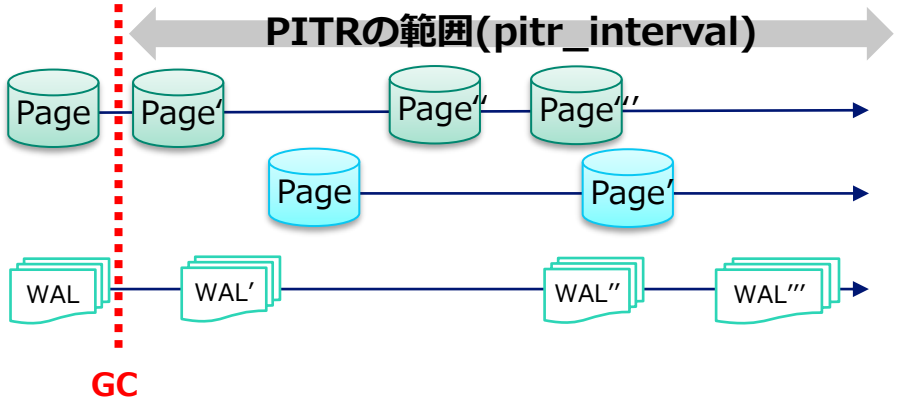


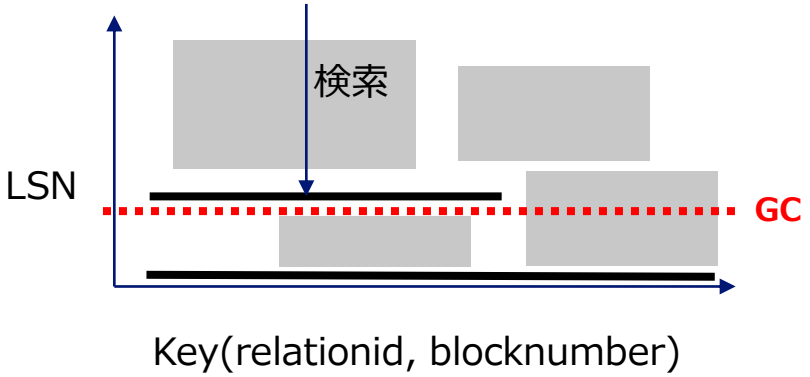


※ シンプルに抽象化しており
実際の構造とは異なります。

メンテナンス

- Garbage Collection
  - pitr_interval に基づき、どのブランチからも参照されないディスク上のデータ（デルタレイヤー、イメージレイヤーファイル）を削除する





Install

インストール by docker-compose

- Neonコミュニティでは、docker-composeのyamlファイルが提供されており、気軽に試せます

```
# リポジトリを入手
$ git clone https://github.com/neondatabase/neon.git

# docker-composeでNeonを起動
$ cd neon/docker-compose

# PG_VERSION: ComputeのPostgreSQLバージョン。v14～16 をサポート
# TAG: Neonのdocker imageのtagバージョン(*)
$ PG_VERSION=16 TAG=3953 docker-compose up --build -d
(省略)
creating docker-compose_compute_is_ready_1   ... done
```

(*) Dockerhubで提供されているneonのtagバージョンを指定します。
　4桁の数字は、何コミット目のソースコードでビルドされたイメージかを示しています。
https://registry.hub.docker.com/r/neondatabase/neon/tags

インストール by docker-compose

```
# 起動したコンテナを確認
$ docker ps --format '{{.Names}}'  | sort
docker-compose-compute_1                # PostgreSQL起動しており、クエリを受けつけるコンテナ
docker-compose_compute_is_ready_1       # PostgreSQL起動状況を確認するコンテナ
docker-compose-minio_1                  # WAL/Pageデータのバックアップ先のコンテナ
docker-compose-pageserver_1             # ページデータを管理するコンテナ
docker-compose-safekeeper1_1            # WALデータを管理するコンテナ(冗長化されている)
docker-compose-safekeeper2_1
docker-compose-safekeeper3_1
docker-compose-storage_broker_1         # メタデータを管理するコンテナ
```

インストール by docker-compose

```
# 起動したComputeにpsqlで接続
$ echo "localhost:55433:postgres:cloud_admin:cloud_admin" >> ~/.pgpass
$ chmod 600 ~/.pgpass
$ psql -h localhost -p 55433 -U cloud_admin
psql=#

# 後は普通のPostgreSQLと同じように利用可能
psql=# CREATE TABLE t (key int primary key, value text);
CREATE TABLE
psql =# insert into t values(1,1);
INSERT 0 1
psql =# select * from t;
 key | value
-----+-------
   1 | 1
(1 row)
```

Cloud Storage に格納されるデータ

- Cloud Storage には、以下のようなデータが格納される

TenantID timelineID

```
neon /
   ├ pageserver / tenant / 57b57732e5adb041c182e62c5e4bf943 /
   |                          └ timeline / 0e802eac0b3da4b3c8400dd50c1ba587 /
   |                                                ├ 00000-FFFFF__00000000014EEA00-00000000014EEA79
   |                                                ├ 00000-FFFFF__00000000014EEA79-0000000002F18699
   |                                                └ index_part.json
   |
   └ safekeeper / 57b57732e5adb041c182e62c5e4bf943
                                  └ 0e802eac0b3da4b3c8400dd50c1ba587
                                                ├ 000000010000000000000001
                                                └ 000000010000000000000002
```

＊ 一部ファイル名を省略しています。

Cloud Storage に格納されるデータ

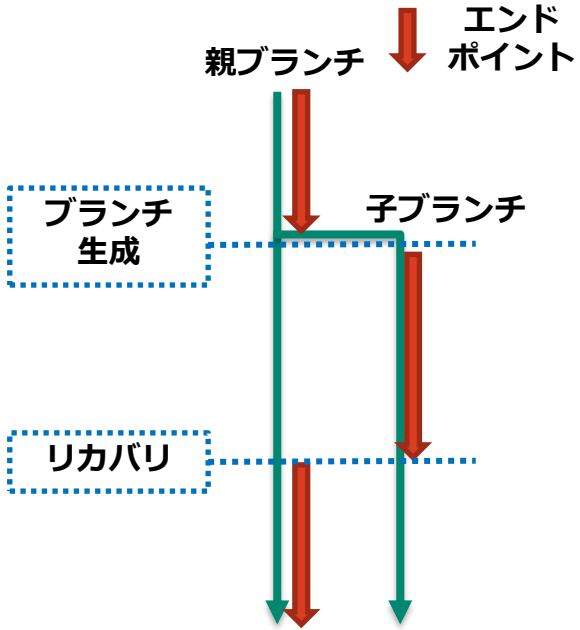
- index_part.json


```
{
    "version": 4,
    "layer_metadata": {
        "000000000000000000000000000000000000-FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF__00000000014EEA79-0000000002F18699": {
            "file_size": 28024832
        },
        "000000000000000000000000000000000000-FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF__00000000014EEA00-00000000014EEA79": {
            "file_size": 23289856
        },
        "000000000000000000000000000000000000-FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF__0000000022D9FAC9-0000000032468809": {
            "file_size": 267247616
},
    "disk_consistent_lsn": "0/32468808",
    "metadata_bytes": [
        149,
        123,
        147,
（以下省略）
```

Tips

ポイントインタイムリカバリ（ブランチ機能を使用）

- **ブランチ機能を利用したポイントインタイムリカバリ**の方法もある
- ブランチの切り替えにより実現するため、高速なリカバリが可能
  - https://neon.tech/blog/point-in-time-recovery
- **リカバリ基準地点の作成**
  1. 新しいブランチを作成する
     - » これがリカバリの基準地点
  2. エンドポイントを新しいブランチに切り替える
     - » この時点で新しいブランチに移行、この状態で運用される
- **リカバリ実行**
  1. エンドポイントを親ブランチに切り替える
  2. 使わなくなったブランチを削除する



レイヤーファイルの作成

- デルタレイヤー、イメージレイヤーファイルの生成シミュレーション
  - https://pgbench.vercel.app/

最近の変更

- PostgreSQL 16に対応
- Azure BLOB ストレージをサポート（#5546）
- ページサーバ間のテナント移行に対応（#5029）
  - ページサーバ及びS3からのテナントのデータ削除に対応（#4855）

マネージドサービス向けの機能

- ブランチ機能を利用したポイントタイムインリカバリ
  - https://github.com/neondatabase/restore-neon-branch

その他

- 公式リポジトリではないが、K8sへのデプロイのための Helm Chart も存在する
  - https://github.com/itsbalamurali/neon-helm

おわりに

- サーバレスなPostgreSQL
  - ✓ クラウドメガベンダとも共通して、**Compute/Storageを分離した**アーキテクチャを採用
    - ✓ Neonは、それらを参考に、PostgreSQLベースで同じアーキテクチャを実現しようとしている
  - ✓ PostgreSQL ベースで将来的には**拡張機能（Extension）での実現**を目指している
- 開発も活発で毎日数件の変更がコミットされている
  - ✓ OSSとして公開されているので、柔軟に利用できる
    - ✓ 機能追加などの開発貢献・コントリビュート
    - ✓ フォークによる独自開発
- Neonが提供するマネージドサービスを試用できる
  - ✓ ユーザ目線で見ると通常のPostgreSQLと変わらないが、Neon特有の機能（ブランチ機能等）が試せる

参考文献

- Neon 公式サイト
  - https://neon.tech/
- Neon GitHub
  - https://github.com/neondatabase/neon
- Neon: Serverless PostgreSQL! (Heikki Linnakangas)
  - https://www.youtube.com/watch?v=rES0yzeERns
- Amazon Aurora アーキテクチャ概要
  - https://pages.awscloud.com/rs/112-TZM-766/images/01_Amazon%20Aurora%20%E3%82%A2%E3%83%BC%E3%82%AD%E3%83%86%E3%82%AF%E3%83%81%E3%83%A3%E6%A6%82%E8%A6%81.pdf