Crossing makes the Future.

まじわる力でみらいを創る

フォトクリエイト社様における PostgreSQL 8.3 から Amazon Aurora への移行事例

NTTテクノクロス株式会社

上原　一樹

NTTテクノクロス

目次

はじめに

自己紹介、弊社業務紹介

自己紹介

上原 一樹

PostgreSQL

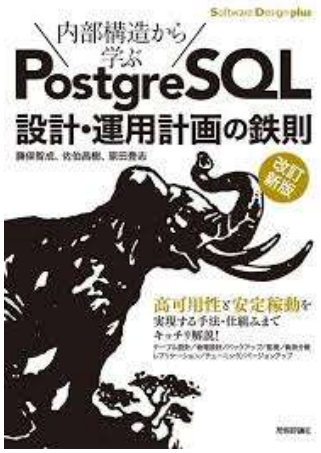
コンサルティング, 技術支援, トラブルシュート

NTTテクノクロス　業務概要

- OSSクラウド基盤トータルサービス

https://www.ntt-tx.co.jp/products/osscloud/

- PostgreSQL本の執筆メンバ在籍
  - 鉄則本、大好評発売中！



本講演について

- 2019年8月にフォトクリエイト社様のシステムで行われたEC2上のPostgreSQLからAmazon Auroraへの移行事例と案件を通して得られた知見についてご紹介させていただきます。

フォトクリエイト社様における
DB移行の背景と結果

フォトクリエイト社様のご紹介

- 2002年1月 創業
- 2019年4月 キタムラ・ホールディングス

プロカメラマンが撮影した
クオリティの高い写真をお届け

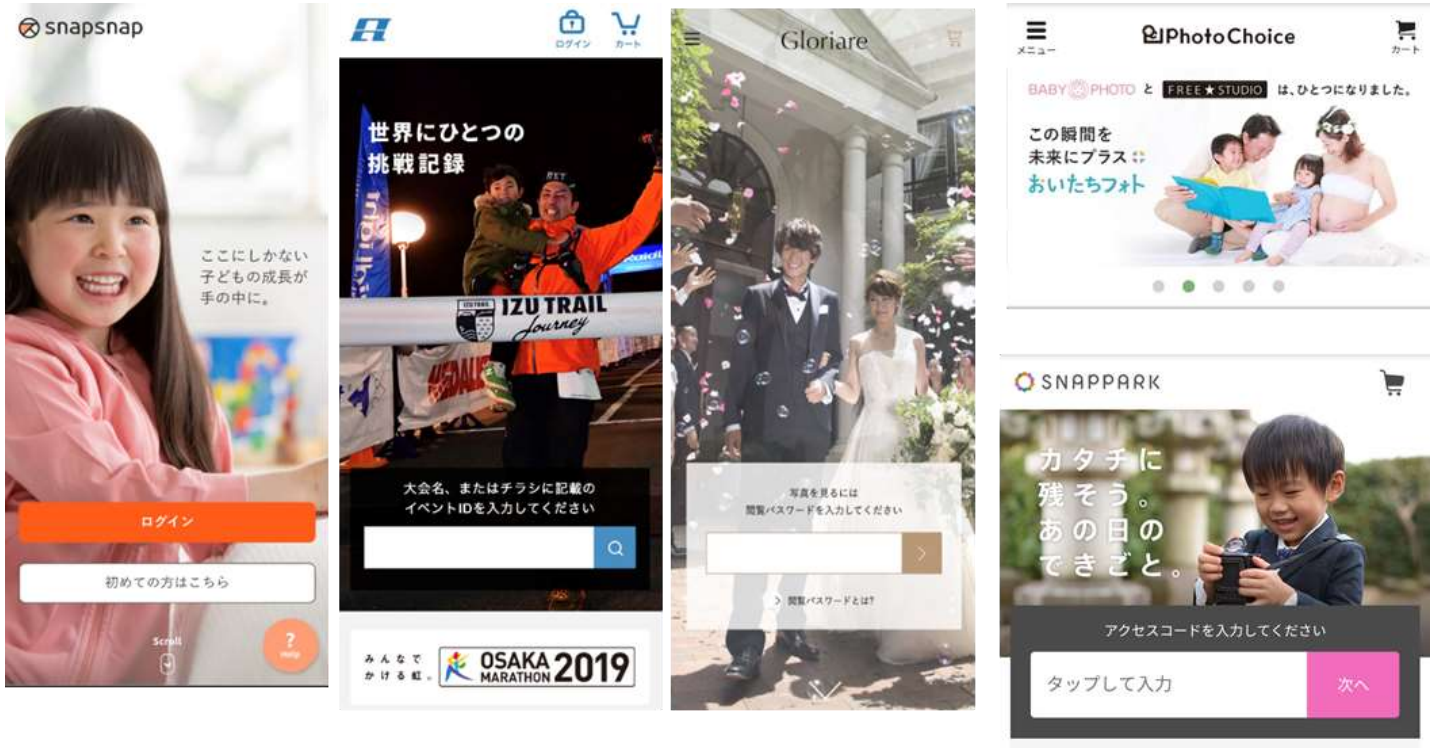
出生から七五三、入学式、成人式、
結婚式などの人生のステージ、
スポーツや音楽などの輝く瞬間を
1,700人を超える提携プロカメラマンが
撮影した写真を販売するインターネット
写真販売サービス。

プロダクト概要

インターネット写真販売事業として下記のサービスを展開

snapsnap　ALL SPORTS community　Gloriare

PhotoChoice　SNAPPARK



プロダクト概要

インターネット写真販売事業として下記のサービスを展開

snapsnap　ALL SPORTS community　Gloriare

PhotoChoice　SNAPPARK

月間アクセスユーザ
60万人

全国 **9000**校
幼保学校で導入

マラソン分野
全国 **90**%の大会
網羅

サービス概要

1. イベントにアサインされたカメラマンによる撮影

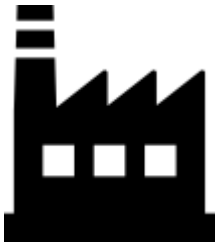
2. ゼッケン番号等で検索し、写真を注文

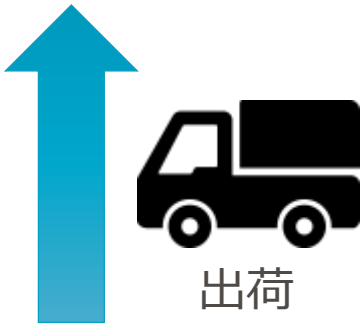
3. 手元にプロが撮影した写真が届く

写真データはゼッケン番号や顔の認識処理を行いDBに登録

コンテンツ検査

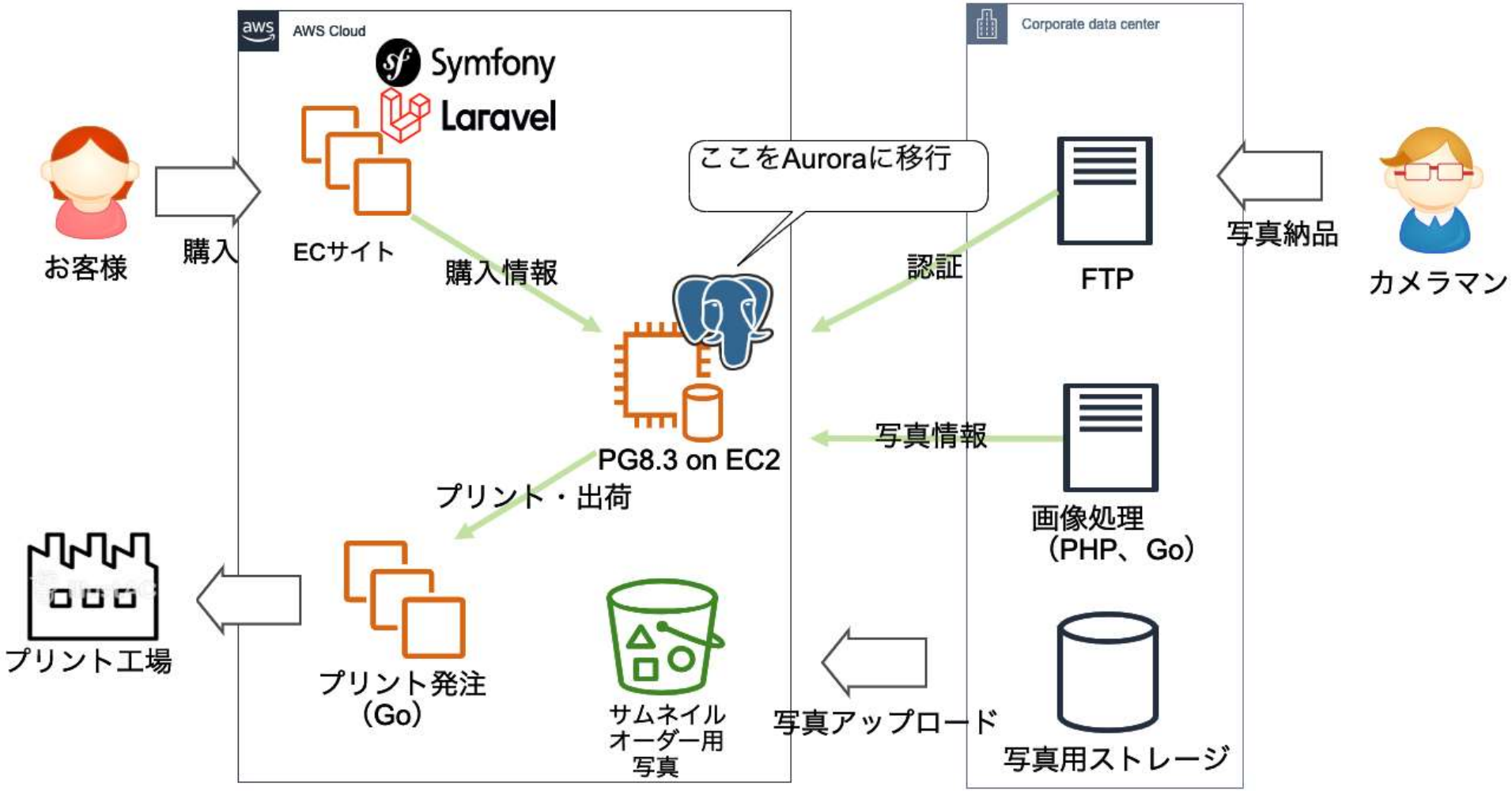
写真データ、注文データをプリント工場へ送信





出荷

移行対象となったシステムの構成（概略図）



移行前のシステムにおける課題

課題１：システム管理の専任の担当者がおらず、創業当時の手順が踏襲され、運用面の見直しが行われていない状況だった。

Slonyの扱いが難しい
- 負荷によってはレプリケーション遅延が起きることがある
- DDLは伝播されないのでプライマリとセカンダリ、個別実行が必要

バックアップ・リストア
- 日次フルバックアップ（論理）のみの運用
- データロストを伴う障害時のリストア・リカバリ運用に不安

課題２：期待するパフォーマンスが出ていなかった。

PostgreSQLのバージョン
- 旧バージョン利用のため、性能改善や新機能の恩恵を受けられない

移行後の成果

課題１：システム管理の専任の担当者がおらず、創業当時の手順が踏襲され、運用面の見直しが行われていない状況だった。

Slonyの扱いが難しい
- 負荷によってはレプリケーション遅延が起きることがある
- DDLは伝播されないのでプライマリとセカンダリ、個別実行が必要

バックアップ・リストア
- 日次フルバックアップ（論理）のみの運用
- データロストを伴う障害時のリストア・リカバリ運用に不安

- Amazon Auroraの高可用性機能によるSlonyの撤廃
- スナップショット機能によるRPOの改善
  (論理バックアップ取得時→直近5分前)

移行後の成果

課題２：期待するパフォーマンスが出ていなかった。

PostgreSQLのバージョン
- 旧バージョン利用のため、性能改善や新機能の恩恵を受けられない。

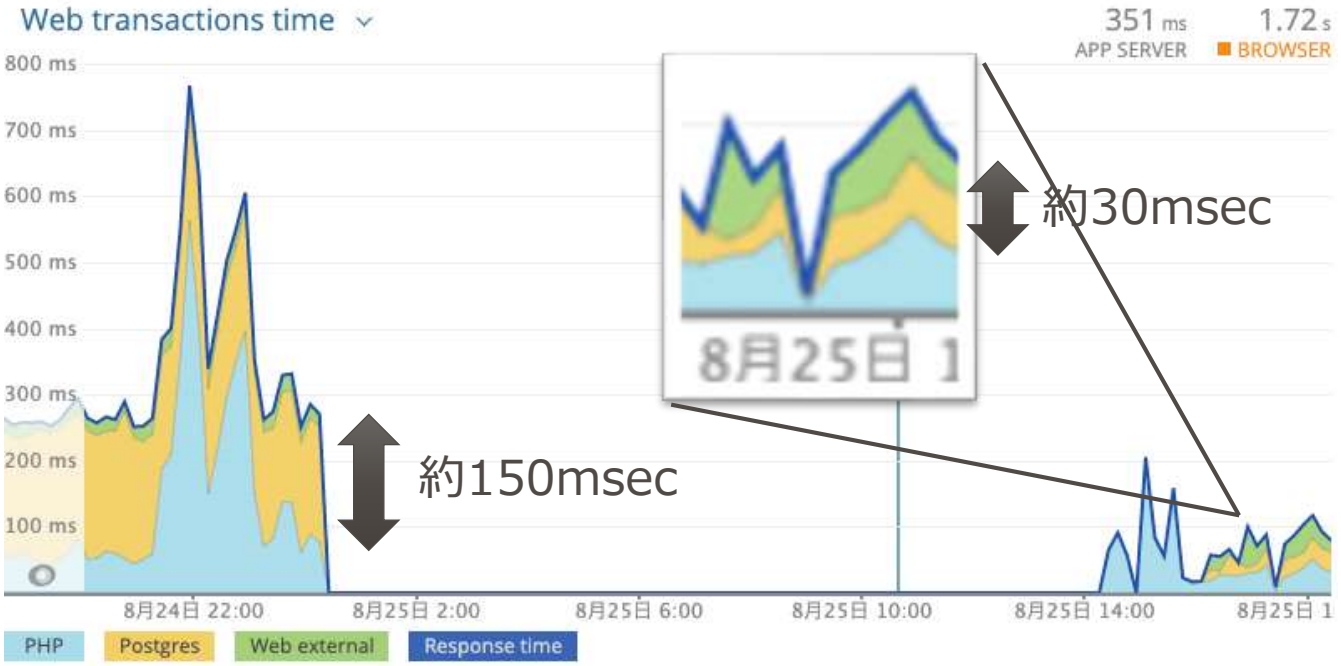
- DBの応答性能が向上した（詳細は次ページを参考）

その他課題：開発効率の改善

開発環境向けDB作成に2日以上掛かっていた。

- 開発環境向けDB作成が3時間で完了

参考：移行による応答性能の改善



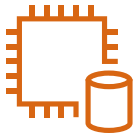
- DB移行後、サービス応答性能がおよそ5倍に改善
- 体感でも早くなったという反響をいただいている
  - 社内メンバだけなく、ユーザや取引先からも反響あり

移行実施における課題と対策

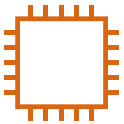
移行作業に立ちはだかる課題とその対策について

移行概要

- 対象プロダクト
  - PostgreSQL8.3　→　Amazon Aurora（PostgreSQL 10.7）
- データベースサイズ：約2TB
- テーブル数：大小合わせて400程度
- 移行の実施要件
  - サービス完全停止時間内にAPの最終確認まで終えられること
  - DBのリファクタリングを実施すること
    - 権限分掌、データ型の変更、インデックス見直し　等
- 移行用ステージング環境



DB instance
検証環境用に本番環境
データをマスクしたもの



EC2 Instance
検証作業インスタンス

Amazon Aurora
検証環境用に作成されたもの

移行実施における課題

移行計画の検討で直面した**2つの課題**

移行方式の制限

移行時間の圧縮

移行実施における課題

移行計画の検討で直面した**2つの課題**

- 移行方式の制限
- 移行時間の圧縮

最適な移行方式の選定

- 移行サービスを利用する（AWS Database Migration Service(DMS)等）

メリット	デメリット（移行に利用する場合の懸念点）
• サービスを継続した状態で移行可能。 • メジャーバージョンが異なる場合や異種DBにも対応可。	• 古いバージョンのPostgreSQLからの移行には使えない。 （DMSはPostgreSQL9.4以降がサポート対象）

- 物理コピー（cpコマンド等）

メリット	デメリット（移行に利用する場合の懸念点）
• 手順が単純。 • データ転送だけで移行が完了するため、比較的短時間で済む。	• サービス停止が必要。 • メジャーバージョン違いや異種DB等、互換性の無い環境では対応不可。

- 論理コピー（pg_dump、COPY等）

メリット	デメリット（移行に利用する場合の懸念点）
• メジャーバージョンが異なる場合や異種DBにも対応可能。 • 最も柔軟性が高く、DBやテーブル単位等、行を選択した移行が可能。	• 書き込みの制限等、部分的なサービス停止が必要。 • リストアに時間がかかる。

最適な移行方式の選定

今回のケースでは・・・

- 移行元がPostgreSQL8.3

  → 移行サービスはサポート対象外

- 移行先がAmazon Aurora

  → 物理コピーしたファイルを配置不可

- 移行時にDBのリファクタリングを実施

  → 物理コピーでは対応できない

最適な移行方式の選定

- 移行サービスを利用する（AWS Database Migration Service(DMS)等）

メリット	デメリット（移行に利用する場合の懸念点）
• サービスを継続した状態で移 • メジャーバージョンが異なる 異種DBにも対応可。	バージョンのPostgreSQLから には使えない。 はPostgreSQL9.4以降がサポート対象）

不採用

- 物理コピー（cpコマンド等）

メリット	デメリット（移行に利用する場合の懸念点）
• 手順が単純。 • データ転送だけで移行が完 め、比較的短時間で済む。	ス停止が必要。 ャーバージョン違いや異種DB 互換性の無い環境では対応不可。

不採用

- 論理コピー（pg_dump、COPY等）

メリット	デメリット（移行に利用する場合の懸念点）
• メジャーバージョンが異なる場合や異種DBにも対応可能。 • 最も柔軟性が高く、DBやテーブル単位等、行を選択した移行が可能。	• 書き込みの制限等、部分的なサービス停止が必要。 • リストアに時間がかかる。

参考：移行方式とは直接関係はありませんが・・・

- メジャーバージョンが異なる場合は以下の点についても検討が必要になります。
  - 設定ファイルのパラメータ項目の変化/デフォルト値の変化
  - 新機能の扱い（有効/無効の判断）
  - 廃止された機能の使っている場合の対処（代替機能要否）

フォトクリエイト社様のシステムでは、以下の機能を採用。
- ホットスタンバイ（参照クエリの分散）
- 宣言的パーティショニング（パージ処理の簡易化）
- BRINインデックス（インデックスサイズ圧縮）
- パラレルクエリ（参照性能の向上）

移行実施における課題

移行計画の検討で直面した**2つの課題**

移行方式の制限

移行時間の圧縮

移行時間の圧縮

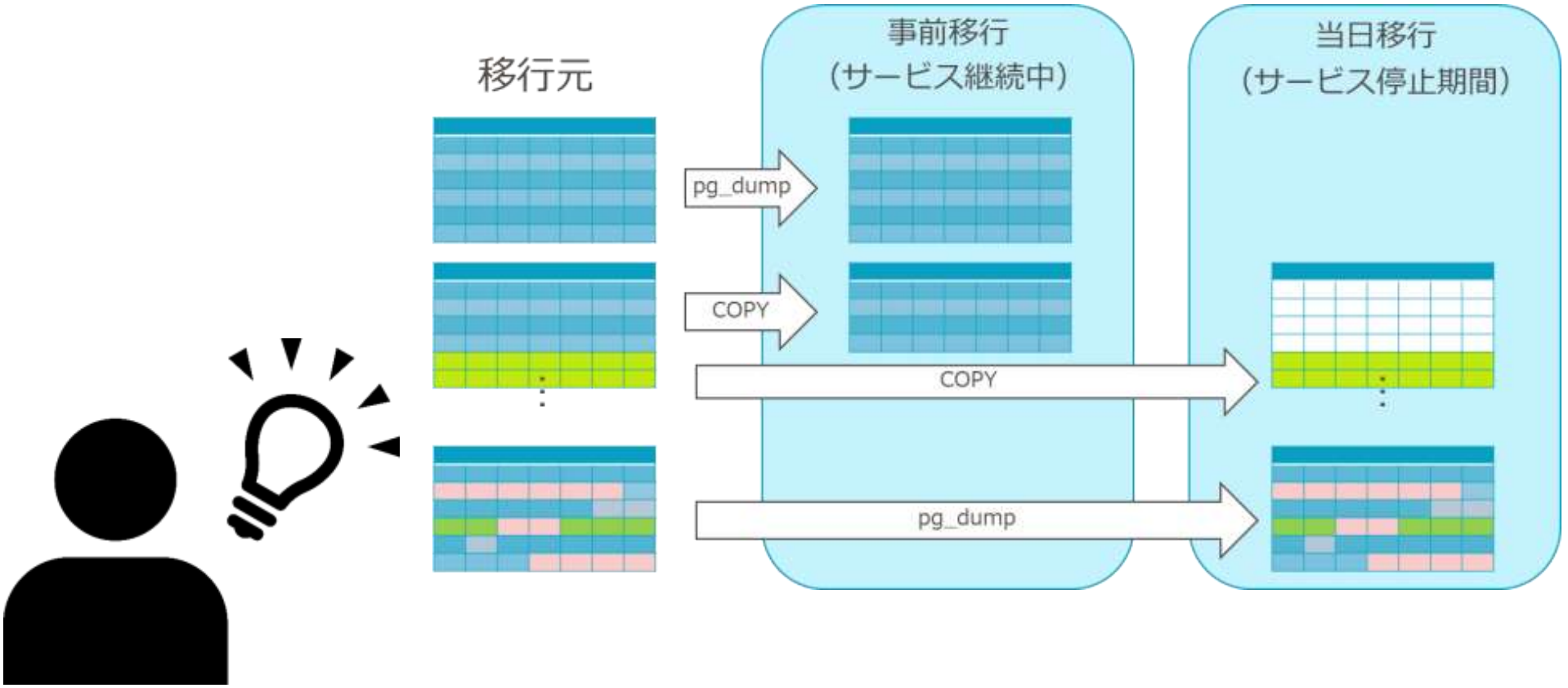
移行対象DB(2TB)を
論理コピーから
リストアする時間は...

サービス停止時間
（22時間）の半分はAPの
検証時間に残す必要が...

移行失敗時の
再走行時間を
確保しないと...

移行対象テーブルの分類

- サービス停止前日と当日で段階的な移行を実施することで、当日の移行時間を最小限に抑える案を検討

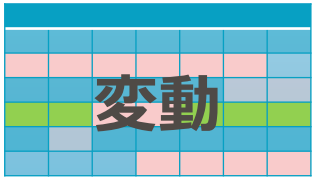


移行対象テーブルの分類

- フォトクリエイト社様協力の元、移行対象テーブルを以下に分類しました。
  1. 移行実施前後の数日間において、差分が発生しない

  2. 古いデータへ更新や削除はなく、差分を取得可能

  3. テーブル全体で更新や削除があり、差分を取得不可

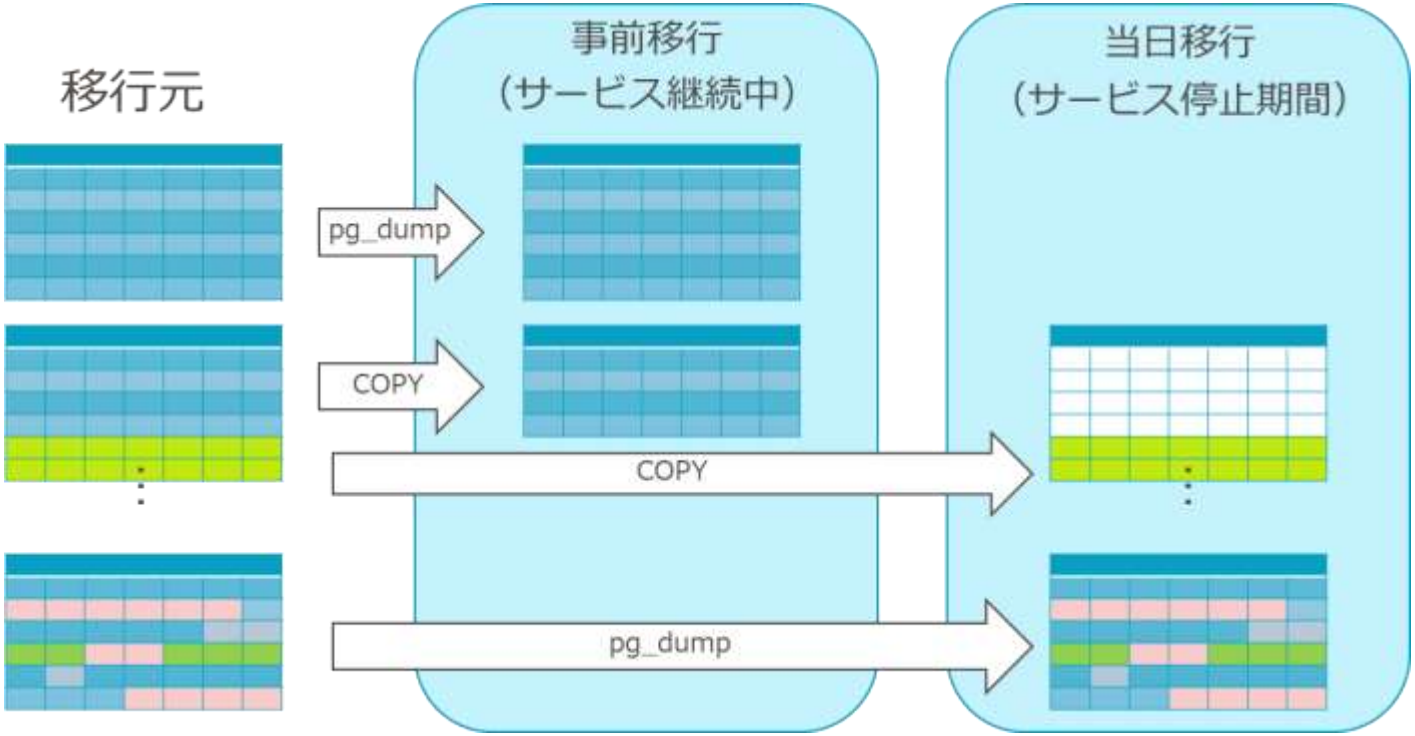


移行実施のイメージは以下のとおり

1. 移行実施前後の数日間において、差分が発生しない
2. 古いデータへ更新や削除はなく、差分を取得可能
3. テーブル全体で更新や削除があり、差分を取得不可

pg_dumpでテーブル全体を移行

COPY文による部分的な移行



策定した移行手順

- 段階的な移行として、以下の手順を策定しました。

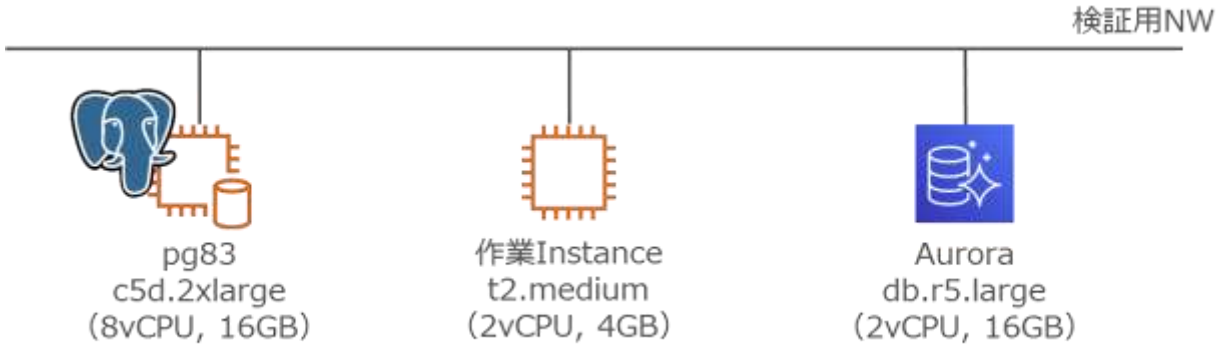
・事前移行データのEXPORT
1. pg_dump
2. 事前移行対象データのEXPORT（¥copy）

・事前データのIMPORT
3. 手順1のデータのIMPORT（psql -f）
4. 手順2のデータのIMPORT（¥copy）
5. インデックス作成

・当日データのEXPORT
6. pg_dump
7. 手順2の残りのデータをEXPORT（¥copy）

・当日データのIMPORT
8. 手順6のデータのIMPORT（psql -f）
9. 手順7のデータのIMPORT（¥copy）

- 移行用ステージング環境の構成（再掲）

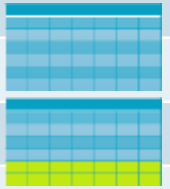


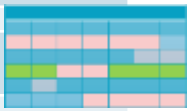
移行検証

作業Instance：t2.medium（2vCPU, 4GB）
pg83：c5d.2xlarge（8vCPU, 16GB）
Aurora：db.r5.large（2vCPU, 16GB）

- チューニングのためのベースラインの取得を実施。

実施内容	時間	対象テーブル
事前移行データのEXPORT (作業Instance <- pg83)		
1. pg_dump		差分が発生しないテーブル
2. 事前移行対象データのEXPORT（¥copy）		差分を取得可能なテーブル
事前データのIMPORT (作業Instance -> Aurora)		
3. 手順1のデータのIMPORT（psql -f）		
4. 手順2のデータのIMPORT（¥copy）		
5. インデックス作成		
当日データのEXPORT（作業Instance <- pg83)		
6. pg_dump		差分を取得できないテーブル
7. 手順2の残りのデータをEXPORT（¥copy）		差分を取得可能なテーブル
当日データのIMPORT（作業Instance -> Aurora)		
8. 手順6のデータのIMPORT（psql -f）		
9. 手順7のデータのIMPORT（¥copy）		



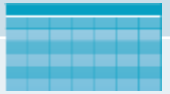
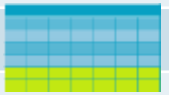
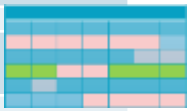


移行検証

作業Instance：t2.medium（2vCPU, 4GB）
pg83：c5d.2xlarge（8vCPU, 16GB）
Aurora：db.r5.large（2vCPU, 16GB）

- ベースラインの取得結果は以下のとおり。
  - 事前移行：34h　当日移行：9h　（※整合性チェック時間を含まない）

実施内容	時間	対象テーブル
事前移行データのEXPORT (作業Instance <- pg83)		
1. pg_dump	00h03m	差分が発生しないテーブル
2. 事前移行対象データのEXPORT（¥copy）	06h30m	差分を取得可能なテーブル
事前データのIMPORT (作業Instance -> Aurora)		
3. 手順1のデータのIMPORT（psql -f）	00h03m	
4. 手順2のデータのIMPORT（¥copy）	11h00m	
5. インデックス作成	14h00m	
当日データのEXPORT（作業Instance <- pg83)		
6. pg_dump	01h00m	差分を取得できないテーブル
7. 手順2の残りのデータをEXPORT（¥copy）	01h00m	差分を取得可能なテーブル
当日データのIMPORT（作業Instance -> Aurora)		
8. 手順6のデータのIMPORT（psql -f）	05h10m	
9. 手順7のデータのIMPORT（¥copy）	02h10m	


移行検証

作業Instance：t2.medium（2vCPU, 4GB）
pg83：c5d.2xlarge（8vCPU, 16GB）
Aurora：db.r5.large（2vCPU, 16GB）

- ベースラインの取得結果は以下のとおり。
  - 事前移行：34h　当日移行：9h　（※整合性チェック時間を含まない）

実施内容	時間	対象テーブル
事前移行データのEXPORT (作業Instance <- pg83)		
1. pg_dump	00h03m	差分が発生しないテーブル
2. 事前移行対象データのEXPORT（¥copy）	06h30m	差分を取得可能なテーブル
事前データのIMPORT (作業Instance -> Aurora)		
3. 手順1のデータのIMPORT（psql -f）	00h03m	
7. 手順2の残りのデータをEXPORT（¥copy）	01h00m	差分を取得可能なテーブル
当日データのIMPORT（作業Instance -> Aurora）		
8. 手順6のデータのIMPORT（psql -f）	05h10m	
9. 手順7のデータのIMPORT（¥copy）	02h10m	

目標値の設定：
失敗を想定し、再走行するための時間を捻出する必要がある。
事前移行：12h以内　当日移行：6h以内

移行検証

作業Instance：t2.medium（2vCPU, 4GB）
pg83：c5d.2xlarge（8vCPU, 16GB）
Aurora：db.r5.large（2vCPU, 16GB）

- ベースラインの取得結果は以下のとおり。
  - 事前移行：34h　当日移行：9h　（※整合性チェック時間を含まない）

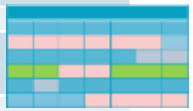
実施内容	時間	対象テーブル
事前移行データのEXPORT (作業Instance <- pg83)		
1. pg_dump	00h03m	差分が発生しないテーブル
2. 事前移行対象データのEXPORT（¥copy）	06h30m	差分を取得可能なテーブル
事前データのIMPORT (作業Instance -> Aurora)		
3. 手順1のデータのIMPORT（psql -f）	00h03m	
4. 手順2のデータのIMPORT（¥copy）	11h00m	
7. 手順2の残りのデータをEXPORT（¥copy）	01h00m	差分を取得可能なテーブル
当日データのIMPORT（作業Instance -> Aurora)		
8. 手順6のデータのIMPORT（psql -f）	05h10m	
9. 手順7のデータのIMPORT（¥copy）	02h10m	

Next Step：
効果の大きい項目を優先的に解析し、処理時間を削減する。

事前移行対象データのEXPORT処理時間の改善

- ベースラインの取得結果は以下のとおり。
  - 事前移行：34h　当日移行：9h　（※整合性チェック時間を含まない）

実施内容	時間	対象テーブル
事前移行データのEXPORT (作業Instance <- pg83)		
1. pg_dump	00h03m	差分が発生しないテーブル
2. 事前移行対象データのEXPORT（¥copy）	06h30m	差分を取得可能なテーブル
事前データのIMPORT (作業Instance -> Aurora)		
3. 手順1のデータのIMPORT（psql -f）	00h03m	
4. 手順2のデータのIMPORT（¥copy）	11h00m	
5. インデックス作成	14h00m	
当日データのEXPORT（作業Instance <- pg83)		
6. pg_dump	01h00m	差分を取得できないテーブル
7. 手順2の残りのデータをEXPORT（¥copy）	01h00m	差分を取得可能なテーブル
当日データのIMPORT（作業Instance -> Aurora)		
8. 手順6のデータのIMPORT（psql -f）	05h10m	
9. 手順7のデータのIMPORT（¥copy）	02h10m	



事前移行対象データのEXPORT処理時間の改善

解析対象：2. 事前移行対象データのEXPORT（¥copy）

- ボトルネックとなっている箇所を特定するために
  - Amazon CloudWatchを利用してリソースの使用状況を確認する。

CPU

作業Instance

約8%
（2vcpu）

pg83

約15%
（4vcpu）

→　¥copy実行中、CPUはボトルネックにはなっていない。

事前移行対象データのEXPORT処理時間の改善

解析対象：2. 事前移行対象データのEXPORT（¥copy）

ディスク

作業Instance（idle）

作業Instance（書き込み帯域）

→　ディスクにもボトルネックは見られない。

※本資料には未掲載ですが、実際はこれらの他にNW帯域の使用状況、ディスクについては作業Instanceだけではなく、pg83側についても調査し、ボトルネックが無いことを確認していました。

今回確認した範囲では、Amazon CloudWatchからは
特に問題なる箇所は見られなかった。

では、何が原因で処理が長時間化しているのか？

事前移行対象データのEXPORT処理時間の改善

解析対象：2. 事前移行対象データのEXPORT（¥copy）

- ボトルネックの原因
  - 今回の手順では、事前移行対象データのEXPORTはpsql経由での「¥copy」で行っているが、psqlは仕様上、結果セットをメモリ上で作成している。
  - つまり、事前移行対象のテーブル（数百GBのデータ）の抽出をわずか4GBの中で行っていたことになり、メモリ不足がボトルネックの原因だった可能性が高いと考えられる。

- 実施した対策
  - 対策1：作業Instanceのインスタンスタイプをメモリ重視に変更
    - t2.medium(4GB) -> r5.xlarge(32GB)
  - 対策2：EXPORT処理の並列化
    - Amazon CloudWatchの確認結果からリソース（CPU、NW、ディスク）に余裕があることがわかったため、並列化により時間当たりの処理量増加を行う。

ボトルネック解消による効果

- 作業Instanceの強化、EXPORT処理の並列化実施による効果
  - 処理時間を　6h30m　から　2h00m　に短縮できた。

CPU（pg83）

約15%
（4vcpu）

CPUを使い切れるようになった

約100%
（4vcpu）

ディスク（作業Instance）

6MiB/s

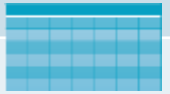
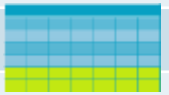
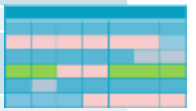
書き出し量はベースラインの
およそ3～4倍になった

20MiB/s

手順2のデータのIMPORT処理時間の改善

- ベースラインの取得結果は以下のとおり。
  - 事前移行：34h　当日移行：9h　（※整合性チェック時間を含まない）

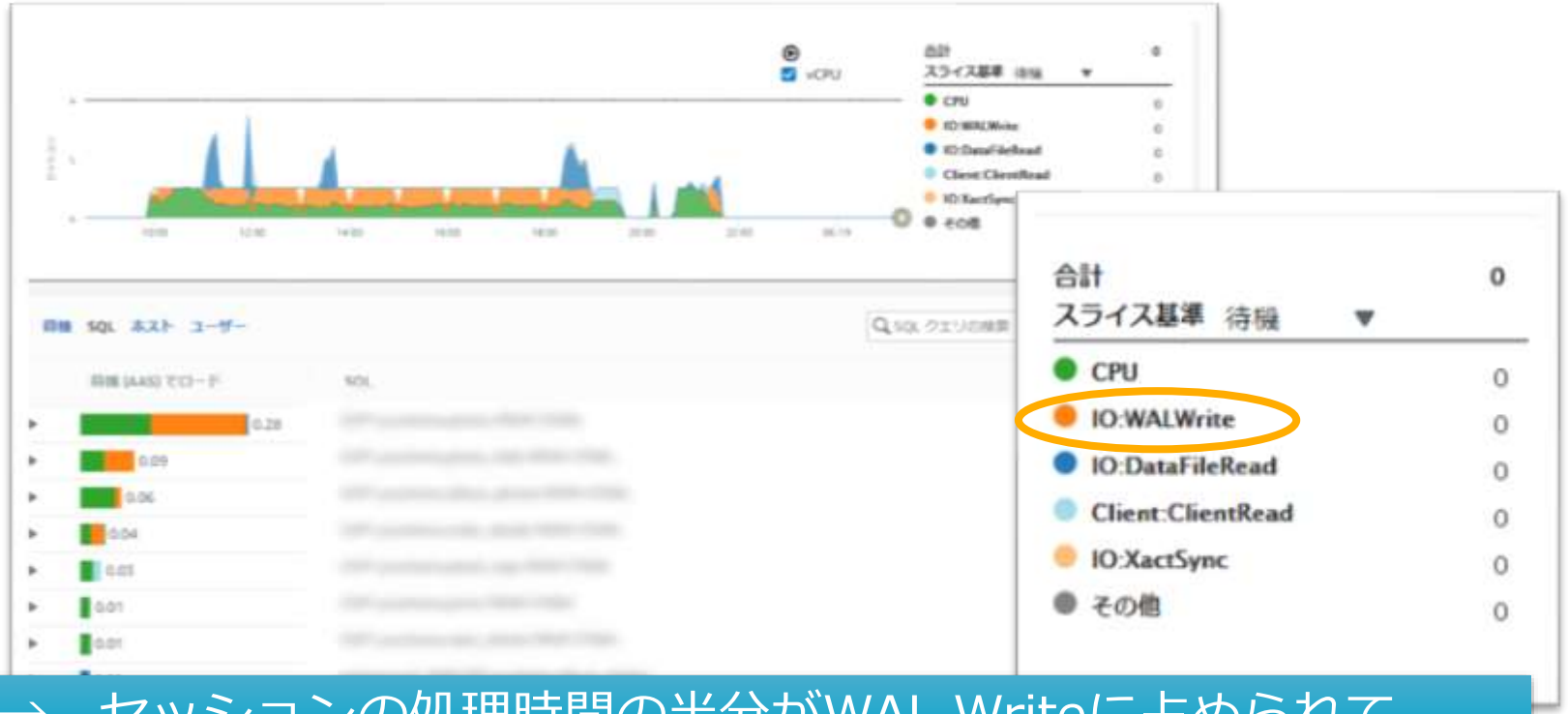
実施内容	時間	対象テーブル
事前移行データのEXPORT (作業Instance <- pg83)		
1. pg_dump	00h03m	差分が発生しないテーブル
2. 事前移行対象データのEXPORT（¥copy）	06h30m	差分を取得可能なテーブル
事前データのIMPORT (作業Instance -> Aurora)		
3. 手順1のデータのIMPORT（psql -f）	00h03m	
4. 手順2のデータのIMPORT（¥copy）	11h00m	
5. インデックス作成	14h00m	
当日データのEXPORT（作業Instance <- pg83)		
6. pg_dump	01h00m	差分を取得できないテーブル
7. 手順2の残りのデータをEXPORT（¥copy）	01h00m	差分を取得可能なテーブル
当日データのIMPORT（作業Instance -> Aurora)		
8. 手順6のデータのIMPORT（psql -f）	05h10m	
9. 手順7のデータのIMPORT（¥copy）	02h10m	


手順2のデータのIMPORT処理時間の改善

解析対象：4. 手順2のデータのIMPORT（¥copy）

- Performance InsightsからAmazon AuroraでのSQLの実行状況を確認する。



→　セッションの処理時間の半分がWAL Writeに占められていることからWAL書き込みがボトルネックと考えられる。

手順2のデータのIMPORT処理時間の改善

解析対象：4. 手順2のデータのIMPORT（¥copy）

- ボトルネックの原因
  - WALの書き込み待ちが発生しているため、単純にディスク性能が不足しているか、他の処理と競合していることが考えられる。

- 実施した対策
  - 対策1：Amazon Auroraのインスタンスタイプの強化
    - db.r5.large（2vCPU、16GB） -> db.r5.large（8vCPU、64GB）
    - Amazon Auroraのインスタンスタイプを強化することでWALの書き込み待ちが解消できることが、別検証の結果として出ていたため、本対策を実施した。
  - 対策2：IMPORT処理の並列化
    - Performance Insightsの情報から対策1の効果（セッションの実行状況にWAL Writeがなくなったこと）を確認できたため、空きCPUを有効利用するために並列化を実施しました。

ボトルネック解消による効果

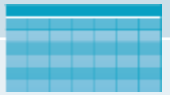
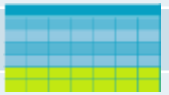
- Amazon Auroraインスタンスの強化、IMPORT処理の並列化実施による効果
  - 処理時間を　11h00m　から　3h00m　に短縮できた。

→　ベースラインの1/4で処理が終えられるようになった。
処理の並列化による新たにWAL Writeも出ているが
十分に処理時間を短縮することができた。

インデックス作成の処理時間の改善

- ベースラインの取得結果は以下のとおり。
  - 事前移行：34h　当日移行：9h　（※整合性チェック時間を含まない）

実施内容	時間	対象テーブル
事前移行データのEXPORT (作業Instance <- pg83)		
1. pg_dump	00h03m	差分が発生しないテーブル
2. 事前移行対象データのEXPORT（¥copy）	06h30m	差分を取得可能なテーブル
事前データのIMPORT (作業Instance -> Aurora)		
3. 手順1のデータのIMPORT（psql -f）	00h03m	
4. 手順2のデータのIMPORT（¥copy）	11h00m	
5. インデックス作成	14h00m	
当日データのEXPORT（作業Instance <- pg83)		
6. pg_dump	01h00m	差分を取得できないテーブル
7. 手順2の残りのデータをEXPORT（¥copy）	01h00m	差分を取得可能なテーブル
当日データのIMPORT（作業Instance -> Aurora)		
8. 手順6のデータのIMPORT（psql -f）	05h10m	
9. 手順7のデータのIMPORT（¥copy）	02h10m	

インデックス作成の処理時間の改善

解析対象：5. インデックス作成

- Performance InsightsからAmazon AuroraでのSQLの実行状況を確認する。

→　セッションの処理時間中、問題無く動作していた。
（ボトルネックは発生していない。）

インデックス作成の処理時間の改善

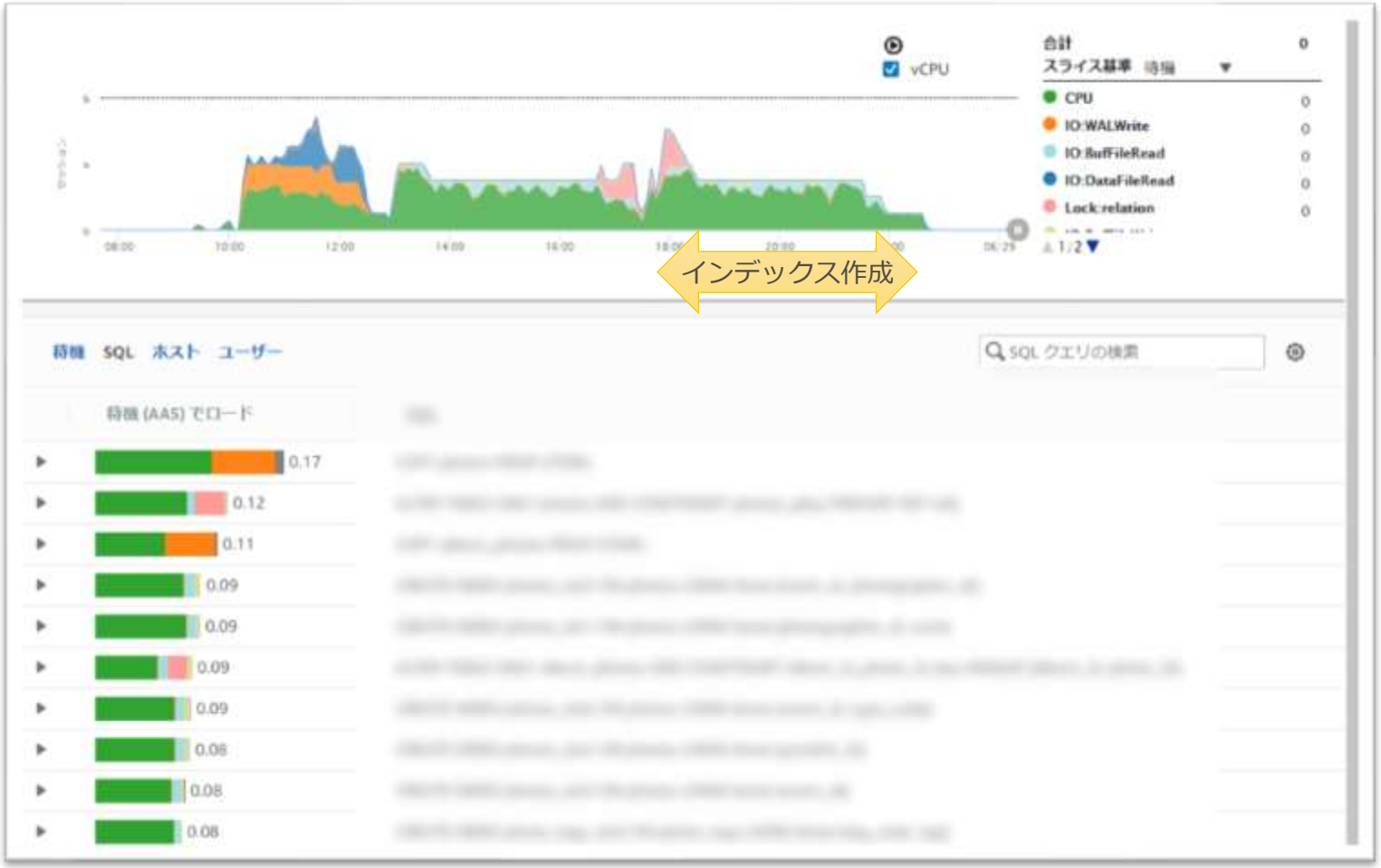
解析対象：5. インデックス作成

- ボトルネックの原因
  - 特になし。
  - Amazon CloudWatchからリソース情報も確認したが、DB側のリソースはまだまだ余裕があり、ボトルネックは発生していない。

- 実施した対策
  - 対策：複数のテーブルに対するインデックス作成プロセスの並列化
    - CREATE INDEXは実行時、対象テーブルにはSHAREロックを取得して行うため、インデックス作成は並列化することが可能。
    - ただし、ALTER TABLEの制約付与はACCESS EXCLUSIVEのため、並列実施は不可。

ボトルネック解消による効果

- IMPORT処理の並列化実施による効果
  - 処理時間を　14h00m　から　4h00m　に短縮できた。



当日移行対象データのIMPORT処理時間の改善

- ベースラインの取得結果は以下のとおり。
  - 事前移行：34h　当日移行：9h　（※整合性チェック時間を含まない）

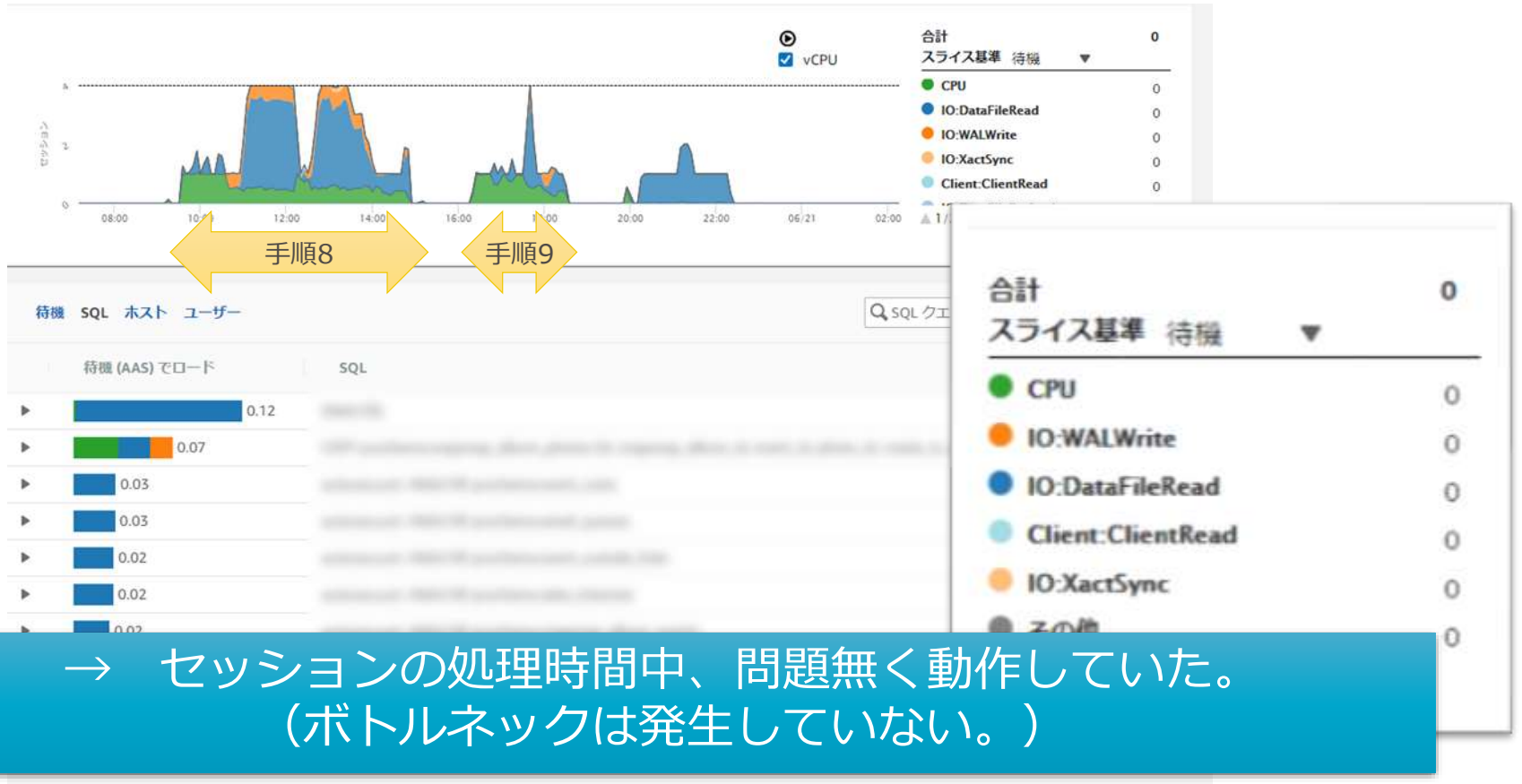
実施内容	時間	対象テーブル
事前移行データのEXPORT (作業Instance <- pg83)		
1. pg_dump	00h03m	差分が発生しないテーブル
2. 事前移行対象データのEXPORT（¥copy）	06h30m	差分を取得可能なテーブル
事前データのIMPORT (作業Instance -> Aurora)		
3. 手順1のデータのIMPORT（psql -f）	00h03m	
4. 手順2のデータのIMPORT（¥copy）	11h00m	
5. インデックス作成	14h00m	
当日データのEXPORT（作業Instance <- pg83)		
6. pg_dump	01h00m	差分を取得できないテーブル
7. 手順2の残りのデータをEXPORT（¥copy）	01h00m	差分を取得可能なテーブル
当日データのIMPORT（作業Instance -> Aurora)		
8. 手順6のデータのIMPORT（psql -f）	05h10m	
9. 手順7のデータのIMPORT（¥copy）	02h10m	

当日移行対象データのIMPORT処理時間の改善

解析対象：8.手順6のデータのIMPORT（psql -f）

9. 手順7のデータのIMPORT（¥copy）

- Performance InsightsからAmazon AuroraでのSQLの実行状況を確認する。



→　セッションの処理時間中、問題無く動作していた。
（ボトルネックは発生していない。）

当日移行対象データのIMPORT処理時間の改善

解析対象：8. 手順6のデータのIMPORT（psql -f）

9. 手順7のデータのIMPORT（¥copy）

- ボトルネックの原因
  - 特になし。
    - Amazon CloudWatchも確認したが、DB側のリソースはまだまだ余裕があり、ボトルネックは発生していないことが確認できた。

- 実施した対策
  - 対策：IMPORT処理の並列化
    - ベースライン測定ではシリアルなデータロードを行っている。
    - データはテーブル単位にEXPORTしているため、IMPORT処理が競合することはないので、並列実施により単位時間あたりの処理量を増やすことが可能。

最終的な改善結果

実施内容	ベースライン	改善後
事前移行データのEXPORT (作業Instance <- pg83)		
1. pg_dump	00h03m	00h03m
2. 事前移行対象データのEXPORT（¥copy）	06h30m	02h00m
事前データのIMPORT (作業Instance -> Aurora)		
3. 手順1のデータのIMPORT（psql -f）	00h03m	00h03m
4. 手順2のデータのIMPORT（¥copy）	11h00m	02h40m
5. インデックス作成	14h00m	04h00m
当日データのEXPORT（作業Instance <- pg83）		
6. pg_dump	01h00m	00h20m
7. 手順2の残りのデータをEXPORT（¥copy）	01h00m	01h00m
当日データのIMPORT（作業Instance -> Aurora）		
8. 手順6のデータのIMPORT（psql -f）	05h10m	1h40m
9. 手順7のデータのIMPORT（¥copy）	02h10m	1h00m

目標：12h以内

事前移行所要時間：
34h → 9h

目標：6h以内

当日移行所要時間：
9h → 4h

まとめ

1. 最適な移行方式の選定と移行計画
2. 検証によるボトルネックの特定
3. 効果の高いところに対する課金
4. 性能改善の目標値を定める

tips

移行検証時に遭遇したトラブル

no-localeデータベースのpublicスキーマの権限

- お客様の依頼でデータベースの権限分掌を検討時に発生
- 遭遇した事象
  - Amazon Auroraではno-localeデータベースのpublicスキーマはrds_adminがOWNERとなるため、publicスキーマの権限をREVOKEできない。
    - PostgreSQLにはデフォルトで作成されるpublicスキーマが存在する。
    - publicスキーマではデフォルトでどのROLEもCREATE権限を持つため、権限分掌を検討する場合、publicスキーマの権限はREVOKEする必要がある。
- 実施した対策
  - 移行先では新しいスキーマを作成し、テーブルは新スキーマに対して定義するように変更した。
    - 元の環境では、superuserの権限をもつROLEで操作を行っており、オブジェクトはpublicスキーマに作成していた。
    - search_pathのパラメータを変更するpublicスキーマを使わないようにしている。（publicスキーマへの権限は変更できないため根本解決は困難。）

8並行インデックス作成時の一時領域溢れ

- 移行時間の圧縮のため、並列度をフルに上げた手順での移行についても確認を行った際に発生
- 遭遇した事象
  - 以下のエラーメッセージが出力され、並列で実施していたインデックス作成の処理が失敗していた。

```
ERROR:  could not write block 1998929 of temporary file: No space left on device
```

- 実施した対策
  - 打つ手はなく、並列度を下げて実施した。
    - Amazon Auroraは自動で領域を拡張する仕様であるため、一時領域等、ユーザからは手が出すことはできない。
    - オンプレミス環境であれば、一時ファイルの領域を指定できるので一時ファイルのサイズの見積もりができれば対応可能。