ちょっぴりDiveDeepするAWSの時間
AWS Dev Day 2023 Tokyo 延長戦

実践データ移行 ～はてなダイアリーや魔法のiらんどの事例と共に～

株式会社はてな 組織・基盤開発本部 サービスプラットフォーム部
瀧澤 崇

自己紹介

- 瀧澤 崇
- 2015年11月入社
- 業務歴
  - カクヨム（KADOKAWA様）
  - GigaViewer
  - はてなダイアリー終了

自己紹介

引越し経歴

- はてなダイアリー終了
  - はてなダイアリー → はてなブログ
- 魔法のiらんどリニューアル
- その他にもデータ連携等

本日お話しすること

本日お話しすること

- 2020年4月にリニューアルした「魔法のiらんど」のデータ移行
- 事例から得た「データ移行」の際に考慮すべきこと

魔法のiらんど

魔法のiらんど

- KADOKAWA様が運営する「日本最大級のガールズエンタテイメントサイト」
- 1999年に無料ホームページ作成サービスとしてスタート
- 2020年4月に小説に特化したサービスとしてリニューアル

2020年のリニューアル

- システム刷新
- 新システムへの小説データ等の移行
    - 今回主にお話しするのはこちらの話題

データ移行

- 旧システムから新システムへのデータ移動
  - 単に「データベースを移す・新しくする」というような話ではない
  - 旧システムのデータを新システムでも使えるように移動

「旧システムのデータを新システムでも使えるように移動」とは

- 旧システムと新システムのスキーマをマッピングする
  - マッピングした先にデータを移動
- データの変換
  - 例）小説の記法が新旧システム間で変わっている

魔法のiらんどデータ移行概観

- データ量そのものが多い（20年分）
- データ間の依存関係が多い
  - 作者 > 作品 > エピソード
- データ移行そのものにかかる時間が読みづらい
  - 作者に紐づく作品数、作品に紐づくエピソード数が違う
  - 記法変換

データ間の依存関係と処理の順番

- ユーザー移行 -> 作品移行 -> 記法変換 という順で移行する
  - 「紐づく先」がないと移行できない
  - 例）ユーザーがいないと作品が移行できない

やりたいこと

- 依存するデータを順番に移行したい
  - 前が終わったら次の処理を開始してほしい
- 並列に移行処理を進行したい
  - 移行時間を短くしたいため

やりたいこと

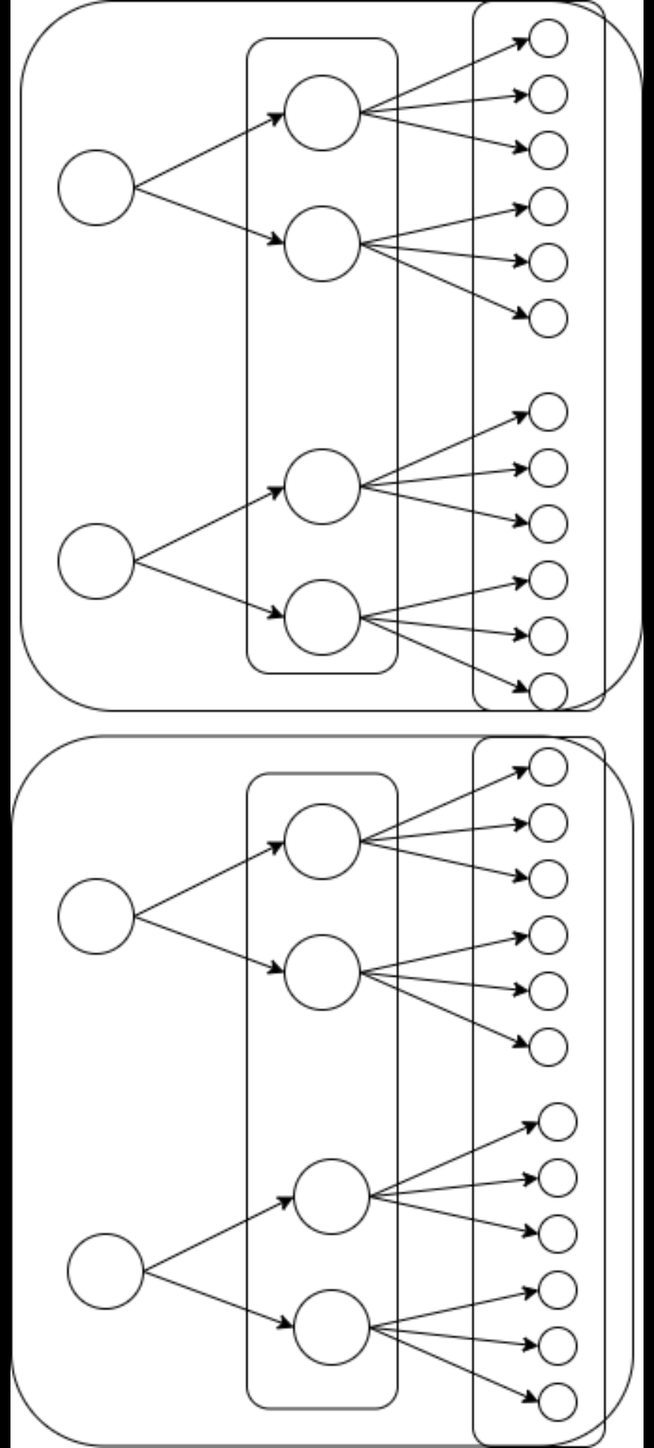
- 失敗した場合のリカバー手段を確立したい
  - 手動でもいいのであとでリトライしたい
- バッチサイズをできるだけ揃えたい
  - 作業時間の見積り精度を高めるため

StepFunctions + AWS Batch + S3

- AWS StepFunctions + AWS Batch で依存関係のある移行処理を並列実行し一気に終わらせる作戦
  - StepFunctionsで依存関係のある各工程を順に処理するワークフローを作る
  - 移行処理自体はAWS Batchで実行
- S3
  - 移行元と移行先が書かれたJSONをS3に置く

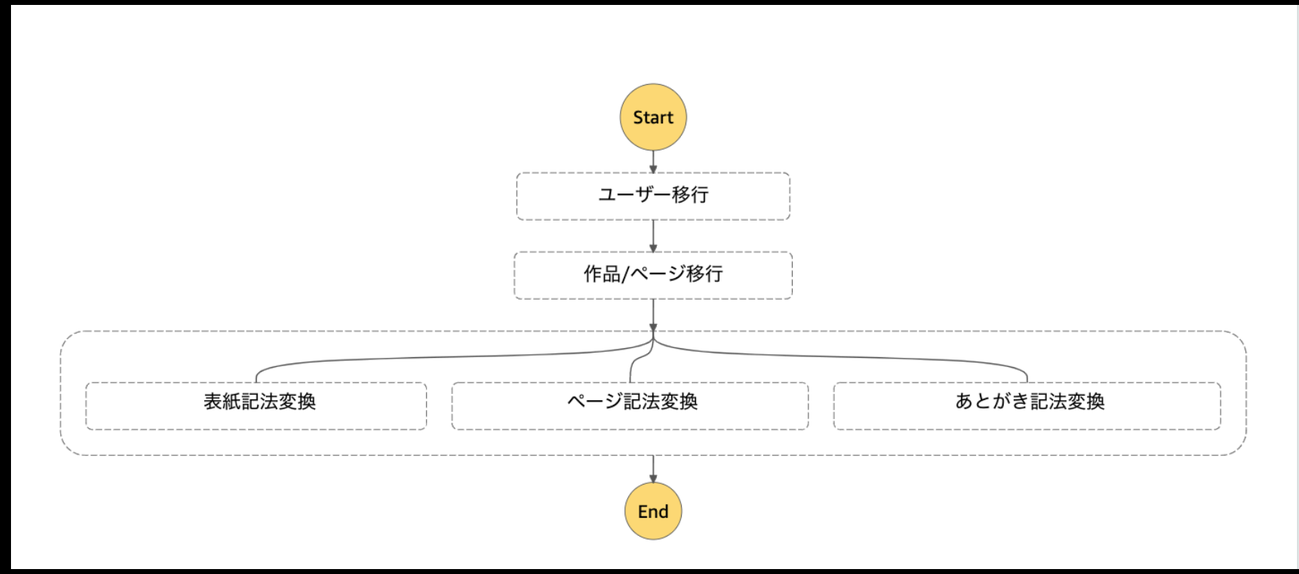
移行の作戦

- 並列実行可能な単位でグルーピングしてバッチサイズをできるだけ揃える
  - サイズを揃えることで移行対象数に差がある場合のバラつきを減らす
- 各グループの実行状態をS3に保存しておき、進捗確認・リトライ可能な仕組みを作る
  - 実行状態はS3のキーで表現する、各処理にはS3のキーだけ渡す



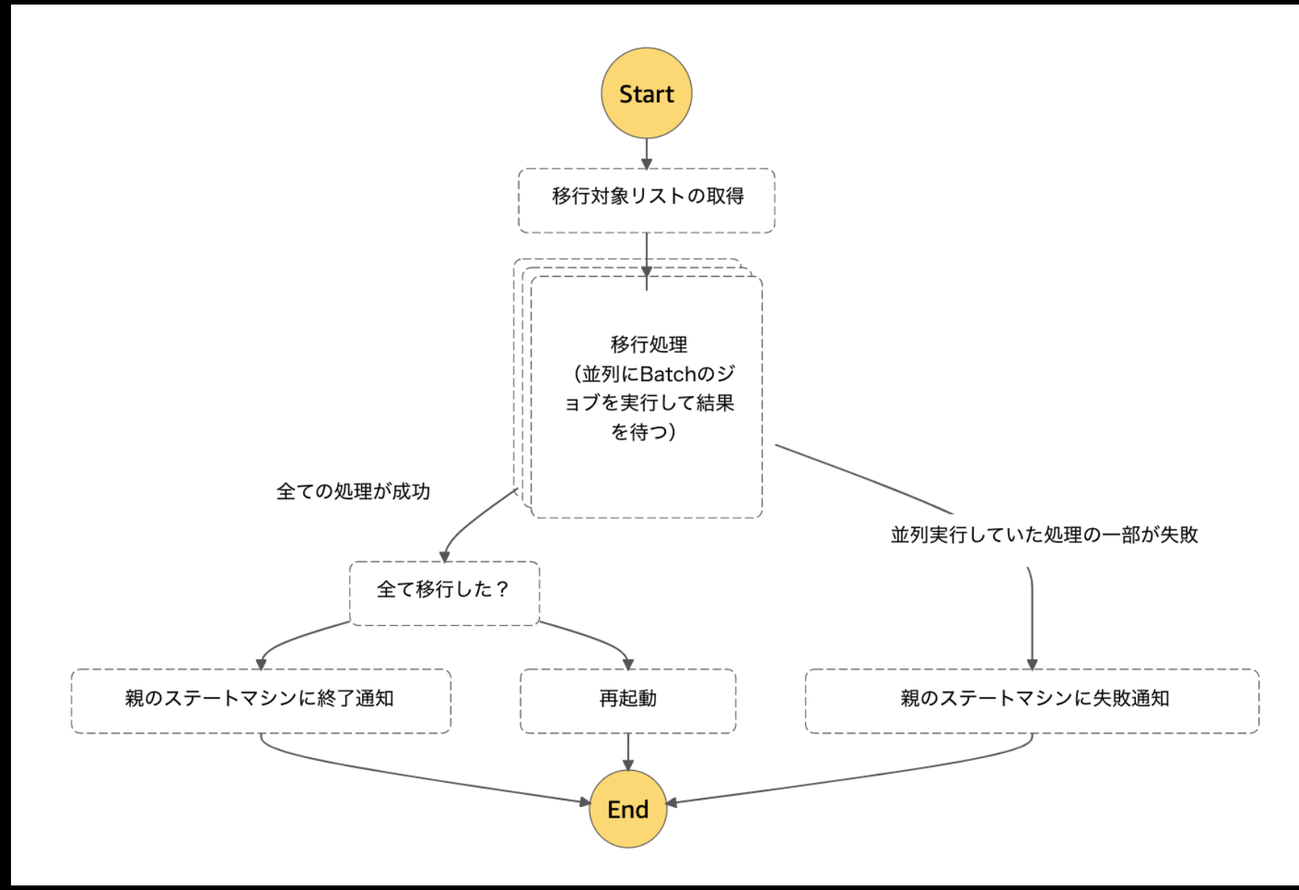
ステートマシン

- ステートマシンを二段構成にする
  - 各工程を親で管理し、細かい処理は子に寄せる
- 親ステートマシン
  - 工程間の依存を管理する



ステートマシン

- 子ステートマシン
  - 細かい処理はこちらに書かれている
  - batchジョブを並列に大量発行



Savepoint

- S3におかれる、「移行元/移行先」が記録されているJSON
  - S3のキーを渡すと処理のはじめにJSONを取得してきて、移行処理を行う
- だいたい同じ件数（batch_size）になるように均されている
  - それぞれ並列実行可能

Savepoint

- S3のキーで「移行工程/グループ/移行状態」を表現する
- `{$batch_type}/batch_id={$batch_id}/status={$status}/{$uuid}`
  - batch_type: 移行工程
  - batch_id: 全工程で不変の値で、グループ分けのためのID
  - status: 移行状態。pending/succeeded/failed
  - uuid: 全体でのユニークなID

Savepoint

- /batch_id={$batch_id}/status={$status} をトレースすることで進捗がわかる
  - どのグループがどこまで進んでいるかがわかる
  - statusをトレースすると成功/失敗の件数と残り件数がわかる
- failedなキーについては移行処理修正後にキーをpendingにすることで再実行可能

結果

- 20年分の小説をだいたい合計10h程度で記法変換まで完了して移行可能な仕組みができた
  - ユーザー移行 -> 小説移行 -> 記法変換を順番に移行できた
- バッチサイズが揃っており、時間の見積もりの精度を高めることができた
  - 本当に時間内に終わるか、の見通しをつけられた

結果

- エラー発生時のリトライも簡単だった
  - 失敗した処理がsavepointで記録されている
  - 再実行の場合キーのstatusをpendingに戻すだけ

（改めて）事例から学ぶデータ移行

移行準備

- 見積り
  - データ量 / 種類
- 移行順の検討
  - データの依存関係の洗い出し
- 変換処理実装

移行準備

- データ量見積り
  - 全体移行時間の見積り
  - メンテナンスウインドウの検討
- 種類
  - 何を移すか
  - 扱い方が変わるものは変換

移行準備

- 移行処理の高速化
  - bulk insert
- 移行順の検討/依存の確認
- 例) 小説の場合作者と作品が紐づく
  - 作者がいないと作品データが作れない（移せない）
  - 作者を移行した後に作品を移行する、という順番になる

まとめ

- AWSのマネージドサービスを利用して、依存のあるデータのデータ移行を高速かつリカバーしやすく構築できた
- データ移行するにあたっては事前の準備が重要
  - 全体把握 / 見積り
  - 移行処理実装

ありがとうございました