Engineer

移行

新しいバージョンやプラットフォームへ計画的に移すこと

どのようなものか

移行とは、既に使っているものを、新しいバージョンやプラットフォームへ段階的かつ計画的に移し、最後に古い方を廃止する作業です。対象は、ワークロード全体、アプリケーション、データ一式、データを保存する構造などです。作り直すこととの違いは、移動中も使い続ける点です。利用者、記録、連携先も引き継ぎ、新しいものが役割を引き継ぐまで、古いバージョンは動き続けます。

エンジニアは、この言葉を二つの規模で使います。どちらを指すか明示しないことも多くあります。

大きい方は、新しいプラットフォームや新しいメジャーバージョンへの移動です。MicrosoftのCloud Adoption Frameworkは、オンプレミスのサーバーや別のクラウドからAzureへワークロードを移す作業を、計画、ワークロードの準備、実行、最適化、移行元の廃止という五段階で扱います。Esriの顧客は、地図とデータをArcMapからArcGIS Proへ移します。Next.jsアプリケーションは、Pages RouterからApp Routerへ移り、`getServerSideProps` の代わりにServer Components内でデータを取得します。PostgreSQLクラスターは、新しいメジャーバージョンへ移ります。システムテーブルの構造が変わることが多いため、`pg_upgrade` か、ダンプと復元でデータを移します。

小さい方の移行は、データベースの構造に加える一つの変更です。バージョンを付け、リポジトリーのファイルとして保存し、ツールが決まった順に適用します。Djangoは、自らの移行を、データベーススキーマのバージョン管理システムと呼びます。各ファイルは、モデルの変更を記録し、コードと一緒にコミットされ、開発者のノートPC、ステージング、本番で同じ順に動きます。GitLabは、アップグレード後に大きなテーブルを少しずつ書き換える、バッチ単位のバックグラウンド移行を提供します。整数列を、あふれる前に `bigint` へ広げる例などです。データ移行もこちらに含まれます。列ではなく行を変えるもので、二つの古いフィールドから、新しいフィールドを埋めるといった処理です。

同じ考えを、二つの規模で行っています。どちらにも、既知の開始状態、順序のある手順、移行先、手順が失敗したときの戻り方があります。プラットフォームの移動は、通常、スキーマ移行を含みます。一方、長年のスキーマ移行の積み重ねは、利用者が使うプラットフォームを、何年もかけて少しずつ変えることになります。

概念を判断に生かす

「移行とは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 業務を止めずに、老朽化したシステムからサポートのある環境へデータを移します。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 PostgreSQLでGIS地図を10倍に高速化 — 10x MSSQLからの移行で速くなった地図

PostgreSQLでGIS地図を10倍に高速化 お問い合わせ →

なぜ重要なのか

ソフトウェアは、作った当時の基盤より長く残ります。クラウド契約は再交渉され、フレームワークは仕組みを変え、提供元は製品を廃止します。十年間使い続けるシステムは、その間に何度も移行します。そのため、最初にどれだけ良いプラットフォームを選んだかより、チームがどれだけうまく移行できるかの方が、最後には重要になります。

- **費用がかかるのは、新旧に分かれた期間です。** 一部が古いプラットフォーム、一部が新しい方で動く間は、両方を運用、監視し、一貫した状態に保つ必要があります。Cloud Adoption Frameworkは、その期間を短くし、直接依存する部品を同じ移行グループとしてまとめて移すよう勧めます。
- **データベースは、再デプロイでは置き換えられません。** コードは再ビルドして置き換えられますが、本番の行ではできません。スキーマ移行は、使用中の構造の変更を、レビューし、コピーに対して試し、順に適用し、大半の変更では元に戻せるファイルにします。そうしなければ、本番のスキーマは、最後にコンソールを使った人が残した状態そのものになります。
- **計画した移動は、元に戻せます。** Cloud Adoption Frameworkは、移行を始める前にロールバックの基準と手順を決め、ロールバックを試し、途中に続行か中止かを判断する地点を置くよう求めます。

  利点と代償が集中するのは、切り替えです。Cloud Adoption Frameworkは、停止を伴う移行と、ほぼ停止しない移行を分けます。後者では、データを継続して複製し、最後に切り替えます。PostgreSQLのメジャーバージョン更新では、`pg_upgrade` が切り替えを短くします。既存のデータファイルを再利用し、この種の更新で通常必要なダンプと復元を省きます。ファイルをコピーしない `--link` モードでは、さらに大幅に速くなります。

実際の活用例

このサイトの三つのプロジェクトに、三つの規模が見られます。プラットフォーム全体から、一つのリポジトリーのファイルの順序までです。

一つは、クラウド提供元から離れる移行でした。顧客のAWS環境では、必要以上のリソースが確保されており、実際に何を使っているかの確認から始めました。不要なサービスを除き、残りの容量を調整した後、ワークロードはAWSから移りました。案件の記録が裏付けるのは、このインフラ変更です。移動前に使わないサービスを減らせば、運ぶものは減ります。ただし、同じ条件で比べた費用の比率、生産性の成果、停止のない移行を示す記録ではありません。

二つ目は、移行先の構造も新しく設計した、エンジンの変更でした。地図アプリケーションのデータベースを、MSSQLから、PostGIS拡張を加えたPostgreSQLへ移しました。移行先は、古い構造のコピーではありません。新しいスキーマは、空間データ向けに設計し、GiSTとSP-GiSTのインデックス、空間座標の検証規則を強制する厳密な外部キー制約とチェック制約を置きました。5,000万件を超える空間レコードを、EPSG:4326へ変換するETLパイプラインで移しました。新しい設計へ移ることは、データの規則をスキーマ自体に書く機会です。ページは、空間クエリー性能が10倍になり、バックエンドのCPU負荷が65%下がったと報告します。

三つ目は、何年も続けた、小さい方の移行です。一つのプラットフォームのスキーマは、036から1102まで番号を付けた1,022件の移行の積み重ねで、ページは、その順序を保つ規律を、地味でも譲れないものと呼びます。リポジトリーの履歴は、本番の状態の説明です。開発者が新しく作るコピーから、復元したものまで、どのデータベースでも、最初のファイルから最後まで再適用して作り直せる間だけ、その説明は正しく保たれます。

問題が起きるところ

計画は切り替えに集中しますが、問題は前後の数週間に集まりがちです。よくある形は四つあります。

新旧二つのシステムが、予定より長く並行稼働します。 移行が途中で止まり、難しい部分が残り、チームは何か月も両方を運用します。利用者にも継ぎ目が見えます。Next.jsのガイドは、Pages RouterのページとApp Routerのページの間の移動は、ページ全体を読み直す遷移になり、`next/link` は、両者をまたいで事前取得しないと記します。移行グループを小さくし、次の一歩の負担を小さくしま

す。ここのフロントエンドの一つは、一往復でページを読み込む共通のファクトリーへ、一ページずつ移しており、移したページは、すべてその動作を引き継ぎます。

切り替え後に戻れません。 移行先だけでなく、移行元も変える手順があります。`pg_upgrade --link` では、新しいクラスターが起動した時点で、古い方と共有するファイルへ書き込むため、古い方はバックアップから復元する必要が生じます。Esriは、ArcGIS Pro向けに更新したデータ集合が、ArcMapでは使えなくなる可能性を警告し、先にコピーするよう求めます。Djangoの `RunSQL` で、逆向きのSQLを書かなければ、元に戻す移行をした瞬間に `IrreversibleError` が出ます。最初の手順より前に、ロールバックの基準を決め、戻り方を練習し、検査にします。ここのリポジトリーの一つでは、検証項目に移行をロールバックできるかの検査を含むガードエンジンを動かしています。

実行済みの移行を編集します。 移行ツールは、どのファイルを適用したかを記録し、以後は飛ばします。本番へ届いたファイルを変えると、本番は古い版を保ち、新しいデータベースは変更後を受け取ります。一つの履歴に二つのスキーマが生まれます。適用済みの移行は変えず、新しい移行で直します。何もない状態から履歴を再適用し、移行済みのコピーと比べるビルドなら、そのずれを見つけられます。

更新初日にデータベースが遅くなります。 `pg_upgrade` は、プランナーの統計の大半を移しますが、全部ではありません。残りを作り直すまでは、プランナーは推測します。`--check` モードは何も変えず、古いサーバーが処理中でも動かせるため、先に実行します。更新直後に不足分の統計を再作成し、その後、残りを収集します。

```
pg_upgrade --check -b "$OLD/bin" -B "$NEW/bin" -d "$OLD/data" -D "$NEW/data"
pg_upgrade -b "$OLD/bin" -B "$NEW/bin" -d "$OLD/data" -D "$NEW/data"
vacuumdb --all --analyze-in-stages --missing-stats-only
vacuumdb --all --analyze-only
```

参考資料

- 移行を計画する — 移行グループ、切り替えの選択、ロールバック計画を含む、Azureへワークロードを移すMicrosoftの方法です。
- 移行 — バージョン管理したスキーマ変更、元に戻す方法、データ移行を扱う、Djangoのガイドです。
- pg_upgrade — 検査モード、リンクモード、元に戻す方法、その後の統計処理を含む、PostgreSQLの更新ツールです。

- PagesからApp Routerへ移行する方法 — Next.jsアプリケーションを、新しいルーターへ一ページずつ移す方法です。
- 既存の作業を移行する — ArcMapからArcGIS Proへ引き継げるもの、変換が必要なもの、先にバックアップするものです。
- 更新前に移行を確認する — 更新後にGitLabのバックグラウンド移行が大きなテーブルをバッチ単位で書き換える仕組みと、完了の確かめ方です。

用語集

実際の制約に合わせて選ぶ

代表的なデータセット、そこから答えを得たい問い、現在のボトルネックをお知らせください。品質、処理量、復旧の測り方を合意できます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「移行とは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く 連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/migration/