内製の著作権管理システムをGCPからAWSへ
― 移行を通して見えた「設計の真価」

株式会社毎日放送
山下 遼河

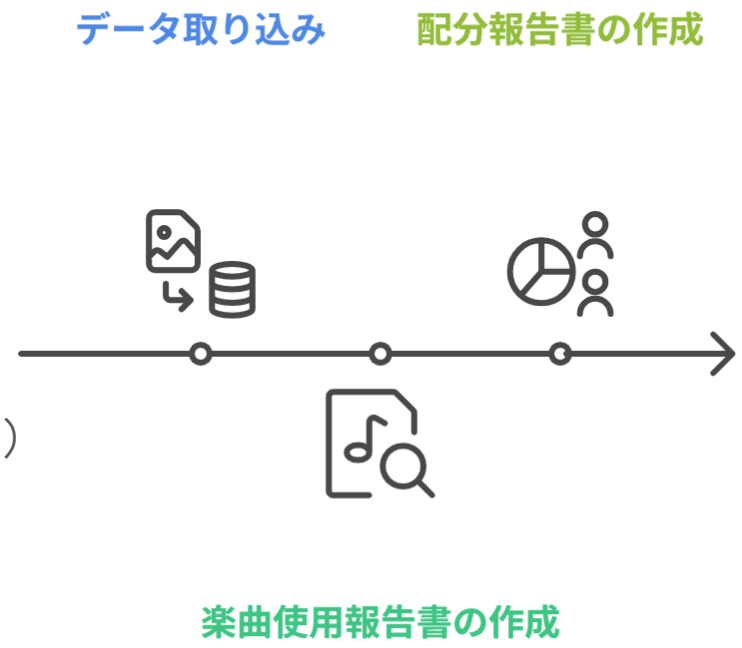
著作権管理システム

システムが担う業務

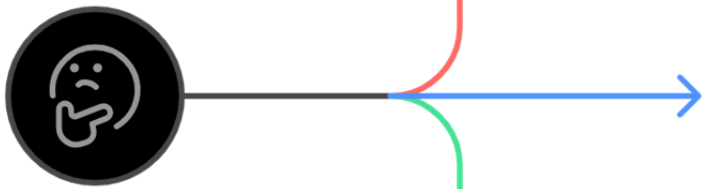
- データ取り込み(楽曲利用や再生数等)
- 楽曲の使用報告書の作成
- 配分報告書の作成

前提

- 内製・内部向け（外部公開サービスではない）
- データ処理/バッチが多く、変更も度々入る
- 「回り続けること」が価値の中心



内製の内部向けシステムの設計



Too Much

可用性・冗長化・オートスケールを盛りすぎる
作るのも運用するのも重い

Just Fit

要件にピッタリ合わせようとして設計が細かくなりすぎる
単純に設計をするのがしんどい
仕様変更や引き継ぎに弱い

Moderately Fit

完璧ではないが、壊れても戻せる・回し続けられる
「運用と開発が続けられること」を優先

内製の内部向けシステムの設計

Too Much

可用性・冗長化・オートスケールを盛りすぎる
作るのも運用するのも重い

Just Fit

要件にピッタリ合わせようとして設計が細かくなりすぎる
単純に設計をするのがしんどい
仕様変更や引き継ぎに弱い

Moderately Fit

完璧ではないが、壊れても戻せる・回し続けられる
「運用と開発が続けられること」を優先

今回目指した設計→

移行後システムの設計軸

1. 調査性

- システムが問題を起こしたとき、どこまで内部状態を把握し、原因に近づける設計になっているか

2. 同型性

- ローカル・ステージング・本番がどれくらい同じ構造で動くか

3. 複雑さの上限

- 今後の運用・引き継ぎ・変更に耐えられるよう、コストと許容できる複雑さのラインをどこに置くか

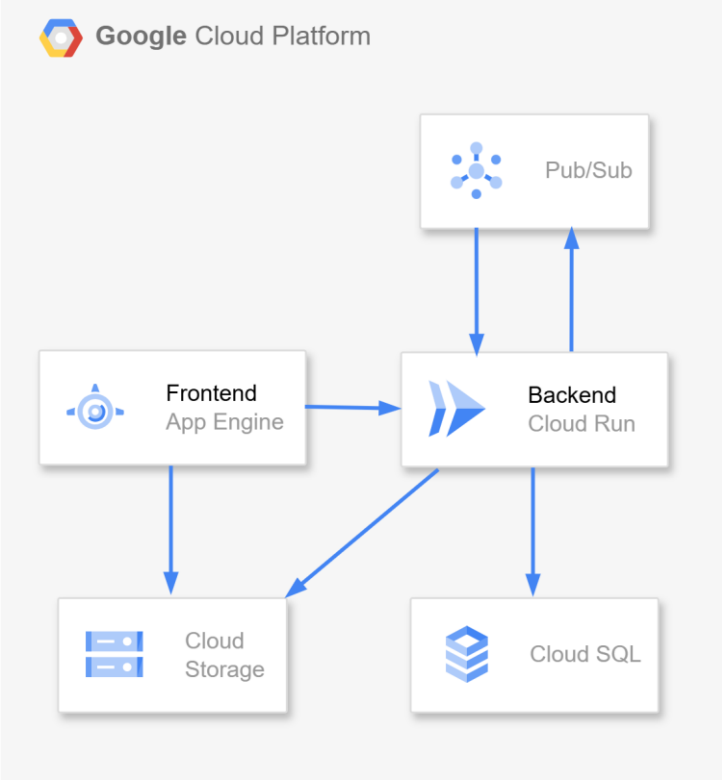
旧システム構成

Google Cloudを中心に構築

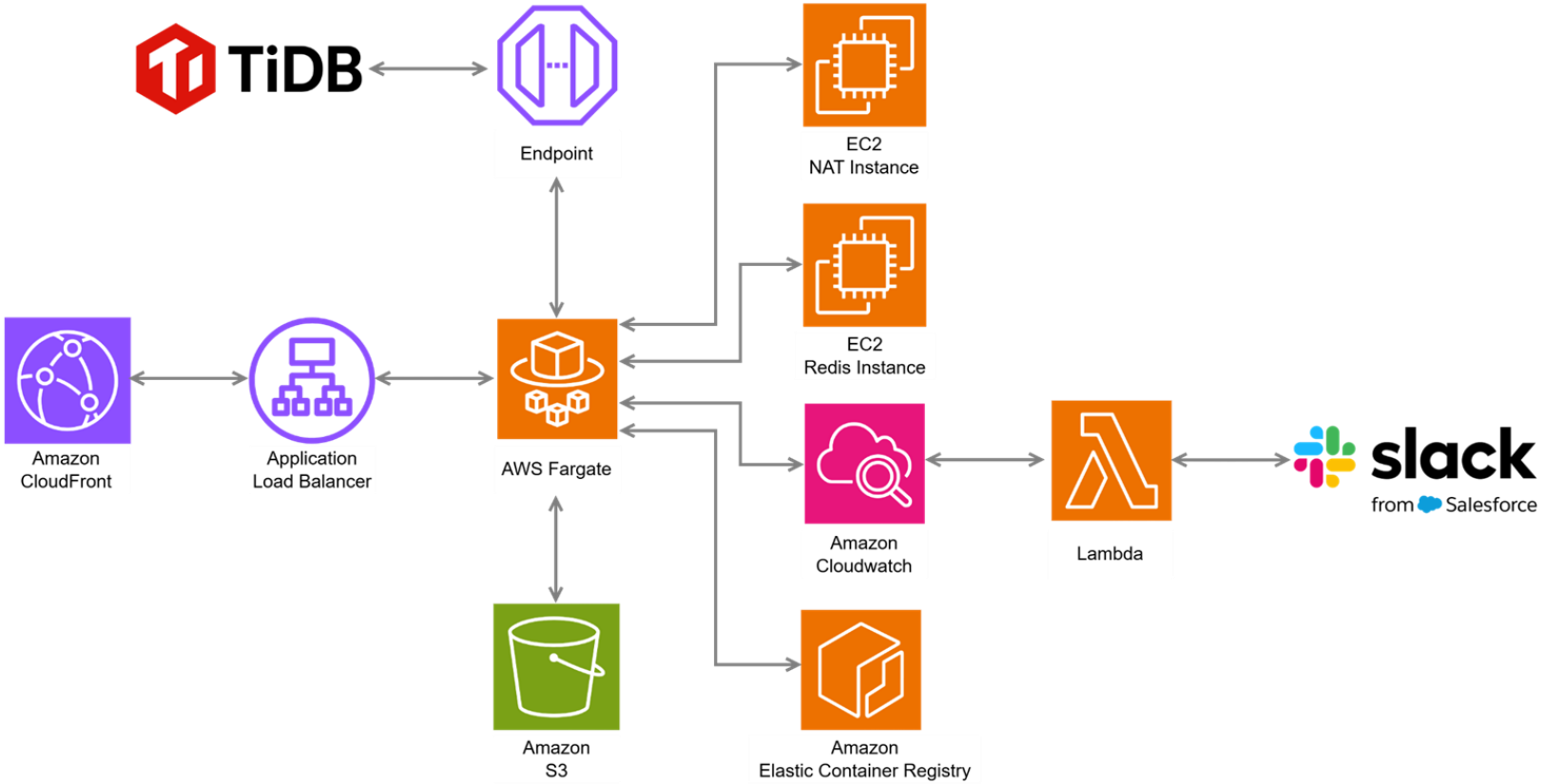
- Cloud Run + αのシンプル構成
- Rails + Reactのバック/フロントエンドで構築
- HTTPリクエスト駆動を前提とした構成

つらかったこと(例)

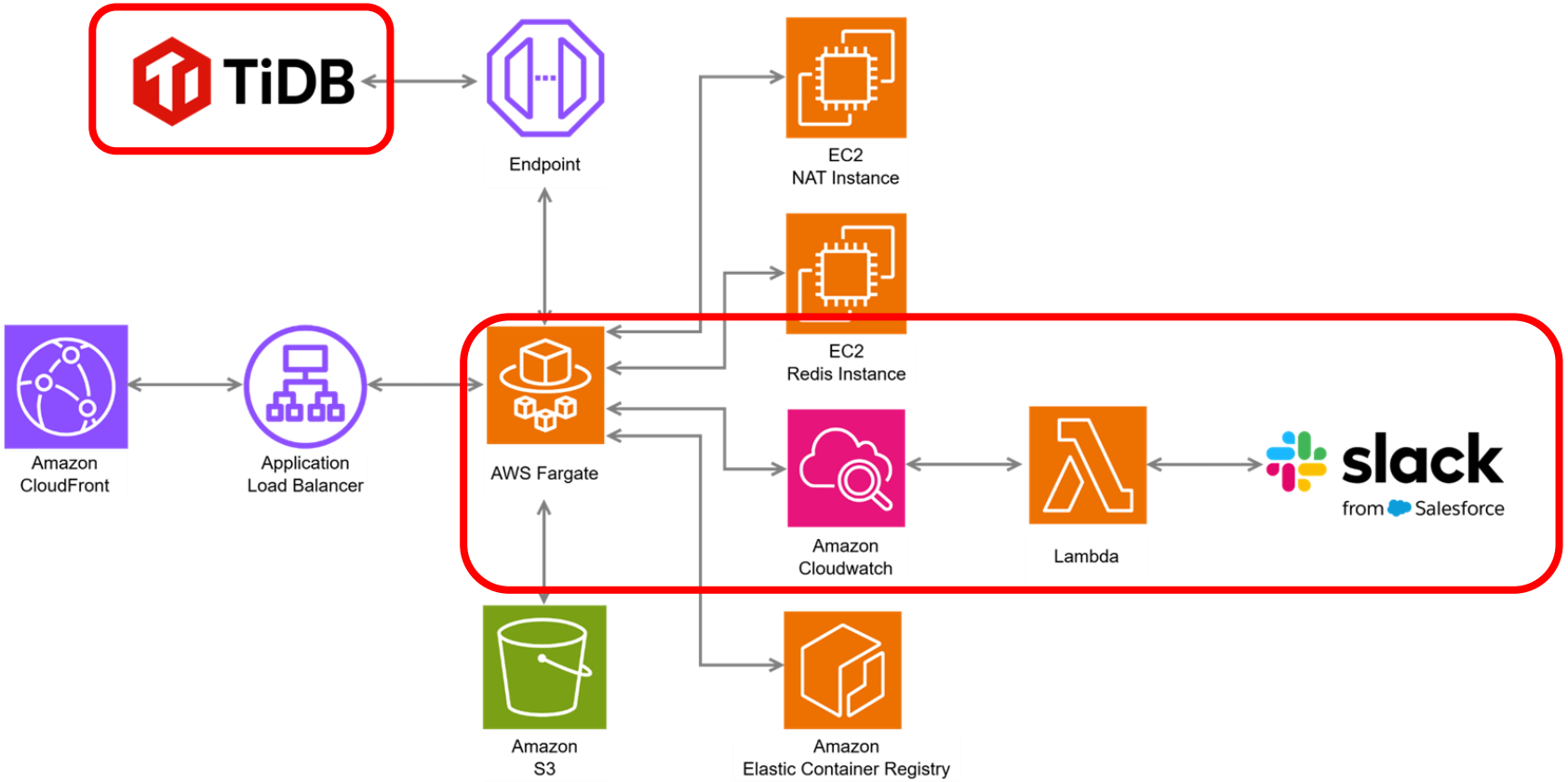
- 調査性の問題
  - コンテナに入っての作業ができない
- 非同期モデルの問題
  - 非同期処理がPub/Sub経由



新システム構成



判断軸① 調査性



判断軸① 調査性

ECS(Fargate)へのアプリのデプロイ

- コンテナに入ってのデバッグが可能
- 実行中コンテナの状態をそのまま確認できる

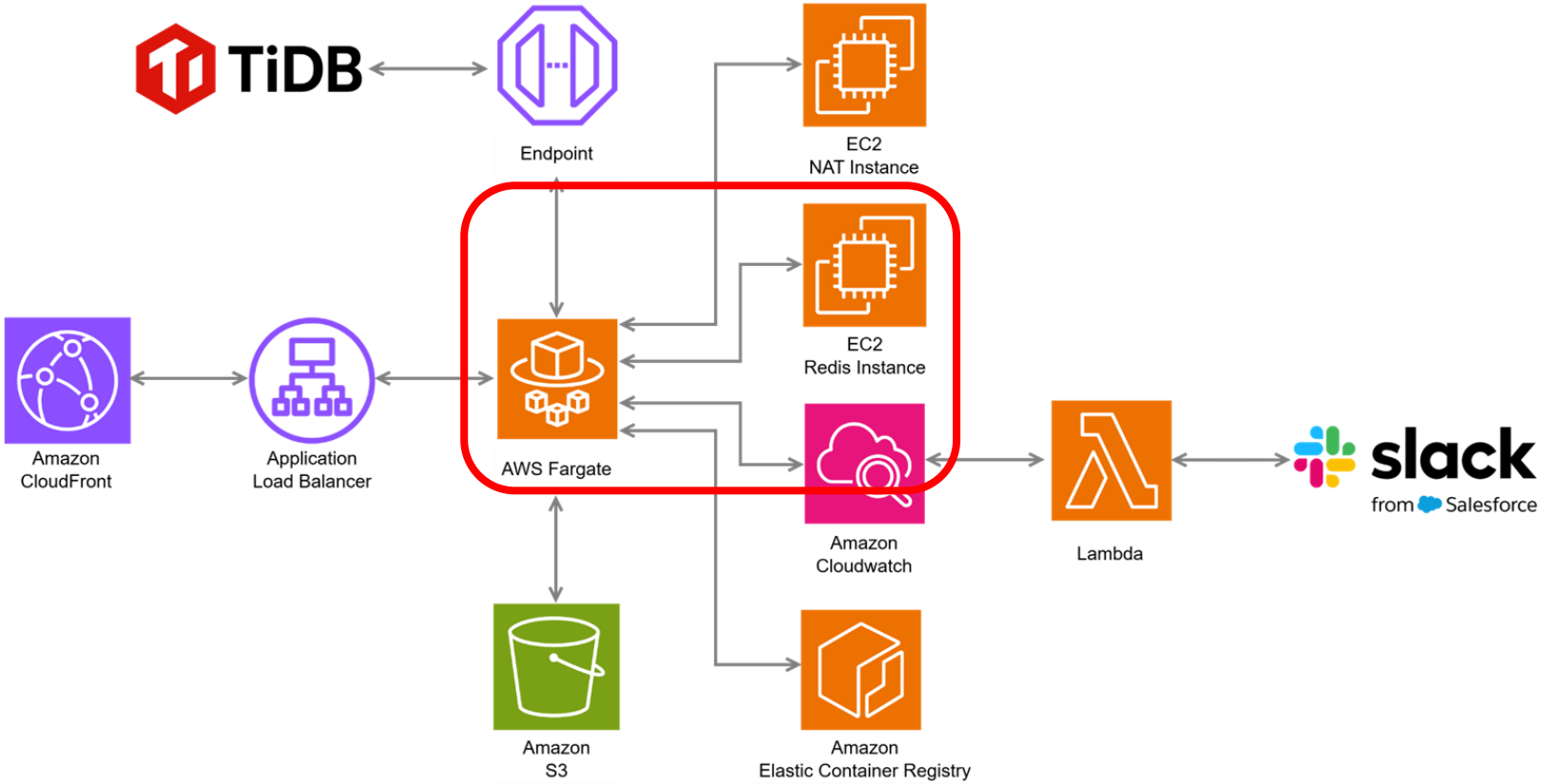
Slackへのエラー+ログ通知

- 何かが起こったとき簡単に確認ができる

TiDBの導入

- Webの管理画面からSQLを簡単に実行可能
- ボタン1つで現在のDBをコピーしたブランチDBが作成できる

判断軸② 同型性



判断軸② 同型性

非同期処理の前提を固定

- 非同期処理を Sidekiq + Redis に統一
- ローカルはコンテナで立てた Redis、本番は本番用 Redis に 向き先を変えるだけ

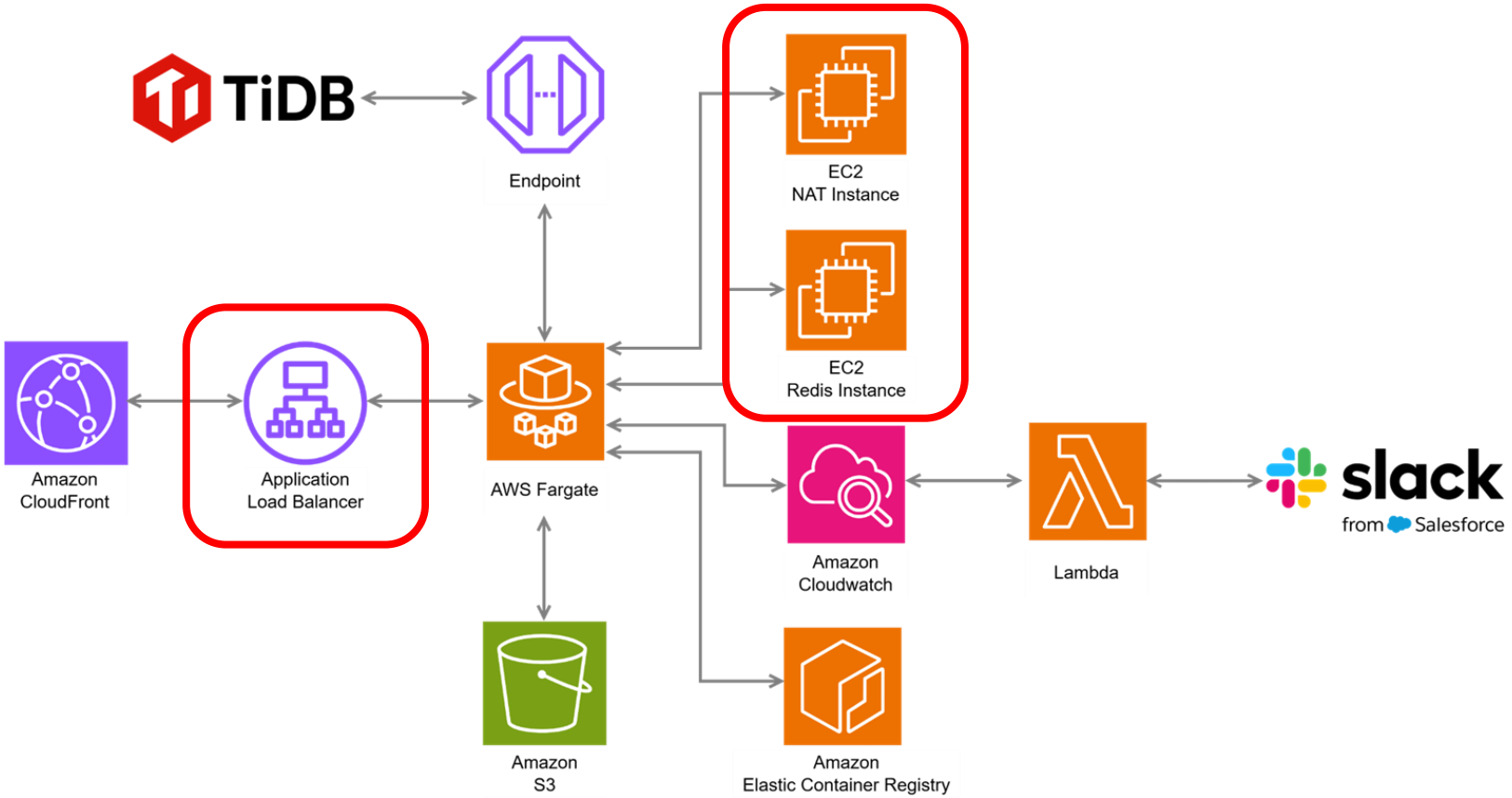
環境差異の少なさ

- ローカル／本番で 同じ非同期実行経路

開発・検証フローの一貫性

- ローカルで再現できた非同期挙動をそのまま本番に持ち込める
- 環境差異を前提にした特別な実装や検証手順が不要

判断軸③ 複雑さ上限



判断軸③ 複雑さ上限

NATはEC2で”必要十分”に寄せる

- NAT Gatewayは小規模利用はコスト的に過剰になる
- NATをEC2にしても複雑さは大して変わらない

RedisはEC2で、用途をSidekiqに限定

- Redisは非同期処理のタスクのキューイングのみの使用
- キューが消失しても再度入れたらよく業務影響が小さいため、
  可用性よりもコストの観点からEC2を選択

SPOF を“許容した上で管理する”

- SPOFをなくすのではなく、上手く付き合える設計・運用にする

曲芸→撤退

曲芸の導入

- 当初は規模のわりにはALBも過剰になると考えた
- そこで、NAT InstanceにFargate経由で立てたコンテナへのルートテーブルを動的に追加する(加えてCloudFrontの向き先をNAT Instanceにする)という実装を行った

撤退

- 作った後に”キッショ 誰が分かるんだよ”となった
- 後任の理解も大変だし、セキュリティ的にも微妙だしということで大人しくALBの導入をした
- Keep It Simple, Stupid

新システム移行時に同時に導入したこと

CDKによるIaCの導入

- 本番/ステージングをさくっとデプロイ

CI/CD周りの整備

- GitHub Actionsでpush時にテストが実行されるように
- 特定branchへのmergeをトリガーにデプロイを実行

Slack通知

- 必要最低限の簡単なエラー・ログを通知

まとめ

最適化より、継続可能性を最適化する

- 月額・性能・理想構成より、「楽に(自分も後任も)運用できる」を優先した

調査性・同型性・複雑さ上限を最初に決める

- 本番で何ができるか／ローカルと何を揃えるか／どこまで複雑にしていいか

壊れていいものとダメなものを分ける

- 非同期キューは再実行前提、でも業務データは守る

自動化は“効くところだけ”入れる

- CDK・CI/CD・Slack通知で、SREごっこはしない

TV局、テックブログやってます。

MBSテックブログ