Google Cloud

SmartHR が他社クラウドから移行後、Google Cloud を「使い倒した」成功体験

SmartHR
プロダクトエンジニア
佐藤 沢彦

自己紹介

2020 年に SmartHR 社に入社後、届出書類機能を担当。Google Cloud の移行や保守運用などにも従事。好きな Google Cloud の機能は Cloud Logging。

佐藤 沢彦
株式会社SmartHR
エンジニア

Table of Contents

今日の発表の背景と今日お話したいこと 01

SmartHR とは？ 02

SmartHR のアーキテクチャ 03

共通ログライブラリを作って Cloud Logging を使い倒した話 04

Cloud Run を使い倒して、PR ごとの検証環境を作った話 05

今後「使い倒し」て行きたい機能 06

01

今日の発表の背景と
今日お話したいこと

今日の発表の背景

- SmartHR は 2022 年 1 月に他社のクラウドから Google Cloud に移行しました
- 移行直後は基本的な機能を利用しており、あまり活用できていなかった
- しかし、最近は Google Cloud の機能を「使い倒す」ことができ始めた

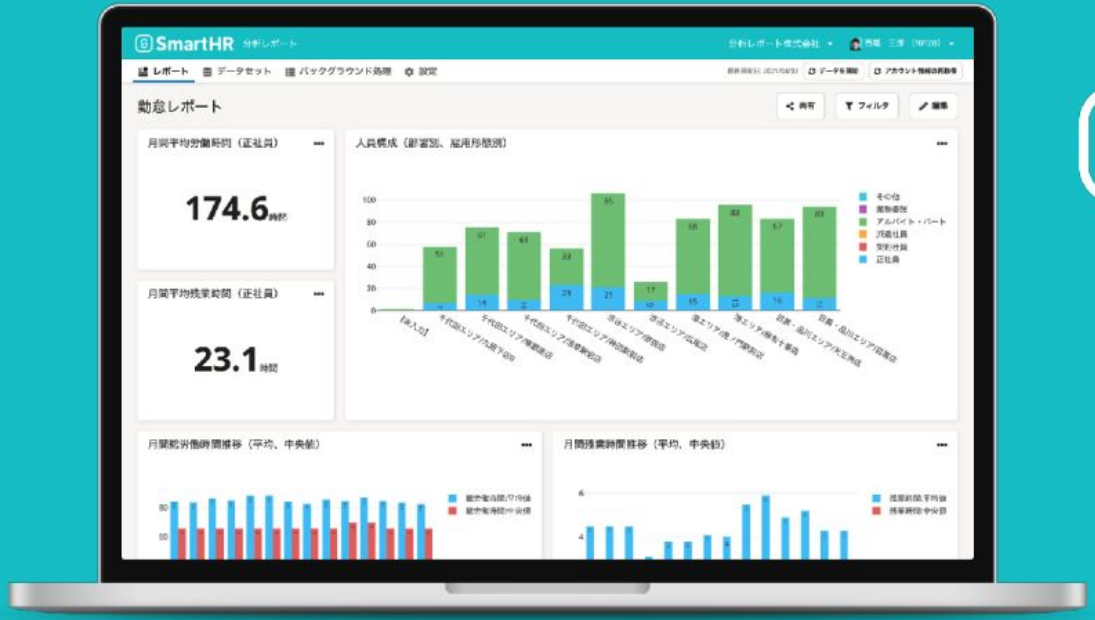
今日お話したいこと

SmartHR が Google Cloud の機能を「使い倒した」事例

- 共通ログライブラリを作って Cloud Logging を使い倒した話
- Cloud Run を使い倒して、PR ごとの検証環境を作った話

02

SmartHR とは？

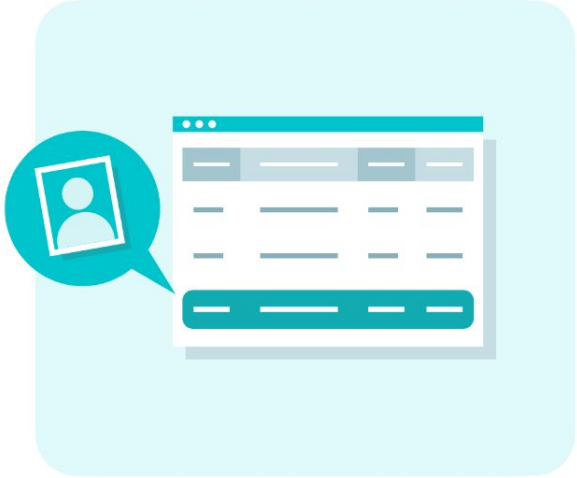


SmartHR

Employee First.

すべての人が、
信頼しあい、
気持ちよく働くために。

SmartHR は人事労務業務の効率化を通じて
生産性の向上・働きたい職場環境の創出を目的とした
クラウド型ソフトウェアです。



人事労務業務の効率化

人事データの一元化

人事データの活用

主な機能

SmartHR

労務管理

- 入社手続き
- マイナンバー
- 電子申請
- 年末調整
- 給与明細
- 社会保険
- お知らせ掲示板
- 多言語化対応
- 文書配付（オプション）
- 通勤経路検索（オプション）

人事データベース

- 従業員データベース
- 申請・承認
- 履歴・登録編集
- 予約管理
- CSVカスタムダウンロード
- 外部連携サービス

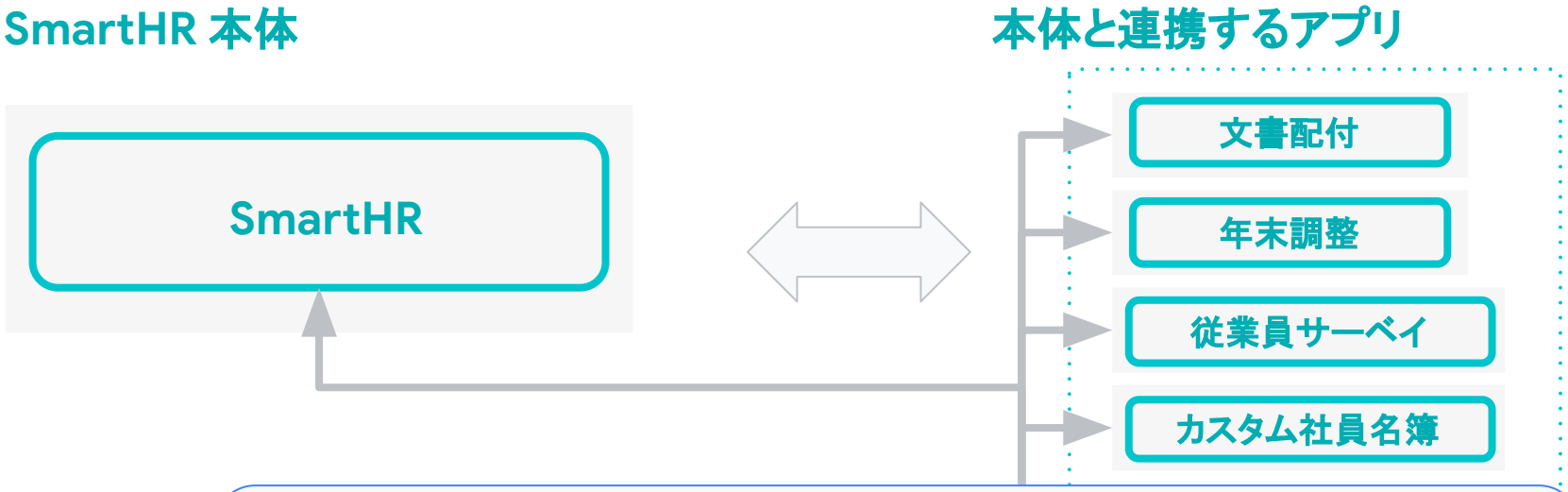
タレントマネジメント

- 組織図
- 社員名簿
- 分析レポート
- 従業員サーベイ
- 人事評価
- 配置シミュレーション

03

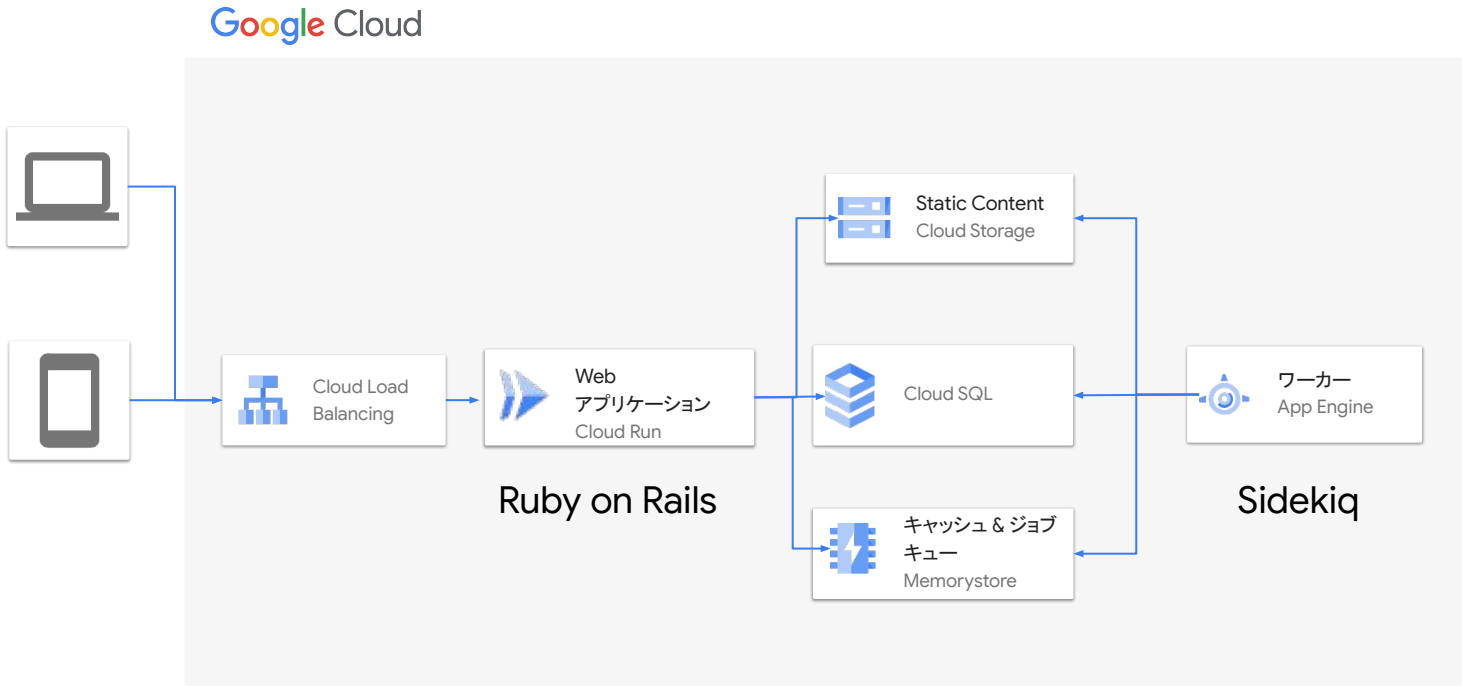
SmartHR のアーキテクチャ

SmartHR の構成



- SmartHR は独立した Web アプリケーションの集合
- Google Cloud のプロジェクトで分離

各アプリケーションの構成



SmartHR のインフラの歴史

ポイント

01 | SmartHR は元々 AWS と Heroku で動いていたが Google Cloud に移行した

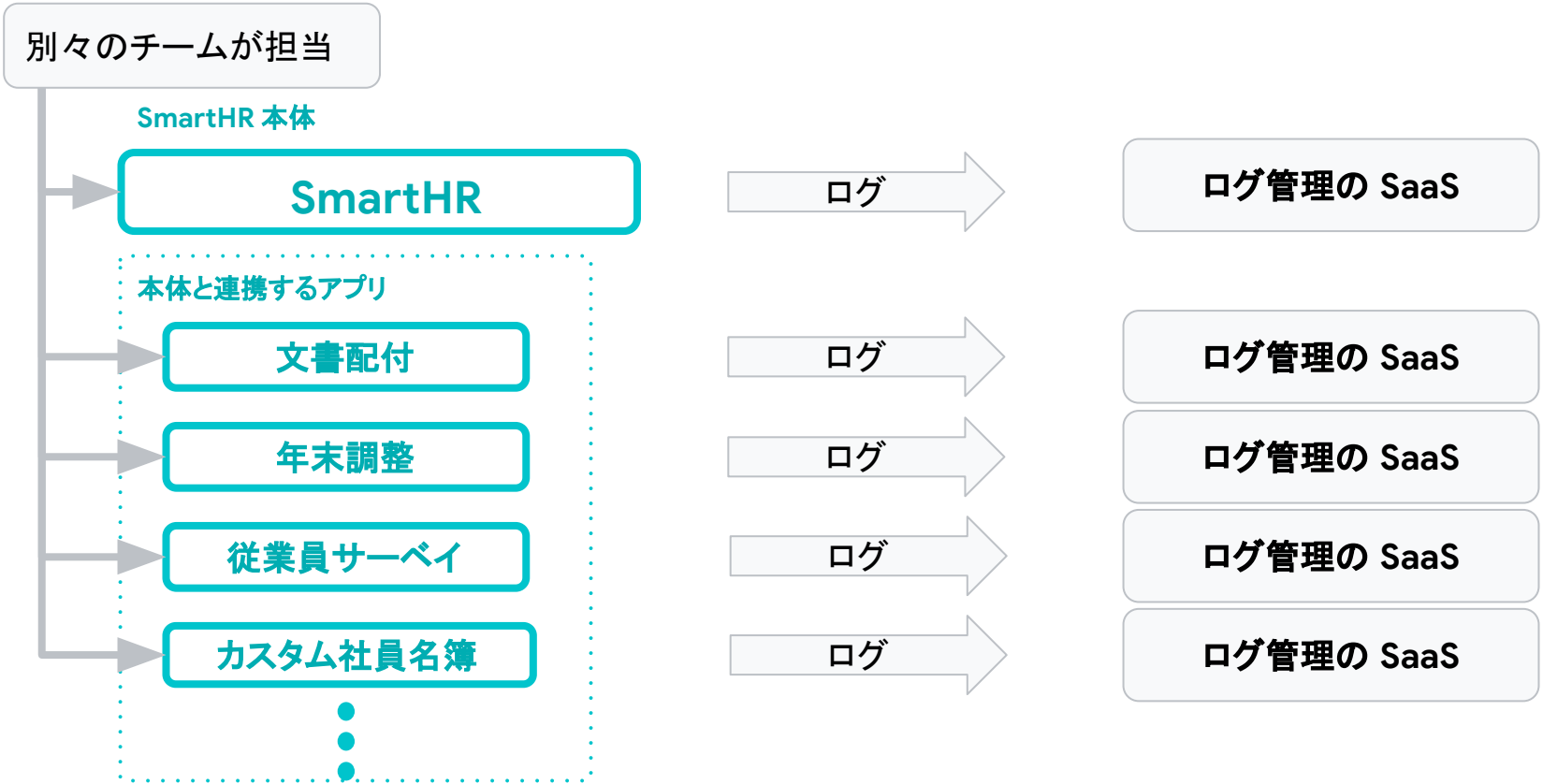
02 | SmartHR は複数の独立した Web アプリケーションの集合体

03 | それぞれのアプリケーションは、Web アプリケーション (Cloud Run) とワーカー (App Engine) を持っている

04

共通ログライブラリを作って Cloud Logging を使い倒した話

Google Cloud 移行前の SmartHR のログ



Google Cloud 移行前のログの課題

- アプリごとにログ出力先の環境がバラバラ
  - 各チームは自分のチームのログしか見れなかった
  - アプリを横断してログを追跡していくことができなかった
- 各チームが好きなようにログを出していた
  - ログのみやすさのための工夫は、チームごとに実施していた
  - 出力内容やフォーマットも統一されていなかった

ログの課題を解決するために行ったこと

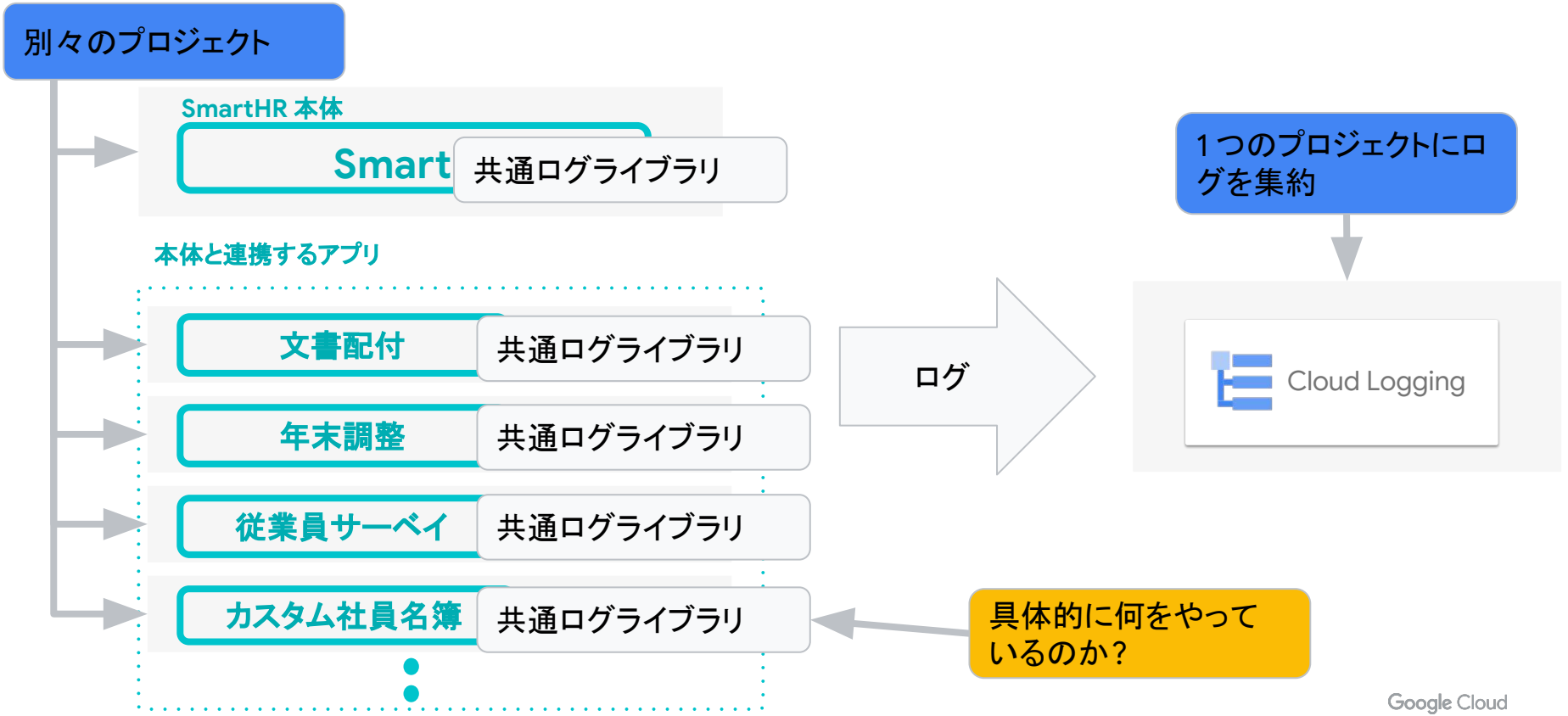
ログを集約するプロジェクトを用意

SmartHR の各アプリケーションの ログを集約するプロジェクトを作り、各アプリケーションのログが横断して見れるようにした

共通ログライブラリの導入

SmartHR の各アプリケーションに共通のログライブラリを導入することで、各チームがログに出力する情報やフォーマットを統一した

Google Cloud 移行後の SmartHR のログ



共通ログライブラリがやっていること

01 | 構造化ログの出力

02 | trace の付与

03 | trace のアプリ間での「引き回し」

構造化ログの出力

- Cloud Logging は構造化ログ (JSON) に対応
- 共通ログライブラリで構造化ログを出力する仕組みを作って置くことで全アプリが構造化ログを利用できるようになった
- 構造化ログは特定の属性に対する検索が可能でとても便利！

trace の付与

- Cloud Logging は trace という属性をログに出すと同じ trace の値のログのみでフィルターしてくれる
- 特定の HTTP リクエストによって生じたログにすべて同一の trace を設定しておくと、とてもログが見やすくなる

trace の付与

Web アプリケーション
Cloud Run

ワーカー
App Engine

- Cloud Run は trace などの情報の入った traceparent ヘッダーを付与してくれる
- 共通ログライブラリは traceparent ヘッダーをから trace を作ってログに出力

リクエストごとにログが絞り込める

- ワーカーの動かすジョブは必ずしもリクエストに紐づかない
- ジョブの id から trace を作ってログに出力

ジョブごとにログが絞り込める

trace のアプリ間での「引き回し」

trace のアプリ間での「引き回し」

共通ログライブラリを作って Cloud Logging を使い倒した話のまとめ

01 | SmartHR では複数のアプリで利用できる共通のログライブラリを作った

02 | 共通ログライブラリは、「構造化ログの出力」、「traceの付与」、「traceの引き回し」などの機能がある

03 | 共通ログライブラリによって、構造化ログや trace などの Cloud Logging の便利な機能が活用できるようになった

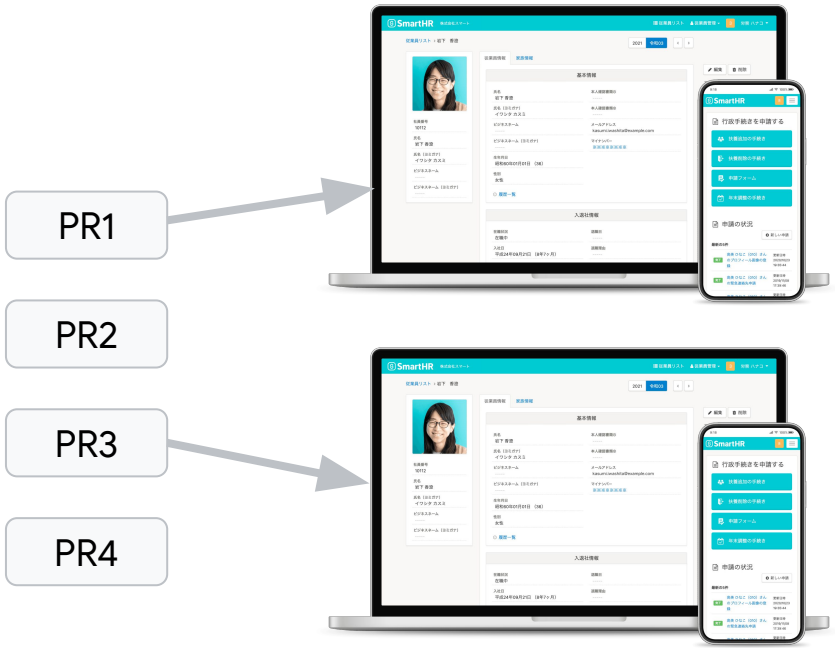
05

Cloud Run を使い倒して、PR ごとの検証環境を作った話

SmartHR のインフラの歴史のおさらい

Heroku Review Apps

- GitHub の PR ごとに使い捨ての検証環境を作ることができる機能



Heroku Review Apps の利点

01 | デザイナー、プロダクトオーナーなどの開発者以外の方に検証中の機能を見てもらいやすい

02 | 負荷試験や性能検証など共用の環境でやると迷惑がかかることもやりやすい

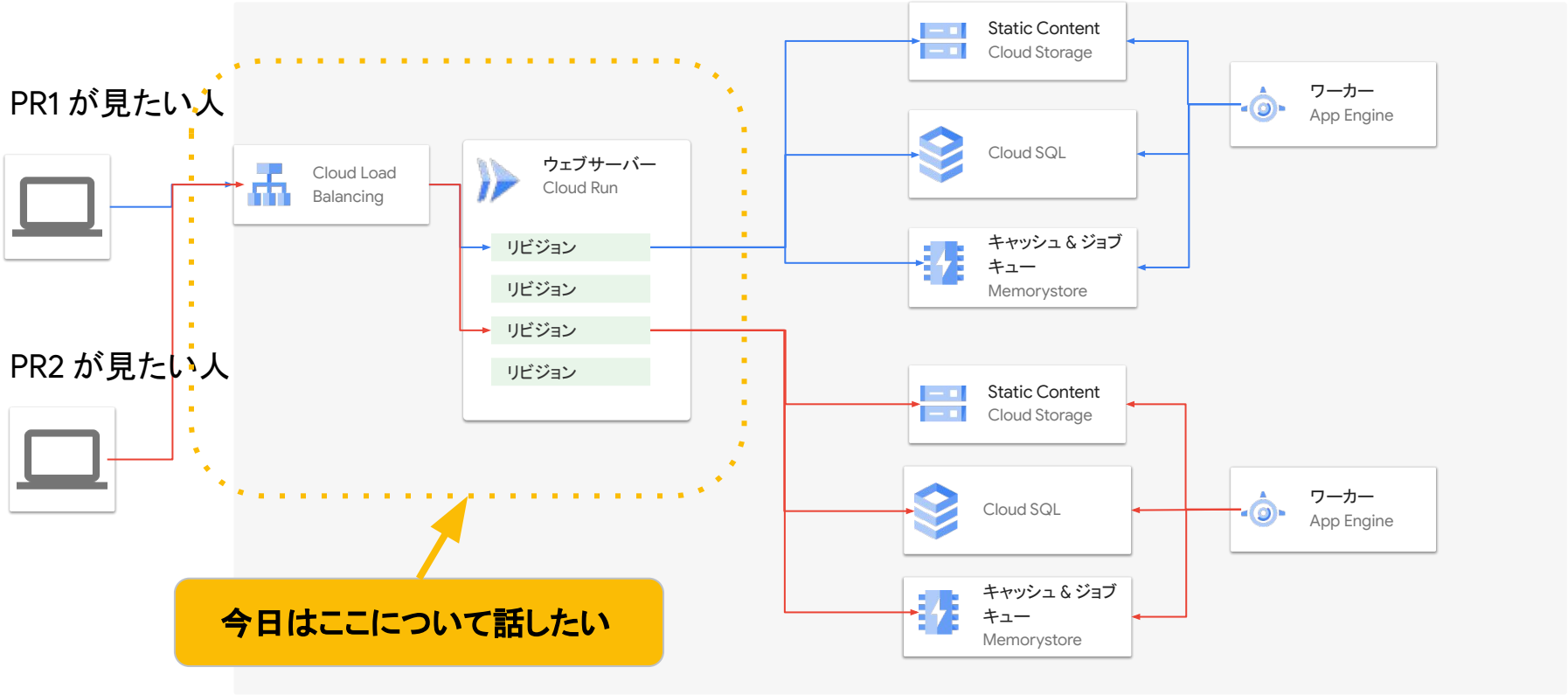
03 | 複数人が絡むような大規模なフィーチャーブランチの結合試験がやりやすい

Heroku Review Apps を使い続ける場合の問題点

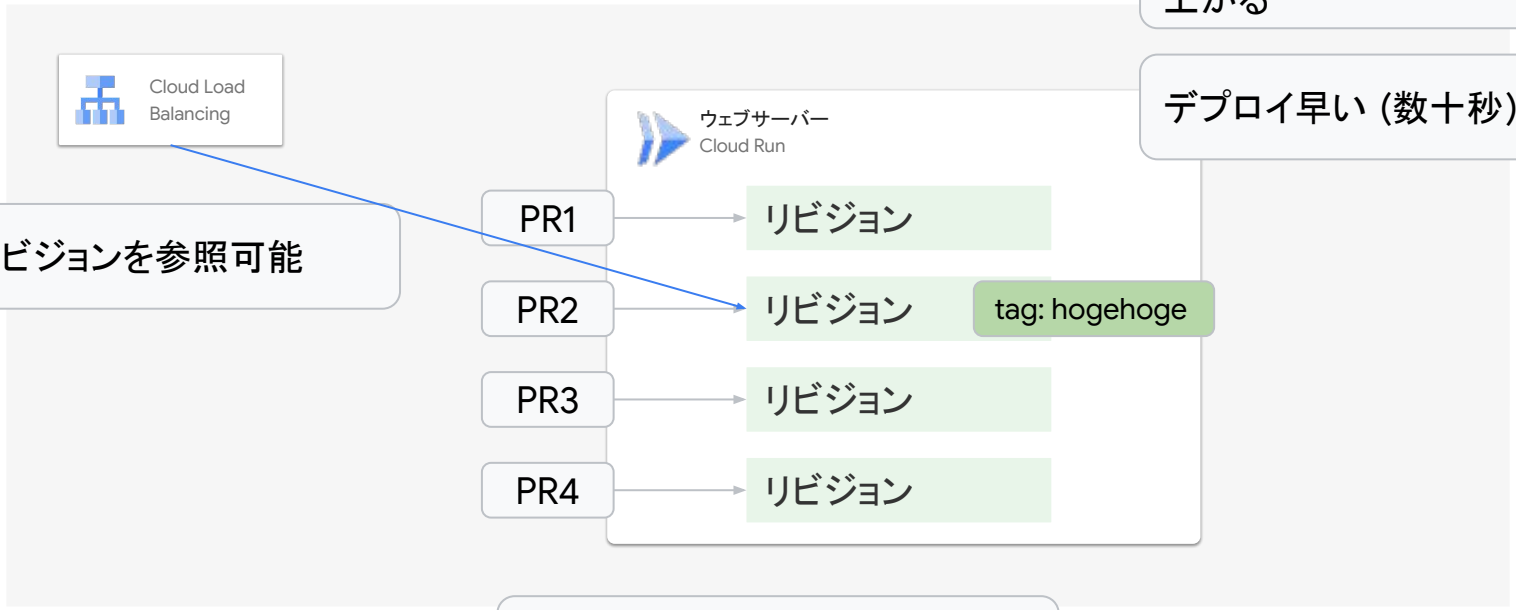
- Google Cloud と Heroku のどちらにもデプロイ可能な状態を保つ必要があり、デプロイの構成ファイルの保守コストが大きくなる
- Cloud Spanner などの Google Cloud 固有のサービスの利用が難しくなる

Google Cloud で同じことができないか？

Google Cloud で PR ごとの検証環境を作るための アーキテクチャ



Cloud Run は PR ごとの検証環境を作るのに都合が良さそう



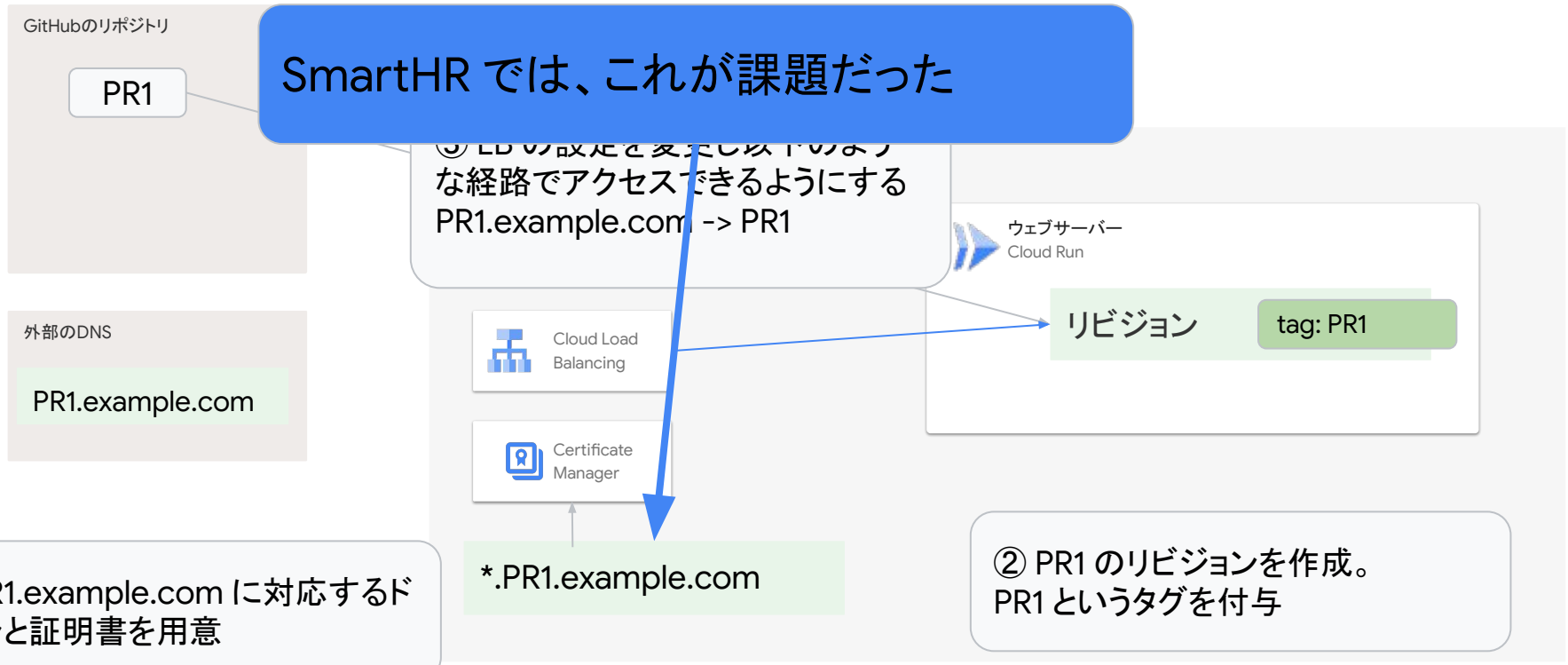
リクエストがあったときだけ立ち上がる

デプロイ早い（数十秒）

特定のリビジョンを参照可能

PR ごとにリビジョンが作れる

どうやって PR ごとの検証環境を作るか



SmartHR の証明書登録の課題

[会社名].smarthr.jp というURL をユーザーに提供

PR1 の検証環境を作る場合、[会社名].PR1.example.com の形でアクセスする必要がある

PR1 の検証環境に対して、*.PR1.example.com の証明書が必要

SmartHR の証明書登録の課題

PR1 に対する検証環境 (PR1.example.com) を作ることを考える場合

一般的な Web アプリの場合

PR1.example.com に対する証明書が必要

事前に *.example.com に対する証明書を1つ取って置けば OK

SmartHR の場合

[会社名].smarthr.jp のような URL をユーザーに提供するので
*.PR1.example.com に対する証明書が必要

PR から環境を作るたびに *.PR1.example.com の証明書を作り有効になるのを待つ必要がある

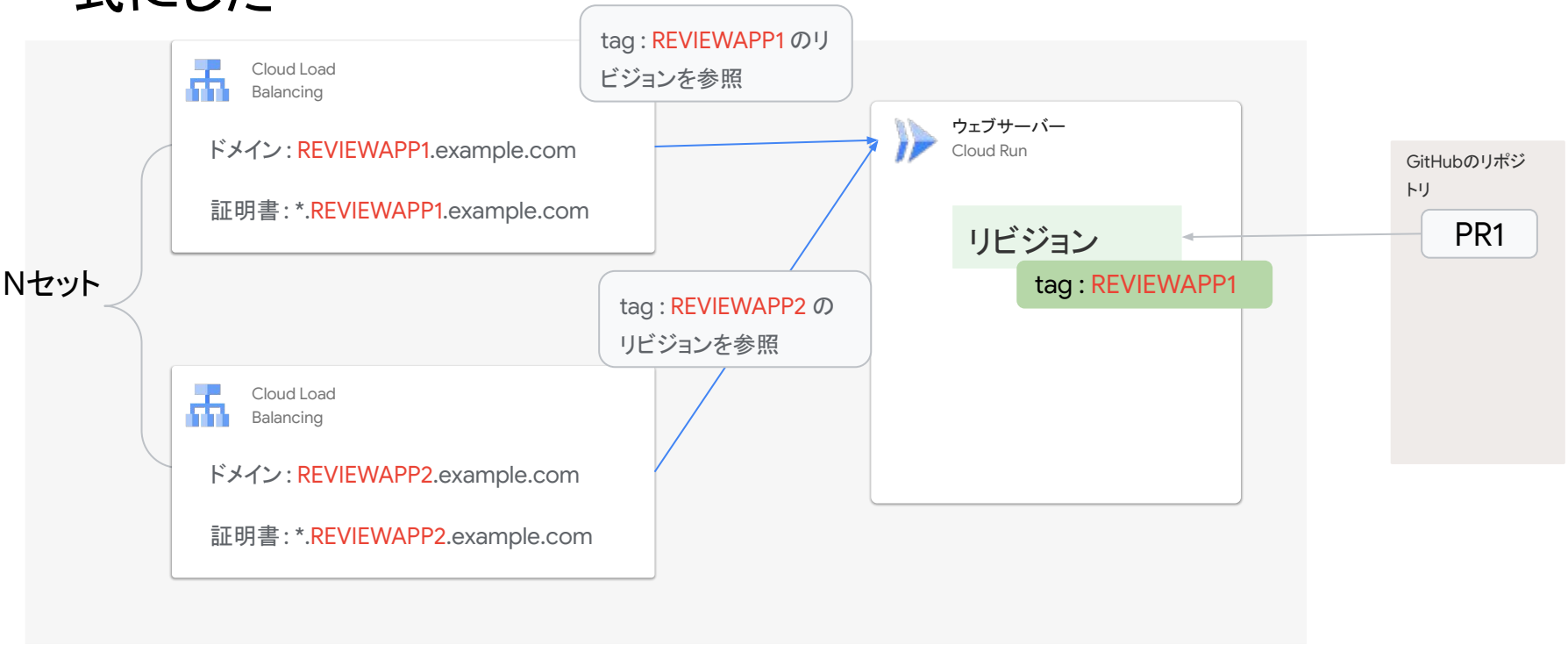
有効になるのを待つのに時間がかかる

SmartHR の証明書登録の課題

PR ごとの環境のため、証明書を用意するのに時間がかかる

最初から、証明書と環境を N セット用意しておこう

N セットの検証環境を事前に用意する方式にした



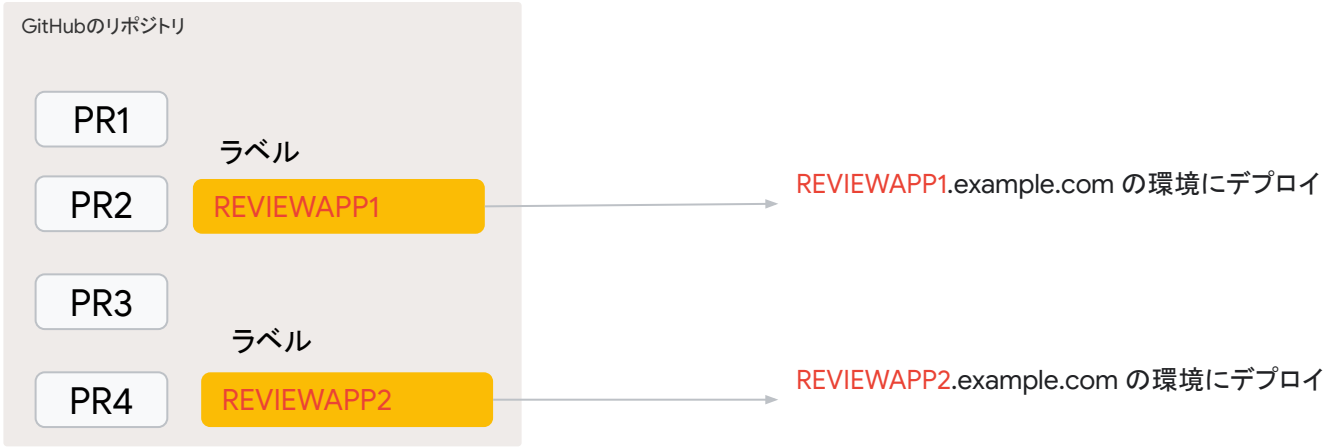
CD に組み込む上での考慮点

01 | ユーザーは、どの環境にデプロイするかを選択する必要がある

02 | N 個の環境のどこがどのプルリクエストに使われているかをユーザーに伝える必要がある

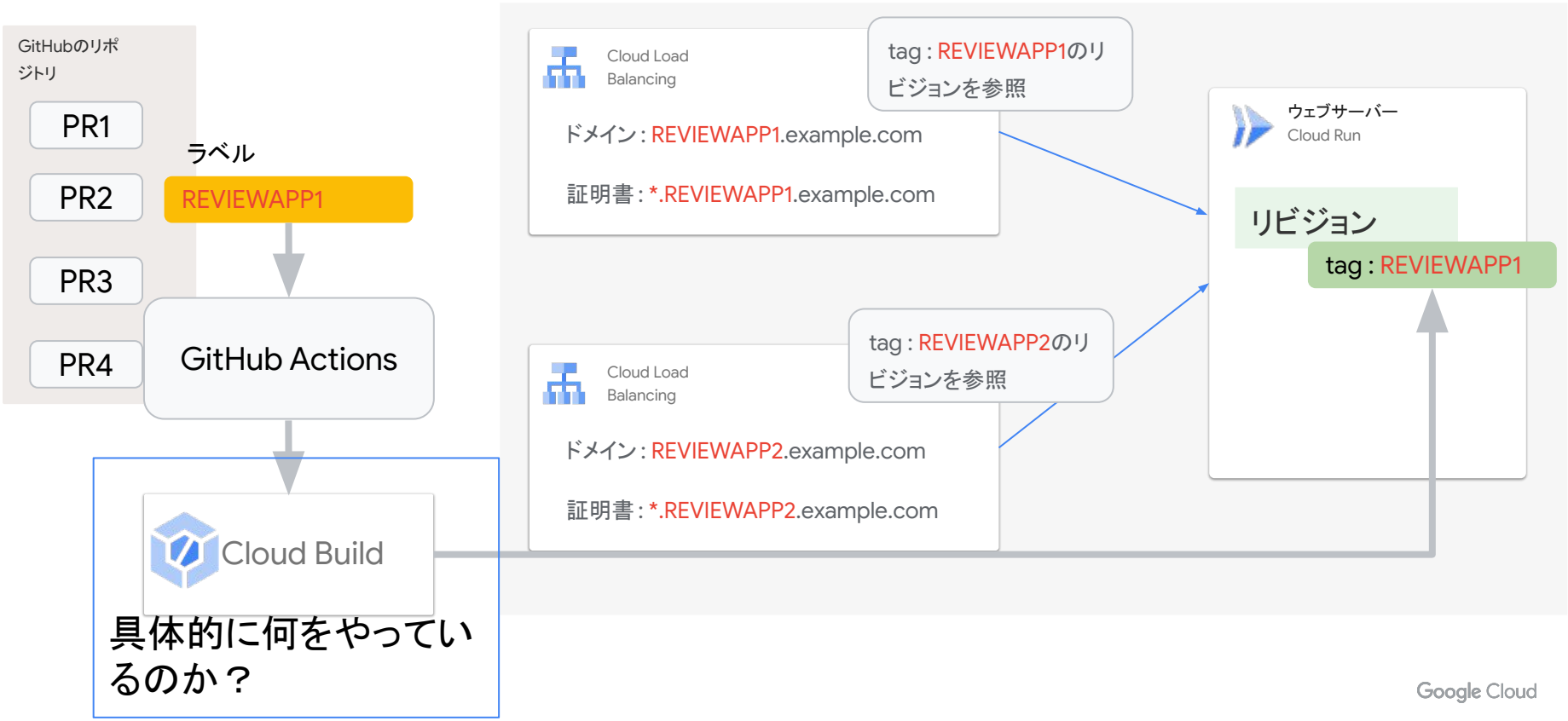
GitHub のラベルを使って CD を作った

GitHub のラベルを使った CD



- ラベルによってデプロイする先を選べる
- N セットのどこをどの PR が使っているのかがわかる

CD のフロー


GitHubのリポジトリ
PR1
PR2
PR3
PR4
ラベル
REVIEWAPP1
GitHub Actions
Cloud Build
具体的に何をやっているのか？
Cloud Load Balancing
ドメイン : REVIEWAPP1.example.com
証明書 : *.REVIEWAPP1.example.com
tag : REVIEWAPP1のリビジョンを参照
Cloud Load Balancing
ドメイン : REVIEWAPP2.example.com
証明書 : *.REVIEWAPP2.example.com
tag : REVIEWAPP2のリビジョンを参照
ウェブサーバー
Cloud Run
リビジョン
tag : REVIEWAPP1

PR 用のイメージの用意

Cloud Run が利用する環境の用意 (例 : Cloud SQL の用意)

Cloud Run のデプロイ

--tag で N セットのうち何番目の環境に配備するかを制御

```yaml
steps:
- id: 'Build and push image'
  name: 'gcr.io/kaniko-project/executor:latest'
  args: # 略
  waitFor: ['-']
- id: 'Ensure Cloud SQL database'
  # Cloud SQLの用意
- id: 'Deploy Cloud Run'
  name: 'gcr.io/google.com/cloudsdktool/cloud-sdk'
  entrypoint: 'bash'
  args:
    - '-c'
    - |-
      gcloud run deploy ${_CLOUD_RUN_SERVICE} \
      --update-env-vars=DATABASE_URL=$${DATABASE_URL} \
      --platform managed \
      --image ${_IMAGE_TAG} \
      --tag ${_REVIEWAPP_LABEL} \
      --no-traffic
  waitFor: ['Ensure Cloud SQL database']
```

実際に運用してみて

少人数のチームのためうまく運用できた

同じ GitHub ラベルを使ってしまう事故や N セットを超えて環境が欲しくなるケースを懸念していたが少人数 (< 7 名) のチームのため、そこまで問題にならなかった

デプロイ時間は運用上問題ないレベル

Cloud Run のデプロイは数十秒程度と高速だったが、イメージのビルドなどのその他の作業に時間がかかっている。PR から環境を作るのにかかる全体の時間は五〜十数分程度。これは運用上問題ないレベルだった。

Cloud Run を使い倒して、PR ごとの検証環境を作った話まとめ

01 | Cloud Run はPR ごとの検証環境を作るのに都合が良い

02 | SmartHR では [会社名].smarthr.jp のような URL が必要で、その場で検証環境を用意するのが難しかった

03 | 事前に検証環境の候補を N セット用意しておき、GitHub のラベルで指定された環境に PR の環境を作るという方法で実現した

06

今後「使い倒し」ていきたい機能

今後「使い倒し」ていきたい機能

- ワーカーを App Engine ではなくCloud Run (--no-cpu-throttling) で作りたい
- Cloud Run の http/1 リクエストサイズの 32 MB 制限の回避のため sidecar 利用して http/2 -> http/1 変換することで対応したい
- 分析処理中心のアプリで AlloyDB を使ってみたい
- 開発で、Cloud Workstations を活用していきたい
- 今回の活用部分で Cloud Build を実用的に使えたので、CD 部分で Cloud Build をもう少し活用したい