Google Cloud

3-shake × FLUX パートナー様と共に成長を目指す Google Cloud 活用事例

生賀 一輝
株式会社スリーシェイク
Sreake 事業部

島田 健人
株式会社 FLUX
開発本部

永友 凜一郎
株式会社 FLUX
開発本部

アジェンダ

証明書発行環境のサーバレス化

データ基盤

01 自己紹介

登壇者

株式会社スリーシェイクSreake事業部

生賀 一輝

株式会社ユーザベースのSRE として従事後、株式会社スリーシェイクに入社。様々な企業様の SRE 支援に関わる。過去にGoogle Cloud Anthos Day や Kubernetes イベント等に登壇。コンテナセキュリティ翻訳（4/12 出版予定）

株式会社FLUX 開発本部

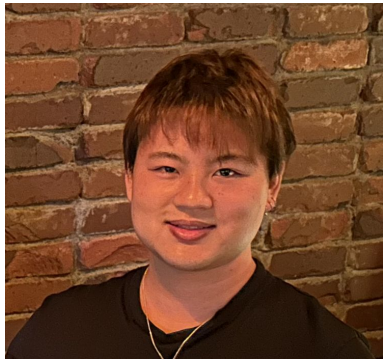
島田 健人

株式会社 Works Human Intelligence にて ERP ソフトウェアの SRE やインフラ周辺ツールのエンジニアを経て、昨年8 月に FLUX に入社。現在は SRE として主にインフラ周辺の開発に従事。

株式会社FLUX 開発本部

永友 凜一郎

大手携帯キャリアにて、データエンジニアやPM を経て、昨年7月に FLUX に入社。現在は、PdM、TL としてデータ基盤の開発に従事。

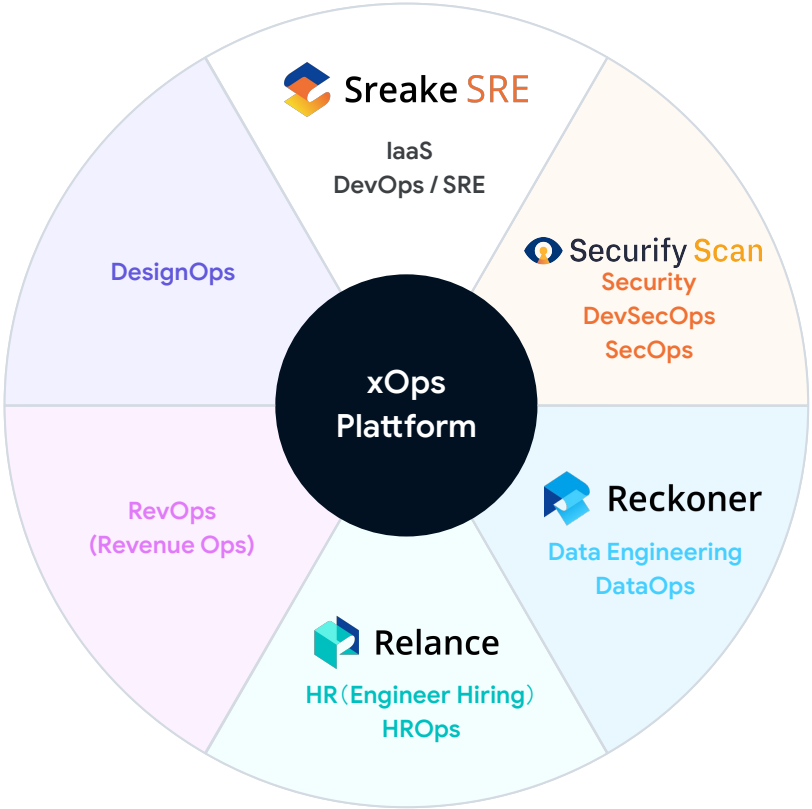


02

会社紹介

About 3-shake

スリーシェイク = xOps 領域のプラットフォーマー



SRE

Sreake SRE

スリーク

SRE 特化支援コンサル

インフラモダナイゼーションの
プロフェッショナル集団

Security

Securify Scan

セキュリファイ

Web 脆弱性診断 SaaS

セキュリティレイヤの
診断内製化を実現

Data Engineering

Reckoner

レコナー

データモダナイゼーション SaaS

クラウド型 ETL
データパイプラインサービス

HR（Engineer Hiring）

Relance

リランス

フリーランスエンジニア エージェント

アプリモダナイゼーションにマッチした
人材に特化してご紹介

Sreake SRE（インフラモダナイゼーション支援）

Sreake SRE

SRE（Site Reliability Engineering）
チームの導入と定着化の伴走型コンサルティングサービス

技術戦略
コンサルティング

システム設計

構築/実装
支援

アセスメント
（パフォーマンス /セキュリティ）

運用支援

- Multi Cloud や Cloud Native なインフラモダナイゼーション及び大規模なサービス運用に強みを持つエンジニアによる技術支援
- SRE の組成/文化定着のための併走型コンサルティング支援

会社紹介(FLUX)

**FLUX は、AI でユーザーを分析して、
各種ツールを最適化するプラットフォームを
開発・提供する会社です。**

企業名　　：株式会社FLUX（FLUX Inc.）
設立　　：2018 年 5 月
従業員数　　：164 名（2023 年 1 月末時点）
代表取締役　：永井元治
所在地　　：東京都渋谷区渋谷3-1-1 PMO 渋谷 II 9F
総調達額　　：12 億円
主要投資家　：DNX Ventures, Archetype Ventures など

AI 活用でオンライン売上向上を実現

タグ設置のみで導入可能なプロダクト群（主要パッケージ）

FLUX ID（クッキーレスID）

↓

自動ユーザーセグメンテーション

↓

ユーザー獲得最適化

広告出稿最適化
（FLUX Targeting）

ウェブサイト最適化

広告収益化
（FLUX Monetize）

ウェブサイト
レイアウト最適化
（FLUX Layout）

分析（開発中）

ウェブサイト分析
/エラー検知
（FLUX Analytics）

ウェブサイト制作

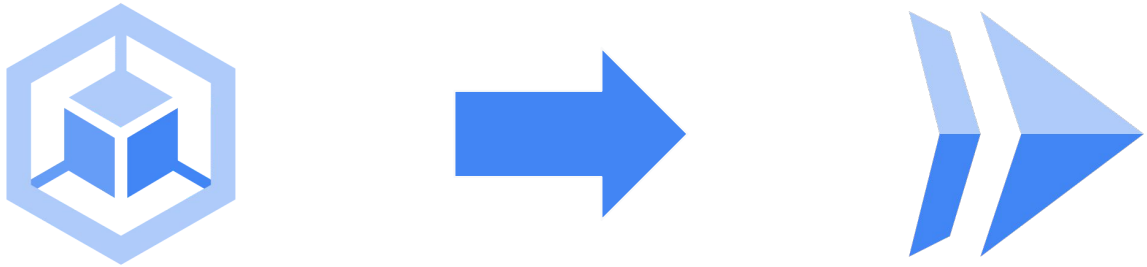
コンテンツ管理
（FLUX siteflow）

証明書発行環境のサーバレス化

03 サーバレス化の背景

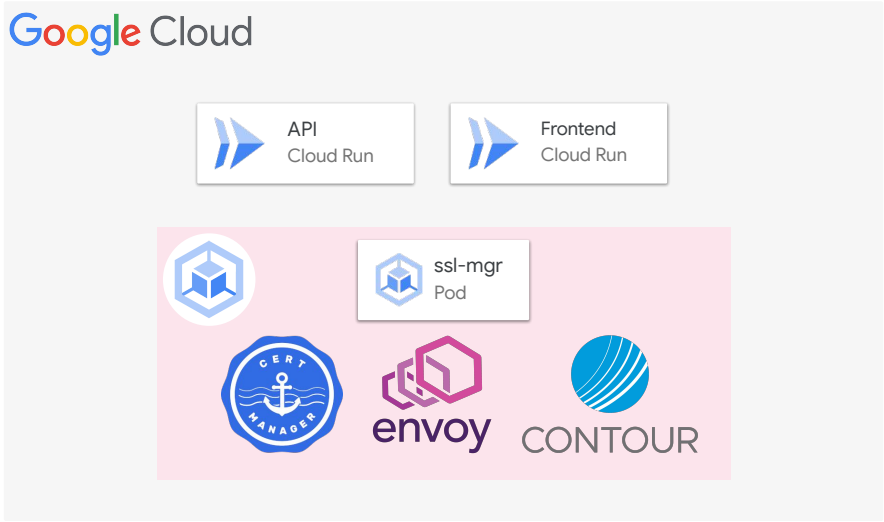
GKE ⇒ Cloud Run 移行に伴う証明書発行機構の再設計

GKE にあった API、Frontend を Cloud Run に移行



GKE ⇒ Cloud Run 移行に伴う証明書発行機構の再設計

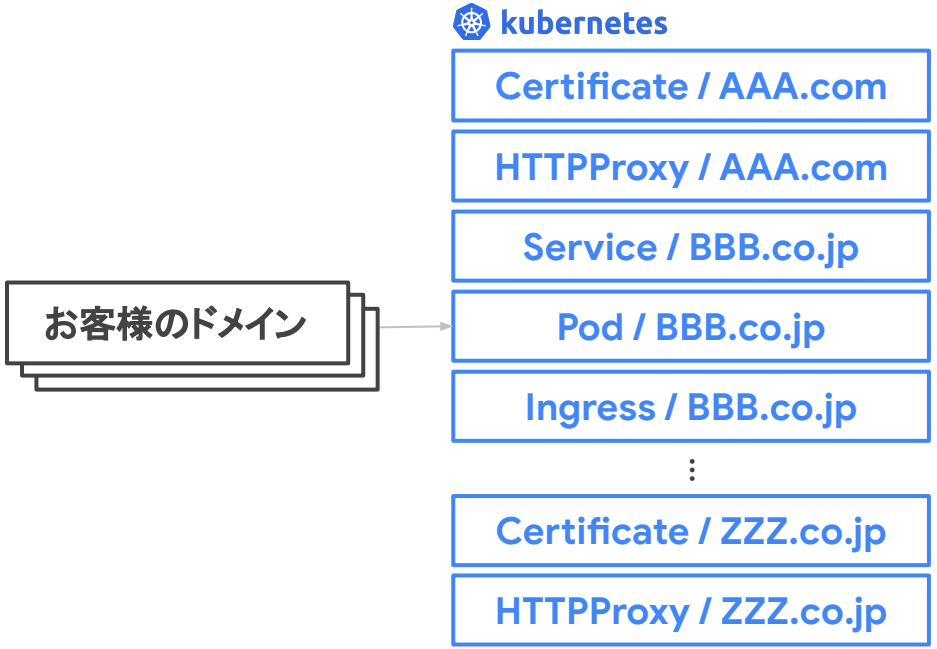
- **現状**
  - 証明書発行バックエンドは OSS である
    Contour や cert-manager などを利用して
    GKE 上に構築されており、
    証明書発行プロセスのみ移行が行われていない
- **問題**
  - 同サービス内のバックエンドが GKE と
    Cloud Run で分かれてしまっている
    ➡ 管理コスト増



GKE ⇒ Cloud Run 移行に伴う証明書発行機構の再設計

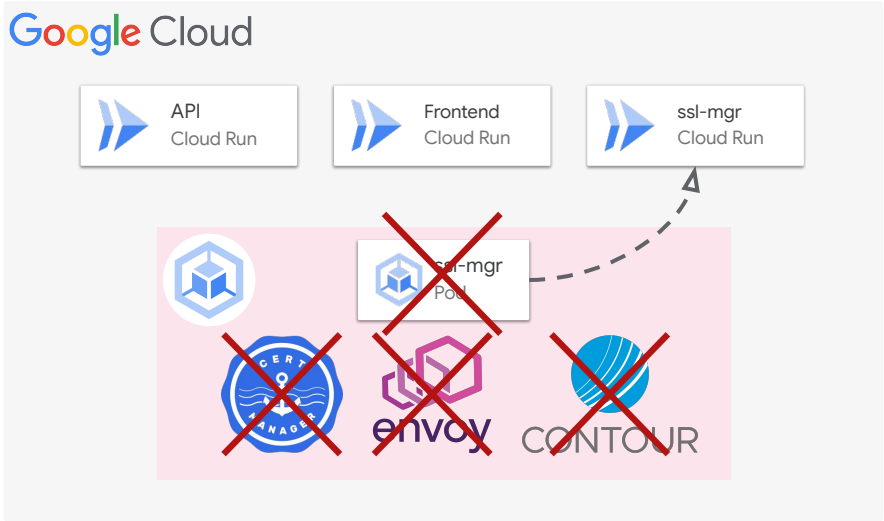
- **問題**
  - Contour や cert-manager のリソースを利用した構成が複雑になっている
    - ドメインごとに HTTPProxy や Certificate、Secret といったリソースが生成される
    - HTTP-01 Challenge が終わらないと、cert-manager から作成される Ingress や Service、Pod が残り続ける

➡ 運用・管理コスト増



GKE ⇒ Cloud Run 移行に伴う証明書発行機構の再設計

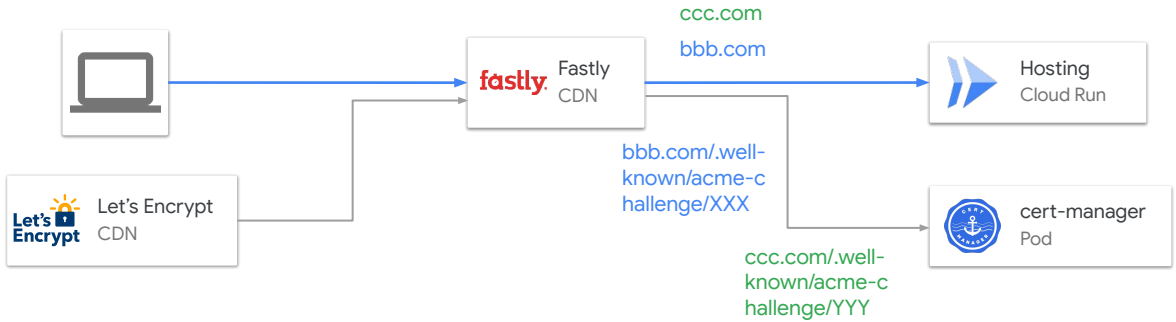
- **目的**
  - GKE と各種エコシステムに依存した証明書発行プロセスからサーバレスで完結させる証明書発行プロセスへ移行



GKE ⇒ Cloud Run 移行によって解決される課題

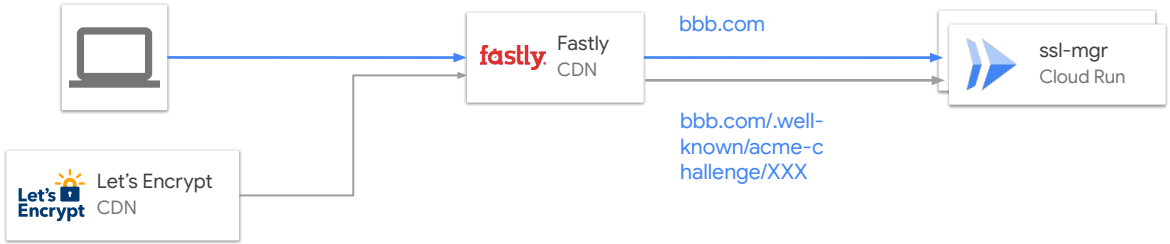
課題 1: 証明書発行とサイトアクセスで、向き先が異なる

⇒ 様々なルーティングが混在する状態になっている　※ ここでは GCLB は省略



GKE ⇒ Cloud Run 移行によって解決される課題

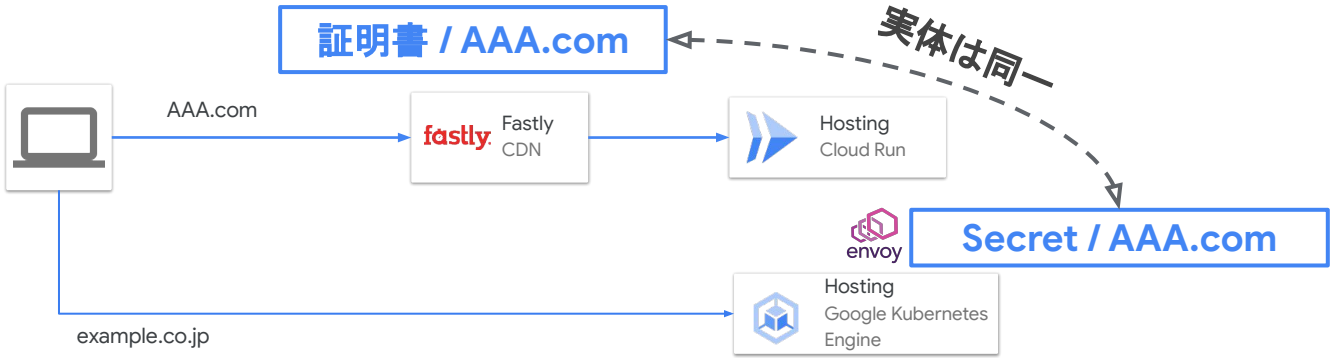
➡ Cloud Run 上に実装することで、
ルーティングをシンプルに



GKE ⇒ Cloud Run 移行によって解決される課題

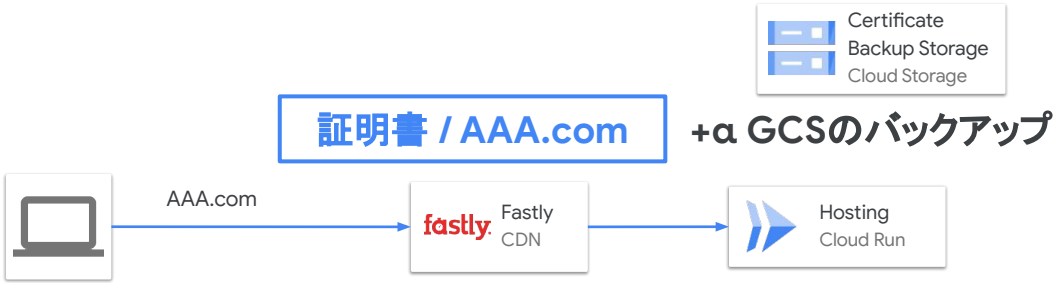
課題 2: GKE と Fastly で証明書が 2 重管理になってしまっている

⇒ Fastly と HTTPProxy が見る Secret で、同一の証明書が存在してしまっている



GKE ⇒ Cloud Run 移行によって解決される課題

➡ Cloud Run 側に発行プロセスを移管することで、
冗長な証明書管理をなくす



04 新証明書発行環境

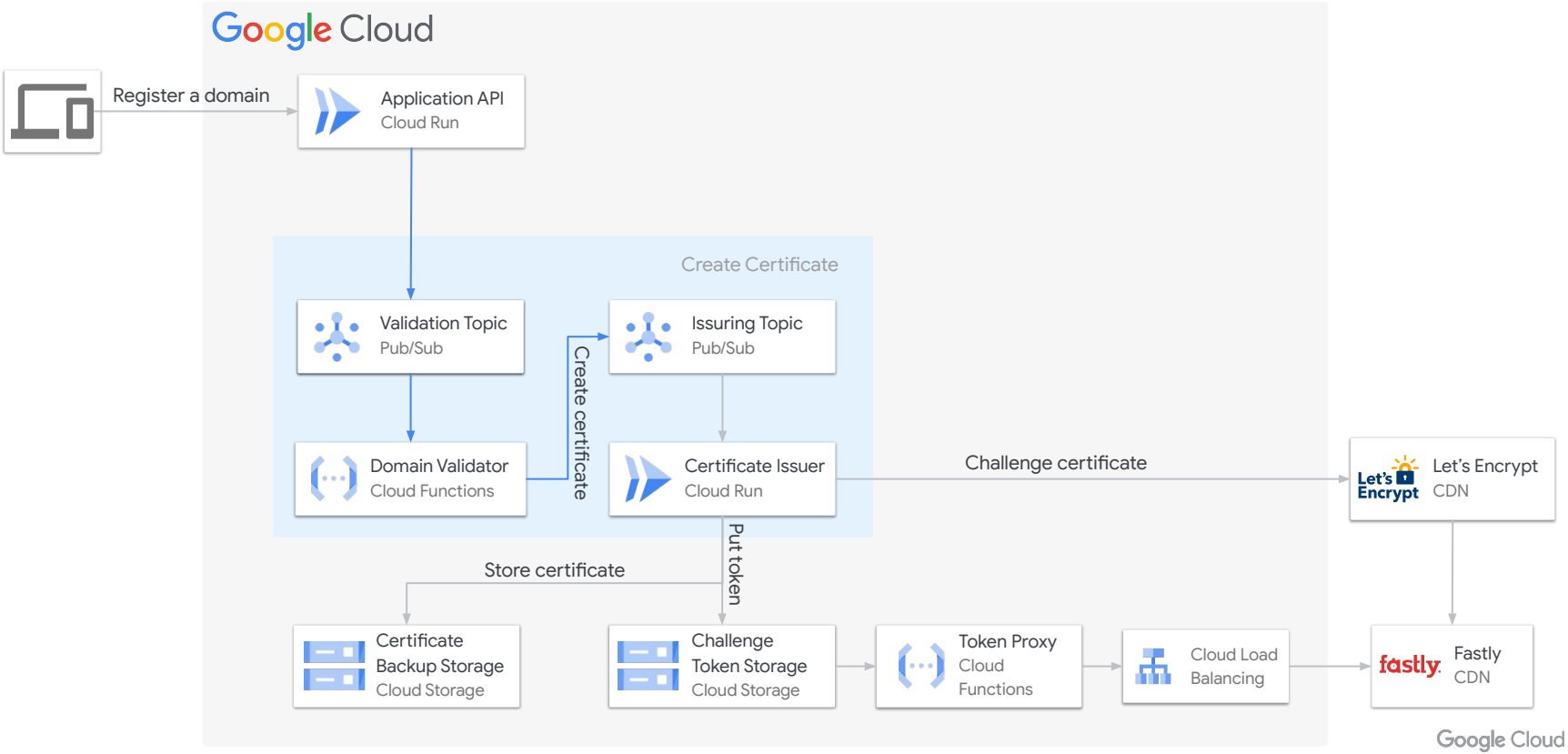
新証明書発行環境

01 **証明書作成プロセス / 証明書発行**

02 証明書作成プロセス / HTTP-01 challenge

03 証明書更新プロセス

証明書発行アーキテクチャ

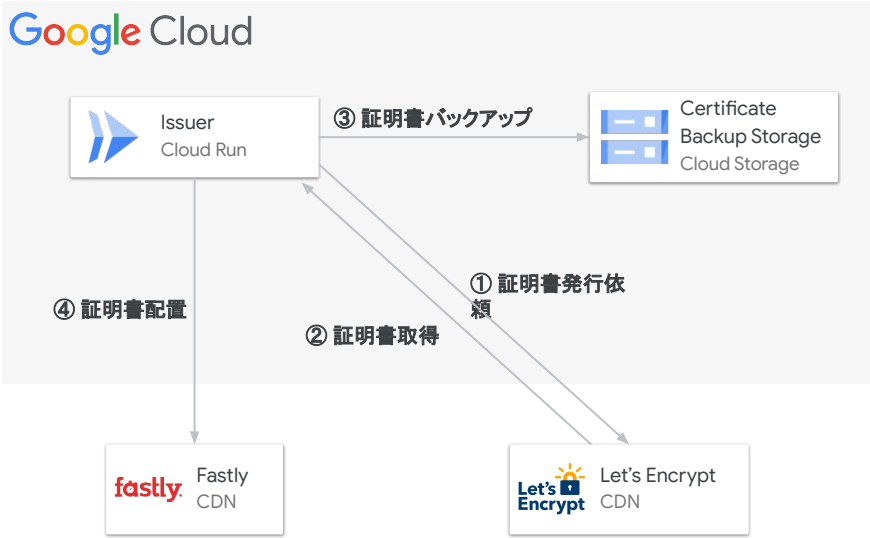


証明書発行プロセス

- **証明書発行**
  - HTTP-01 challenge を実施し、証明書を発行

    ⇒ 詳細は次ページで解説
- **証明書配置**
  - **前提**: Web ページにアクセスするお客様は Fastly を経由して、Cloud Run のリソースに アクセスする
  - 発行された証明書を Fastly に配置
  - 証明書のバックアップは GCS に保管

➡ Issuer コンポーネントが行う



新証明書発行環境

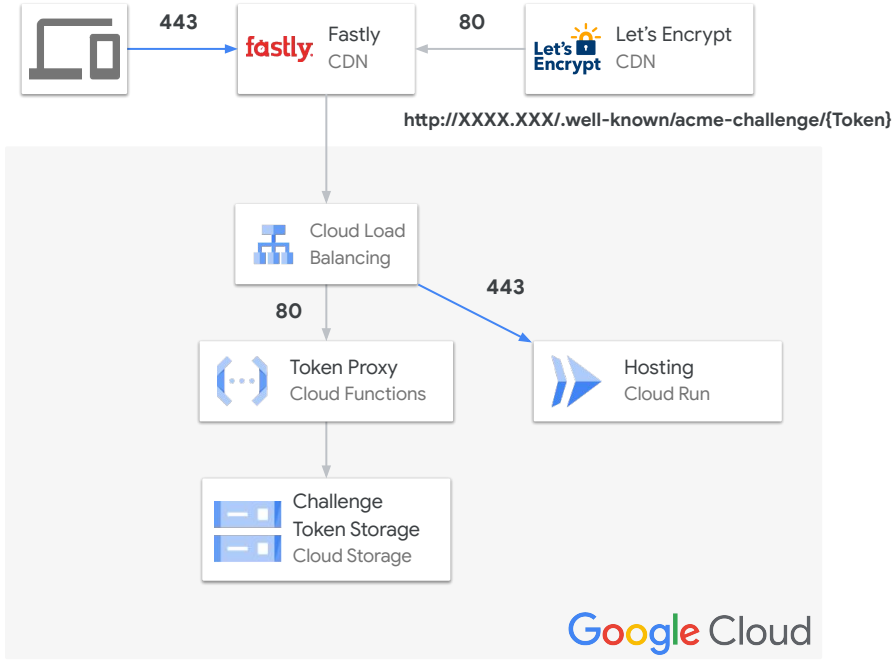
01 証明書作成プロセス / 証明書発行

02 **証明書作成プロセス / HTTP-01 challenge**

03 証明書更新プロセス

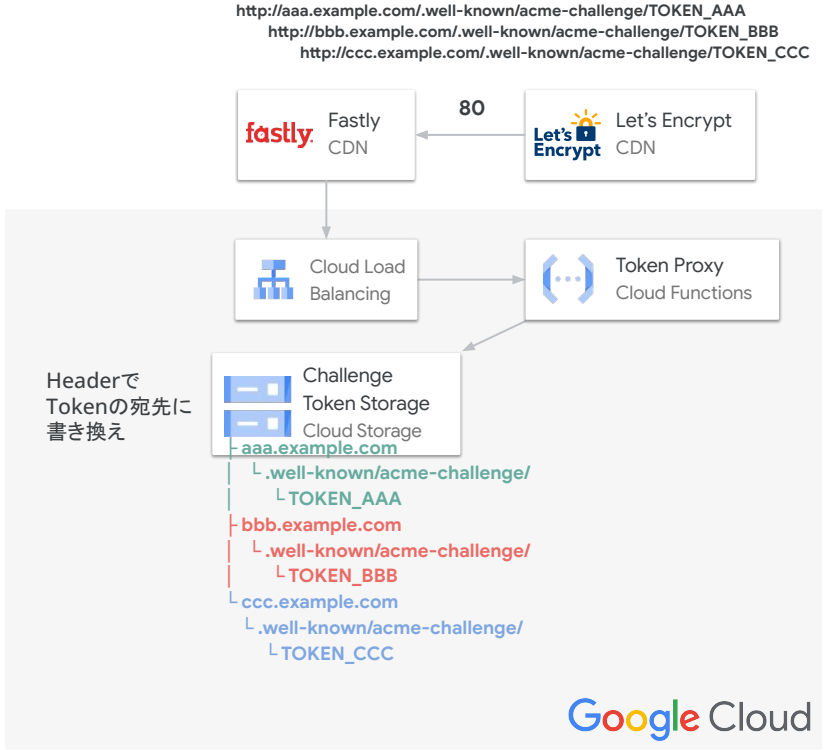
証明書発行プロセス - HTTP-01 challenge

- **ルーティング - Fastly**
  - challenge 用のパスのみ 80 でリッスンし、
    それ以外は 443 で受け付けるようにしている
- **ルーティング - GCLB**
  - **80:** HTTP-01 challenge 用
    ➡ Token Proxy (Cloud Functions) へ
  - **443:** Web ページアクセス用
    ➡ Hosting (Cloud Run) へ



証明書発行プロセス - HTTP-01 challenge

- **HTTP-01 challenge トークンの配置**
  - Issuer がドメインごとに challenge トークンを作成し、GCS に配置する
- **HTTP-01 challenge トークンの取得**
  - Let's Encrypt は Fastly ⇒ GCLB を介して、トークンを取得する
  - Cloud Functions でプロキシを実装し、リクエストされたドメインに基づいて GCS に配置されたトークンを取得する



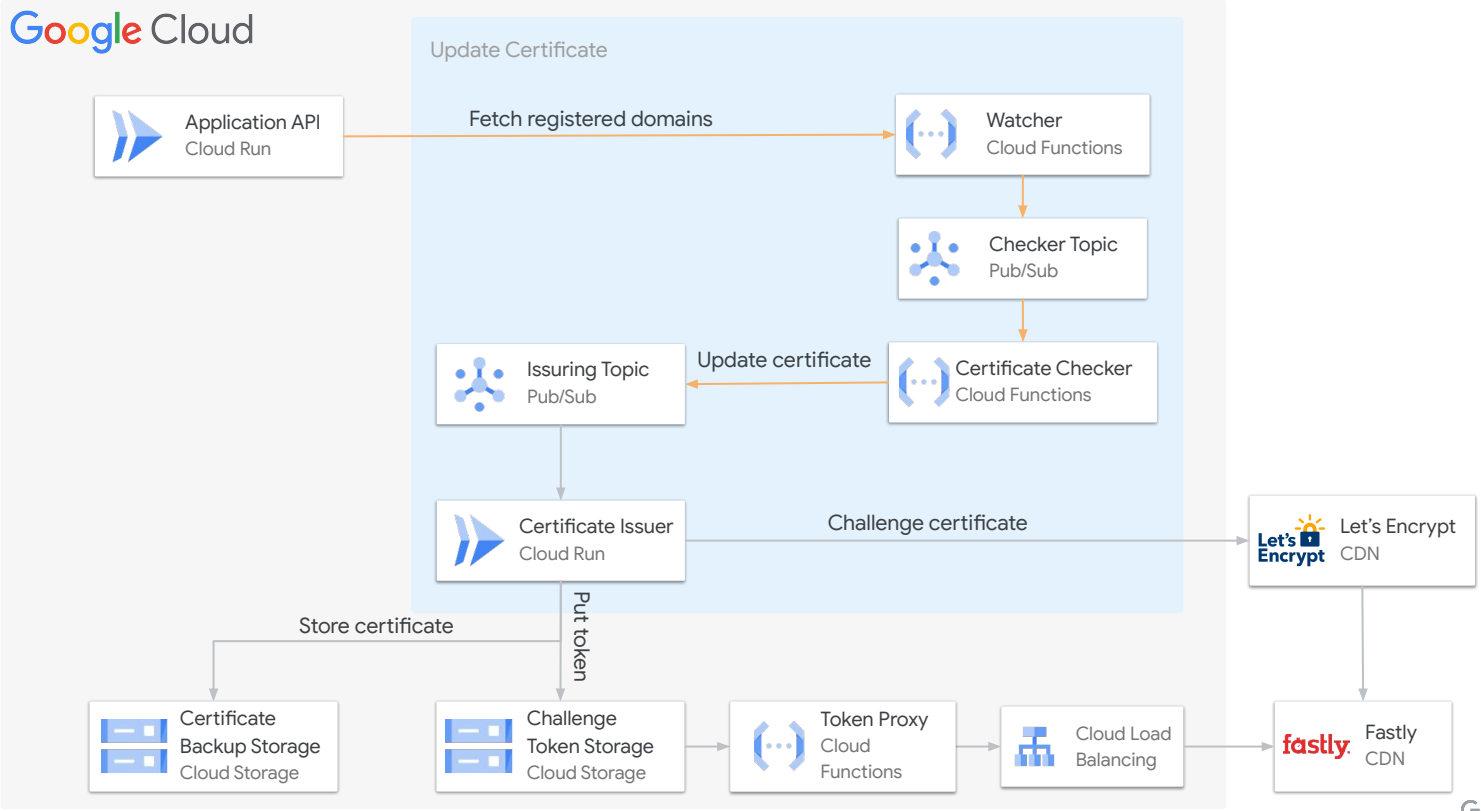
新証明書発行環境

01 証明書作成プロセス / 証明書発行

02 証明書作成プロセス / HTTP-01 challenge

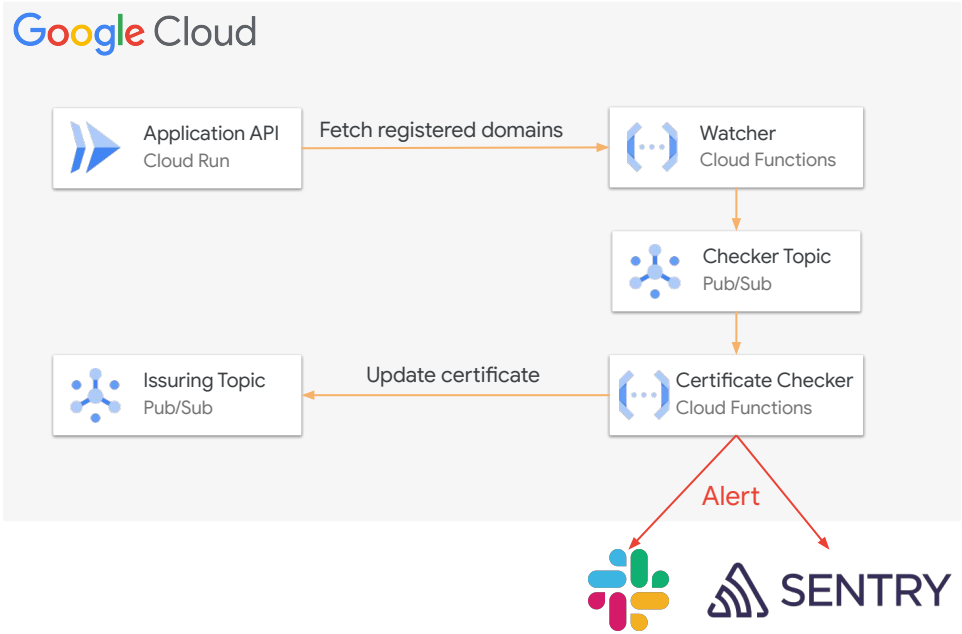
03 **証明書更新プロセス**

証明書更新アーキテクチャ

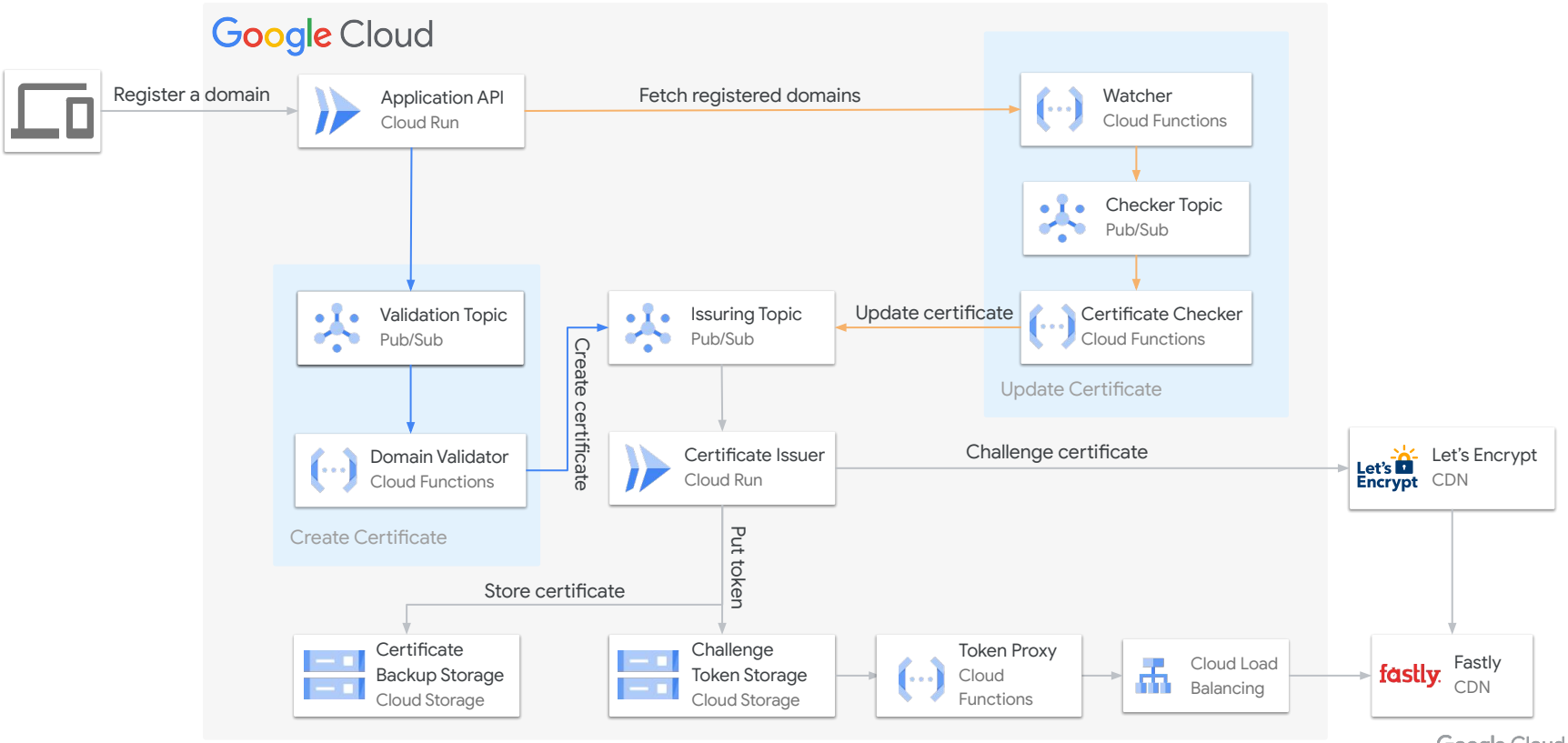


証明書更新プロセス

- **Watcher コンポーネント**
  - API からドメインを一括で取得
  - 取得したドメインを Pub/Sub に流す
- **Checker コンポーネント**
  - 与えられたドメインについて以下を検証
    - 証明書の期限
    - A レコード
    - HTTP ステータス コード
  - **証明書の期限切れが迫っている場合**:
    更新リクエストを Issuer に流す
  - **その他**: Slack / Sentry へアラートを流す



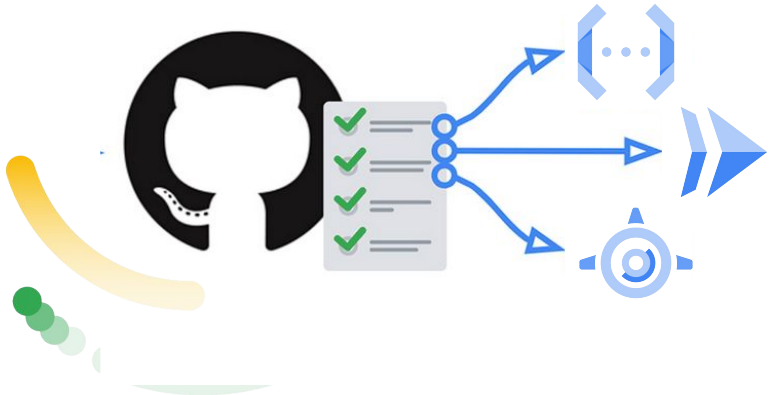
証明書管理アーキテクチャ全体



05 CI/CD

GitHub Actions for Google Cloud

- **Google Cloud 向けのカスタム アクションが提供されている**
  - google-github-actions/auth
    → Google Cloud への認証・認可
  - google-github-actions/setup-gcloud
    → ワークフロー内で gcloud CLI を使用可能にする
  - google-github-actions/deploy-cloud-functions
    → Cloud Functions のデプロイ
  - google-github-actions/deploy-cloudrun
    → Cloud Run のデプロイ
  - etc...



Workload Identity 連携

google-github-actions/auth

- GitHub Actions のジョブが Google Cloud の
  リソースにアクセスする際に GitHub が発行した
  OIDC トークンを用いて認可を行う
- サービス アカウント キーを保管する必要が
  なくなり、セキュリティを向上させることが可能
- OIDC トークンのプロパティに基づいて
  認証する対象を制限することが可能

➡ アカウントキーを保存している場合
こちらに移行を推奨

```
...(省略)
      - name: Authenticate to Google Cloud
        id: google-cloud-auth
        uses: google-github-actions/auth@v1
        with:
          create_credentials_file: true
          token_format: "access_token"
          workload_identity_provider: "/path/to/provider"
          service_account: "gha-XXXXX@XXXXX.iam.gserviceaccount.com"
...(省略)
```

Cloud Functions / Cloud Run デプロイ

- **google-github-actions/deploy-cloud-functions**
- **google-github-actions/deploy-cloud-run**
  - GitHub Actions ワークフローで Cloud Functions / Cloud Run のデプロイが行える
  - ディレクトリごとにデプロイが可能
  - Git で管理されているファイルを基に環境変数の指定が可能
  - デプロイ結果としての エンドポイント URL はアウトプットとして 後続のステップで使用可能

```
...(省略)
      - name: Deploy Cloud Functions
        id: deploy
        uses: "google-github-actions/deploy-cloud-functions@v1"
        with:
          name: "XXX-functions-dev"
          entry_point: "Tester"
          runtime: "go119"
          region: "asia-northeast1"
          source_dir: "tester"
          ingress_setting: "internal-and-gclb"
          env_vars_file: "./tester/env.dev.yaml"
          secret_environment_variables: 'TEST_SECRET=test1,TEST_SECRET2=test2'
          service_account_email: "tester@XXX.iam.gserviceaccount.com"
...(省略)
```

deploy-cloud-functions の例

06

3-shake × FLUX
取り組みとパートナーシップ

3-shake × FLUX のパートナーシップ

豊富な知見・スキルによる支援

3-shake は Google Cloud パートナーであり、Kubernetes や
監視システムの構築、運用支援等の
SRE サービスを提供している。

3-shake の知見・スキルによる技術支援を受けられていることで、
インフラ面を中心にプロダクトの
成長がスムーズに行えている。

ドメイン理解による迅速な開発

技術的部分だけでなく、プロダクトのドメイン領域に関しても深く理解いただいている。

これによって、FLUX のメンバーとも対等にディスカッションが行えており、開発や検証の迅速化に繋がっている。

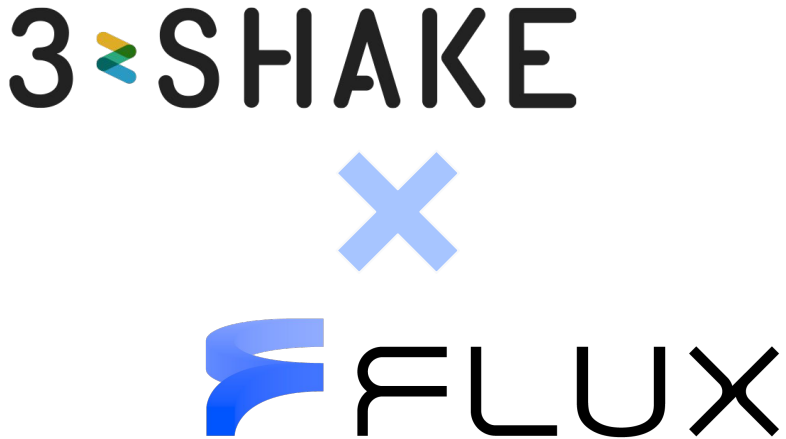
双方向コミュニケーション

「FLUX → 3-shake でのタスク依頼」といったような一方通行の
コミュニケーションになりがちが、
3-shake さんの方から能動的に
アクションを起こしてくださることも多く、
単なる”技術支援請負会社”を超えた働きをしてくれている。

3-shake × FLUX の取り組み

- **具体例**
  - プロダクト インフラの設計・コード化
  - CircleCI, GitHub Actions 等の CI/CD の整備
  - プロダクトやインフラのアラート原因調査
  - PoC に向けた技術調査
  - 言語ごとの CI パイプラインの最適化

➡ インフラの設計・構築だけでなく、
インフラ バックエンドの設計・実装も含め、
トータルに取り組んでいただいている



データ基盤の話

07

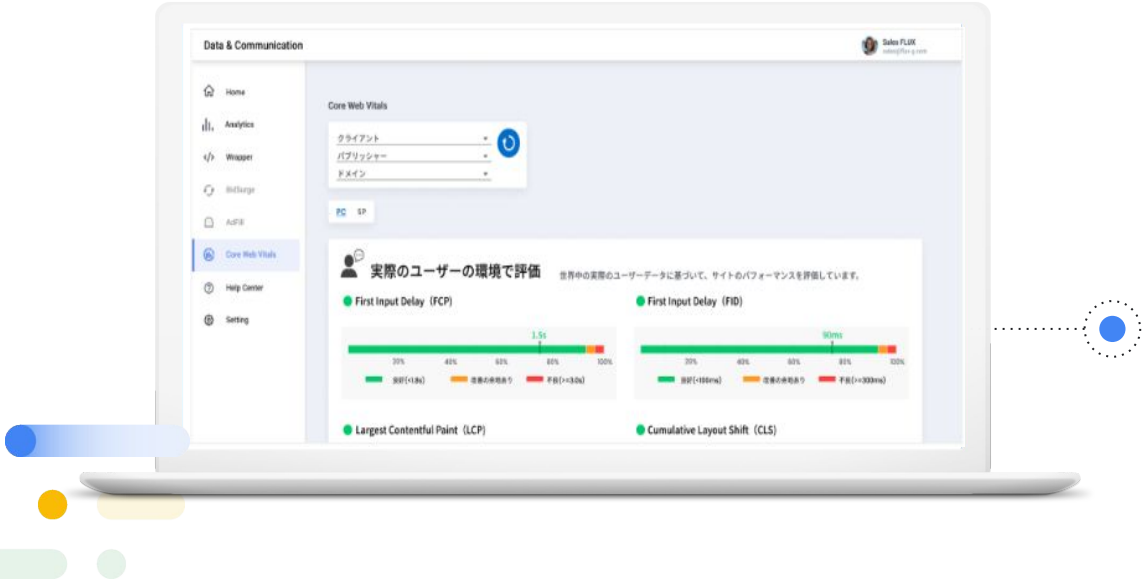
データ基盤事業の説明

FLUX AutoStream

オンライン売上最大化サービス

独自のユーザー id ソリューションベースに、
一つのタグを設置するだけで
ユーザーセグメントを自動で作成し、
広告収益最大化・サブスクリプション /
app ダウンロード最大化・EC コンバージョン
最適化などを実現するサービスです。

大手出版社やテレビ局のウェブサービスを
中心に **1,000 社以上**に導入されています。



データ基盤事業の説明

FLUX の広告ビジネスを支えているデータソース

Bidder データ

閲覧データ

08

データ基盤アーキテクチャ

データ基盤環境

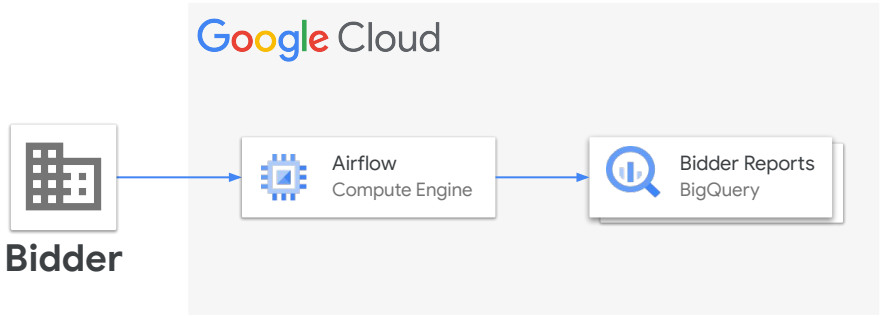
01 **Bidder Report 取得基盤**

02 Web 閲覧データ取得基盤

03 ELT 基盤

旧アーキテクチャ

- **構成**
  - Raw Data を**保存しない**
  - **Pandas** で加工
  - **固定 IP の GCE** で internet へ
  - **特大インスタンス**の Compute Engine (n2-highmem-32)



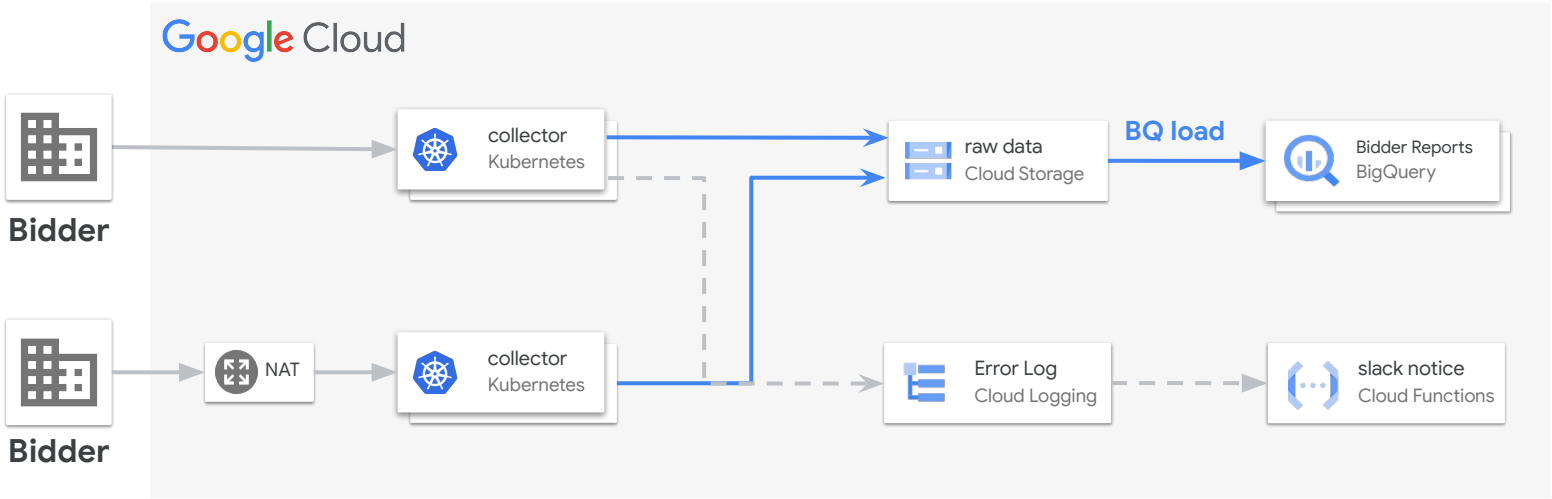
アーキテクチャの変更点

- Raw Data を**保存しない**
- **Pandas** で加工
- **固定 IP の GCE** で internet へ
- **特大インスタンス**の Compute Engine

- **証跡を追いやすく**
- **簡単に transform**
- **コスト最適化**

- Raw Data を**保存**
- **SQL** で加工
- 要求によって **NAT** 経由で internet へ
- **GKE Autopilot** による最適なスケーリング

新アーキテクチャ


Google Cloud
Bidder
Bidder
collector
Kubernetes
NAT
collector
Kubernetes
raw data
Cloud Storage
BQ load
Bidder Reports
BigQuery
Error Log
Cloud Logging
slack notice
Cloud Functions

データ基盤環境

01 Bidder Report 取得基盤

02 **Web 閲覧データ取得基盤**

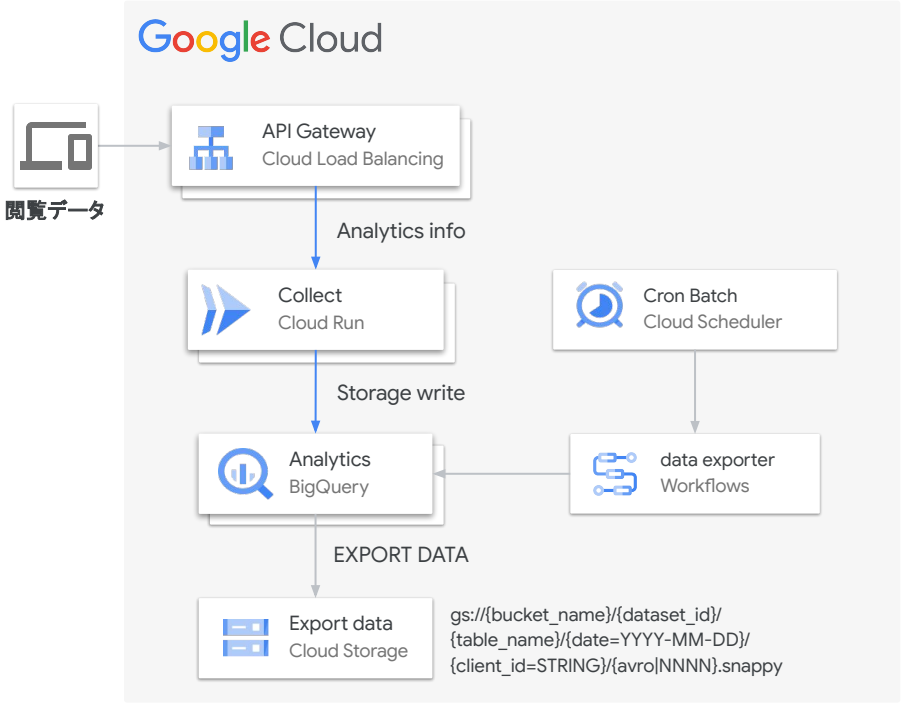
03 ELT 基盤

Web 閲覧データの分析要件

- **要件**
  - 30 TB/day のデータ量
  - 翌営業日までに当日分のデータを BI ツール に表示させたい
  - 生データの利用は現状想定していない
  - コストは低ければ低いほど良い

Web 閲覧データ取得基盤の説明

- **改善点**
  - BQ の storage write API を使用
    - バッファをなるべく短く
  - BQ の外部データを使用
    - Strage/クエリコストを ⅕ 程度に削減
    - Partition 4000 問題が解消



データ基盤環境

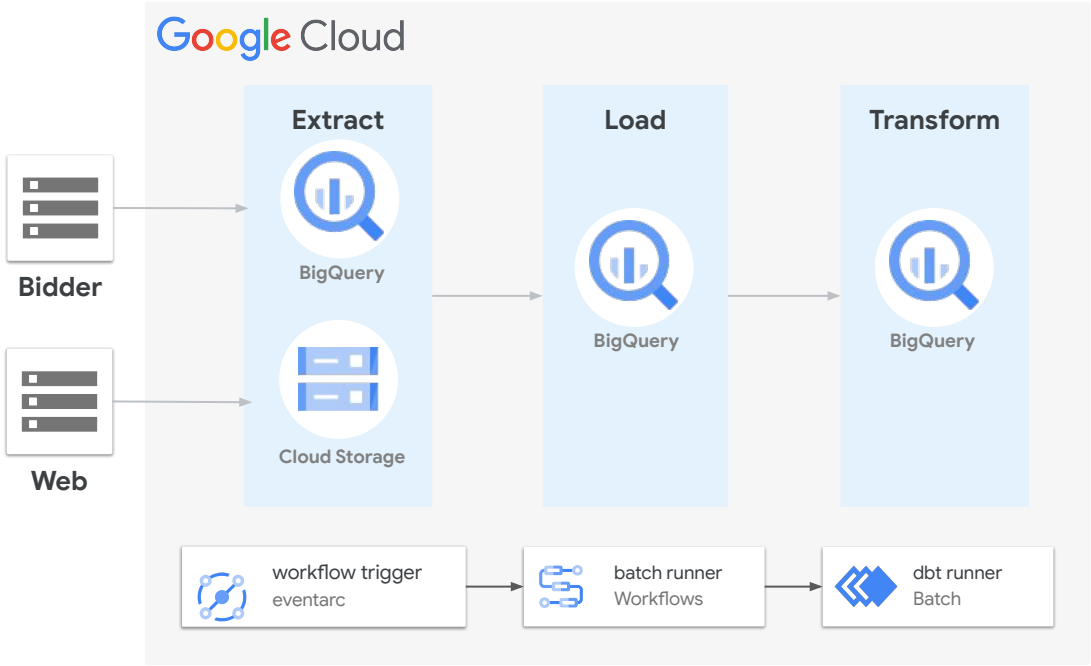
01 Bidder Report 取得基盤

02 Web 閲覧データ取得基盤

03 **ELT 基盤**

ELT ツール

- **従来（ETL ツール）**
  - Scheduled query、Python を使用
  - 手動デプロイ
  - 見えない依存関係
- **新 ELT ツール**
  - dbt™ を使用し、インフラを Workflows と Batch に
  - GitHub Actions によるデプロイ
  - Lineage Graph による可視化



現在のアーキテクチャのメリット

- dbt™ の導入
  - ➡ SQL が理解できれば誰でも、**ELT が実行可能**
- フロントで処理していたものをデータ層に移す
  - ➡ データを扱いやすくなり **開発が容易に**
- GitHub Actions による CI/CD
  - ➡ リリース時の**ヒューマンエラー防止**
- Lineage Graph の導入
  - ➡ 依存関係が可視化され **リファクタリングが簡単に**
- Workflows と Batch の利用
  - ➡ **インフラ部分を Google Cloud に任せられる**

09

Google Cloud のメリット活用事例

メリット活用事例

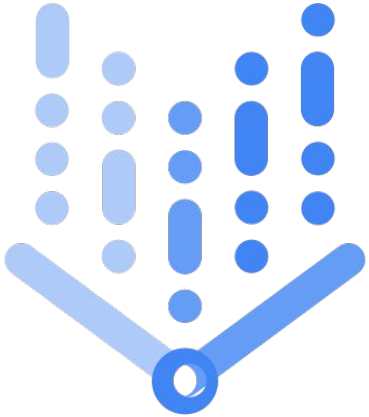
- GKE Autopilot
  - **コスト最適化**
    - 不必要に立ち上がっていたインスタンス料金の削減
  - **スケーリング**
    - ノードの管理が不要
- BigQuery
  - **コスト最適化**
    - ノードを意識せずに集計関数を扱える
  - **外部テーブル**
    - データを読み込ませる手順が不要

10

今後の展望

今後の展望

- Vertex AI を用いて現在手動で行なっている
  ML モデルのデプロイ自動化
- 現在バッチで行なっている分析をリアルタイム分析に
  変更し、PDCA サイクル高速化への手助け
- データの民主化



Thank you.