日本最大級の医療従事者専用サイト m3.com を運営する

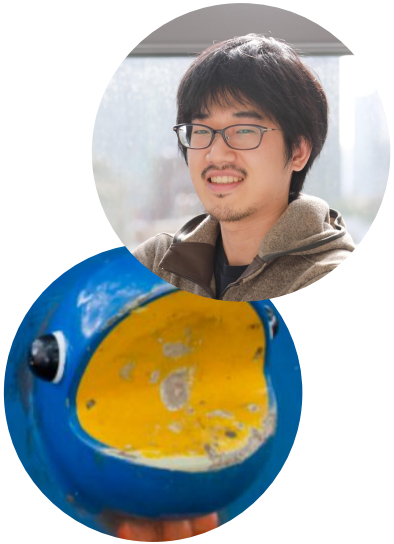
エムスリーのアンケートシステムのアーキテクチャーをご紹介

～マイクロサービスの分割とアンケートデータのやりとり～

滝安 純平

エムスリー株式会社
ソフトウェアエンジニア TL / BIRカンパニー執行役員

自己紹介



滝安 純平（@juntaki）

バックエンドと Web フロントエンドエンジニア、兼プロダクトマネージャみたいなことをやっています

もともと組み込み Linux のカーネル開発をしていました。最近は Flutter でなにか作っています。
好きな言語は Go ʕ◔ϖ◔ʔ

アジェンダ

1. エムスリーと BIR のご紹介
2. アンケートシステムのアーキテクチャー
3. Go と Google Cloud の活用

エムスリーと BIR のご紹介

01

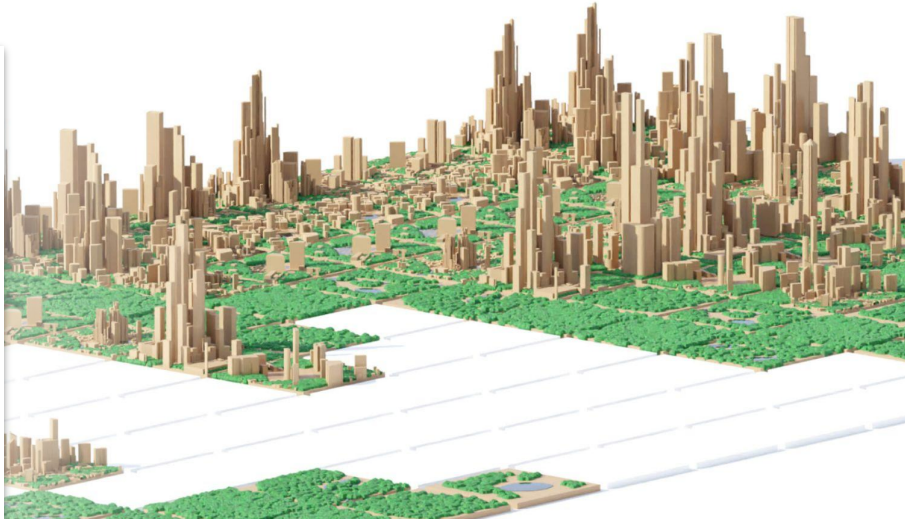
Google Cloud

エムスリーのビジョン

インターネットを活用し

健康で楽しく長生きする人を１人でも増やし、

不必要な医療コストを１円でも減らすこと



BIR - ビジネスインテリジェンス&リサーチ

日本最大級の医療従事者専用サイトm3.com に登録する、医師、薬剤師、看護師などの医療従事者を対象としたアンケートをベースに、クライアントへマーケティング支援などを提供する事業を行っています

アンケート一覧の Web ページ & アプリの回答画面

アンケートビジネスの流れ

1. **アンケートを集める**

2. **データを活用する**

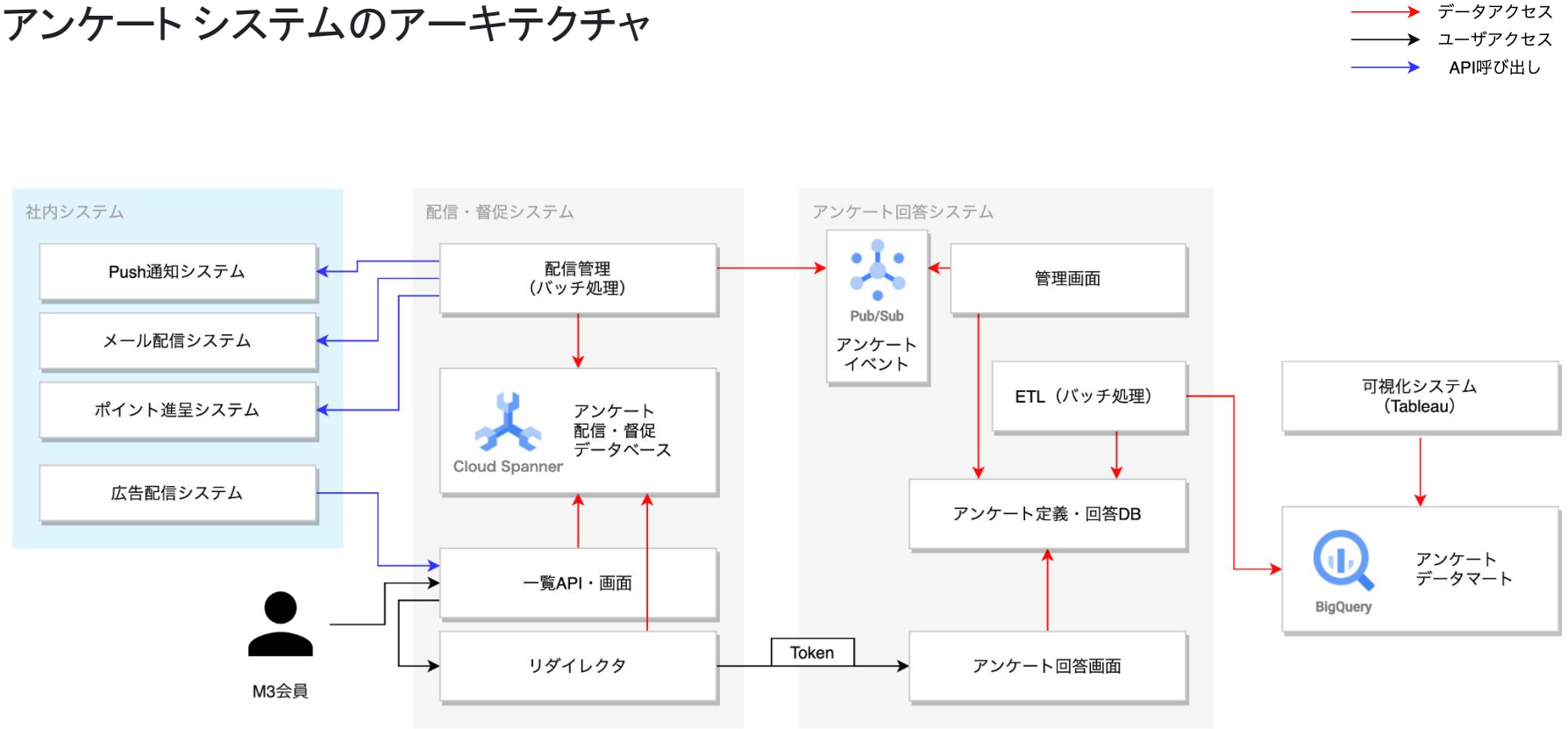
1. アンケートを作る
2. 配信・督促をがんばる
3. データを整理する
4. データを可視化する

システムのアーキテクチャーとデータのやりとり

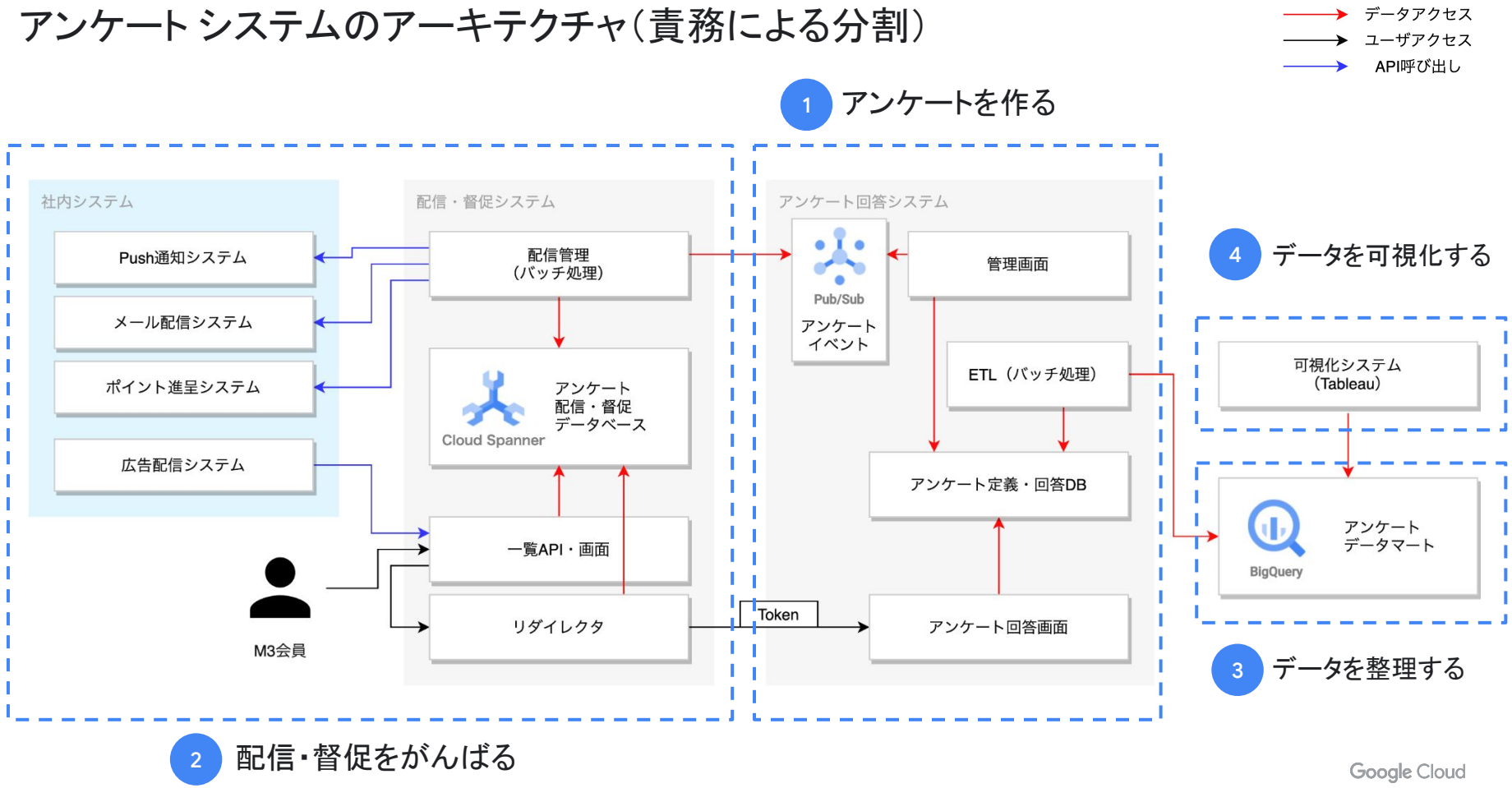
02

Google Cloud

アンケート システムのアーキテクチャ



アンケート システムのアーキテクチャ（責務による分割）



アンケート システムで扱うデータ

配信・督促データ

広告表示・ユーザによる一覧表示など、
参照の頻度が高い

- 配信対象ユーザー一覧
- ユーザーの回答状況
- アンケートの配信状況

アンケート定義

アンケート表示時に参照されるのみで、
配信後は ReadOnly

- 設問の定義
- レイアウト情報

回答データ

アンケート回答時に INSERT される
納品時に集計の対象になる

- ユーザー ID
- 回答内容

設計のポイント

マイクロサービスを責務によって分割する

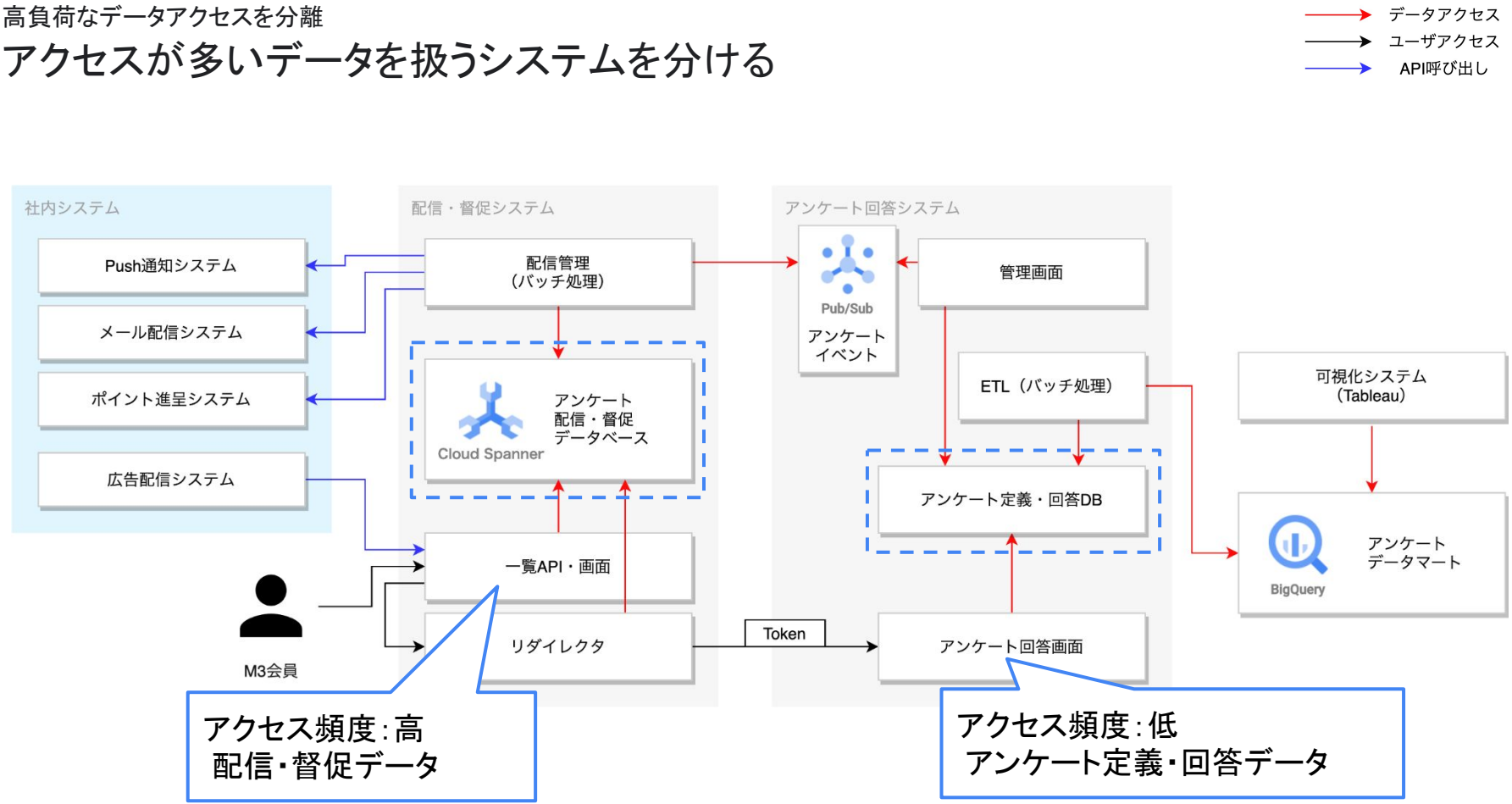
→ **高負荷なデータアクセスの分離**が自然に実現できている

さらに、依存関係を整理する工夫により

→ **アンケート回答システムの独立性**を高め、保守性を向上している

高負荷なデータアクセスを分離

アクセスが多いデータを扱うシステムを分ける



高負荷なデータアクセスを分離

可視化・集計クエリを分ける

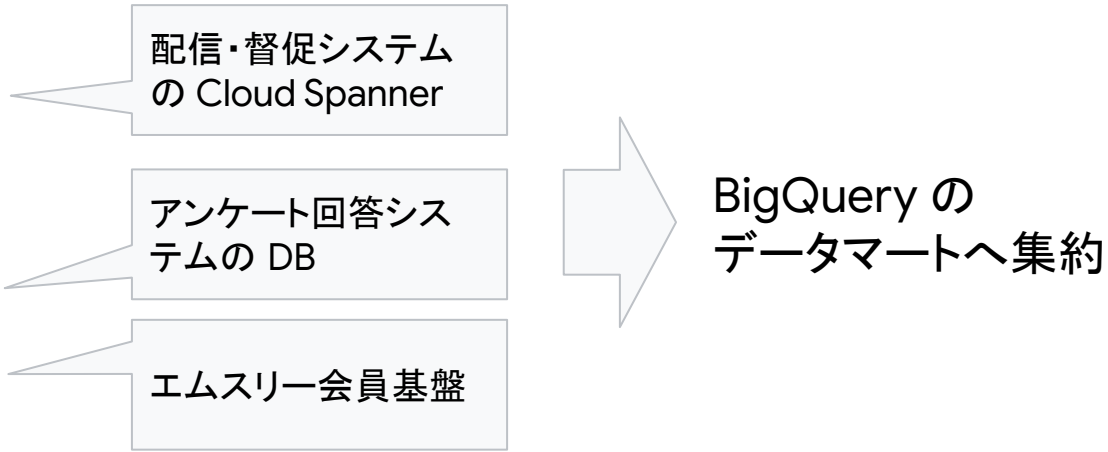
可視化・集計の目的に応じて、データを集約する必要がある
直接参照すると、集計クエリの負荷をそれぞれのシステムで考慮する必要がある

社内用分析

- 回答状況
- 流入元チャネル

納品物

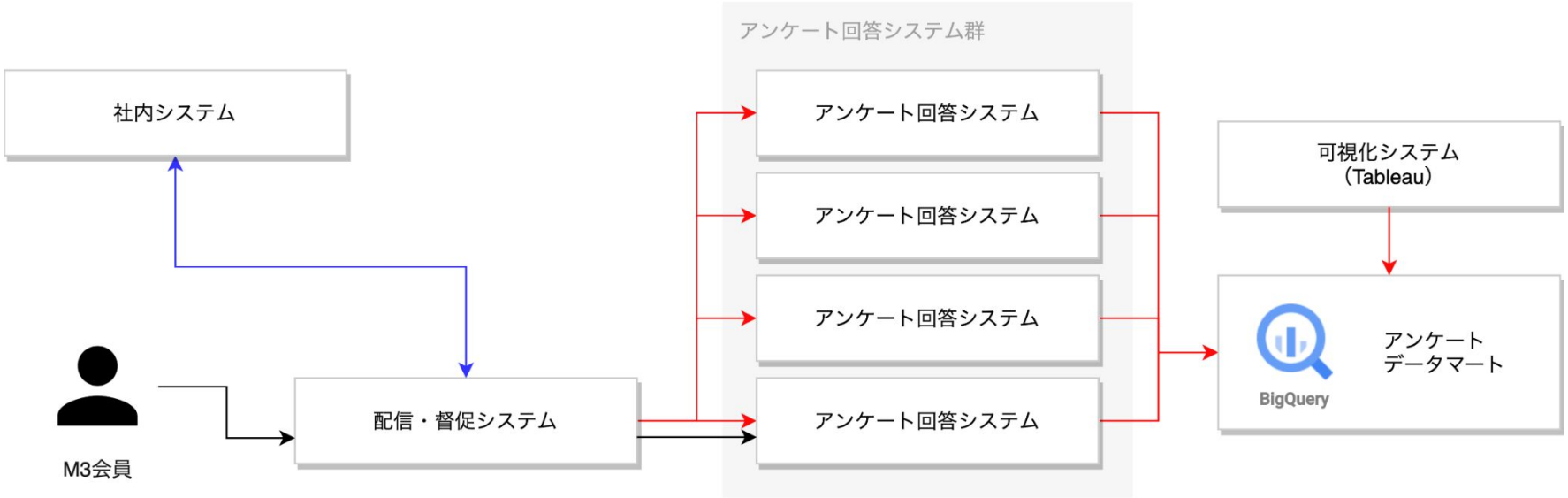
- 回答＆設問
- 会員属性



アンケート回答システムの独立性

アンケート回答システムに着目する理由

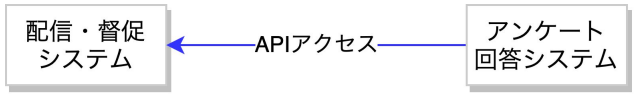
- アンケートの実施形態ごとに、別仕様のアンケート回答システムが存在する
- 多数あるアンケート回答システムの独立性を高めることで、システム全体の保守性が高まる



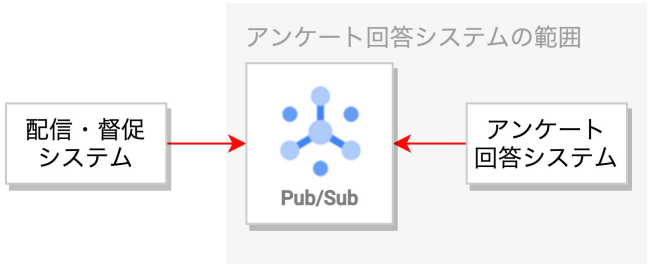
アンケート回答システムの独立性

Pub/Sub による依存関係の整理

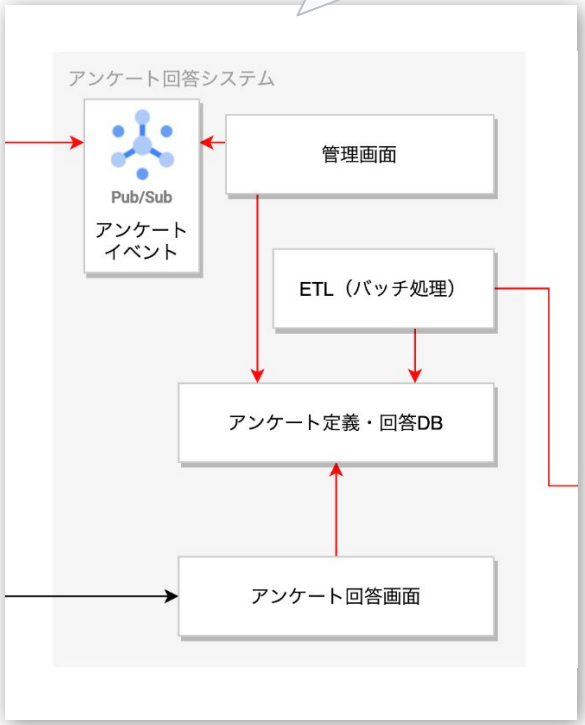
API アクセスでデータ登録すると、呼び出し先に依存



Pub/Sub を介せば依存方向をコントロール可能



アンケート回答システムは
配信・督促システムを知らなくてよい



アーキテクチャのまとめ

マイクロサービスを責務によって分割する

→ **高負荷なデータアクセスの分離**が自然に実現できている

- 配信・督促データを扱うシステムを分け、Spanner でさばく
- 可視化・集計クエリは BigQuery のデータマートで実行する

さらに、依存関係を整理する工夫

→ **アンケート回答システムの独立性**を高め、保守性を向上している

- Pub/Sub を利用し依存関係を反転
- アンケート回答システムは配信・督促を知らなくてよい

Go と Google Cloud の活用
技術選定の方針

03

Google Cloud

最近 BIR で行った開発

時期	できごと (開発コード)	技術要素
2018 年 11 月	集計システムリリース (Racoon)	Go/App Engine（Go 導入/App Engine 導入）
2019 年 8 月	アンケート新管理画面リリース (Tiger)	Kotlin,Spring/オンプレ
2019 年 9 月	新アンケート回答システムリリース (Ibis)	Go/App Engine
2019 年 12 月	配信・督促システムリリース (Dolphin)	Go/App Engine
2020 年 7 月	新アンケート回答システムリリース (Coyote)	Go/Cloud Run（Cloud Run 導入）
2020 年 8 月	配信・督促システムバックエンド Cloud Spanner 化	Cloud Spanner
2020 年 12 月	全アンケート回答システムが配信・督促システムへ統合	🎉
2021 年 3 月	アンケートアプリリリース (Hound) 全システムクラウド移行	Flutter

技術選定方針

ビジネスを加速するために最適な技術選定をしたい

- キャッチアップのコストが少ない技術→ Go
- 高い抽象度で扱うことができるインフラ→ Google Cloud

Go をチームに導入した理由

下記のメリットがあったため（Java / Kotlin が既存システムでは使われていた）

- 静的型付＋オブジェクト指向的な考えで作れる
- いろんな書き方ができない（いい意味で）
- コンパイル・起動が高速

Web フレームワークは利用しない

シンプルな構成で、キャッチアップや保守を効率化

- フレームワークは言語よりも寿命が短い、習得ハードルも高くなる
- net/http とコード自動生成の組み合わせ、Clean Architecture で整理

https://speakerdeck.com/juntaki/godeapisabawohayakutukuru

Google Cloud を導入した理由

インフラを抽象度高く扱うことができる

→ 特にサーバーレスなコンポーネント（App Engine や Cloud Run）を中心に活用している

コントロールできる要素が少ないことがメリットになるようにシステムを組む必要がある

- 負荷に応じたコンポーネントを選定する
- システムを適切に分割し、結合度を低くする

Cloud Spanner の採用と移行

Cloud SQL vs. Cloud Spannerでチーム内で議論

下記に加えエンジニアの技術的な挑戦という観点も加味し Cloud Spanner を採用した

- 可用性
- スケーラビリティ

移行にあたって実施したこと

- 適切な PK やインターリーブを考慮したテーブル設計に変更
- コード上は DI する Repository の差し替えで対応できた

現在の運用

当初 1 Node で稼働

→ 利用範囲が拡大し、負荷の増加に伴い 2 Nodeに拡張

Pub/Sub をつかう上での工夫

at-least-once 配信、順序指定できない※という Pub/Sub の特徴を踏まえて、データベースと整合が取れるようにしている

（※現在はできるようですが、配信・督促システム設計当時はまだ無い機能でした）

ロールバックへの対応

1. トランザクションの中で pubsub_message テーブルに書き込み
2. テーブルのメッセージをバッチ処理で Publish

Publish できたものは、届いたものとして扱えるのが良い点（リトライは　Pub/Sub に任せられる）

更新される可能性がある情報のメッセージ（or 大量の情報）

対応する情報取得 API の URL をメッセージとして Publish する

- 最後に Subscribe した時点で、取得できた情報が最新となる
- 配信対象リストのような大量の情報も Pub/Sub のメッセージとして通知可能

まとめ

マイクロサービスを責務によって分割・依存方向を整理する

- 高負荷なデータアクセスの分離
- 依存を減らしてアンケート回答システムの独立性を高める

ビジネスを加速するための最適な技術選定

- キャッチアップのコストが少ない技術 → Go
- 高い抽象度で扱うことができるインフラ → Google Cloud
  → 運用がかんたんで可用性の高い RDBMS → Cloud Spanner

エムスリーでは技術力で医療業界を前進させるエンジニアを募集しています！

Thank you.