Google Cloud

マネージドサービスを活用した Google Cloud でのモダナイゼーション

坂田 純

Ubie 株式会社

Software Engineer, Site Reliability

スピーカー自己紹介

坂田純
Ubie 株式会社
Software Engineer,
Site Reliability

- ID: @sakajunquality
- Ubie 一人目の SRE として入社
- SRE を軸としつつセキュリティやプラットフォーム全般の業務をこなす
- Google Developers Expert (Cloud)

アジェンダ

- Ubie のビジネスとインフラ・アーキテクチャー
- マネージドサービスの活用事例
  - アプリケーション
  - データベース
  - モニタリング
- まとめ

Ubie のビジネスとインフラ・アーキテクチャー

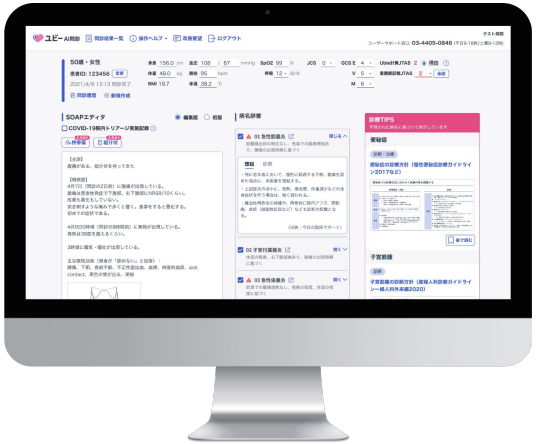
Ubie について

ユビー AI受診相談

症状から受診の手がかりがわかる

ユビー AI問診

診察事務を ⅓ に効率化



クラウドマイグレーションからスタート

2018 年にそれまでは他の PaaS で動いていたものを Google Cloud に移行。

比較的早い段階でのマイグレーションなので容易だった。

2018 年

1	3 ~ 5	~ 15
プロダクト	マイクロサービス	社員

複数プロダクトを展開し事業を加速

徐々にビジネスが拡大し、**複数のプロダクトを同時に展開**

当然全てのプロダクトが順調なわけではなく、**小さく始めて**すぐに開発・運用を**中断**する場合も

また、2021 年からは海外版をリリースし、**数ヶ国**に展開

2018 年	1 プロダクト	3 ~ 5 マイクロサービス	~ 15 社員
2020 年	3 プロダクト	20 マイクロサービス	~ 100 社員
2022 年	~ 10 プロダクト	~ 50 マイクロサービス	~ 200 社員

SRE の採用はなかなか追いつかない

事業が拡大する一方で、**SRE の採用はなかなか追いついてない**という課題も

“Kitchen Sink” *1 の状態が長く続いた
(最近では一部 Embedded のような動きもできている)

2018 年	1 プロダクト	3 ~ 5 マイクロサービス	~ 15 社員	1 SRE
2020 年	3 プロダクト	20 マイクロサービス	~ 100 社員	2 SREs
2022 年	~ 10 プロダクト	~ 50 マイクロサービス	~ 200 社員	6 SREs

*1 https://cloud.google.com/blog/products/devops-sre/how-sre-teams-are-organized-and-how-to-get-started

事業として求められること

01 | スタートアップとして**仮説検証の機会**を多く創出できる環境を作る

02 | 事業の変化に応じてアーキテクチャを**柔軟にアップデート**する

03 | ヘルスケア プロダクトとしての**高水準のセキュリティ**

クラウドを使う上で意識をしていること

01 | マネージドサービス / セルフマネージドに関わらず、
“運用” 業務の負担を限りなく小さくする

02 | 容易に**各コンポーネントがリプレイス可能**であることで、負債解消のコストを下げる

03 | 生産性/コスト/セキュリティの**バランス**を取る

マネージドサービスの活用事例

アプリケーション編

アプリケーション

- Cloud Run / GKE の使い分け
- GKE Autopilot の活用

Cloud Run / GKE の使い分け

- Ubie ではアプリケーションを Cloud Run または Google Kubernetes Engine (GKE) に集約
- アプリケーションに求められる要件によって使い分けている
  - Cloud Run は独立して気軽に利用開始することを目的
  - GKE には Istio や Prometheus などを導入しマイクロサービスを扱いやすくしている
    - またセキュリティ要件で、サービス間の認可ポリシーやセキュリティポリシーを細かく指定する必要がある

Cloud Run / GKE の使い分け

Cloud Run

- HTTP/gRPC のコンテナをデプロイ
- **制約は多いものの**、アプリケーションが立ち上がるまで最速でできる
- まずは**スピード重視でリリース**をする際に使用する事が多い
- 必要に応じて後に GKE に移行することも容易

Google Kubernetes Engine

- 柔軟なワークロードに対応
- Ubie では Istio サービスメッシュなど**マイクロサービス環境の恩恵**を受けられる
- 柔軟な反面、**社内のセキュリティ制約**などに対応する必要もあり
- 中期的に運用していくものや、既存のサービスとの連携を重視するさいに利用

Cloud Run / GKE の使い分け

Cloud Run

e.g.

- 独立して新しく始めたサービス
- 社内の管理画面やツール

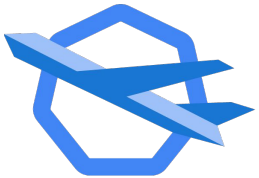
Google Kubernetes Engine

e.g.

- JVM などコールドスタートが気になるサービス
- 既存の他マイクロサービスとの連携を最初から検討しているサービス

GKE Autopilot の活用

- Ubie ではメインのマイクロサービス群を含むクラスターとは別に、**一部顧客ごとにクラスターを用意** している
- もともと **GKE Standard** を使用していたが、数多くのクラスターを運用するのは負担になっていた
- 幸い GKE Autopilot の制約に引っかからないので、 Autopilot にマイグレーションした
- メインのクラスターは Standard を継続して利用



GKE Autopilot / Standard

GKE Autopilot

- Control Plane に加え Worker(Node Pool) もクラウド側で管理されているため、ユーザー側でのクラスターの**運用コストが小さい**
- **制約も多く**ワークロードによっては合わない

GKE Standard

- Worker は自分たちで管理しているので **スケールが速く制約が少ない**
- Release channel などの機能があるものの、自分たちである程度**メンテナンスが必要で運用コストが Autopilot より高い場合**がある

GKE Autopilot / Standard

GKE Autopilot

- Ubie の個別の顧客へのクラスターは、B2B で**突発的なトラフィック増加などもなく**、メインのクラスターほど**複雑な要件もない**
- Autopilot を使うことによる制約よりも**運用負荷軽減のメリットが大きいと判断**

GKE Standard

- メインのクラスターでは B2C のマイクロサービスも含むため**突発的なトラフィックに耐えるスケール性**が必要
- Istio などマイクロサービスを扱いやすくするコンポーネントも Autopilot では動かない
- **運用コストよりも柔軟に使える恩恵のほうが大きいと判断**

01 | プロダクトの**フェーズに適したアプリケーションのランタイムを選択**し**事業の変化に対応**

02 | GKE Autopilot のように**自社にハマるマネージドサービス**を利用し**運用負荷を軽減**

マネージドサービスの活用事例

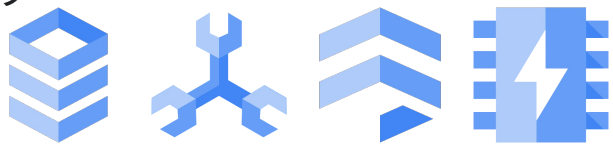
データベース編

データベース

- 全般的にマネージドサービスの利用
- Cloud Spanner へのシフト
- Cloud SQL / Cloud Spanner の機能活用

全般的にマネージドサービスの利用

- GKE クラスターの運用で考慮事項を減らすため、アプリケーションのクラスターは **ステートレスに保つことを徹底**
- データベースはすべてマネージドで **Cloud SQL** または **Cloud Spanner** を多く利用
- 目的に応じて Cloud Firestore や Cloud Memorystore も利用
- どうしても自前での運用が必要な場合は、アプリケーションとクラスターを分けるなど **ライフサイクルを意識** 。



全般的にマネージドサービスの利用

Cloud SQL

- PostgreSQL や MySQL など比較的**開発者が慣れている RDBMS** を利用することができる
- Query Insights や Automatic Backup など**マネージドならではの機能**も豊富

Cloud Spanner

- 複数リージョンを使用した**高い SLA** や**高いスケーラビリティ**を実現
- **ダウンタイムのないスケール**が可能
- 従来の 10 分の 1 のインスタンスサイズが作れるなどコスト面でも優しくなってきた

全般的にマネージドサービスの利用

Cloud SQL

Ubie では クラウド移行前から PostgreSQL を使っていたものや、**計画メンテナンスなどでサービスを止めることができるもの**に使用

Cloud Spanner

新規プロダクトでは積極利用している

Cloud Spanner へのシフト

- プロダクトがスケールするにあたって **Cloud Spanner** のほうが運用上都合がいい場合が増えている
  - e.g. ダウンタイムを伴わないスケールなど
- BtoC で一般の方に利用いただくプロダクトは **Cloud Spanner にマイグレーションを進めている**
- 移行にもそれなりに大変なので、**必要ないものはしない**
- 一方で**開発生産性の観点**もあり Cloud Spanner PostgreSQL Interface などにも今後期待

Cloud SQL / Cloud Spanner

Cloud SQL

- スケールアップ・ダウン、メンテナンスの際にダウンタイムが発生する
- 短くなったとはいえ定期的にメンテナンスが発生する
- リードレプリカのような一般的な負荷分散が出来る

Cloud Spanner

- Processing Unit (i.e. 処理能力) の変更にダウンタイムが伴わない
- この処理の能力をオートスケールしている他社事例も
- スキーマ設計を間違えると通常の RDBMS よりパフォーマンスが出ないことも

Cloud SQL / Cloud Spanner の機能活用

- アプリケーションとは違いデータベースの移行はそう頻繁には行えない
  - 特にプロダクトが成長するとデータ量は増大
- Cloud Spanner / Cloud SQL ともに**マネージドとしての強みを生かして運用**

Cloud SQL / Cloud Spanner

Cloud SQL

- **Query Insights** でのパフォーマンスの可視化やトラブルシューティング
- **自動バックアップ**や **Serverless Export** などによる運用負荷軽減
- **Federated Query** によるデータパイプラインの構築

Cloud Spanner

- **Key Visualizer** によるデータ分散の確認
- **Query Statistics** によるトラブルシューティング
- **Federated Query** によるデータパイプラインの構築

01 | マネージドデータベースを利用することで**アプリケーションのクラスターはステートレス**にする

02 | プロダクトによっては Cloud Spanner を利用して、**スケール性やアップタイム**を確保

03 | データベースの移行は大変なので**利用しているものを最大限有効活用**

マネージドサービスの活用事例

モニタリング編

モニタリング

- Cloud Monitoring と Prometheus の使い分け
- Google Managed Service for Prometheus (GMP) へのマイグレーション

Cloud Monitoring と Prometheus の使い分け

- 基本的に **Cloud Monitoring** を利用
- Logging / Monitoring / Trace を統合して使う
  - **ログはボリュームが大きいとそのままコストに響く** ので
    ログがあるところにモニタリング全般を集約している
- 一方で、Ubie 独自のメトリクスの取得には **Prometheus** も利用
- Cloud Monitoring / Prometheus ともに Grafana でダッシュボードを作り可視化

Prometheus

- **サービスメッシュ**を利用したサービス間通信の可視化や、**アプリケーション固有**のメトリクスの収集に Prometheus を利用
- もともとはオープンソースと他社のマネージドのものを併用していたが Google Managed Service for Prometheus (GMP) に移行

Cloud Monitoring / Prometheus

Cloud Monitoring

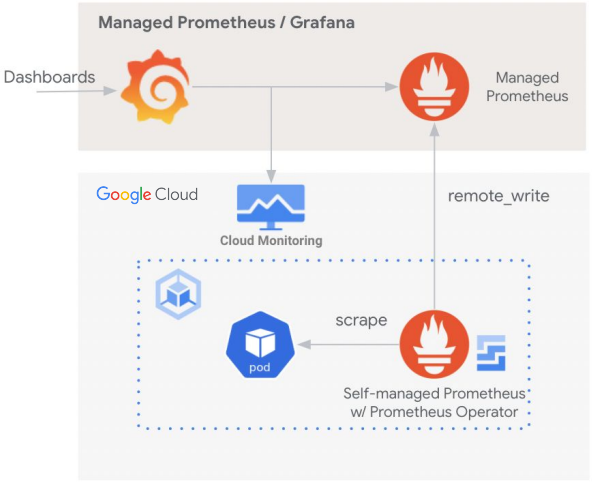
- マネージドサービス自体のメトリクス中心

Prometheus

- Istio サービスメッシュのメトリクス
- Micrometer などのアプリケーションのメトリクス
- Fastly や Sentry など外部サービスのメトリクス

Prometheus (GMP以前)

- GMP 移行以前は GKE 内に Prometheus Operator を利用した Prometheus を構築
- 他社マネージド Prometheus に remote_write という方法で集約していた
- **自前で運用するコストが一定かかりつつも** Cloud Native で標準的な Prometheus を利用し、**Cloud Monitoring だけでは取れない細かなメトリクス**をサービス開発に活かして行くのが目的



GMP で運用負担軽減

メトリクスの収集

- Operator を利用しているものの、**Prometheus 自体の監視や運用が必要**だった
- GMP の **Managed Collection** を利用することでクラスター内で**運用するものがなくなる**

ストレージ

- 他社のマネージド Prometheus は Cortex ベースになっており中長期的なスケールの懸念があった
- コストも右肩上がりで問題になっていた
- GMP はストレージに **Cloud Monitoring と同じ Monarch** が使われておりスケールする

01 | 場合によってはマネージドサービスを**オープンソースで補完**し目的を達成

02 | オープンソースで自前運用のものも、よいマネージドサービスがあれば**積極的にマイグレーション**を行っていく

まとめ

Your end customers do not care how good you are at managing monitoring infrastructure. They care how good you are at keeping your app up”

Lee Yanco

Product Manager , Google Cloud

Introducing Google Cloud's new managed service for Prometheus
https://www.youtube.com/watch?v=7m3CzLULM-8

プロダクトを開発・運用していくにあたって、クラウドを使用することはゴールではない

マネージドサービスを**自社にあった形で**継続的に取り入れて行くことで、より一層、自社の**ビジネスロジックにフォーカス**することが出来る

Thank you.

Google Cloud