4,000 社に利用されるモノリシックな CMS をクラウド ネイティブなヘッドレス CMS にした話

株式会社ディバータ　加藤健太

Google Cloud

Google

最初に

01

自己紹介

Diverta

株式会社ディバータ

主にRCMSをSaaSで提供している会社。

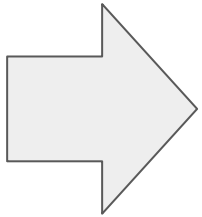
「ただしく、たのしく。」をスローガンにしている。

代表取締役 加藤 健太

twitter: @katoh_kenta

会社経営よりシステム考えてる方が好き。

RCMSからKurocoへ



RCMSの紹介

4000社に利用されている2005年リリースの
国産のSaaS提供型のCMSです。
クラウド型CMSという呼び方もしていますが、2005年当時からSaaS（最初はASPと言ってました）で提供していました。
基本的にはSaaS提供ですが、オンプレや専用サーバでの提供も可能になっています。

Kurocoの紹介

クラウドネイティブ

エンタープライズ向け

API志向

マイクロサービス志向

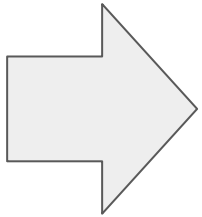
ヘッドレスなCMSです。

RCMSをベースに改造されて、

現在、7月よりクローズドβ提供で、

10月正式リリース予定になっております。

なぜ成長しているサービスをリニューアルしたのか？



ビジネス的な現状

- ARR 4億円くらい
- 年15%程度の安定しているが高くない成長率
- あまりワクワクしない
- 海外マーケットに挑戦したい
- どうしても海外マーケットに挑戦したい

リニューアル理由はほぼ気持ちの問題かもしれない

海外マーケットに挑戦できなかった理由

- 国内でもそこそこ稼げた
- 英語が苦手
- RCMSのフロントエンド構築は独自の記法と理解が必要

- 外国籍エンジニア50%以上、エンジニアは英語でやり取り
- 5年前から全社で英語の勉強を強化
- 技術の変遷でCMSにフロントエンドは必要なくなった

今がちょうど状況が揃ったタイミング

この10年のRCMSのアーキテクチャ

・SaaS

・マルチテナント

・nginx/Apache/PHP(4=>7)/PostgreSQL(7=>12)

・AWS（ただしEC2メイン）

(2011年の4日間に渡る大規模障害を踏んでますので、10年近く利用しています)

ぎりぎりクラウドCMSと呼べるレベルのアーキテクチャ

CMS特有の問題

・データ量やロジック、利用方法の想定が難しい

・リクエスト数の増減の想定が難しい

・様々な機能が複雑に組み合わせて利用される

・CMSというよりはWEBアプリケーションフレームワークとして　利用されることも多い

プラグインなどで機能を分割するようにしてきたが、

やはりモノリシックにサービスが構成されていく

ところで、現在の技術やアーキテクチャの方向性は？

- クラウドネイティブ、サーバレス、マイクロサービス
- DevOps / CI / CD
- 様々なSaaSサービスの成熟とAPI
- AWS/GCP/Azureなどのクラウドベンダーの成熟
- フロントエンド技術の発達
- エッジコンピューティング
- プログラミング言語の多様化

何が起きているのか？

・モノリシックにサービスを作ると考慮しないといけないことが多くな　りすぎて破綻するようになってきた。

・非同期処理がフロントエンドでも利用されるようになってきて、バッ　クエンドへ細かい粒度のリクエストが増えた。

・バックエンド＋CDN（エッジ）＋フロントエンドでそれぞれ処理する　など処理レイヤーの複雑化

・バックエンドは複数レイヤーで構成（マイクロサービス化）

・処理速度の高速化要求と非同期処理の増加

次の10年はどうなるか？

各サービスレイヤーが

・コンパクトになる

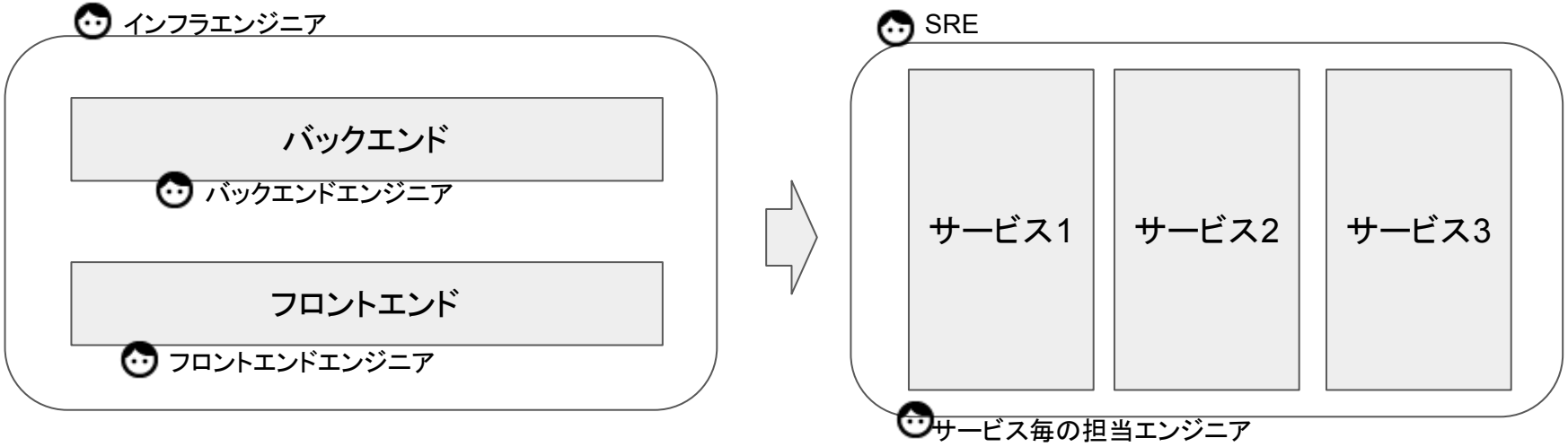
・ネットワークが高速化

・スケーラブルになる

・ロジック部分は多様化・高機能化

そして、人間との接点（UI）が大きな差別化要因に

サービスの中身が横割りから縦割りになる



アーキテクチャ毎での役割分担

サービス毎での役割分担

では、RCMSをどう変えるか？

・フロントエンドとdecouple（分離）して、クラウドとcouple（結　合）する

・モノリシックに作られたサービスを分解して、マイクロサービス　的に再構築する

・サービスのUI構築を支援できるContents as a Service的に

クラウドネイティブなヘッドレスCMSへ

最も重視している方向性

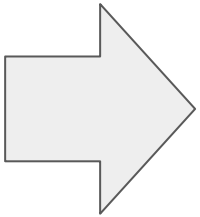
・WEBサービスの発達によりAll in One的なサービス提供での　差別化しづらいので、強みにフォーカス。

・人間に対してはUI、機械に対してはAPIを。

※静的型付け、CLIをDockerで作る、マイクロサービス、X as a code、全て同じ方向性に思える。

今後はマイクロサービス的な発想・考え方が最も大事

なぜプラットフォームとして GCP を選択したのか？



RCMS on AWS

Kuroco on GCP

AWS vs GCP

・AWSは、顧客の要望を元にサービスを構築していくスタイル

・GCPは、Google社内で利用されているサービスを一般向けに改変して提供していくスタイル

AWSにはとてもお世話になってます。大好きです。

でも、マイクロサービスやるならGCPと判断しました。

GCPがマイクロサービスに適しているポイント

マイクロサービス構築で重要なポイントは、

・サービス間のネットワーク（レイテンシ）

ゾーン間のレイテンシがほぼゾーンが気にならないレベル

・サービス間での認証

プロジェクトという各サービスの上位概念があることからも分かるように

サービス間での認証が設定しやすい

・全サービスの詳細で統一されたログ管理

BigQueryへのexport機能もあり、非常に充実している

AWSも使ってます

・Kurocoのデータストレージ部分を独立させたいという要望もあり、その場合は個別の領域を必要とするのでAWSを利用しています。

つまり、

AWSもGCPもどちらもよいサービスなので、使い分けましょう。（Azureはすみません、まだ試してないです）

では、
実際に何をしたか

02

Kubernetes(GKE)に載せる

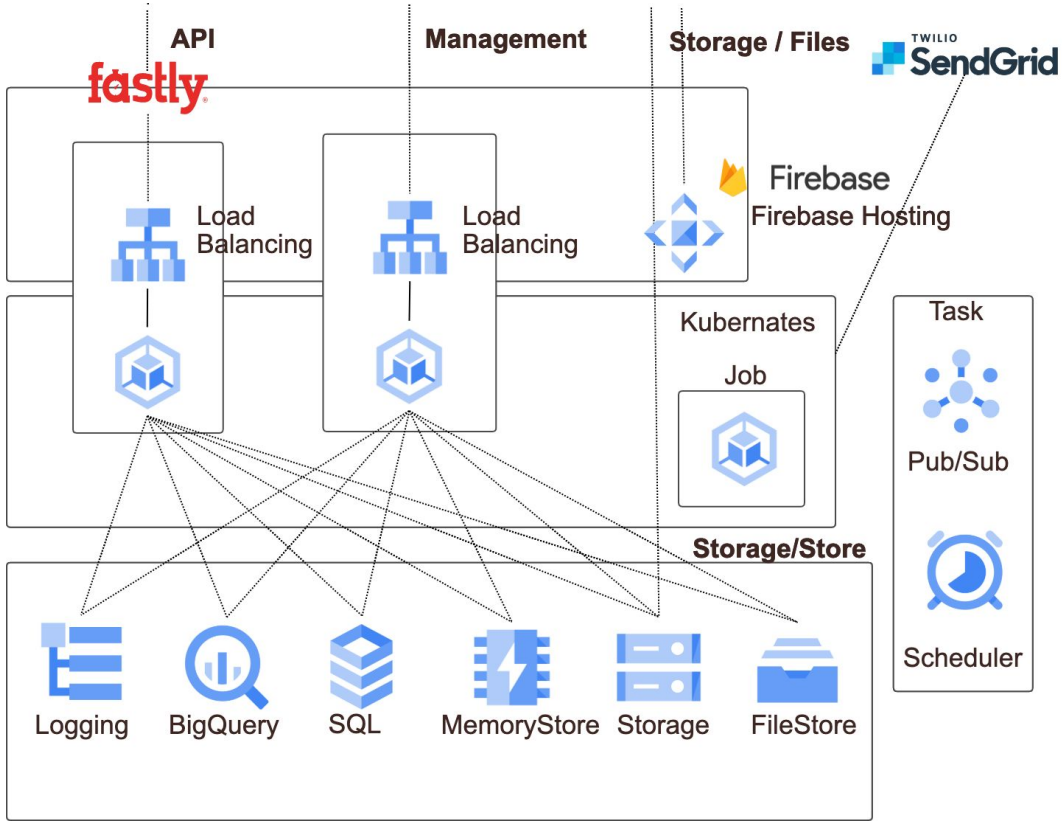
・RDB、メモリDB、ファイルストレージ、メールサーバなどはマ　ネージドサービスへ移行

・Kubernetes上で実行するのか、Cloud Function/Cloud Runで実行　するのかも検討して分割を進める

・Kubernetesであることは将来、効果が発揮される想定

・サービスの細分化、マイクロサービスへの移行準備

サービス構成図



・API/Management/Storageと大きく3ルート

・APIルートはFastlyを必須にしています

機能の大幅な削減

オプション機能だけでも100近い機能を削除

・10年前はスポーツチーム向けを目指してた

ラグビー、サッカー、野球などスポーツ関連

・トランザクション情報の重複機能多数

掲示板、コメント、投票

・項目の定義が違うだけのマスタ情報が多数

会社情報・採用情報・用語辞典、、

15年以上の開発の後方互換対応箇所を削除

各機能を抽象化して重複部分を削除

外部サービスを利用できるところはAPI利用前提に

・メール送信関連はSendGridに委譲

メール送信の様々なロジックを一部委譲

・ファイルストレージはGCSなどのクラウドストレージに

NFSを利用するなど、冗長化で問題が発生しやすい箇所だった

・フロントエンド機能はすべて削除

RCMSの強みの部分もあったが、今後を考えてあえて削除

All in Oneではないが、トータルで機能アップ

機能やプログラムの大幅な削減の効果

- ファイル容量にして、1/2にプログラム量を削減
- 機能やプログラムがシンプルになりリファクタリングしやすく
- 新しいアーキテクチャを利用したリファクタリングしやすく

ソースコードがスッキリ

メンテナンス性やエンジニアの実装効率が向上

Go/Node.jsなど複数言語を利用できるようになった

今後10年への準備ができた

ワークロードの定義と分割

・APIの種類によって利用プロセス数、CPU/Memoryの利用、認証の有　無、キャッシュ動作などを明確に定義

・管理画面(管理画面API)の表示に関する利用プロセス数、CPU/Memoryの利用量を定義

・バッチ処理（非同期処理）の利用プロセス数、CPU/Memoryの利用量　を定義

・CMS特有の様々な利用法に対して、まだまだ細分化してく必要がある

今後やっていくこと

・プログラムのさらに分割してマイクロサービスを増やす

プログラムの管理レポジトリは統一したまま、

実行パスだけ細分化するような方法を検討中

・CD/CIの整備

個人的には今、一番面白いと感じてる

・開発環境の再構築

現在は、Docker-composeでローカル環境作ってるが、、

まだまだやることはたくさんある

やってみて分かったこと

03

コストまあまあ掛かる

RCMS

Compute Engine(EC2) 中心

Kuroco

マネージドサービス中心

コンピューティングコストは約1.5倍のコスト増の感覚値

クラウドの潤沢なリソースに甘えた富豪的な

サービス構成のままであれば当然の結果

CPUとメモリと起動速度を意識する

Compute Engine(EC2) 時代は常時起動VMで、プロセス数の増減を抑える設定とSWAPメモリと大雑把なオートスケール

Kubernetesにモノリシックなサービス全体を移行してから、CPUとメモリの利用が安定しないサービスをCloud RunとCloud Functionなどのサーバレスへ

CPUとメモリの利用量を安定させる

サーバレスサービスはコスト削減の切り札

同じソースコードを利用していたとしても、実行環境を変えることでコストやパフォーマンスを大幅に抑えることが出来る

マイクロサービスのメリットを活かして、効率良くコンピューティングリソースを使うことでコスト削除が可能。

富豪的インフラ・プログラミングをせずに

CPUとメモリ利用量も考えてプログラムする

実行環境を変更する例

/login => Kubernates

DBとの連携や各サービス連携がしやすいように

/list => Cloud Run or Kubernates

検索ロジック次第でDBからのレスポンス待ちや

取得データ量によってメモリ利用量が変わる

/upload => Cloud Functions

頻度が少なく、実行時間やメモリ利用量が大きく変わる

やはり新しい技術は楽しい

・Kubernetes、マイクロサービスはノウハウが少ないが、試行錯誤は楽しい
・なんでも自前で作るのではなく、コアバリュー以外は他のサービスの力を借りる
・今、注目しているのはFastlyやCloudflareなどが出してるエッジコンピューティングサービス

各サービスが安く・早く・シンプルになってる

これらを繋げて面白いものを作るべし

マイクロサービスの良いところ

- これまでの競合が協力相手になった
- 自前でなんでも実装していくことを避けられる
- 使ってみたいプログラミング言語を使える
- リファクタリングしやすい
- リファクタリングがコスト削減に直結する

本当に良いことが多いと感じています

マイクロサービスの辛いところ

・どこで何が起きているのかを直感的に把握しづらい

・やはりレイテンシは気になる

・セキュリティ・認証が複雑化する

ログの上手な取得やネットワークの高速化

セキュリティノウハウの蓄積で解決したい

UI大事

・海外マーケットに展開したいと考えてユーザー調査をしたとこ　ろ、UIに関して特にシビアに評価をされた。

・現在はリリースが遅れても仕方ないくらいの大幅なUI改造に着手　している。

・マイクロサービスを設計しているうちに差別化要因としては

　最後にUIが　残るのではないかと気がついた

APIは機械との接点・UIは人間との接点

小さいピボットしながら進行

・2019年9月

最初はただRCMSにAPI機能を追加してみるプロジェクト

・2019年12月

ヘッドレスCMSとして、フロントエンド機能の削除を決断

・2020年4月

クラウドネイティブにすることやマイクロサービスを意識

・2020年8月

管理画面UIを大幅に改修することを決断

まとめ（ビジネスサイドの方へ）

今後、All in Oneでサービスを作っていくことは大変だと考えられます。
例えば、Shopifyはヘッドレス・コマースと言われることもありますが、このような柔軟で、スケーラブル、UI/UXをコントロールしやすいサービスを作っていくには、様々なサービスを組み合わせて、コアバリューに特化するという観点も必要になってくると思います。

まとめ（エンジニアサイドの方へ）

エンジニアはやるべきこと、知っておくべき事がどんどん増えているように感じます。
バックエンドが楽になったなと思えば、フロントエンドにタスクが移っただけだったり、非同期処理が増えたり。
ただ、技術の流れとしては、様々なものがカプセル化できるようになり、自分達でやれることが増えてきたのも事実ですので、この流れを楽しんで開発していきましょう！

Kurocoの未来

04

B2BのSaaSを作る基盤として

・B2BのSaaSサービスはマルチテナント型なので相性が良い

Kurocoもマルチテナント

・クラウドネイティブなので、従量課金でスケーラブル

・APIを利用して他のサービスに付与するような形の利用が可能

・SaaSサービスを構築する際はコアバリューにフォーカスして、 他の機能はAPIで協働させる方向になる

Kurocoの事例

・ECサイトの商品管理基盤と決済・カート基盤と連動して、
　大規模ECサイトのフロントエンドリニューアール
・UI/UX重視の会員制メディアサイトのバックエンド
・特化型B2Bアンケートシステムのサービス基盤
・様々なSaaSサービスと連動させたリワード提供サービス基盤

※RCMSをKurocoのように利用しているものも含まれています。

Thank you