CUS-31

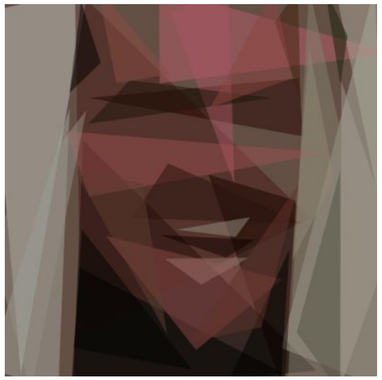
事業と歩む
Ameba システム刷新の道

岩永 翔
株式会社サイバーエージェント
メディア管轄 Ameba事業本部 CTO

難波 和也
株式会社サイバーエージェント
メディア管轄 Ameba事業本部 Engineer

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

本セッション資料や記載内容については一切の転用を禁止しております



Engineer / Ameba 技術責任者

岩永 翔

KAKERU IWANAGA

２年ちょっと前からAmebaの技術責任者してます
Web技術とSRE界隈の話題が好きです
好きなたい焼きの味は栗きんとんです
アマゾン ウェブ サービス (AWS) 大好きです！

Engineer

難波 和也

KAZUYA NAMBA

Amebaでエンジニアをシています
3児の父です。子供達とLEGOで遊んでます。

Contents

CHAPTER1
Amebaの歴史

CHAPTER2
刷新プロジェクト立ち上げ

Phase 1. Vision/Mission策定

Phase 2. プロジェクト化

Phase 3. 実現可能性の精度上げ

CHAPTER3
AmebaPlatformの概要

CHAPTER4
まとめ

CHAPTER

1

Amebaの歴史

History

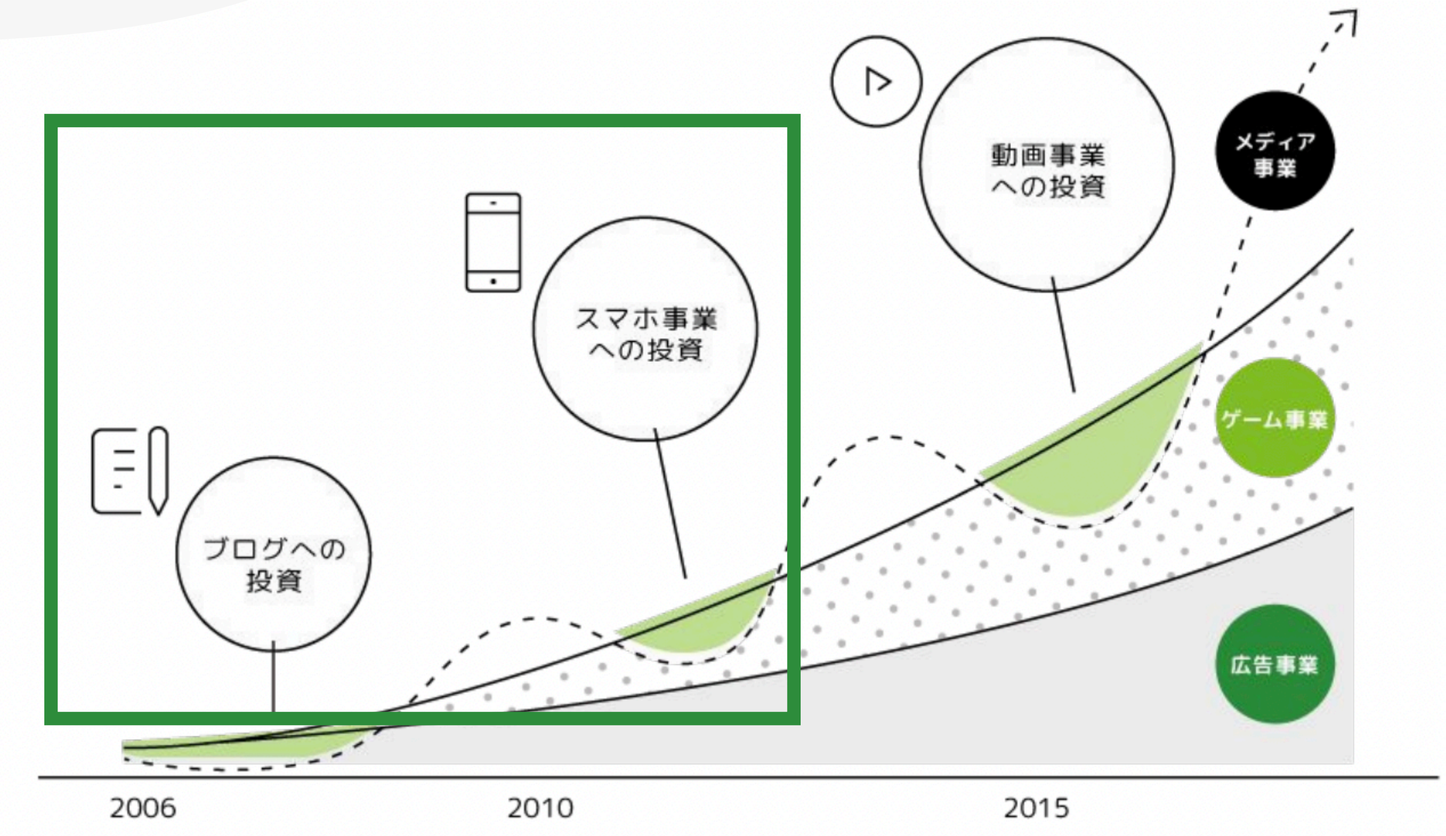
Amebaはブログを中心とした
メディアプラットフォーム

ブランド	Ameba®							
カテゴリ	ブログ	ニュース	マンガ	スキルシェア	占い	ホームページ	ポイント	アバター
サービス	Amebaブログ Ameba Pick	Ameba News	Amebaマンガ	REQU by Ameba	Ameba占い館 SATORI Ameba占い館 SATORI 電話占い	Ownd by Ameba	.money by Ameba	pigg by Ameba

17年の歴史による様々な変化

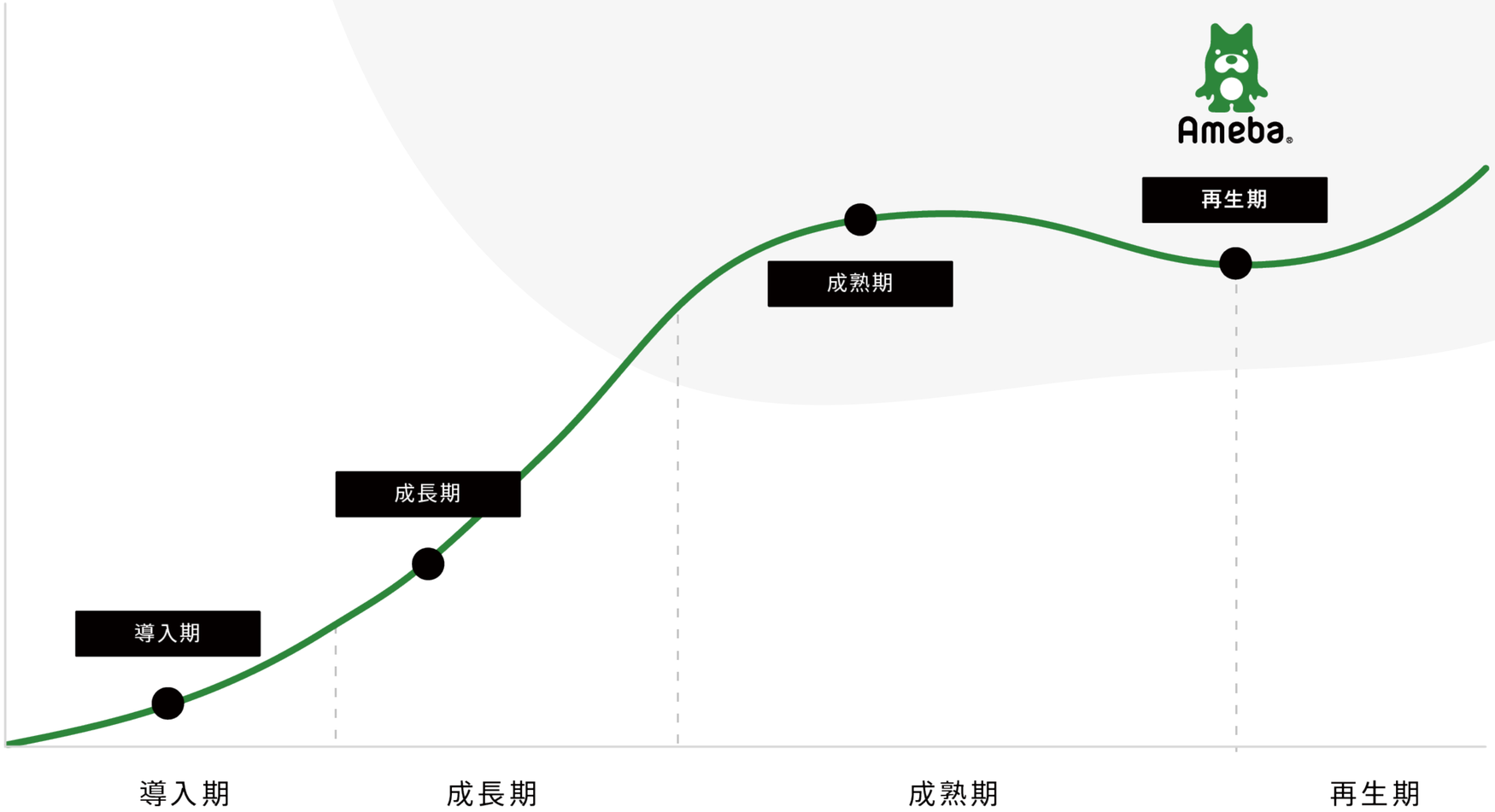
2004	2009	2011	2015	2020
アメーバブログ誕生	Ameba Pigg誕生	スマホシフト	2015 ロゴリニューアル	2020 16周年

メディア事業への投資の歴史



出典：https://speakerdeck.com/cyberagent_recruit/cypitch?slide=23

成熟期から再生期へ



他サービスの台頭

さらに歴史の長さからプロダクトにおいても
古い機能やシステムが多く残っており、
大きな負債も抱えていた。

課題をもう少し深ぼる

事業の継続とともに肥大化したシステム

アカウント数

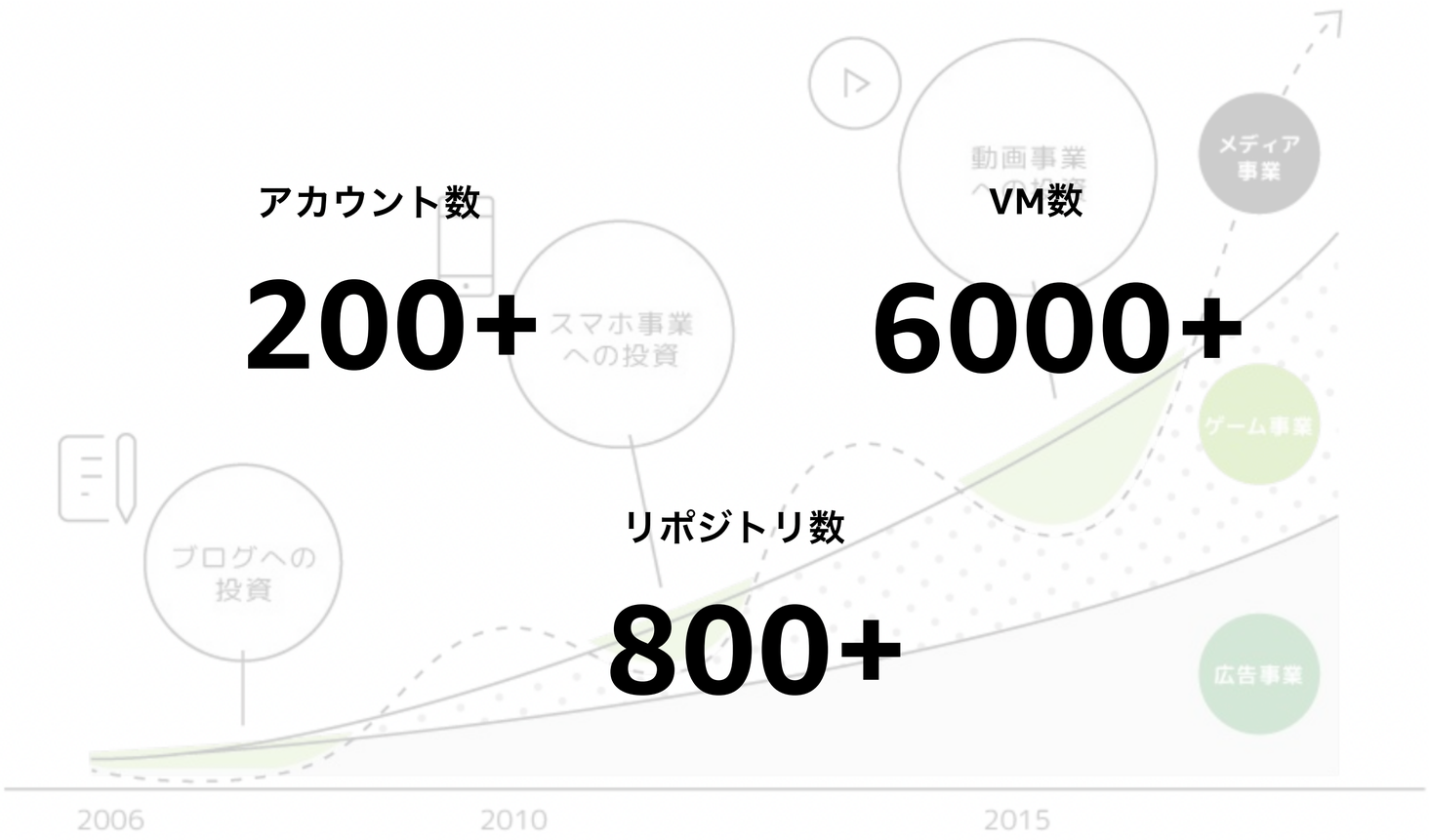
200+

VM数

6000+

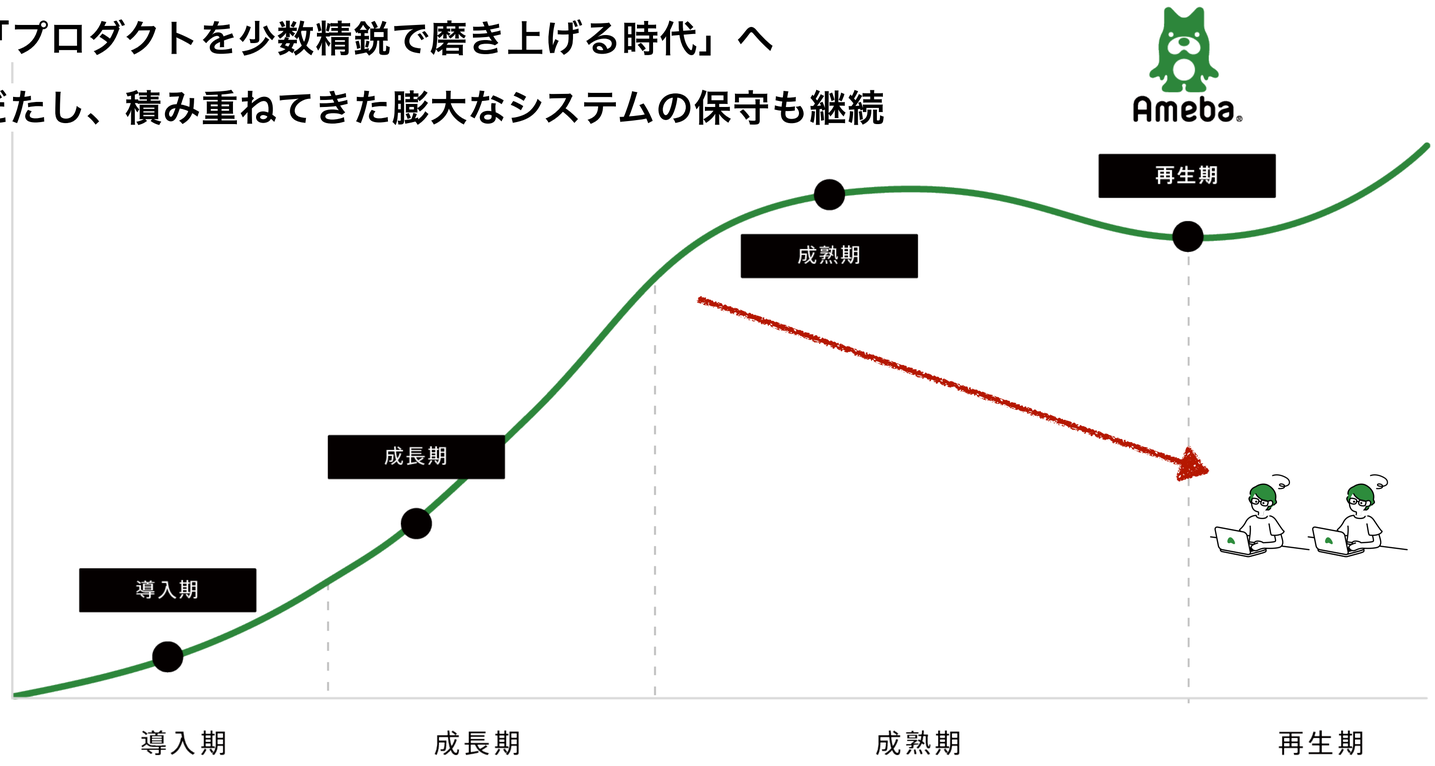
リポジトリ数

800+

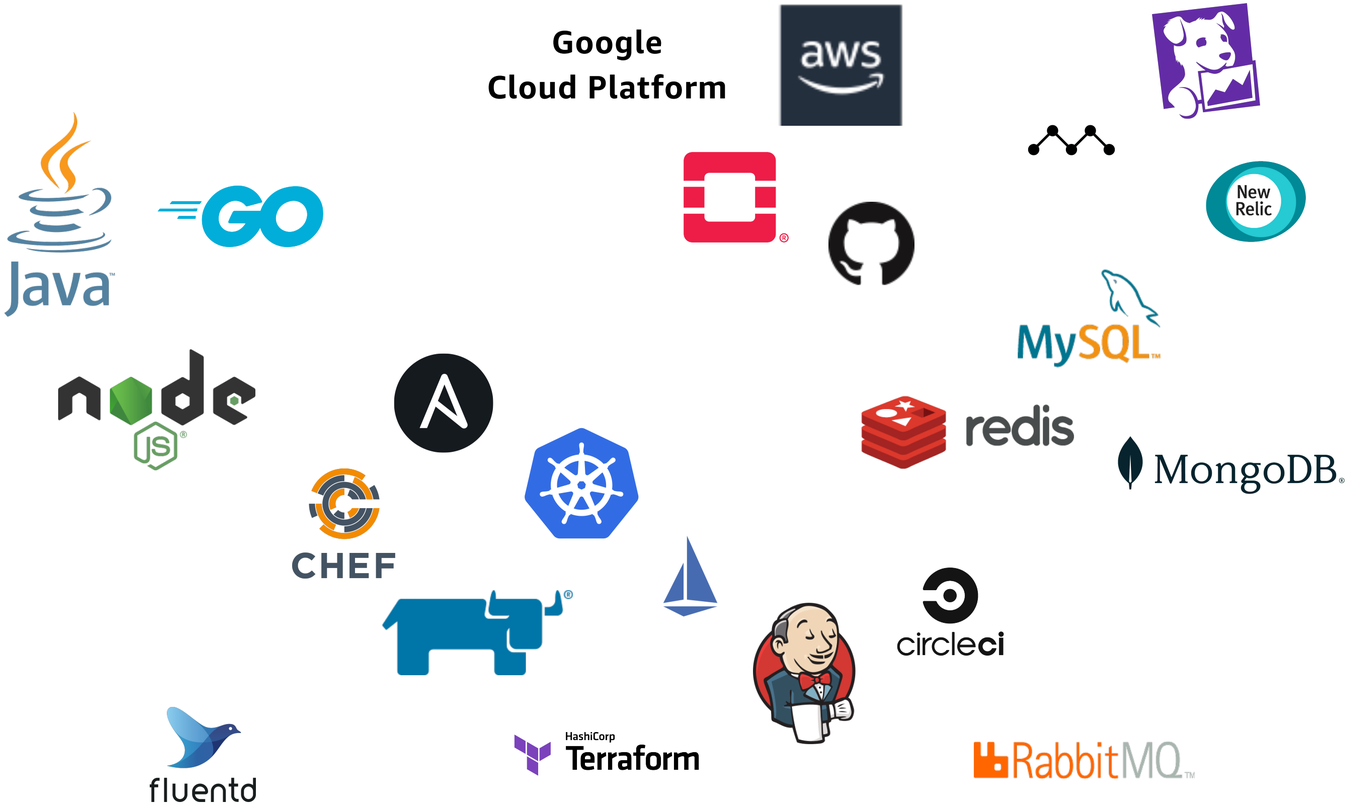


成長期と比較して縮小した組織

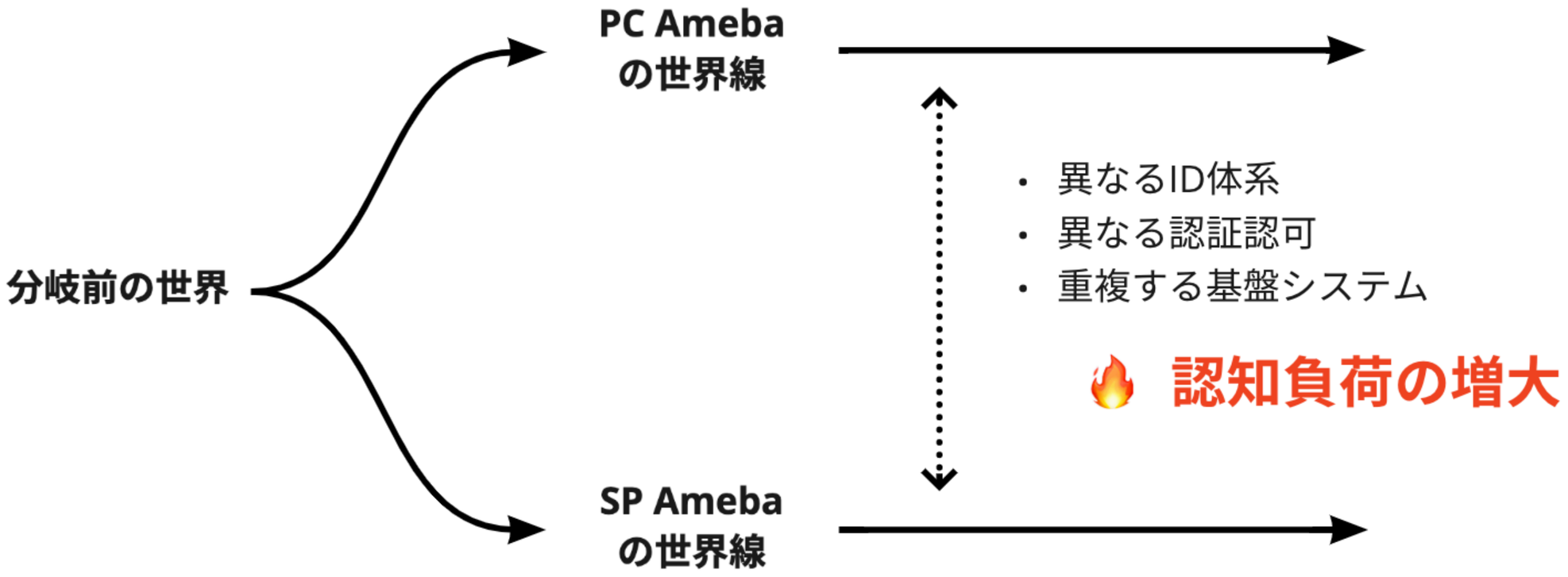
- 「大人数でとにかく新規開発の時代」から、
  「プロダクトを少数精鋭で磨き上げる時代」へ
- だたし、積み重ねてきた膨大なシステムの保守も継続



技術選定の煩雑さによる認知負荷



事業変遷による認知負荷



市場の変化、サービスや組織の変化に合わせて
市場が求めるスピードで、素早く開発できる組織へ
システムも変化する必要がある

CHAPTER

2

刷新プロジェクト立ち上げ

How

AmebaのVisionから
技術戦略で達成する状態目標と
具体プロジェクトへと落としていく

Ameba | Vision

１００年愛されるメディアを創る

Amebaがこの先、

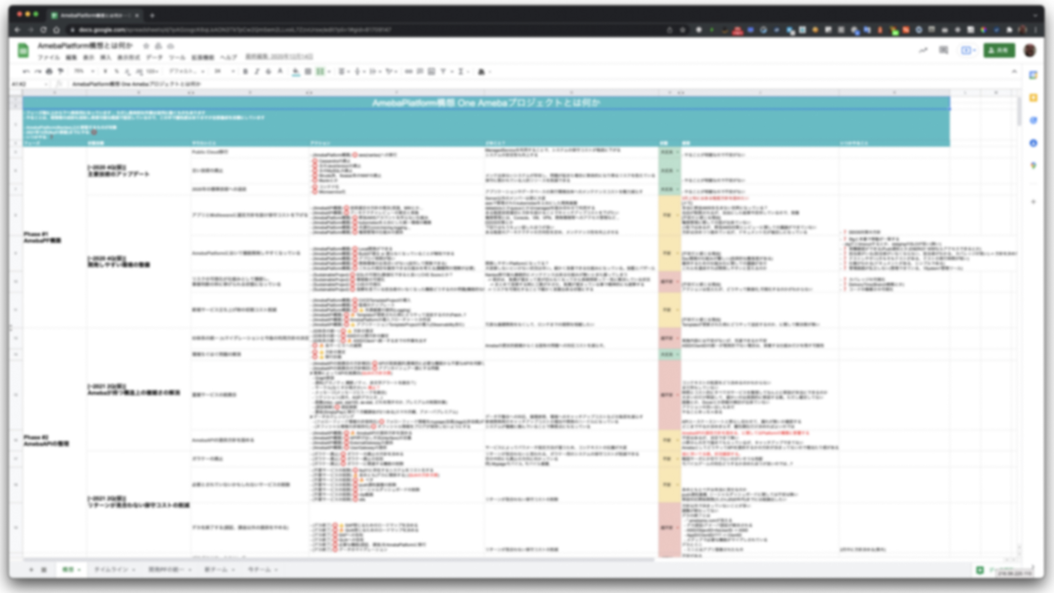
10年開発を気持ちよくスムーズに続けられる状態

事業責任者

PM

エンジニア責任者たち

状態目標のための、成立条件を洗い出すワーク



Amebaが提供したい価値に沿った
サービスとシステムサイズになっている

Amebaとして統合されたアーキテクチャを軸に
システムを再構築できている

負債の可視化が仕組みとして機能し、
事業判断の枠にあげられる状態になっている

Amebaが提供したい価値に沿った

サービスとシステムサイズになっている

Amebaとして統合されたアーキテクチャを軸に

システムを再構築できている

負債の可視化が仕組みとして機能し、

事業判断の枠にあげられる状態になっている

１）刷新プロジェクト前に

統廃合を繰り返し、システム自体をスリム化し

２）残ったシステムが持つAmeba固有の課題を解消し

AWSへフルスクラッチで段階刷新移設

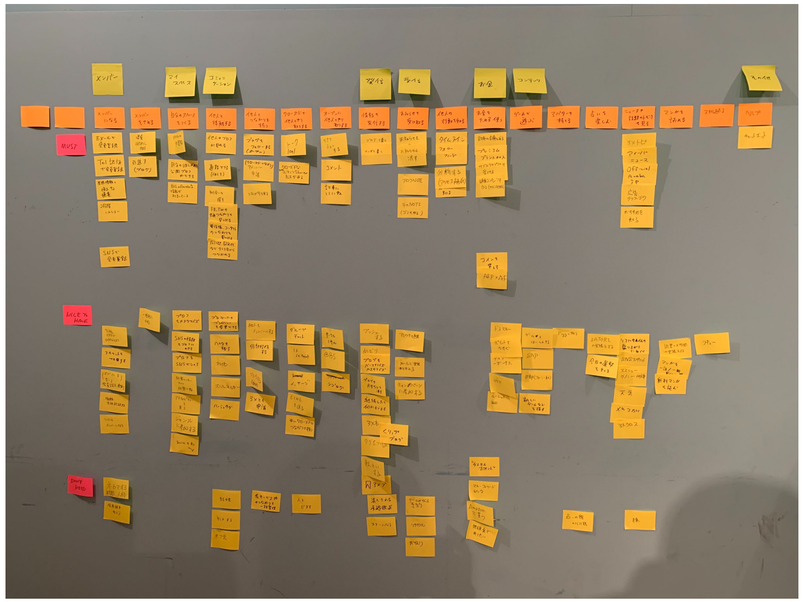
３）刷新成果は

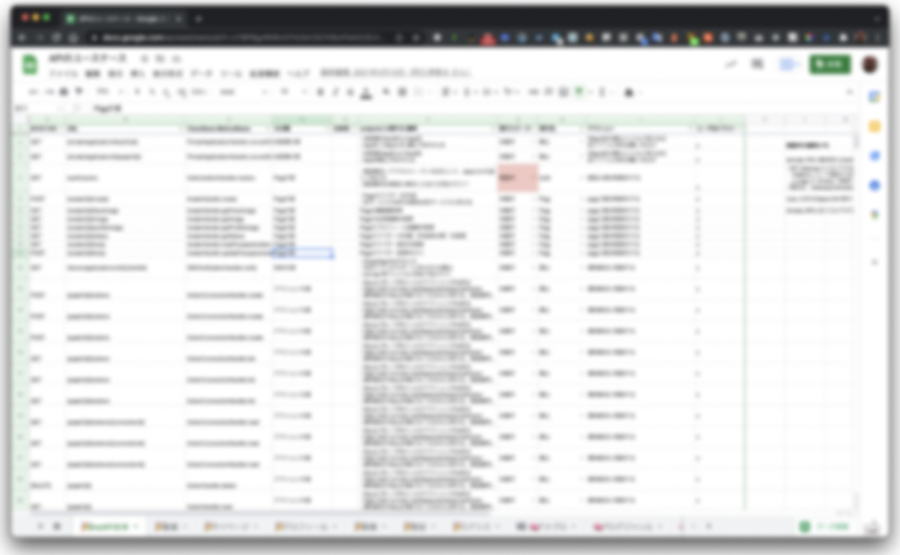
計測可能な指標で判断。指標は経営指標として運用

Amebaが提供したい価値に沿った
サービスとシステムサイズになっている

Amebaとして統合されたアーキテクチャを軸に
システムを再構築できている

負債の可視化が仕組みとして機能し、
事業判断の枠にあげられる状態になっている





Amebaが提供したい価値に沿った

サービスとシステムサイズになっている

Amebaとして統合されたアーキテクチャを軸に

システムを再構築できている

負債の可視化が仕組みとして機能し、

事業判断の枠にあげられる状態になっている

今日はココを

後ほど深ぼります

成立条件の整理 > これらを３軸をMissionとして刷新Projectが動き出すことに > 実行に至るまで

Amebaが提供したい価値に沿った

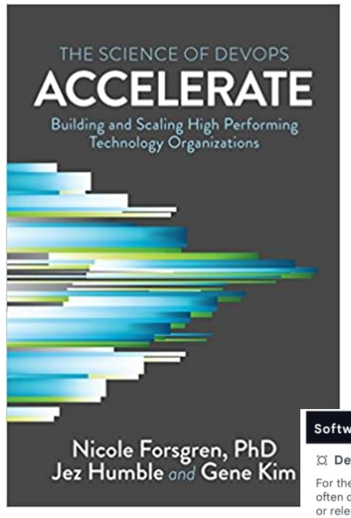
サービスとシステムサイズになっている

Amebaとして統合されたアーキテクチャを軸に

システムを再構築できている

負債の可視化が仕組みとして機能し、

事業判断の枠にあげられる状態になっている



Software delivery performance metric	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months	Fewer than once per six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one hour	Between one day and one week	Between one month and six months	More than six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day	Between one day and one week	More than six months
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0%-15%	16%-30%	16%-30%	16%-30%

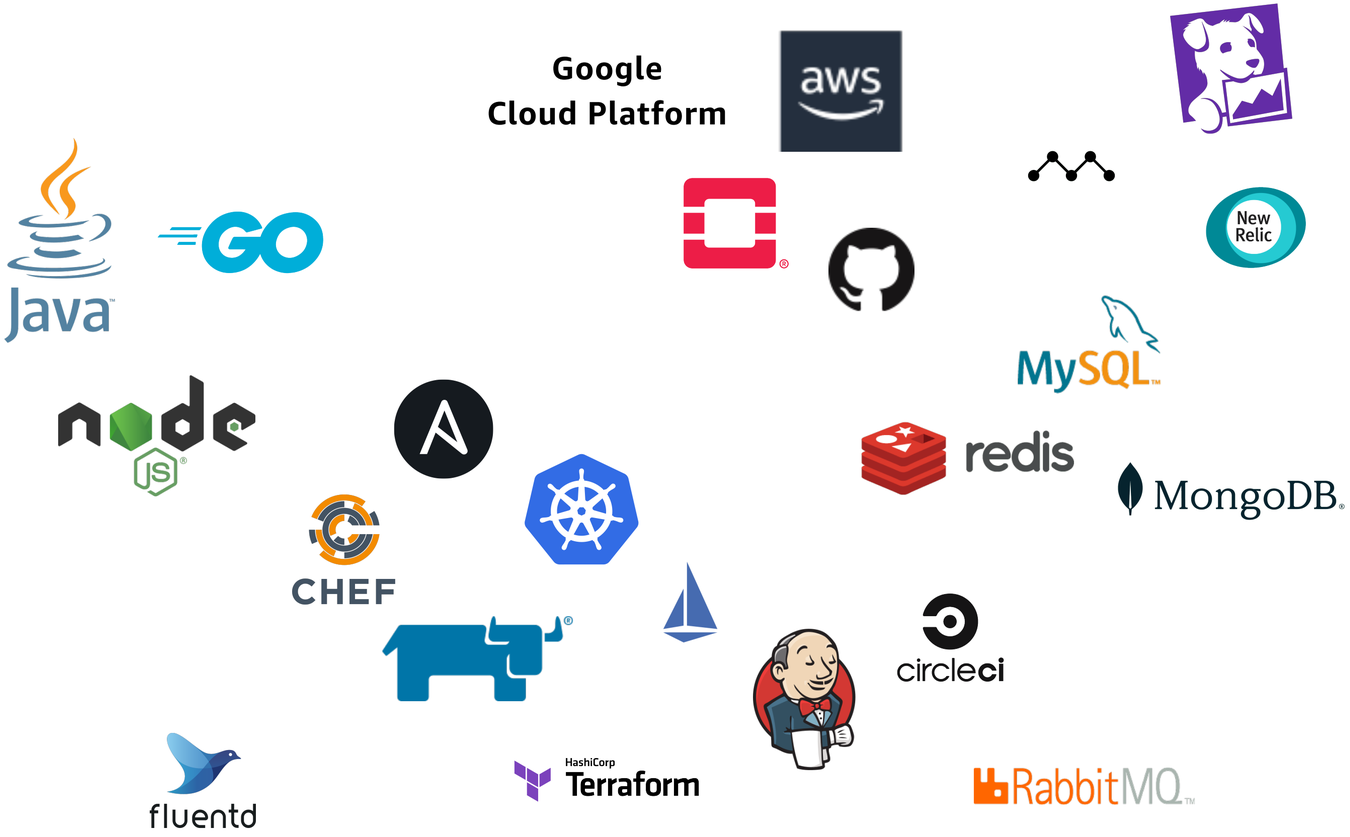
ACTION

それぞれのMissionに対し
どのようにアプローチをしていくか

Mission 2

Amebaとして
統合されたアーキテクチャを軸に
システムを再構築できている

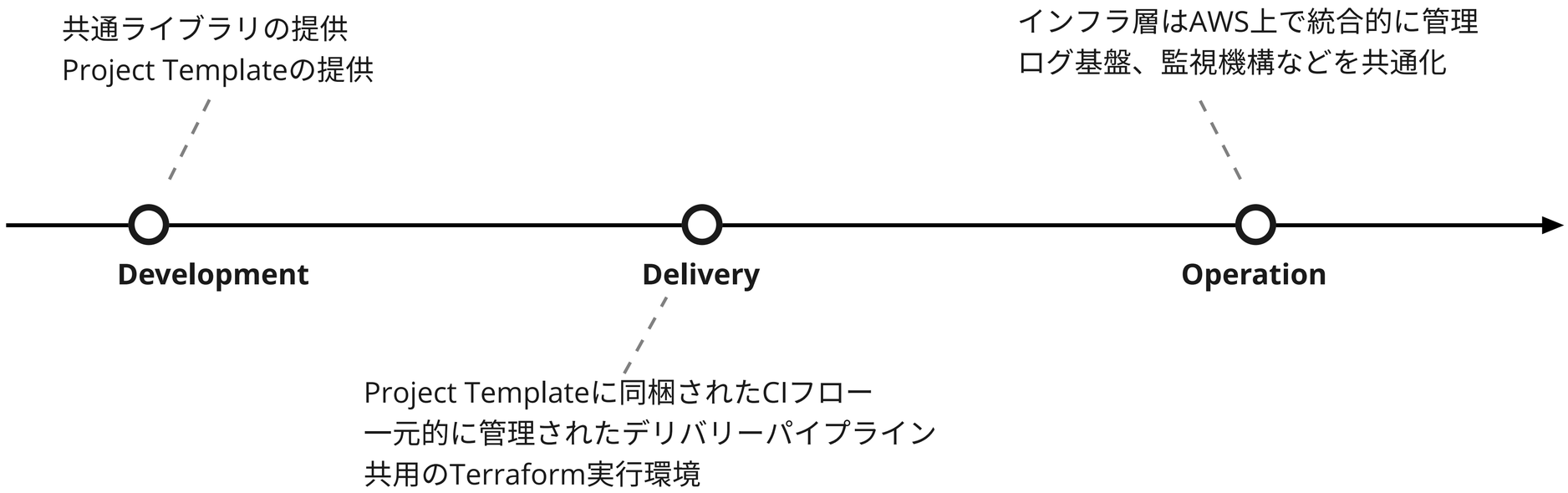
技術選定の煩雑さによる認知負荷



利用する技術を制限

Google
Cloud Platform
Java
GO
node
JS
DATADOG
MySQL
New
Relic
circleci
CHEF
aws
redis
fluentd
RabbitMQ
MongoDB
HashiCorp
Terraform
aws
GO
redis
MySQL
HashiCorp
Terraform
fluentd

開発フローの統合


共通ライブラリの提供
Project Templateの提供
インフラ層はAWS上で統合的に管理
ログ基盤、監視機構などを共通化
Development
Delivery
Operation
Project Templateに同梱されたCIフロー
一元的に管理されたデリバリーパイプライン
共用のTerraform実行環境

実行のアプローチ

- インフラ層をAWS上に統合 = AmebaPlatform
  - AWS選定理由：DBの安定性、Amebaメンバーの習熟度、マネージドサービスの充実度
- 技術選定をあえて制限し、開発フローを共通化
- Amebaシステム固有の問題を解決しつつ、Goで再実装してオンプレからAWSへ移行
- 移行負荷を減らすための共通ライブラリや雛形を提供
- 優先度が低いものは一時的に既存DCで継続

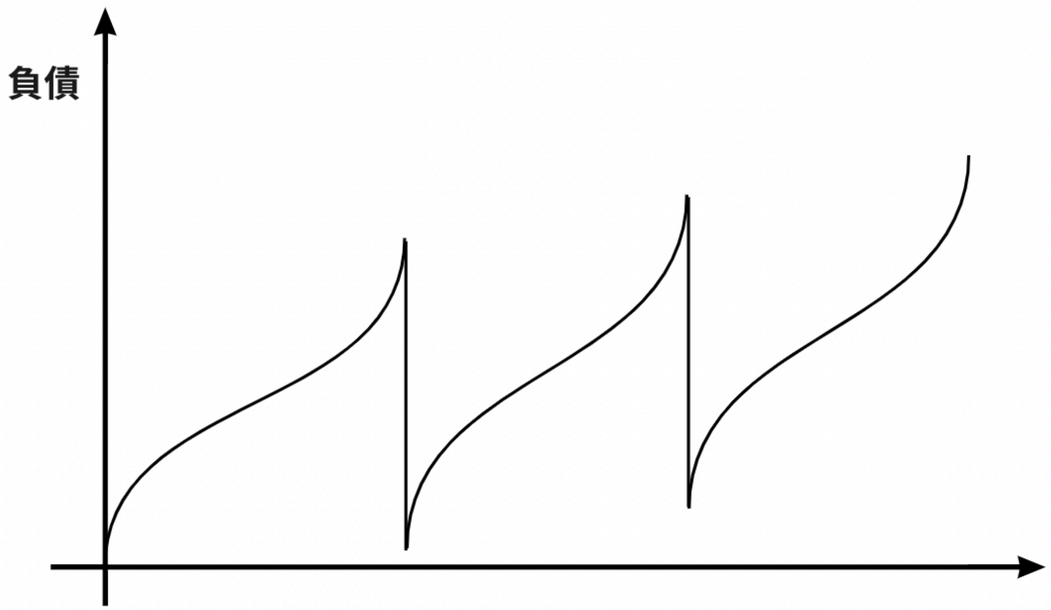
実現されること

- 学習コストと保守負担の軽減
- 開発効率の向上
- 人材の流動性の向上

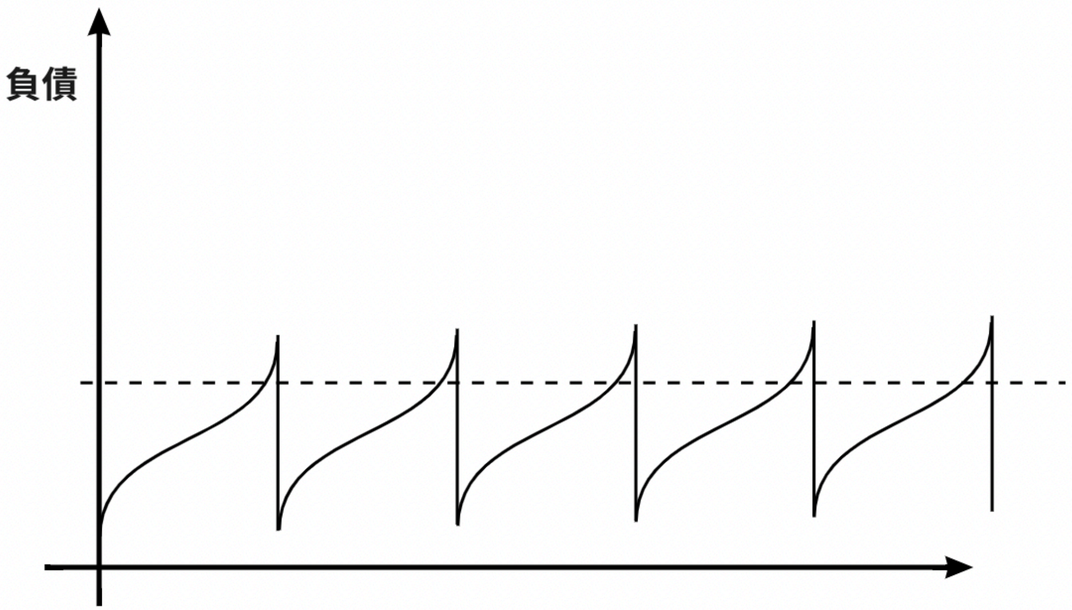
Mission 3

負債の可視化が仕組みとして機能し、
事業判断の枠にあげられる状態になっている

刷新の成功可否
負債の歴史を繰り返さないために
計測可能指標での、合意形成が必要



大規模刷新で負債を返却していく



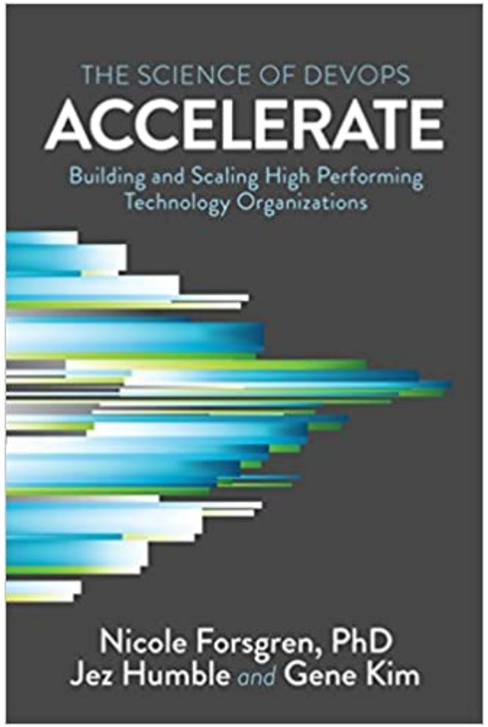
負債を小さく返していく

システムケイパビリティ

- いわゆるシステムのデリバリーパフォーマンスを可視化
- これらの指標が刷新前後で改善されていること、
  また今後も継続的に維持出来る事を目指す
- four keys Project を利用

開発組織ケイパビリティ

- Ameba固有の開発の重さに対してのアプローチ
- バリューストリームマッピングで開発の重さを分析・可視化
  - 刷新で解決したいAmeba固有の認知負荷による工数
  - 運用・開発比率の継続した計測と可視化
- [WIP] asana で継続してボトルネック計測



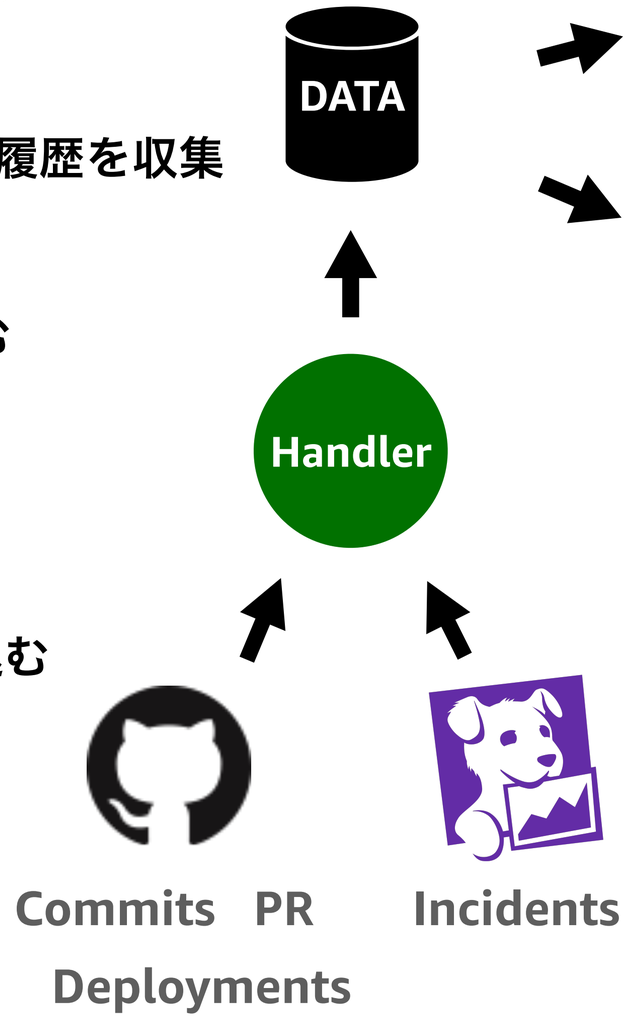
計測対象と、Ameba的定義

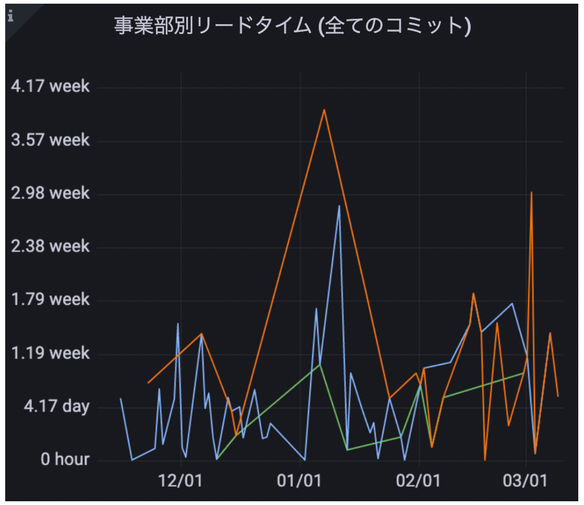
- デプロイ頻度
  - deployの数を取るだけ簡単
- リードタイム
  - デプロイに含まれる全てのコミットの中央値
  - デプロイに含まれる最初のコミットの中央値
- MTTR
  - アプリケーションの復元時間ではなく
  - 監視アラートによる復元時間を測定
- 障害変更率
  - 論文的にそこまで重要視されてなかったので優先度下げ

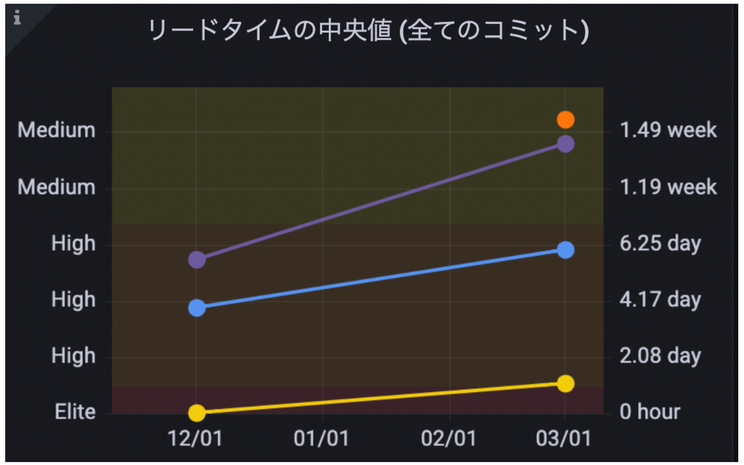
Software delivery performance metric	Elite	High	Medium	Low
Deployment frequency For the primary application or service you work on, how often does your organization deploy code to production or release it to end users?	On-demand (multiple deploys per day)	Between once per week and once per month	Between once per month and once every 6 months	Fewer than once per six months
Lead time for changes For the primary application or service you work on, what is your lead time for changes (i.e., how long does it take to go from code committed to code successfully running in production)?	Less than one hour	Between one day and one week	Between one month and six months	More than six months
Time to restore service For the primary application or service you work on, how long does it generally take to restore service when a service incident or a defect that impacts users occurs (e.g., unplanned outage or service impairment)?	Less than one hour	Less than one day	Between one day and one week	More than six months
Change failure rate For the primary application or service you work on, what percentage of changes to production or released to users result in degraded service (e.g., lead to service impairment or service outage) and subsequently require remediation (e.g., require a hotfix, rollback, fix forward, patch)?	0%-15%	16%-30%	16%-30%	16%-30%

集計システムと、Ameba的計測手法

- デプロイ頻度・リードタイム
  - GitHub Deployments api から commit 履歴を収集
  - GitHub Actions workflowを事前に準備
    - 対象repoに Actions をコピペで仕込む
- MTTR
  - 既存のインシデントフローを
    - Datadog incident 管理に置き換える
  - Datadog API から データを取って投げ込む







システムケイパビリティ

- いわゆるシステムのデリバリーパフォーマンスを可視化
- これらの指標が刷新前後で改善されていること、
  また今後も継続的に維持出来る事を目指す
- four keys Project を利用

開発組織ケイパビリティ

- Ameba固有の開発の重さに対してのアプローチ
- バリューストリームマッピングで開発の重さを分析・可視化
  - 刷新で解決したいAmeba固有の認知負荷による工数
  - 運用・開発比率の継続した計測と可視化
- [WIP] asana で継続してボトルネック計測

まとめ

事業が提供したい価値とシステムのサイズを定め

システムをシンプルにしていく

その後、技術の標準化と共通化を進め

開発しやすいシステムへの移行を実施

結果は、

計測可能指標で振り返る

ACTION

刷新プロジェクトの
実現可能性の精度上げ

AWSへの全移行は巨大な投資

エンジニア目線での理想論だけではなく
経営から見た投資スべき理由の整理が必要

事業計画

コスト

成果

役員

事業責任者

PM

エンジニア
責任者

オンプレDC
責任者

刷新計画、と実現コスト

- 200+ アカウント全ての移行計画とコストの試算
  - 刷新アリナシ、AWSアリナシ、パターン全てで算出
- コスト試算の観点
  - インフラ固定費
    - AWS / GCP / private cloud
  - 刷新途中の開発人件費・運用人件費
    - 移行計画での並行稼動の費用の試算
  - 刷新後の開発人件費・運用人件費
    - オンプレ&AWS特有の運用コストを試算
    - エンジニア全員の稼働実績をチケットベースで集計
      - AWS移行前後でのコスト差の試算

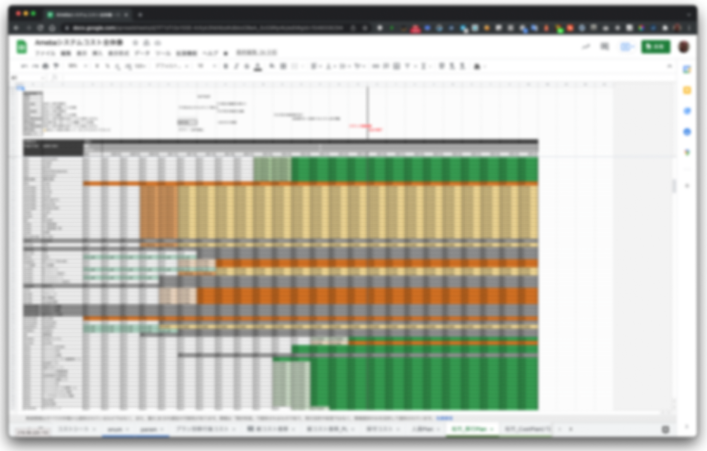
刷新成果、と実現コスト

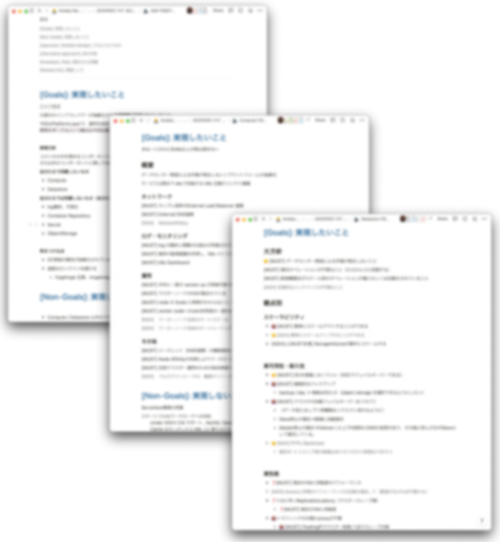
- AWS マネジドメリットをいまのまま実現するコストも試算
  - Amebaが求めるマネジドメリットに対しDesignDocsを記載
    - オンプレエンジニアと協業し、
      - オンプレで自作マネジドを作る場合の実現要件、実現手法、開発＆維持コストを試算
    - Compute、Databaseを中心に議論

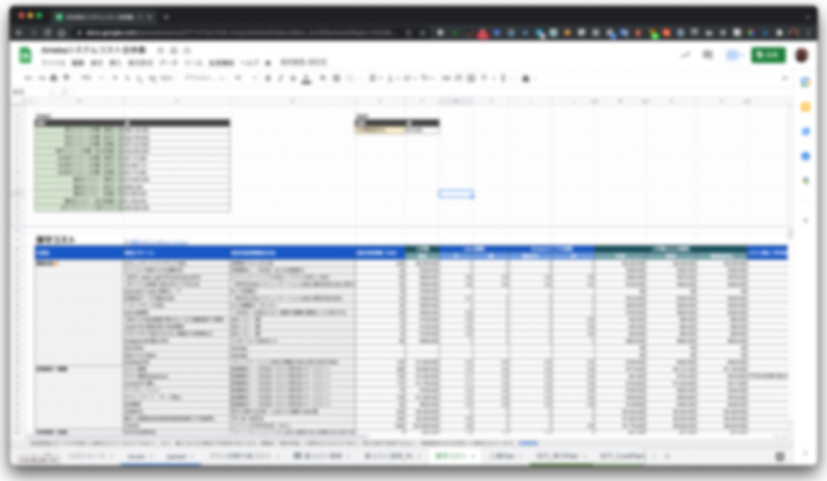
刷新プロジェクト、と事業計画

- 100年愛されるメディアを創る
  - その前提に立った時に、開発しやすいを得るまでの
    - 大規模刷新プロジェクトを数年行うリスク
    - 投資効果を回収出来るまで事業が待てるか
  - それら全てを事業計画に引き直し、最終的にフラットに判断

実行までの壁 > 実施の過程







出てきた事実

- 新規開発より運用保守に割く時間が多いという実態
- AWS移行により保守比率を変えられそうな見通し
- 最小構成の自作マネジドを実現するだけなら
  開発＆維持コスト含めてもオンプレの方が安い

変えたこと

- 刷新成功の結果指標に、開発と運用比率が逆転している、を追加
- 半年に渡る、コスト構造のテコ入れプロジェクトを実施
  - 現行システムのインフラ構成、アプリ構成、開発体制、広く見直し修正
- コスト構造の改革により、
  - 並行運用時期、完全移行後、共にプロジェクト実施前よりもコストを低く抑えられる見通し
  - Amazon EC2 Reserved Instances、Migration Acceleration Programも併用し、
    移行後システムのさらなるコストカットも実施

まとめ

事業から落とした刷新における軸を作り、
ゴール設定は適切な状態目標と計測可能指標で行う

刷新における狙うマトと実行の意思決定まで、
事業責任者と共に歩み、一気に決める

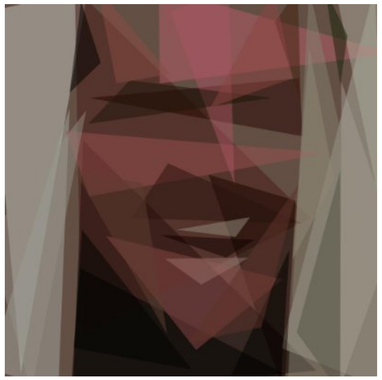
事業計画と並走した刷新計画で、
事業の継続可能性と発展を見据えた打ち手に。

CHAPTER

3

AmebaPlatformの概要

Overview



Engineer / Ameba 技術責任者

岩永 翔

KAKERU IWANAGA

２年ちょっと前からAmebaの技術責任者してます
Web技術とSRE界隈の話題が好きです
好きなたい焼きの味は栗きんとんです

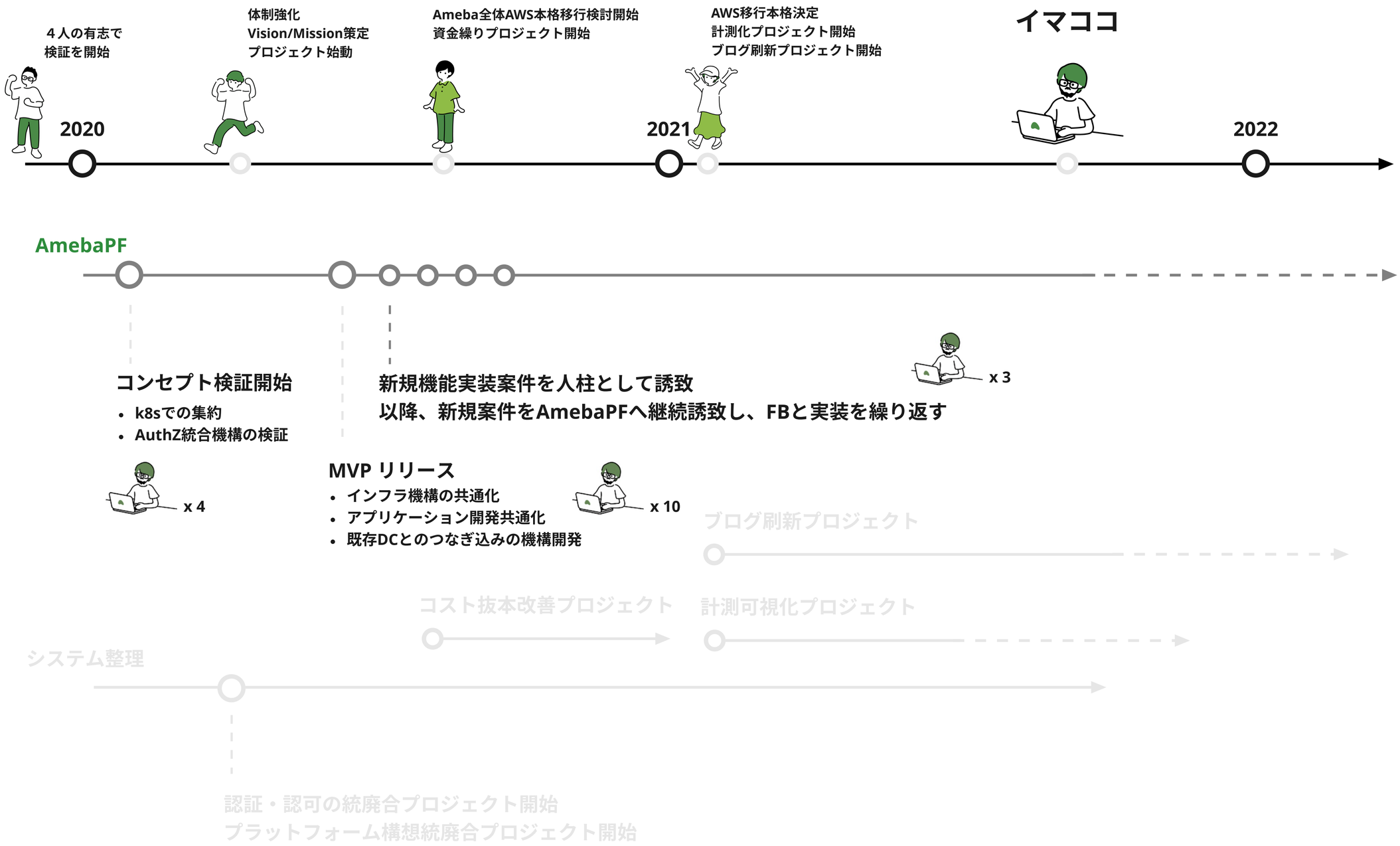
Engineer

難波 和也

KAZUYA NAMBA

Amebaでエンジニアをしています
3児の父です。子供達とLEGOで遊んでます。

AmebaPlatformができるまで


４人の有志で
検証を開始
2020
体制強化
Vision/Mission策定
プロジェクト始動
Ameba全体AWS本格移行検討開始
資金繰りプロジェクト開始
2021
AWS移行本格決定
計測化プロジェクト開始
ブログ刷新プロジェクト開始
イマココ
2022
AmebaPF
コンセプト検証開始
k8sでの集約
AuthZ統合機構の検証
x 4
新規機能実装案件を人柱として誘致
以降、新規案件をAmebaPFへ継続誘致し、FBと実装を繰り返す
x 3
MVP リリース
インフラ機構の共通化
アプリケーション開発共通化
既存DCとのつなぎ込みの機構開発
x 10
ブログ刷新プロジェクト
コスト抜本改善プロジェクト
計測可視化プロジェクト
システム整理
認証・認可の統廃合プロジェクト開始
プラットフォーム構想統廃合プロジェクト開始

AmebaPlatformで
実現したかったこと

AmebaPlatformの事例で紹介する利用AWSサービス

- Amazon Elastic Kubernetes Service
- Amazon Kinesis Data Streams
- Amazon Aurora
- Amazon OpenSearch Service
- Amazon ElastiCache
- Amazon Simple Storage Service
- Amazon Simple Queue Service
- Amazon Simple Notification Service
- AWS Lambda
- AWS Secrets Manager

AmebaPlatformで実現したかったこと

- 環境の分離
- 負荷試験環境の提供
- IaCの実践
- オンプレとクラウド間の接続
- マイクロサービス間の認可
- スターターセットの提供
- CI/CDの統合
- ログ基盤の提供
- 監視の一元化
- セキュリティの担保

環境の分離

- 開発効率を重視し、必要最低限の分離に

シングルアカウント (= NOT マルチアカウント)

- 開発体験としてのシンプルさを優先
- ServiceQuotaのHardLimitは事前に要検討

マルチテナント (=NOT シングルテナント)

- クラスタ管理コストを最小化
- 障害時の影響範囲が大きくなる事を許容できるか

- 共用環境、負荷試験環境はアカウントを分割

負荷試験環境の提供

実現したいこと

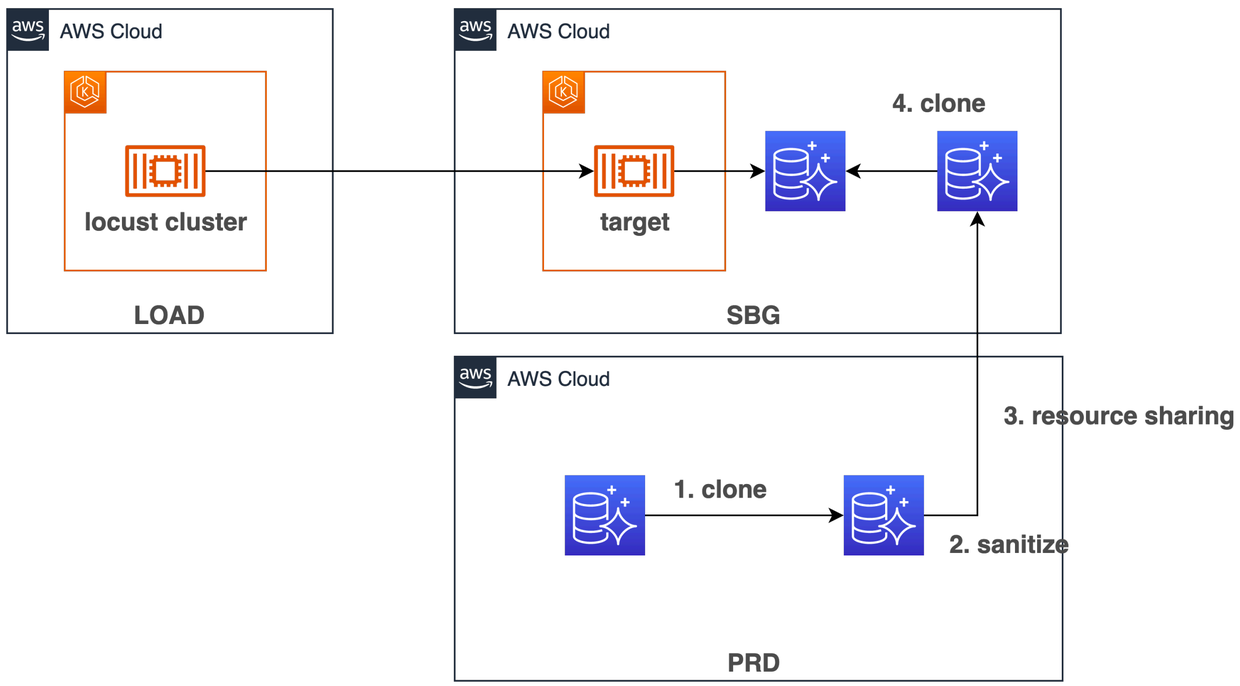
- 独立した環境に、本番同等の環境を再現
- 現実的な費用での運用

必要なリソースに限定して構築

- 必要なリソースを試験毎に構築
  - terraformとkubernetesを利用して実現

本番環境のデータを利用する仕組み

- Amazon Aurora の Cloning機能 を利用した高速な複製
- Resource Access Manager による安全なリソース共有

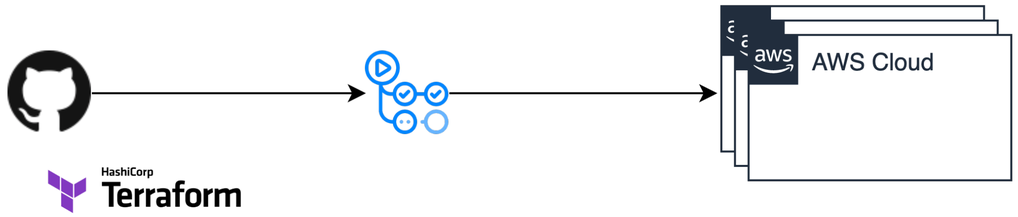


IaCの実践

リソースの一元管理、野良リソースの抑止

- AWS上のリソースはほぼ全てTerraformで管理
- 単一リポジトリで全てのリソースを定義
  - 共通基盤とMS毎にtfstateを分割
  - 個別にCODE_OWNERSを設定
  - Moduleに関しては別リポジトリで管理
- Github Actions で実行
  - self-hosted runner のmyshoesを利用
  - [参考: CyberAgent における OSS の CI/CD 基盤開発]

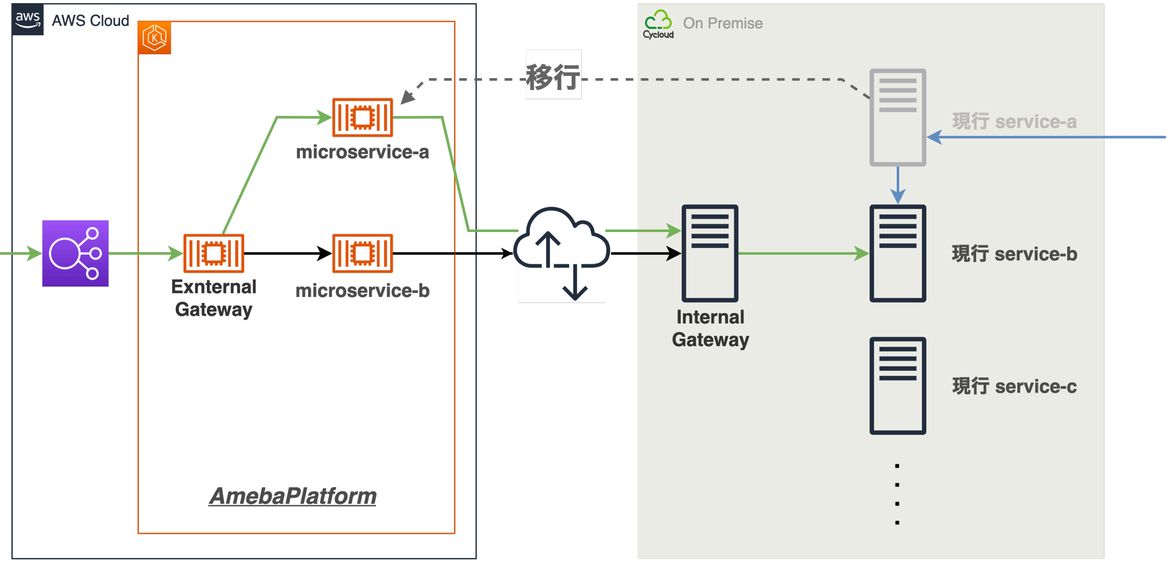
```
envs
├── develop
│   ├── platform
│   │   └── sample.tf
│   │       ⋮
│   └── services
│       ├── microservice-a
│       │   └── sample.tf
│       │       ⋮
│       └── microservice-b
│           └── sample.tf
│               ⋮
└── production
    ⋮
```



オンプレとクラウド間の接続 > 方針

システム移行中に実現したい課題

- 1）オンプレへのセキュアな通信機構
- 2）AWS移行後システムへのセキュアな通信機構
  - どちらも、IP制限の運用はしたくない
  - PF上での認証認可の仕組みを利用できればよりGood
- DXは以下の観点で選択肢から除外
  - SLAがない
  - CA全社で専用線を共有し負荷が課題
  - ただし、DBのMigrationには利用



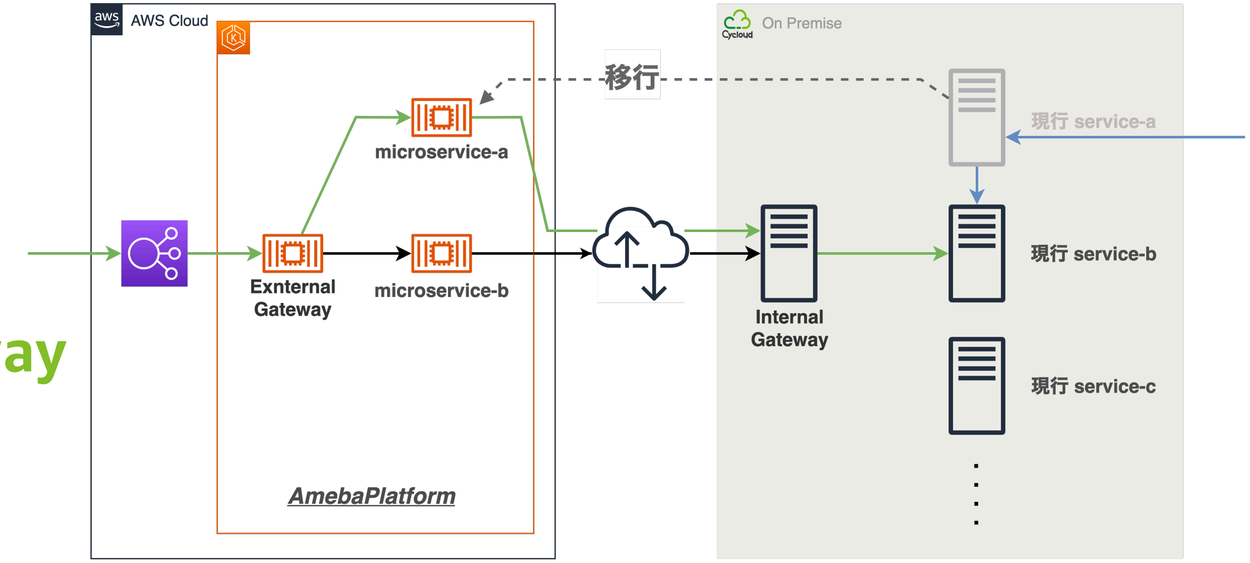
オンプレとクラウド間の接続 > 実装

AWS to オンプレの通信 / Internal Gateway

- mTLS + ClientID/Secret ベースの認証
- ClientIDベースのアクセス制御
- nginx (mTLS) + go 自前 proxy (認証, AC)

オンプレ to AWSの通信 / External Gateway

- ClientID/Secret ベースの認証
- ClientIDベースのアクセス制御
- HTTP/gRPCの変換
- .protoは利用せずreflectionで動的に変換



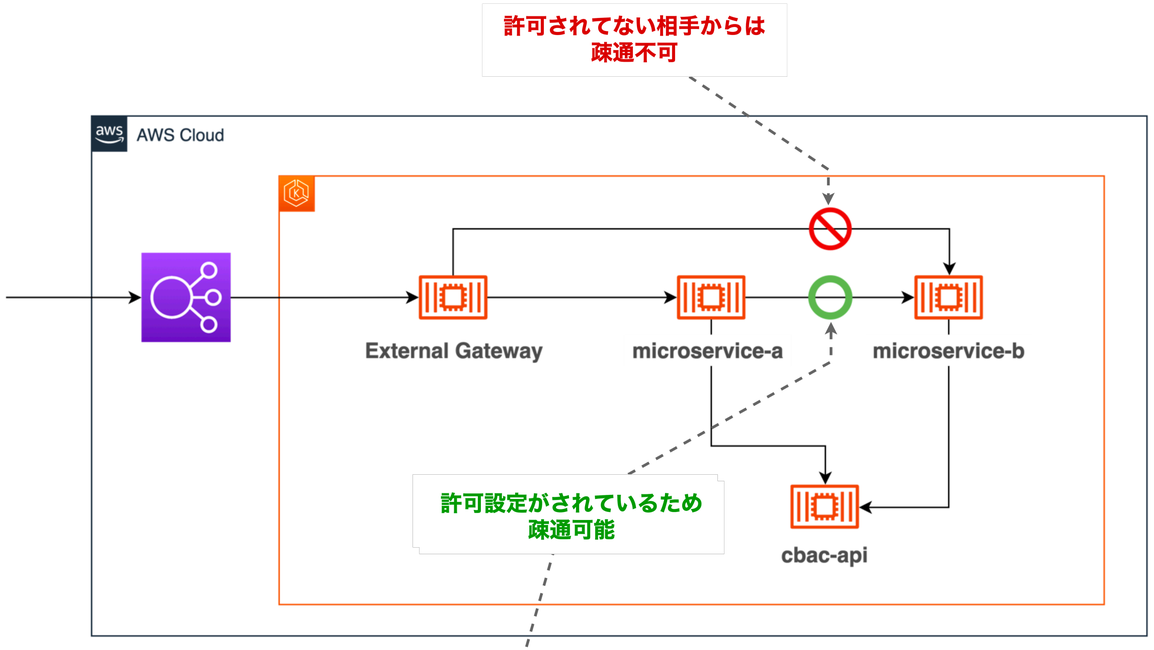
マイクロサービス間の認可

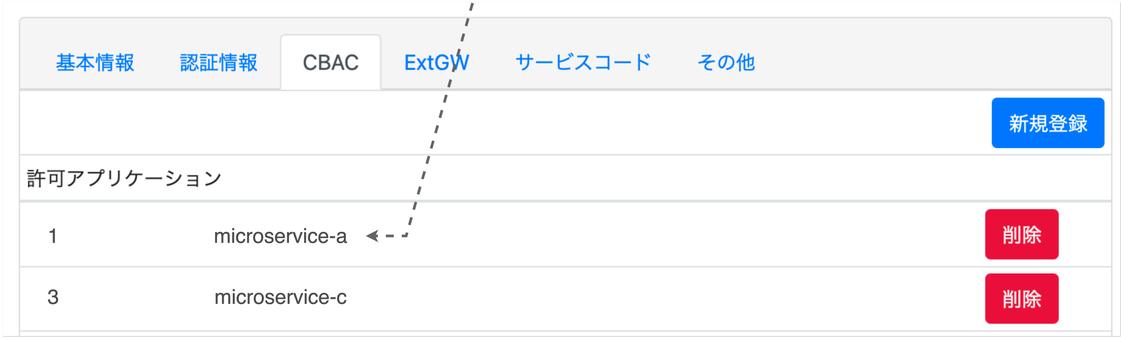
マルチテナントな構成における課題

- 意図しない経路による通信の抑止
  - 互いの理解不足から発生する相互依存
  - 無意味な経路を迂回した参照、など…

ClientIDを利用した認可の仕組み

- CBAC(Context Based Access Control)の提供
- 許可されたClientIDにのみアクセスを許可
  - 管理ツールから許可設定を登録
- 共通ライブラリでInterceptorを提供





スターターセットの提供

共通ライブラリの提供

- Go(libgo)とNodejs(libjs)を提供
- 共通化することでライブラリのVersionへの追従を統合的に管理
  - Dependabotの運用
- HTTP/gRPC server, client, logger, metrics, tracing, etc...

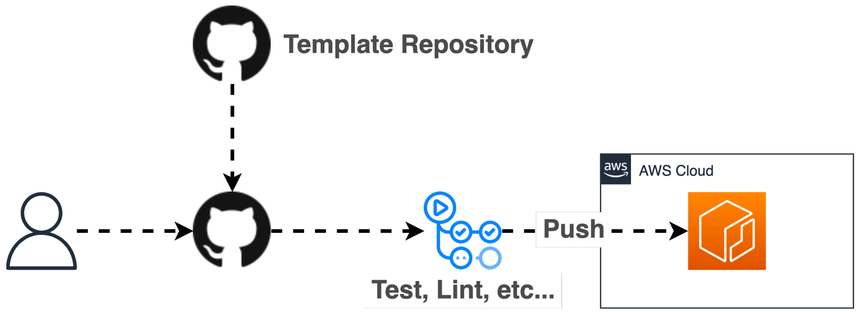
Project Template の提供

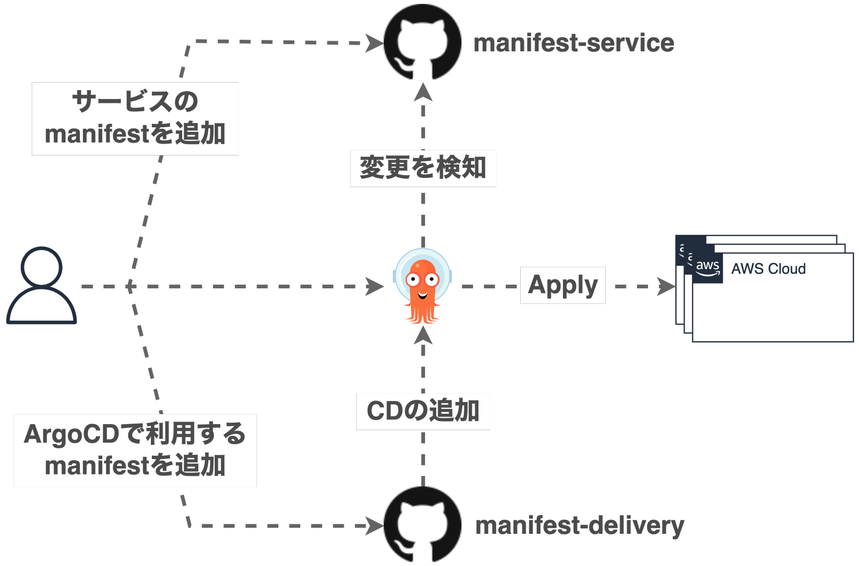
- GithubのTemplate Repository を利用して新規にMSを作成する際の雛形を提供
- Local Development, CI, HTTP/gRPC Server, Configuration, etc...

CI/CDの統合

統一されたデリバリーパイプラインの提供

- CI: Github Actions
  - Project Templateでベースを提供
  - Test、Lint、Container Image Build & Push、etc...
- CD: ArgoCD
  - manifestは一元管理
    - manifestとclusterの状態を一致させる(gitops)
  - fluxcdを利用したimage tagの自動更新
  - kubevelaを利用して効率化
    - [参考: シングルクラスターマルチテナンシーを目指しているEKS上でkubevelaの運用をしてみた]

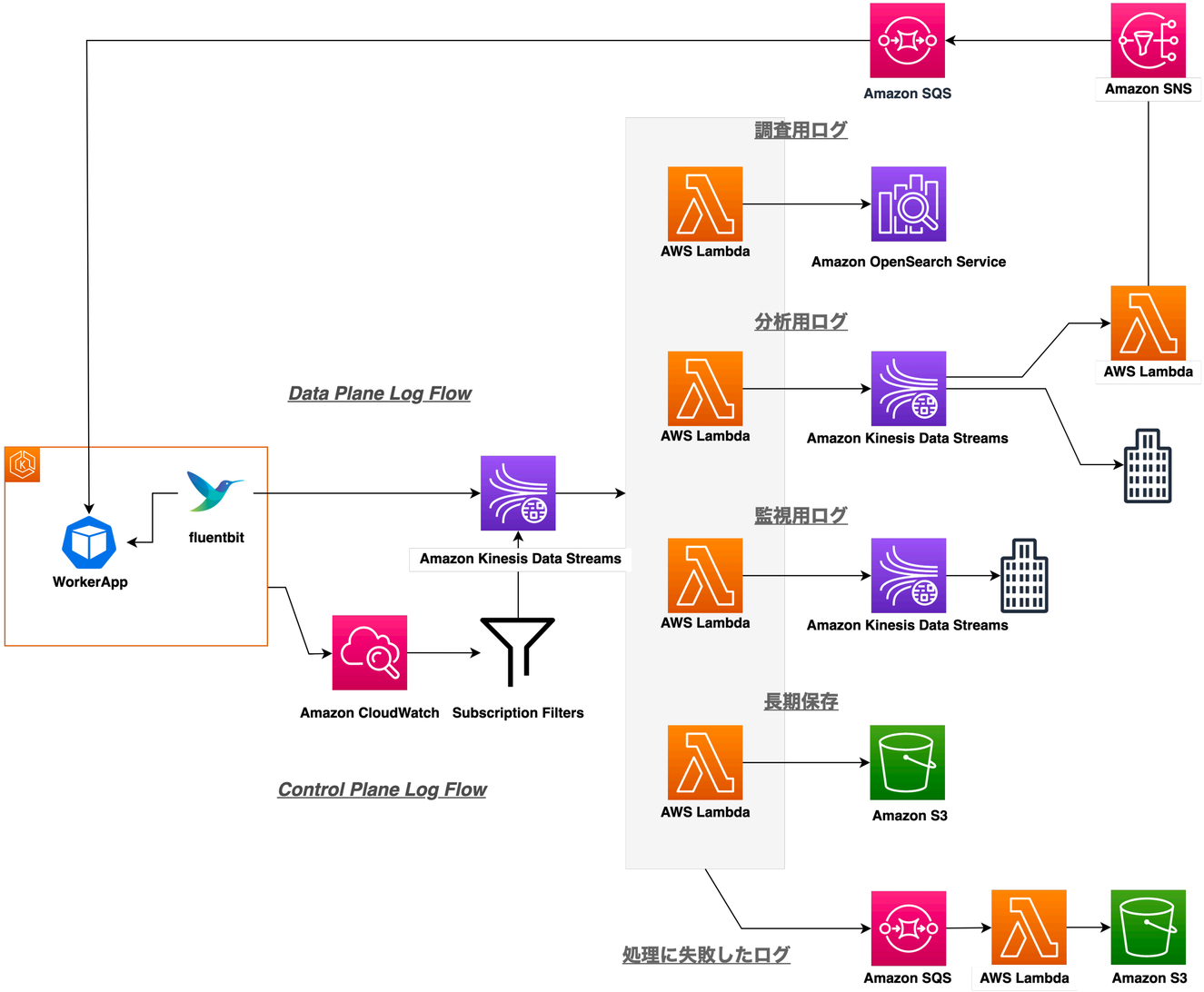




ログ基盤の提供

Amazon EKS のログを集約

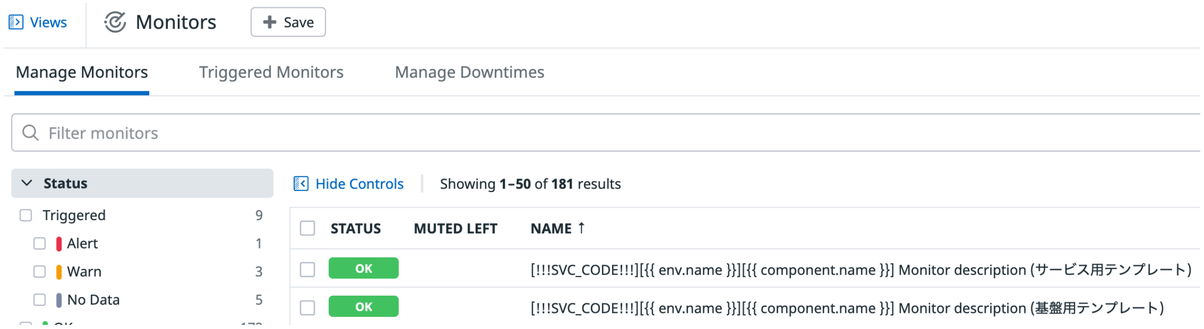
- 複数の配信先への対応
  - Amazon KDS を介して複数の経路に分割
- 共通ライブラリでLoggerを提供
  - 共通のLogFormatを定義
  - 標準出力に吐かれたログを収集
- コストの最適化
  - Sampling, Amazon S3 LifeCycle, etc...

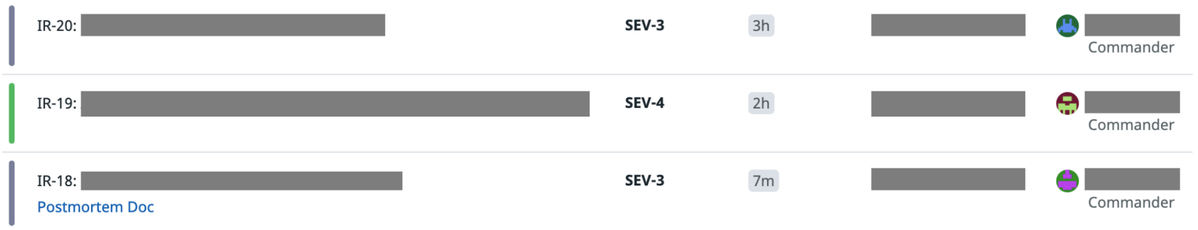


監視の一元化

Datadog で監視を一元化

- AWS Resources と Application Metrics
- Dashboard, Monitoring, APM など
- 共通ライブラリでmetricsの出力機構を提供
- kubevelaでannotationを定義しmetricsを調整
- Monitorの命名規則、通知内容の見直しなど、継続的な改善
- Datadog Incidentと Postmortem を利用した障害の計測、可視化

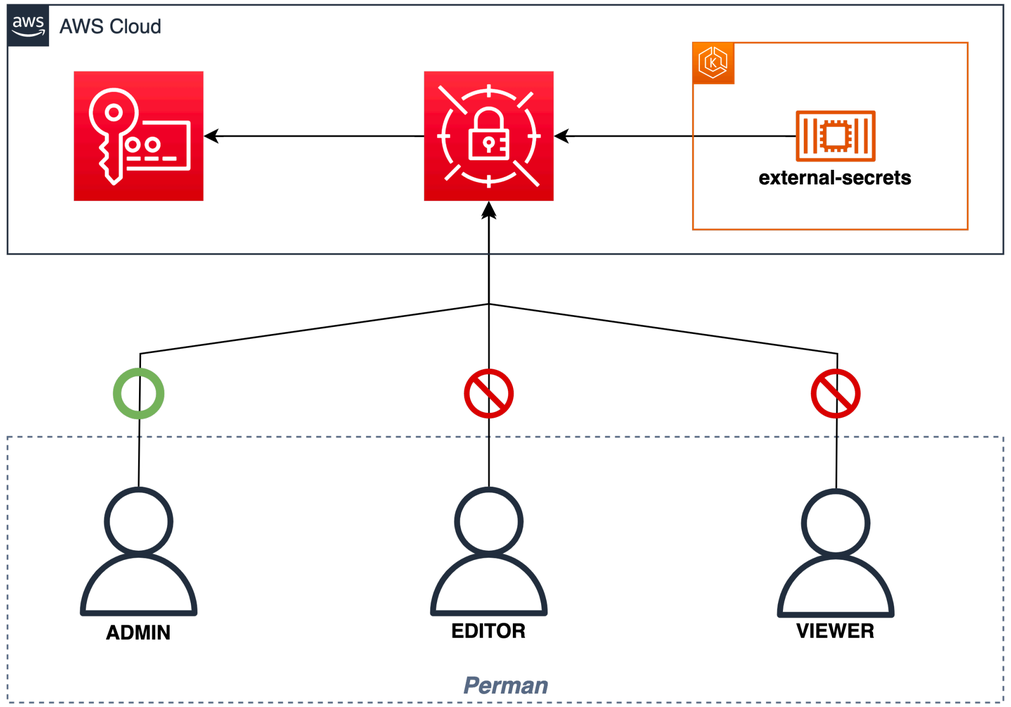




セキュリティの担保

開発効率とセキュリティの両立

- 社内横断で導入されている予防/発見的ガードレール
  - 参考: 600アカウントのセキュリティを見る
- 権限管理に社内ID基盤と連携してRoleを管理
  - 参考: 認証/認可基盤PERMANの紹介
- kubevelaでSecurityContextを一律設定
- SecretsManager + external-secretsでCredentialを管理
- MITRE等を参考に潜在的なリスク、脅威を整理している



移行を進めてみて

失敗したこと、苦労していること

考慮漏れ等によるトラブル

- 大量のログファイルが格納されたS3 BucketにLifeCycleを設定した瞬間コストが急増
- 移行システムが想定より増えたことでIP枯渇の懸念が発生しVPCを再作成

マルチテナント等の要素による難易度

- ブランチ戦略、マルチライン開発など、理想を求めるとどれも非常に難しい

良かったこと

定量的な成果

- 実際に27サービスがAmebaPFに移行、23が本番稼働開始
- Four keys指標リードタイムで７倍の改善

定性的な成果

- 新規案件をAmebaPF上で実装、効率よく開発できたとのFB
- 長年の負債は着実に解消している

CHAPTER

4

まとめ

Summary

まとめ

- Amebaの刷新にいたる背景
  - 事業とシステムサイズの乖離
  - サービス固有の問題や技術要素による認知負荷と開発力の低下
- 刷新PJTの立て付けと進め方
  - 事業計画と並走した刷新計画
  - 適切な状態目標と計測可能指標を伴うゴール設計
- AmebaPlatformの概要
  - 目指したもの、どう実現したか

これから (1/2)

刷新でFour keys指標に改善は見られたものの、

HighPerformerなので改善の余地あり。

計測カバレッジも70/800repoなのでこれから📈👩‍💻

もう一つの指標である運用工数の比率を下げるも、

システム並行稼動が多くまだまだこれから💪

これから (2/2)

ブログ本体の刷新は始まったばかり。

これから１年半かけて移行していく

残りのAmebaのサブシステム郡も、

統廃合を繰り返しAmebaPF上に作り直していく。

AmebaPFはまだまだ最低限の機能

刷新PJと並走しProductivityに向き合い続ける🦮

100年愛されるメディアを創る、を
システムから後押しする

We Are Hiring!!

Thank you!

岩永 翔
株式会社サイバーエージェント
メディア管轄 Ameba事業本部 CTO

難波 和也
株式会社サイバーエージェント
メディア管轄 Ameba事業本部 Engineer

© 2022, Amazon Web Services, Inc. or its affiliates. All rights reserved.

本セッション資料や記載内容については一切の転用を禁止しております