



# 失敗しないアプリケーション モダナイゼーションの考え方

Google Cloud 北瀬 公彦、村上 大河

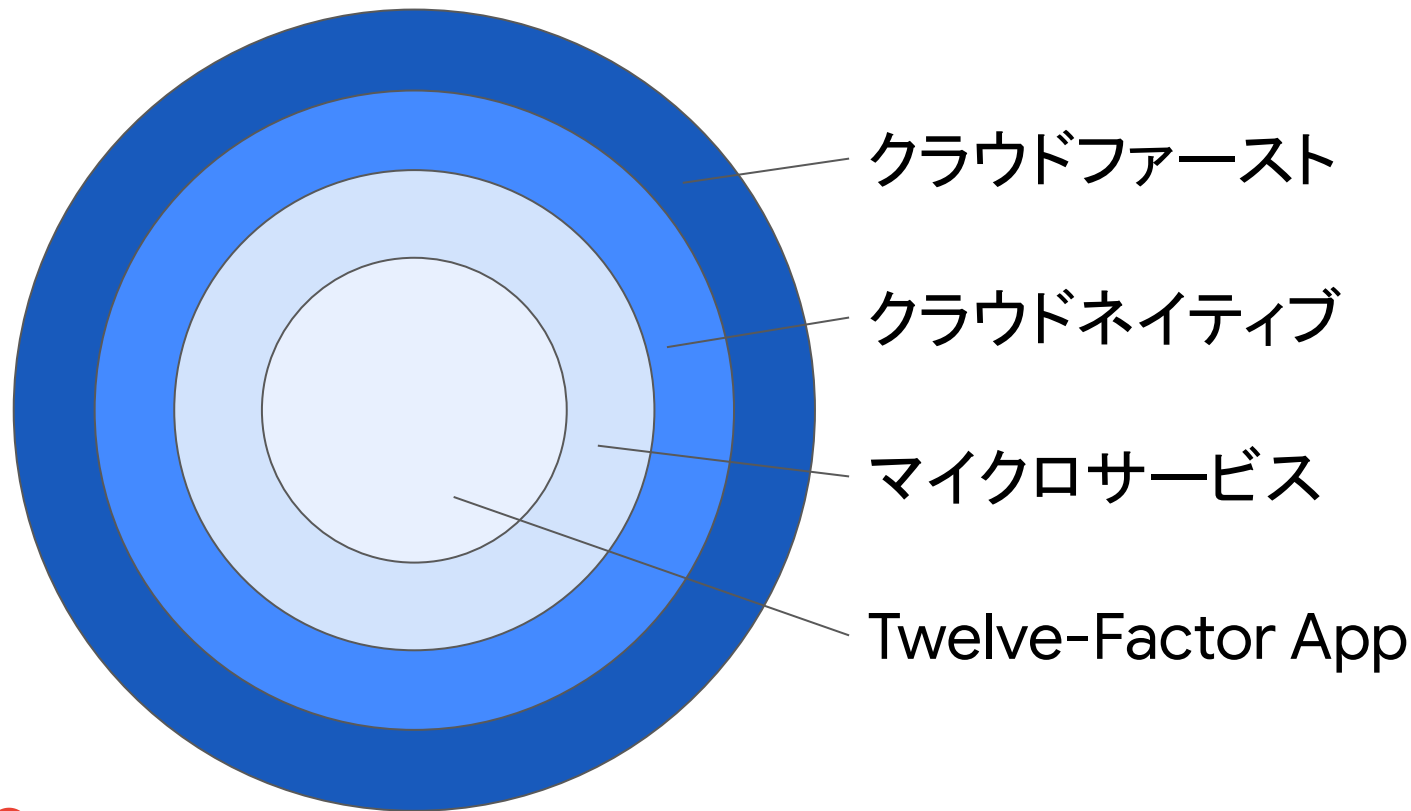
株式会社フリークアウト執行役員 CTO 西口 次郎 様

富士フイルムソフトウェア株式会社

ソフトウェア技術本部 基盤技術グループ 主任研究員

ムサヴィジャハン アバディ セイド モハマド 様

# モダナイゼーション



# Twelve-Factor App

“セットアップ自動化のために 宣言的なフォーマットを使い、プロジェクトに新しく加わった開発者が要する時間とコストを最小化する。

下層のOSへの依存関係を明確化し、実行環境間での移植性を最大化する。

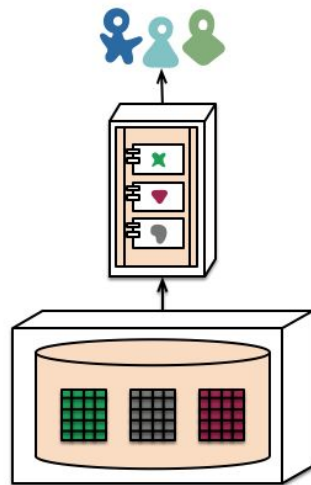
モダンなクラウドプラットフォーム上へのデプロイに適しており、サーバー管理やシステム管理を不要なものにする。

開発環境と本番環境の 差異を最小限にし、アジリティを最大化する継続的デプロイを可能にする。

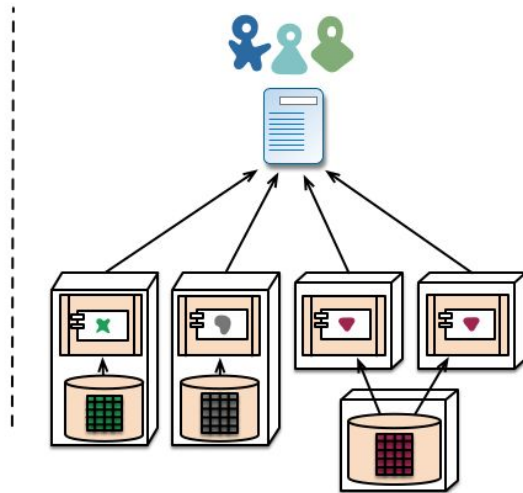
ツール、アーキテクチャ、開発プラクティスを大幅に変更することなくスケールアップできる。”

# マイクロサービス

“マイクロサービスは、一つのアプリケーションを小さなサービスを組み合わせて開発するアーキテクチャスタイルである。各サービスはそれぞれのプロセス内で実行され、軽量なメカニズムを使って通信する。HTTP Resource APIを使うことが多い。”



monolith - single database



microservices - application databases

# クラウドネイティブ

“クラウドネイティブ技術は、パブリッククラウド、プライベートクラウド、ハイブリッドクラウドなどの近代的でダイナミックな環境において、スケーラブルなアプリケーションを構築および実行するための能力を組織にもたらします。このアプローチの代表例に、コンテナ、サービスメッシュ、マイクロサービス、イミューダブルインフラストラクチャ、および宣言型APIがあります。

これらの手法により、回復性、管理力、および可観測性のある疎結合システムが実現します。これらを堅牢な自動化と組み合わせることで、エンジニアはインパクトのある変更を最小限の労力で頻繁かつ予測どおりに行うことができます。”



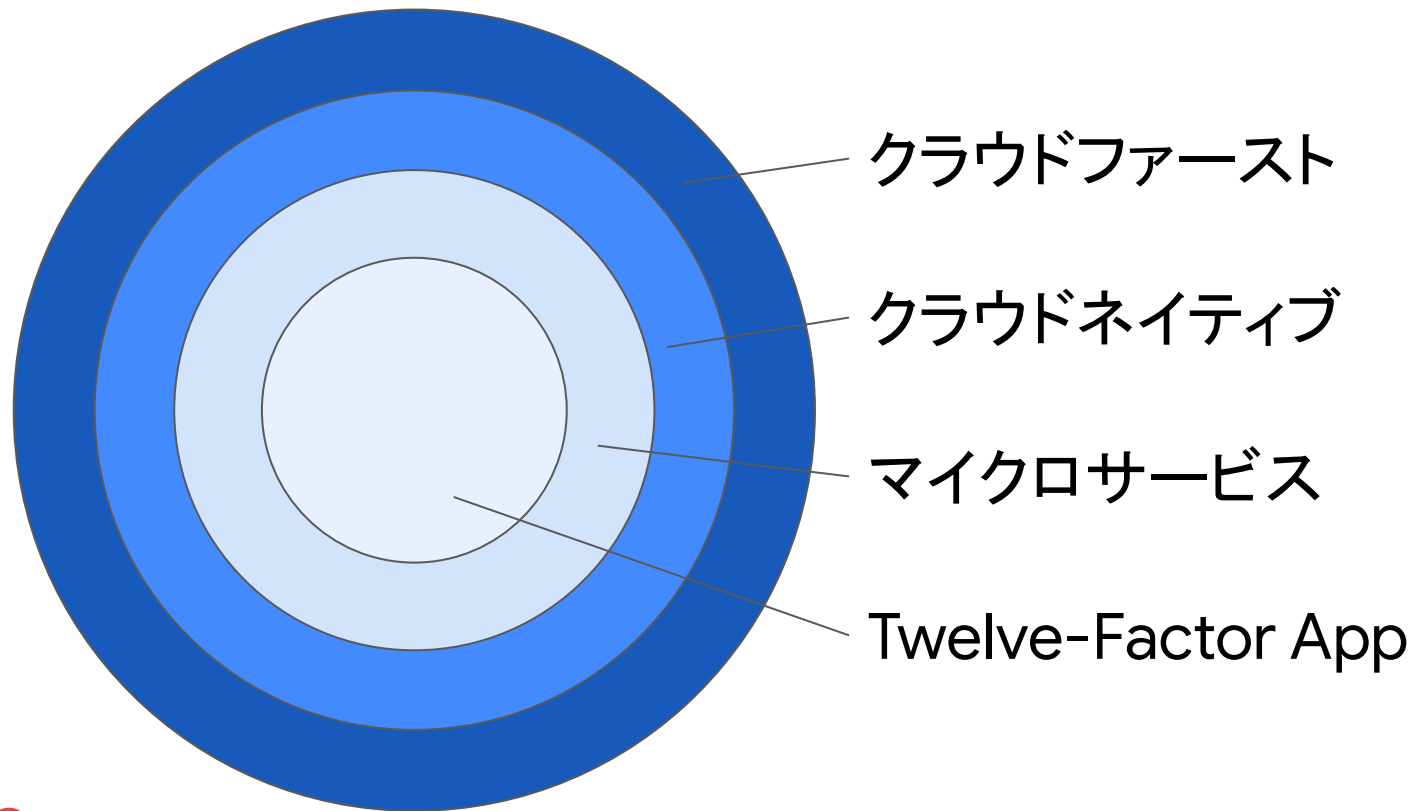
Microservices

Container

CI/CD

DevOps

# モダナイゼーション



Application discovery

Application assessment

Performance requirement

Network sensitivity

Application dependency

Data requirement

Visualization support

Shared infrastructure

App-to-migration path mapping

Retire

Retain

Rehost

Replatform

Refactor

Replace

Rebuild

Rehost

↓ コスト



Server



VM migration



Compute Engine



Database



VM migration



Compute Engine

Replatform

↓ コスト

↑ スケーラビリティ



Server



VM migration



Compute Engine



Database



DB migration



Cloud Storage



Cloud Spanner

Refactor

↓ コスト

↑ スケーラビリティ

↑ 開発速度



Server



Code refactor to microservice



Microservice



Kubernetes Engine



Container Registry



Cloud Build



Database



DB/Schema Migration



Cloud Storage



Cloud Spanner



Stackdriver




BigQuery

Hipster Shop

Guest

frontend-external-static.default.svc.hipstershop.us-central1-b.microservices-demo-app.apps.kubernetes.tips/product/1YMWWN1N4O

USD ▾View Cart (0)



## Home Barista Kit


USD 124.00

**Product Description:**  
Always wanted to brew coffee with Chemex and Aeropress at home?


Quantity2 ▾

Add to Cart


Products you might like




Vintage Record Player



Terrarium



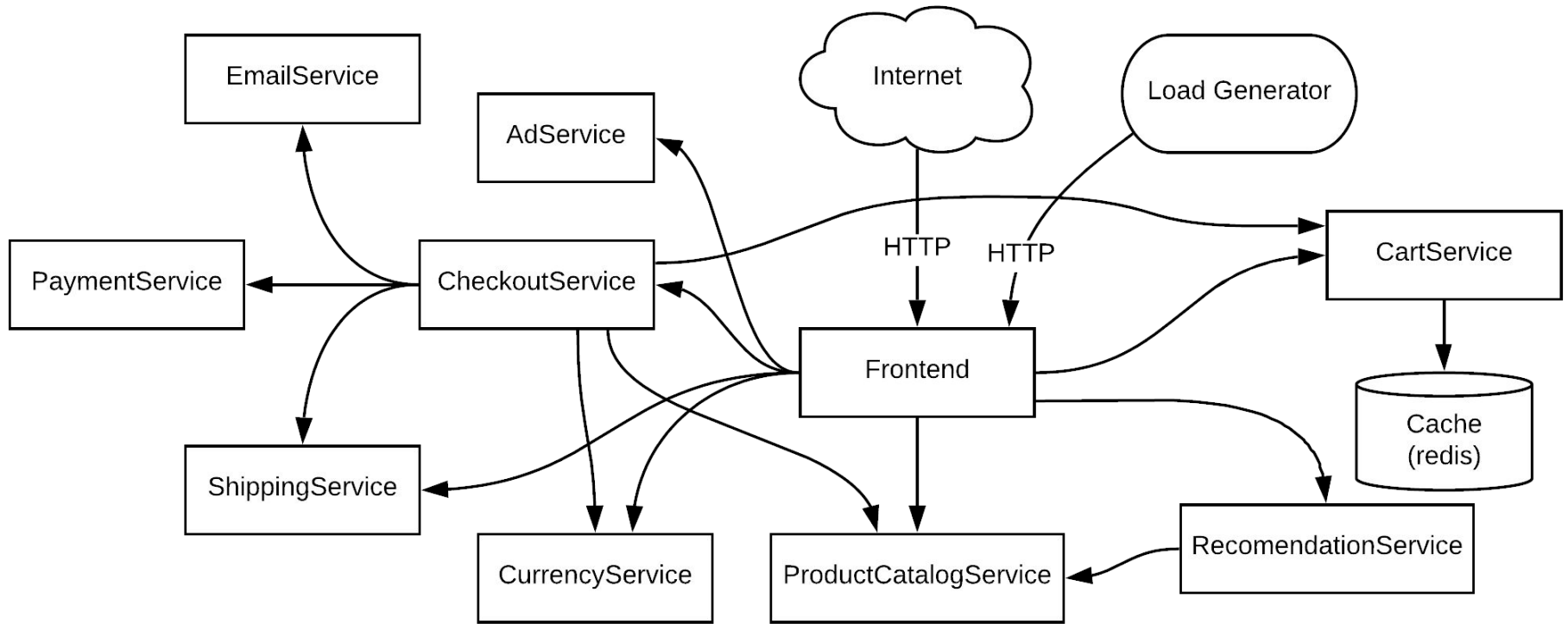
City Bike



Vintage Camera Lens

Waiting for frontend-external-static.default.svc.hipstershop.us-central1-b.microservices-d







# ユーザー事例紹介



西口 次郎 様

株式会社フリークアウト  
執行役員 CTO



ムサヴィジャハン アバディ セイド モハマド 様

富士フイルムソフトウェア株式会社  
ソフトウェア技術本部 基盤技術グループ 主任研究員

# 自己紹介、 サービスの紹介



# FUJIFILM Prints & Gifts

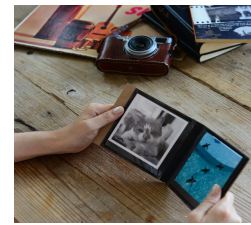
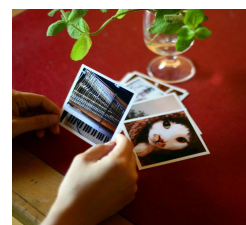
富士フィルムのプリントや写真雑貨を注文出来るサービス

<https://pg-ja.fujifilm.com/home>

バックエンドの注文管理システム(以降 OMS )を開発



## 商品一例



# FreakOut - Red for Publishers



- 西口 次郎
- 株式会社フリークアウト 執行役員 CTO
- 2017/07 Red for Publishers をローンチ
  - プレミアムメディア向けアドプラットフォーム
  - 自社アドネットワークのシステム基盤としても使用
    - Poets (日本)  poets
    - FreakOut Native (海外 10 カ国)  **FreakOut**  
Native
  - ほぼすべてを GCP で構築 ([Google による事例紹介](#))
  - [2019/03/26 Google Cloud Kubernetes Day](#) での発表



# アーキテクチャ、 DevOps の導入 について



# OMS モダナイゼーション

## 課題: レガシーシステム(運用 10年 超え)のモダナイゼーション(リプレース)

- 負荷量変動への対応(季節イベント、キャンペーン)⇒ スケーラビリティの確保
- モノリシックアプリの課題:
  - 品質担保の難しさ ⇒ 要員学習コスト ⇒ 保守・運用コストの増大
  - 頻繁なコード修正やシステム構成変更コスト ⇒ 機能改善スピードの向上

## アーキテクチャ: マイクロサービス

- 実現方法: コンテナ(Docker)、コンテナ管理(Kubernetes)
- 実行環境: クラウド上の Kubernetes マネージドサービス(学習・構築・運用の費用削減)
- クラウドベンダ: GCP(当時、GCP のみが Kubernetes マネージドサービスを提供。GKE 3年実績)

## 効果:

- スケーラビルの確保: 出来た。他国展開等で横展開しやすい構造。導入後ダウンタイム無し
- 保守・運用コストの増大: ランニングコスト減(約 3/5)、機能搭載費用約 1/2
- 機能改善スピードの向上: 影響範囲が見極めやすくなり、対応速度が向上した(約 2 倍)

## 残課題(対応中・検討中):

- DevOps 導入、サービスメッシュによる管理性 & 可観測性の向上、サービス内横展開

# Red for Publishers アーキテクチャ



- GKE (2019/09 現在 1.13.7) を使用
- すべてのアプリケーションを GKE で稼働
  - Deployment は 10 個程度
  - バッチは Cloud Composer で実行
- ログは Fluentd から BigQuery に Streaming Insert
- CircleCI を利用した継続テスト
- Cloud Build による継続的インテグレーション
- 構成変更は Developer、SRE 問わず相互レビューにて実施



# チーム、体制に ついて



# 取り組む上での課題&対策(組織、開発)

## 体制:

- 開発チーム、運用設計・構築チーム、QA チーム、技術支援チーム

## 課題&対策:

- 課題 ①: 周りの理解を得る (経営層、技術者層)
- 課題 ②: 知識・実績がなかった (マイクロサービス、コンテナ、オーケストレーション)
- 課題 ③: 失敗リスクの影響範囲の最小化
  - 技術学習(社内技術セミナー + 独学 + ハンズオンで基礎固め)
  - 解説資料作成 & 説明行脚
  - **スモールスタート** (小規模 PJ) で手触り感を上げながら段階的に拡大
  - Google カスタマーサポート技術者の支援体制
- 課題 ④: 連帯感の重要性 (開発メンバー、インフラエンジニア、部門横断のエンジニア)
  - 認識合わせ、技術習得などの連携強化が必要
- 課題 ⑤: サービス分割
  - バランスが必要。サービスが分割過ぎると、制御 & 障害対応時の難しさ

# Red for Publishers チーム、体制

- Developer 11 名: 広告配信、広告管理 UI、SDK
- SRE 2 名: インフラ構成変更、モニタリング、自動化開発
- Product Manager 2 名
- 大規模なインフラ構成変更は SRE が中心
  - Kubernetes Update、Cloud Composer 導入など
  - Terraform 導入中
- 日々の構成変更は Developer も行う
  - ConfigMap (fluentd.conf, nginx.conf など) 修正
  - MySQL、BigQuery スキーマ変更

これからアプリケー  
ションを  
モダナイズしようとし  
ている  
企業に向けて





# 失敗しないアプリケーション モダナイゼーションの考え方

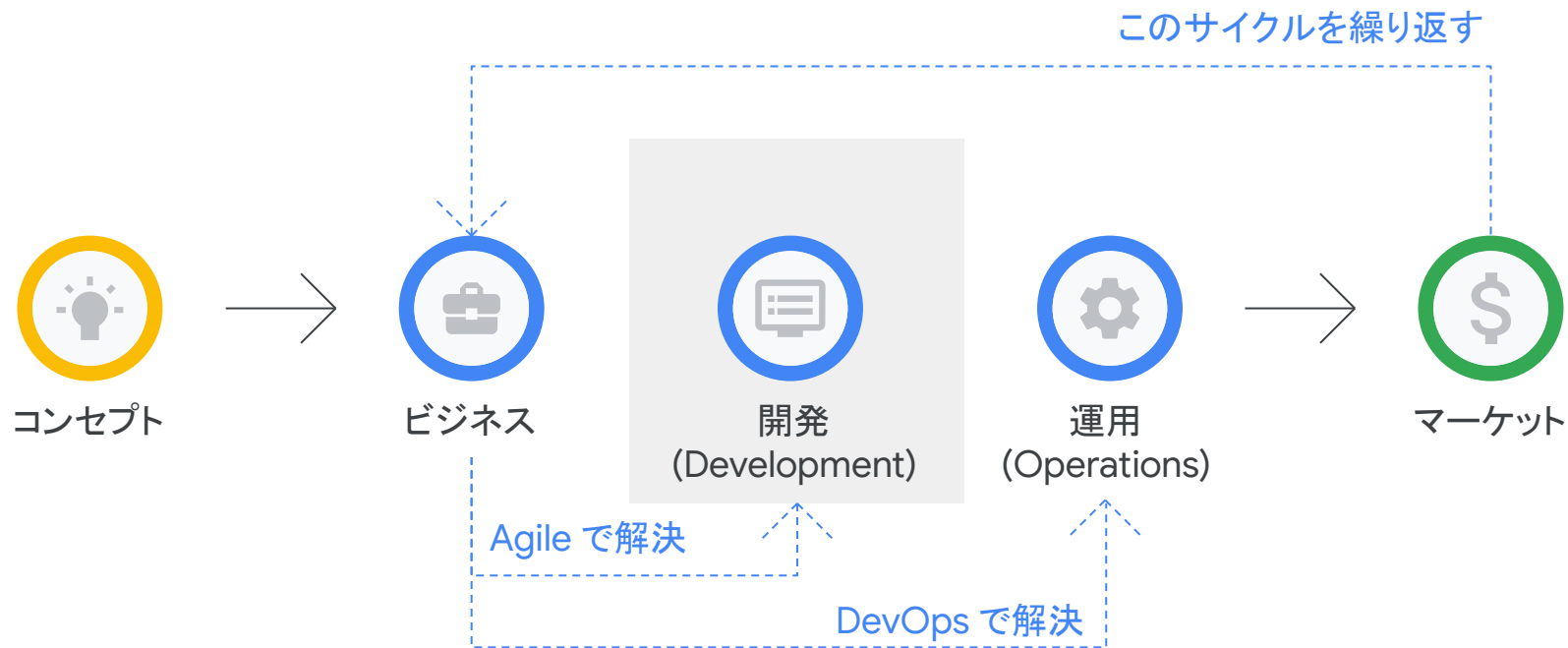
DevOps の重要性



# ソフトウェアが世界を飲み込む Software is eating the world

*Marc Andreessen, Why Software Is Eating the World - WSJ*

# DevOps でアイデアをマーケットにより早く持ち込む



# DevOps の重要性

## 1.53x

優秀なチームは組織の目標を達成する、もしくは目標を超える可能性が 1.53 倍も高くなります

### 事業目標

収益性

生産性

マーケットシェア

カスタマーの数

### 非事業目標

サービスやプロダクトの数

運用の効率性

顧客満足度

サービスやプロダクトの品質

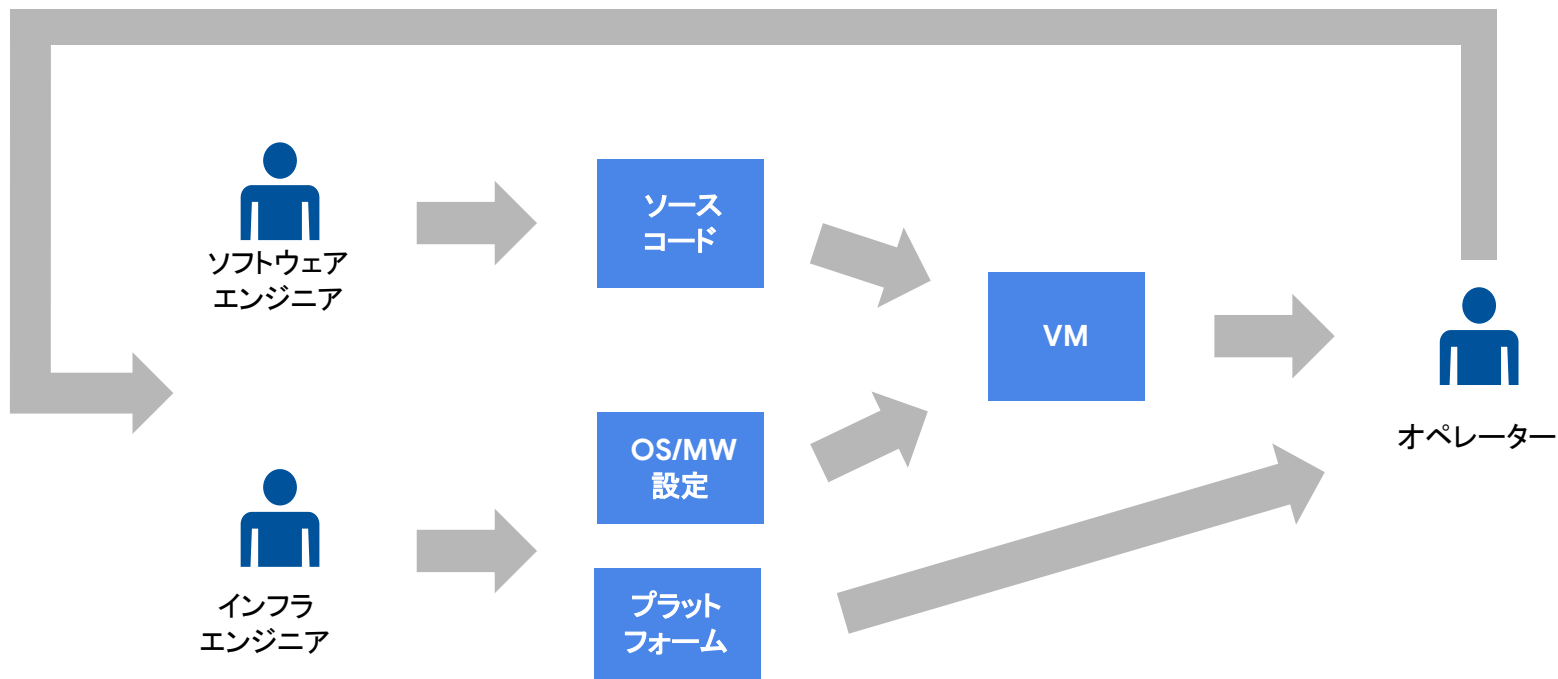
組織やミッションの実現



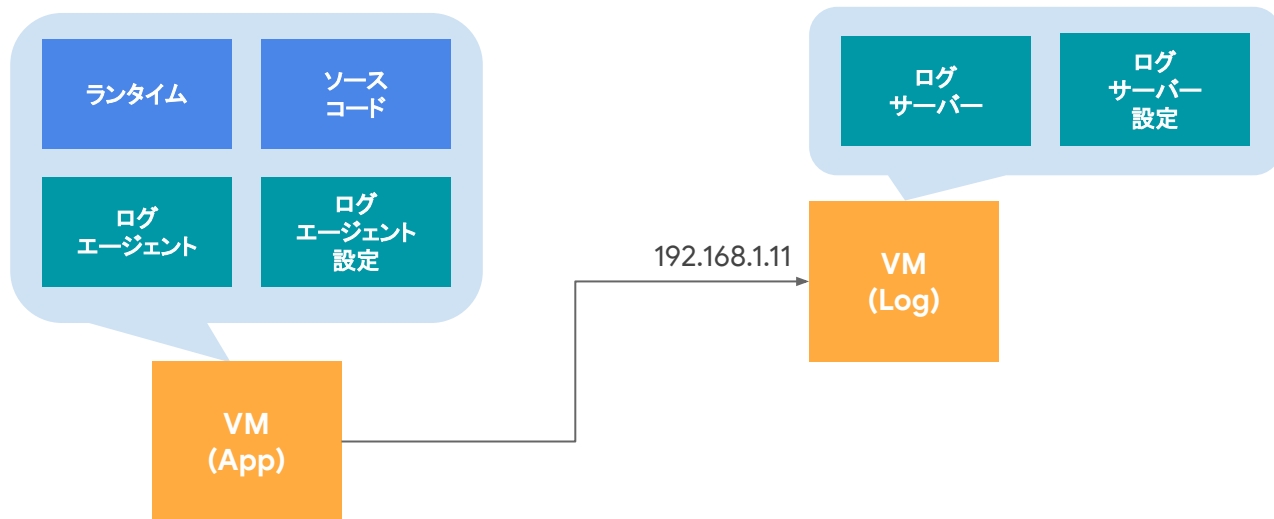


# Kubernetes 登場による変化

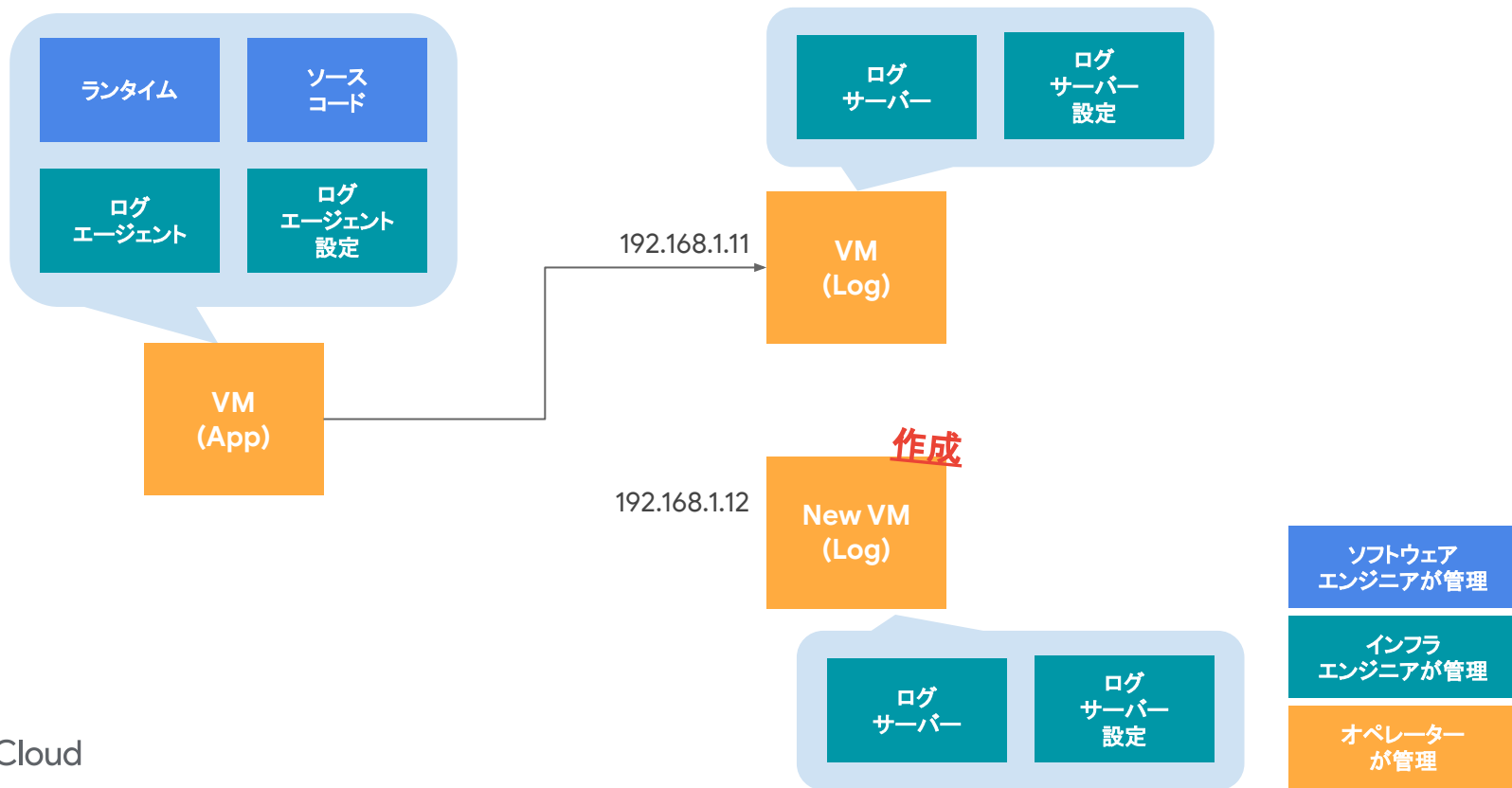
# 従来の開発運用



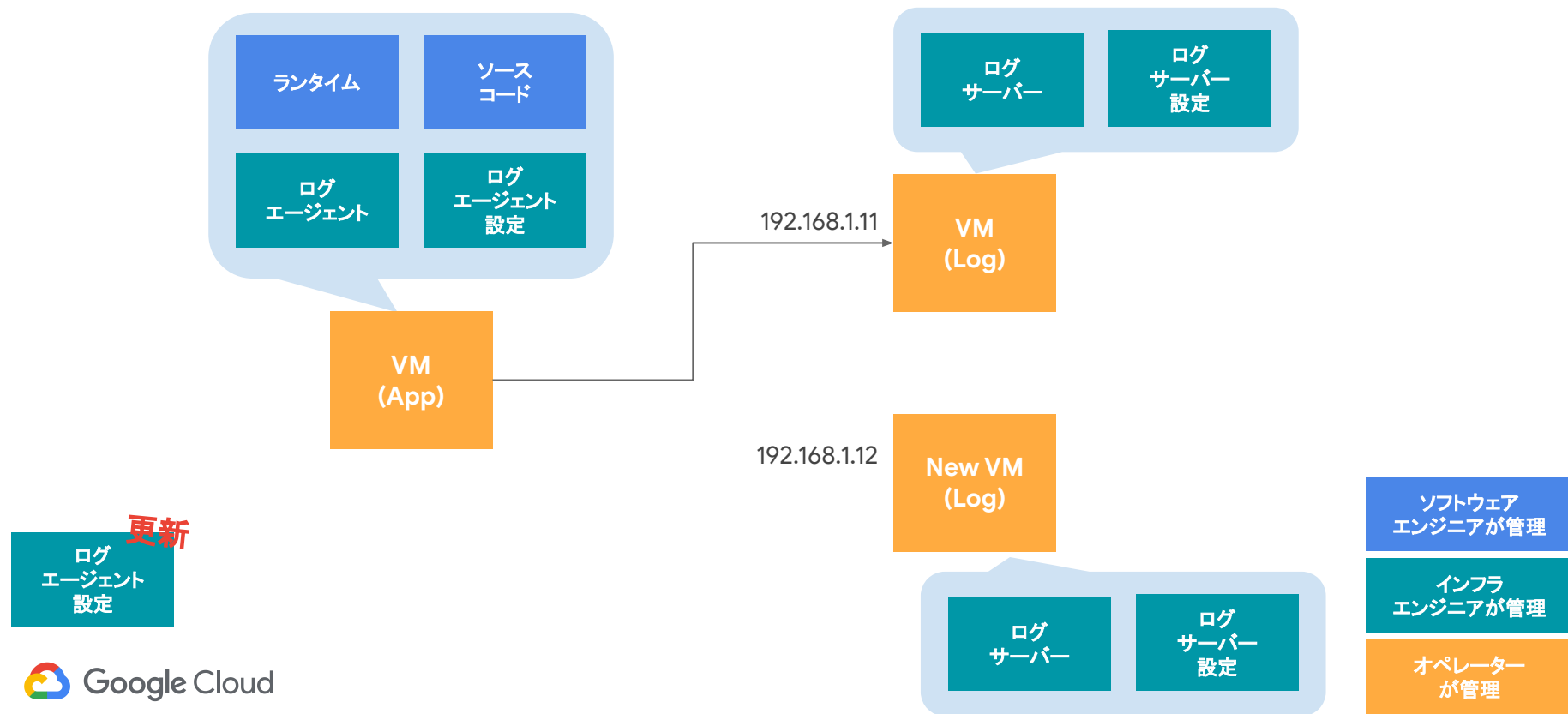
# 例：ログサーバーのリソース不足対応



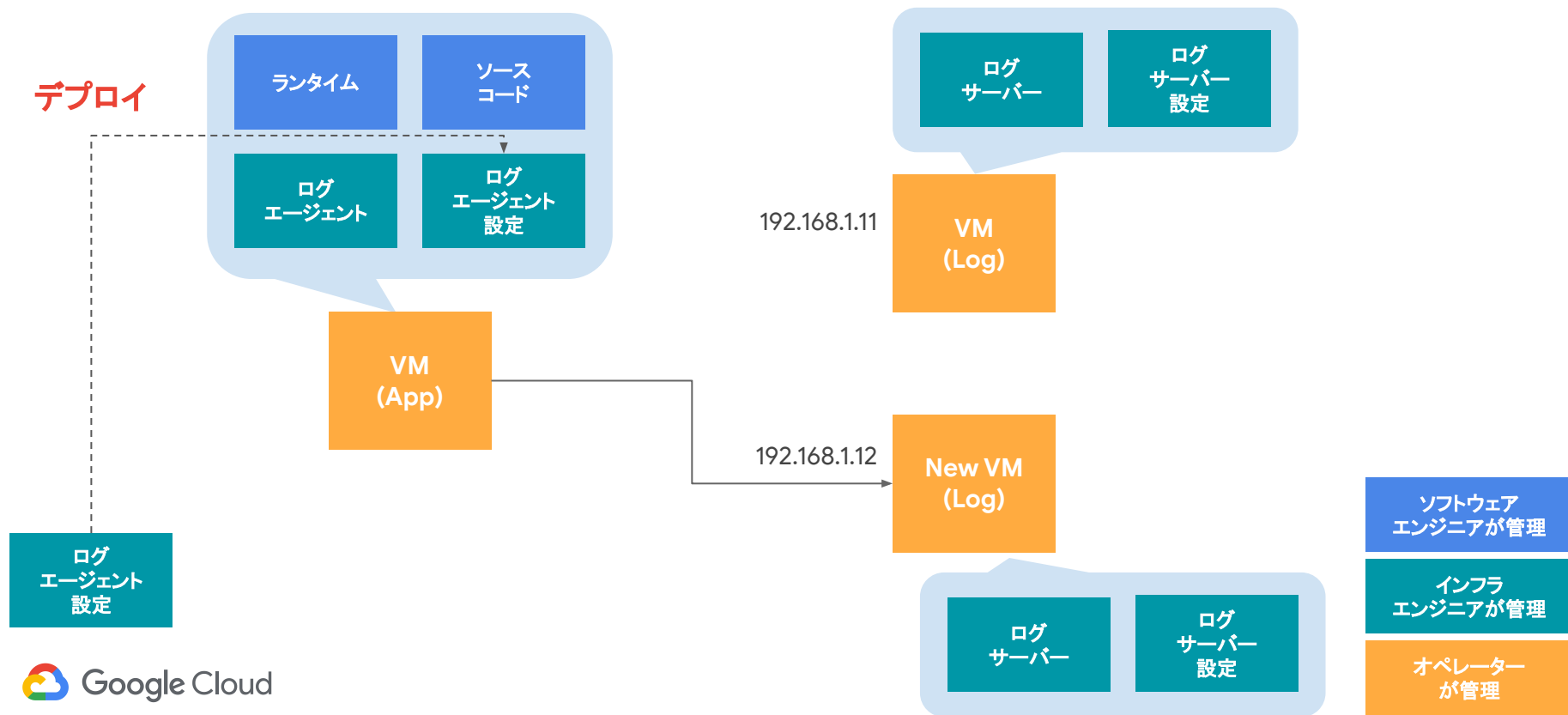
# 例: ログサーバーのリソース不足対応



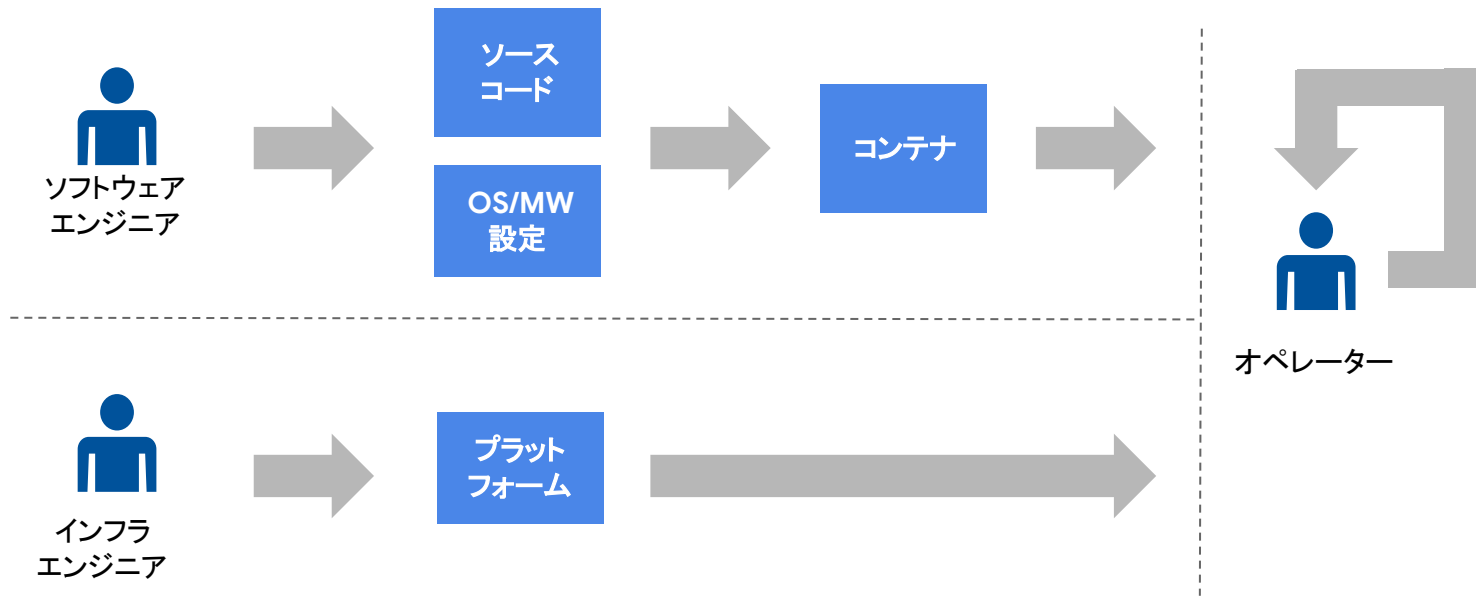
# 例: ログサーバーのリソース不足対応



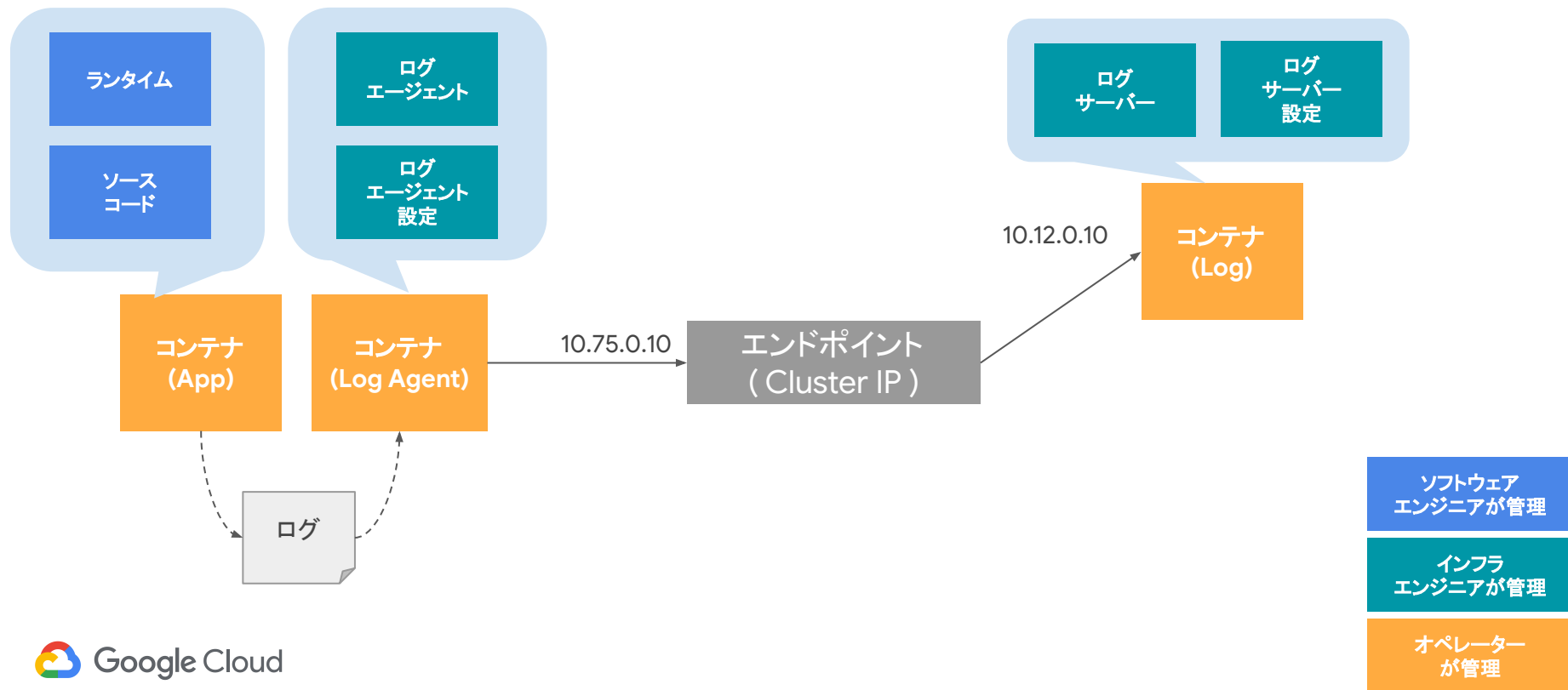
# 例: ログサーバーのリソース不足対応



# コンテナ (Kubernetes) を使った 開発運用の変化

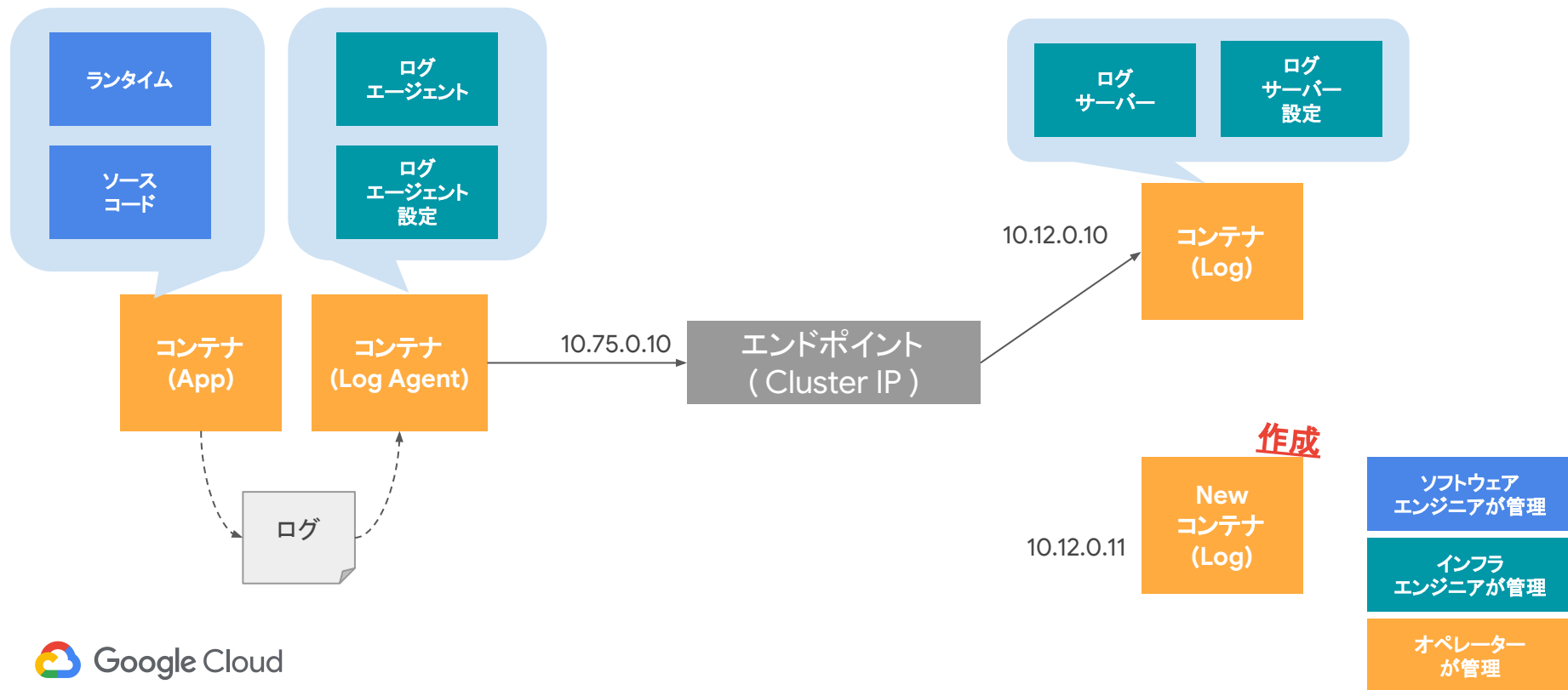


# 例: コンテナを使ったログサーバーの リソース不足対応

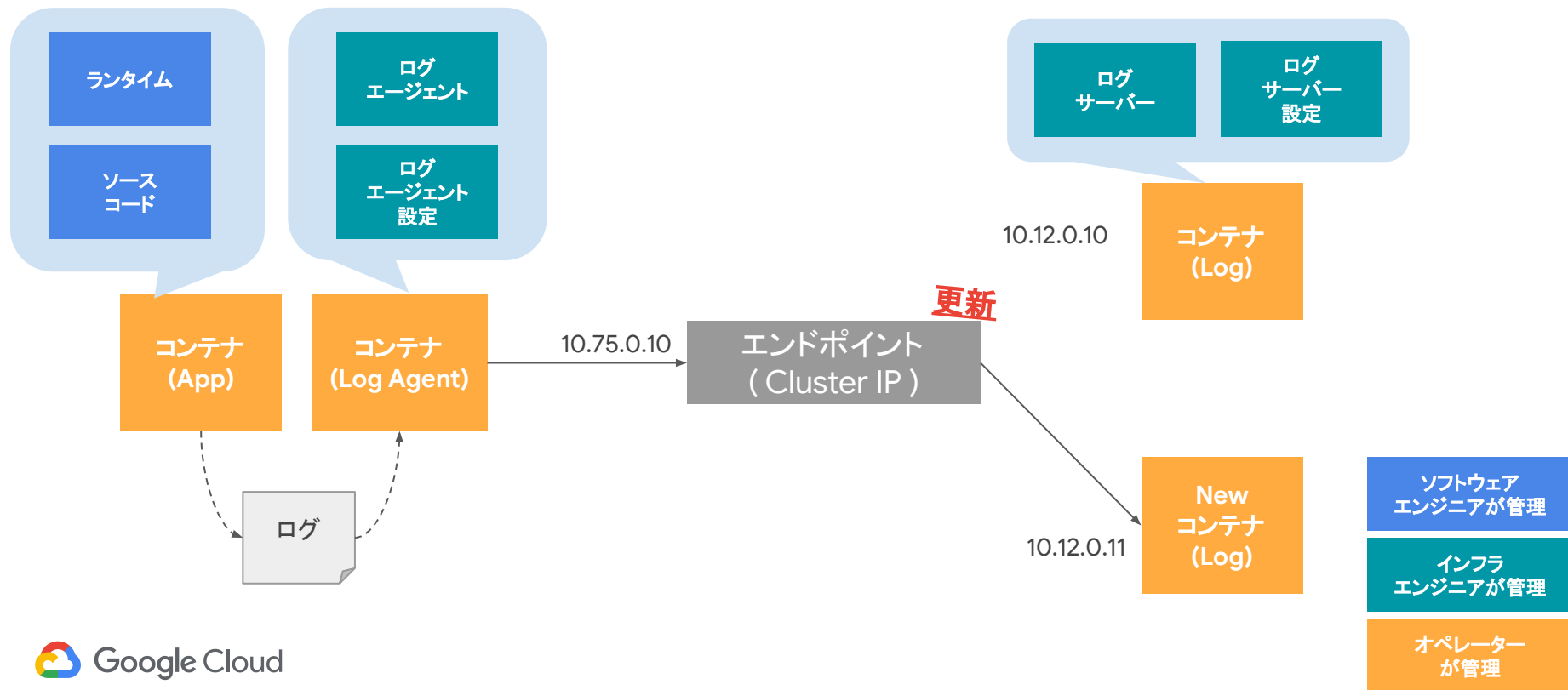




# 例: コンテナを使ったログサーバーのリソース不足対応



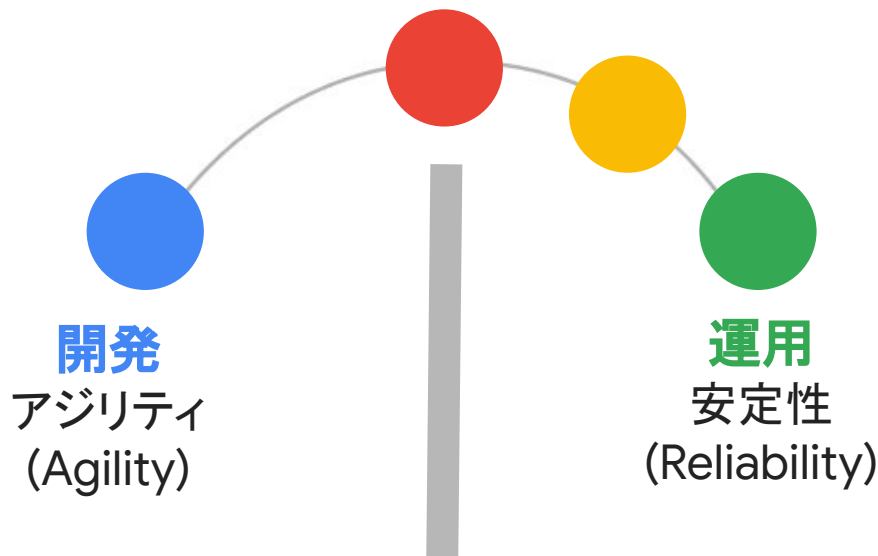
# 例: コンテナを使ったログサーバーのリソース不足対応



# DevOps を実践する際の問題

# 「開発」と「運用」

インセンティブが一致しない



# よくある問題と解決案

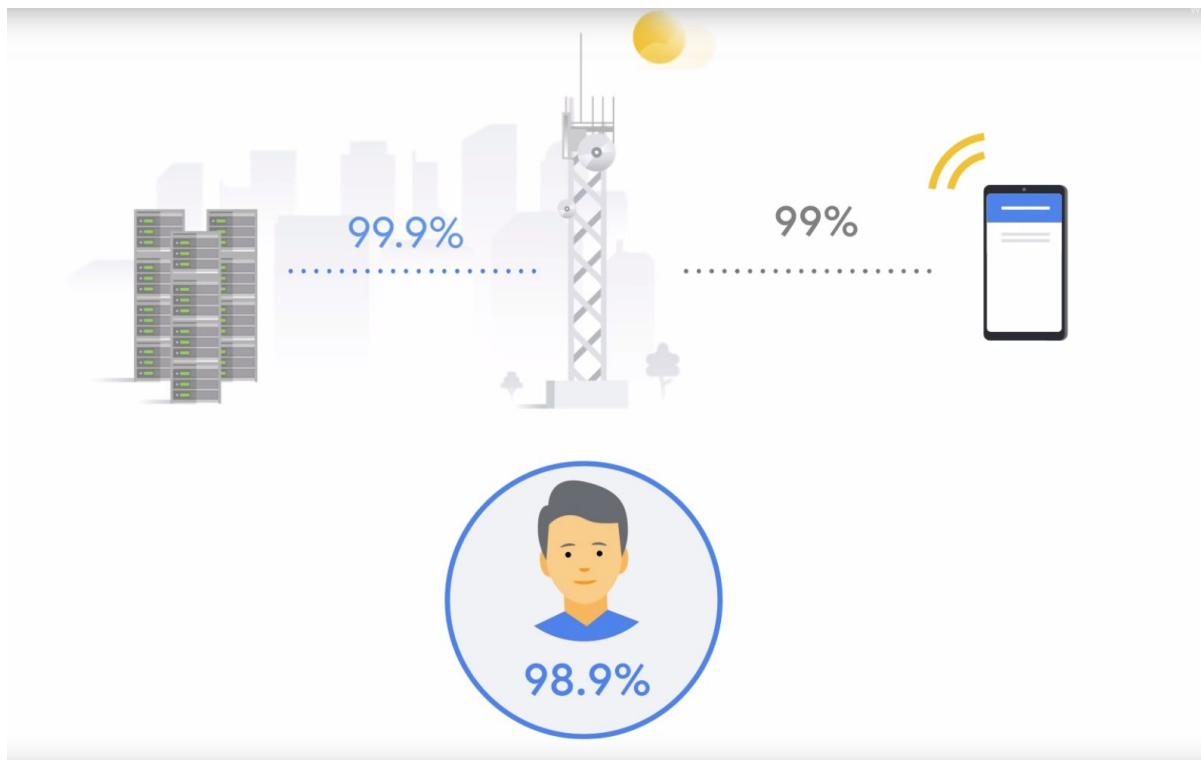
## 合理的でない安定性の定義

- ダウンタイム無し、
- 99.9999999999% の稼働率

## 解決策

- カスタマー・エンドユーザーが潜在的に要求する ”安定” を定義して計測する
- 経営層・決定権のある人からのサポートを得て、組織・チーム間の壁を取り払う

# 安定性の定義



# 安定性の計測

## ▶ 率直なアプローチ:

$$\text{Availability} = \frac{\text{good time}}{\text{total time}}$$

## ▶ 人間にとって直感的

## ▶ アップタイムなどのバイナリーメトリクスより比較的簡単に計測可能

## ▶ マイクロサービスの場合、計測はチャレンジが必要

- リクエストが来ていないマイクロサービスはダウンしているか？
- 3 つ中 1 つのマイクロサービスがダウンしている場合、それはサービスダウンなのか？



# Ops に求められる事

# 安定性の最適化

$$\begin{array}{c} \text{安定性} \\ \text{(Reliability)} \end{array} = \frac{\text{Mean Time Between Failure (MTBF)}}{\text{Mean Time Between Failure (MTBF) + Mean Time To Recover (MTTR)}} \times \text{Impact}$$

ITシステムにおける MTTR = Time To Detect (TTD) + Time To Medicate (TTM)

# 改善の方向性

MTBF = 障害報告書

TTD = モニタリング

TTM = ツール, トレーニング, 自動化, ドキュメント

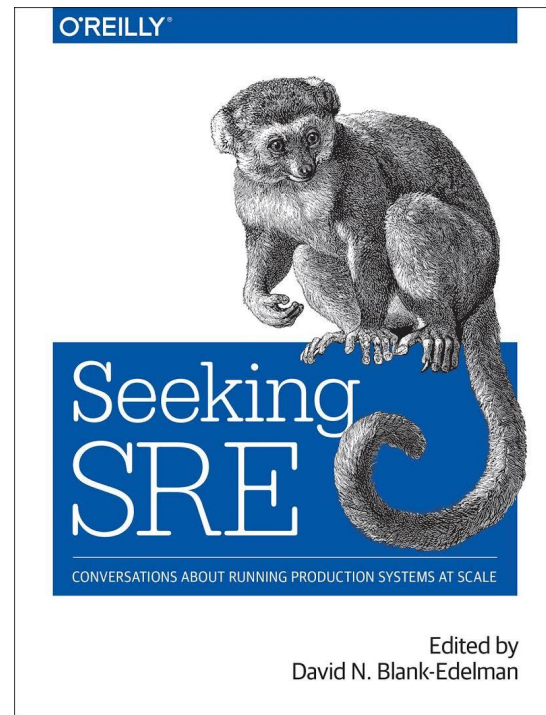
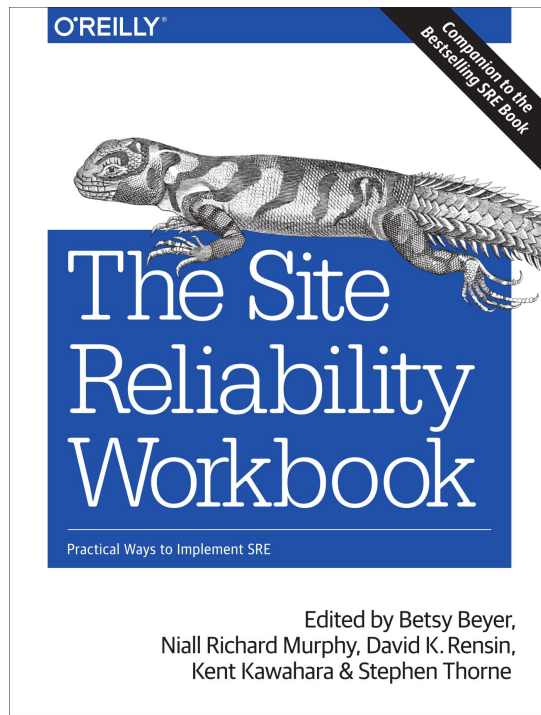
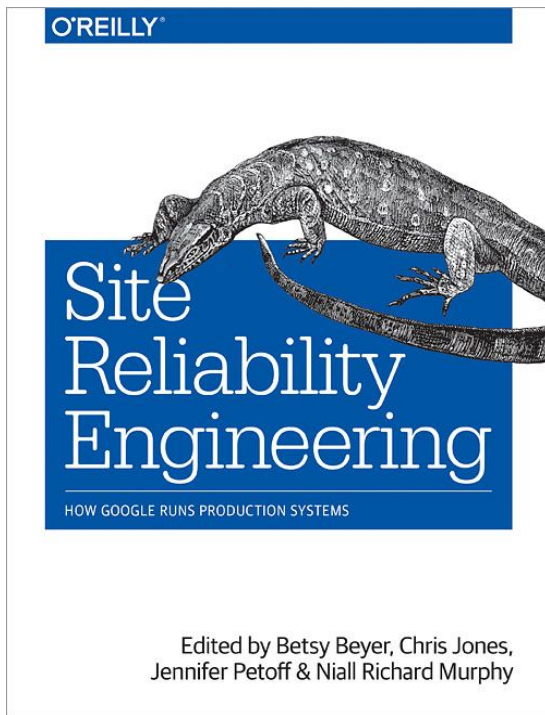
Impact = エンジニアリング, チェンジマネジメント

まとめ

# まとめ

- ソフトウェアがビジネスの中心になり、以前にも増してDevOpsの重要性が増す
- Kubernetes の登場により、技術的に DevOps を実現しやすくなった
- Ops には安定性 (Reliability) の最適化が求められ、役割は以前にも増して重要になってくる

**DevOps** も一緒にはじめよう



# Thank you