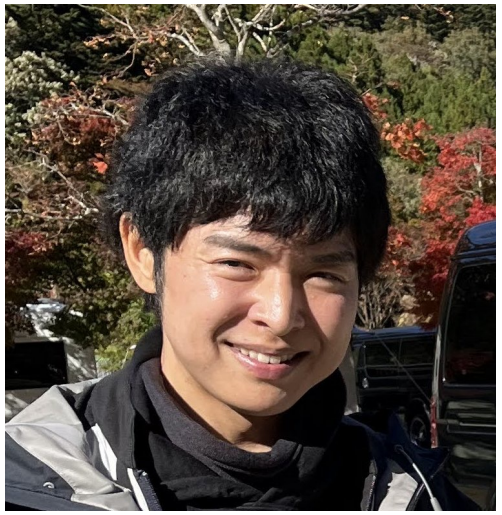
AWS オンラインセミナー プラットフォームエンジニアリングって何？

チームとプラットフォームをクラウドネイティブな開発に最適化する

Masatoshi Hayashi

Specialist Solutions Architect, Containers
AWS Japan

自己紹介



林 政利, @literalice

Specialist Solutions Architect, Containers / AWS Japan

好きな AWS のサービス

Amazon Elastic Kubernetes Service (Amazon EKS)
AWS Cloud Development Kit (AWS CDK)

SIer

フリーランス

Java/Ruby 開発者
Kubernetesインフラ設計
(Web 企業)

Containers SA, Support Engineer
(クラウド製品ベンダー)

AWS Japan
Containers SA

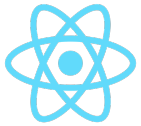
本セッションの目的

- 対象者
  - AWS でアプリケーションを開発、運用している
- ゴール
  - モダンなクラウドネイティブ開発において、
    - どのようなチームモデルが必要か理解する
    - プラットフォームの理想はどのようなものか理解する

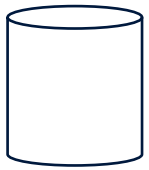
アプリケーション開発と運用を楽にする

アプリケーションはコードだけでは動かすことができず、様々なインフラストラクチャーを必要とする

コード開発



デプロイする環境の用意



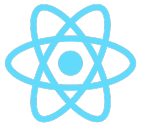
データベースの用意

ネットワークの構築

プラットフォームエンジニアリング

アプリケーションの開発に集中できるプラットフォームの構築

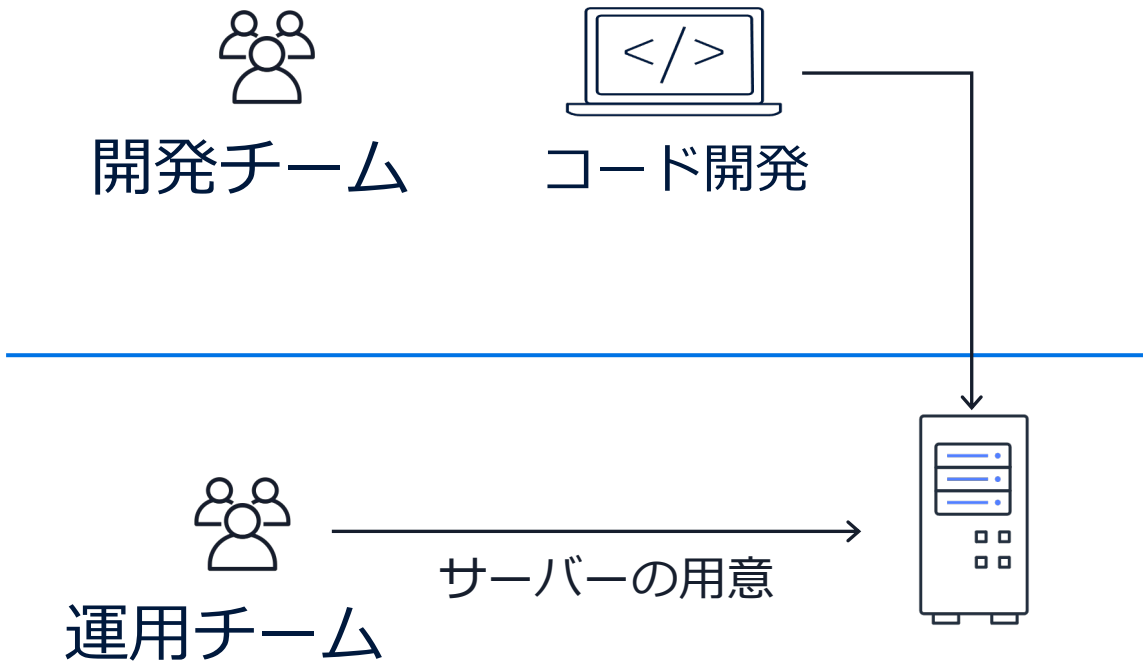
コード開発



アプリケーションが動いてアクセスするURLが払い出される
セキュリティも考慮してくれる
アプリケーションを監視して障害を検知したらロールバック

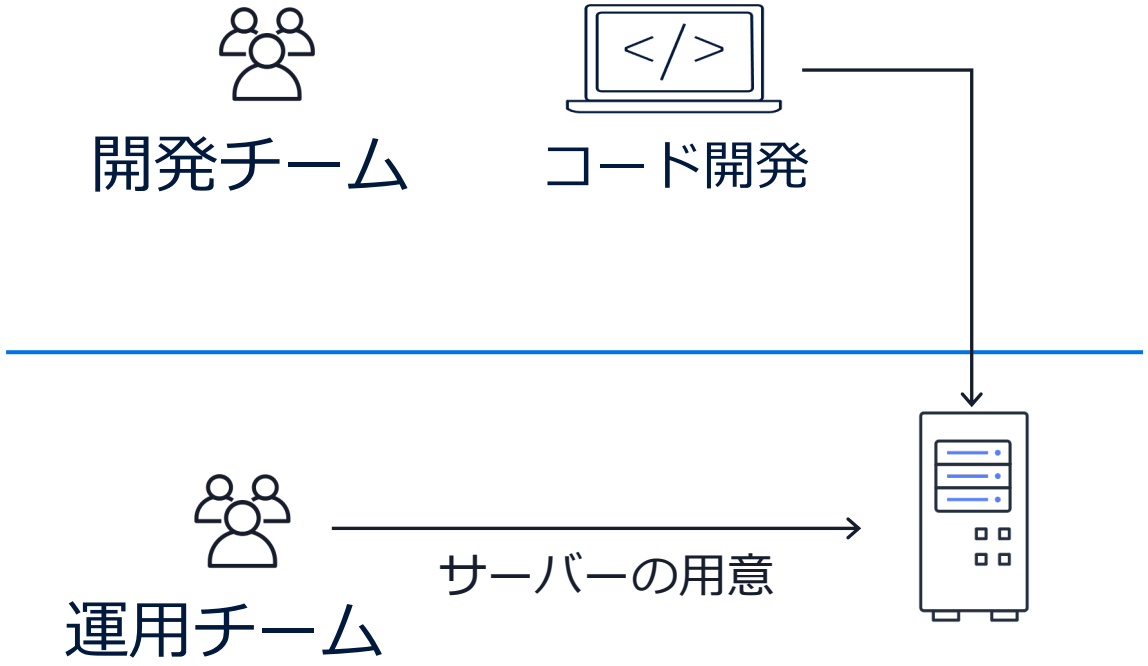
いい感じのプラットフォーム

チームとプラットフォームに求められる責務



チームとプラットフォームに求められる責務

アプリケーションとインフラストラクチャーの垣根が曖昧な環境で理想的なチームとプラットフォームとは？



Amazon RDS — アプリケーションのデータベーススキーマ管理

Elastic Load Balancing — 負荷分散の設定

AWS Auto Scaling — スケーリングポリシー設定

Amazon Inspector — 脆弱性の管理

バッチアプリケーション

Amazon EventBridge

AWS Step Functions

AWS Lambda

チームとプラットフォームに求められる責務

アプリケーションとインフラストラクチャーの垣根が曖昧な環境で理想的なチームとプラットフォームとは？

開発チーム

コード開発

運用チーム

Amazon RDS  アプリケーションの
データベーススキーマ管理

Elastic Load
Balancing  負荷分散の設定

AWS Auto Scaling  スケーリングポリシー設定

Amazon
Inspector  脆弱性の管理

コードをデプロイする環境のプロビジョニング

いい感じのプラットフォーム

チームとプラットフォームに求められる責務

アプリケーションとインフラストラクチャーの垣根が曖昧な環境で理想的なチームとプラットフォームとは？

開発チーム

コード開発

運用チーム

コードをデプロイする環境のプロビジョニング

アプリケーションのデータベーススキーマ管理

負荷分散の設定　スケーリングポリシー設定　脆弱性の管理

いい感じのプラットフォーム

従来のチームとプラットフォームの課題

従来のチームとプラットフォームの課題

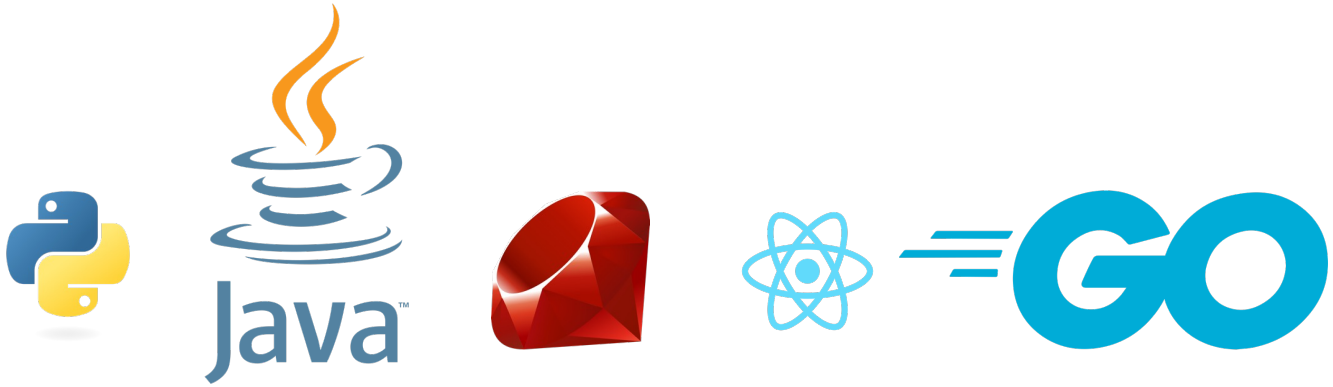
クラウドネイティブなチームモデル

プラットフォームに求められる機能

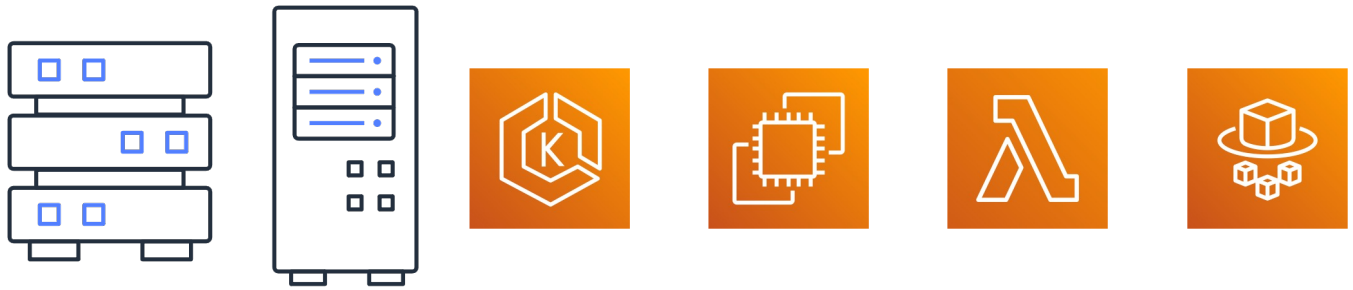
プラットフォームエンジニアリングの実践

運用チームと開発チーム

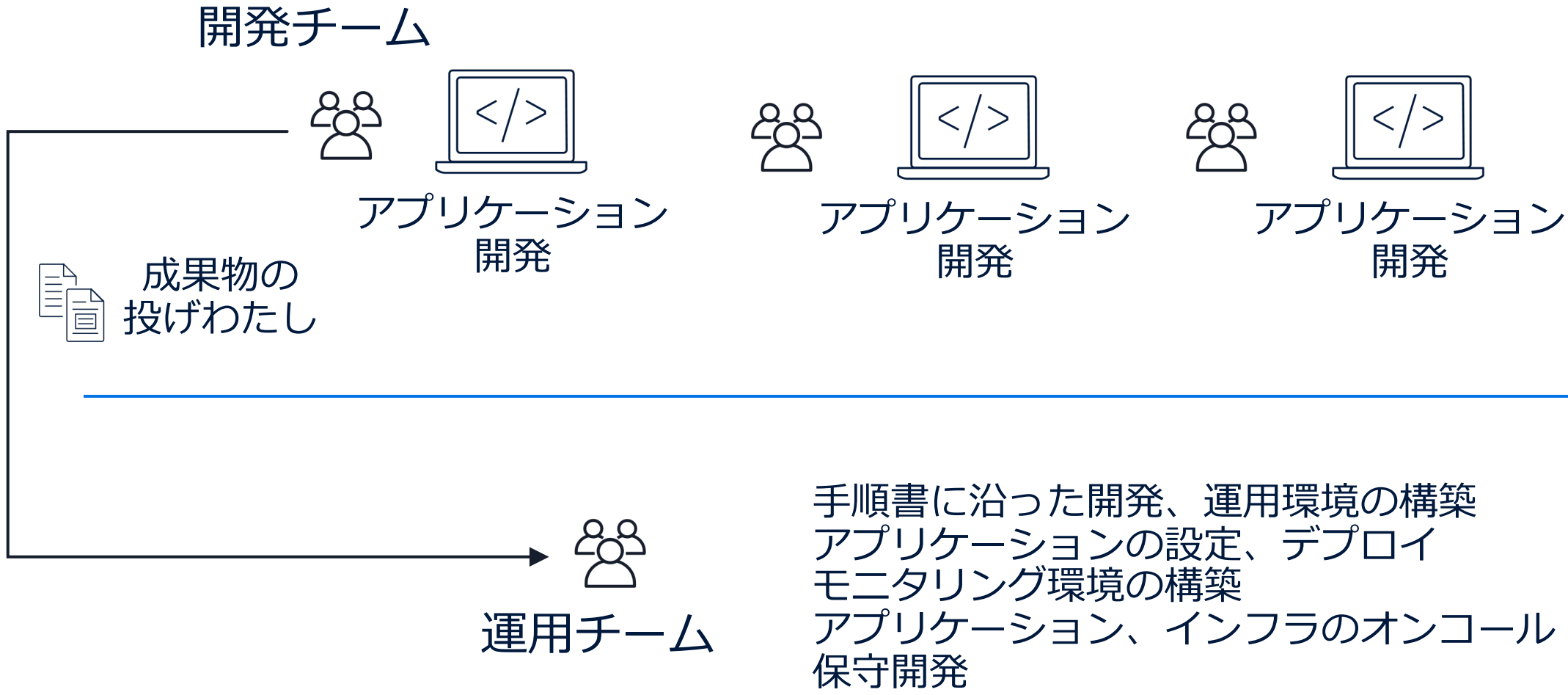
開発チーム



運用チーム



「引き継ぎ」コラボレーションモデル



「引き継ぎ」コラボレーションモデルの課題

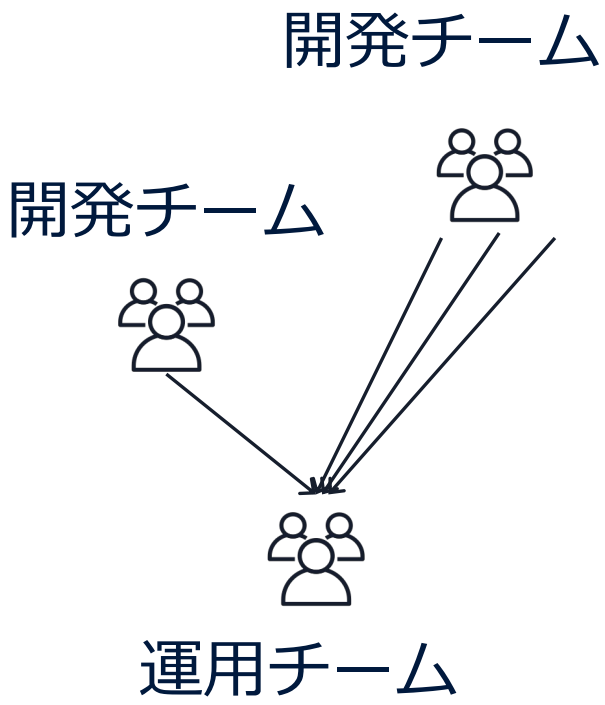
組織の規模が大きくなり、アプリケーションの変更回数が増えてくるとスケールの課題に直面する

運用チームの負荷がスケールしない

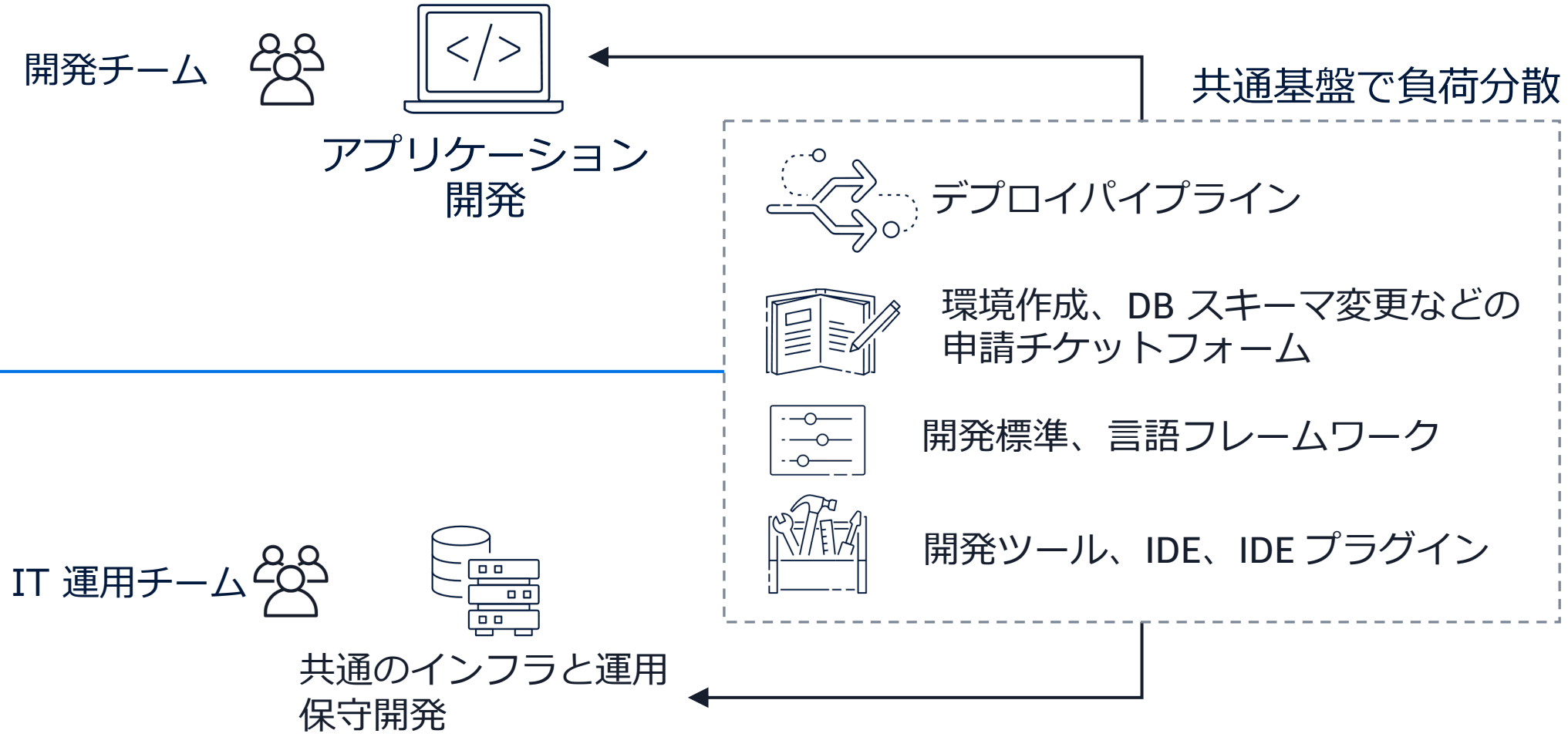
- 開発チームのリクエスト数 (アプリ更新など) 増加
- 開発チームの増加

運用と開発が利益相反しやすい

- 開発は早く要件を実装してどんどんデプロイしたい
- 運用はなるべく環境を変更したくない



共通基盤 - サイロ化されたチームの負荷分散



アプリケーションの観点に欠ける基盤

共通基盤を導入しても、組織がサイロ化していると業務の改善につながらない

開発チーム

Kubernetes API の呼び出し

アプリケーションデプロイ
ロードバランサー割り当て
オートスケール
etc.


Amazon SQS


Amazon RDS


AWS IAM

Kubernetes クラスター

共通基盤

運用チーム

- Kubernetes の習熟が必要
  - Kubernetes の機能、API は運用チーム向けのため、開発チームの負荷が高い
- Kubernetes 外のインフラは申請ベースで変更依頼

開発チームの負荷軽減に繋がらない

サイロ化されたチームと共通基盤の課題

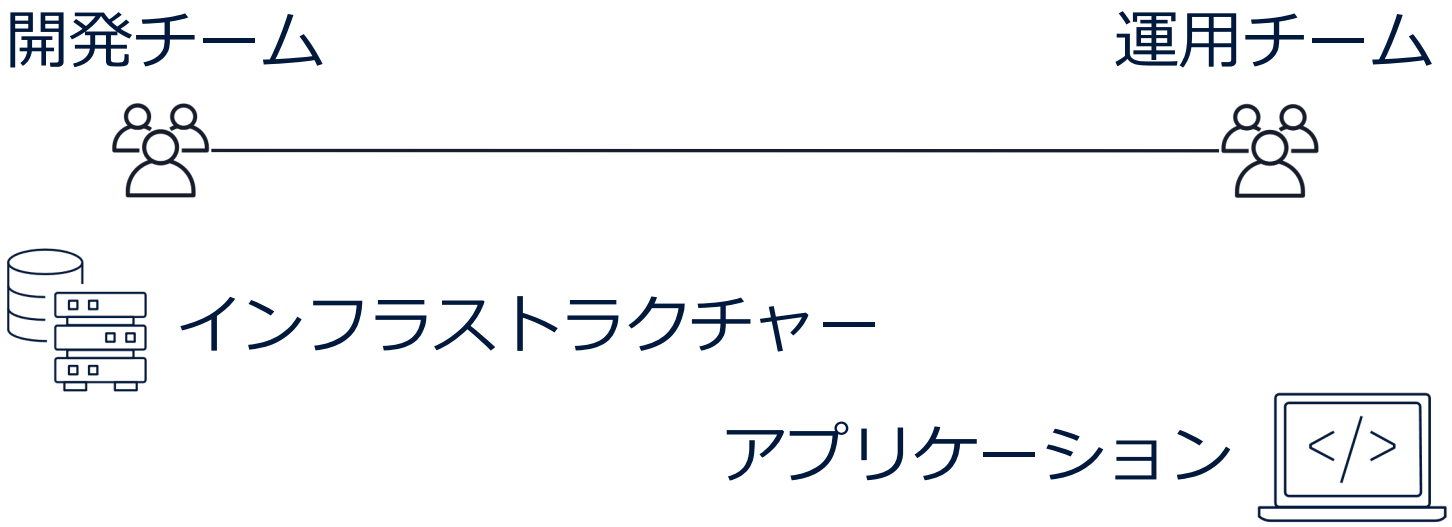
アプリケーション開発の視点で統合されていない

- 組織内外にバラバラに存在するプロセスや資料の調査が必要
- 基盤の機能
- チケットシステム
- プラットフォームやクラウドの知識、ベストプラクティス

アプリケーション開発と直接関係ないことを
たくさん把握する必要がある

クラウドネイティブな開発とチームモデル

- 「引き継ぎ」を最小限にする必要がある
- アプリケーションとインフラストラクチャーの垣根があいまいに
  - 開発チームと運用チームがよりよくコラボレーションする文化の重要性
- チームの役割自体も変わる必要がある



クラウドネイティブなチームモデル

従来のチームとプラットフォームの課題

クラウドネイティブなチームモデル

プラットフォームに求められる機能

プラットフォームエンジニアリングの実践

逆コンウェイの法則とチームの役割

コンウェイの法則

システムのアーキテクチャは組織構造やチーム間のコミュニケーションを反映したものになる

逆コンウェイ戦略

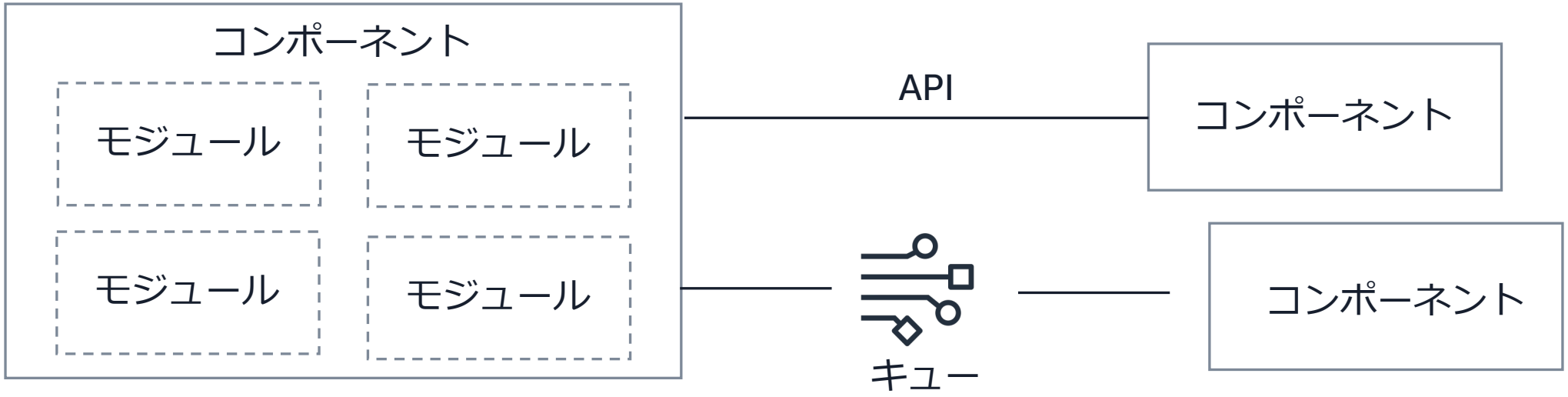
理想的なアーキテクチャを実現したいなら、アーキテクチャに沿ったチーム構成をとればいい

クラウドネイティブな開発には
クラウドネイティブなチーム構成が必要

クラウドネイティブなアーキテクチャ

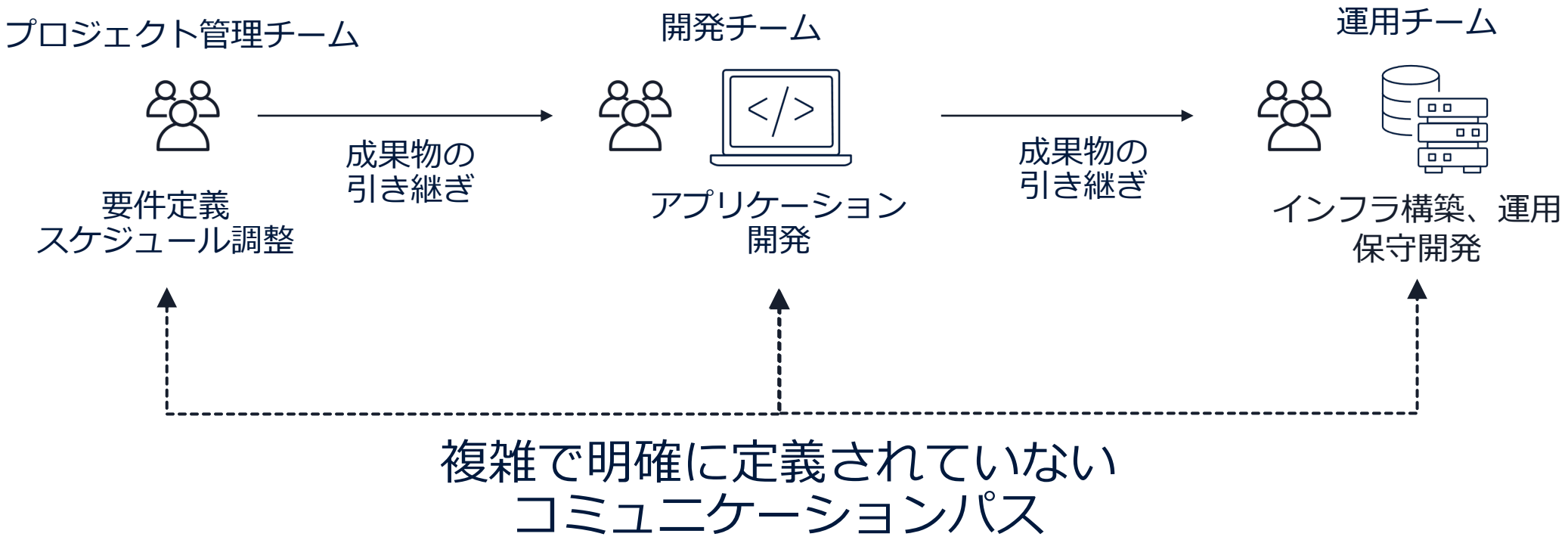
クラウドネイティブなアーキテクチャを特徴づける最も重要な性質 – 疎結合

- 責務が明確なコンポーネントを組み合わせるアーキテクチャ
- コンポーネント間の連携は明確に定義されたインターフェース (API) で行う
- 別のコンポーネントの詳細を把握する必要がない
- AWSのさまざまなマネージドサービスと連携できるモデル



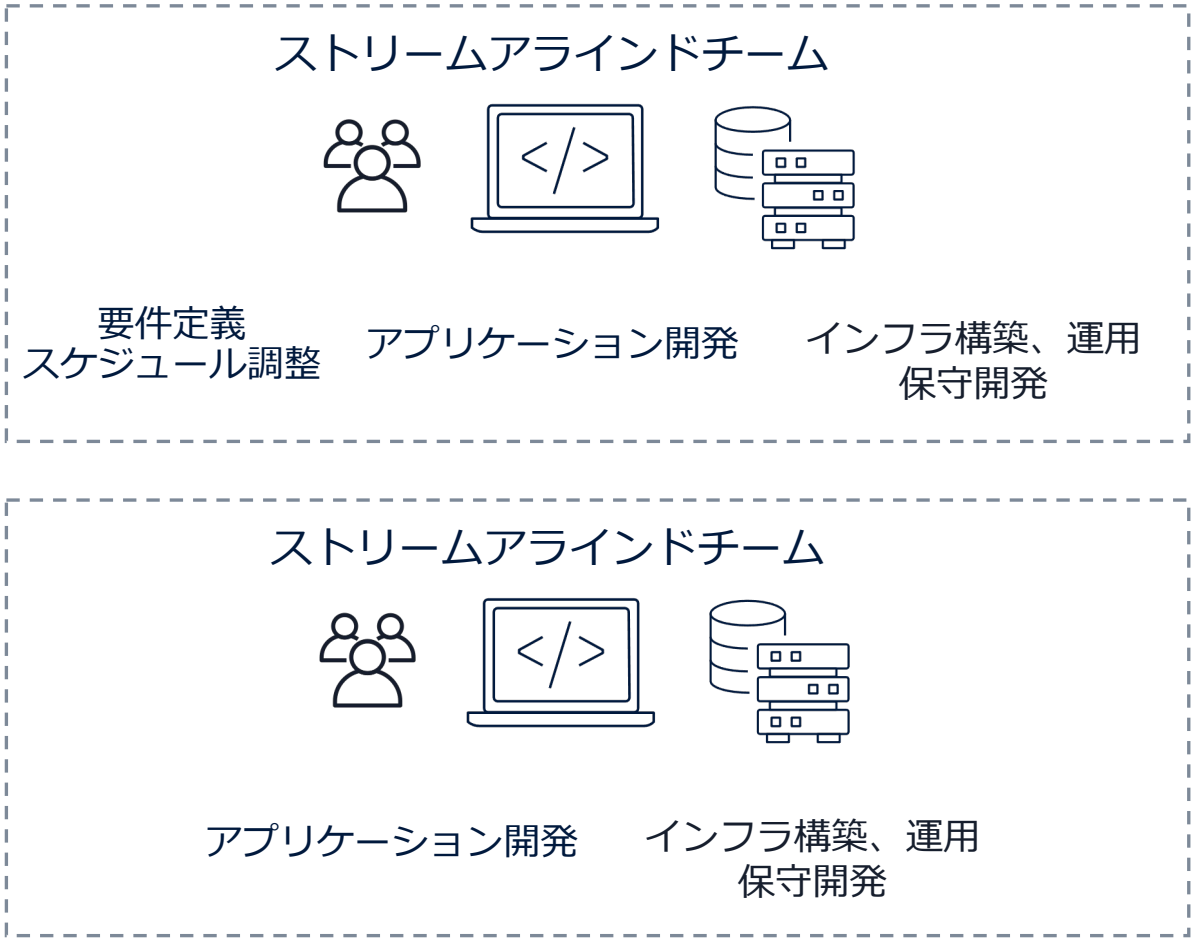
クラウドネイティブではないチーム構成

依存関係が定義されていない、密結合なアーキテクチャのチーム構成



クラウドネイティブで疎結合なチーム構成

凝集性の高いストリームアラインドチームを目指す



ストリームアラインドチーム

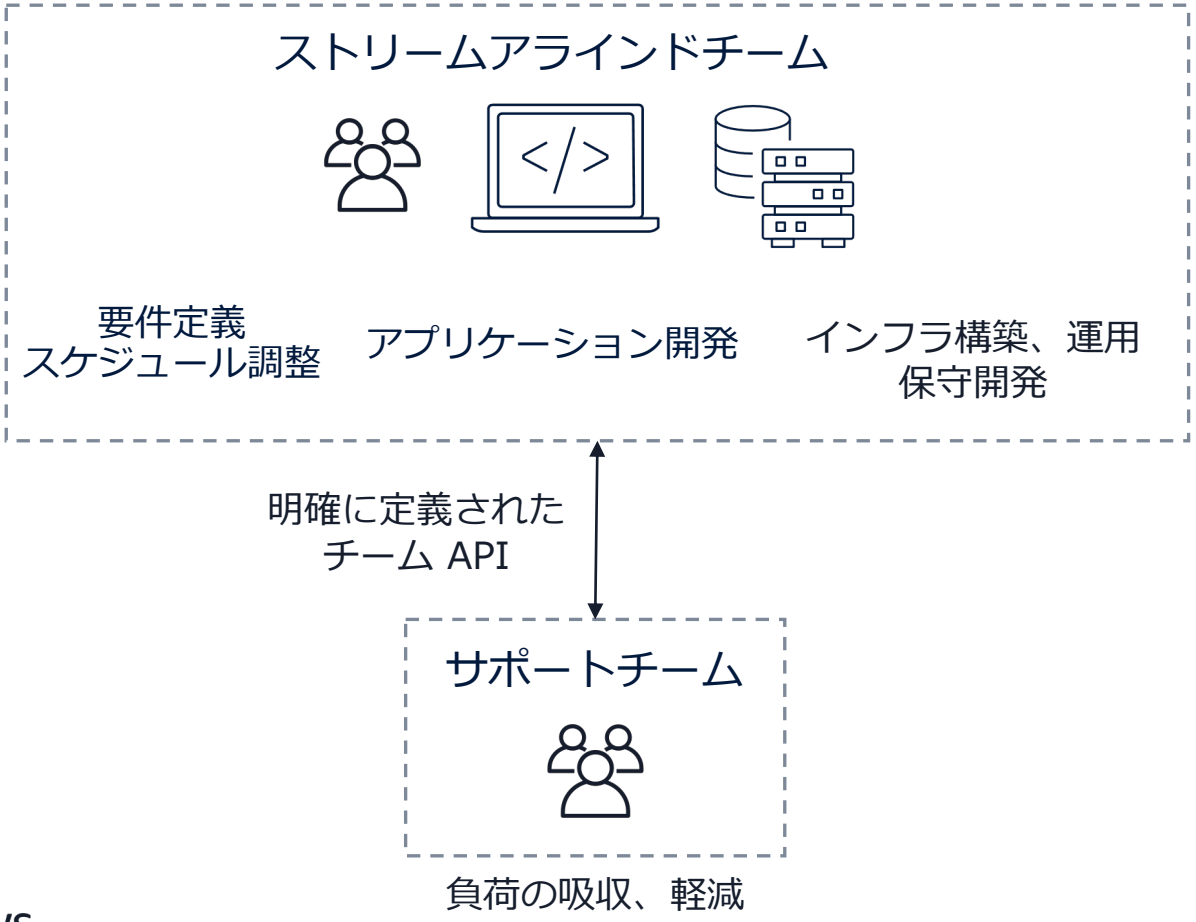
- チームトポロジー　価値あるソフトウェアをすばやく届ける適応型組織設計 より

- 「引き継ぎ」が発生しない
- 価値のあるサービス、ユーザーストーリー設計、開発、運用を一貫して行う
- 自己完結できる、明確な責任の境界をもつ

引き継ぎなく設計から運用まで担当するため、膨大な学習コストと負荷がかかる

クラウドネイティブで疎結合なチーム構成

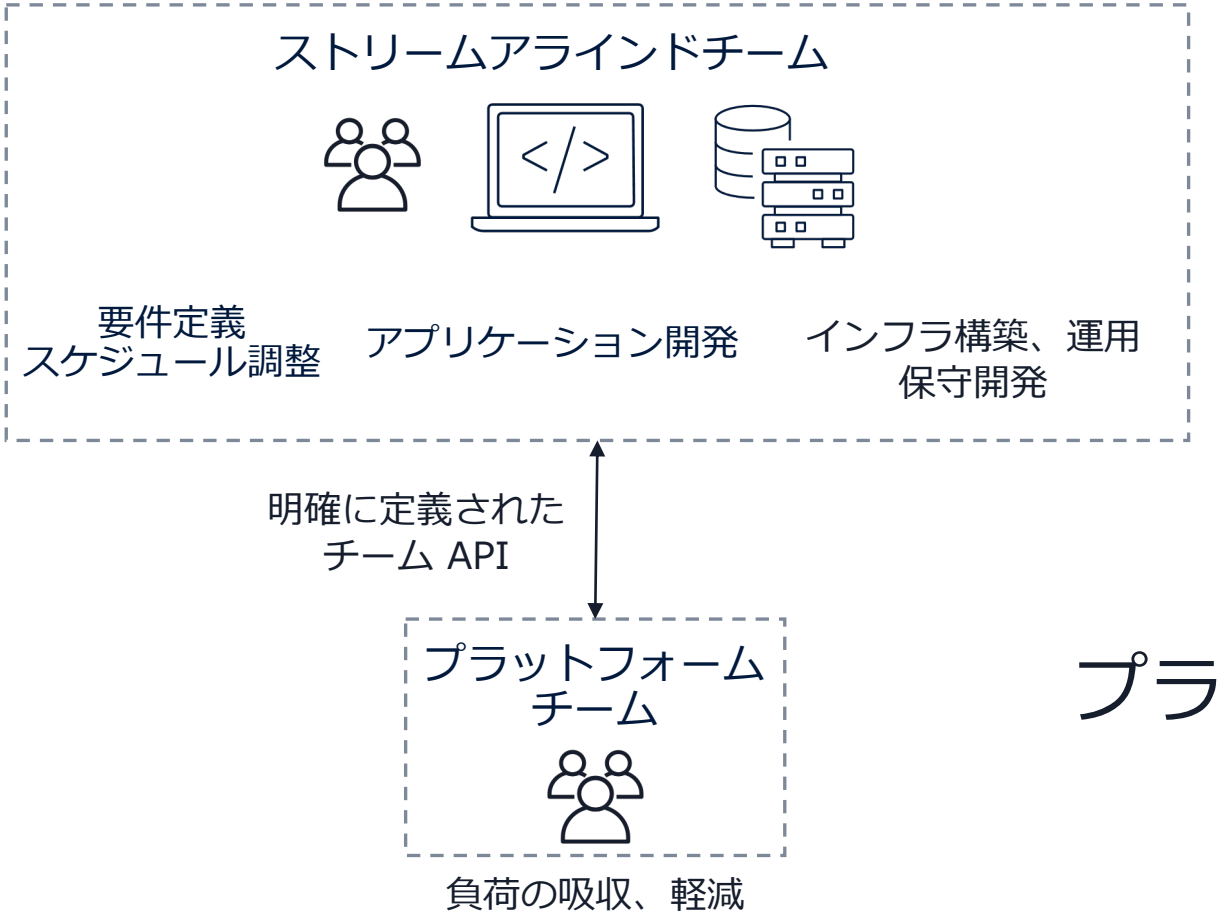
ストリームアラインドチームの負荷を軽減するサポート型チーム



- UX スペシャリスト
  アプリケーションのユーザビリティを検証
- セキュリティスペシャリスト
  アプリケーションのアーキテクチャをレビュー
- サブシステム構築
  再利用が容易、または特殊な専門知識が必要になる認証やアルゴリズム基盤などを再利用可能なサブシステムとして提供
- トレーニング
  ストリームアラインドチームの自走を支援

サポート型チームのプラットフォーム構築

プラットフォームチームによりストリームアラインドチームの負荷を軽減



- プラットフォームの提供
  アプリケーションが企業や業界のポリシーに準拠できるよう「ガードレール」を設定
  アプリのネットワークやコンピューティングなど下位レイヤを抽象化

プラットフォームエンジニアリング

プラットフォームエンジニアリング

2022年ごろから注目を集めるプラットフォーム構築のアプローチ

プラットフォーム・エンジニアリングとは何か？

2023年1月15日

https://www.gartner.co.jp/ja/articles/what-is-platform-engineering

「プラットフォーム・エンジニアリング」とは、アプリケーションのデリバリとビジネス価値の創出を加速させるための、テクノロジに対する新しいアプローチです

インフラストラクチャ・オペレーションの自動化とセルフサービス機能により、開発者エクスペリエンスと生産性を向上させます

- クラウドネイティブなチームが前提
- 「ストリームアラインドチームの負荷を軽減する」が出発点
- 「開発者体験」が最重要視される

プラットフォームに求められる機能

従来のチームとプラットフォームの課題

クラウドネイティブなチームモデル

プラットフォームに求められる機能

プラットフォームエンジニアリングの実践

ゴールデンパスとテンプレート

ストリームアラインドチームの負荷を軽減

ストリームアラインドチームには膨大な学習コスト・認知負荷がかかる

- CI/CD、監視、セキュリティ、ポリシー準拠などにすべてに対応することが困難なケースが多い

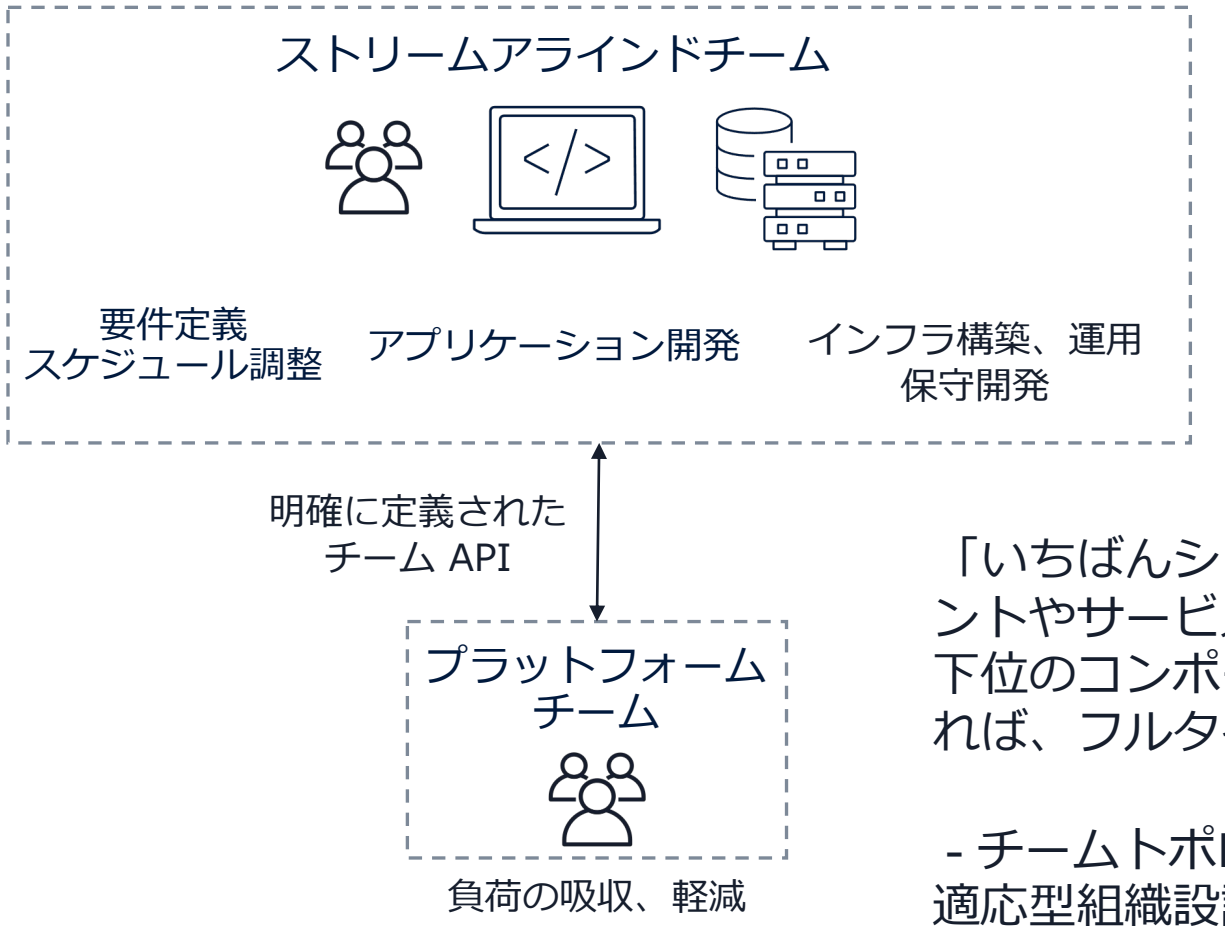
組織のクラウドオペレーションをいかにモダナイズするか
https://aws.amazon.com/jp/blogs/news/how-organizations-are-modernizing-for-cloud-operations/

プラットフォームチームがツールセットやテンプレートを提供

- マネージド IaC オーケストレーションサービス (AWS Service Catalog、Amazon CodeCatalyst など) の活用
- カスタム内部開発者プラットフォームの構築 (Backstage、または自社構築)
- GitOps モデルとツール (Flux、ArgoCD など) の実装
- 再利用可能な中央リポジトリ内のテンプレートとドキュメントの管理
- カスタムライブラリあるいはモジュールの構築 (AWS Cloud Development Kit (CDK) コンストラクト/ライブラリ、Terraform モジュールなど)

セルフサービス化

チームを疎結合に保ちつつ、ゴールデンパスを提供する – チームが自律的に開発、設計、運用できるようにする



プラットフォームチームの目的

ストリームアラインドチームの負荷を下げて、
自律的に設計、開発できるようにする

「いちばんシンプルなプラットフォームは、下位のコンポーネントやサービスについて書いた単なるWikiページ上のリストだ。下位のコンポーネントやサービスが常に確実に動作するのであれば、フルタイムのプラットフォームチームは必要ない」

- チームトポロジー　価値あるソフトウェアをすばやく届ける適応型組織設計

内部開発者プラットフォーム/ポータル

Internal Developer Platform/Portal, IDP – 簡単にゴールデンパスに乗ることができるプラットフォーム

プラットフォームエンジニアリングの実践

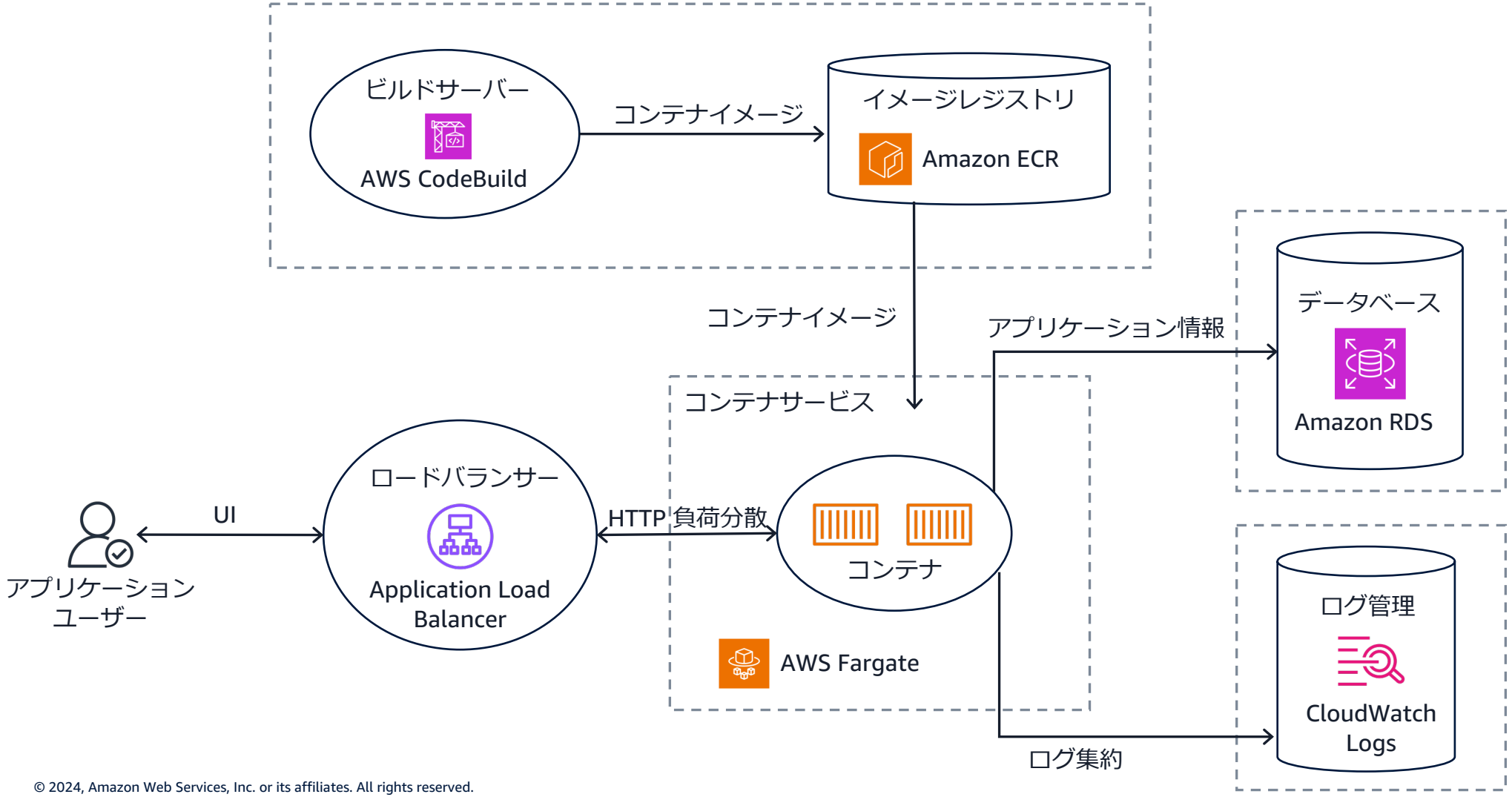
従来のチームとプラットフォームの課題

クラウドネイティブなチームモデル

プラットフォームに求められる機能

プラットフォームエンジニアリングの実践

アプリケーションの例



ストリームアラインドチームの認知負荷の例

プラットフォームチームでどのようにサポートするか

- ストリームアラインドチームの負荷
  - VPC、クラスター、ALB のプロビジョニング
  - CI/CD の構築
  - セキュリティ対策
    - CloudWatch Logs へアクセスできるユーザーを限定する
    - CloudWatch Logs のデータ保護機能を有効化する
    - アプリケーションで使用しているライブラリの脆弱性をスキャンする
  - 監視の構築
  - etc.

最小限のプラットフォームから始める

実体のある「プラットフォーム」でなくても、ストリームアラインドチームの負荷を下げることができる

- ドキュメントの整備
  - 準拠するべきポリシー、利用できる AWS サービスなど
- 特定のストリームアラインドチームと協業して、IaC サンプルコードを提供
  - サンプルコードをベースにストリームアラインドチームをサポート
  - 不明点の回答、サンプルコードの修正など
- アーキテクチャレビュー
- トレーニング

最小限のプラットフォームから始める

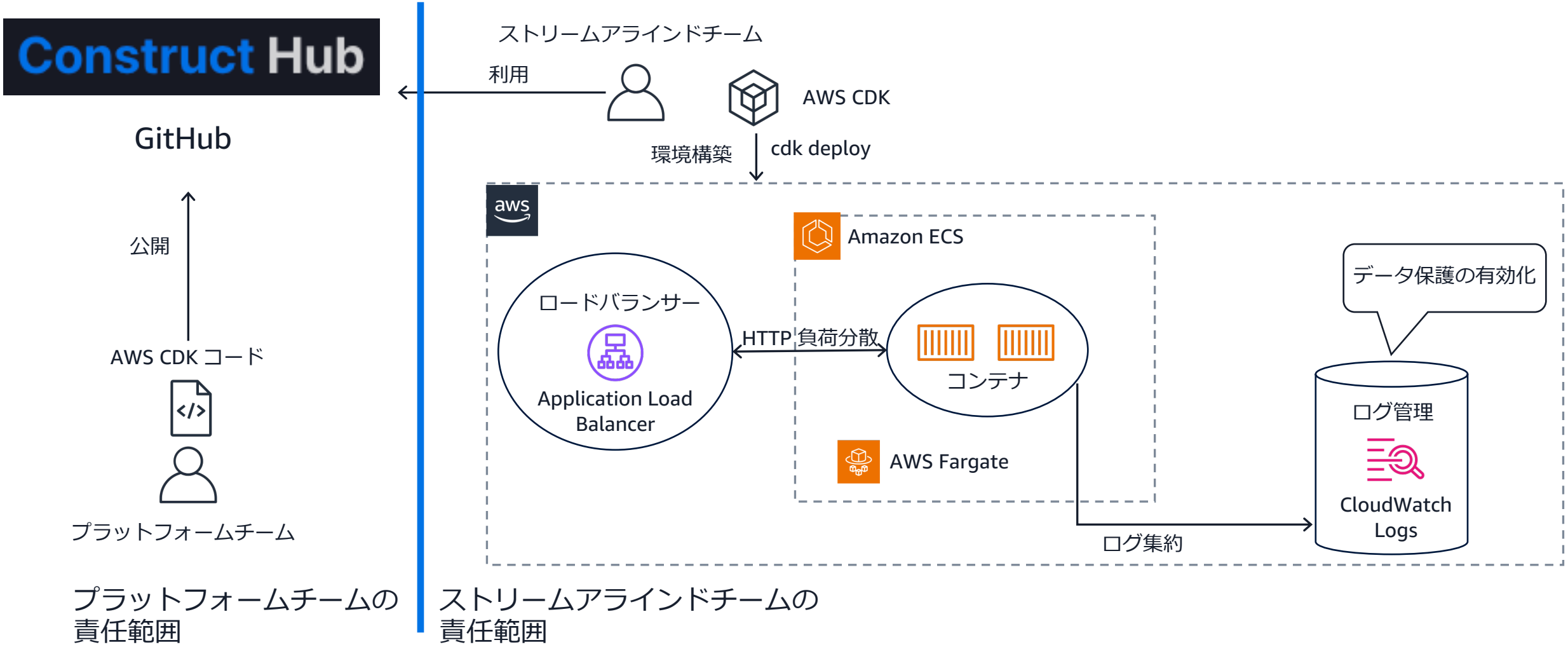
トレーニング、アーキテクチャレビュー、IaCサンプルコードの提供など

```
// アプリケーションのログを保存する場所
const dataProtectionPolicy = new logs.DataProtectionPolicy({
  identifiers: [
    logs.DataIdentifier.IPADDRESS, // IPアドレスを保護
    logs.DataIdentifier.EMAILADDRESS, // メールアドレスを保護
    logs.DataIdentifier.CREDITCARDNUMBER, // クレジットカード番号を保護
    new logs.CustomDataIdentifier('EmployeeId', 'EmployeeId-\\d{9}')], // 特定
});

const logGroup = new logs.LogGroup(this, 'LogGroupLambda', {
  logGroupName: 'TicketStore',
  dataProtectionPolicy: dataProtectionPolicy,
});

// AWS Fargate にコンテナをデプロイし、Application Load Balancer で負荷分散する構成を
const loadBalancedFargateService = new ecsPatterns.ApplicationLoadBalancedFa
  cluster,
  desiredCount: 2,
```

AWS Cloud Development Kit (AWS CDK) でのサンプルコード例

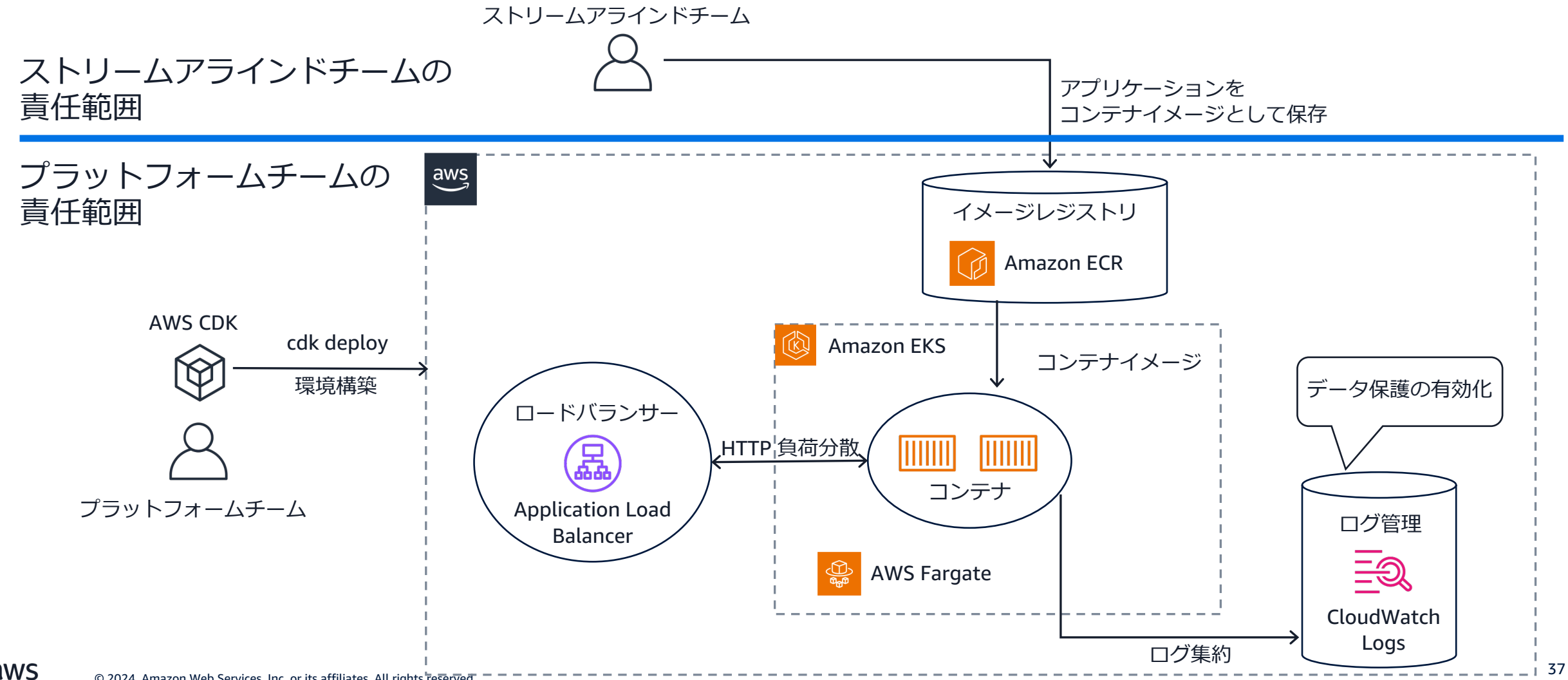
有用なソリューションや IaC コードの配布

フィードバックの内容を受けて有用なソリューションを配布し、適用するチームを広げる



中央集権型のプラットフォームモデル

統制を強化できるが、チームやワークロードの種類が増えるとプラットフォームチームの負荷が課題になりやすい

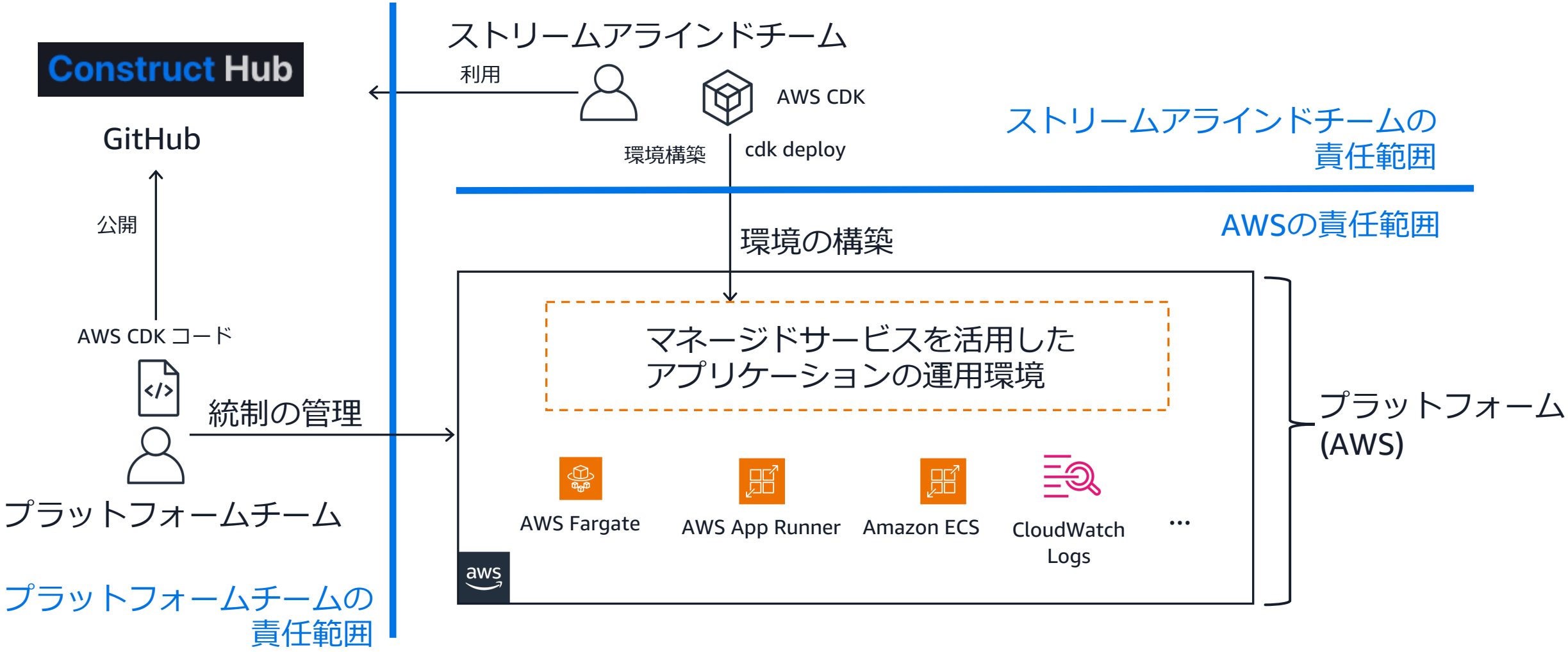


プラットフォームのモデル

- 分散型のプラットフォーム
  - ドキュメント、トレーニング、レビュー、サンプルコードの提供
  - IaC コード、ライブラリの配布
  - ストリームアラインドチームがインフラストラクチャーを管理
- 中央集権型のプラットフォーム
  - 独自のプラットフォームを構築して運用する
  - プラットフォームチームがインフラストラクチャーを管理

分散型のプラットフォーム

AWS はそれ自体がプラットフォーム – プラットフォームチームはベストプラクティスと統制の管理に集中する



中央集権型のプラットフォーム

AWS 上に、プラットフォームのレイヤーを設けて独自の運用を行う

プラットフォームは開発チームの「邪魔にならない」ようにする。開発チームが開発する上での前提条件をなるべく少なくするのだ。新しい開発者がプラットフォームを使い始めるのがどれだけ簡単かどうかは、DevExの達成状況についての良いテストになる

チームトポロジー　価値あるソフトウェアをすばやく届ける適応型組織設計

まとめ

- プラットフォームエンジニアリングはクラウドネイティブなチームが前提
- 最小限のプラットフォームから始める
- プラットフォームの種類
  - 分散型のプラットフォーム
    - AWS をプラットフォームとして捉える
  - 中央集権型のプラットフォーム
    - 独自のプラットフォームレイヤーを設けて管理する

Thank you!

Masatoshi Hayashi

hayshim@amazon.co.jp