AWS DEV DAY Online Japan

Day 1 / B-2

AWS におけるアプリチームとインフラチームのコワーク設計

株式会社野村総合研究所
新井 雅也

テックリード / インフラチームリーダー

新井 雅也 @msy78

Masaya ARAI

1987年生まれ。北海道出身。

2012年、野村総合研究所に入社。

金融業界のお客様に向けたビジネス提案やシステム設計、開発、運用を担当。

UI/UXデザインやスマホApp、サーバサイドAppなどフルスタックな

守備範囲を持ちつつ、クラウドアーキテクチャの設計と開発が得意。

業務以外においても、登壇や寄稿、AWSコミュニティ運営など幅広く活動中。

最近の趣味はビザールプランツ集めとそのお世話。

APN Ambassador 2021、 AWS Well-Architected Lead

プレゼンのゴール

本日は、AWSを通じたアプリチームとインフラチームのコワークのあり方について
実際の金融クレジットカード決済系ビジネスの立ち上げからシステム構築経験を基に
一つのプラクティスとしてお伝えします。

すべてのユースケースおける正解・ベストプラクティスではないですが、
チームで設計や構築を協力・調停していくために一つの考え方として、
皆さまのヒントになれば幸いです。

※**インフラとアプリ開発の役割・責任がチームで分かれている前提のお話**となります。

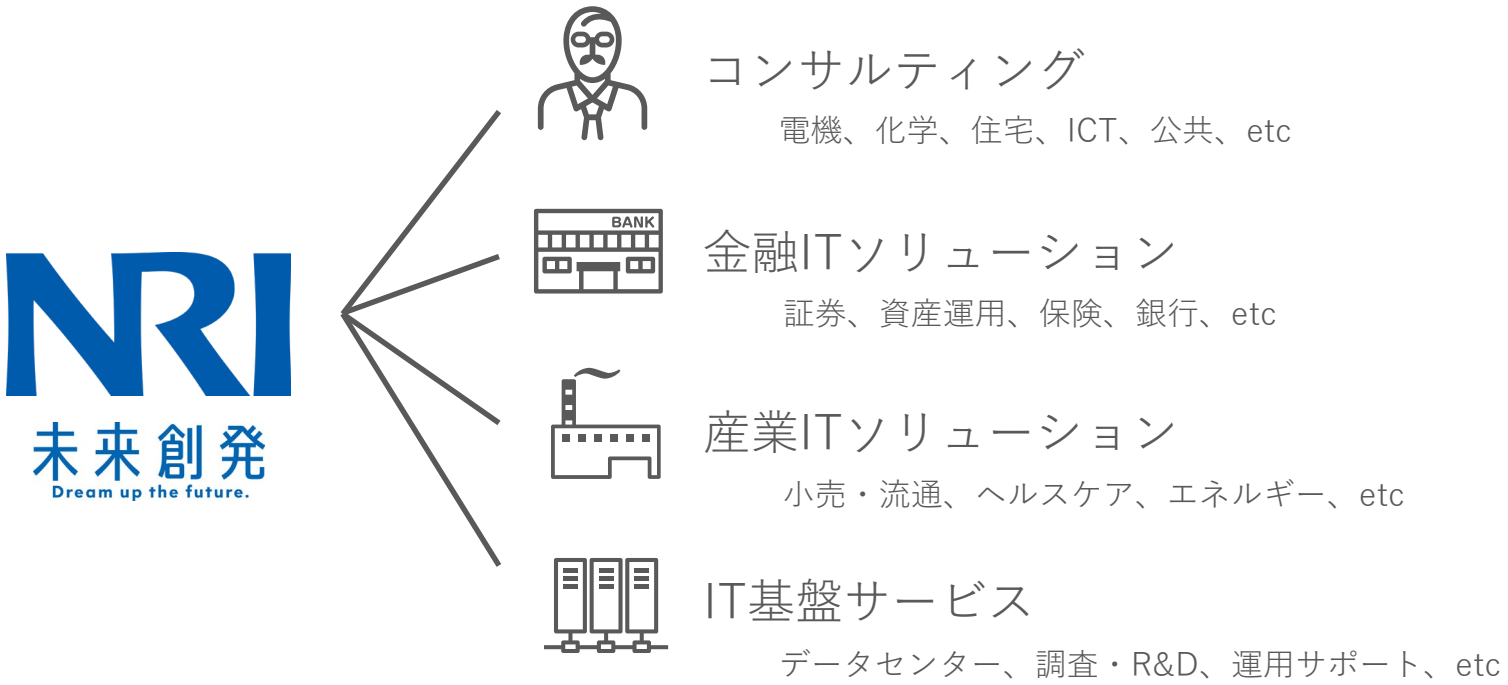
本日のアジェンダ

1. プロジェクトの背景とお客様の特徴
2. サービスを実現するための要件と解決すべき課題
3. チーム編成とそれぞれの役割
4. AWSにおけるチームのコワーク設計
5. チームとしてAWS開発を推し進めるために
6. まとめ

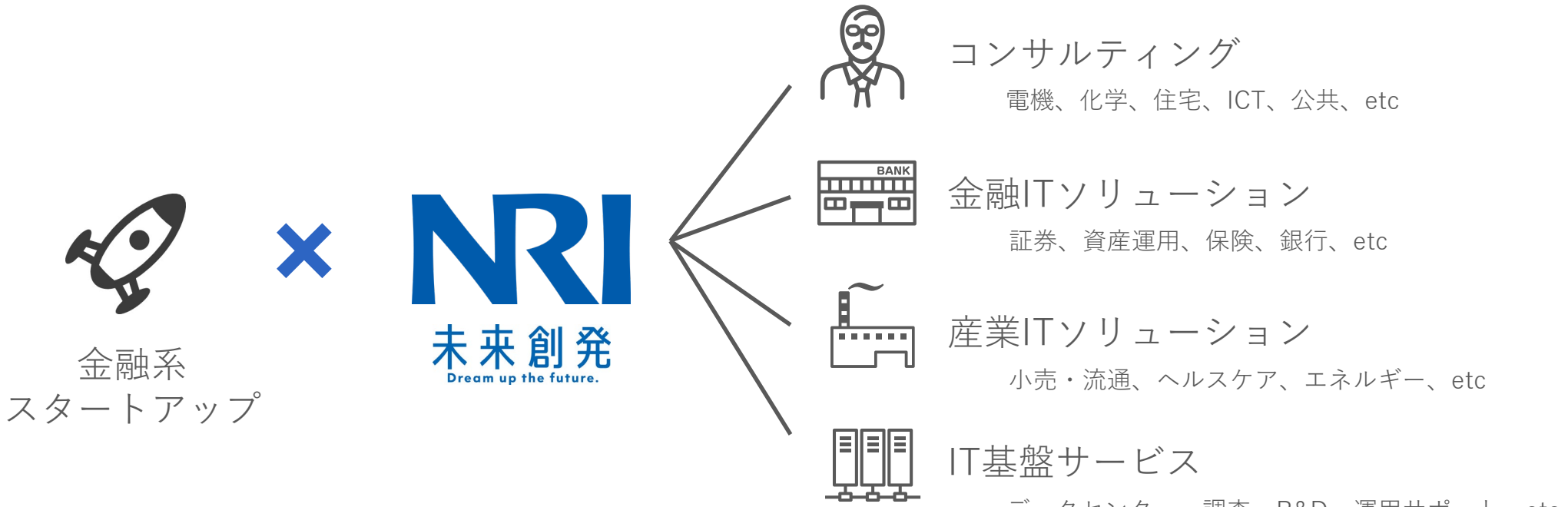
01.　プロジェクトの背景とお客様の特徴

NRIという会社のかんたんなご紹介

NRIという会社のかんたんなご紹介



NRIでは、エンタープライズだけでなくスタートアップのお客様もご支援



NRIでは、エンタープライズだけでなくスタートアップのお客様もご支援

今回ご支援させていただいたお客様は金融系のスタートアップ企業。
クレジットカード✕スマホアプリを主軸にしたFintechサービスであり、ビジネスのスキーム作りから一緒に参画。

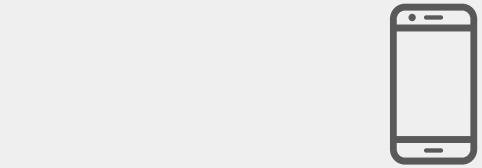
※お客様からは許可を頂いた上、本発表のテーマとして利用させていただいています。

お客様ビジネスのご紹介（本プレゼンのテーマ）

スマホアプリのバックエンド機能はAWSにてシステムを構築。

- 会員申込(審査)
- 決済の管理
- 明細の表示
- 返済の管理

金融系
スタートアップ



スマホアプリの
バックエンド機能や
管理者機能を提供

※お客様からは許可を頂いた上、本発表のテーマとして利用させていただいています。

お客様ビジネスのご紹介：会員申込

スマホアプリから住所入力や本人確認書類をアップロードなど数ステップすることで申し込みが完了。
審査が通過すれば、数日でクレジットカードが受け取れる。

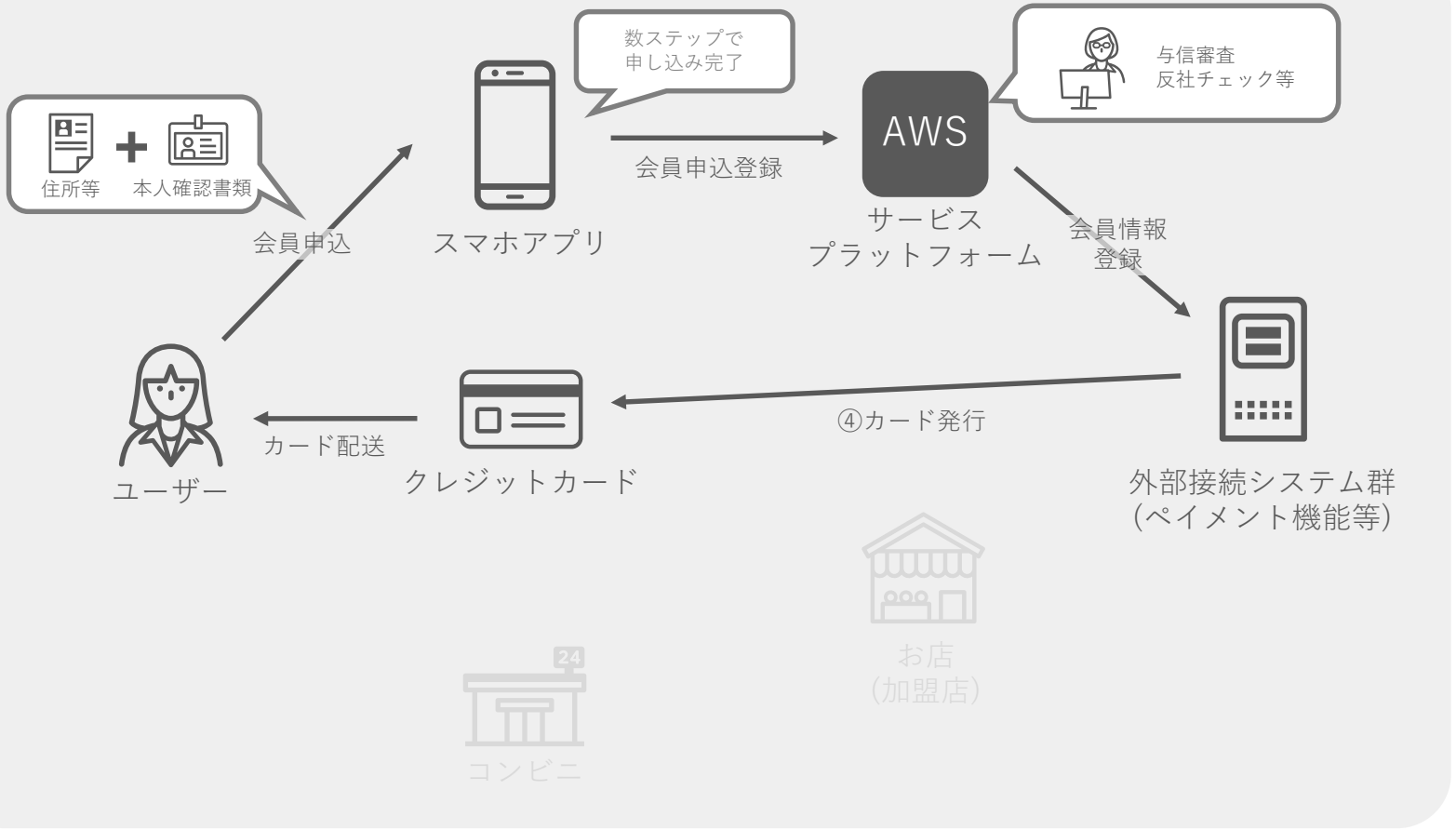
会員申込(審査)

決済の管理

明細の表示

返済の管理

金融系
スタートアップ



※お客様からは許可を頂いた上、本発表のテーマとして利用させていただいています。

お客様ビジネスのご紹介：決済の管理

クレジットカードを利用してお店（加盟店）で商品を購入すると、外部システム側に決済情報が蓄積。
AWSで構成されたプラットフォームがその情報を取得し、利用者に決済の旨を通知。

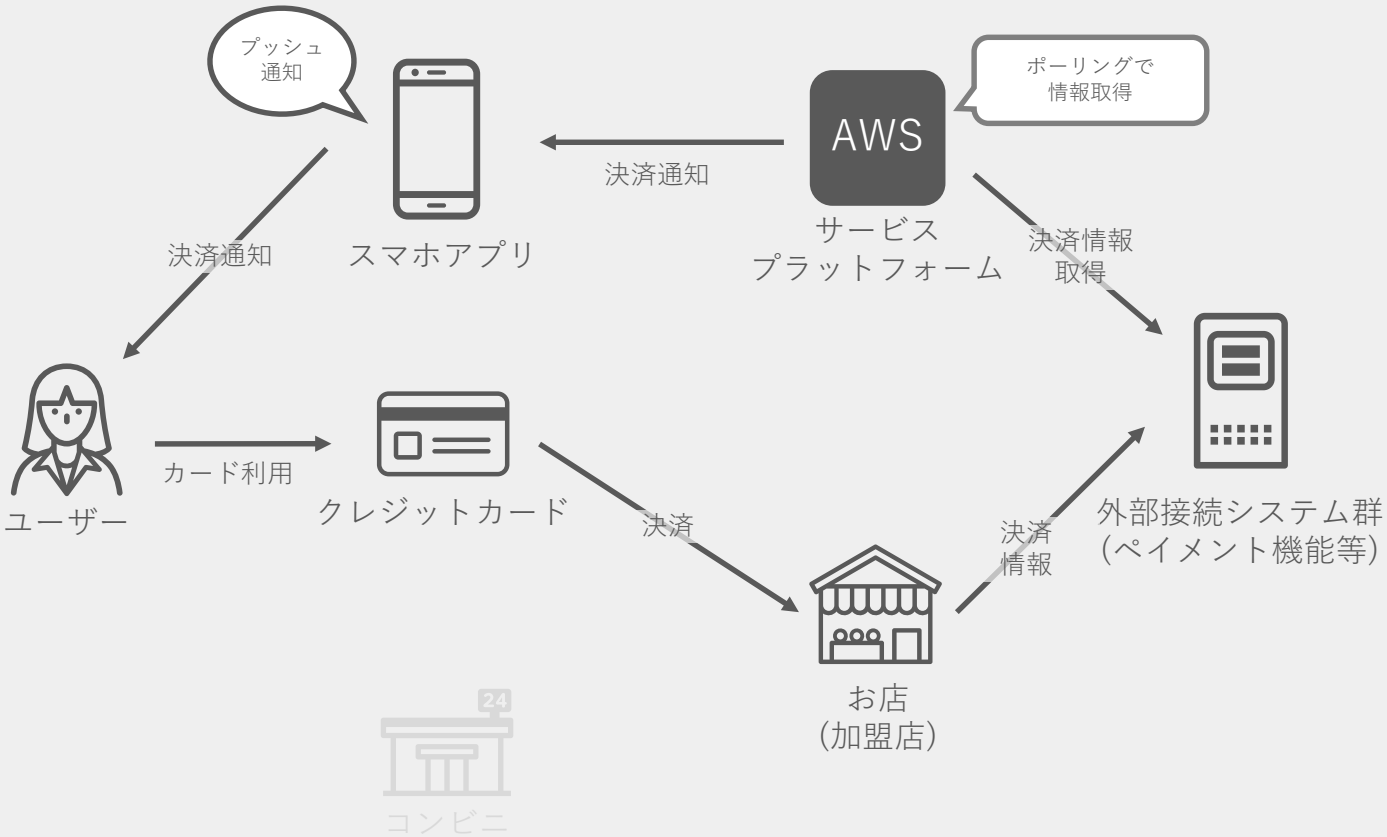
会員申込(審査)

決済の管理

明細の表示

返済の管理

金融系
スタートアップ



※お客様からは許可を頂いた上、本発表のテーマとして利用させていただいています。

お客様ビジネスのご紹介：明細の表示

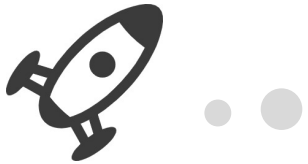
決済が確定後、利用明細情報（キャシュフロー情報）が外部システムからAWSサービスプラットフォームに連携。
ユーザーはスマホアプリから各種明細を確認できる。

会員申込(審査)

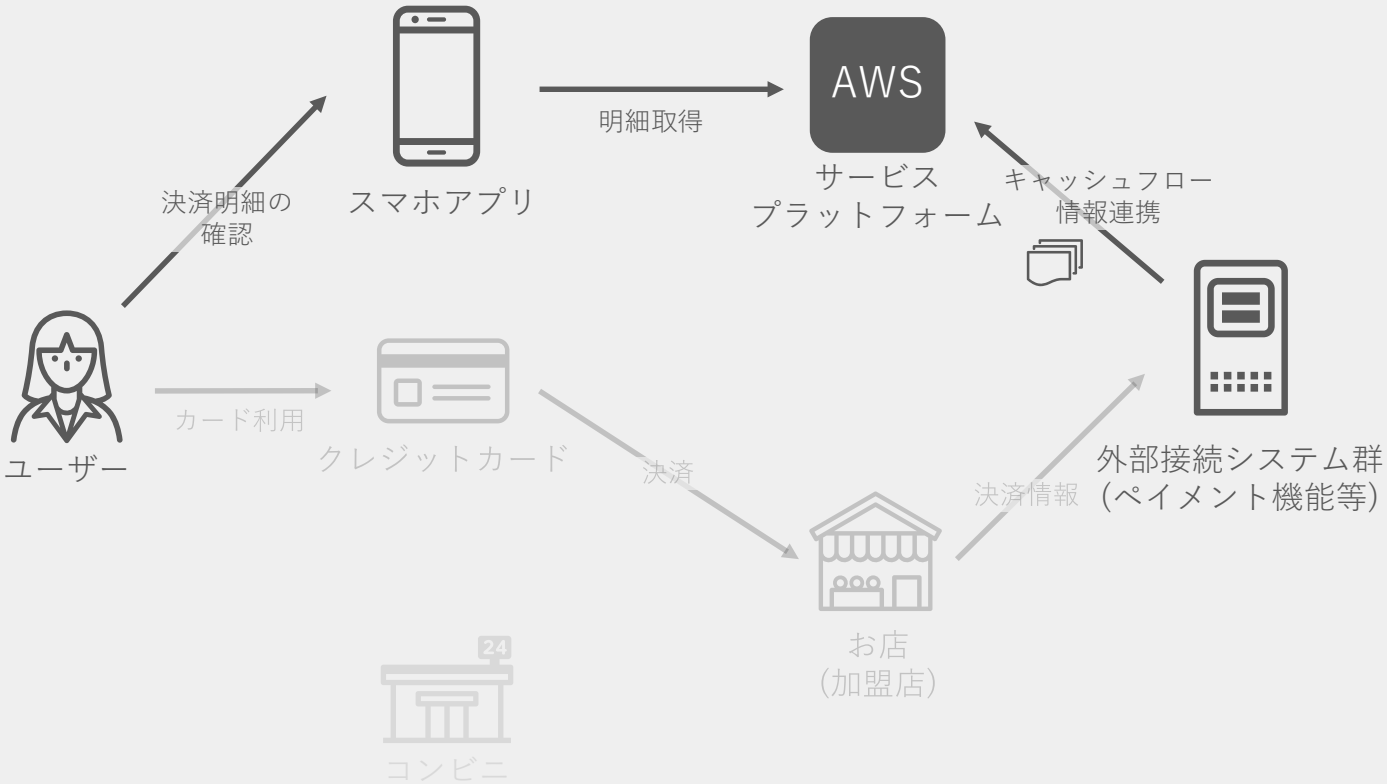
決済の管理

明細の表示

返済の管理



金融系
スタートアップ



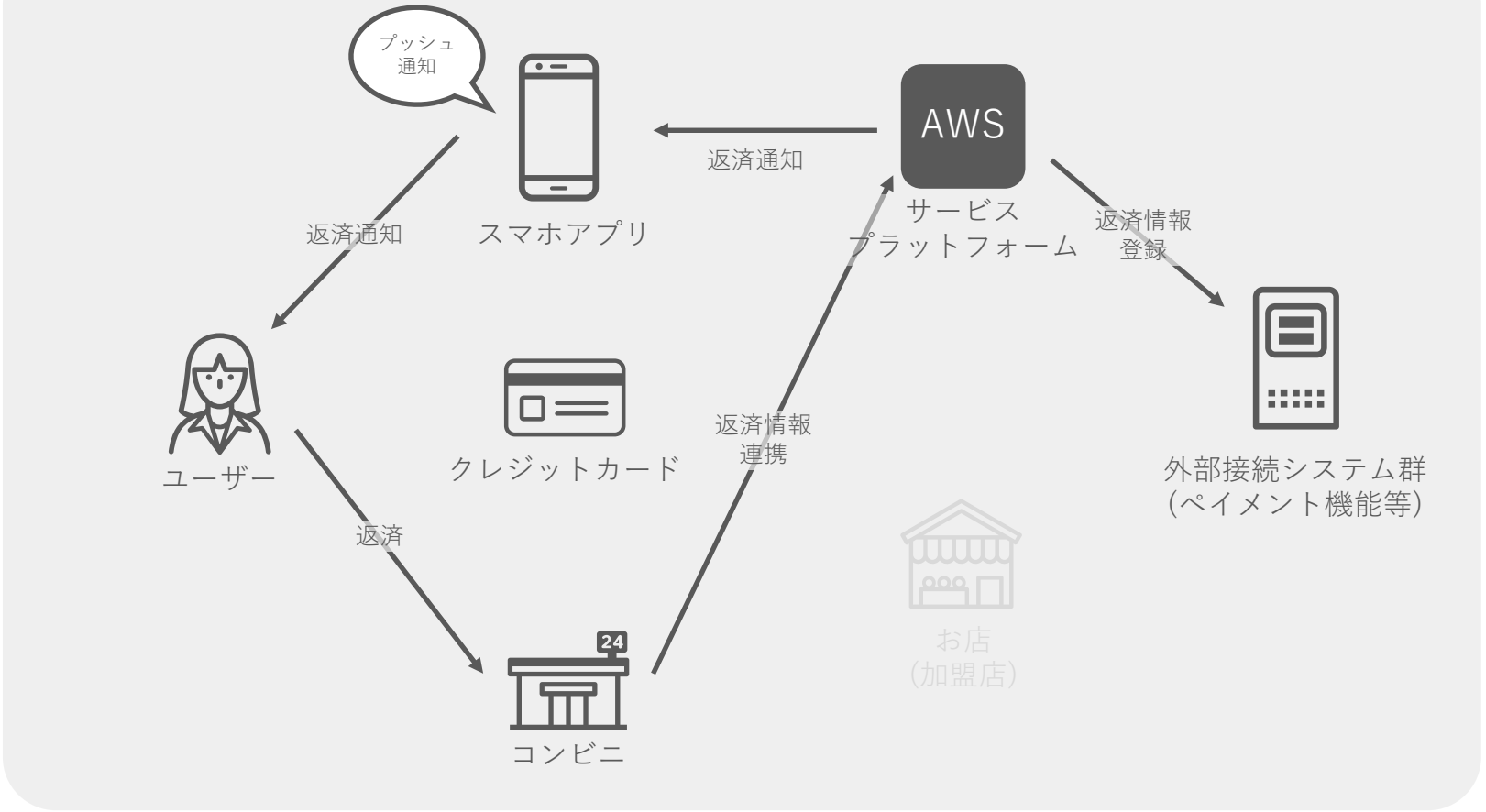
※お客様からは許可を頂いた上、本発表のテーマとして利用させていただいています。

お客様ビジネスのご紹介：返済の管理

コンビニのATMで好きなタイミングに返済が可能。
QRコードを読み取り、金額をATMに投入すると、AWSサービスプラットフォームに蓄積され、スマホアプリに通知。

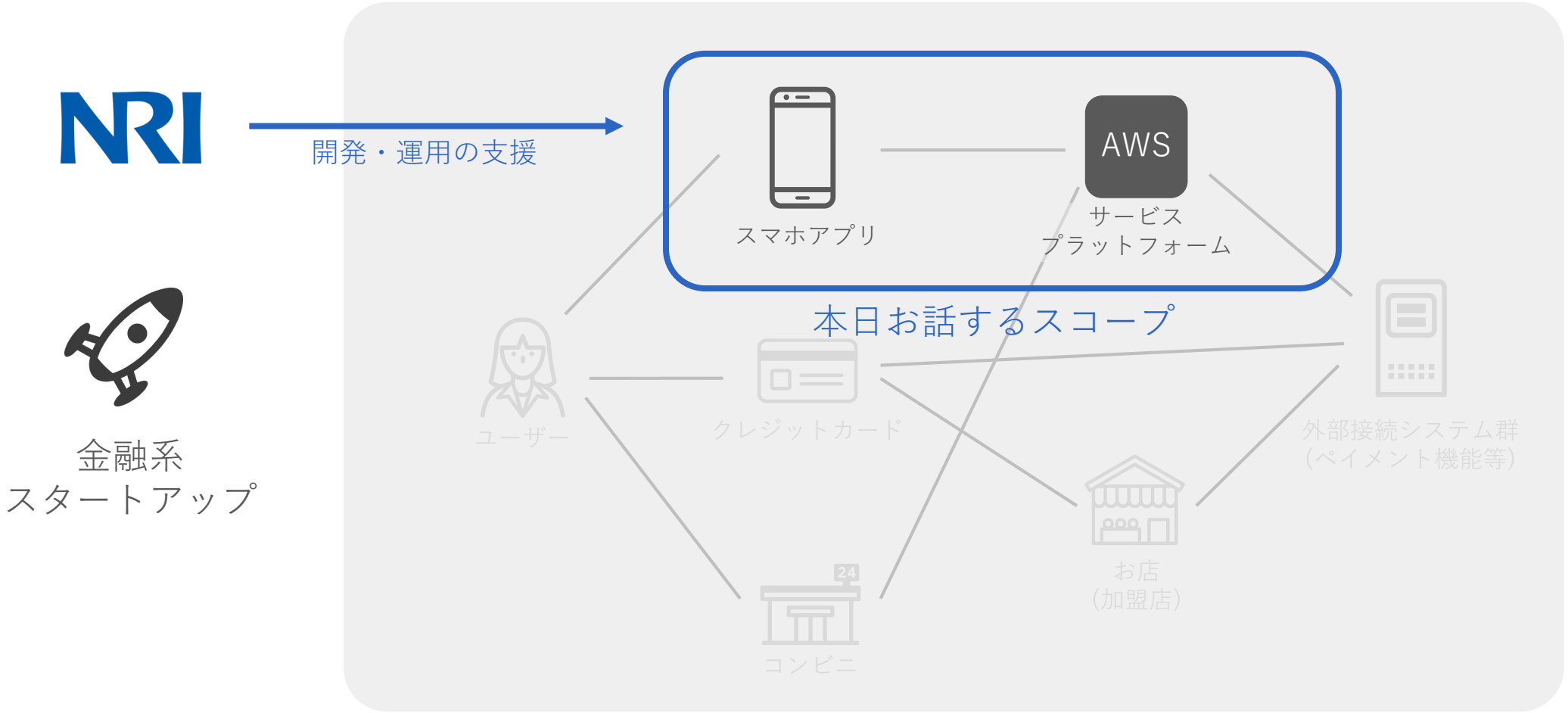
会員申込(審査)
決済の管理
明細の表示
返済の管理

金融系
スタートアップ



※お客様からは許可を頂いた上、本発表のテーマとして利用させていただいています。

NRIはスマホアプリ+AWSサービスプラットフォームの開発・運用をご支援



※お客様からは許可を頂いた上、本発表のテーマとして利用させていただいています。

02.　サービスを実現するための要件と解決すべき課題

利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。

UI/UXを高めるための
リリースサイクル確立

複雑な業務に対する
アプリ方式の実装

外部接続システム含めた
バッチ処理の実装

イシュアとしてのPCIDSS 12要件 331項目への対応方針検討とセキュリティ実装

利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。

UI/UXを高めるための
リリースサイクル確立

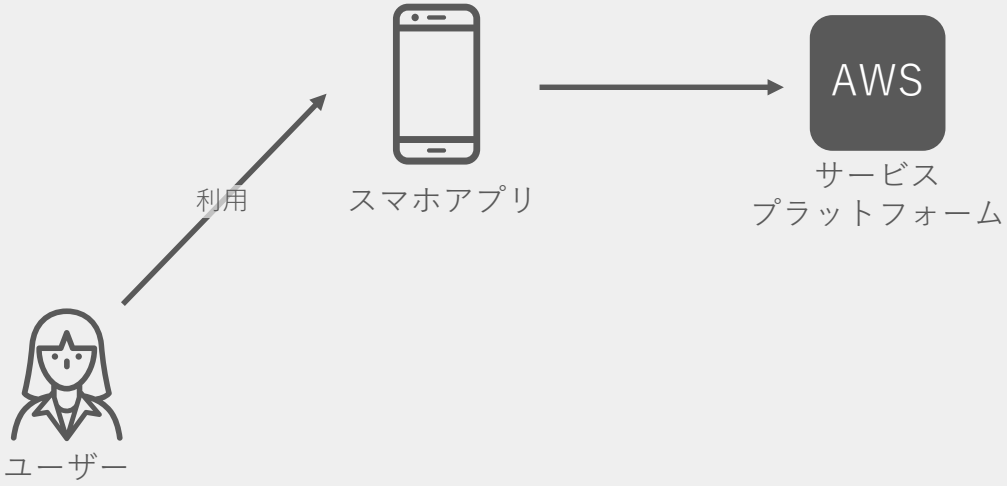
複雑な業務に対する
アプリ方式の実装

外部接続システム含めた
バッチ処理の実装

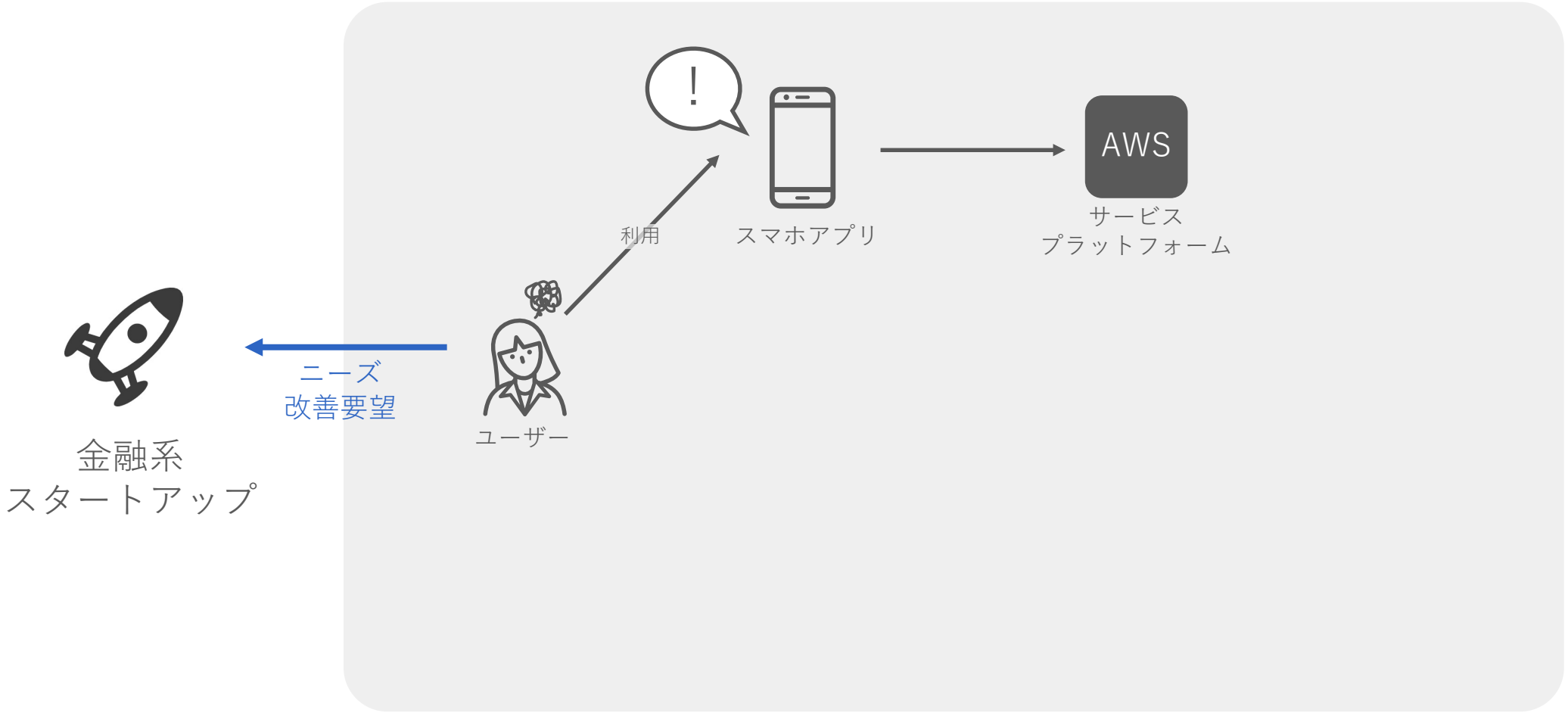
イシュアとしてのPCIDSS 12要件 331項目への対応方針検討とセキュリティ実装

スマホアプリのユーザーより得られるフィードバックやニーズから、素早くUI/UXを改善できるリリースの仕組みが不可欠。

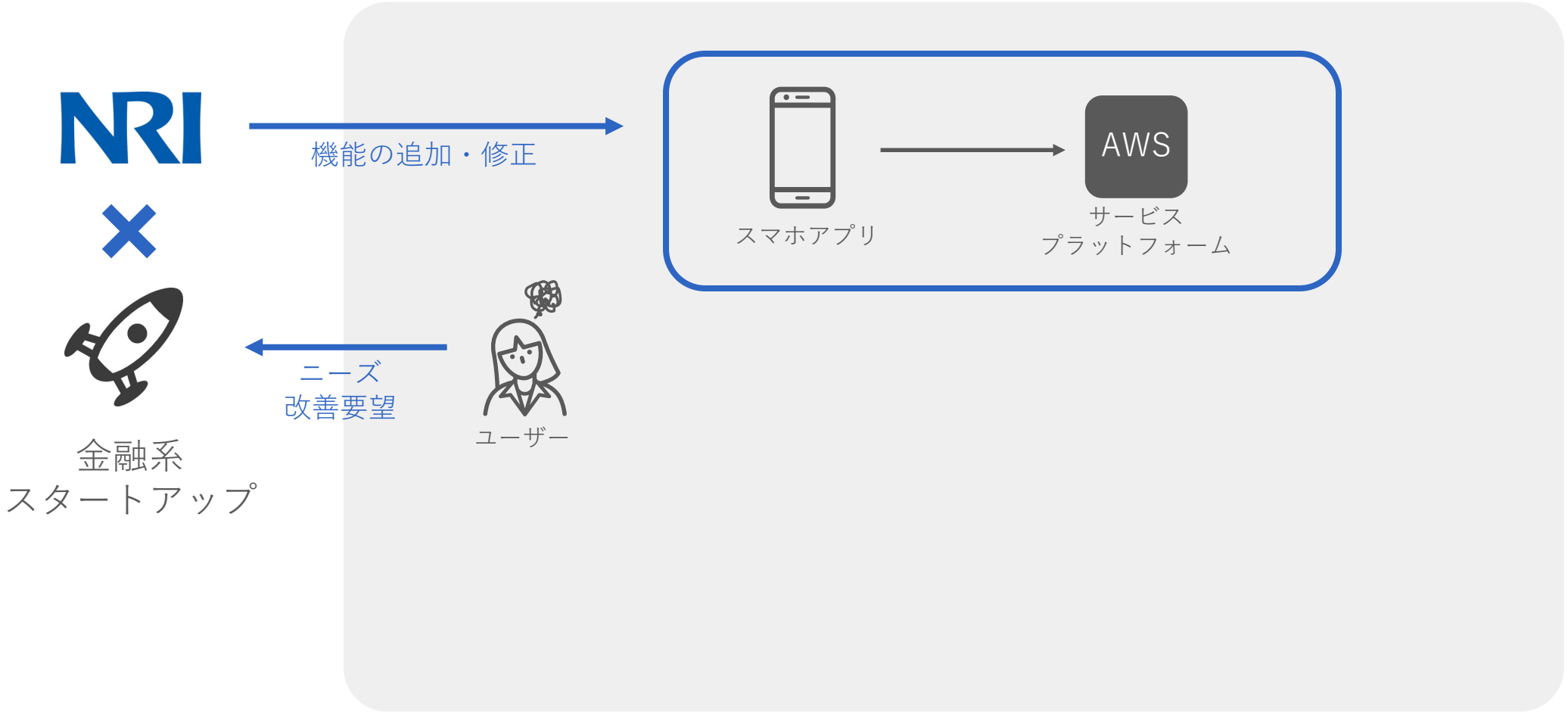
金融系
スタートアップ



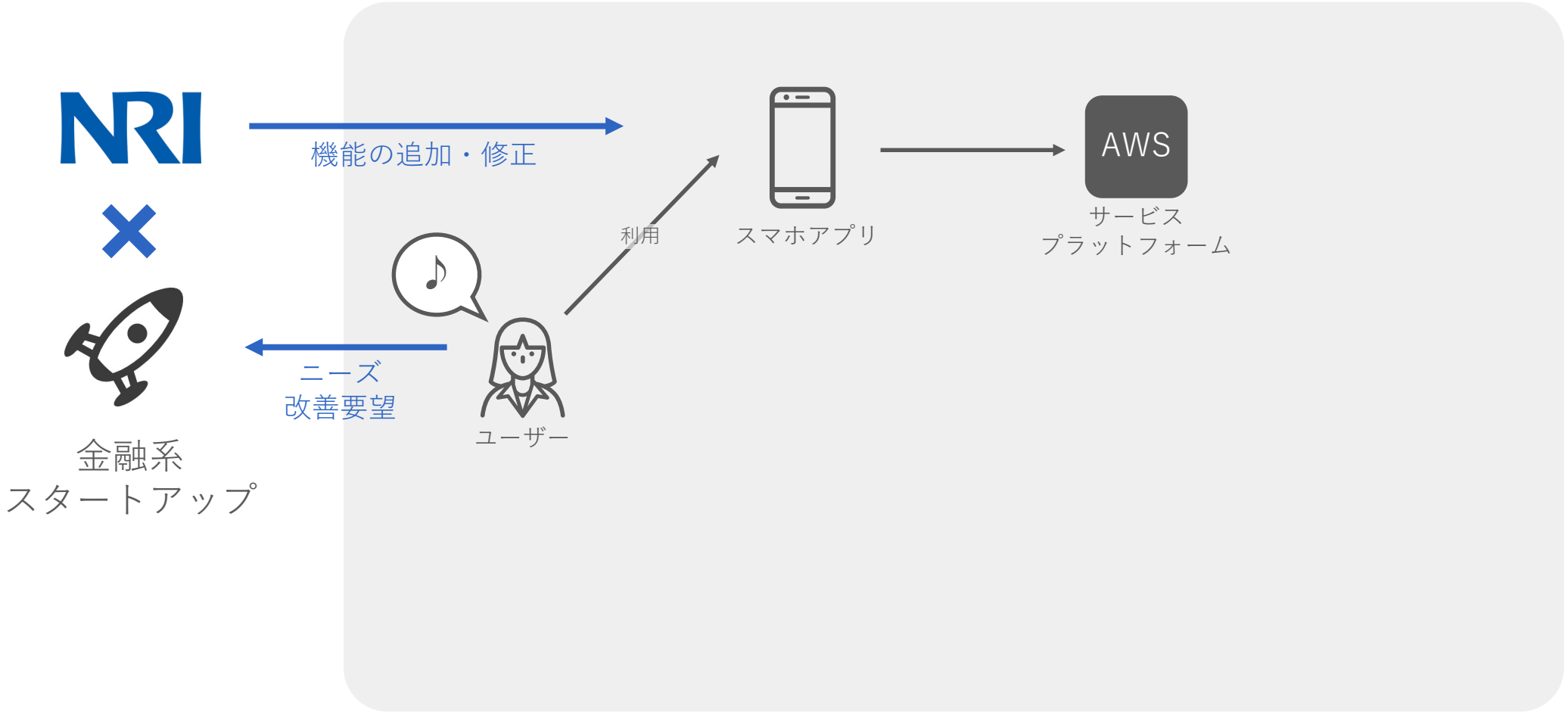
スマホアプリのユーザーより得られるフィードバックやニーズから、素早くUI/UXを改善できるリリースの仕組みが不可欠。



スマホアプリのユーザーより得られるフィードバックやニーズから、素早くUI/UXを改善できるリリースの仕組みが不可欠。



スマホアプリのユーザーより得られるフィードバックやニーズから、素早くUI/UXを改善できるリリースの仕組みが不可欠。



利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。

UI/UXを高めるための
リリースサイクル確立

複雑な業務に対する
アプリ方式の実装

外部接続システム含めた
バッチ処理の実装

イシュアとしてのPCIDSS 12要件 331項目への対応方針検討とセキュリティ実装

利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。

UI/UXを高めるための
リリースサイクル確立

**複雑な業務に対する
アプリ方式の実装**

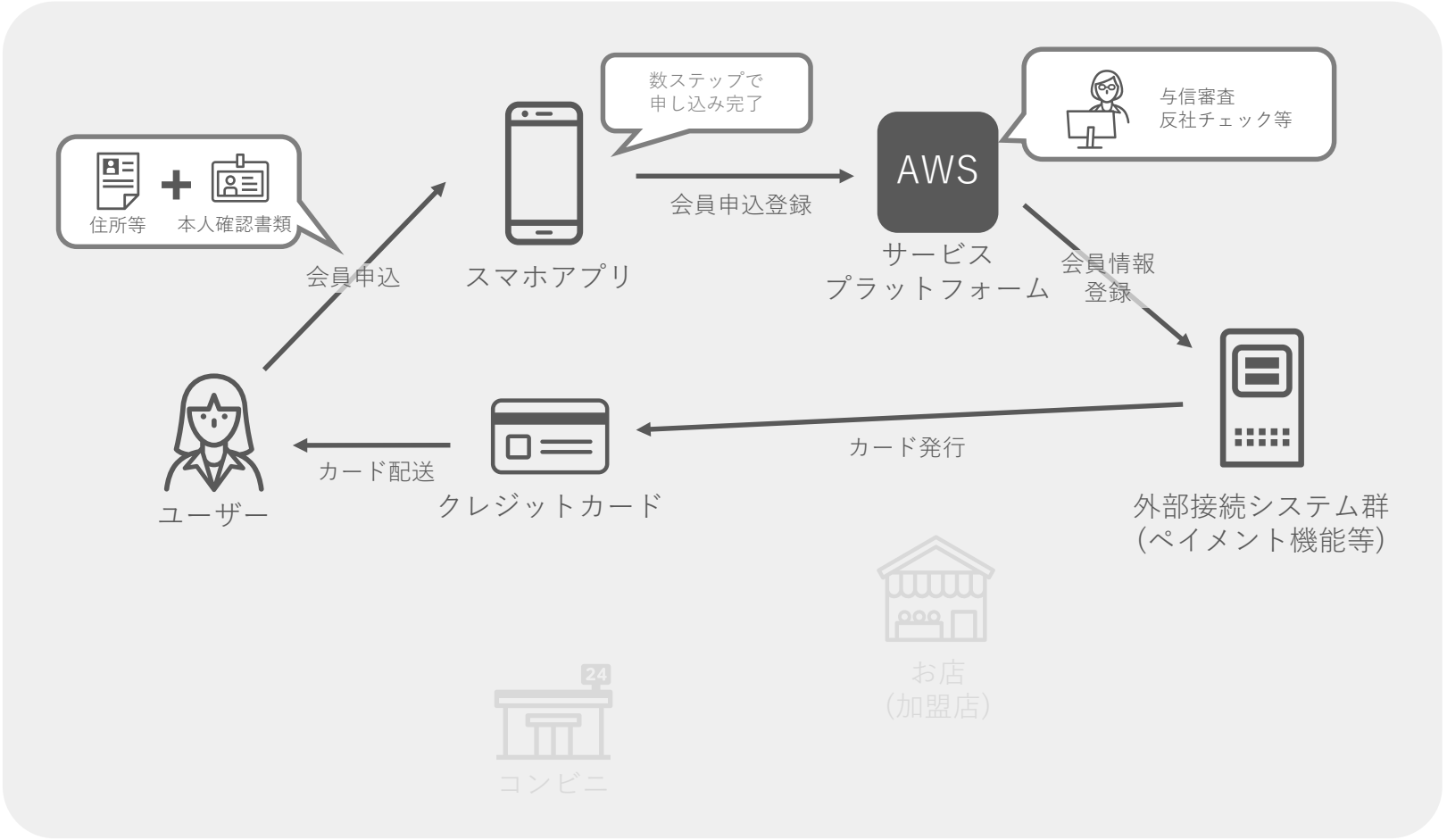
外部接続システム含めた
バッチ処理の実装

イシュアとしてのPCIDSS 12要件 331項目への対応方針検討とセキュリティ実装

再掲

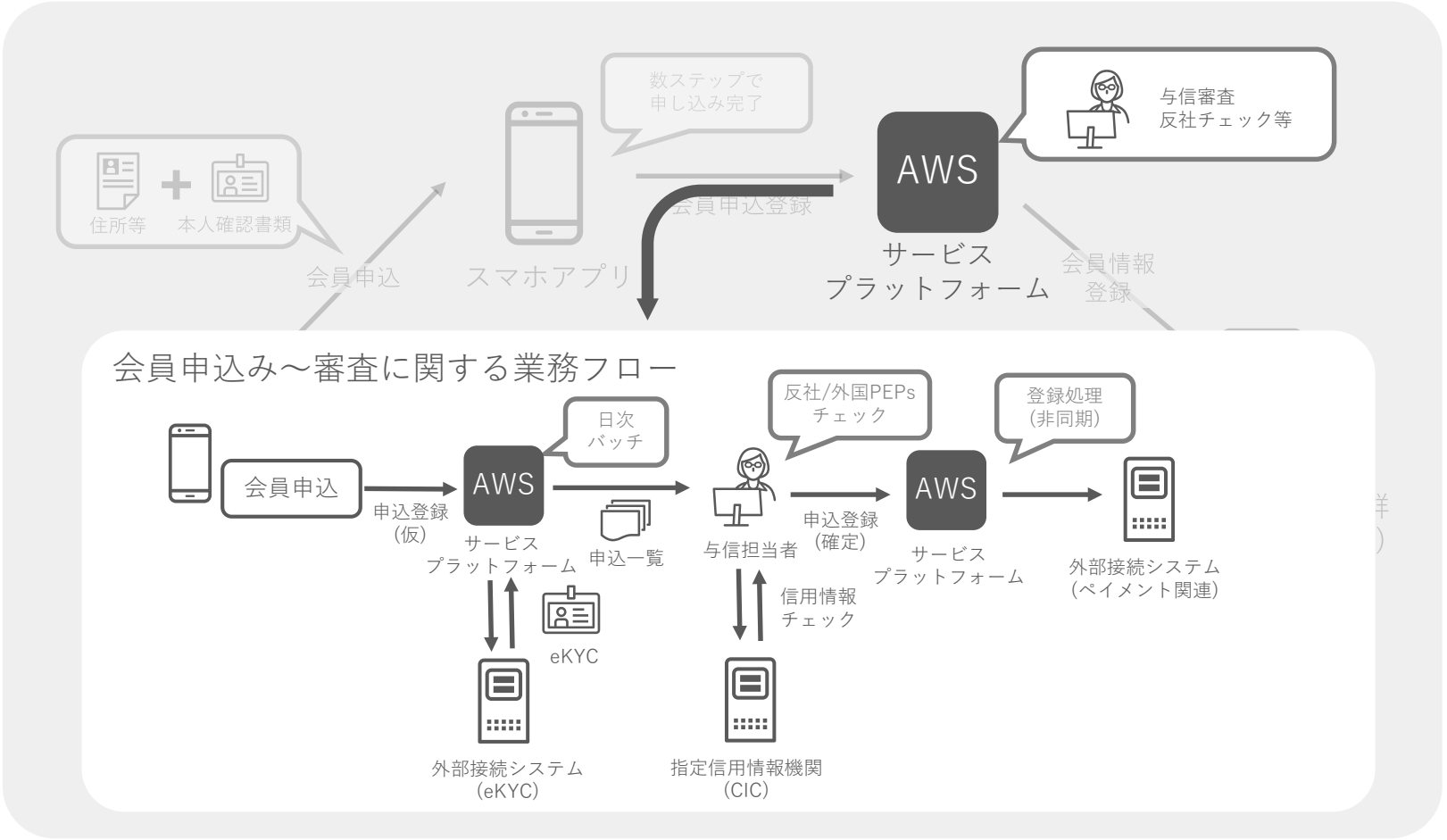
入会～審査を始めとしたクレジットカード特有の複雑な業務を紐解き、方式の検討から実装までを検討する必要がある。

金融系
スタートアップ



入会～審査を始めとしたクレジットカード特有の複雑な業務を紐解き、方式の検討から実装までを検討する必要がある。

金融系
スタートアップ



利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。

UI/UXを高めるための
リリースサイクル確立

**複雑な業務に対する
アプリ方式の実装**

外部接続システム含めた
バッチ処理の実装

イシュアとしてのPCIDSS 12要件 331項目への対応方針検討とセキュリティ実装

利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。

UI/UXを高めるための
リリースサイクル確立

複雑な業務に対する
アプリ方式の実装

外部接続システム含めた
バッチ処理の実装

イシュアとしてのPCIDSS 12要件 331項目への対応方針検討とセキュリティ実装

再掲

外部システム側から連携されるファイルを取り込む関係上、いくつかのバッチ処理を実現しなければならない。

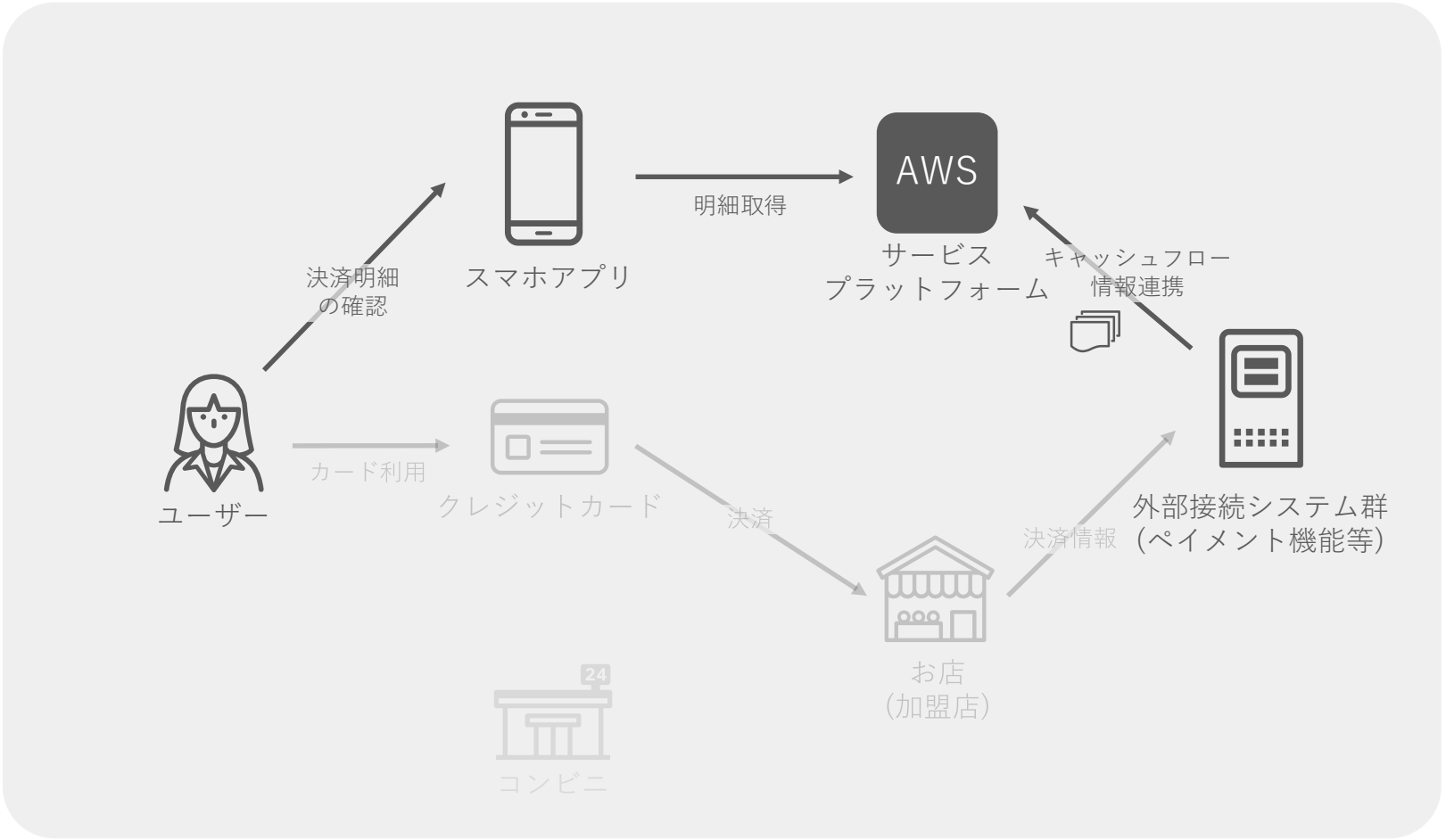
会員申込(審査)

決済の管理

明細の表示

返済の管理

金融系
スタートアップ



※お客様からは許可を頂いた上、本発表のテーマとして利用させていただいています。

外部システム側から連携されるファイルを取り込む関係上、いくつかのバッチ処理を実現しなければならない。

会員申込(審査)

決済の管理

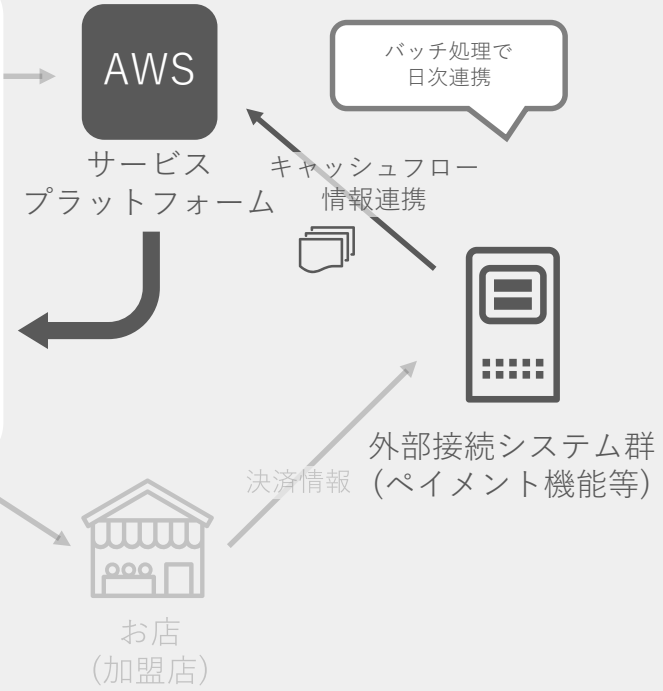
明細の表示

返済の管理

金融系
スタートアップ

バッチ処理

・キャッシュフローに関する取り込み
・与信用申込一覧の作成
・ユーザーの利息発生状況通知
・金利計算
...etc



利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。

UI/UXを高めるための
リリースサイクル確立

複雑な業務に対する
アプリ方式の実装

外部接続システム含めた
バッチ処理の実装

イシュアとしてのPCIDSS 12要件 331項目への対応方針検討とセキュリティ実装

利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。

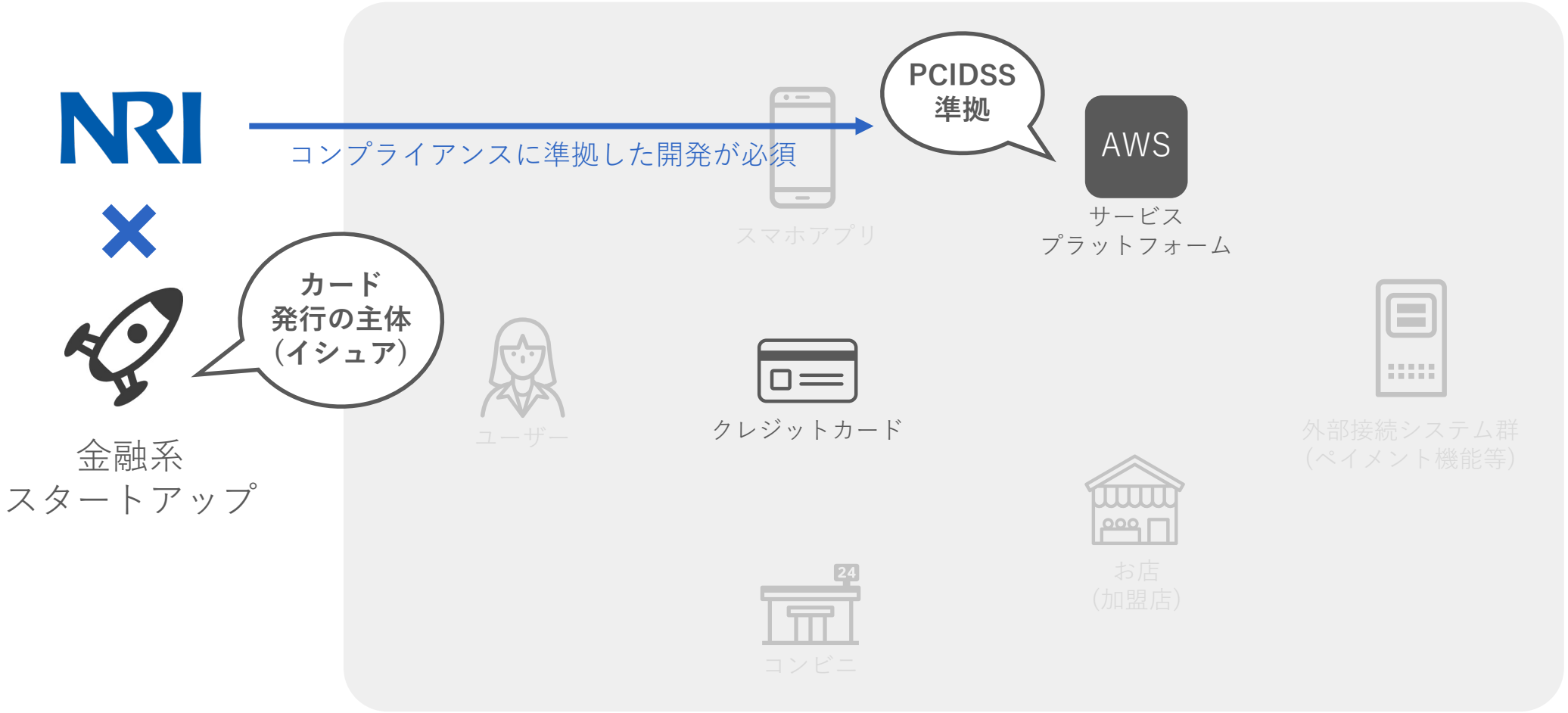
UI/UXを高めるための
リリースサイクル確立

複雑な業務に対する
アプリ方式の実装

外部接続システム含めた
バッチ処理の実装

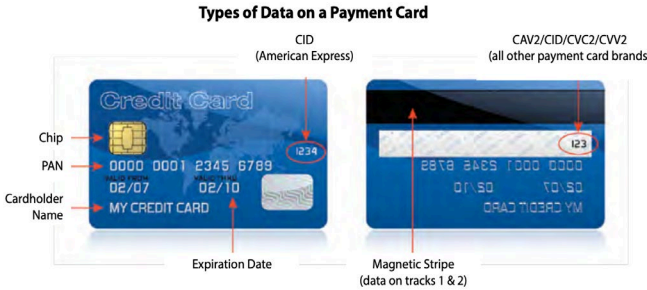
イシュアとしてのPCIDSS 12要件 331項目への対応方針検討とセキュリティ実装

クレジットカードに関するサービスを提供するためには
グローバルセキュリティ基準であるPCIDSSの準拠が求められる。



PCIDSSは12要件から成り、イシュア(カード発行事業者)は331項目の方針検討と準拠が必要

クレジットカード情報が「追加」「処理」「保存」される箇所がPCIDSSの規制対象となる。



安全なネットワークの構築維持	1.カード会員データを保護するために、ファイアウォールをインストールして維持する 2.システムパスワードおよびその他のセキュリティパラメータにベンダ提供のデフォルト値を使用しない
カード会員データの保護	3.保存されるカード会員データを保護する 4.オープンな公共ネットワーク経由でカード会員データを伝送する場合、暗号化する
脆弱性管理プログラムの維持	5.マルウェアにしてすべてのシステムを保護し、ウィルス対策ソフトウェアを定期的に更新する 6.安全性の高いシステムとアプリケーションを開発し、保守する
強力なアクセス制御手法の導入	7.カード会員データへのアクセスを、業務上必要な範囲内に制限する 8.システムコンポーネントへのアクセスを識別・認証する 9.カード会員データへの物理アクセスを制限する
ネットワークの定期的な監視およびテスト	10.ネットワークリソースおよびカード会員データへのすべてのアクセスを追跡および監視する 11.セキュリティシステムおよびプロセスを定期的にテストする
情報セキュリティポリシーの維持	12.すべての担当者の情報セキュリティに対応するポリシーを整備する

Ref. https://www.pcisecuritystandards.org/documents/PCI_DSS-QRG-v3_2_1.pdf

利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。

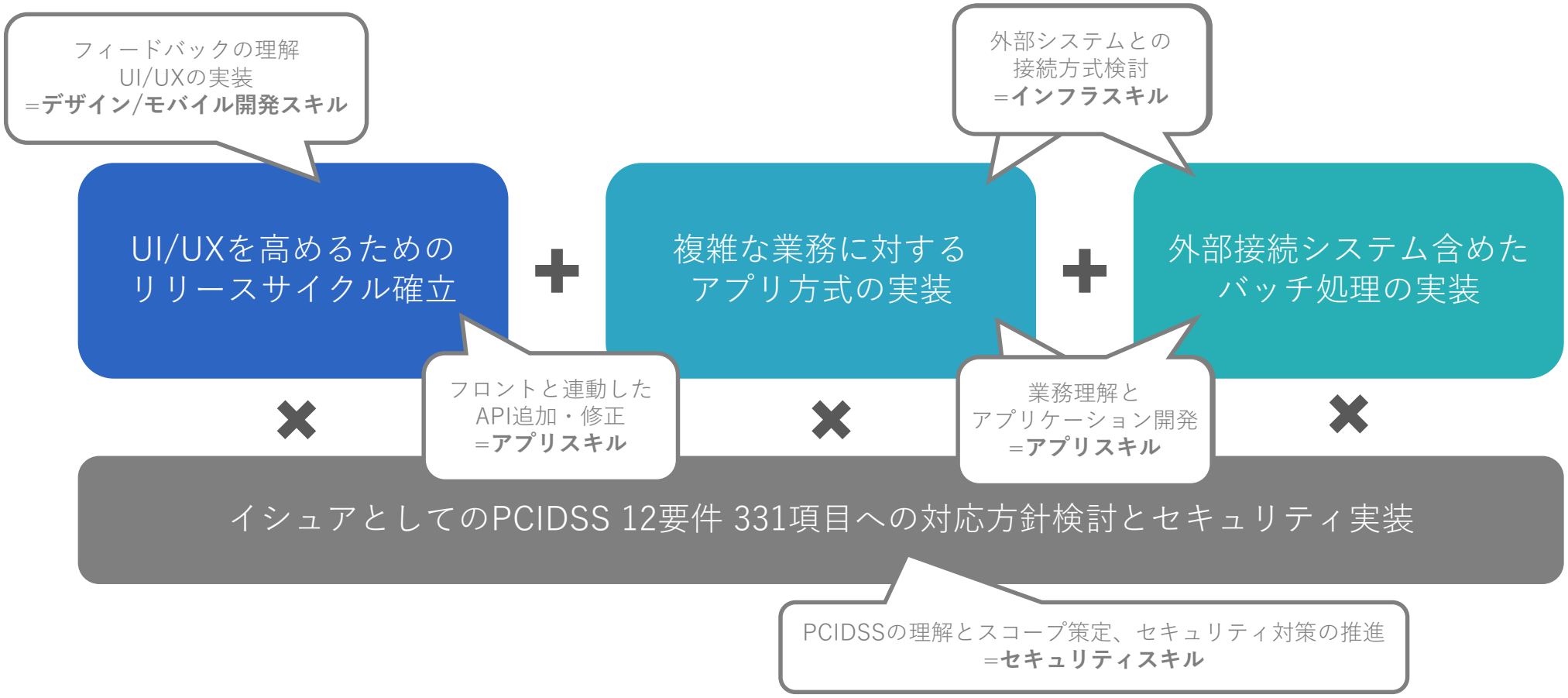
UI/UXを高めるための
リリースサイクル確立

複雑な業務に対する
アプリ方式の実装

外部接続システム含めた
バッチ処理の実装

イシュアとしてのPCIDSS 12要件 331項目への対応方針検討とセキュリティ実装

利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。



利用者に対するUI/UXの改善はもちろん、クレジットカードに対する深い業務への理解や厳格なコンプライアンスに準拠した環境構築が必須。

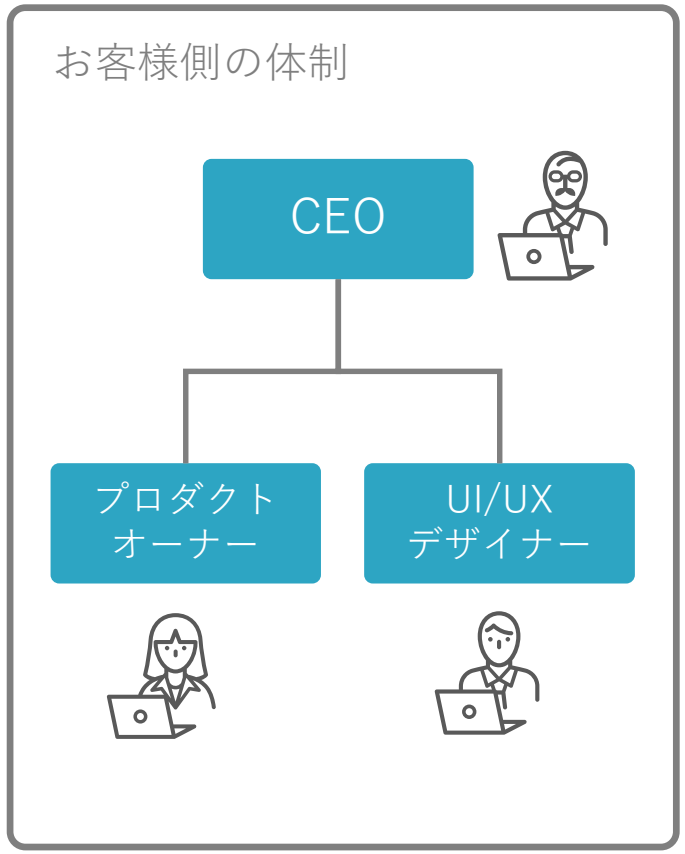
各領域において、PCIDSSにて要求されている項目への適切な準拠や、
対応に高度な専門性の要求が見込まれたため、
[各スキルを有するメンバーでチームを編成](#)する方針とした。

03.　チーム編成とそれぞれの役割

NRI側はアプリとインフラというそれぞれ専門性が異なるチームで構成

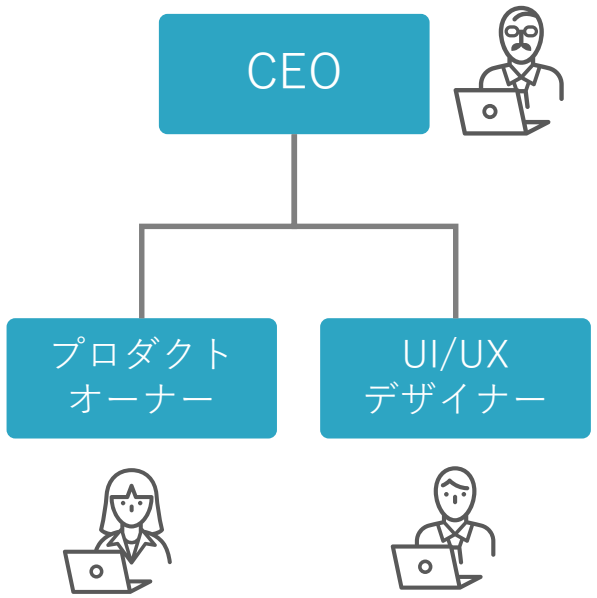
お客様側の体制

NRI側はアプリとインフラというそれぞれ専門性が異なるチームで構成



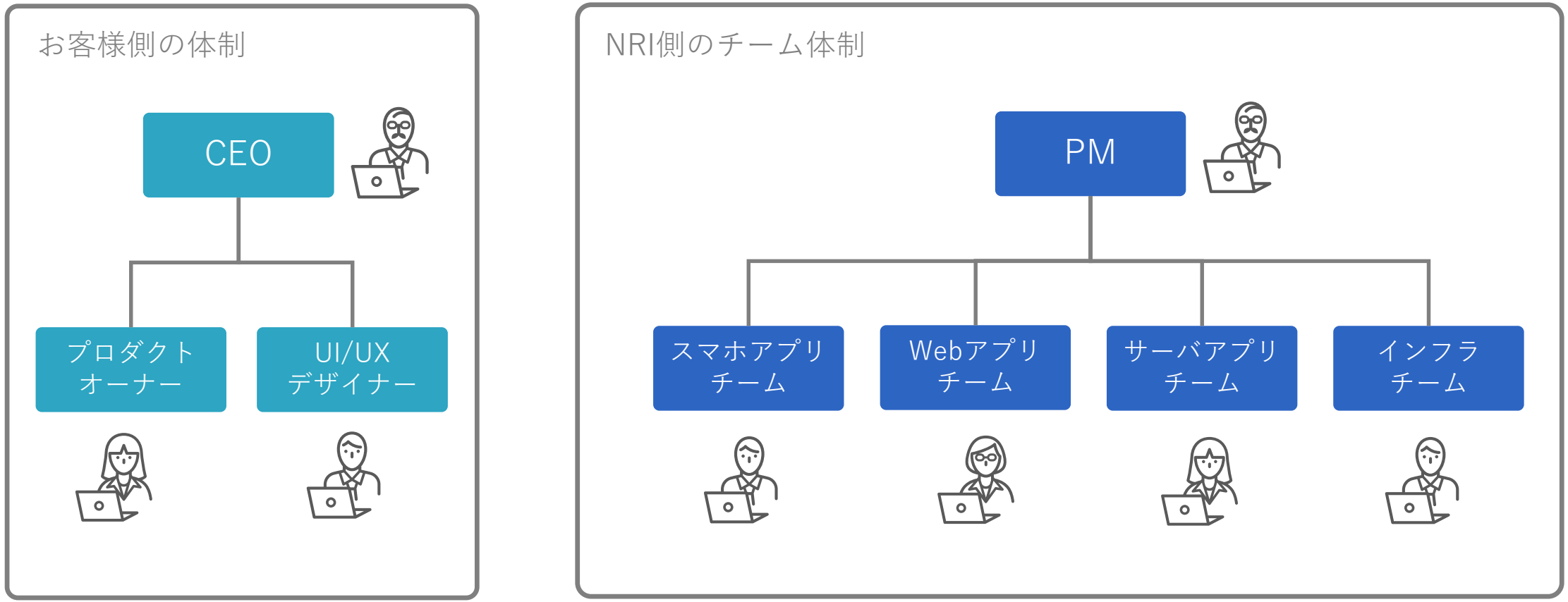
NRI側はアプリとインフラというそれぞれ専門性が異なるチームで構成

お客様側の体制

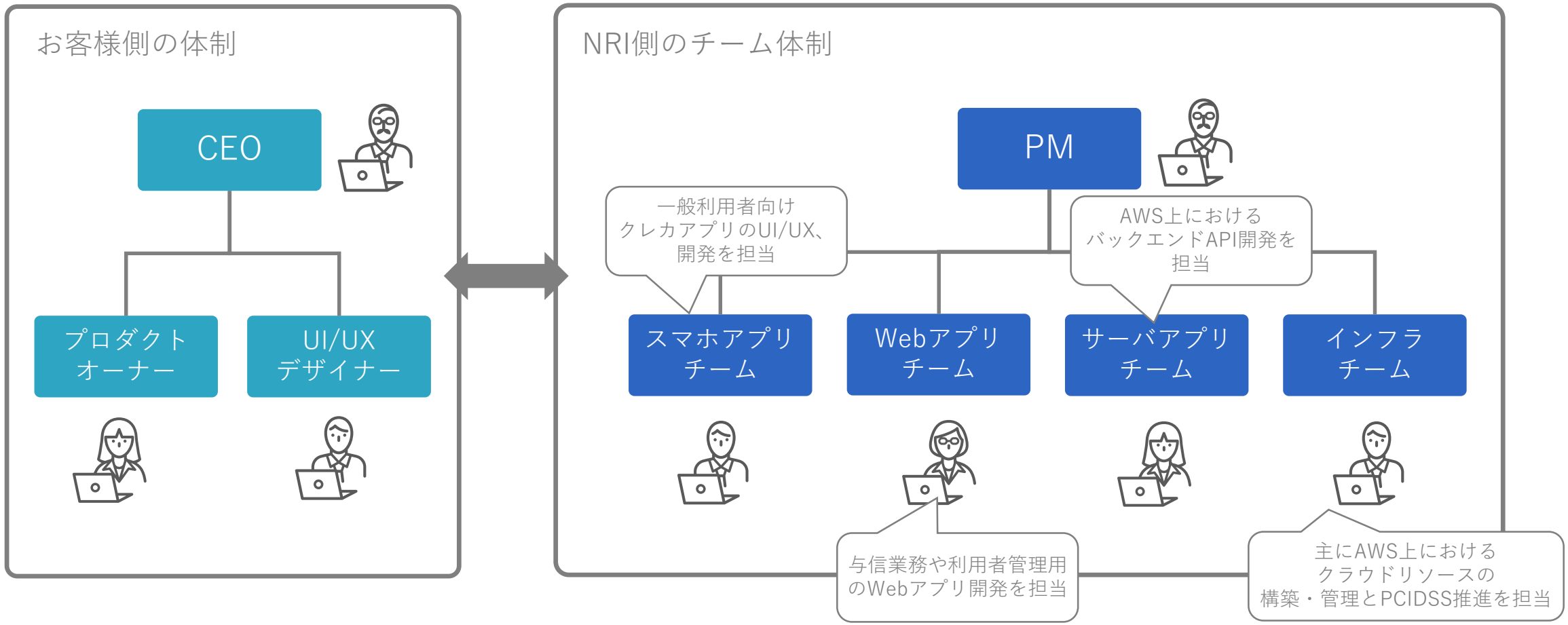


NRI側のチーム体制

NRI側はアプリとインフラというそれぞれ専門性が異なるチームで構成

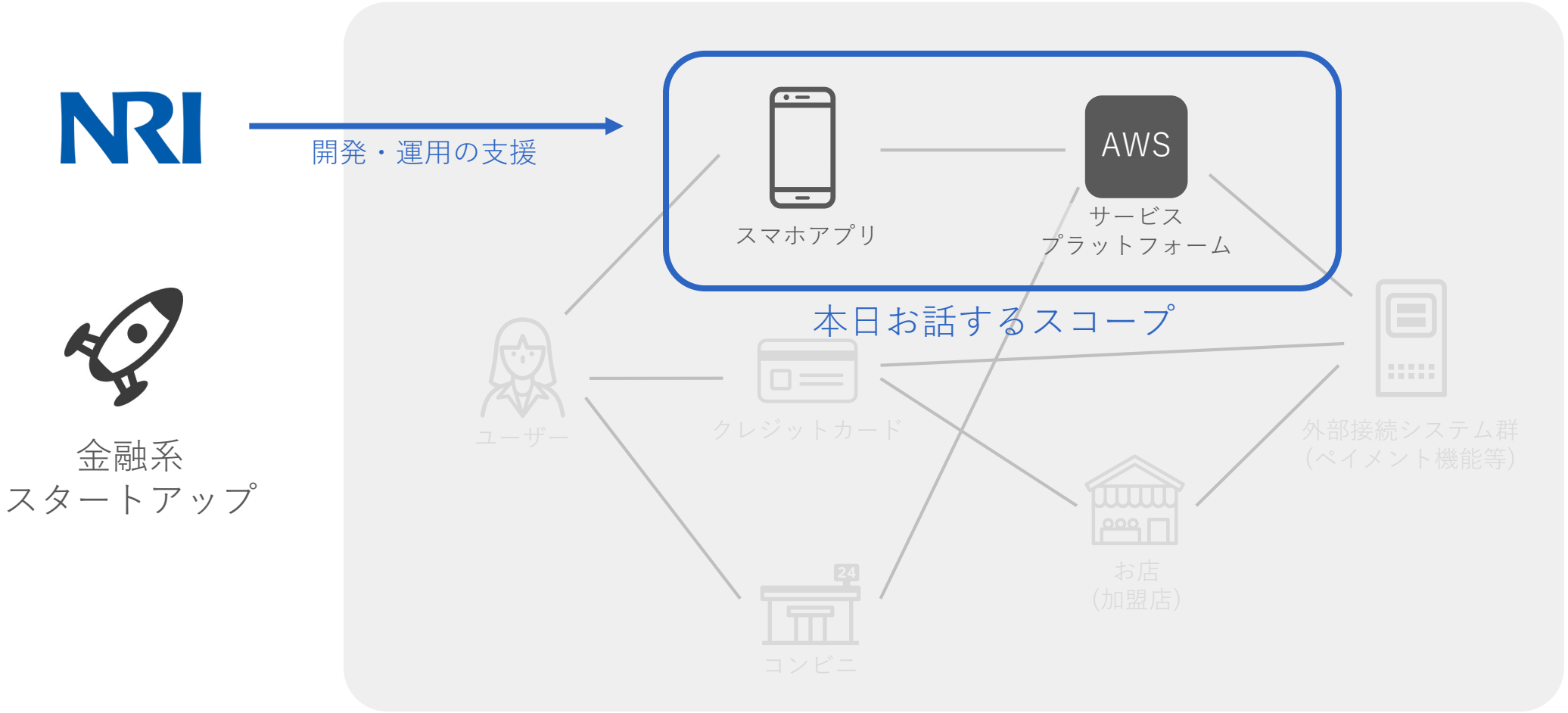


NRI側はアプリとインフラというそれぞれ専門性が異なるチームで構成

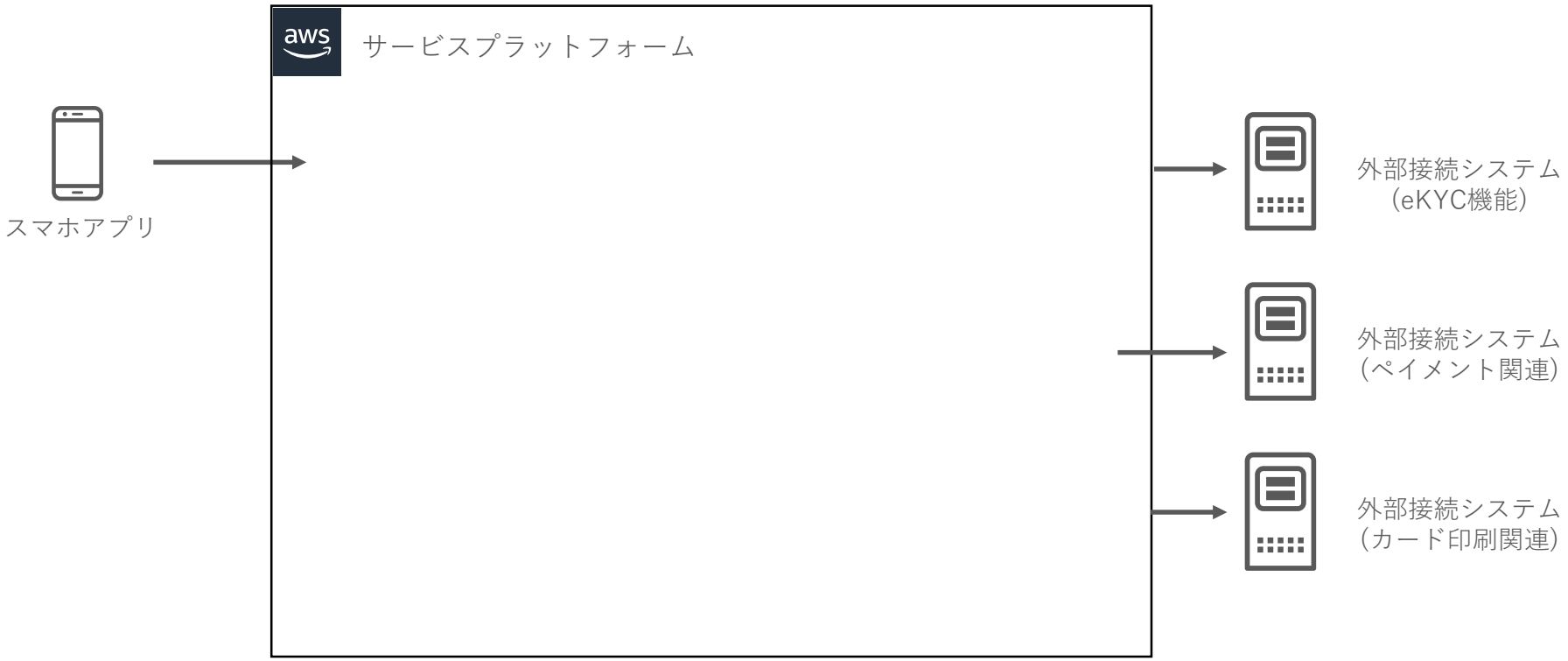


再掲

NRIはスマホアプリ+AWSサービスプラットフォームの開発・運用をご支援

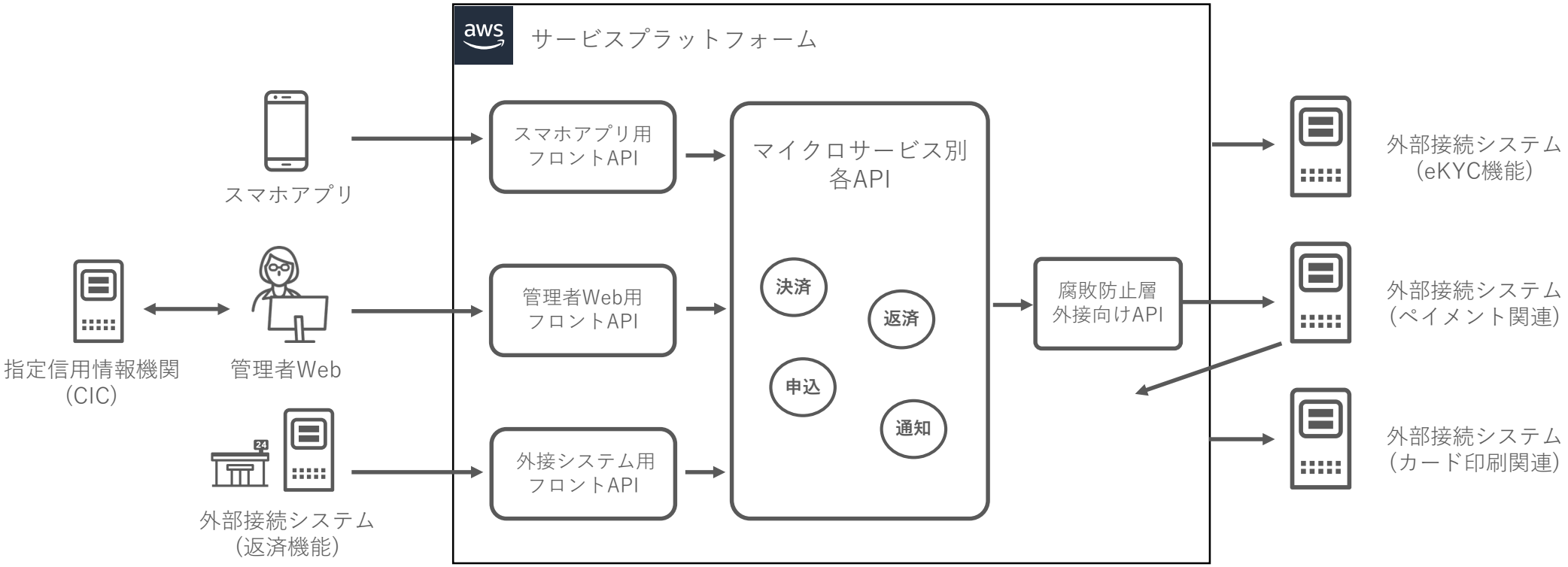


AWSを中心として、スマホアプリ・管理者Webのための バックエンド機能とインフラを構築している



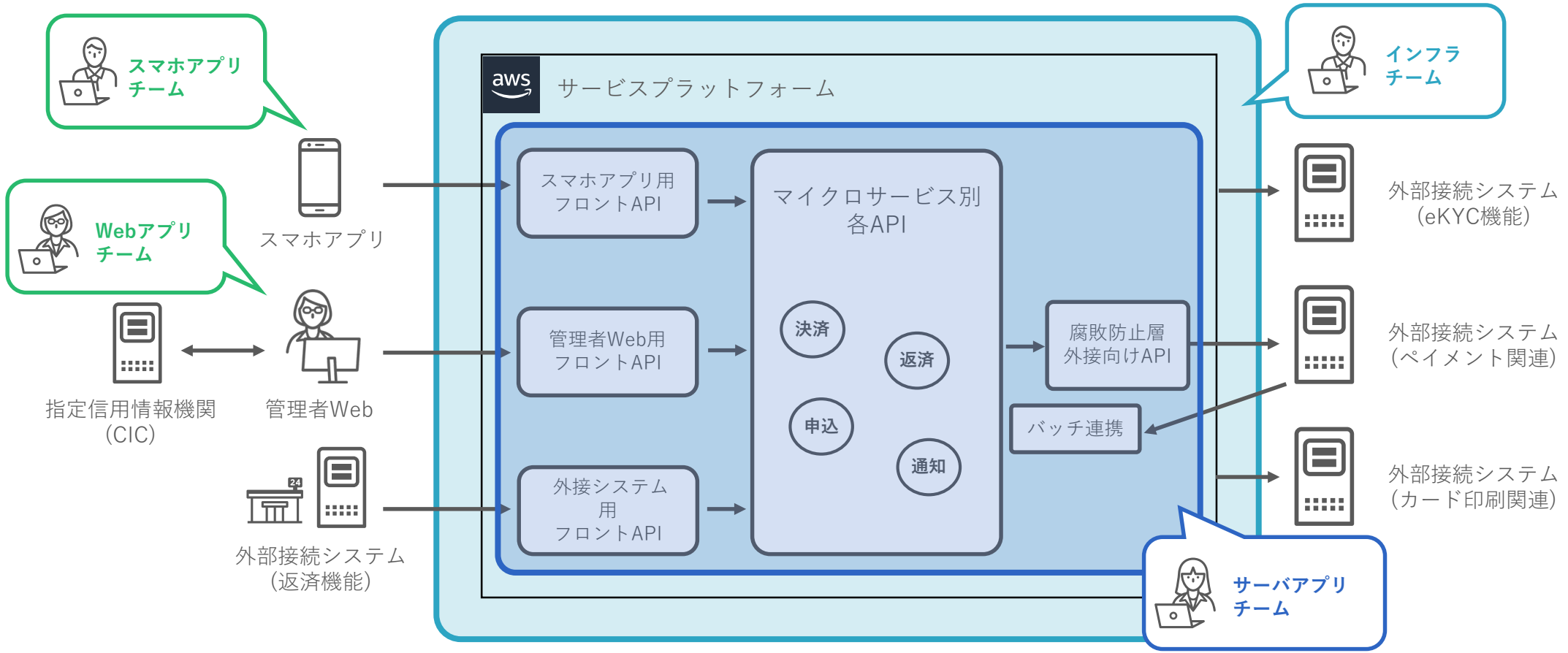
AWSを中心として、スマホアプリ・管理者Webのための バックエンド機能とインフラを構築している

バックエンド機能はマイクロサービスをベースとしたAPI群で構成。



サーバアプリチームとインフラチームが中心に AWS上のリソースを活用しながら構築。

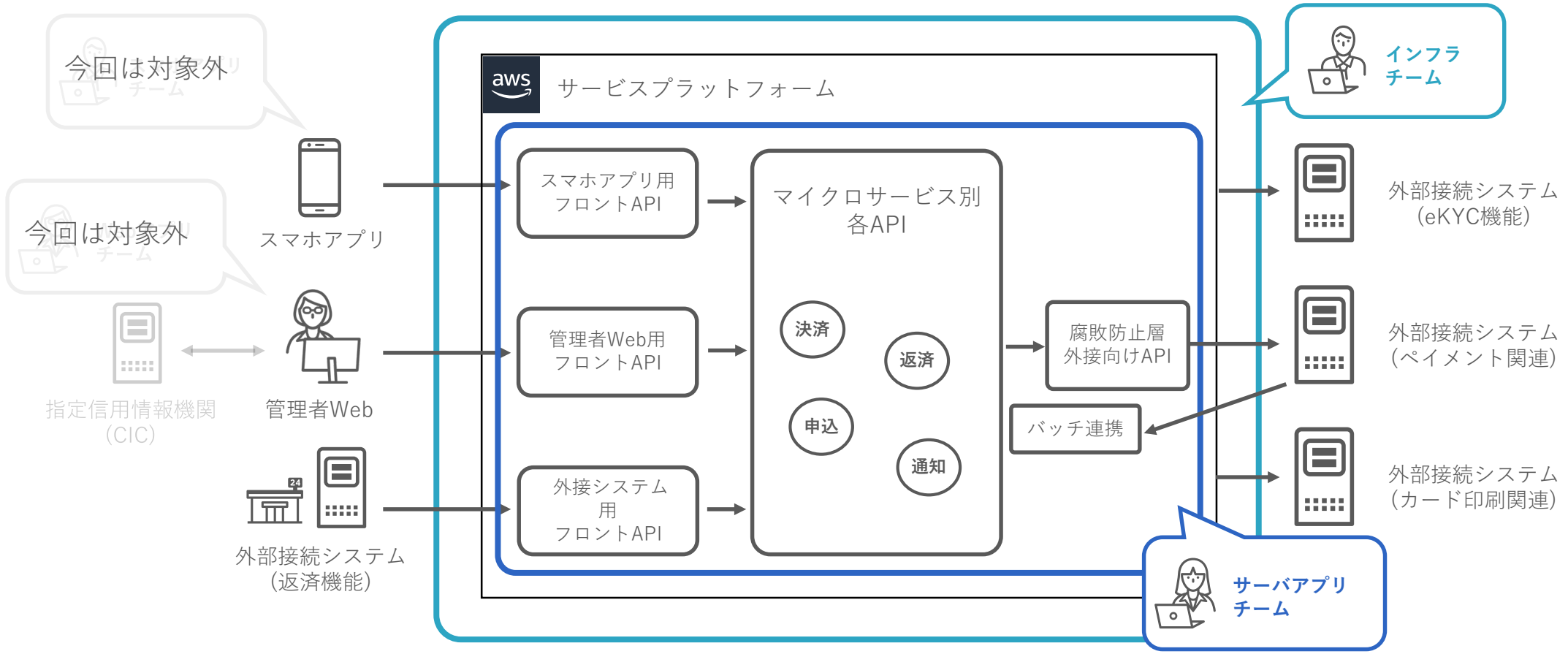
バックエンド機能はマイクロサービスをベースとしたAPI群で構成。



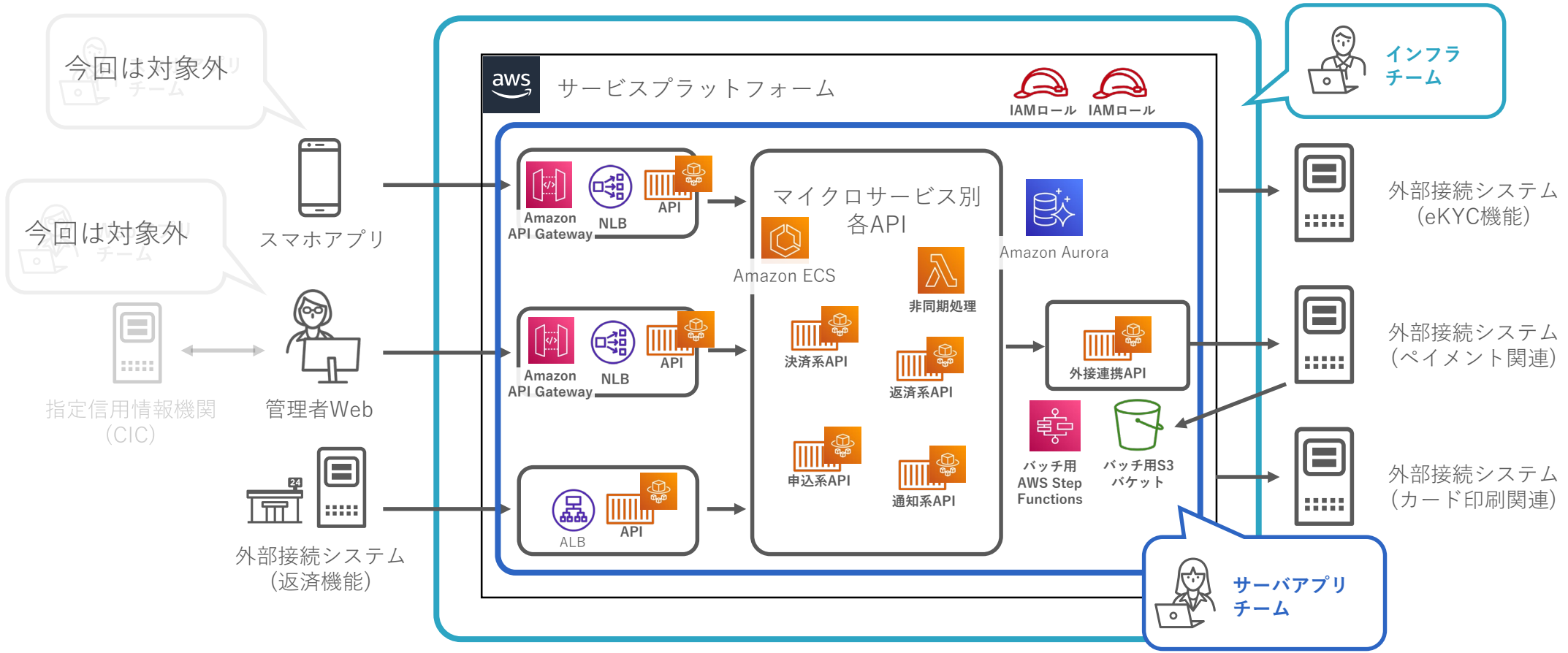
04.　AWSにおけるチームのコワーク設計

再掲

サーバアプリチームとインフラチームを中心に AWS上のリソースを活用しながらサービスプラットフォームを構築



サービスプラットフォームは
Amazon ECS / AWS Fargateを中心としマネージド・サービスで構成



サービスプラットフォームは
Amazon ECS / AWS Fargateを中心としマネージド・サービスで構成

モノの管理主体(ルール)を決めておかないと、ポテンヒットが生じる

ポテンヒットとは、野球で使われる言葉。
内野手と外野手の間にフラフラとあがったフライが落ちて、結果としてヒットとなるような打球のこと。

転じて、**「そっちのチームがやってくれると思ってた」状態**のこと。

サーバアプリ
チーム

インフラチームが提供してくれた
Dockerfileをそのまま使っていたから、
インフラ側の管理だと思っていた！

最初にコンテナスキルセットを
持っていたインフラチームが
Dockerfileのサンプルを提供しただけで
、コードの管理自体はアプリチーム側の
管理だと思っていた！

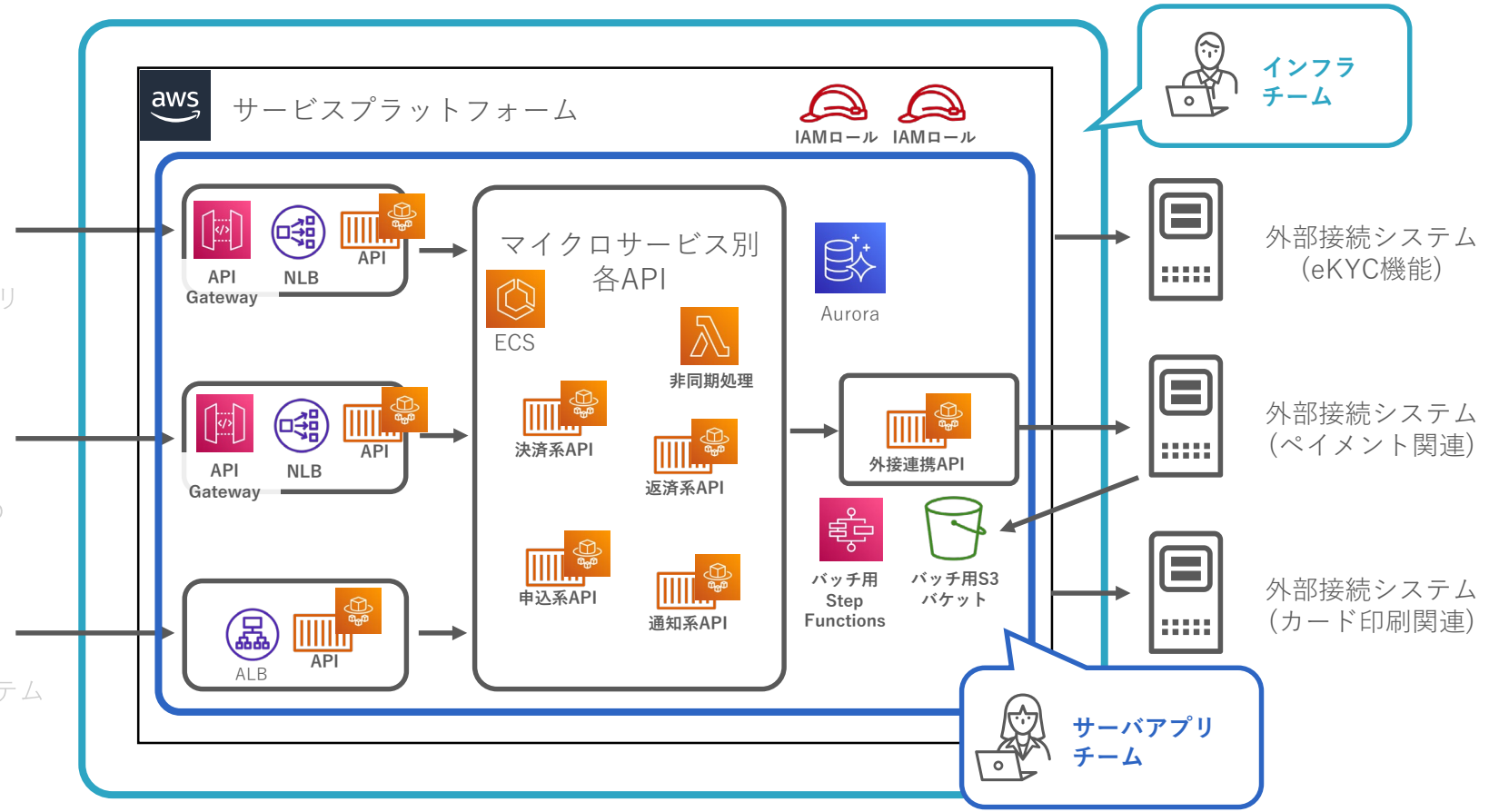
インフラチーム

・このフライを見逃さないためにも、チーム戦ではルール作りやコミュニケーションがとても大切。
・役割から明らかな対象を除き、共有するリソースにはルールを決めないと、「押し付け」の文化が蔓延する。

本発表では、コンテナCI/CD、AWS Lambda開発、AWS Step Functionsによるバッチ開発を例に、ポテンヒットを防ぐために行ったコワーク設計を紹介

コワーク設計のテーマ

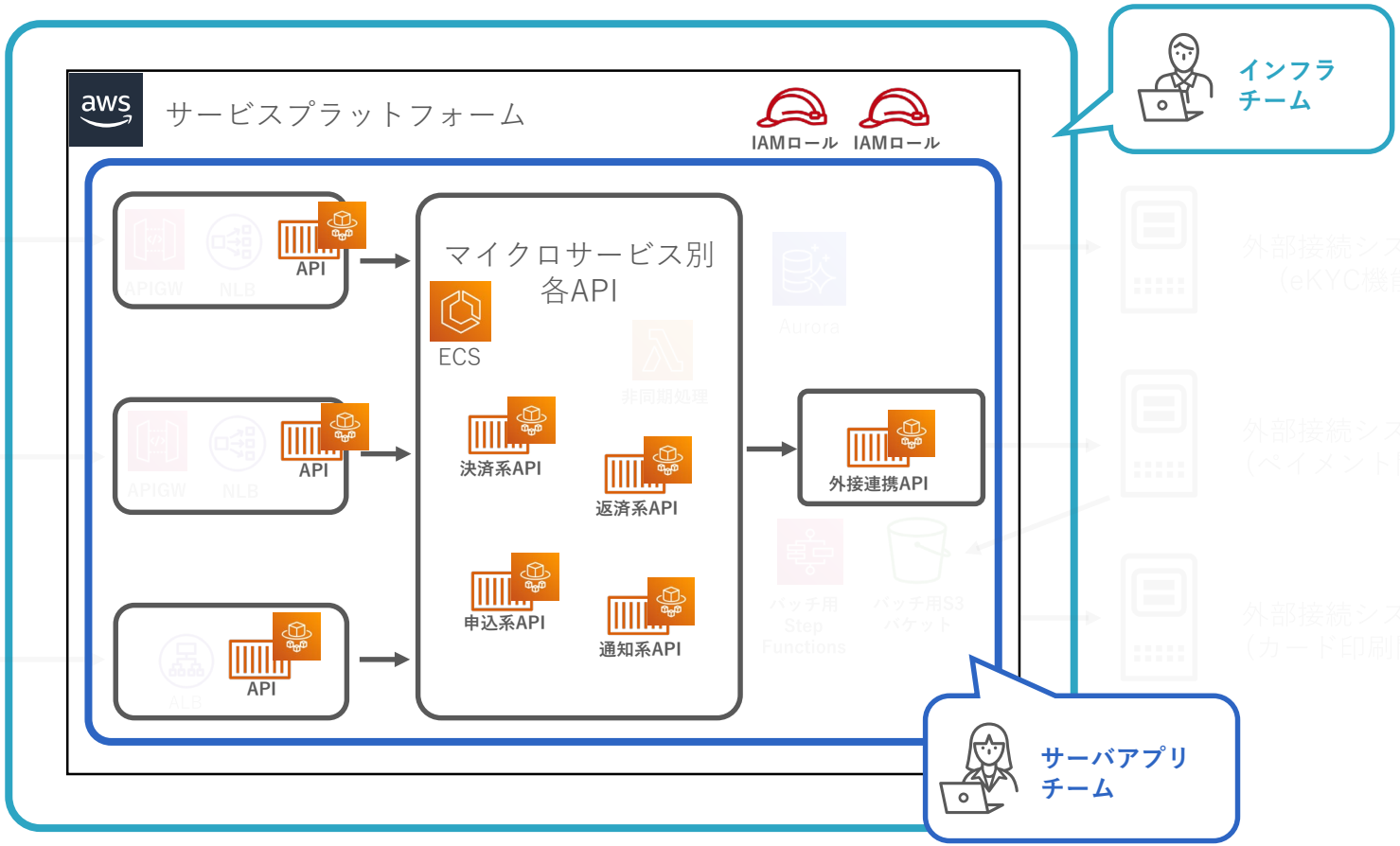
- コンテナCI/CD
- Lambda開発
- Step Functionによるバッチ処理



本発表では、コンテナCI/CD、AWS Lambda開発、AWS Step Functionsによるバッチ開発を例に、ポテンヒットを防ぐために行ったコワーク設計を紹介

コワーク設計のテーマ

- コンテナCI/CD
- Lambda開発
- Step Functionによるバッチ処理

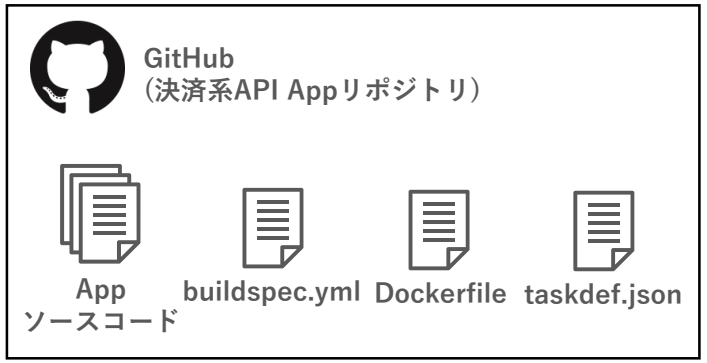


コンテナCI/CD：GitHub/AWS Codeシリーズを基軸にしたCICDパイプライン

各種バックエンドAPIはECS/Fargateタスクでマイクロサービスとして提供。
コードはGitHubで管理し、AWS Codeシリーズにてコンテナイメージビルドからデプロイまでを構築。

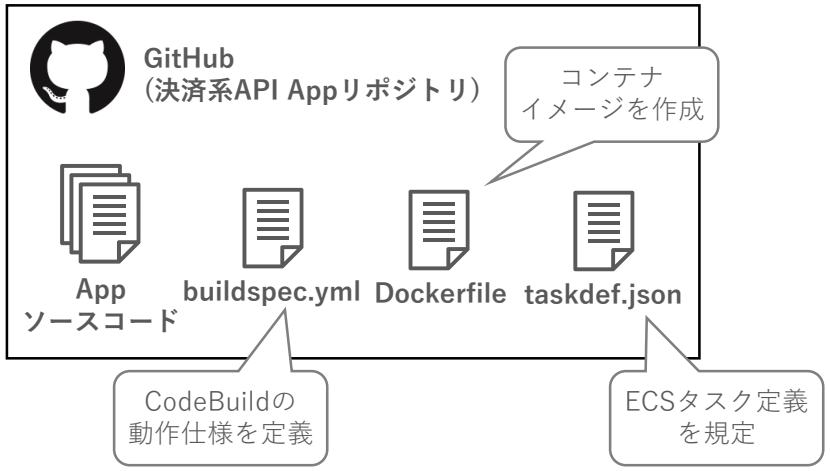
コンテナCI/CD：
GitHub/AWS Codeシリーズを基軸にしたCICDパイプライン

各種バックエンドAPIはECS/Fargateタスクでマイクロサービスとして提供。
コードはGitHubで管理し、AWS Codeシリーズにてコンテナイメージビルドからデプロイまでを構築。



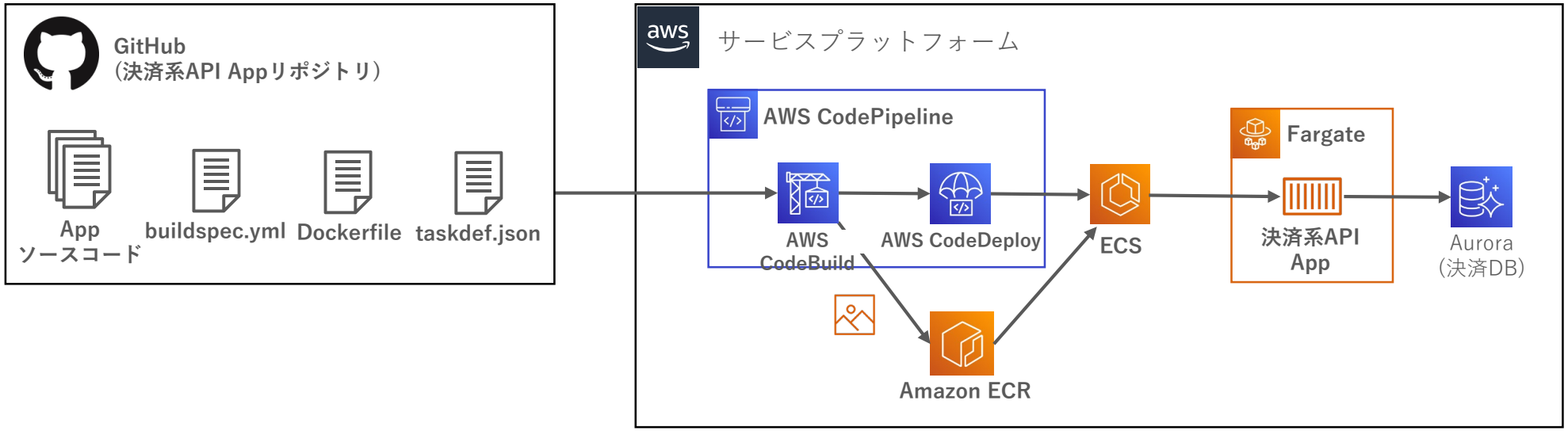
コンテナCI/CD：
GitHub/AWS Codeシリーズを基軸にしたCICDパイプライン

各種バックエンドAPIはECS/Fargateタスクでマイクロサービスとして提供。
コードはGitHubで管理し、AWS Codeシリーズにてコンテナイメージビルドからデプロイまでを構築。



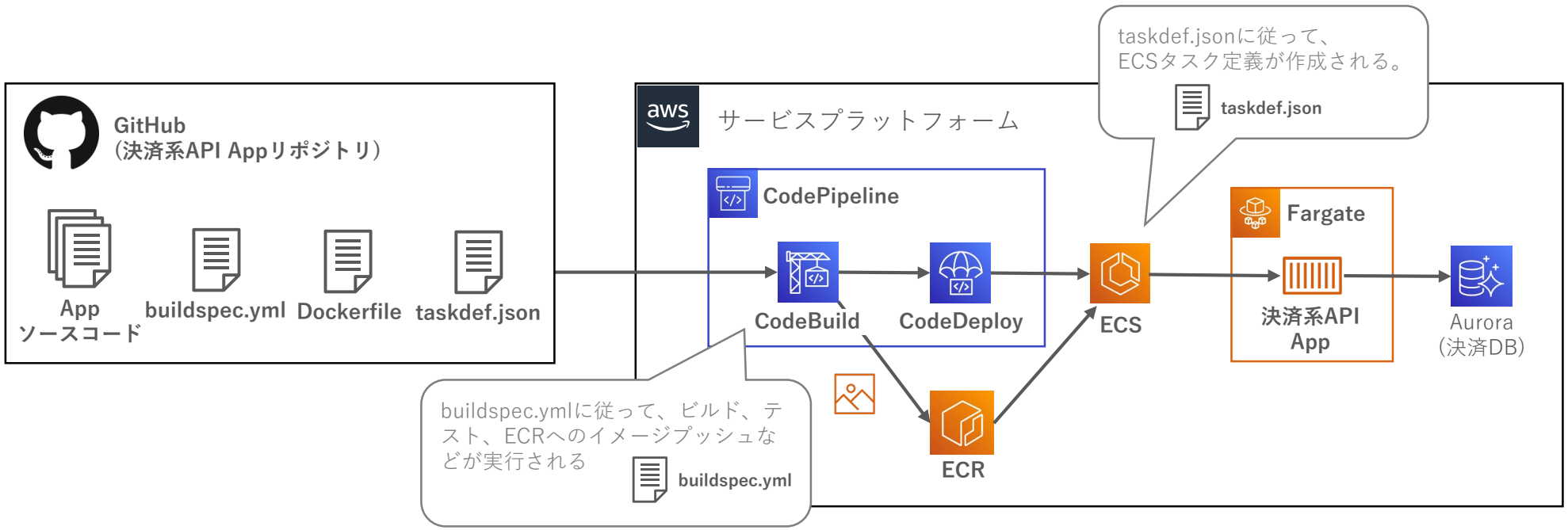
コンテナCI/CD：
GitHub/AWS Codeシリーズを基軸にしたCICDパイプライン

各種バックエンドAPIはECS/Fargateタスクでマイクロサービスとして提供。
コードはGitHubで管理し、AWS Codeシリーズにてコンテナイメージビルドからデプロイまでを構築。



コンテナCI/CD：
GitHub/AWS Codeシリーズを基軸にしたCICDパイプライン

各種バックエンドAPIはECS/Fargateタスクでマイクロサービスとして提供。
コードはGitHubで管理し、AWS Codeシリーズにてコンテナイメージビルドからデプロイまでを構築。

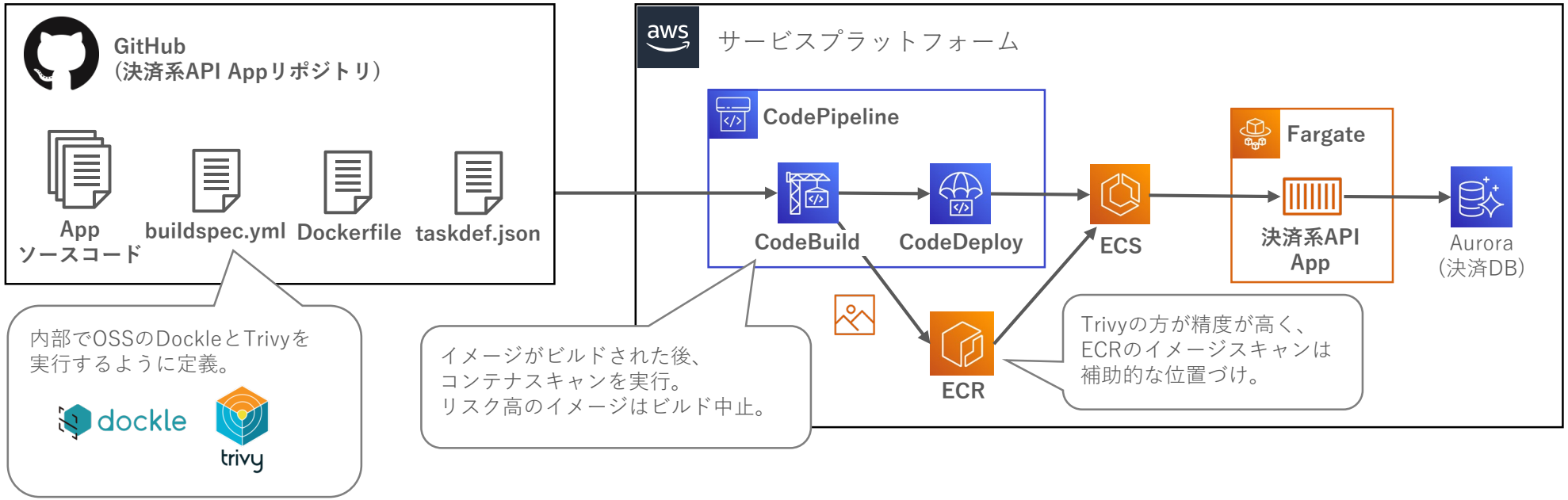


コンテナCI/CD ：
PCIDSSに準拠すべく、DevSecOpsによるコンテナスキャンが必要。

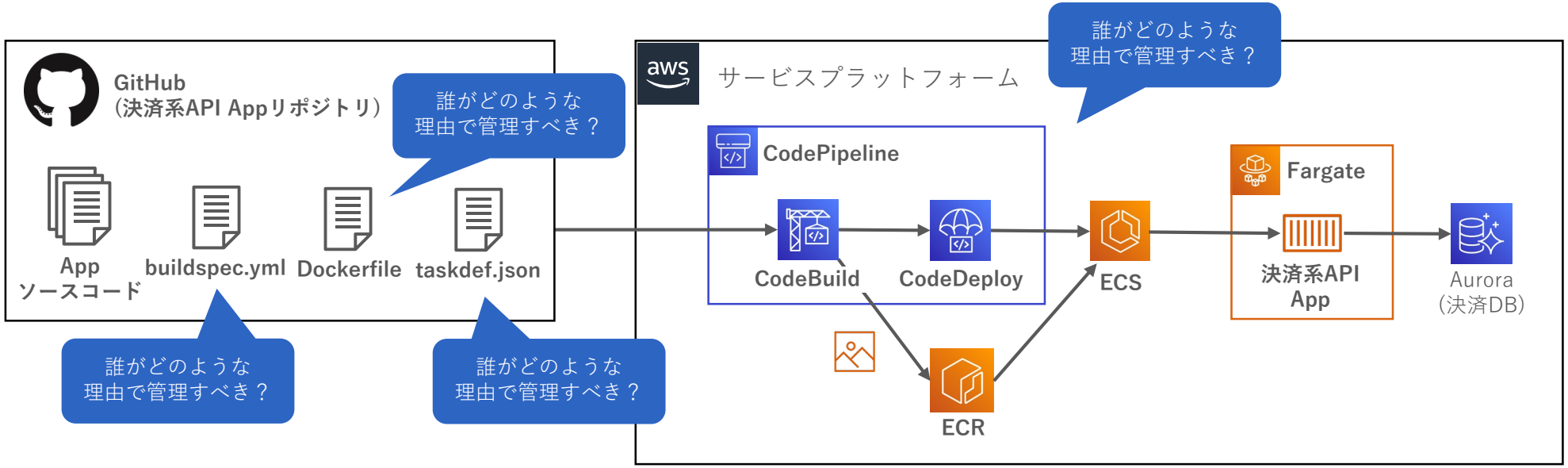
PCIDSS

・6.1 セキュリティ脆弱性情報の信頼できる社外提供元を使って**セキュリティの脆弱性を特定**し、
新たに発見されたセキュリティの脆弱性にリスクのランク（「高」、「中」、「低」など）を割り当てるプロセスを確立する。

・6.2 すべてのシステムコンポーネントとソフトウェアに、ベンダ提供のセキュリティパッチがインストールされ、
既知の脆弱性から保護されている。**重要なセキュリティパッチは、リリース後 1 カ月以内にインストール**する。



コンテナCI/CD ：
ポテンヒットを防ぐために、リソース毎に管理すべきものを洗い出す



※Appソースコードは明らかにサーバアプリチーム管理なので除外

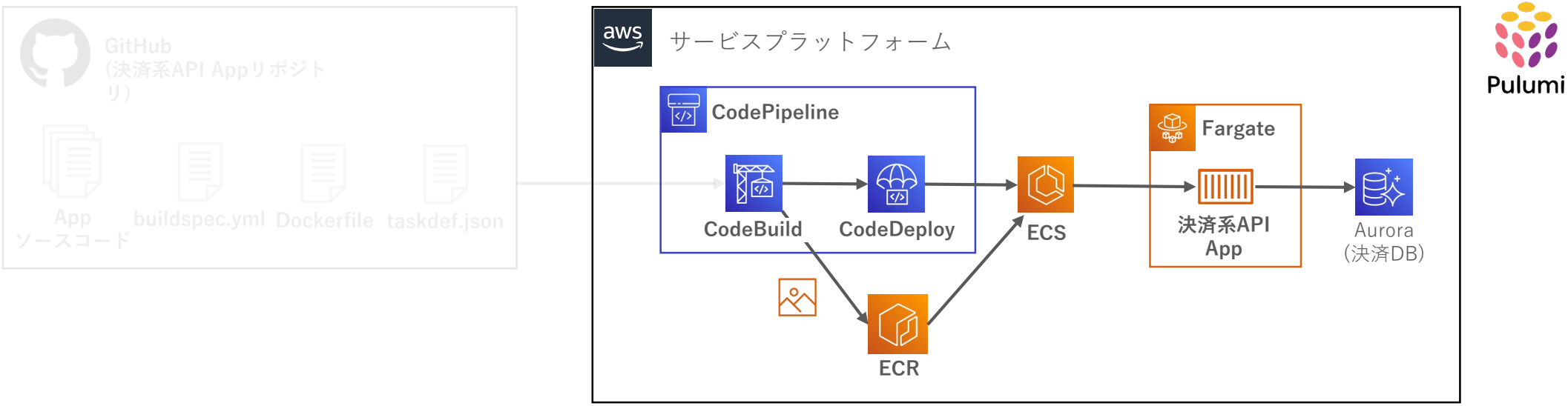
コンテナCI/CD ：AWSリソース全般はインフラチーム管理。アプリチーム用に参照権限+αのIAMロールを付与。

サーバアプリチーム

・**主体的にAWSサービスを変更する要件はなし**。
・一方、リリース作業に関連して、リソースの参照権限や、CodePipelineの実行権限はほしい。

・AWSサービス構築はIaCツール(Pulumi)で実装。
→**ある程度コードで管理してしまった方が、リソース整合性や自動化の観点から都合が良い**。
・PCIDSS推進上、適切なセキュリティ実装やコンプライアンス準拠を担保したい。

インフラチーム



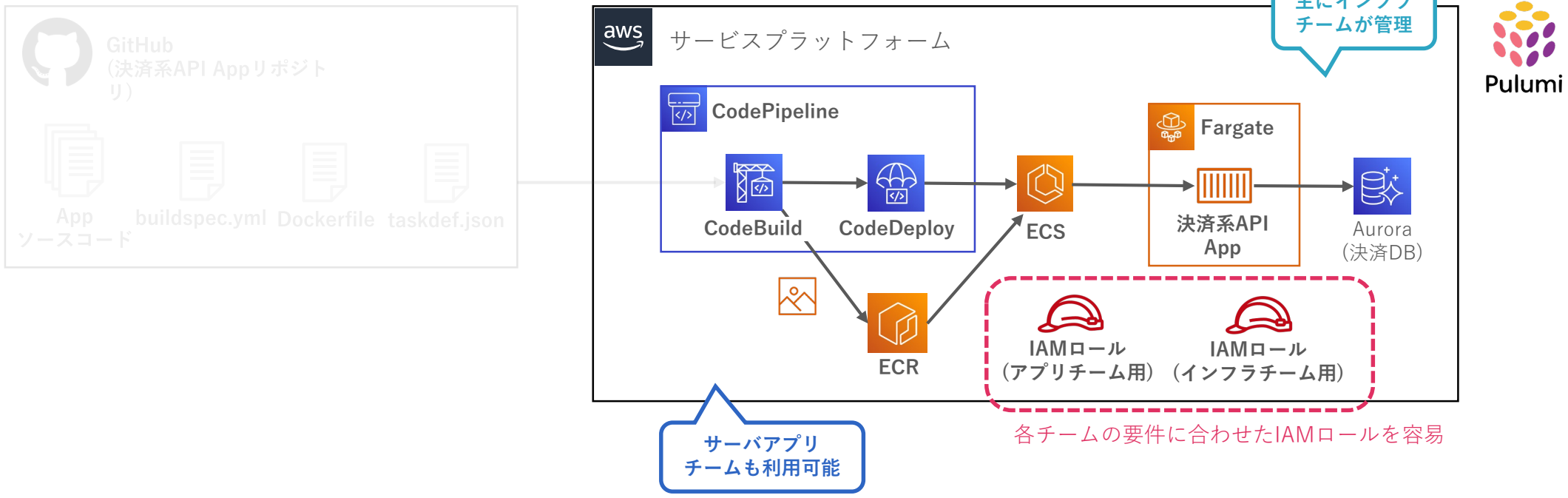
コンテナCI/CD ：AWSリソース全般はインフラチーム管理。アプリチーム用に参照権限+αのIAMロールを付与。

サーバアプリチーム

・**主体的にAWSサービスを変更する要件はなし**。
・一方、リリース作業に関連して、リソースの参照権限や、CodePipelineの実行権限はほしい。

・AWSサービス構築はIaCツール(Pulumi)で実装。
→**ある程度コードで管理してしまった方が、リソース整合性や自動化の観点から都合が良い**。
・PCIDSS推進上、適切なセキュリティ実装やコンプライアンス準拠を担保したい。

インフラチーム



各チームの要件に合わせたIAMロールを容易

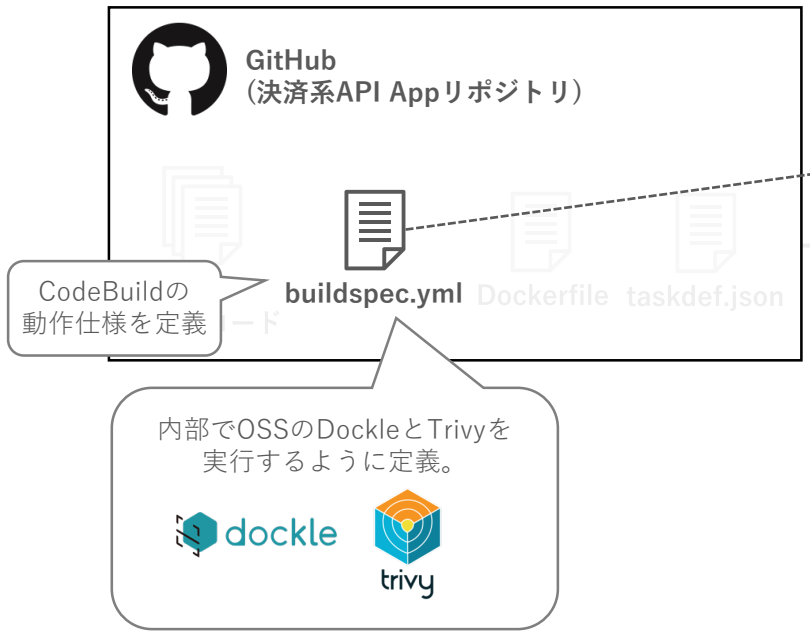
コンテナCI/CD ：ビルド定義はインフラチーム所有としつつも、アプリチーム側からの変更もWelcomeに。

サーバアプリチーム

・ビルドに必要なライブラリなどをインストールする場合、アプリに関連する処理が記述される。
・一度記述した後は頻繁な変更は発生しない(Dockerfile側で吸収)ため、所有の主体はどちらでもOK。

・PCIDSS準拠のためのコンテナスキャン処理をbuildspec.ymlに実装したい。
・ECRなど、AWSサービスとの連携が定義される。
・他のマイクロサービス含めてDRYに管理したい。

インフラチーム




```
buildspec.yml
---

phases:
  install:
    - dockleとtrivyのインストール

  pre_build:
     - 環境変数の設定(ECRイメージタグの設定等)
    :

  build:
     - Dockerイメージビルド処理
     - ECRイメージプッシュ用のタグ付け

  post_build:
     - dockleによるCISベストプラクティスチェック
     - Trivyによるコンテナスキャン
     - ECRへのイメージプッシュ
```

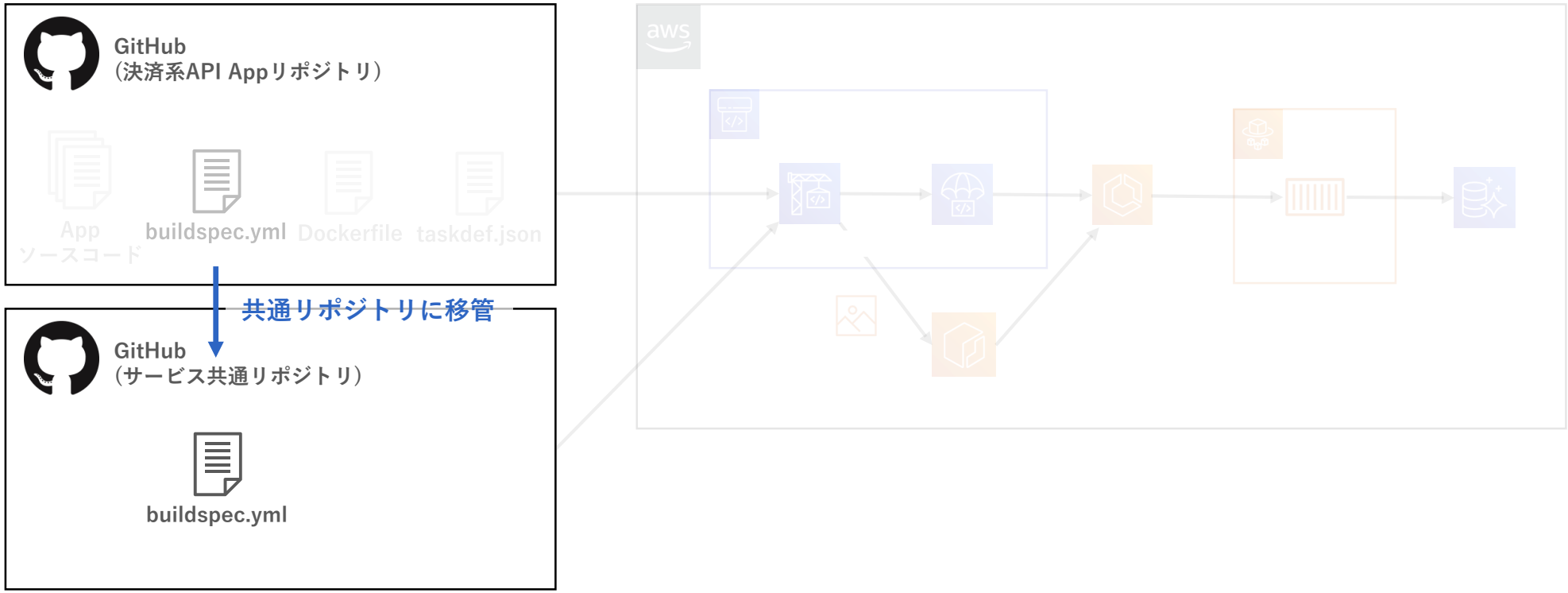
コンテナCI/CD ：ビルド定義はインフラチーム所有としつつも、アプリチーム側からの変更もWelcomeに。

サーバアプリチーム

・ビルドに必要なライブラリなどをインストールする場合、アプリに関連する処理が記述される。
・一度記述した後は頻繁な変更は発生しない(Dockerfile側で吸収)ため、所有の主体はどちらでもOK。

・PCIDSS準拠のためのコンテナスキャン処理をbuildspec.ymlに実装したい。
・ECRなど、AWSサービスとの連携が定義される。
・他のマイクロサービス含めてDRYに管理したい。

インフラチーム



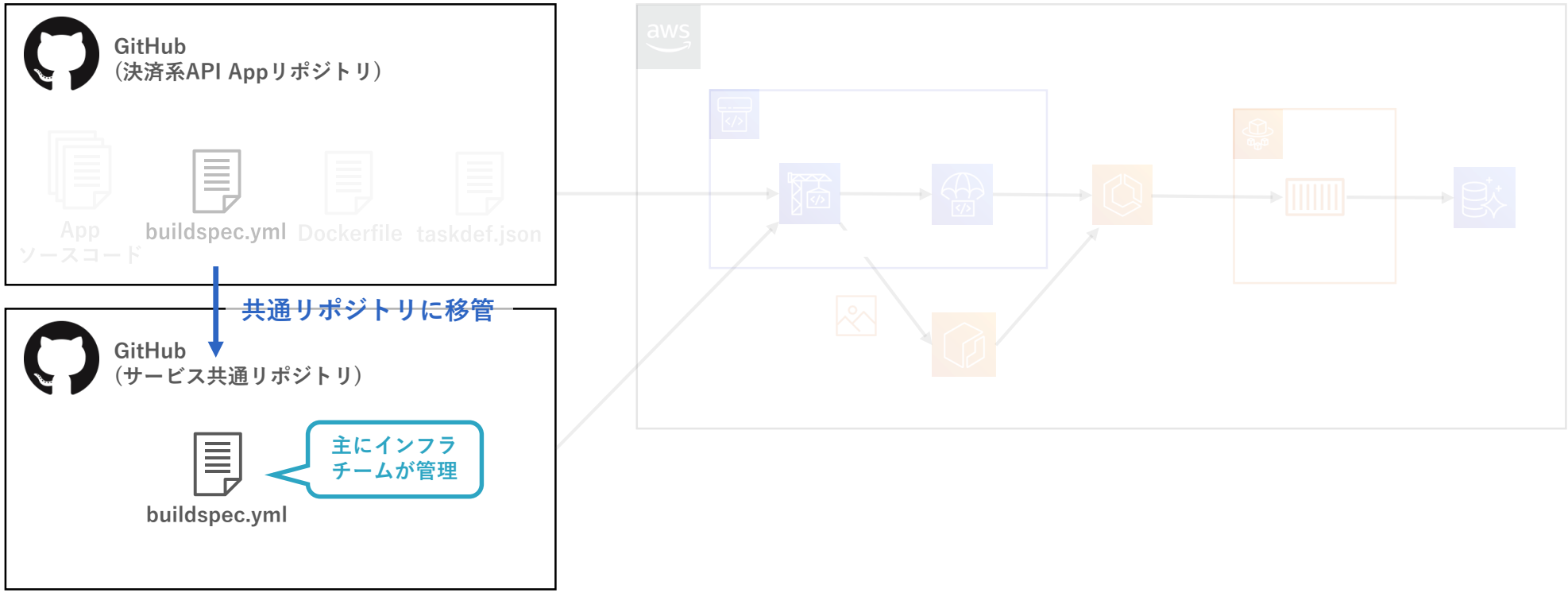
コンテナCI/CD ：ビルド定義はインフラチーム所有としつつも、アプリチーム側からの変更もWelcomeに。

サーバアプリチーム

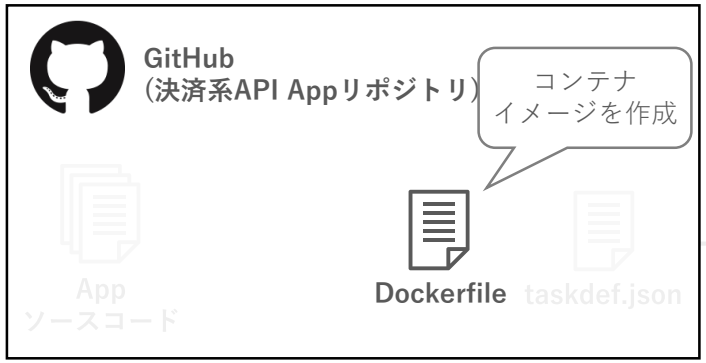
・ビルドに必要なライブラリなどをインストールする場合、アプリに関連する処理が記述される。
・一度記述した後は頻繁な変更は発生しない(Dockerfile側で吸収)ため、所有の主体はどちらでもOK。

・PCIDSS準拠のためのコンテナスキャン処理をbuildspec.ymlに実装したい。
・ECRなど、AWSサービスとの連携が定義される。
・他のマイクロサービス含めてDRYに管理したい。

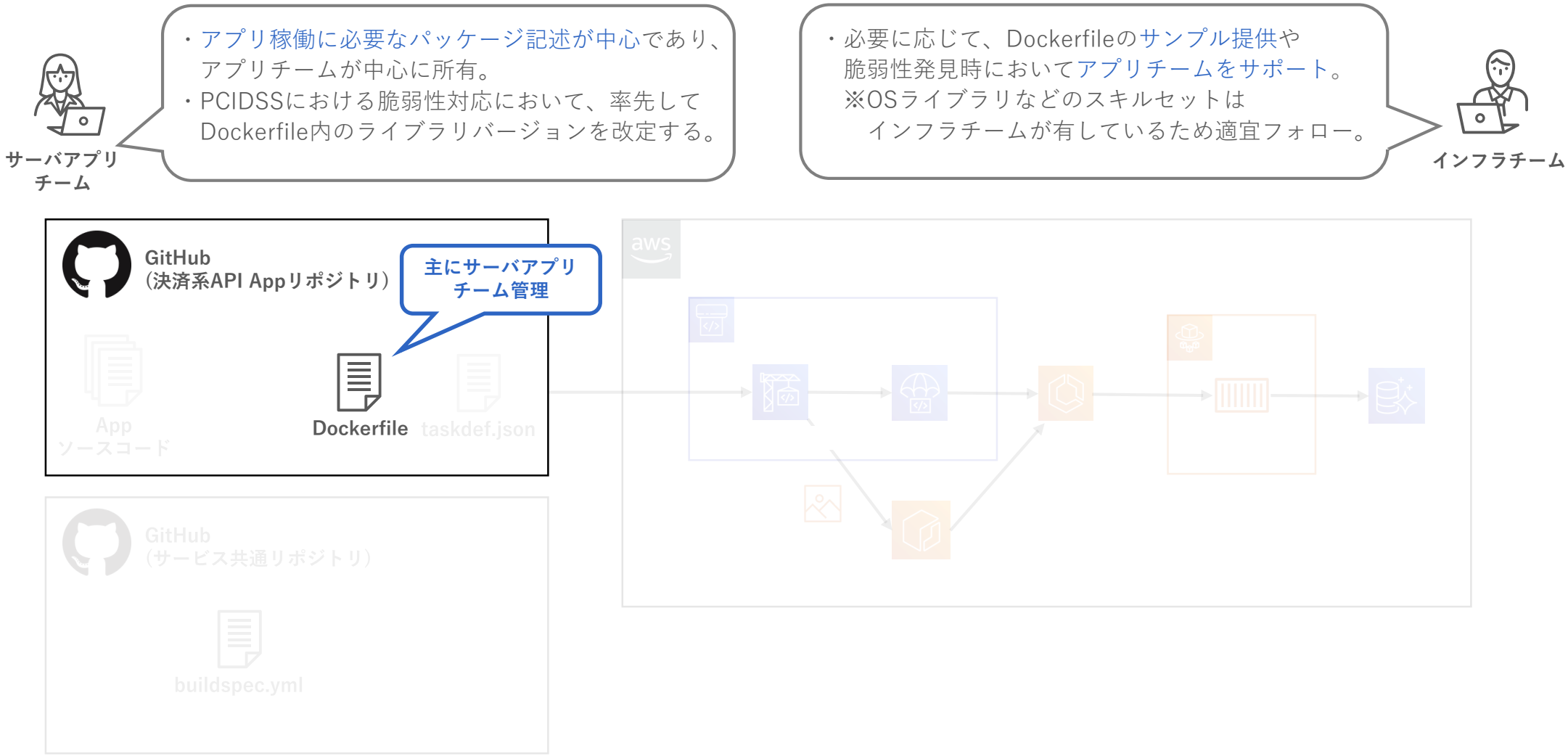
インフラチーム



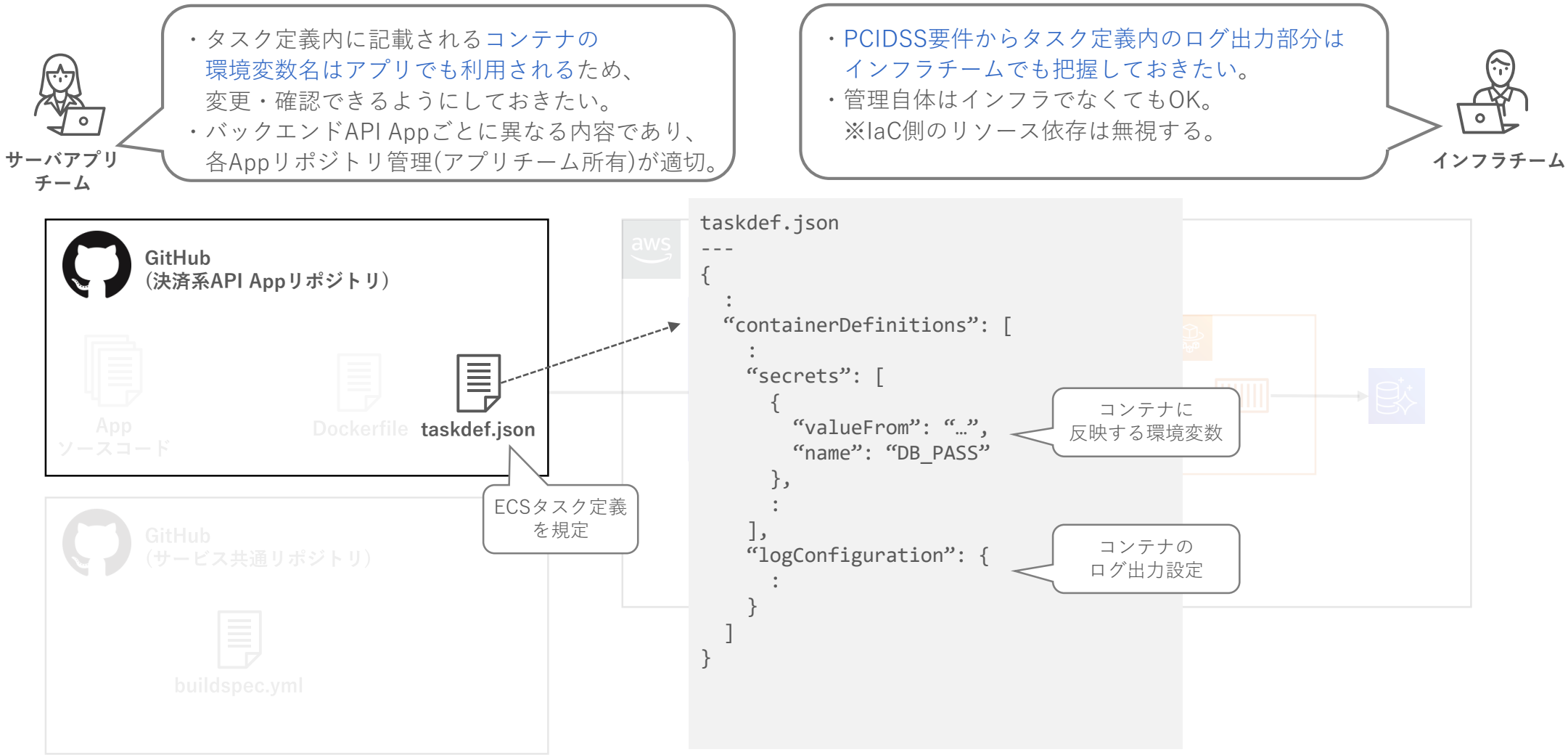
コンテナCI/CD ：Dockerfileはアプリ稼働に必要な要件で構成されることから、アプリチーム所有としつつインフラチームはサポート。



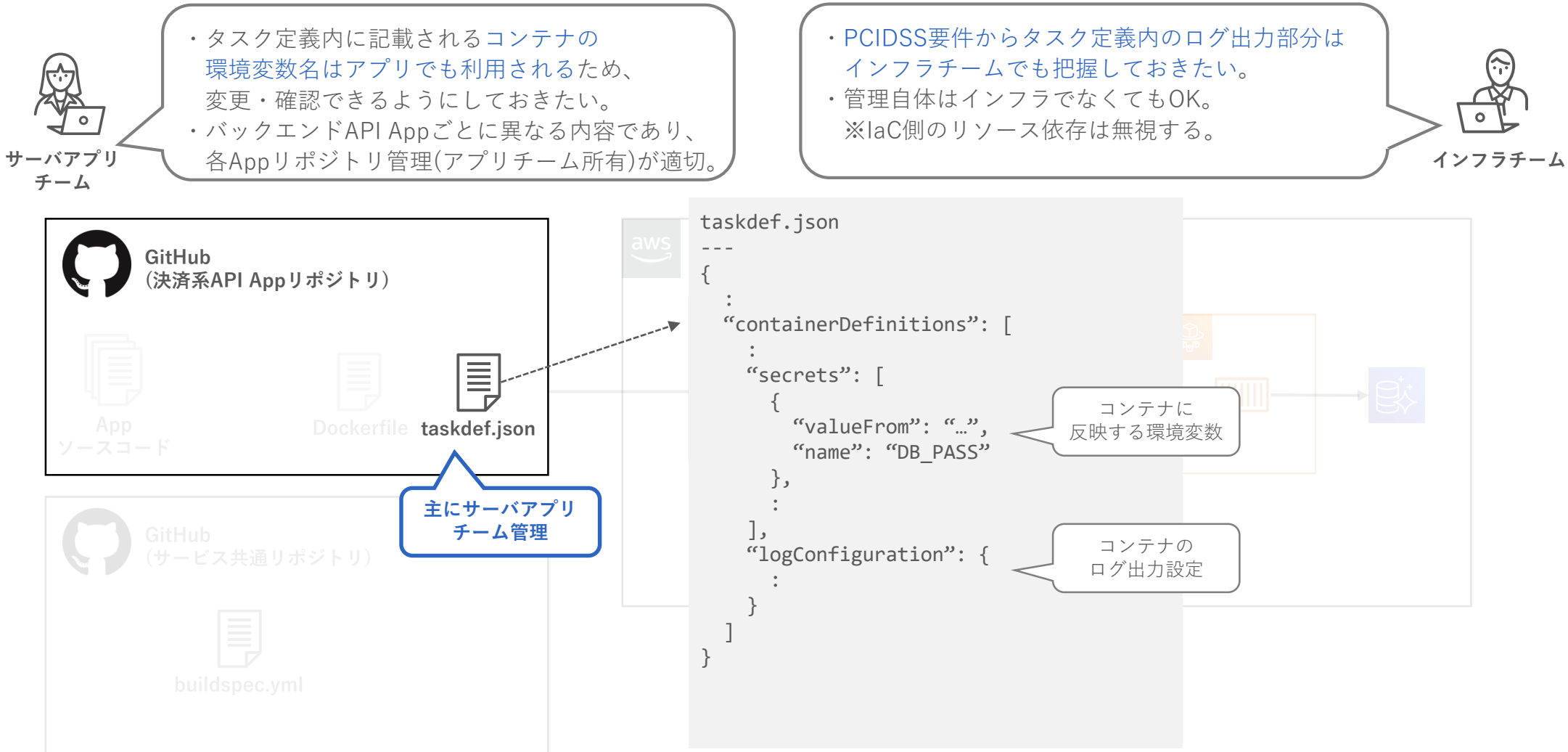
コンテナCI/CD ：Dockerfileはアプリ稼働に必要な要件で構成されることから、アプリチーム所有としつつインフラチームはサポート。



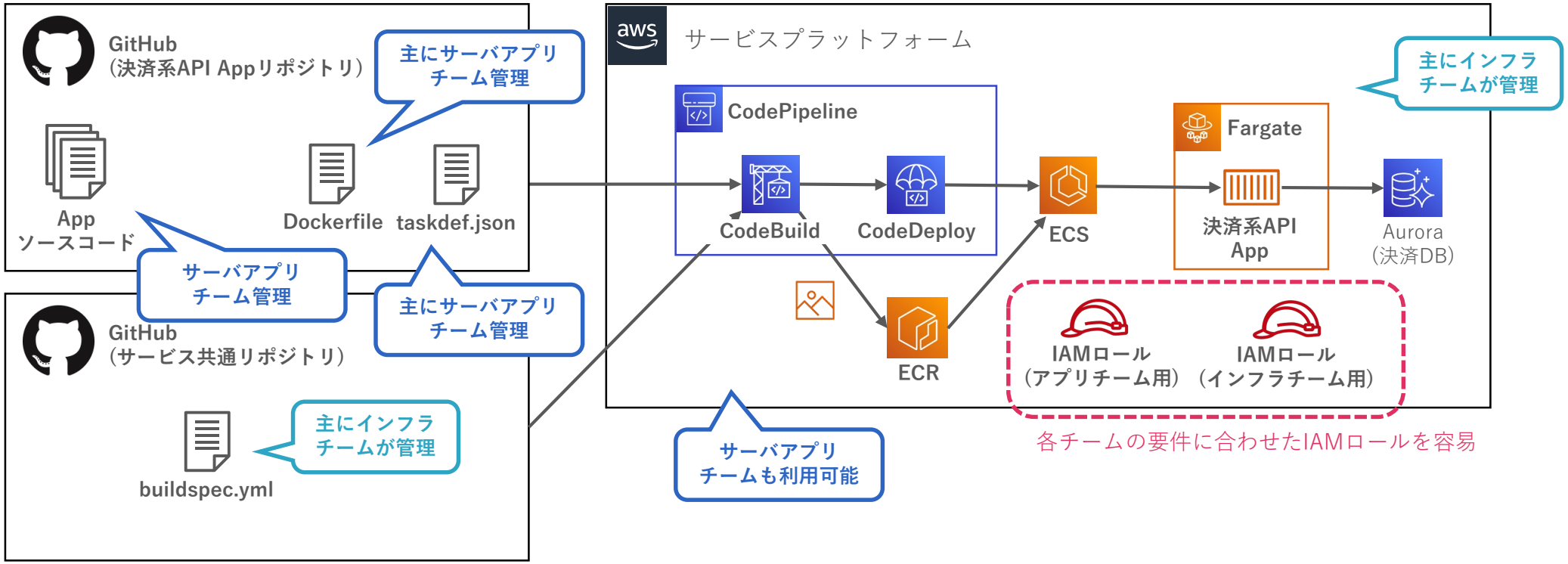
コンテナCI/CD ：ECSタスク定義は両チームの要件が含まれる。
タスク定義内における両者が気にするポイントをお互い理解し、所有を決める。



コンテナCI/CD ：ECSタスク定義は両チームの要件が含まれる。
タスク定義内における両者が気にするポイントをお互い理解し、所有を決める。



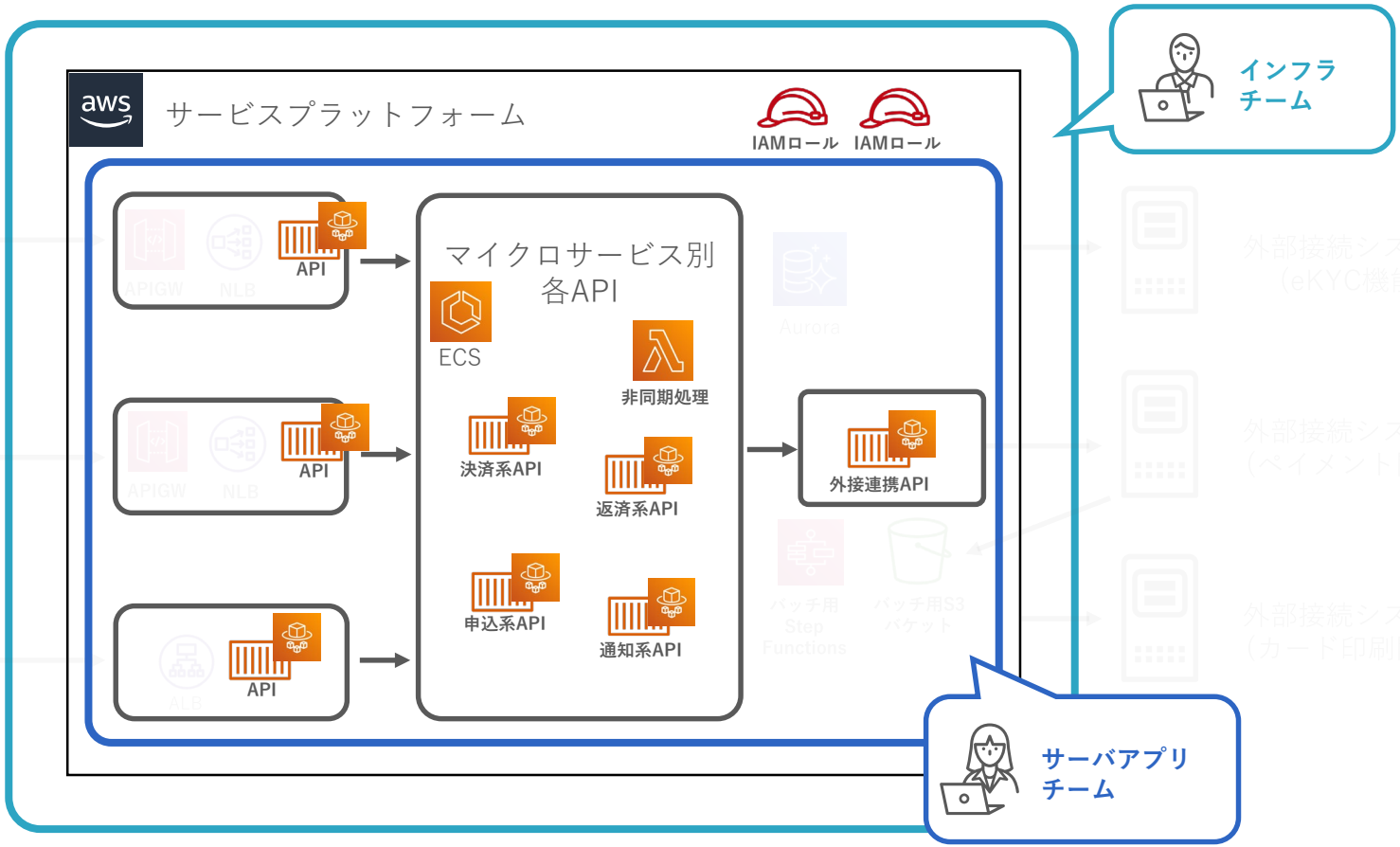
コンテナCI/CD ：まとめ



本発表では、コンテナCI/CD、Lambda開発、Step Functionsによる バッチ開発を例に、ポテンヒットを防ぐために行ったコワーク設計を紹介

コワーク設計のテーマ

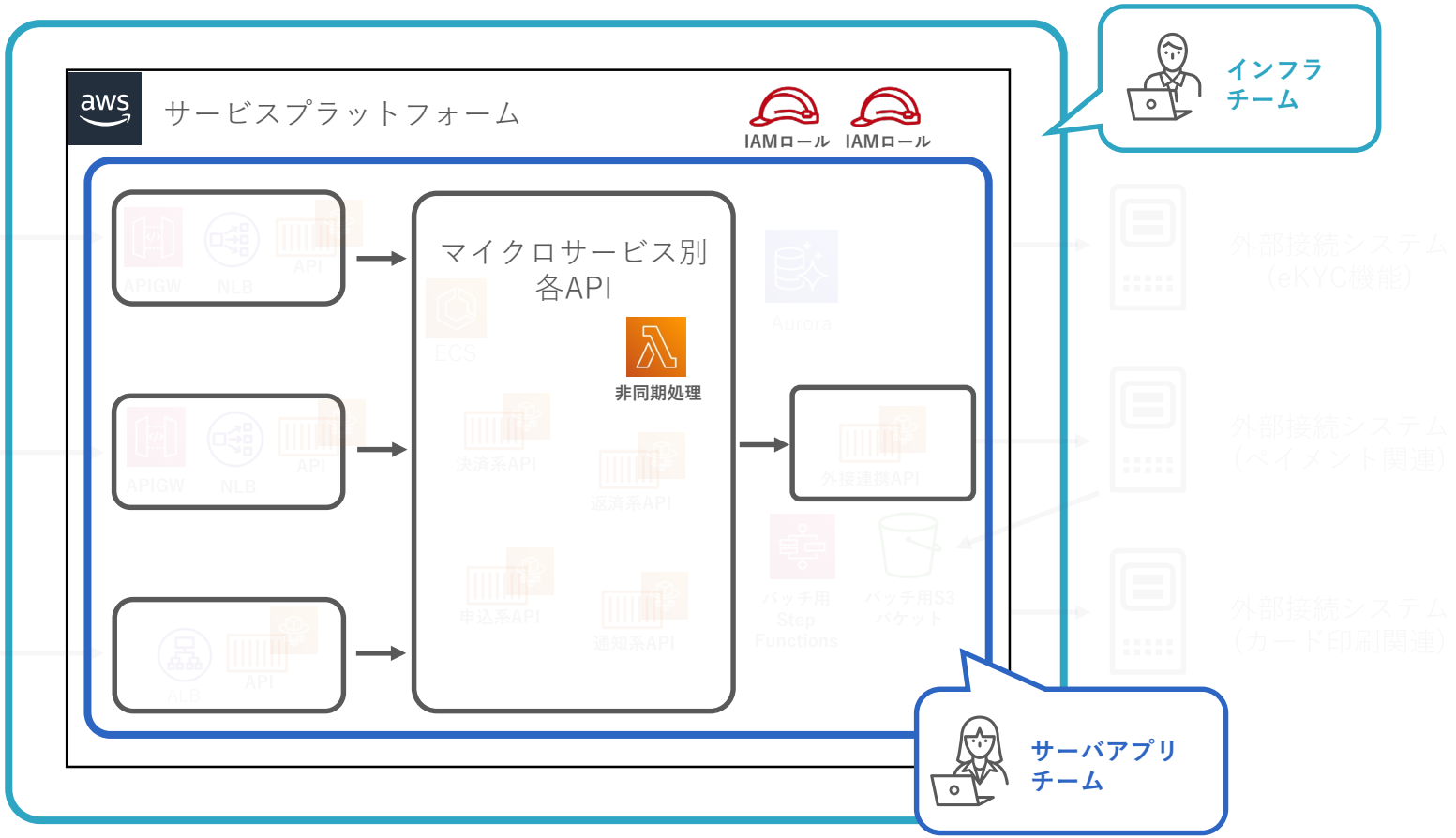
- コンテナCI/CD
- Lambda開発
- Step Functionsによる バッチ処理



本発表では、コンテナCI/CD、Lambda開発、Step Functionsによる バッチ開発を例に、ポテンヒットを防ぐために行ったコワーク設計を紹介

コワーク設計のテーマ

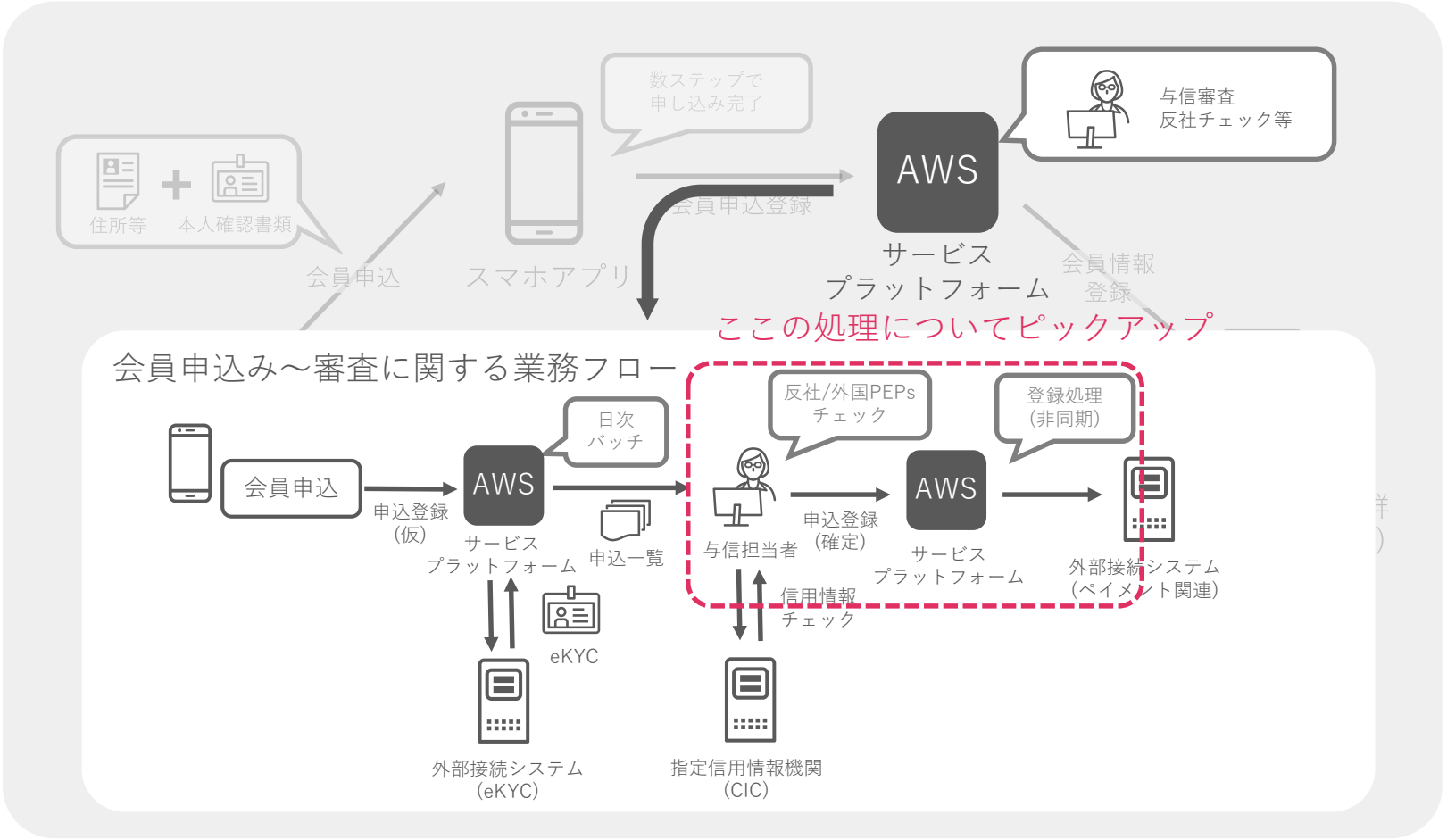
- コンテナCI/CD
- **Lambda開発**
- Step Functionsによる バッチ処理



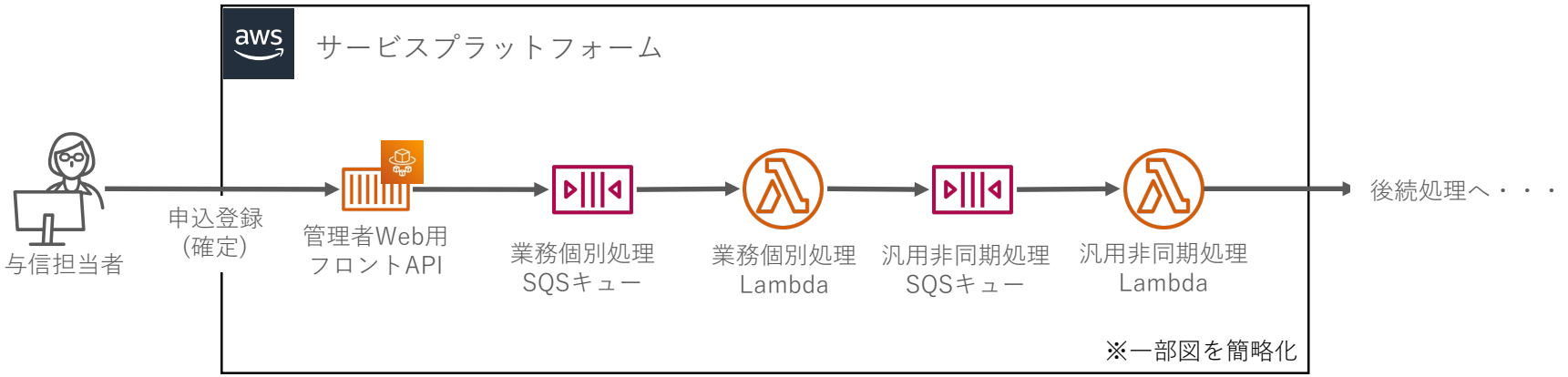
再掲

入会～審査を始めとしたクレジットカード特有の複雑な業務を紐解き、方式の検討から実装までを検討する必要がある。

金融系
スタートアップ

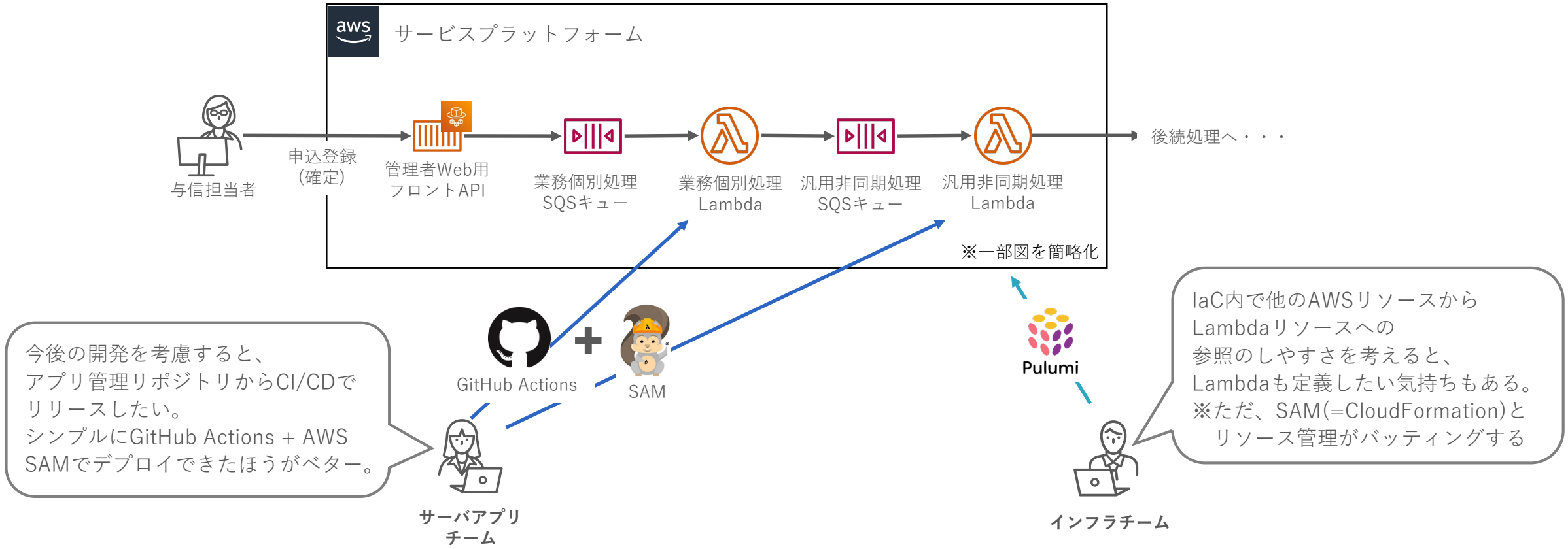


審査後の申込登録を行うと、Amazon SQS / AWS Lambdaにて非同期処理がなされる。

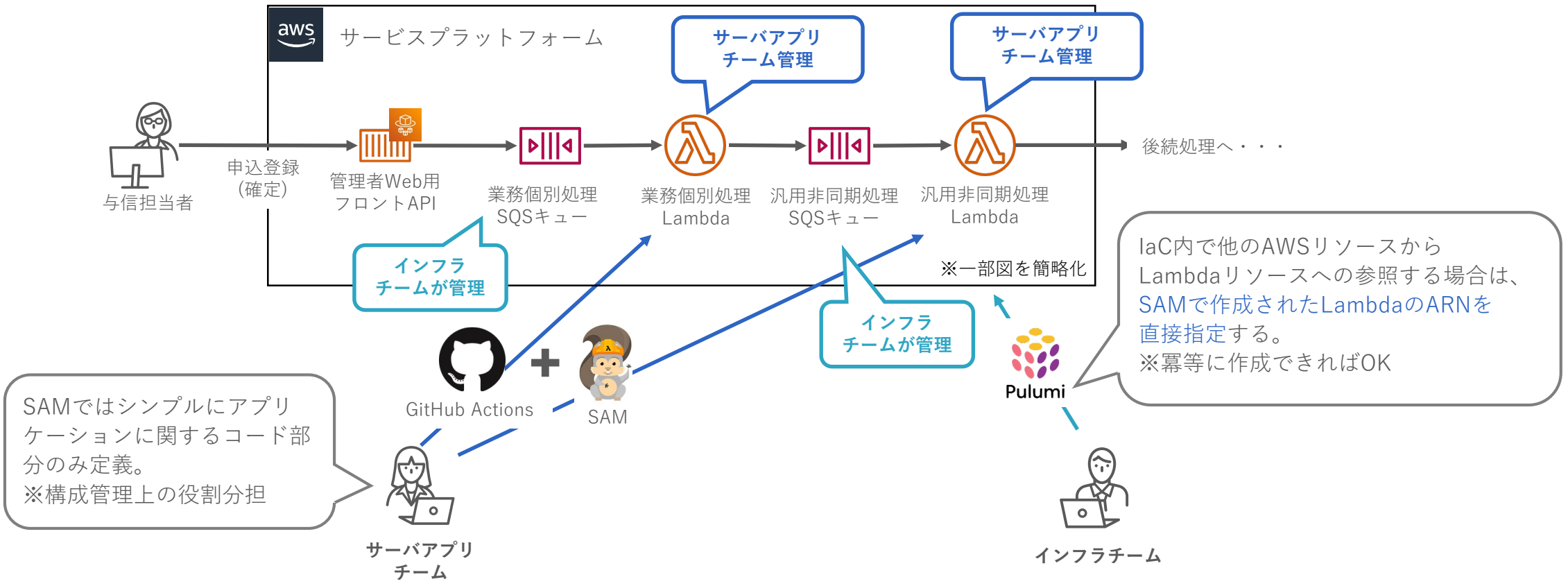


審査後の申込登録を行うと、Amazon SQS / AWS Lambdaにて非同期処理がなされる。

インフラチームは各種AWSリソースをIaC(Pulumi)でプロビジョニングしている。
一方、Lambda開発はサーバアプリチーム主体であり、IaCからの都度更新は適さない。



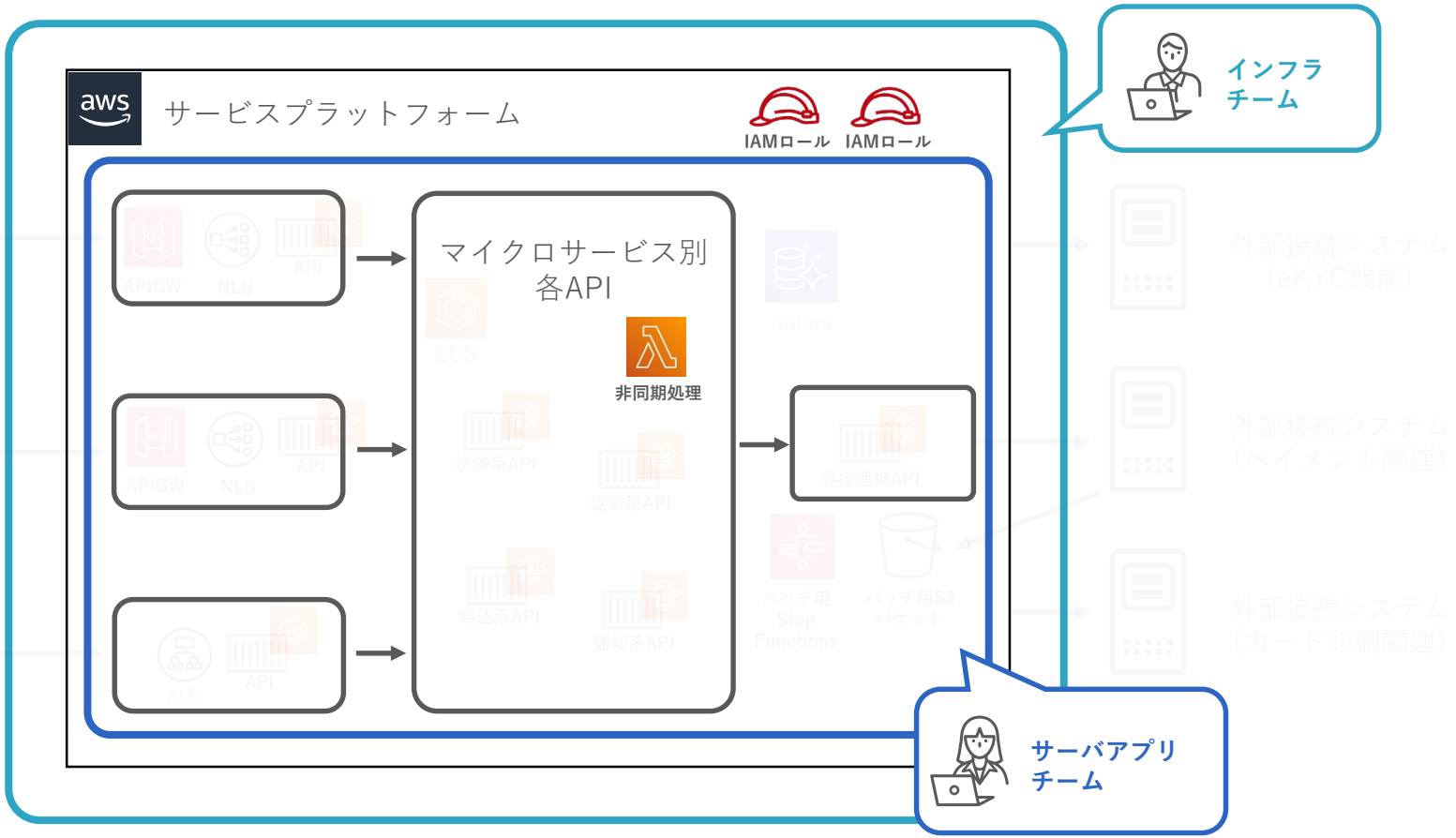
IaCですべてのAWSリソースを無理に定義しようとせず、他チームの開発ライフサイクルを考慮しながら妥協することも必要



本発表では、コンテナCI/CD、Lambda開発、Step Functionsによる バッチ開発を例に、ポテンヒットを防ぐために行ったコワーク設計を紹介

コワーク設計のテーマ

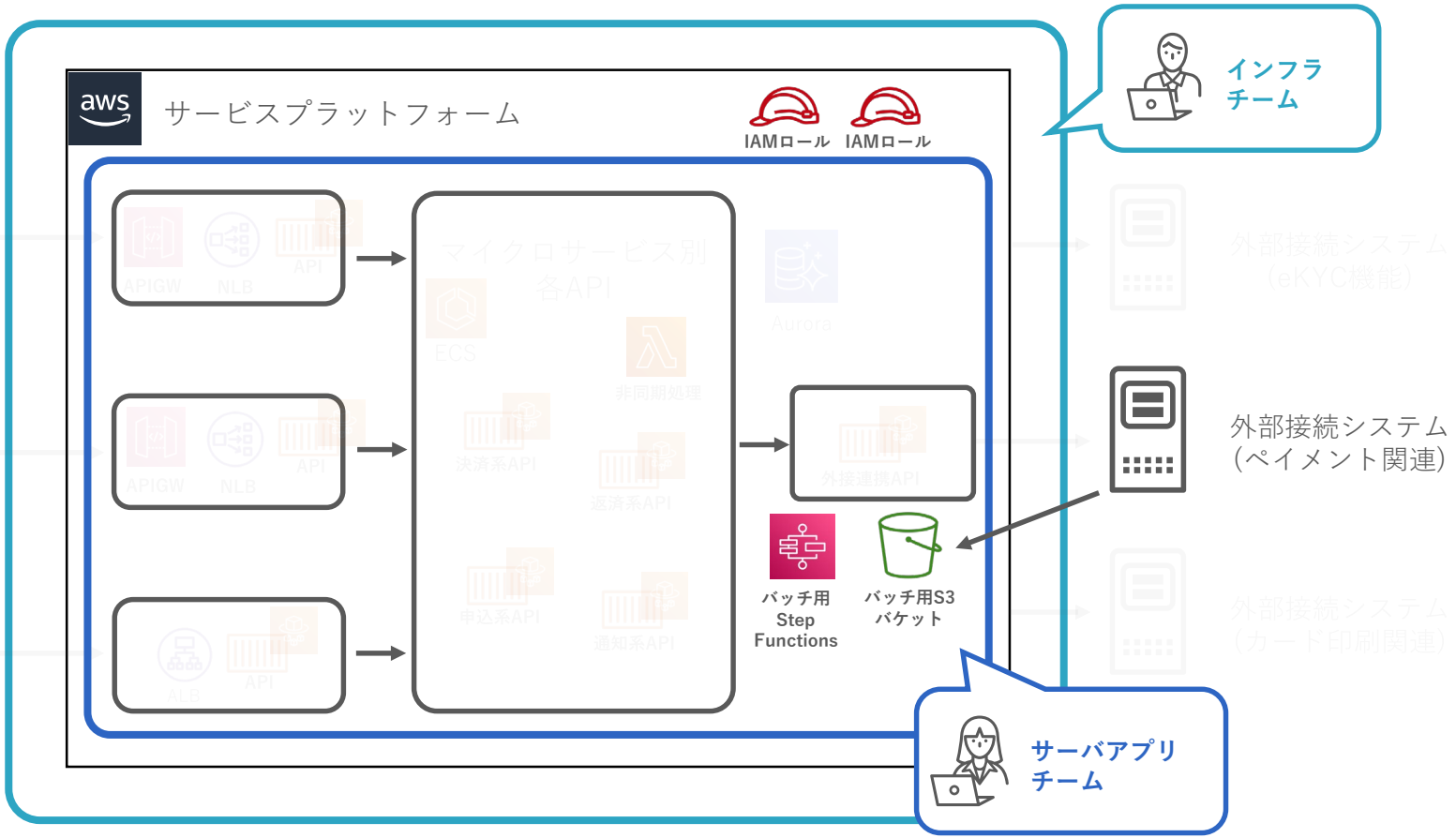
- コンテナCI/CD
- **Lambda開発**
- Step Functionsによる バッチ処理



本発表では、コンテナCI/CD、Lambda開発、Step Functionsによる バッチ開発を例に、ポテンヒットを防ぐために行ったコワーク設計を紹介

コワーク設計のテーマ

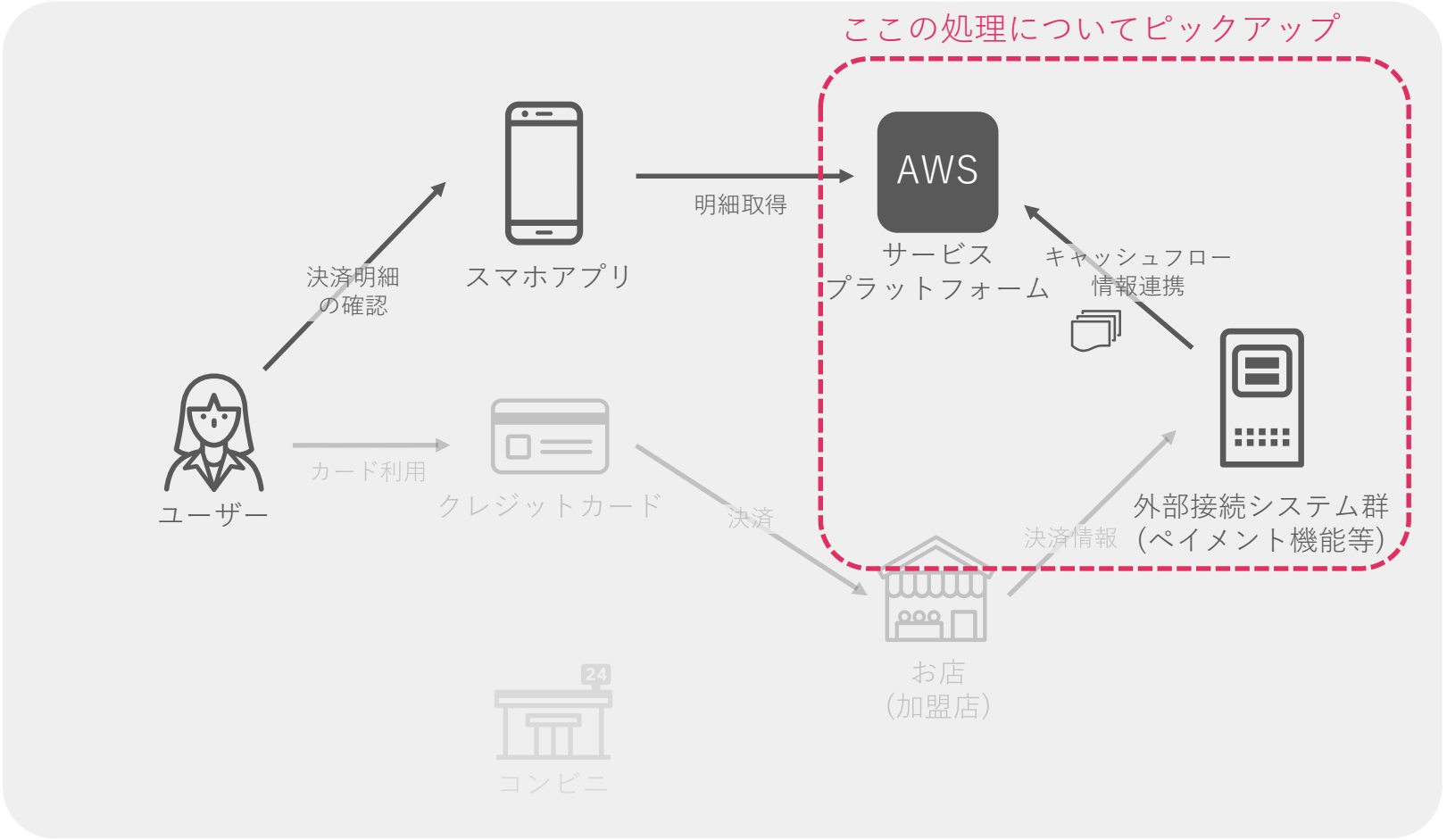
- コンテナCI/CD
- Lambda開発
- Step Functionsによる バッチ処理



再掲

外部システム側から連携されるファイルを取り込む関係上、いくつかのバッチ処理を実現しなければならない

金融系
スタートアップ



※お客様からは許可を頂いた上、本発表のテーマとして利用させていただいています。

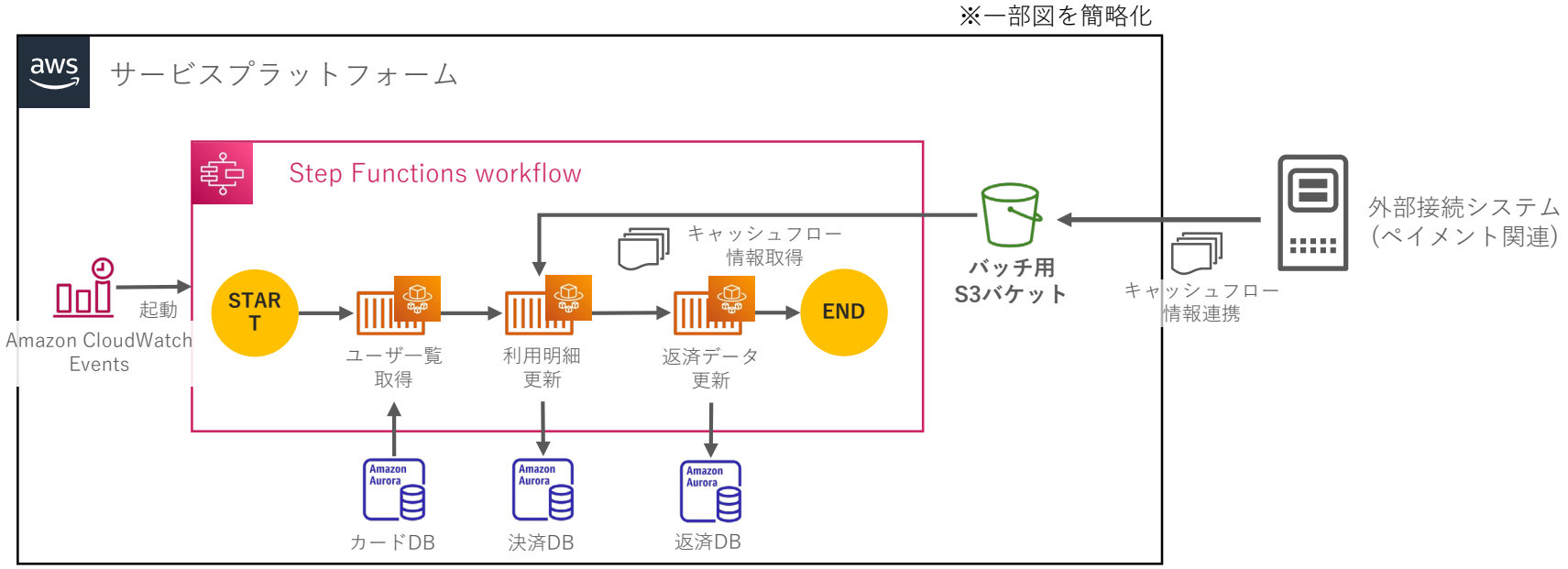
日次バッチ業務はStep Functions + ECS / Fargateにて処理

外部接続システムから日次でキャシュフローファイル(決済明細に関するデータ)が連携。

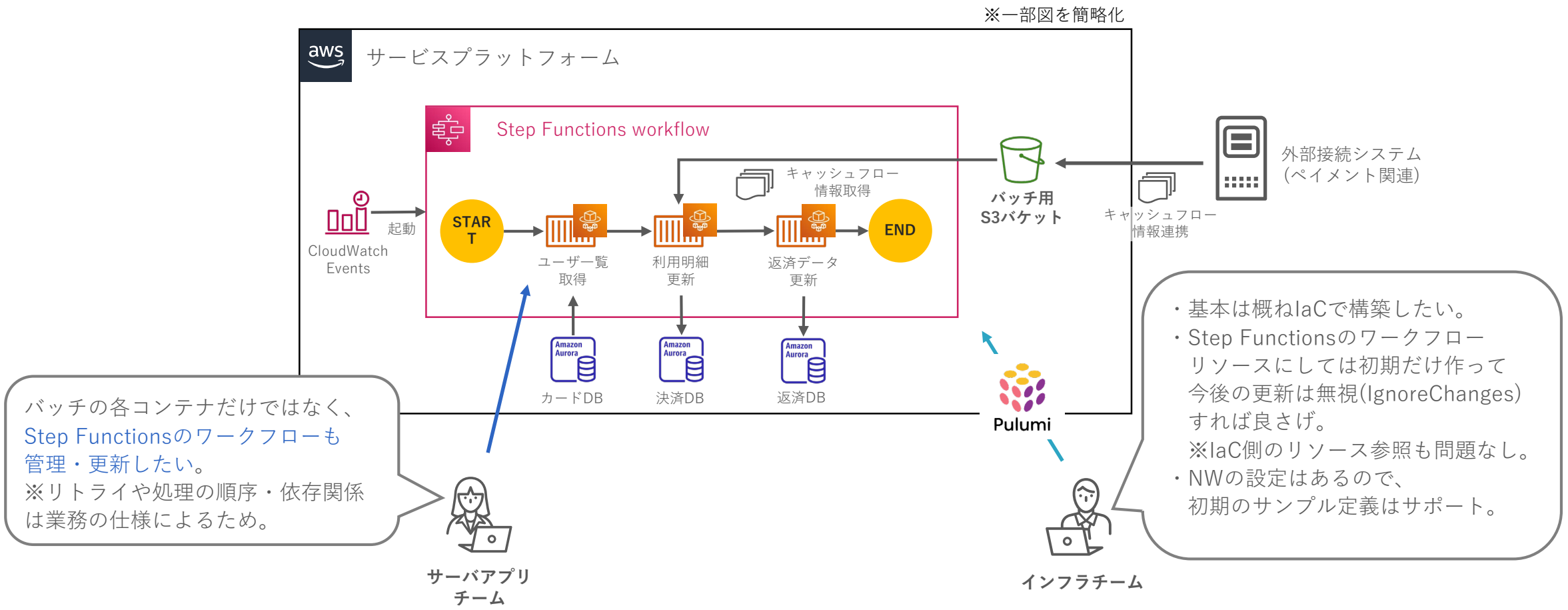
日次バッチ業務はStep Functions + ECS / Fargateにて処理

CloudWatch Eventsにより、日次時刻起動でStep Functionsが起動。
内部でECS/Fargateタスクが稼働し、バッチ処理としてキャッシュフローファイルの取り込みが行われる。

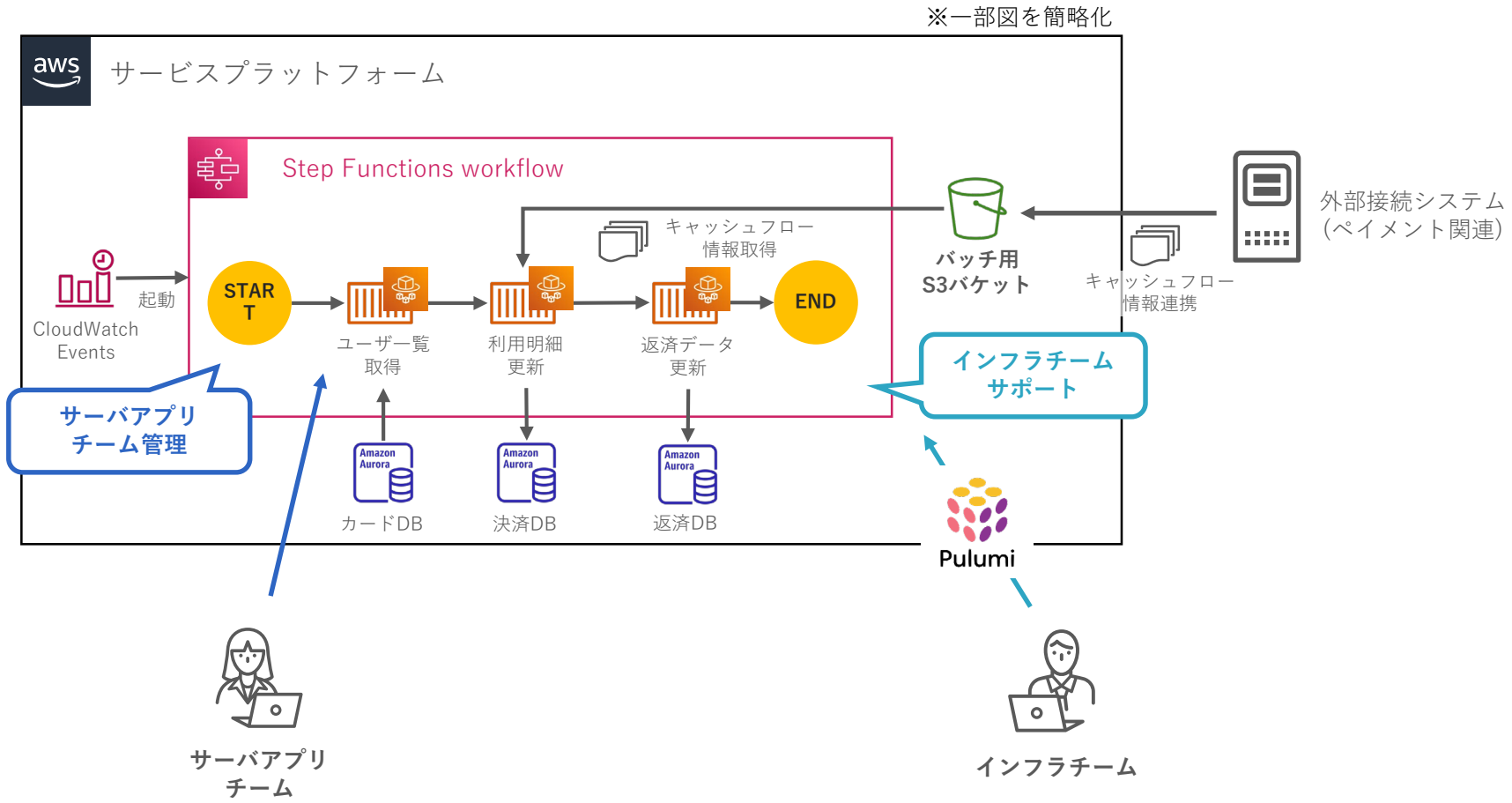
※一部図を簡略化



業務バッチにおけるStep Functionsのワークフローは サーバアプリ要件で内容が決まるケースが多い



Step Functionsのワークフローは
サーバアプリで主体的に管理する方針とし、インフラチームは適宜サポート



05.　チームとしてAWS開発を推し進めるために

AWSの各サービスが生まれた背景やユースケースをしっかりと押さえる

・今日では、AWSはマネージドサービスが多数存在している。
  ・インフラのレイヤが抽象化され、ビジネスにリソースを集中できる機会が増えてきた。
  ・ビジネスに注力できる = インフラを意識せずに、アプリ開発中心な状況が生まれる。
    ・インフラとアプリの境界がなくなっていき、役割分担も曖昧になる。

・AWSの各サービスがどのような課題を解決しようとしているのか(その課題を解決するのはどのようなロールか)を把握することが、AWS上でのチームコワーク設計の第一歩。

お互いのチームのロールを知り、互いのメンタルモデルを築く

・メンタルモデルとは、「個人にとって思考の前提を成すもの」。

→設計・開発・構築において相手のチームが期待することを前提として取り組む（思いやり一種）。

・**成果物やリソースの仕様ベースで会話をする。**

→相手チームの期待と自分達のチーム側のズレを調整する。

e.g. ECSタスク定義では、コンピュータリソースの他に環境変数なども指定できて・・・あ、アプリとインフラ両方にまたがるね、等

DX(Developer eXperience)を阻害することはチーム・組織として解決する

・**チームが担う成果物やAWSリソースの管理主体は適宜見直したほうが良いということも念頭に置く。**

→構築や運用の経験を積むと、スキルアップによりできることが増える。

→既存の役割分担のスキームが非効率と感じるようになる。

e.g. 自分たちでサクッと修正できる＆したいのに、相手チームがボトルネックになって・・・

・**なんでもかんでも縛りすぎない。**

→開発環境やサンドボックス環境を用意し、お互いの管理主体にアクセスできる状況を整えるべき。

→コンプライアンス要件やセキュリティ要件による制御は例外。

・**他のチームに貢献できる余地を残す。**

→スキルが身につけば、他チーム側にfeature作成してpull requestなど。

→プロジェクトチームや組織全体で強くなる。

06.　まとめ

まとめ

・金融スタートアップのプロジェクトを例に、
アプリチームとインフラチームに関するコワークの進め方を紹介しました。

・お互いのチーム間でポテンヒットをへらすために、
ECSやLambda, Step Functionsに対する取り組み方の一例に触れました。

・AWSにおいてチーム間でコワークするためには、
AWSサービスのユースケースを知り、そして相手のメンタルモデルを築くことが重要です。

最後に少しだけ宣伝・・・

AWSのコンテナに関する商業誌を執筆しました。
これからコンテナに触れる方も、AWS x コンテナでスキルアップを目指したい方もぜひお手に取ってください。

発売日　2021年10月21日(木)

AWSコンテナ設計・構築[本格]入門

著者名：佐々木 拓郎(監修) 新井雅也(著者) 馬勝淳史(著者)

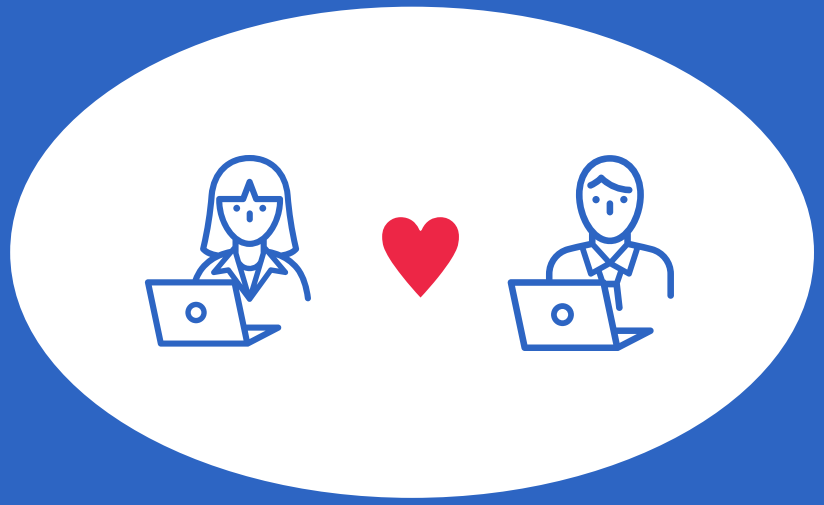
定価：3,300円（本体3,000円+税10%）　ご意見・ご感想

ISBN:	978-4-8156-0765-4
サイズ:	A5判
ページ数:	448
付録・付属:	-

購入する

Ref. https://www.sbcr.jp/product/4815607654/

Thank you for your attention.



BETTER TOGETHER WITH intel

aws DEV DAY

ONLINE JAPAN | SEPTEMBER 28-30,2021