マネージドサービスによる
アプリケーションセキュリティの実装

Ryuhei Shibata

Amazon Web Services Japan G.K.

自己紹介

柴田　龍平

アマゾンウェブサービスジャパン
シニアソリューションアーキテクト

SaaS / ソフトウェアベンダーのお客様や
セキュリティに課題をお持ちのお客様を中心に
技術的なご支援しています。

アジェンダ

AI コーディング時代のアプリケーションセキュリティの課題

AWS におけるセキュリティ確保のメカニズム

マネージドサービスの活用によるセキュリティ対応のオフロード

ビジネスの差別化要因としてのセキュリティ

コードは AI が書く時代

- 1940 年代の登場以来ずっと
  ソフトウェアは人の手で書かれていた
- 2024 年末ごろに登場した
  AI コーディングエージェントは
  自律的にソフトウェアを開発できる

ソフトウェアをつくる機械 の登場

コーディング支援のレベル

マイル
ストーン

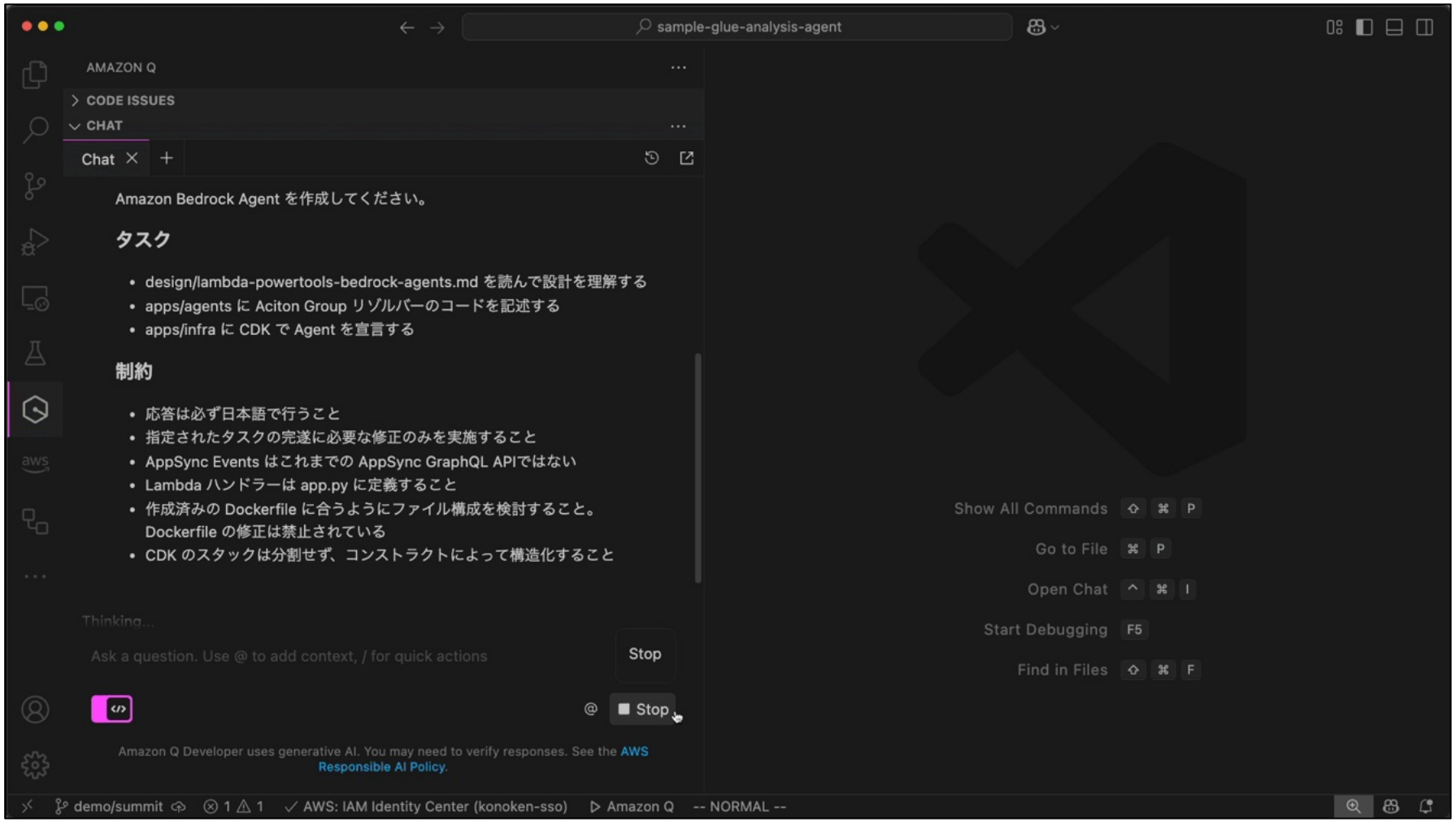
チケット

ブロック

行

AI Agent による
自律的な探索、
コード修正、テスト

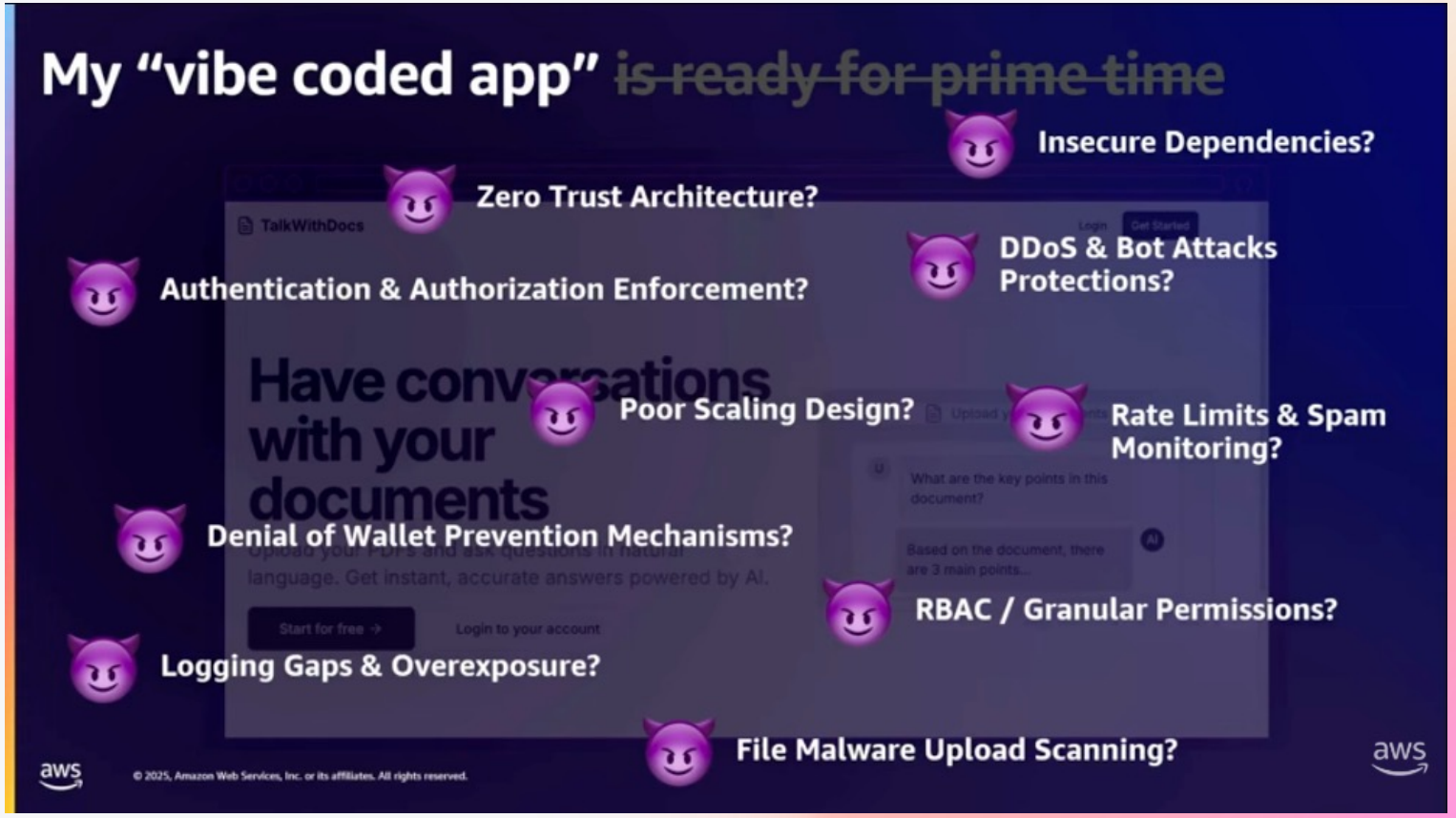
AI の作業単位



Amazon Q Developer in IDE - agentic coding

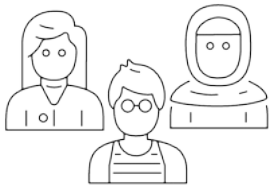
Vibe Coding したコードをそのまま本番稼働させられるか？

[参考] AWS re:Inforce 2025 - Why vibe coding isn’t enough: Building secure AI apps that scale (COM322)



https://youtu.be/0ANFTzktNXA?si=xb_B7pP51ULyEnKe

AI が書いたコードのセキュリティは誰が責任を持つ？



開発チーム

このプロダクトをセキュリティ観点で
レビューしてください

AI が書いたコードなので実装の詳細までは
把握しきれてないんですがまずいところの
指摘をいただければ修正します

AI が書いたアプリのレビューばかりで
工数が逼迫するのでサポートできません・・・
各機能についても詳しくないので・・・

セキュリティ
チーム

AWS では、すべての人がセキュリティのオーナー

"自分たちが作り、運営するもののセキュリティは自分たちのもの"

サービス
チーム

Owner

AWS
セキュリティ
チーム

Org owner

Enabler

”組織全体のセキュリティを管理し、サービスチームが安全にサービスを提供し、運用できるようにしていく”

Culture of Security

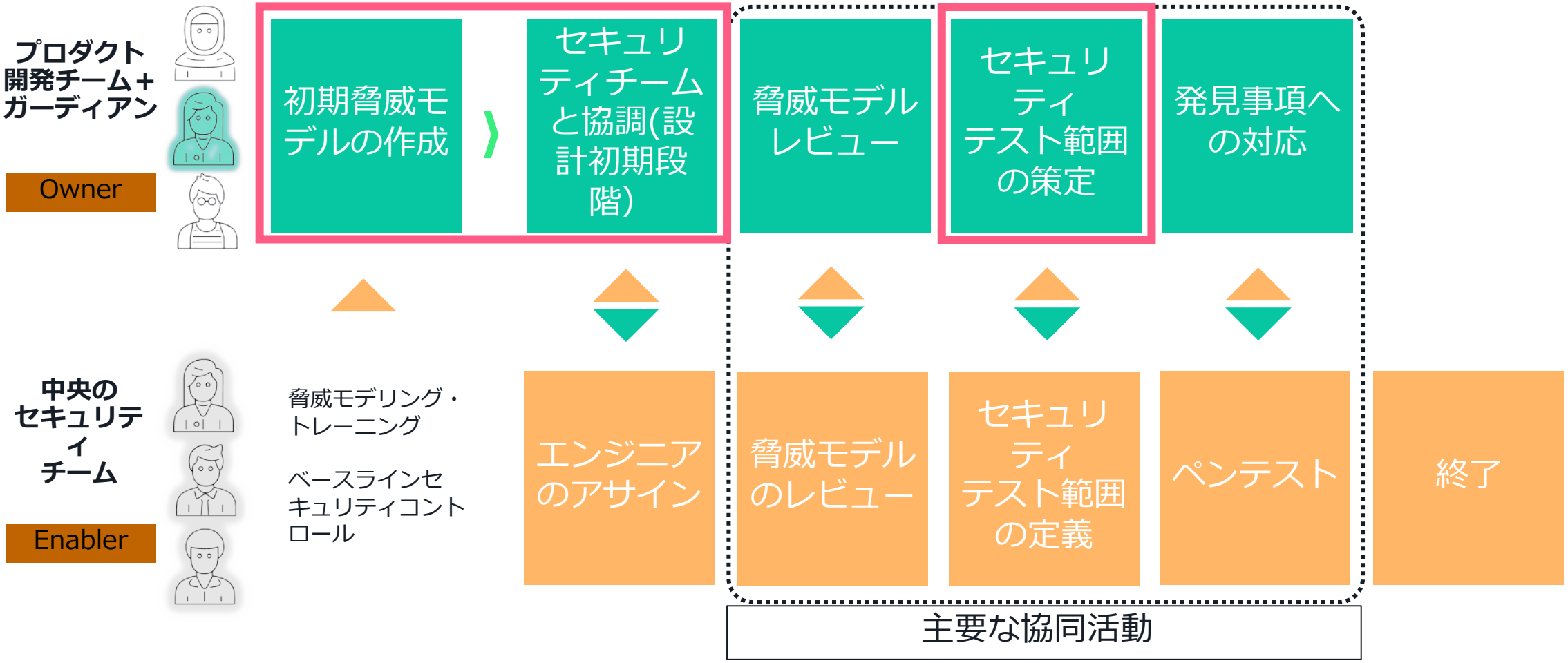
セキュリティにおいては**組織全体にわたるカルチャー**を育み、常にセキュリティを優先させる慣習が必要

セキュリティのカルチャーは一夜にしてならず、意図的かつ献身的な努力に基づく、一貫した強化と投資が必要

AWSは**長い時間をかけて**セキュリティのカルチャーを構築

AWS におけるセキュリティ確保のメカニズム

ガーディアンとのセキュリティ審査プロセスの迅速化

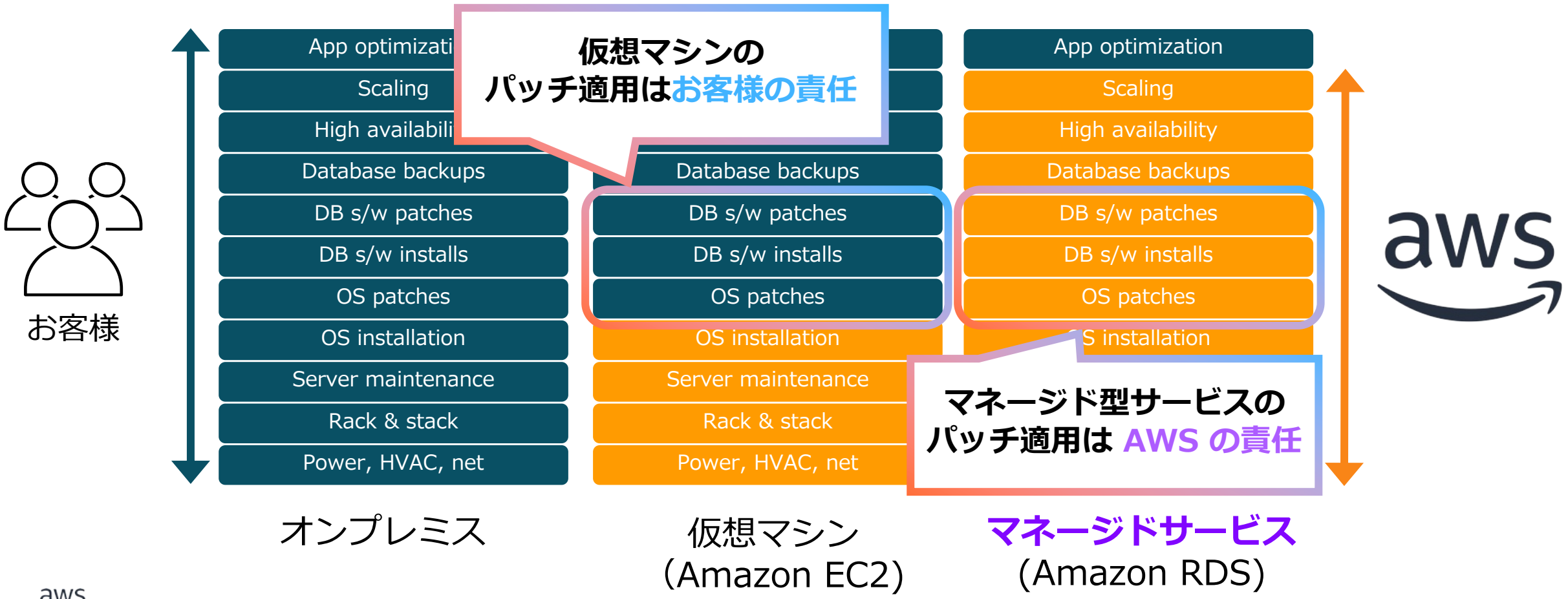


[再掲] 脅威への対応戦略を適切に考える

対応戦略	対策内容	対策例
1. 回避	**リスクを発生させないように変更を加える** リスクの可能性をゼロにするか、その重大性を無関係にする。	実装方法を完全に変更するなど。
2. 低減	**許容可能なレベルまでリスクを小さくする** 脆弱性の可能性を低くするように設計された手法 (セキュアな設計、特定のプログラミング言語、APIなど) を使用するなど。	認証情報の有効期間を短くすることで、認証情報が漏洩して悪用された場合の影響を低減することができる。
3. 移転	**リスクを第三者に転嫁する** サイバー保険に加入したり別の第三者の管理するコンポーネントに変更するなど。	AWS のマネージドサービスを利用することで、システムリスクの責任の一部を AWS に転嫁したものと考えることもできる。
4. 受容	**意図的にリスクを受け入れる** リスクを緩和するためにかかるコストがリスクを悪用された場合の代償よりも大きい場合などは、あえてリスクを許容することも考えられる。	-

マネージドサービスによるセキュリティ対応のオフロード

マネージドサービスを活用することで、お客様のセキュリティ対応範囲を減らすことができる

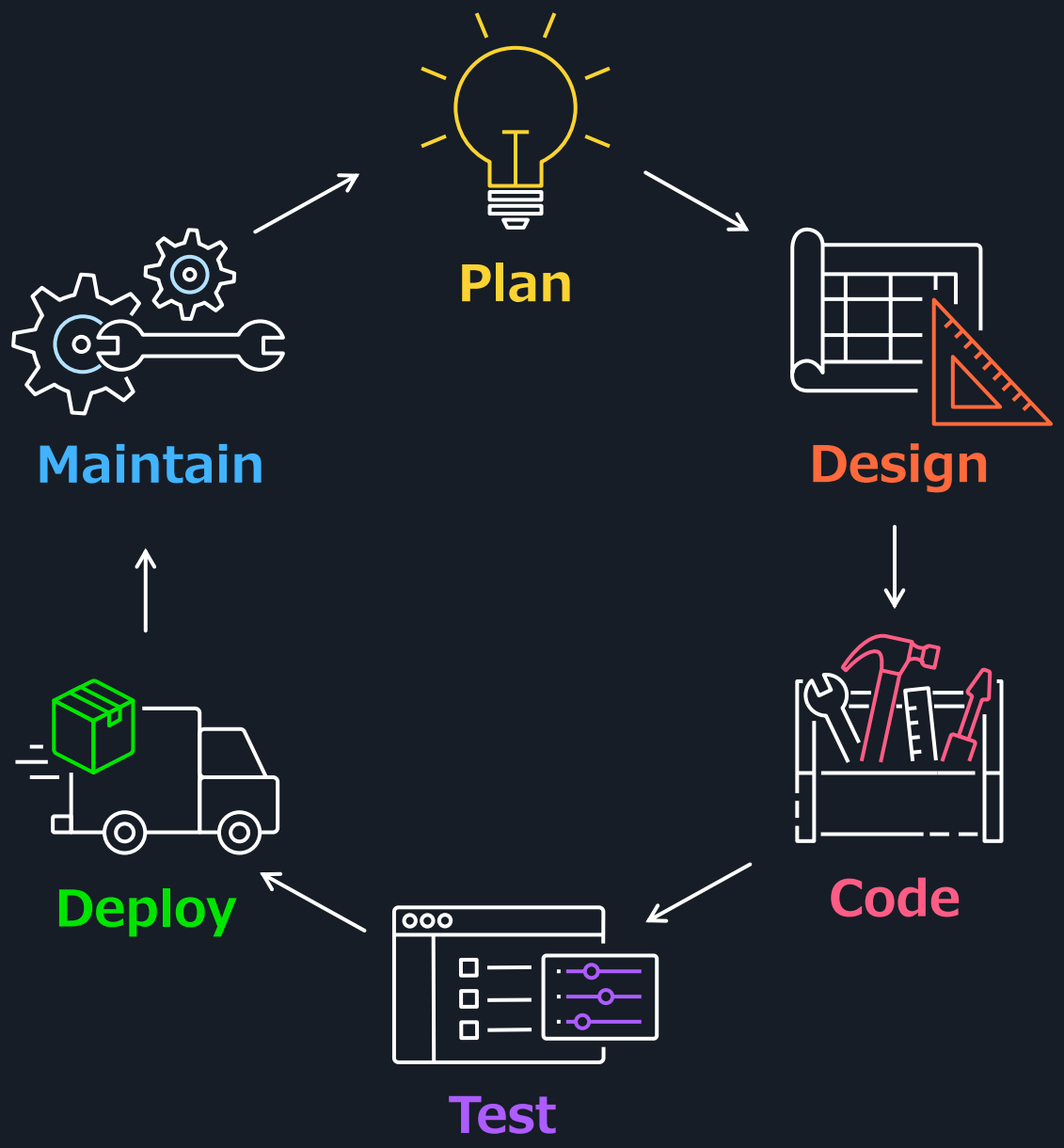


マネージドサービス vs 自前実装の工数の比較

開発チーム

どうせ AI が書くのであれば自前で実装しても
マネージドサービスでも工数は変わらないの
では？

初期の学習コストを下げたいので
とりあえずスピードを優先してマネージド
サービスを使わずに AI + 自前で
ミニマムな実装をします



ソフトウェア開発ライフサイクル

コーディングはソフトウェア開発
ライフサイクルのあくまで一部分。
リリース後の維持や改善に不要な
コンテキストが必要にならないように
初めからマネージドサービスや
スタンダード、ベストプラクティスに則った
設計を選択する

Undifferentiated Heavy Lifting

他との差別化になりづらい重労働

by Jeff Bezos, Founder and former CEO of Amazon.com

Undifferentiated Heavy Lifting

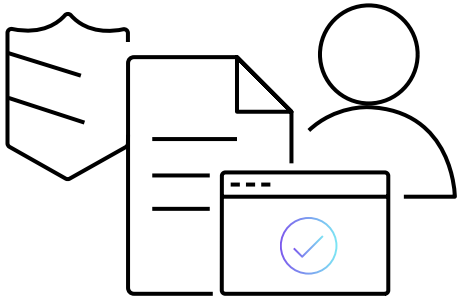
他との差別化になりづらい重労働

by Jeff Bezos, Founder and former CEO of Amazon.com

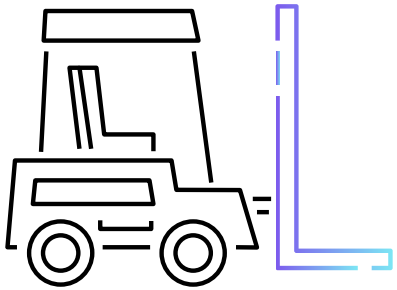
優れたセキュリティの仕組みは
本当に差別化要因にならないのか？

例えば・・・認証や認可

ユーザ管理の仕組みを
素早く実装したい

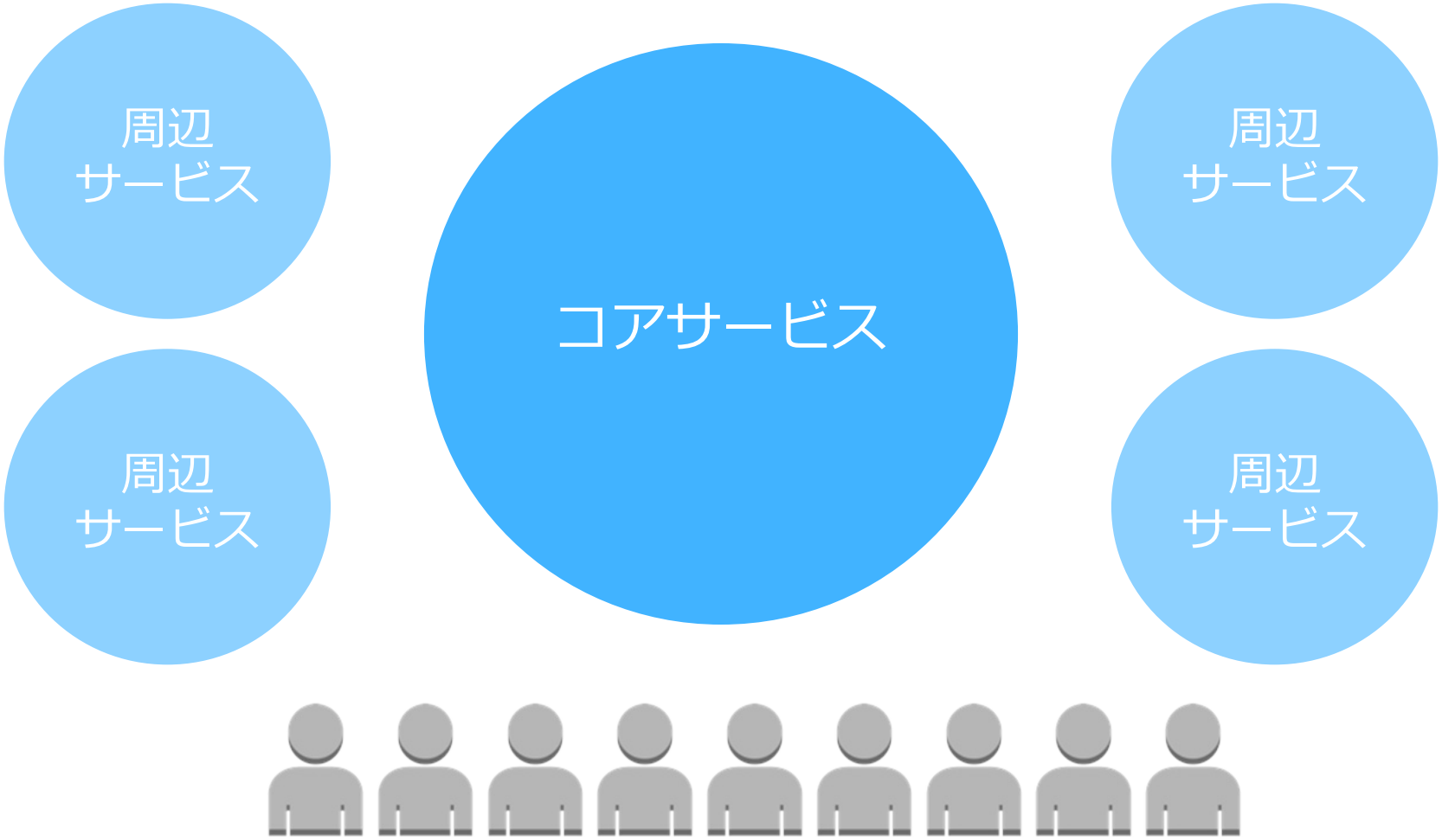


標準に基づいた認証・認可の
仕組みと先進的な
セキュリティを実装したい

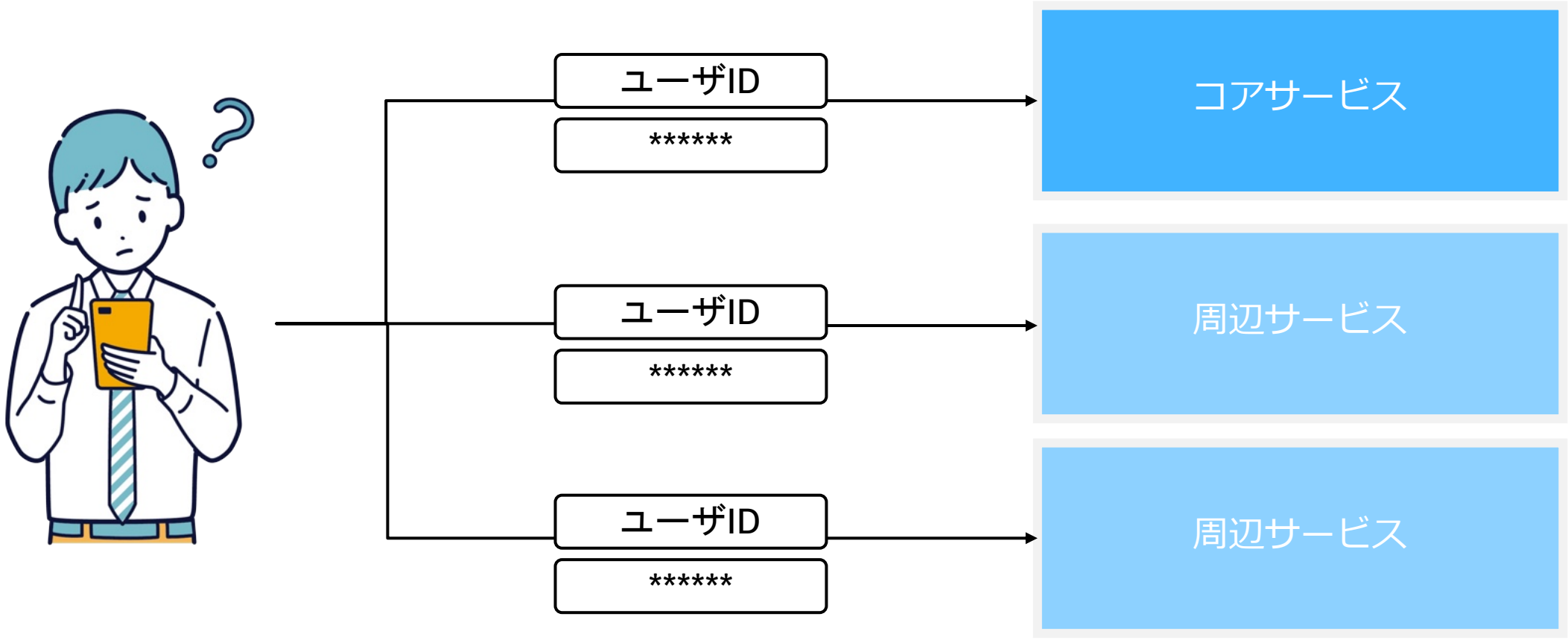


ビジネスの差別化要因
ではないにも関わらず
重厚で専門性を求められる
・・・

自社ブランド戦略とID管理

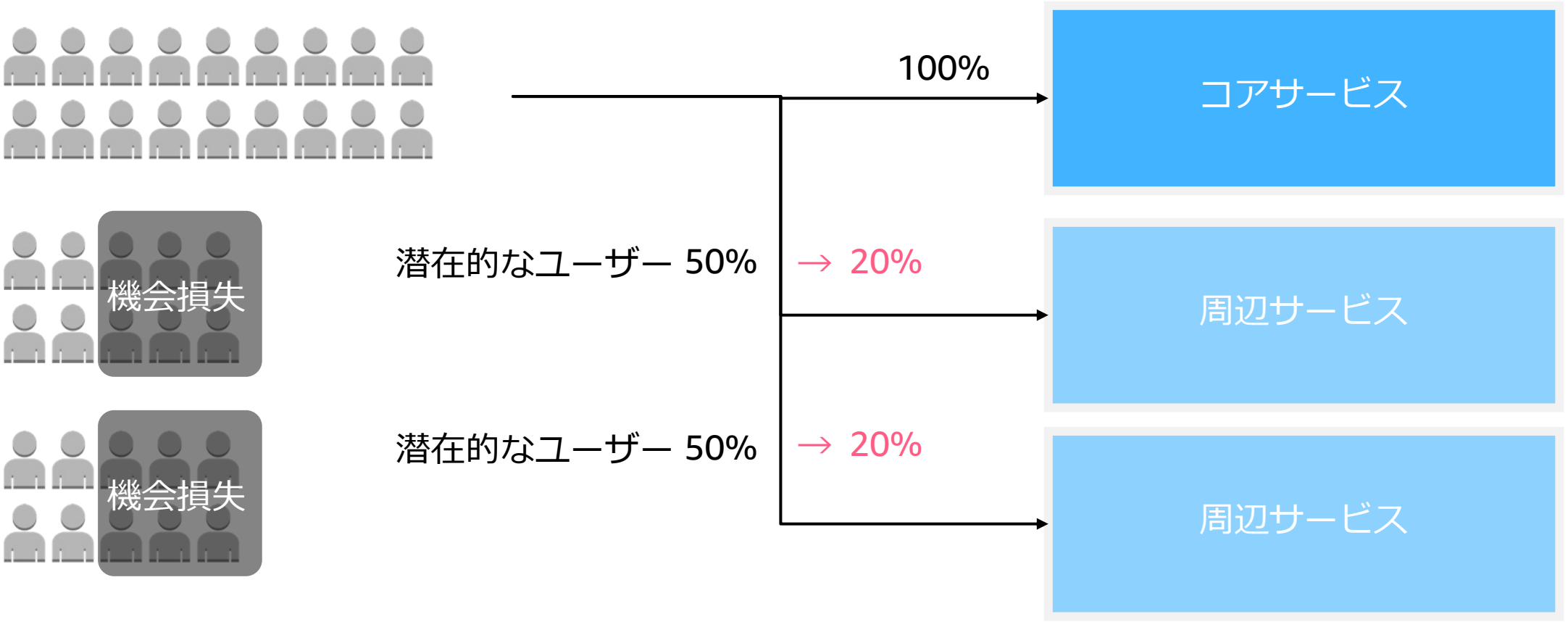


認証が十分に検討されていないと・・・



例：サービスごとに複数のユーザID・パスワードを作成

認証が十分に検討されていないと・・・



コアサービスの顧客基盤を引き継ぐことができない

自社ブランドのID戦略の重要性

コアサービス

例：ペット専門SNSサービス

周辺サービス

ペット専門ECサイト

「　ペットフード・シーツの配達は
便利でお得な定期購読を！　」

自社ブランドID

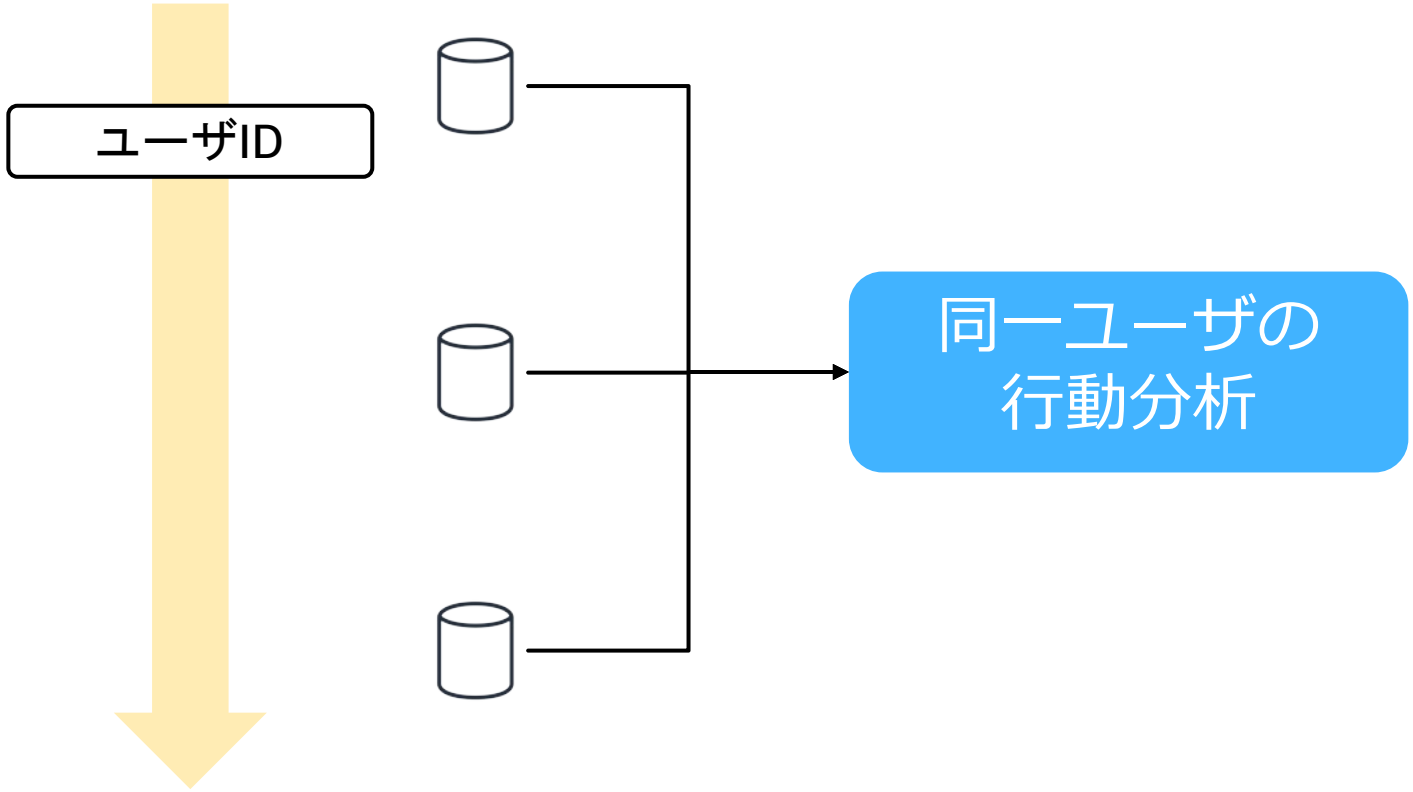
ユーザ認証を引き継ぐことで、シームレスに周辺サービスへ

自社ブランドのID戦略の重要性 - データ戦略

例：ペット専門SNSサービス

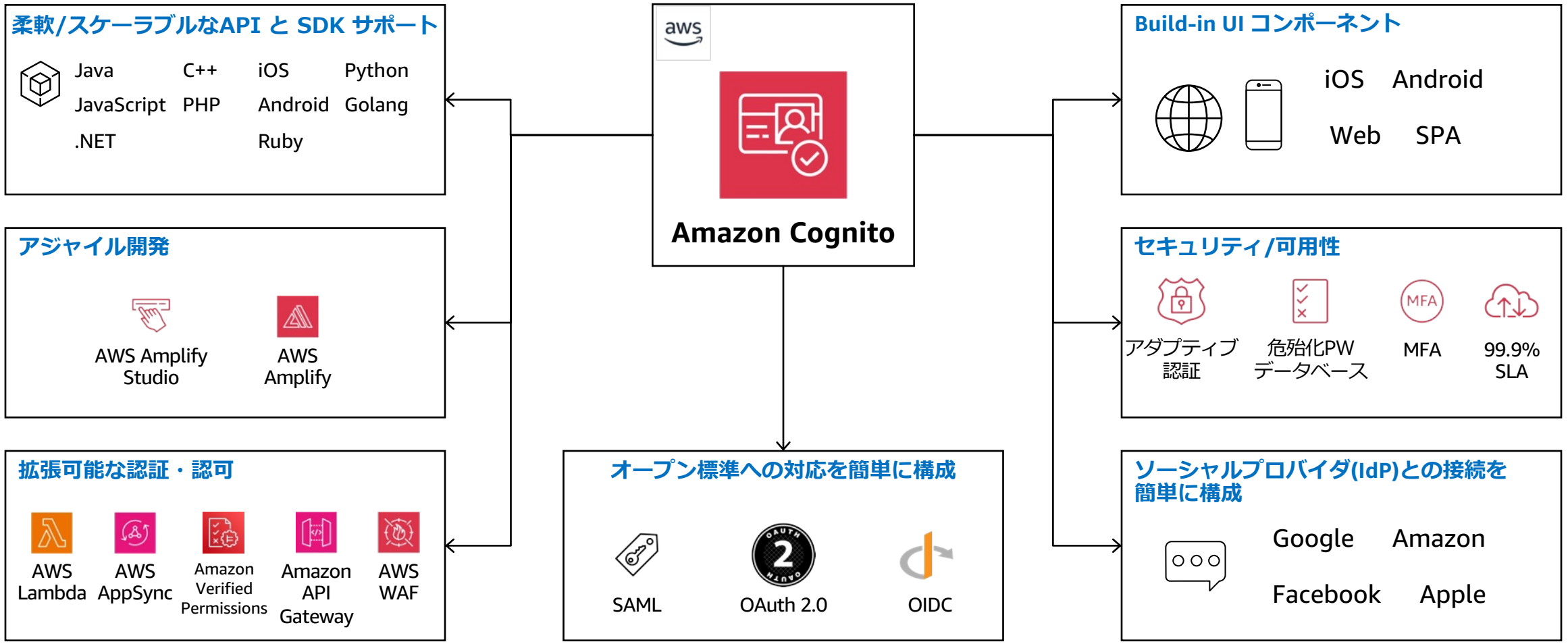
ペット専門ECサイト

レンタルサービス



横断的な行動データが獲得でき、より望ましいユーザー体験を提供することができる

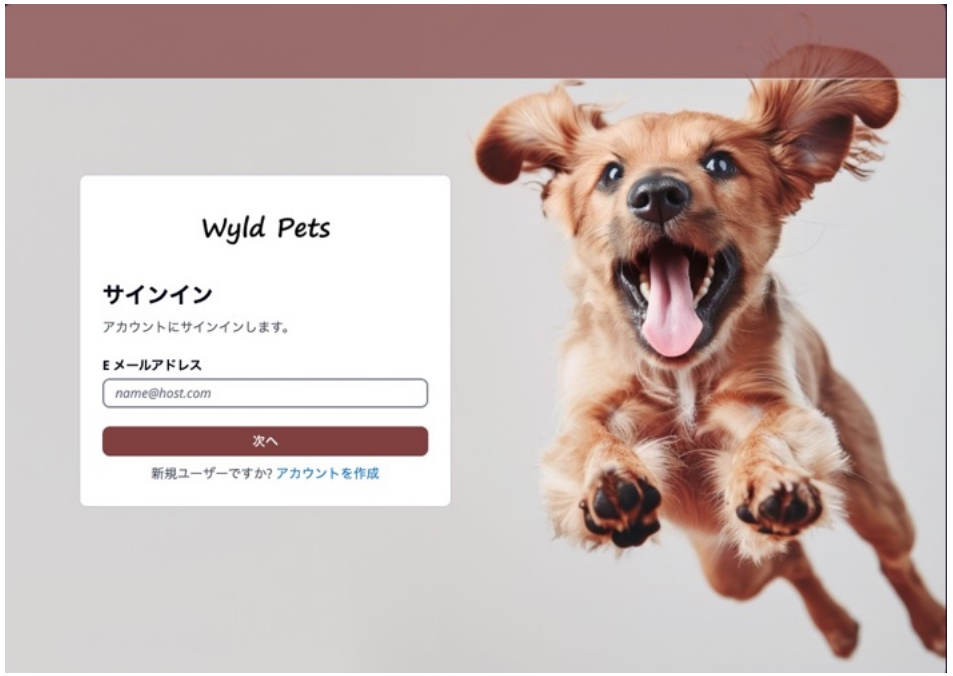
Amazon Cognito – 柔軟かつフルマネージドなアプリケーション向けアイデンティティ・サービス



Managed Login

Managed Login が提供する機能について

- 多要素認証(MFA)、Passkey、SMSとEmailのOne-time-password認証が可能
- OpenID Connect 互換のインターフェイスによる**シングルサインオン**
- ブランディングデザイナー機能で柔軟に企業のブランドに合わせた UI が作成できる
- **日本語含めた多言語に対応している**
- 外部 ID プロバイダーを使ったサインインが可能
- Essential、Plus プランが必要



OIDC 対応によってシングルサインオンを実現

標準仕様 OIDC (OpenID Connect) ベースでの実装により
複数サービス間で認証したユーザーの情報を連携し、シングルサインオンを実現

アプリケーションコードにおける認可の課題

認可

```
def get_book(request):
    if not db.query(request.bookId).isPublic:
        if not db.query(request.user).admin:
            if db.query(request.bookId).owner != request.user:
                return 'AccessDenied'

        if not request.multiFactorAuth:
            return 'AccessDenied'

    log("Handling book request " + request.id)
    book = db.query(request.bookId)

    return {
        'id': book.id,
        ...
```

ビジネスの成長に伴い、
アプリケーションのいたるところに
認可ロジックが分散し
開発の速度が低下

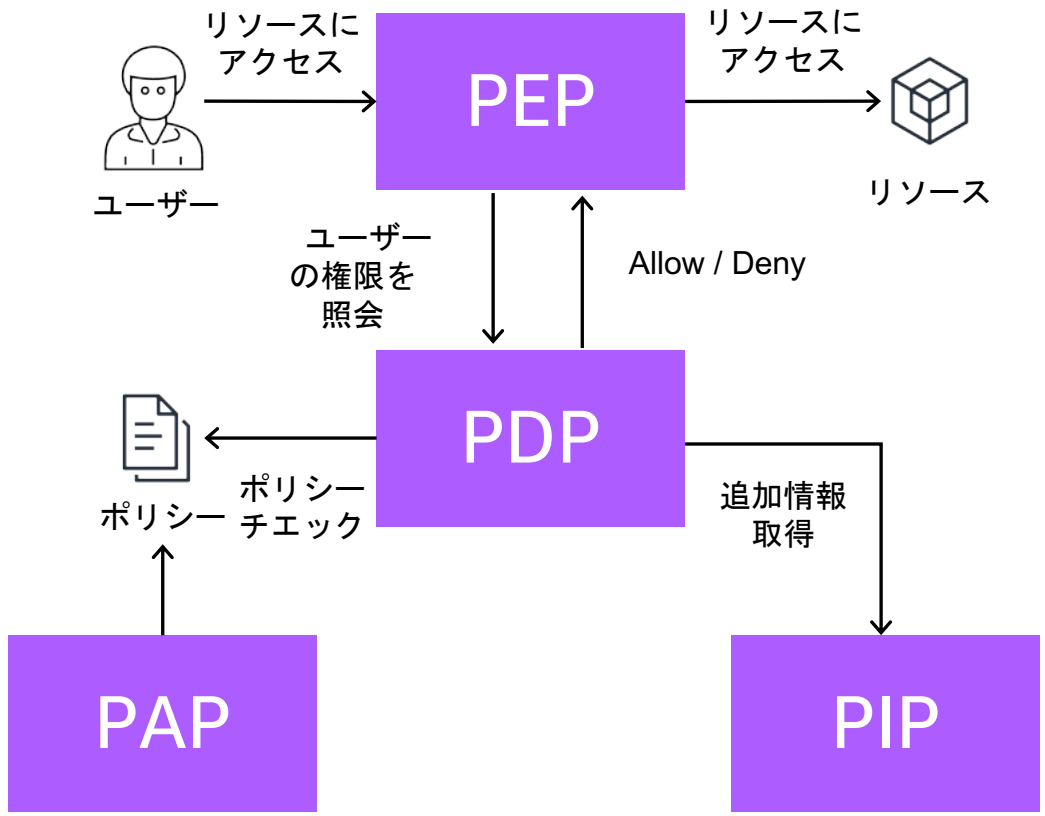
ロジックが複雑化し
ルールが合っていることの
確認が困難に・・・

認可ロジックを外部の仕組みに任せてシンプルに！

```
def get_book(request):
    if not client.is_authorized(...):
        return 'AccessDenied'

    log("Handling book request " + request.id)
    book = db.query(request.bookId)

    return {
        'id': book.id,
        ...
```



Amazon Verified Permissions

認可

外部化
ポリシーベースの認可モデル

アプリケーション構築
動的でリアルタイムな認可

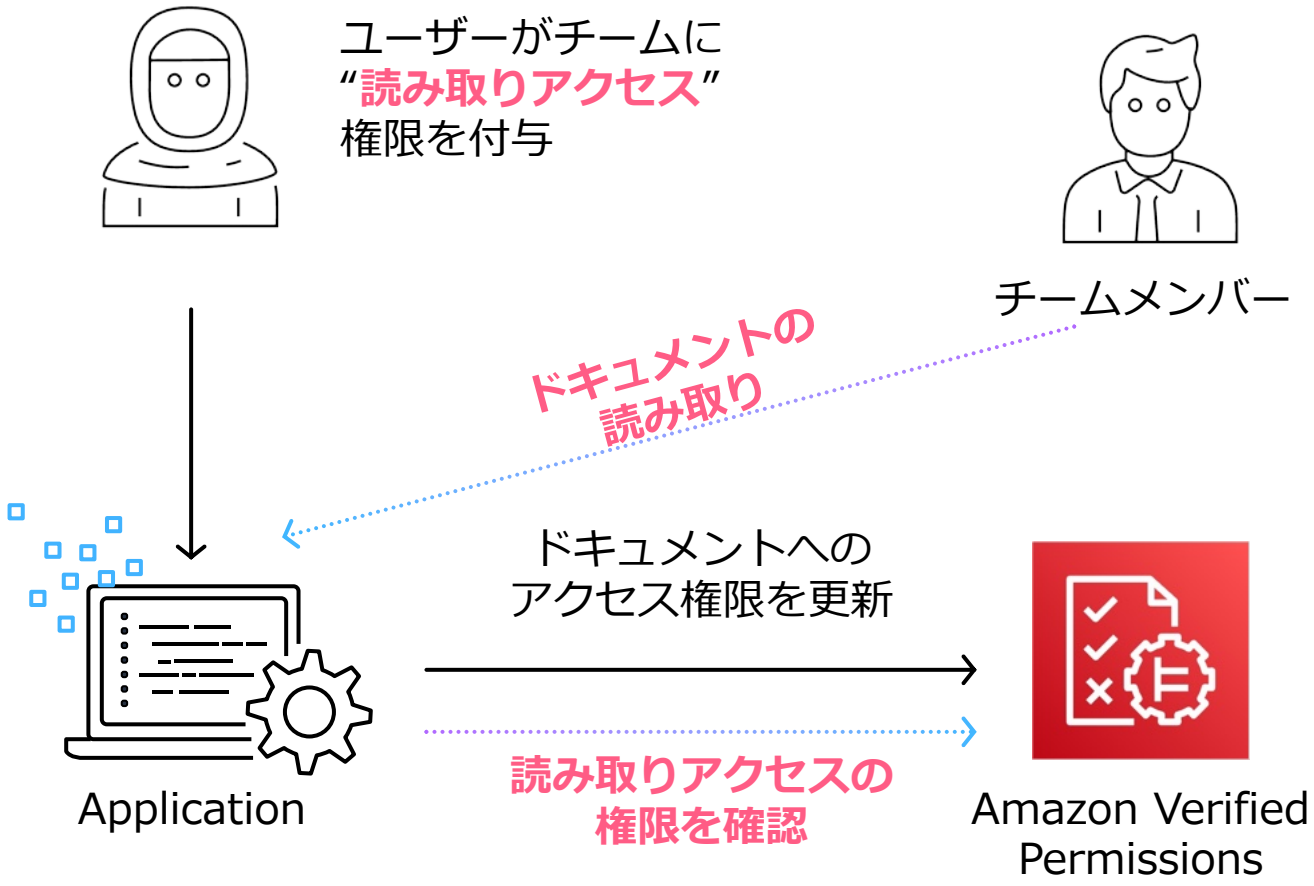
統合
アイデンティティプロバイダーとの同期

管理
きめ細かな権限設定

簡素化
監査や監視フローの簡素化が可能

分析
アプリケーションの権限を分析

認可の外部化によって、動的な権限制御を実現



分散された権限管理によって、
管理作業を効率的に

権限の変更と評価に関して
全ての **監査ログ** を取得

アプリケーションリソースへの
グループやロール、ユーザ単位
でのアクセス

優れた認証認可の仕組みは、
ビジネスの差別化要因であり

重要なビジネス戦略の根幹である

なぜなら、ユーザ体験 に直結するから

AI 時代でも「車輪の再発明をしない」判断を

開発チーム

AI エージェントが自前で認証機能や
認可ロジックを書こうとしている。

将来の運用性や拡張性を考えて
人間が同様の設計を行った時と同じように
車輪の再発明を止めるように誘導しよう。

プロンプト：「認証部分については独自実装
せずにマネージドサービスを利用します。
Amazon Cognito のドキュメントを読んで必
要な設計をしてください」

まとめ

AI コーディング時代のアプリのセキュリティをスケールする

- 脅威モデリング、シフトレフトというテクニックその前提となるカルチャーが重要

マネージドサービス活用によりセキュリティ対応をオフロード

- とりあえず・・・で AI に書かせるのではなく、初めからマネージドサービスやオープンスタンダードな設計を選択し、SDLC 全体の負荷を最適化

優れたセキュリティの機能はそれ自体が差別化要因になり得る

- マネージドサービスを活用し、車輪の再発明を抑制しながらユーザーの体験や利便性を向上

Thank you!