

# 生成AI時代にこそ求められるSRE

信頼性を勝ち取るための「コンテキスト」と「ガードレール」

山口能迪 (@ymotongpoo)

アマゾンウェブサービスジャパン合同会社

シニアアデベロッパーアドボケイト



# 自己紹介

山口 能迪 (やまぐち よしふみ)

アマゾンウェブサービスジャパン合同会社  
シニアデベロッパーアドボケイト

## 専門領域

- オブザーバビリティ
- SRE全般



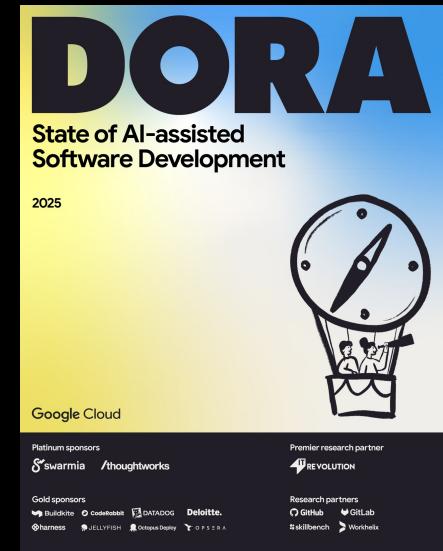
@ymotongpoo



# AI 利用の是非を問うフェーズの終了

Q: AI がコードや設定を書く時代に、SRE は不要になるのか？

A: SRE の重要性は、かつてないほど高まっている



90%

回答者が業務でAIを利用

<https://dora.dev/research/2025/>



© 2026, Amazon Web Services, Inc. or its affiliates. All rights reserved.

# AI は組織の能力を増幅するアンプ

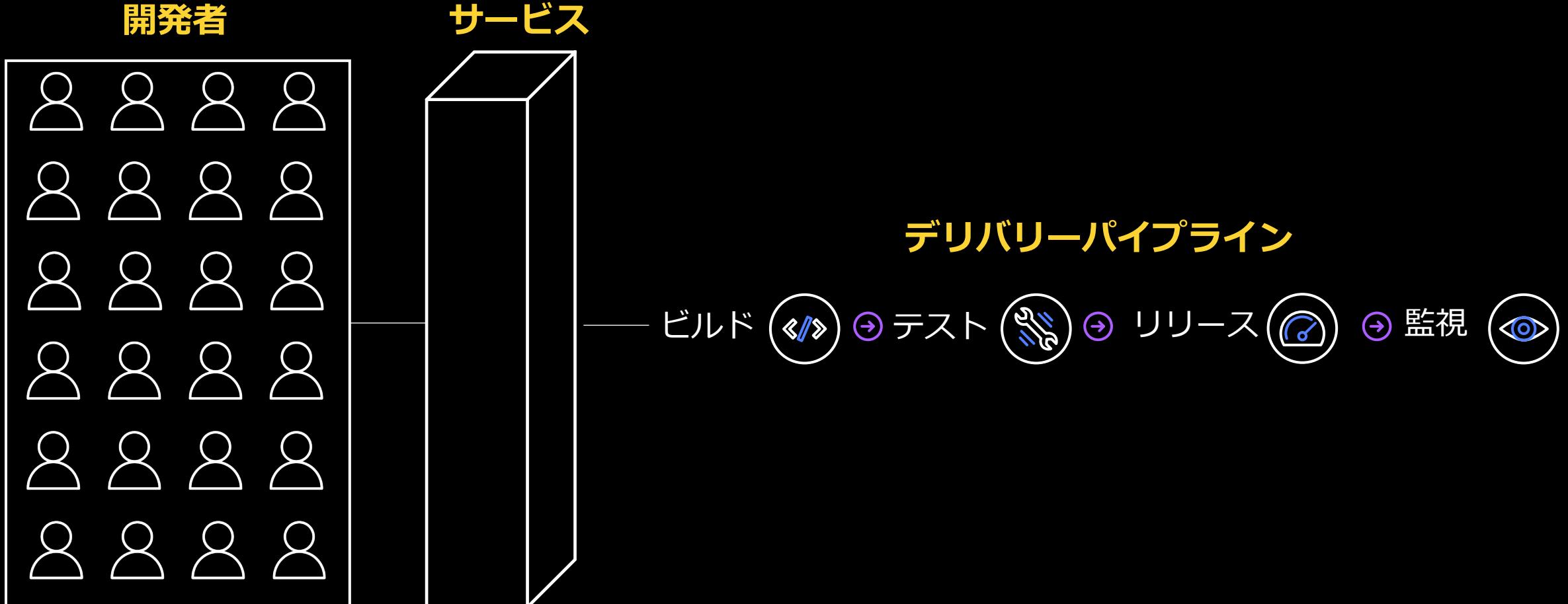
優れた組織はより強化され、課題のある組織は弱点を増幅させる

良いところも課題もすべてが増大

- 開発速度
- システムの不安定さ
- 変更失敗率



# SRE やモダンな開発は何を変えたのか



# SRE やモダンな開発は何を変えたのか



# AI 駆動な開発は何を変えたのか

こここの議論が多いが

今日はこここの話



# ソフトウェア開発は社会技術システム

ソフトウェア開発 = **人間系 + 技術系** の複雑な絡み合い

AI という強力な変数は広範囲に影響を与える

- コード生成
- チームのコミュニケーション
- 意思決定プロセス
- エンジニアの心理

## SRE の責務

AI の爆発的な生産性を、カオスではなく、持続可能なユーザー価値へと変換する

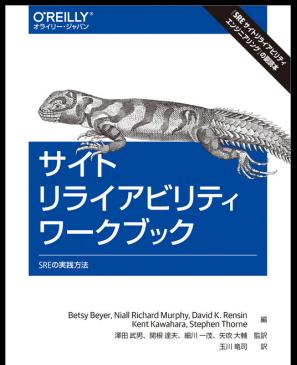
# コンテキストの補強

AIがよりよく動作するための「コンテキストの補強」はSREが求めてきたもの



オブザーバビリティを用いたデバッグでは、与えられたリクエストの周りのコンテキストをできるだけ多く保存し、斬新な障害モードにつながったバグを誘発した環境や状況を再構築できるようになります。

1章 オブザーバビリティとは？



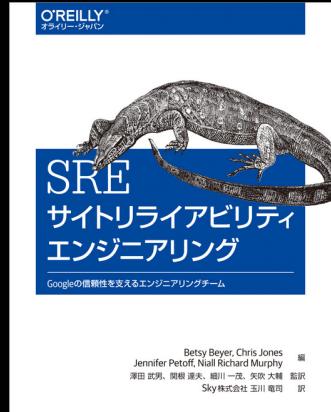
十分に考えられて選択されたSLOは、信頼性に関する作業の機会コストについてデータに基づく判断をしたり、その作業の適切な優先順位を決定する上で鍵となるということです。...SLOは、どのエンジニアリング作業を優先するかの決定を助けるツールです。

2章 SLO の実装



# ガードレールの強化

開発速度を維持したまま品質を維持する手法は SRE が勤しんできたもの



...機能フラグフレームワークが考え出されたプロダクトがあります。そのフレームワークの中には、新しい機能を 0% から 100% まで逐次的に新機能をロールアウトしていくために設計されたものもあります。... 深刻なバグや副作用があった場合、即座に変更を独立してロールバックできる。

## 27.4.2 機能フラグフレームワーク



SLI と SLO は、必要なときにちょっとしたガイドを示してくれます。『そこは、ああするよりもこうしたほうが、顧客のレイテンシーが改善されるはずです』であったり、『新しいバージョンをデプロイして本当にいいですか?』であったり、『これが、ユーザーにとって重要なことなんですね。私たちも注意を払ったほうがいいかもしれません』などです。

序文



AI 時代にあらためて SRE がもたらす価値を

- **コンテキスト:** AI がより良く動作するための下地
- **ガードレール:** AI の失敗を予防・回復するための保険

という観点から考える

# コンテキスト



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

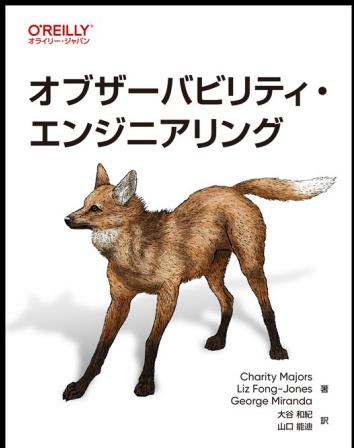
# コンテキスト1: オブザーバビリティ

AI (LLM) が知っているのは知識カットオフ時の一般的な情報だけ

**システム固有の情報**はコンテキストとして与えるしかない

- 動的な情報: テレメトリー
- 静的な情報: 仕様、アーキテクチャ図

# コンテキスト1: オブザーバビリティ



オブザーバビリティの本当の威力は、問題をデバッグする際に、**事前にそれほど多くのことを知る必要がない**ことです。システムをよく知らない（あるいはあまり知らない）場合でも、体系的かつ科学的に次々とステップを踏んで、答えを見つけるための手がかりを整然と追っていけるはずなのです。

8.2 第一原理からのデバッグ

## 従来のメトリクス

```
{  
  "payment_api_errors_total": 1500,  
  "cpu_utilization_avg": 0.85  
}
```

## 高次元なイベント

```
{  
  "status": "error",  
  "latency": 210,  
  "timestamp": "2026-01-06T13:30:15Z",  
  "service.name": "payment-gateway",  
  "trace.id": "abc-123-xyz-789",  
  "event.name": "stripe_charge_failure",  
  "customer.id": "cus_a1b2c3d4",  
  "payment.plan": "premium_monthly",  
  "app.version": "v2.1.5-canary",  
  "cloud.region": "ap-northeast-1"  
}
```



# コンテキスト1: オブザーバビリティ

既に AI 支援のテレメトリー分析機能が多く存在する  
eg. AWS DevOps エージェント

The screenshot shows the AWS DevOps Agent interface for an incident. It includes:

- Root causes**: A section listing the root cause of performance degradation due to an inefficient DynamoDB access pattern.
- Observations**: A section showing five observations related to the incident.
- Agent-ready spec**: A section providing implementation guidance for development and automated agents, including change requirements and a copy button.

Root CauseとしてDynamoDBへアクセス実装  
が不適切であることを正しく分析

裏付けるデータが示される  
(レイテンシーの悪化、タイムアウトの増加)

復旧案の提示

# コンテキスト1: オブザーバビリティ

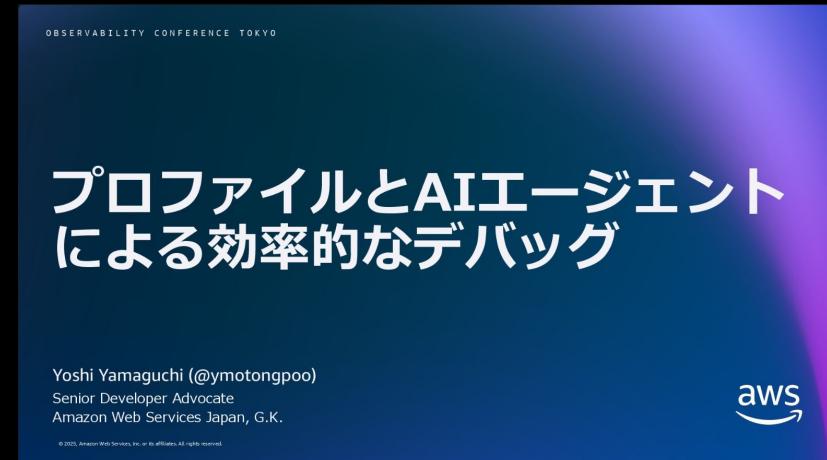
AIアシスタントを使って手元で連携することも可能

**MCPサーバー**

```
{  
  "mcpServers": {  
    "awslabs.cloudwatch-mcp-server": {  
      "autoApprove": [],  
      "disabled": false,  
      "command": "uvx",  
      "args": [  
        "awslabs.cloudwatch-mcp-server@latest"  
      ],  
      "env": {  
        "AWS_PROFILE": "my-aws-profile",  
        "FASTMCP_LOG_LEVEL": "ERROR"  
      },  
      "transportType": "stdio"  
    }  
  }  
}
```

**ローカルコマンド**

AIアシスタントにpprofを使わせてデバッグする例を  
Observability Conference Tokyoで話しました



# コンテキスト2: Infrastructure as Code

AI (LLM) が知っているのは知識カットオフ時の一般的な情報だけ

**システム固有の情報**はコンテキストとして与えるしかない

- 動的な情報: テレメトリー
- 静的な情報: 仕様、アーキテクチャ図

本当にこの通りに設定されているとは限らない

# コンテキスト2: Infrastructure as Code

AI (LLM) が知っているのは知識カットオフ時の一般的な情報だけ

**システム固有の情報**はコンテキストとして与えるしかない

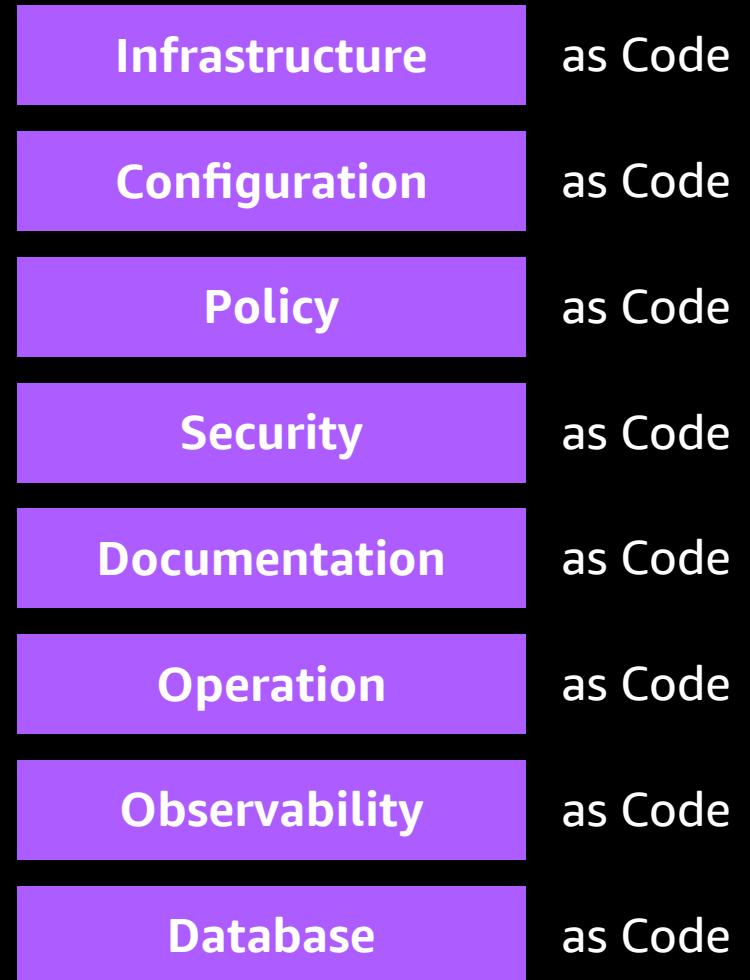
- 動的な情報: テレメトリー
- 静的な情報: 仕様、アーキテクチャ図、ソースコード、XaC (IaC、PaC、NaC...)

# コンテキスト2: Everything as Code

あらゆるものをコードで管理することで変更を追跡可能にし、再現性と一貫性を確保する



- **プレーンテキスト**で書かれたコードは人間にも AI にも理解しやすい
- **人間しか知らないことをなくす**
  - 手動での設定変更
  - 暗黙的なルールやオフラインの会話



# コンテキスト3: ポストモーテム

AI が作成を支援し、AI のためのコンテキストとなる

c.f. トイルの削減

## AI 導入以前

### 作成方法

1. Slack履歴をクロール
2. ダッシュボードのリンクを検索
3. 障害のタイムラインの認識の食い違い
4. 苦労してポストモーテムドラフト作成



## AI 活用後

### 作成方法

1. Slack、Zoom録画、アラート履歴、コミットログ、テレメトリーデータを統合
2. タイムライン自動作成
3. 過去の類似障害との比較
4. 根本原因分析の提案
5. ドラフトを自動作成

## 用途

人間が読む

## 用途

人間も AI も読む

# ガードレール



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

# ガードレール1: 密封ビルドとサプライチェーンセキュリティ

## 密封 (Hermetic) ビルド

ビルド環境に依存せず、常に同じ結果を再現できる、完全に決定論的なビルドプロセス

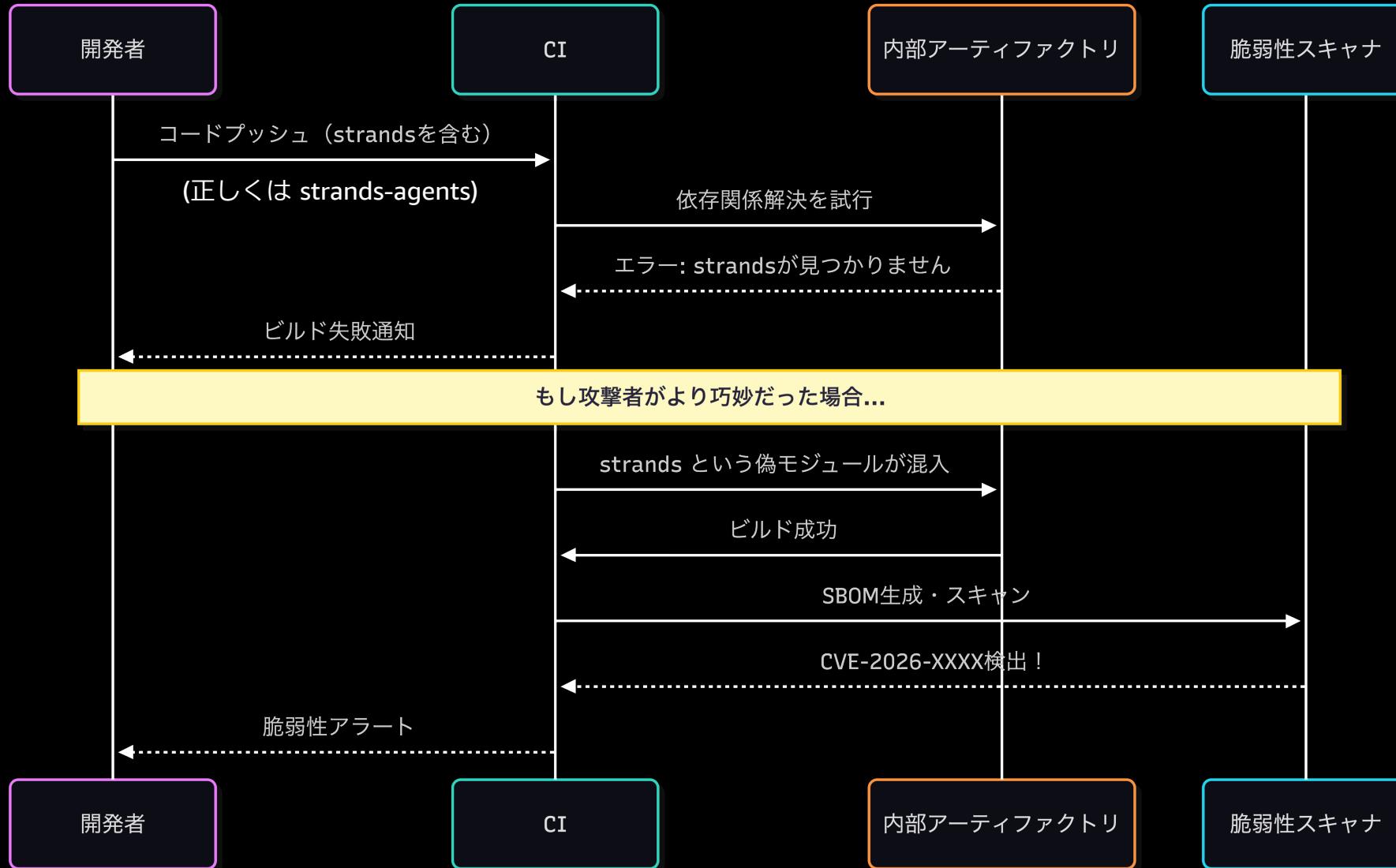
## サプライチェーンセキュリティ

ソフトウェアの企画、設計、開発、ビルド、テスト、配布、運用に至るまでの開発パイプラインにおいて、外部から混入されるマルウェアや脆弱性、悪意のあるコードからシステムを守るための対策

## AI で加速するリスク

- ハリシネーションによる存在しないライブラリ名
- 古いコードの混入による脆弱な依存関係

# 例: サプライチェーン攻撃の防御



# 対策: 多層的防衛

密封ビルドと検証の可能性を高める

- **内部アーティファクトリポジトリ:** 検証済みライブラリのみをミラー
- **SBOM有効化:** 全依存関係のリスト化と脆弱性スキャン
- **SLSA導入:** ビルドプロセス自体の信頼性証明

# ガードレール2: テスト

AIによるコードの生産量が激増 = 従来のレビュー・テストがボトルネックに

## 発展的なテスト手法

- ファジング: 予期せぬ入力での脆弱性やシステムダウンを発見
- プロパティベーステスト: コードが満たすべき「性質」を定義
- ミューターションテスト: ユニットテストの「品質」をテスト

→ AIによるデバッグのため  
のコンテキストでもある

## SRE の責務

発展的なテスト手法を開発者が「簡単に」「意識することなく」利用できる  
CI/CD パイプラインを構築・提供する

# ガードレール3: Policy as Code による統制

ポリシーをプログラムとして記述し、CI/CDで自動検証

AI が生成するコードやインフラ構成は、組織のポリシーを違反する可能性

## 主要ツール

- Open Policy Agent (OPA): Regoで汎用ポリシーを記述するポリシーエンジン
- Conftest: OPAラッパーの設定ファイルテストツール
- AWS Config: AWSリソースの設定の継続的監視

# ガードレール4: SLO に基づいたリリース

カナリアリリースや機能フラグとの組み合わせでユーザーへの影響を低減

## シナリオ例

1. 新バージョン v2 を 1% にデプロイ
2. SLO ベースの自動分析
  - 例: レイテンシーSLOを超過した
3. 閾値超過で自動ロールバック
4. 通知 + 自動起票

# ガードレール5: SRE 文化

AI の出力結果に対してノーと言える文化と組織作り

## 自動化バイアス (Automation bias)

人間が自動化された意思決定システムからの提案を優先し、それが正しくても自動化されていない矛盾する情報を無視する傾向。

- AI の提案は説得力があるが、常に正しいとは限らない
- エンジニアが AI に意義を唱えられる心理的安全性
- 非難のない文化の拡張
  - 「与えたコンテキストが何かを欠いていたのではないか」
  - 「AI の判断プロセスは改善できないか」

# おわりに



© 2025, Amazon Web Services, Inc. or its affiliates. All rights reserved.

# SRE は AI を安心して迎え入れるための土台

今日から始められる小さな一歩

- オブザーバビリティを推し進めてみる
  - トレースやイベント（構造化ログ）はありますか？
- SLOを定義してみる
  - ユーザー視点でサービスの信頼性を見れていますか？
- CI/CD パイプラインを見直してみる
  - 密閉ビルドしていますか？
  - どんなテストを走らせてていますか？
  - ポリシー検証してますか？



AI を正しく導くためにも  
SRE を実践して  
**コンテキストとガードレール**  
を手に入れましょう！

# AI で『Challenge SRE!』

大事なのは**信頼性**

- SRE に必要なツールや設定も AI に助けてもらえば良い
- 決定性があるツールや仕組みを作ることが重要

みなさんの SRE の実践が AI で加速することを願っています！

# AWS Builder Center でも知見を共有してください

