DevOpsを加速させるテスト、DevOpsで加速するテスト

ダイキン工業 テクノロジー・イノベーションセンター

前川 博志

はじめに

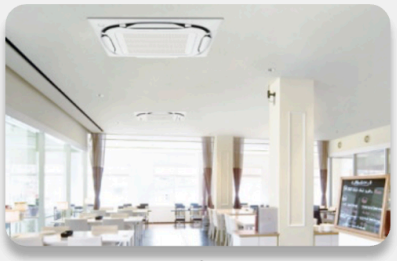
私はテスト/QAの専門家ではありません。DevOpsやSREといったプラットフォーム領域に強みを持つ ソフトウェア技術者 から見たテストの話をします。

ただ、私としてはテストやQAはソフトウェア技術者が最も学ぶべきものの一つと思っており、それを意識することが開発をどう前に進めていくのか、というのをお伝えできればなと思います。

会社紹介

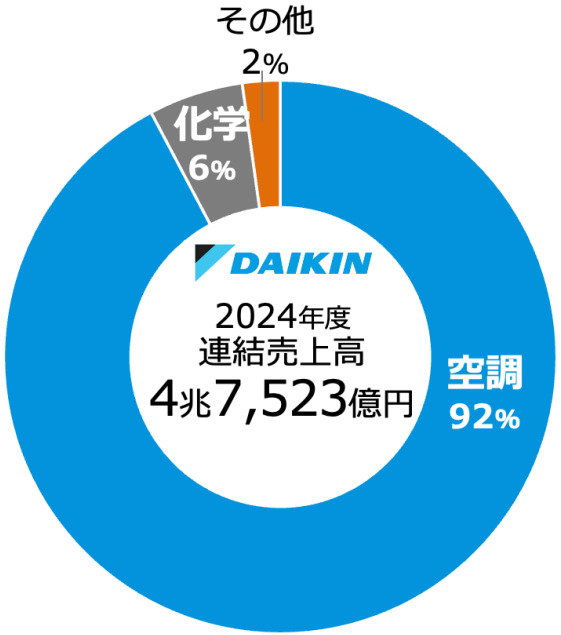
空調事業

住宅用



業務用

サービス



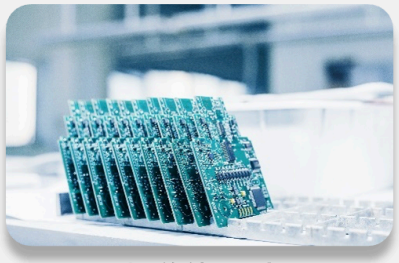
その他事業

油圧機器

酸素濃縮機

化学事業

冷媒



半導体用途

自動車用途

自己紹介

ダイキン工業 テクノロジー・イノベーションセンター 主任技師

（最近本社に引っ越しました）

基本的には「プログラマー」

- Typescript、Java、C#、Python
- AWS
- アジャイル、DevOps、Platform Engineering

社内外の活動

社内で取り組んできた仕事

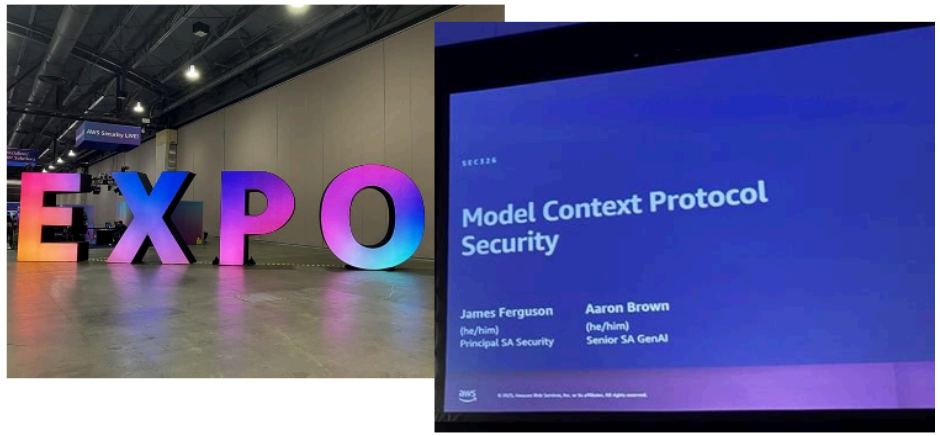
- 空調IoT基盤のSRE
- サービスマン支援システムのアーキテクト
- AWS設計指針 + リファレンスアーキテクチャ
- アジャイルコーチ
- 社内AWSコミュニティ主催
  - 

社外活動

- 京都アジャイル勉強会共同主催
- CCoE実践者コミュニティ関西主催

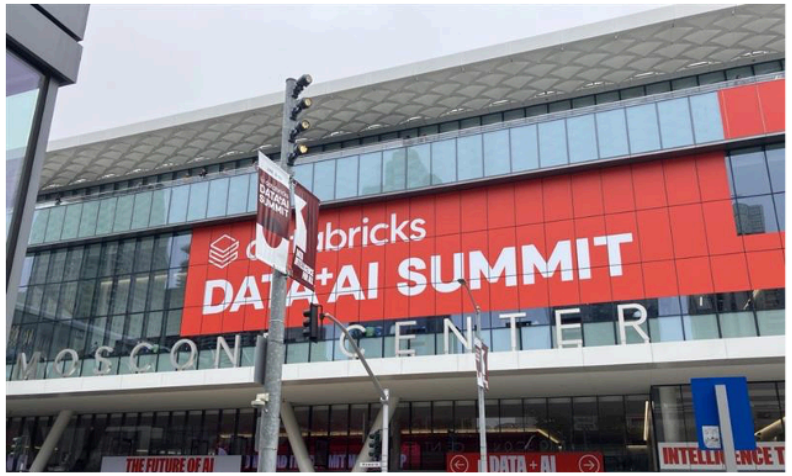
6月はなぜかあんまり日本にいませんでした

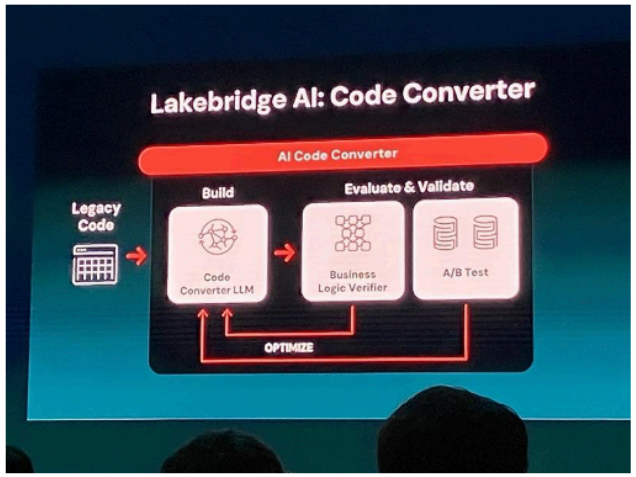
-Philadelphia!!



And last week in ...

-San Francisco!!





ところで、お久しぶりです！！

ソフトウェアテストシンポジウム 関西　2013　（JaSST '13 Kansai）プログラム

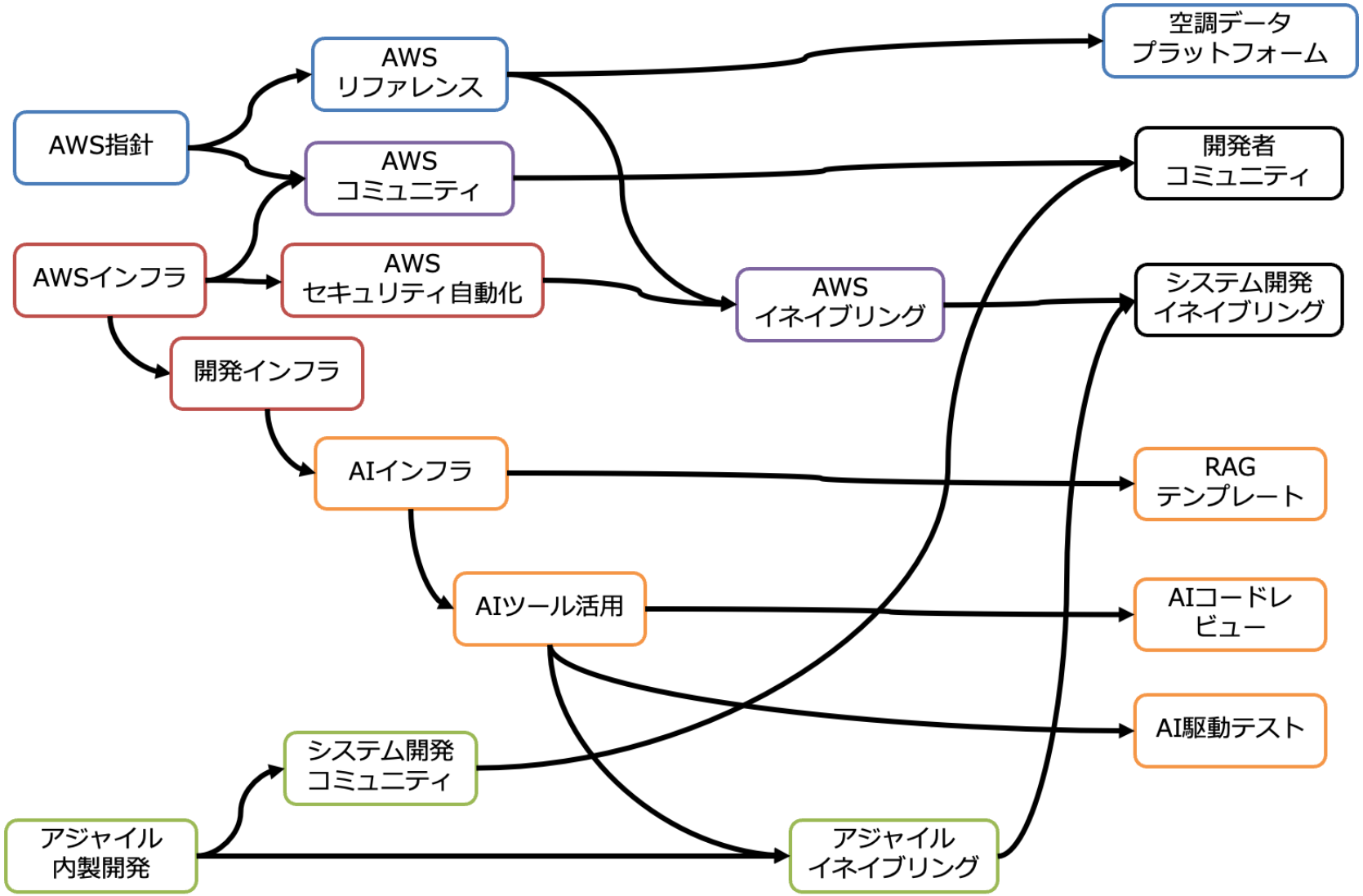
［やってみよう、広げよう、現場からの改善］

クリエイターズプラザ（大阪府 東大阪市）

（クリエイション・コア東大阪　南館3F）

事例発表1	セッション3-B1≪35分≫ テスト自動化セッション 「ようこそ、TABOKの世界へ」 前川 博志／井川 将（TABOK（テスト自動化知識体系）勉強会関西）
	セッション3-B2≪35分≫ テスト自動化セッション 「システムテスト自動化環境を構築してみた！ ～組込みシステムでの自動化の考え方～」 水野 昇幸（WARAI 関西ソフトウェアテスト勉強会実行委員）

今の仕事の全体像


AWS指針
AWSインフラ
AWS
リファレンス
AWS
コミュニティ
AWS
セキュリティ自動化
開発インフラ
AIインフラ
AIツール活用
AWS
イネイブリング
システム開発
コミュニティ
アジャイル
内製開発
アジャイル
イネイブリング
空調データ
プラットフォーム
開発者
コミュニティ
システム開発
イネイブリング
RAG
テンプレート
AIコードレ
ビュー
AI駆動テスト

DevOpsを加速させるテスト

組織内でのDevOps適用は結構大変

もはや当たり前の言葉にもなったDevOpsだが、その取組を組織の中に広げることは（とくに大企業では）難しい

組織のサイロ化（運用組織は別会社、ということも多い）

- 社員だけで24/365の体制を作れるのか？
- そもそも運用がコア業務と認識されていない

一般的に言われるDevOpsの利点を大きく主張しづらい

- 「4keys」といったDevOps指標の改善
- 高速なフィードバックを回す

「開発を加速させるDevOps」は難しい

デイリーリリース？なにそれおいしいの？

４ Keysを向上したら何がうれしいの？そもそもリリース回数上げるほうがいいとかとか正気？仕事増えるんだけど！

「結局アジャイルみたいな適当な開発すればいいってことでしょ。わたしたちの組織には不要かなぁ」になる。

4keys再訪

DORA（DevOps Research and Assessment）が提唱した、
DevOpsの成功を測る4つの主要指標。
「LeanとDevOpsの科学」などで有名に

- リリース数
- 開発リードタイム
- MTTR
- 変更失敗率

開発の効率性（リリース数・開発リードタイム） x 品質保証の効率性（MTTR・変更失敗率）

しかし現実は開発の効率性が優先されて語られていないか？

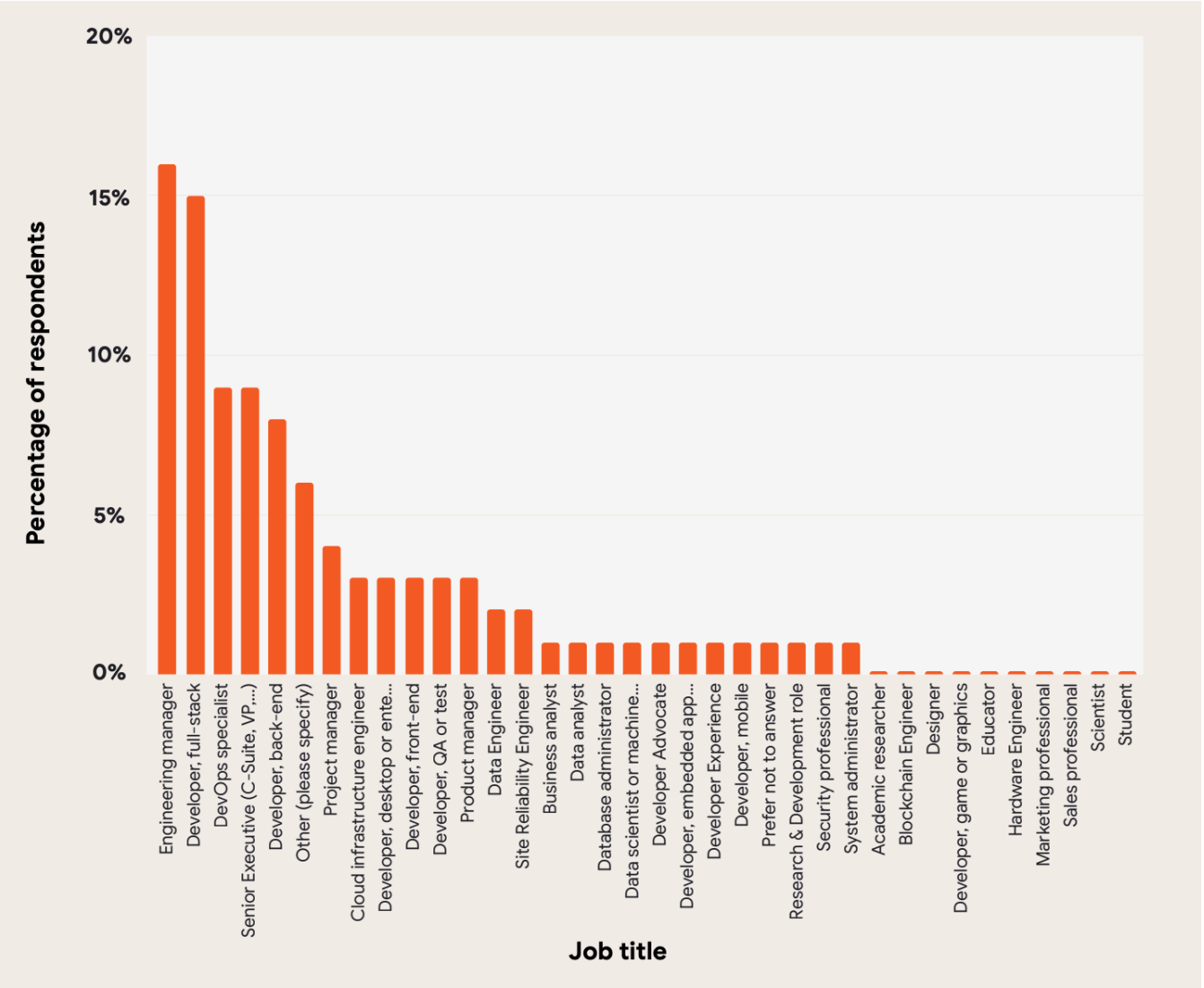
DevOpsの「不都合な真実」

多くの組織は「開発主導」のDevOpsになる

- IT運用側は通常業務に忙殺
- 故に最新の開発スキルも獲得しづらい
- 良くも悪くも守りのIT運用になる傾向が強い

「開発主導」ならばいいが、行き過ぎると 暴走 になる。

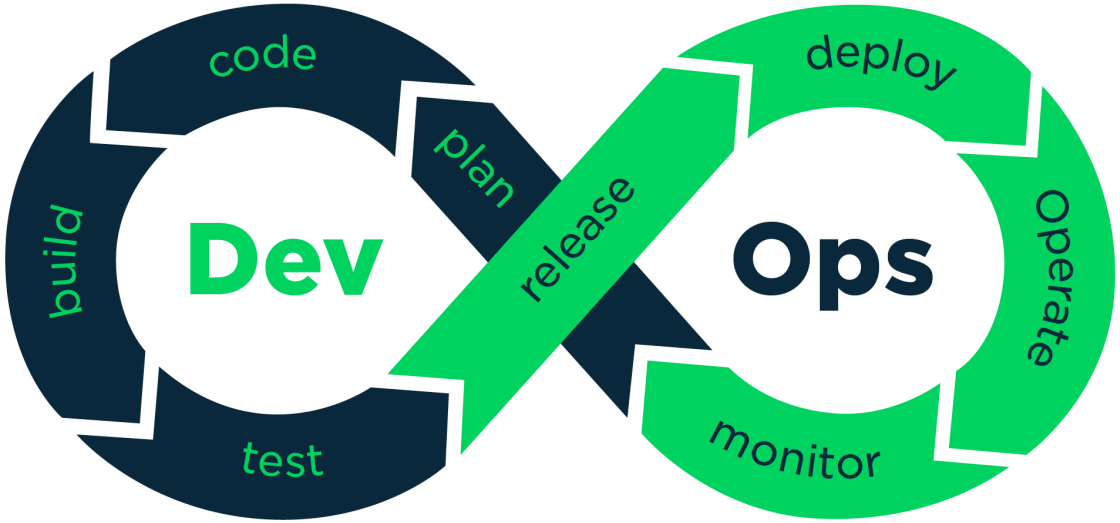
参考: State of Devops 2024の回答者分布



DevOpsにおけるテストを改めて捉える

DevOpsの一体化 = Devの間からOpsを考える

⇒「開発してからの面倒事」を織り込んで開発を行うということ



▌DevOpsはQAなどの後工程をフロントローディングするための
プラクティス（とも言える）

「テストを加速させるDevOps」と捉え直そう

DevOpsのプラクティスは品質を高め続けるフィードバックを回し続けること

- リリースの高度化・自動化
- 可観測性の強化
- 自動テストの強化
- セキュリティの早期確認

このあたりを含めると、 DevSecOps や DevQaOps という言葉を足した方が通りはいいかも？

ともかく、開発スピードとは別軸の語り口 が必要になる

実際にダイキンでやってきたこと

もともとDevOps的な取り組みは「ベストプラクティス」の共有だった

AWS設計指針（ベストプラクティス）

社内のセキュリティルールなども参照しながら、AWSを使う上での「ベストプラクティス」として、様々な設計指針を記述していった

AWSリファレンスアーキテクチャ

設計指針を具体的な設計に落とし込んだものとして、社内でよく用いられるユースケースを想定し、ドキュメントとIaCを提供するもの

DevOpsのスタートの紹介: AWS設計指針

目次

DevOpsのスタートの紹介: AWS設計指針

要約

AWSにおけるネットワーク設計について、VPCを中心に説明する。

本章の対象読者

- EC2やRDSといったVPCリソースを利用する
- VPCをまたがったシステム間で連携する
- AWSアカウント外の外部システムと連携する

・本セクションの対象となるユースケースや目的
・読むべき章かの判定に利用する

内容の要約

・セクションの内容を要約
・概要を理解する

得られる知識

- VPC、サブネットの設計指針
- VPC間、VPC外の接続方法

・このセクションを読んだあとに得られる知識

基本指針

- VPC設計
  - [推奨] システム・アプリケーションなどの目的別にVPCを分割する
  - [推奨] 基本的にはパブリックサブネット/プライベートサブネット構成を取る
    - [必須] 本番環境などの高可用性が求められる環境ではサブネットをマルチAZ構成にする
    - [推奨] ただし、NATゲートウェイの利用コストに注意し、プライベートサブネットの要否については深く検討する

・セクション内で示している設計指針
・[必須] は必ず守るべき項目
・[推奨] は可能な限り取り入れるべき項目

関連するAWS Workshop

- VPC ENDPOINT WORKSHOP
  - VPC内リソースからVPC Endpointを経由してS3やSQSにアクセスする手順を学ぶ
- NETWORKING IMMERSION DAY
  - AWSにおけるネットワーキング全般について学ぶことができる
  - 本章の内容は、LAB 1: MULTI-VPC ACCOUNT ARCHITECTUREやLAB 4: SECURITY CONTROLSの内容が該当する

・セクションに関連するAWSのWorkshopのリンクを紹介
・AWSリソースを動かしながらより深く理解ができる

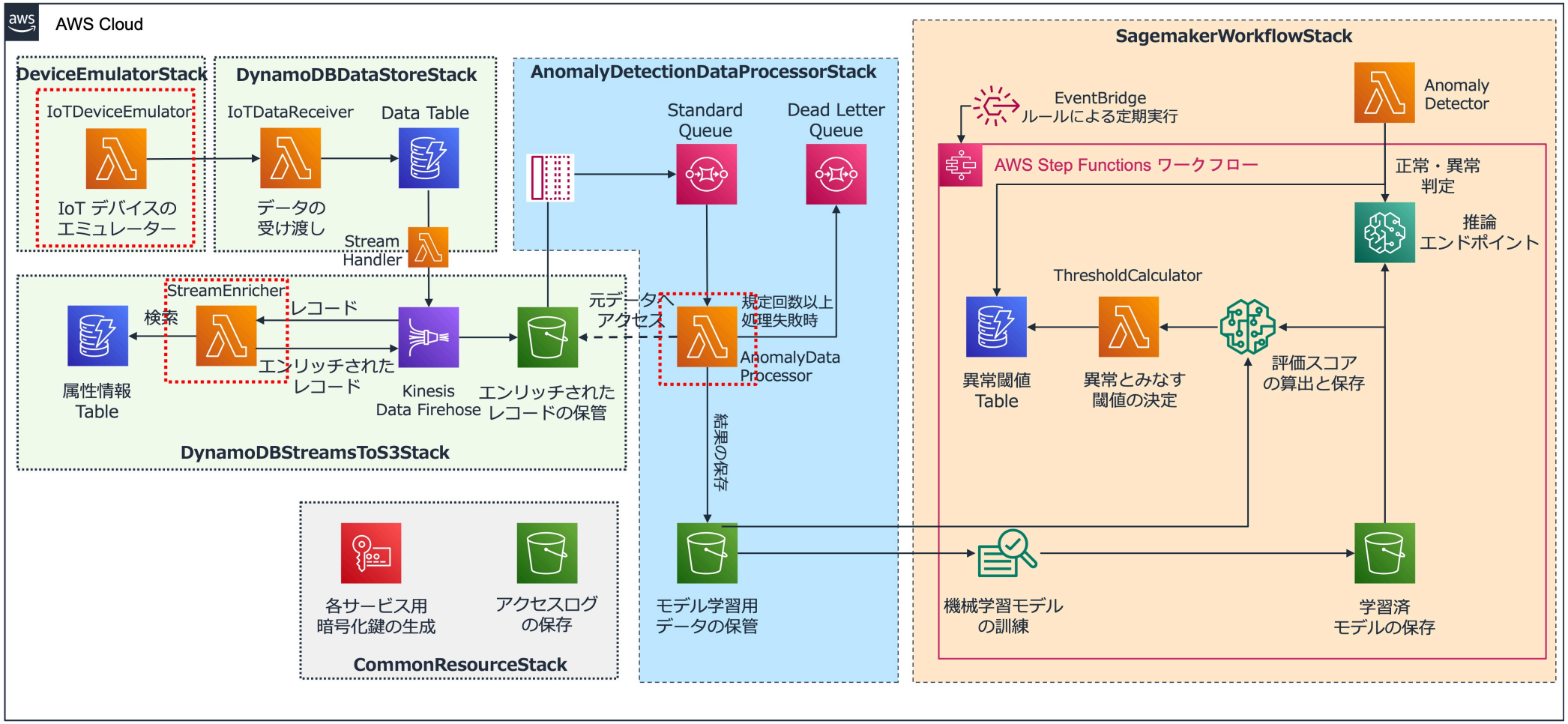
- 対象読者
- 内容の要約
- 各セクションの基本指針
- 関連するAWS Workshop

などを記載。

DevOpsのスタートの紹介: リファレンスアーキテクチャ

テンプレート名	テンプレートの概要	テンプレートのレベル	ビルド/テストステータス	仕様書
Web3層アプリテンプレート	一般的なWeb3層アプリのインフラを構築します	status stable	On pull request web-app passing	無
Google API認証モジュール	GoogleDriveなどのGoogle APIをコールする際に利用するTokenをキャッシュするモジュールを構築します	status stable	On pull request google-api passing	無
DynamoDBデータ処理パイプライン	DynamoDBをソースとしてデータ加工を実行するデータパイプラインを構築します	status experimental	On pull request data-processor/ddb-stream-processor passing	無
コスト異常検出	AWSコスト異常検出機能をデプロイします	status experimental ※	On pull request cost-detect passing	無
SES 検証済ドメイン発行	SESの検証済ドメインを作成します	status experimental	On pull request ses failing	無

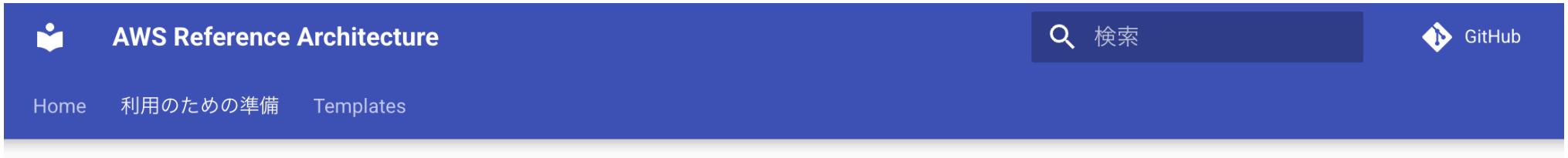
リファレンスアーキテクチャー例



凡例： ユースケース①のスタックを改変　ユースケース②を元にスタックを再作成　ユースケース③で新規実装　共通リソース

改変・再作成の際に元の実装から変更を加えた要素

利用のためのドキュメンテーション整備



利用のための準備

本リポジトリを利用するための事前準備

まず、このリポジトリをCloneしてコードを取得してください。

```
git clone https://github.com/daikin-tic/aws-reference-architecture.git
```

さらに、本リポジトリで提供されるテンプレートを実行するために、以下をインストールしてください。

- AWS CLI

  `AWS CLI`のインストールと合わせて、プロファイルの設定も行ってください。

- Node.js

  各テンプレートは、AWS CDKを利用します。AWS CDKのインストールのためにインストールします。

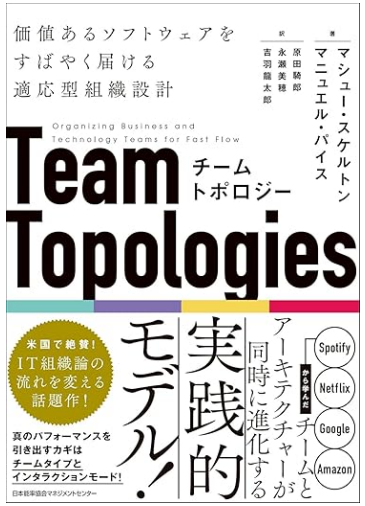
単なる「良いツール」では広がらない

あくまで現状の改善ツールとして広げていこうとしたが、なかなか広がらない

あると便利なのはわかるけど、自分たちには必要ないかな

無反応（そもそも自分たちの設計に課題があると思っていない）

▌仕組みはただ作るだけではなく、リアルな利用シーンを想定し、イネイブリングしていく必要がある



「品質」に焦点を当てた取り組み

実際の開発に入り込み、品質課題をDevOps的視点で指摘

⇒ AWSアーキテクチャのイネイブリング

AWSのセキュリティチェックをプロセスに取り組む

⇒ AWSセキュリティダッシュボード

生成AIアプリ乱立にQAを意識した仕組みを構築

⇒ RAGテンプレート

AWSアーキテクチャのイネイブリング

リファレンスアーキテクチャの適用部門にイネイブリングとして積極的に参画

⇒ リファレンスを起点にDevOpsを活用した品質向上を実施した

実施した取り組み例

- デプロイメントのプロセス構築
- 社内規定遵守のためのログ取得の仕組み構築
- IaC、静的解析の導入などの下回りサポート

「本当に必要な人には届いていない」を実感

- 「自マシン上で動かしていたものがサーバに乗ればいいです」

⇒そもそもCLIプログラムで誰かが裏でずっと見ておく必要あり

⇒「顧客環境で動かす」ために求められるありとあらゆる品質が考慮できていない

- 基盤との通信認証に使う秘密鍵はローカルに置いてそのPCはPoC現場に置いときます

⇒ その鍵が漏れたらIoT基盤全体にDDoSかけられるんですけど

AWSセキュリティダッシュボード

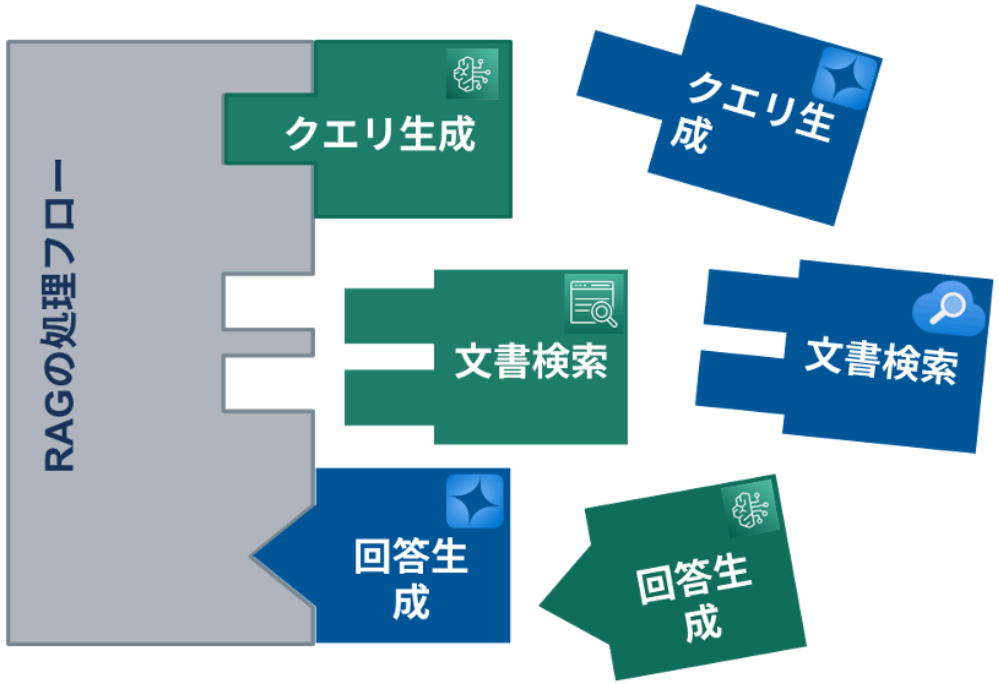
AWSのセキュリティチェックを社内ルールを参照して行い、その結果をダッシュボードにまとめる

RAGアプリケーションテンプレート

生成AI（特にLLM）の猛烈な広がりで、社内RAGの需要が急増

⇒ 低品質のアプリ乱立を防ぐため、テンプレート化を提案

もちろんコード自体のメンテナンス性や拡張性は考慮したが...



✅ 各フローで用いるサービスを簡単に**差し替え可能**

✅ 用意されていない処理を実装するときにも**インターフェース仕様に従うことで簡単に実装可**

DevOpsのエッセンスをレポジトリ自体に入れ込む

- Github Actions Readyな構成
- コストがかさむコンポーネントを無効化できる仕組み
- コードの単体テスト、IaCのスナップショットテスト導入
- もちろん標準的な静的解析もReadyに

広く利用されうる仕組みだからこそ、DevOps Raadyな状況を積極的に組み込み、品質を担保

知らず知らずのうちに、DevOpsの仕組みを利用し、享受している状態に。
⇒ 手作業だと４時間構築に必要だったが、Actions利用で１時間以内の構築を可能に

DevOpsを組織内で広げる補助線としての「テスト」

テストや品質といったキーワード、取り組みは企業の中でDevOps的な取り組みを取り入れる良い取っ掛かりになる

「テストをうまく実施するため」「品質向上のため」にDevOpsを活用するのは、組織にDevOpsの文化を根付かせる一つのアプローチ

- AWSの指針・リファレンス ⇒ 安全に構築するためのイネイブリング
- セキュリティを守るための仕組み
- 生成AIアプリを1番楽で安全に作る方法にDevOpsパイプラインを組み込む

DevOpsで加速するテスト

「DevOps抜きのテスト」をちょっと想像してみてください

- 自動テストの仕組みがない根性の手動テスト
- IaC化されないテスト環境（「おま環」問題）
- Gitがない
- Gitはあるけどブランチ管理・タグ管理指針がない

本質的に「DevOpsはテストを加速させる」もの

▌⇒ それをさらに加速させうるのが「AI」

DevOps + AI で加速する品質

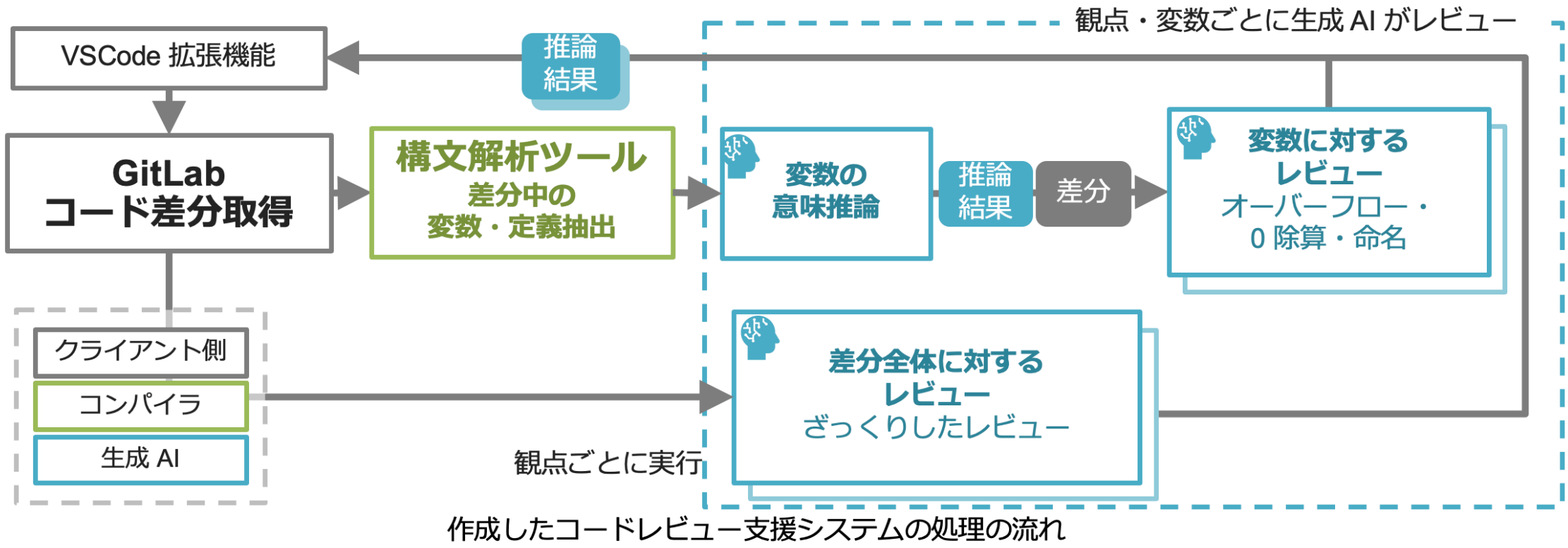
AIは万能にみえるが、万能ではない

① 駆動点はまだまだ人。探索的なものはそれでも良いが、「自動で手間なく動く」仕組みがあれば、AIをより活用できる

⇒特にテスト・QA領域への取り込みは効果的
現在、AIを活用したコードレビューを検証中

② AIはチューニングに終わりはない。AIの品質を高め続けるためにも、改善サイクルをDevOpsとして組み込む

AIによる自動レビューの仕組み

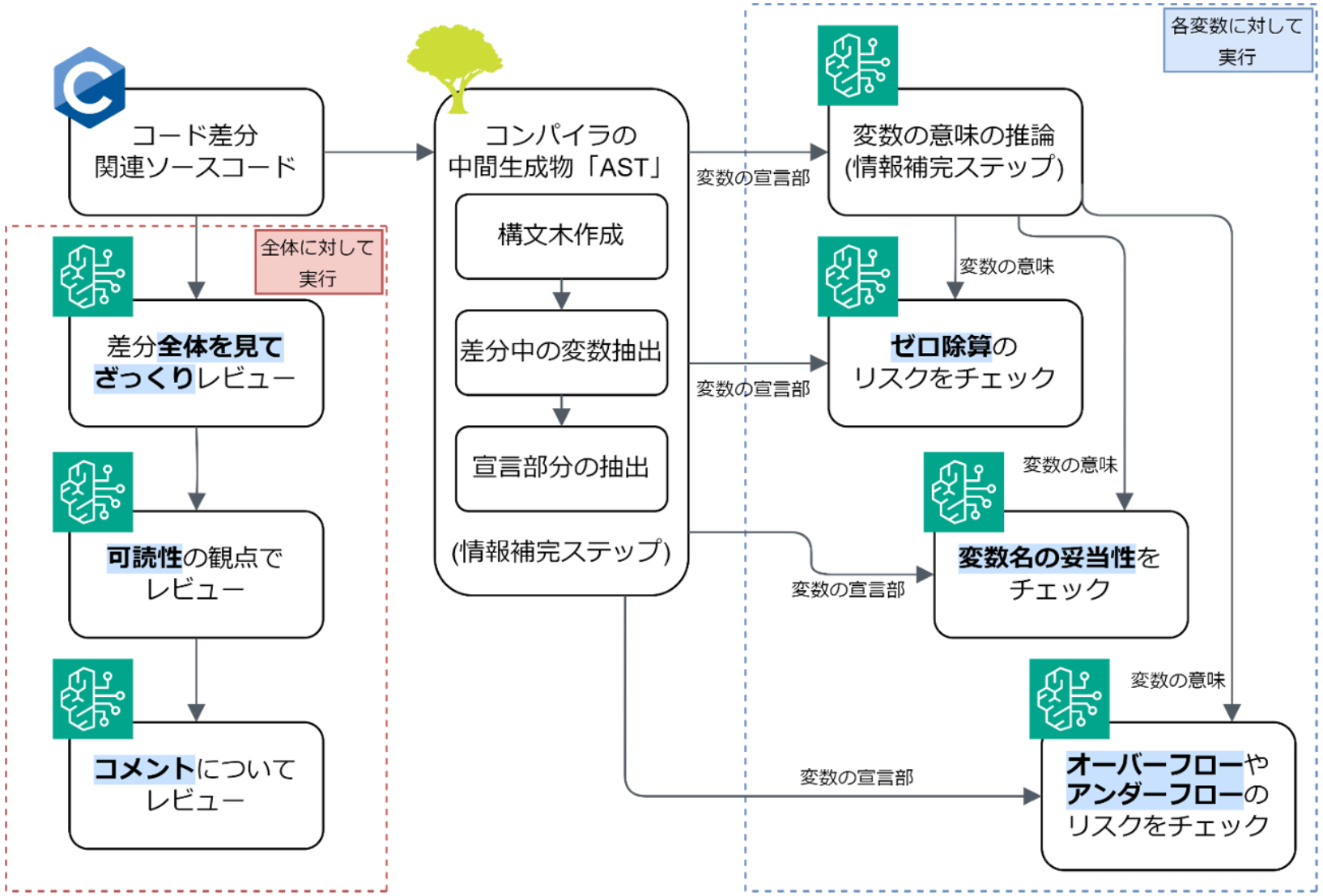


作成したコードレビュー支援システムの処理の流れ

GitLabのパイプラインを起点にコードレビューを自動で行う仕組みを構築

Daikin AI Reviewer on your Code: **D-Arc**

D-Arcの中の仕組み: Agenticなレビュー



AI時代のテストを加速させる必須な相棒、DevOps

現在は人手で書いたコードを中心にしているが、近い将来AIが生産したコードを評価する必要性が必ずでてくる。

その時に人がボトルネックにならないためにも、DevOpsパイプラインにテストを組み込むことが必須。

非決定論的なAIを、同じく非決定論的なAIでレビューするのは不安が残る

AIのコードを評価するためには、決定論的なしくみを取り込んだパイプラインを作る必要がある

- 既存の静的解析ツールとの組み合わせ
- MCPなどを使った周辺環境とのコミュニケーション

AI品質を上げるテストも、DevOpsで加速する

AIの出力を評価し、フィードバックを得られる仕組みは今後の展開のためにも必須

継続的にAIの出力を評価し続ける「シフトライト」

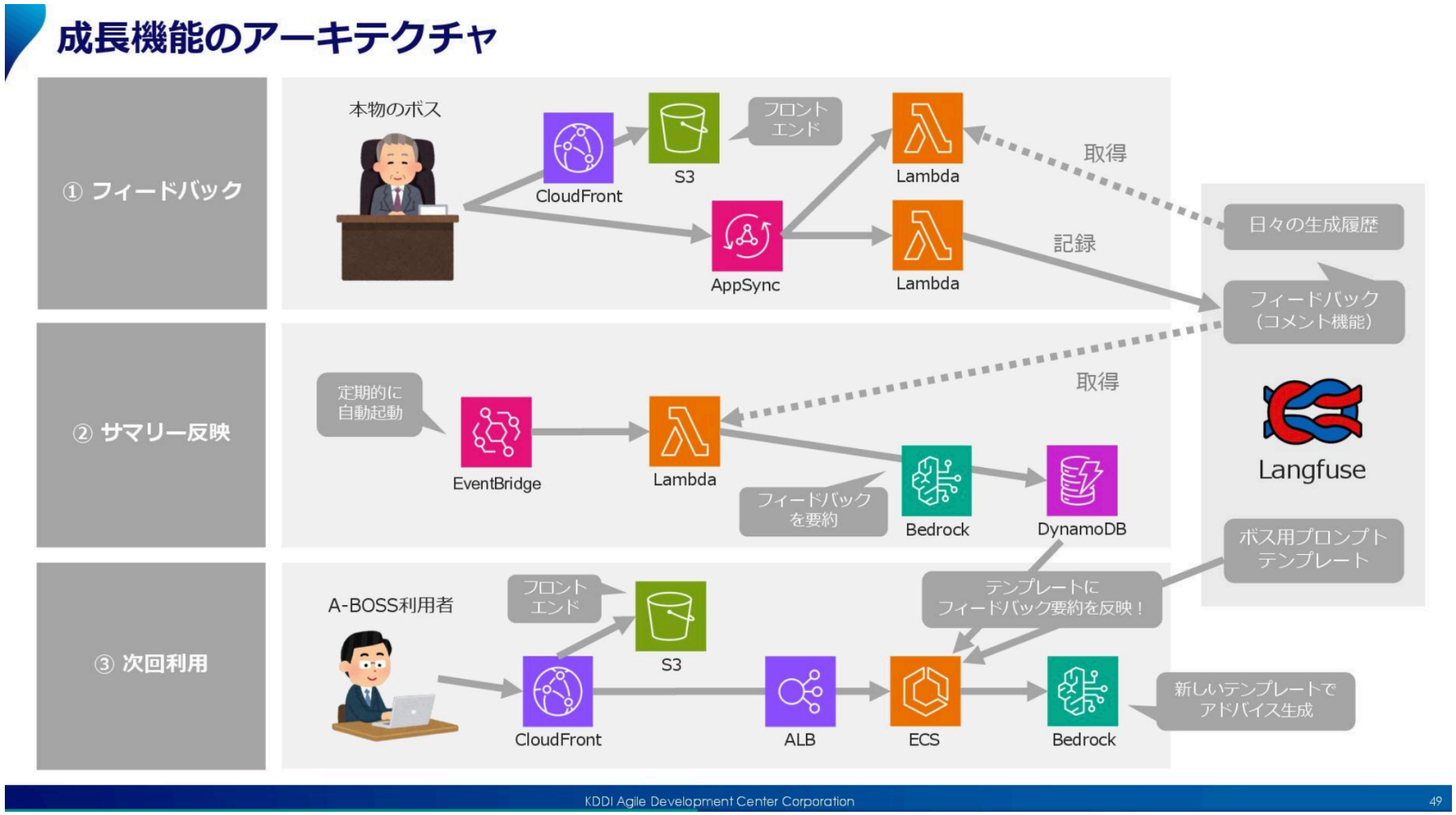
オブザーバビリティ + 自動テスト + 手動評価

AIに対するフィードバック機構の構築

- 継続的事前学習
- ファインチューン/蒸留
- メモリ

一例としてはこんな感じ

KDDIの事例



参考: https://speakerdeck.com/minorun365/ben-bu-chang-nodai-wariniti-an-shu-rebiyu-kddiying-ye-gamei-ri-shi-uaieziento-a-boss-kai-fa-mi-hua?slide=50

おわりに

AI時代を乗り切るために

「高速なフィードバック」を半ばあきらめてしまったけど、変化が激しいAIに対応していくには本当は必要

だからこそ、どんなことをしてでもDevOpsのような高速検証サイクルをReadyにしておかないといけない

大変だけど、まだ先行者利益はある。無理矢理にでもプロセスに取り込む 覚悟が必要

▌DevOps x テスト x AI で、その覚悟に見合う対価を受け取りにいきたい

なので一緒に頑張りましょう

とはいえ大きな組織で大きな変化を実行するのは大変

ただ、技術が急激に入れ替わっている今はその変化を起こす絶好の機会

今まで打ち壊せなかったしがらみを、技術でぶち壊して、もっと面白い未来を一緒に作っていきましょう！！

というわけで、せっかくなので、、、

エンタープライズ内製組織コミュニティ（仮）を準備中！

これまで、アジャイル開発やDevOpsをキーワードに、数社と交流会実施

⇒ 取り組みや課題感を共有するのは大きな学びになると手応え

大企業の中で内製化を試行錯誤しながら
進めている仲間たちともっと繋がりたい！！

コミュニティの立ち上げ準備中！

現在参加希望者や活動ニーズを探るアンケートを実施中。興味ある方は是非ご回答ください！