
aws
SUMMIT
ONLINE
JAPAN | MAY 11-12, 2021

AWS-02

Amazon CodeGuru

機械学習で実現するコードレビュー自動化と アプリケーションパフォーマンス最適化

金杉 有見子

ソリューションアーキテクト
アマゾン ウェブ サービス ジャパン株式会社

自己紹介

Yumiko Kanasugi (金杉有見子)

- 所属

  アマゾン ウェブ サービス ジャパン株式会社

  技術統括本部

  ソリューションアーキテクト

- 好きなAWSサービス

Amazon CodeGuru

AWS Support

- リモートワークの過ごし方

  毎日夕方愛犬と散歩 🐕

Key Takeaways

対象者	アプリケーションデベロッパー (Java, Python)
Key Takeaways	• ソフトウェアデリバリープロセスでは俊敏性と品質の両立が求められている • Amazon CodeGuru Reviewer はコードレビューを自動化し、コード品質の向上に役立つ • Amazon CodeGuru Profiler は最も実行コストが高いコード行を特定し、インフラコストの削減およびパフォーマンス最適化を手助けする

Agenda

- ソフトウェアデリバリーにおける課題
- Amazon CodeGuru 利用の全体像
- Amazon CodeGuru Reviewer
- Amazon CodeGuru Profiler
- まとめ

ソフトウェアデリバリーにおける課題

Amazon CodeGuru

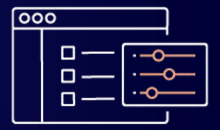
コードの問題を特定するの
に労力と時間がかかる

パフォーマンスエンジニア
リングの深い知見が必要

ベストプラクティスを
学習する必要がある

継続的に改善を行う
必要がある

実用的な推奨事項が
欲しい



修正の優先順位
を付けたい

俊敏性 × 品質

ソフトウェア開発における俊敏性と品質の担保

開発ワークフローを加速

限られたリソースを有効活用し
ボトルネックを最小化

発見が困難な課題点を早期に発見

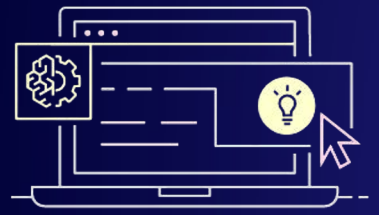
アプリケーションパフォーマンス
を継続的に改善

運用コストを最小化

Amazon CodeGuru 利用の全体像

Amazon CodeGuru

コードに欠陥がある部分や**アプリケーションで最も実行コストが高い箇所**を特定し、
改善方法含め推奨事項を生成する機械学習をベースとした開発者向けのサービス



Amazon CodeGuru Reviewer

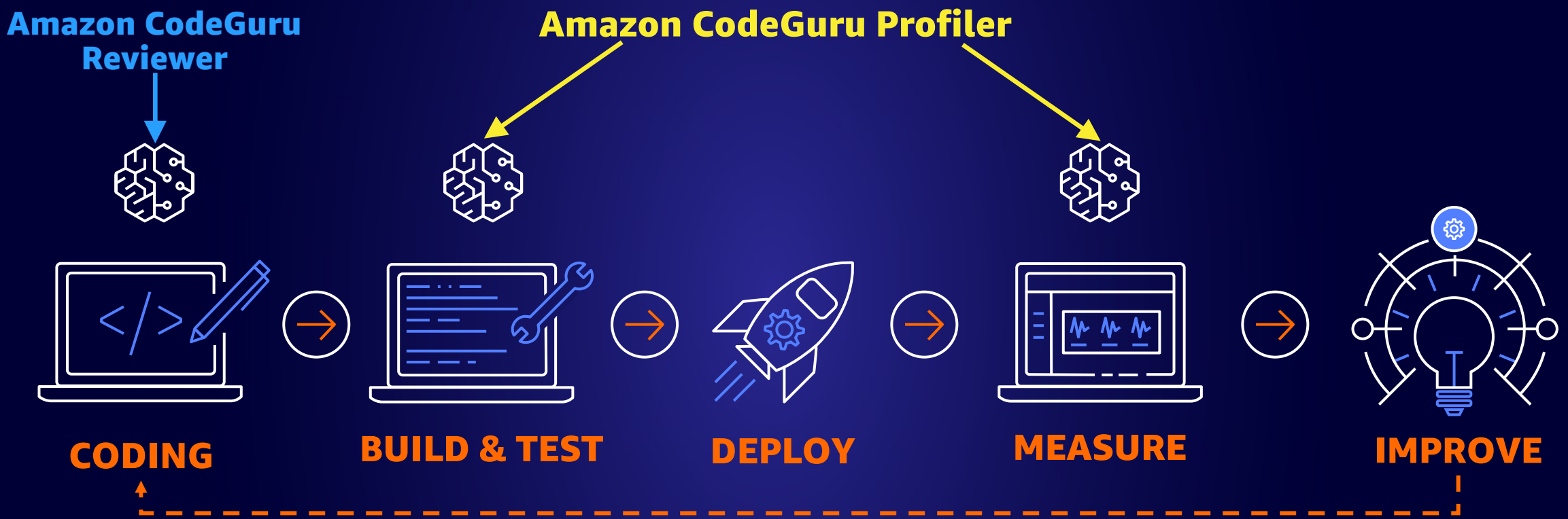
- ソースコードのクリティカルな問題や発見が困難なバグを特定
- 改善方法、ドキュメントを合わせて提示
- Java、Python (発表時点でプレビュー) に対応

Amazon CodeGuru Profiler

- アプリケーションのパフォーマンス状況を可視化し、最も実行コストが高いコード行を特定
- 改善方法、ドキュメントを合わせて提示
- Java/JVM ベースの言語、Python (発表時点でプレビュー) に対応

２つの機能は独立して利用可能

Amazon CodeGuru

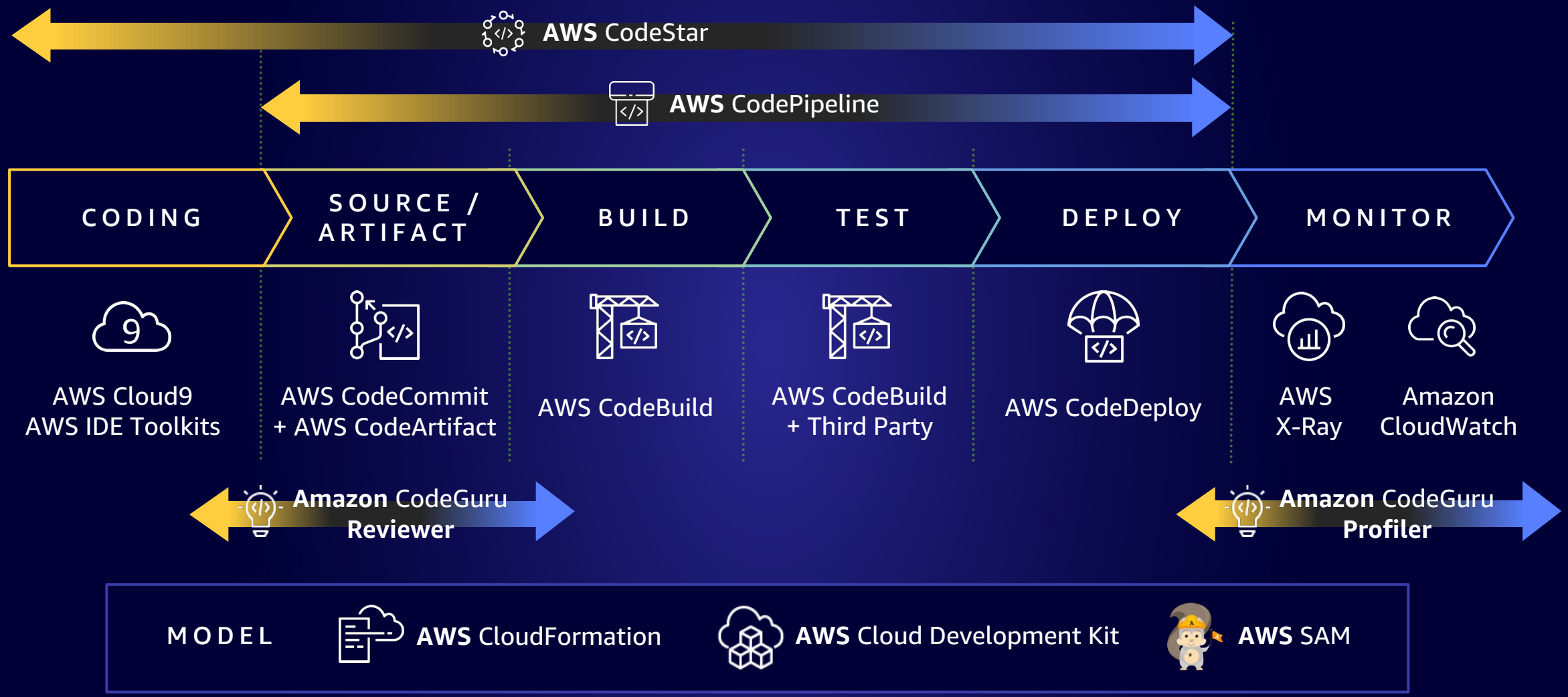


実用的な推奨事項を生成するビルトインのコードレビュー

もっとも実行コストが高いコード行の検出および最適化

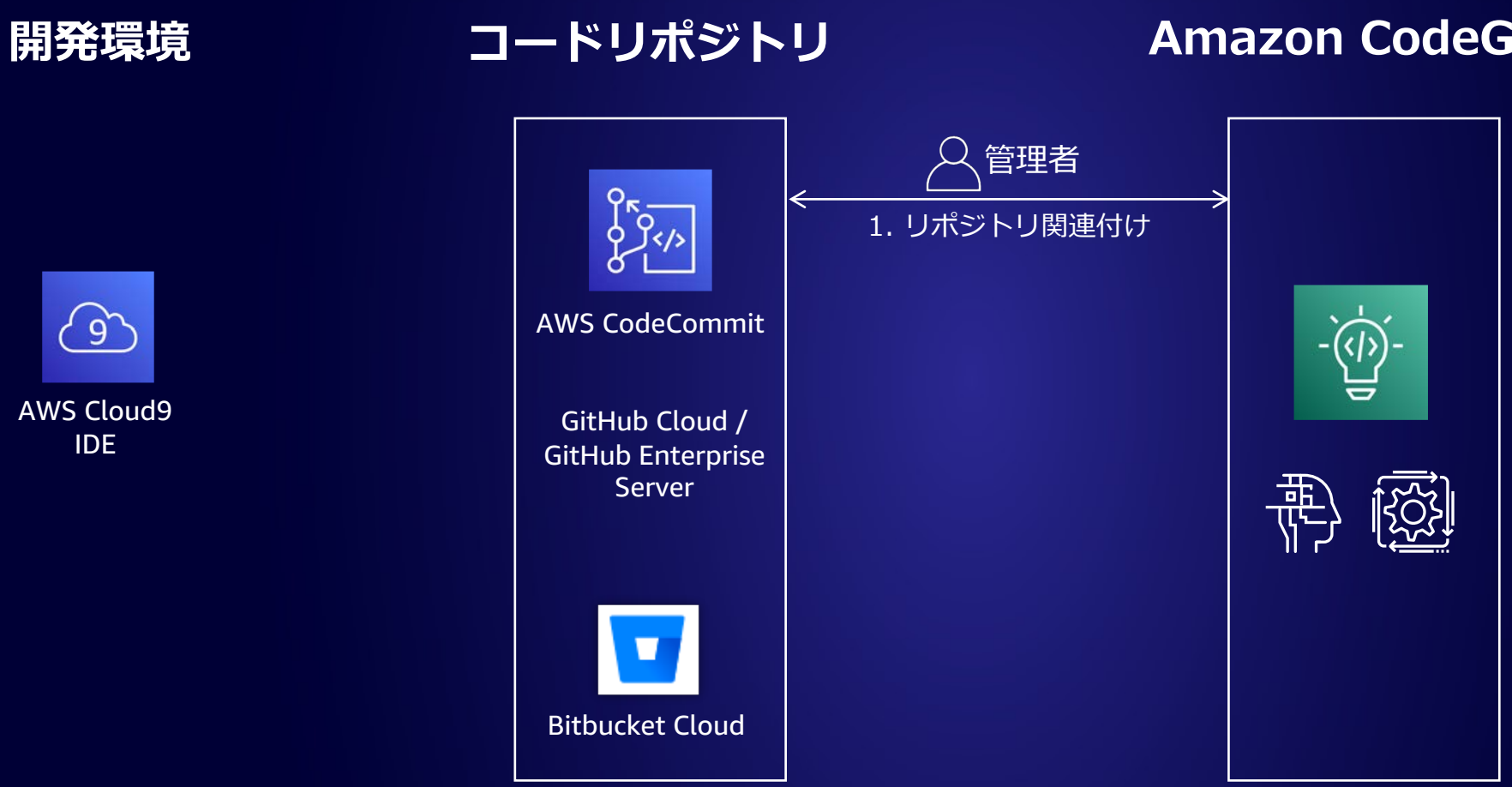
本番環境でパフォーマンスとコストにおける改善点を容易に特定

ソフトウェアデリバリーパイプラインにおける立ち位置



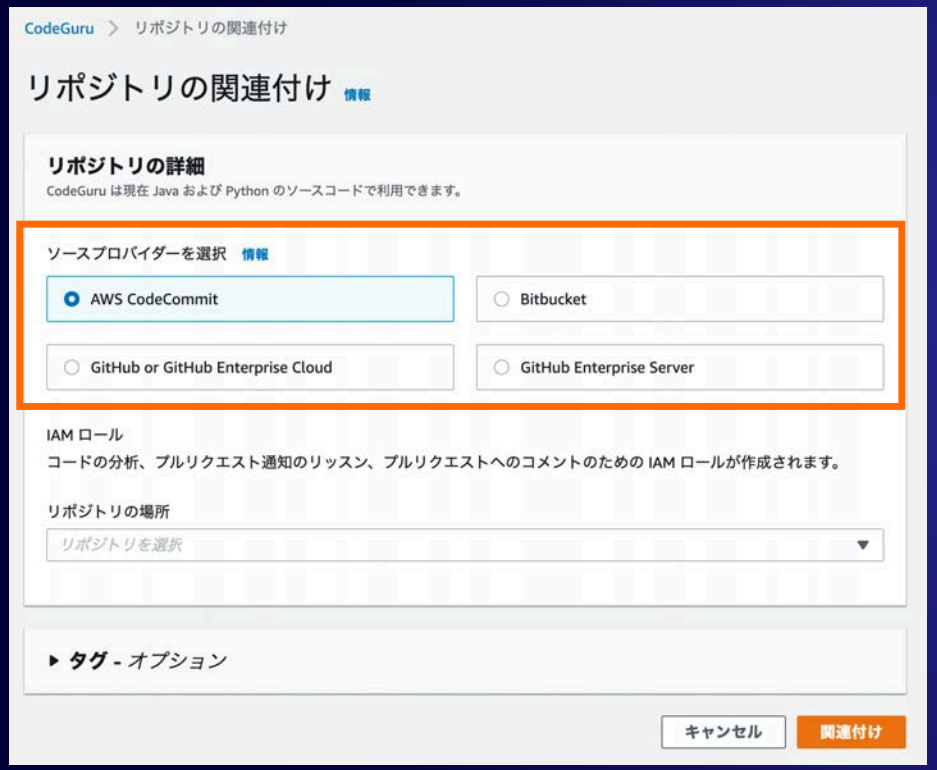
Amazon CodeGuru Reviewer

CodeGuru Reviewer の動作イメージ



CodeGuru Reviewer リポジトリの関連付け

CodeGuru > Reviewer > リポジトリ

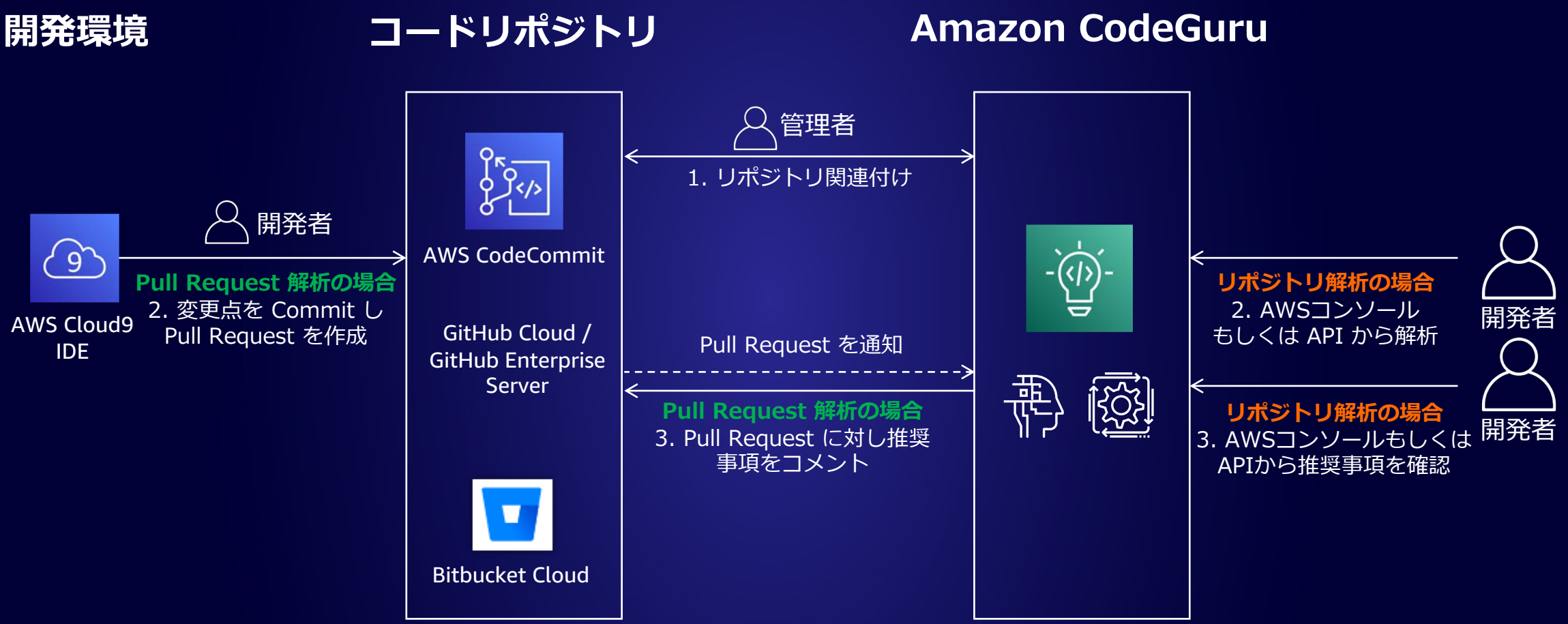


ソースプロバイダ及びリポジトリを選択

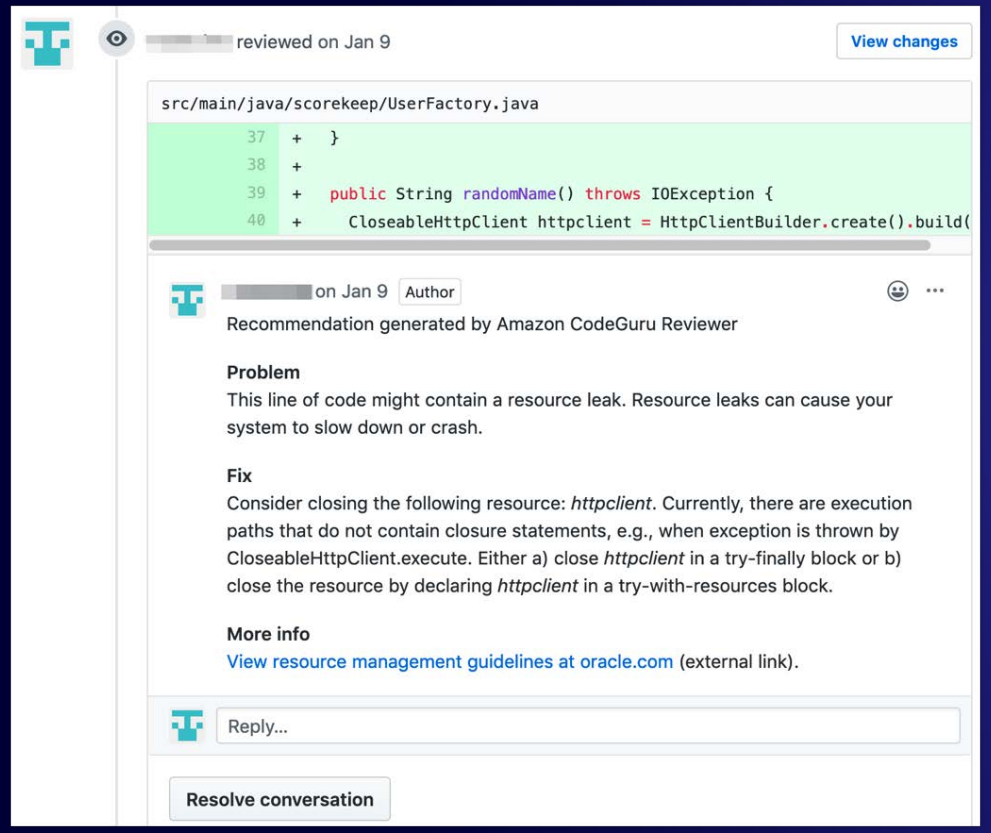
CodeCommit > リポジトリを作成

CodeCommit リポジトリ作成時にも有効化可能

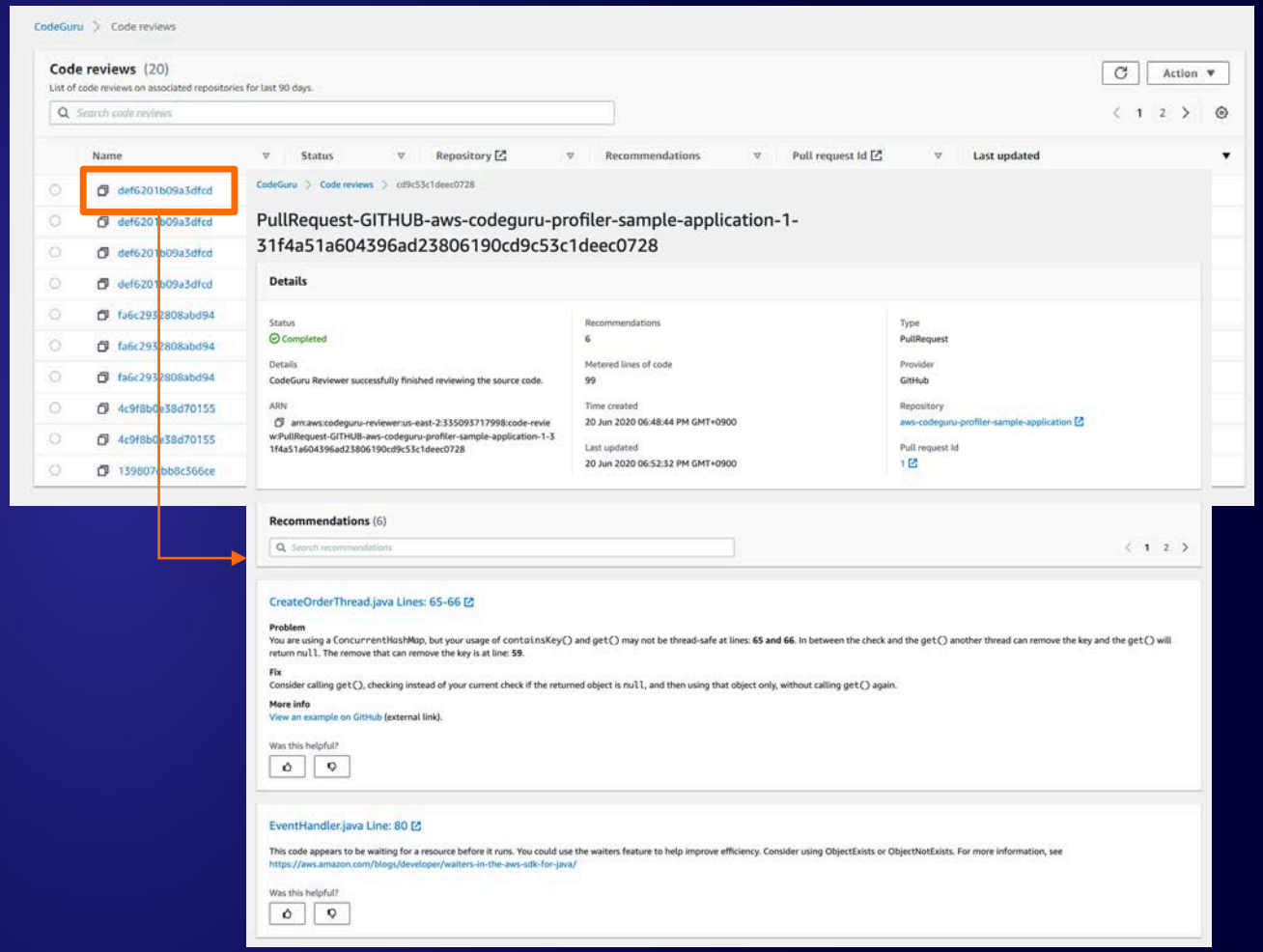
CodeGuru Reviewer の動作イメージ



CodeGuru Reviewer 推奨事項の確認

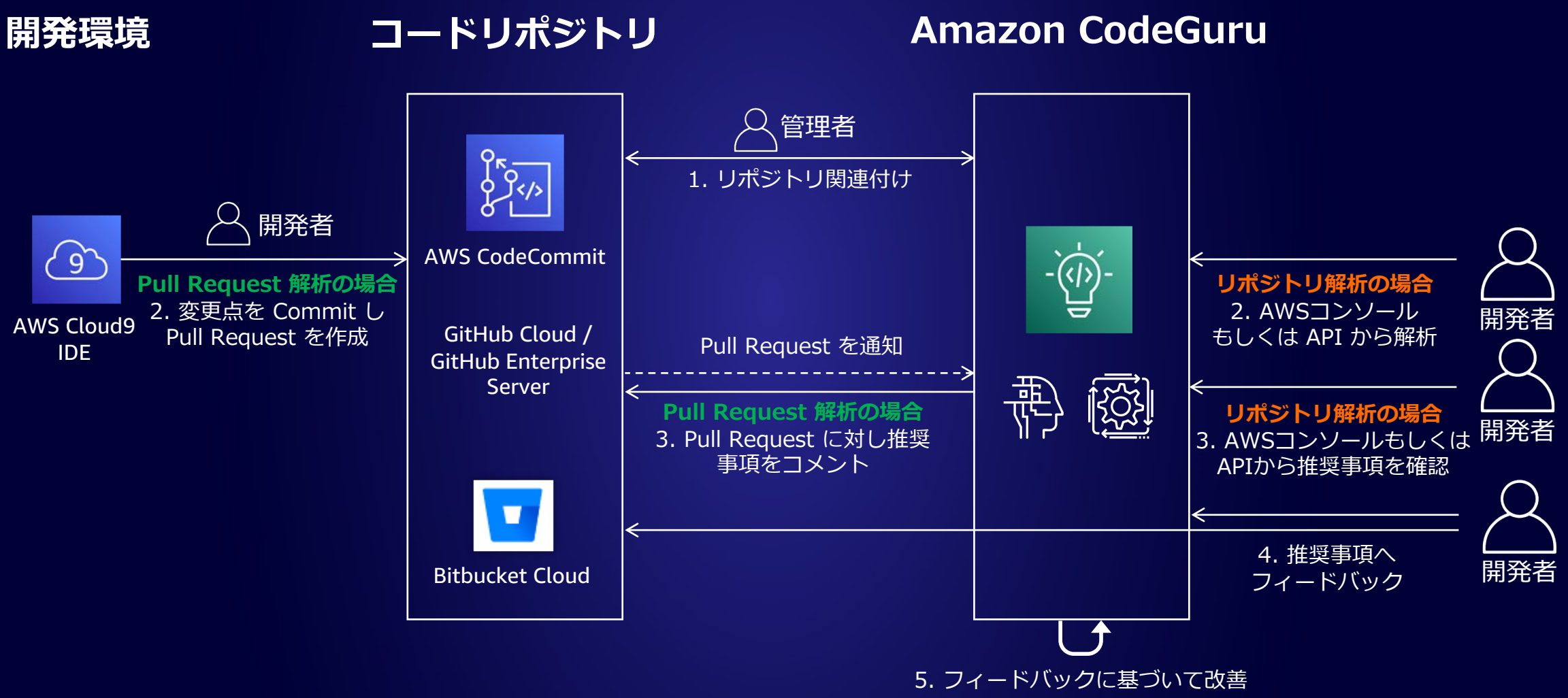


Pull Request へのコメント

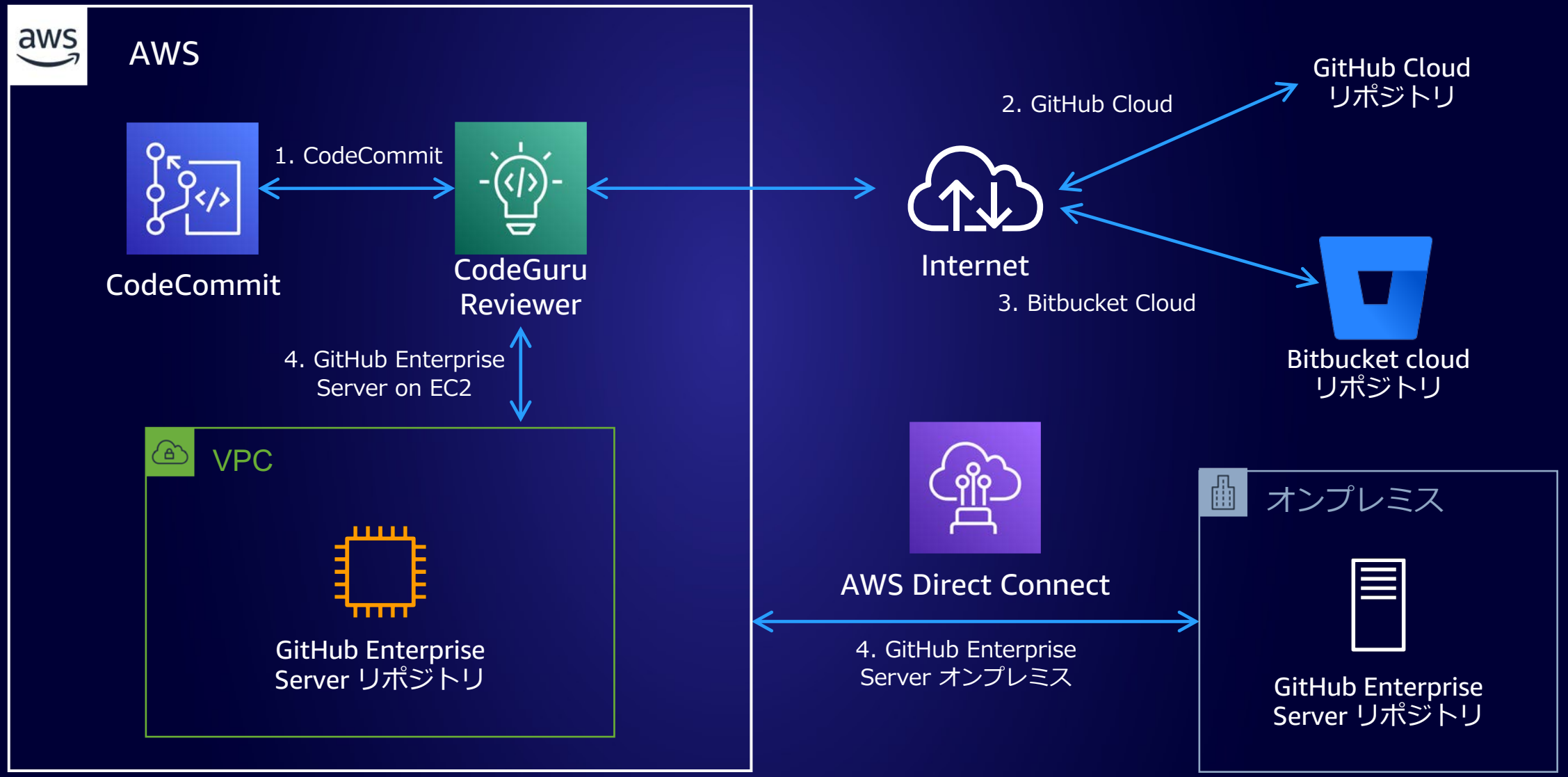


CodeGuru Reviewer > Code Review の詳細画面

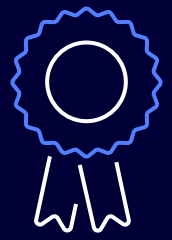
CodeGuru Reviewer の動作イメージ



CodeGuru Reviewer と各種リポジトリの連携

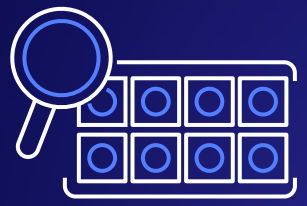


CodeGuru Reviewer 推奨事項



AWS ベストプラクティス

正しい AWS API の使い方



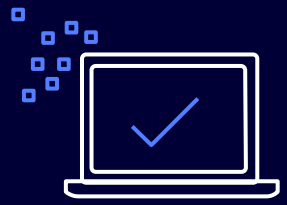
並列処理

マルチスレッド処理の適切な実装

New!

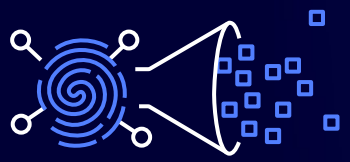
セキュリティ

セキュリティ
ベストプラクティス



リソースリーク防止

正しいリソースの扱い



機密データの漏洩

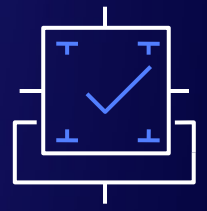
機密情報の不必要な後悔を防止

コーディング
ベストプラクティス

コード欠陥の発見

リファクタリング

冗長なコードの特定



インプット
バリデーション

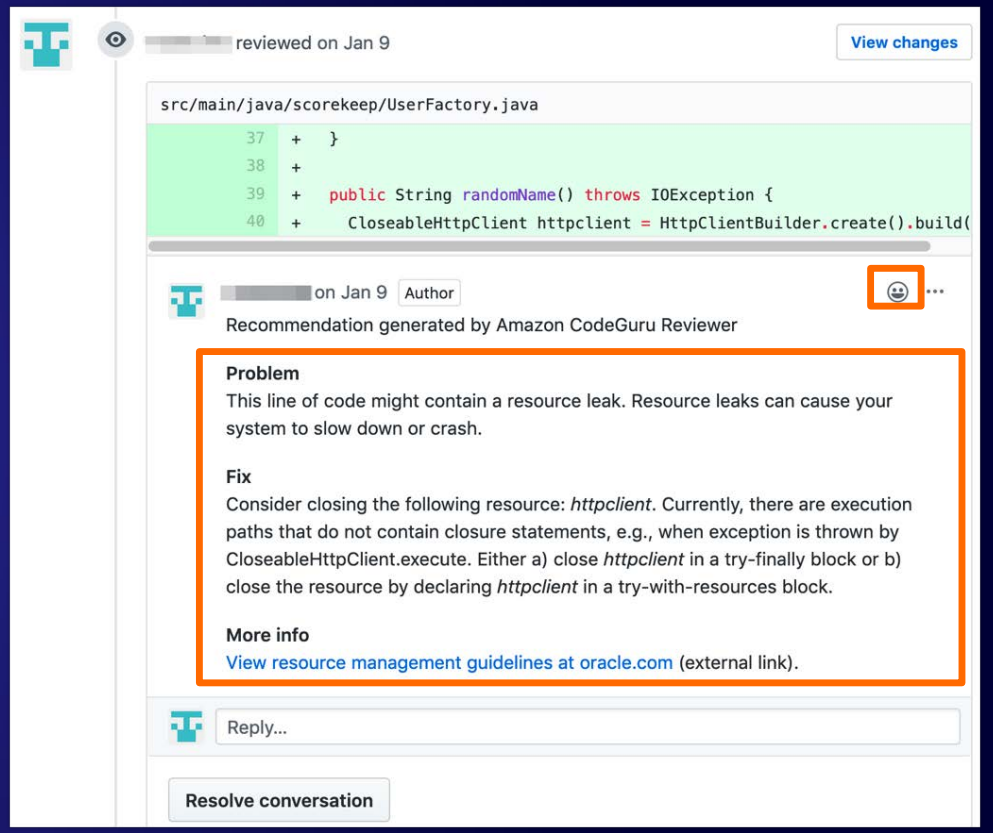
入力形式の確認

コードクオリティ

一般的な品質向上

CodeGuru Reviewer 推奨事項

- 各推奨事項には Problem, Fix, More info などといった改善方法や関連するドキュメントへのリンクも含まれる
- コメントあるいはEmojiリアクションを通して推奨事項へフィードバックすることで CodeGuru Reviewer の精度向上に繋がる
- コードレビューや推奨事項の一覧はマネジメントコンソールあるいはAPIで取得可能



例: リソースリークの可能性があるコードを指摘

CodeGuru Security Detector

re:Invent 2020 で発表された新たな推奨事項の検出カテゴリ

AWS API セキュリティ ベストプラクティス	**Java 暗号化ライブラリ** ベストプラクティス
セキュアウェブ アプリケーション	**AWS セキュリティ** ベストプラクティス

Javaのみサポート
AWSコンソールからソースコードとビルドアーティファクトをアップロードする必要がある

Java Crypto Library Best Practices

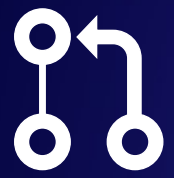
Code

```
import javax.crypto.Cipher;

static final String CIPHER = "DES";

public void run() {
  cipher = Cipher.getInstance(CIPHER);
}
```

Recommendation

It looks like your code uses a cipher object with an insecure transformation. To make your code more secure, use one of the following algorithms with a built-in integrity check: AES/GCM/NoPadding, or for Java 11 or newer, ChaCha20-Poly1305. Learn more about javax.crypto.Cipher.getInstance.

CodeGuru Reviewer 3種類のワークフロー



Pull Request ワークフロー

リポジトリ解析ワークフロー

セキュリティ解析ワークフロー

レビュー内容	コードレビュー	コードレビュー	コードレビュー及びセキュリティ解析
レビュー方法	Pull Request	リポジトリ解析 (コンソールもしくは API で開始)	リポジトリ解析 (コンソールで開始)
対象リポジトリ	CodeCommit, GitHub, GitHub Enterprise, BitBucket	CodeCommit, GitHub, GitHub Enterprise, BitBucket	S3 にソースコードとビルドアーティファクトをアップロード
対応言語	Java, Python (プレビュー)	Java, Python (プレビュー)	Java
推奨事項	Pull Request 画面、Code Reviews コンソールまたはAPI	Code Reviewsコンソール、API	Code Reviewsコンソール、API
ユースケース	日々の開発で継続的にコードレビューを行いたい	CodeGuru Reviewer の評価や定期的にコードクオリティを確認したい	コードセキュリティに関するレビューを行いたい

Amazon CodeGuru Profiler

CodeGuru Profiler の動作イメージ

ランタイムのプロファイリングデータを継続的に収集し、パフォーマンス改善のためのインサイトを提供する

JVM ベースのアプリケーション

アプリケーション
Profiler スレッド

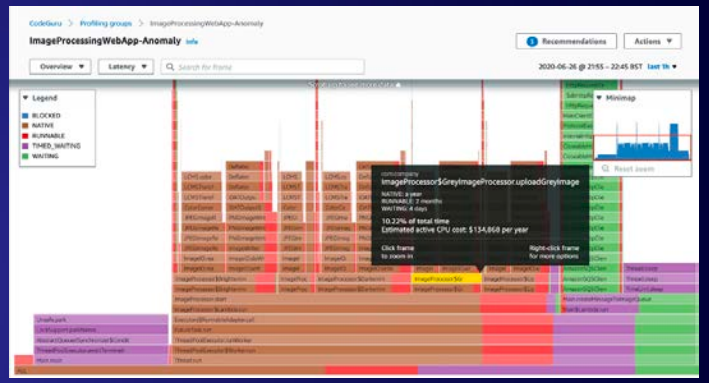
アプリケーション
Profiler スレッド

Javaランタイム Lambda関数

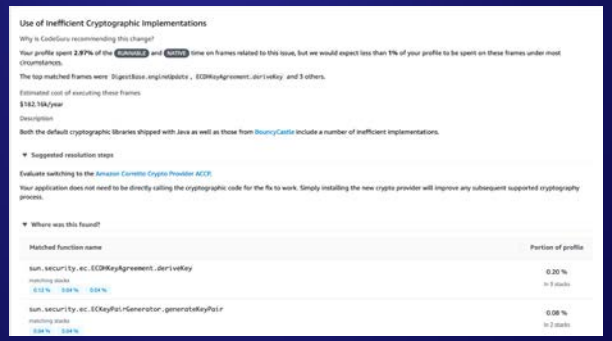
Lambda関数
Profiler スレッド

Lambda関数
Profiler スレッド

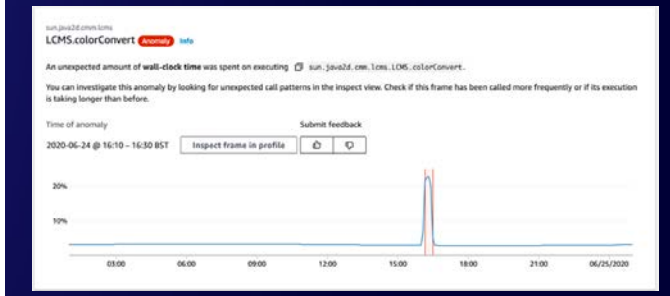
Amazon CodeGuru Profiler



プロファイルデータのビジュアライゼーション



推奨事項



異常検知

CodeGuru Profiler の始め方

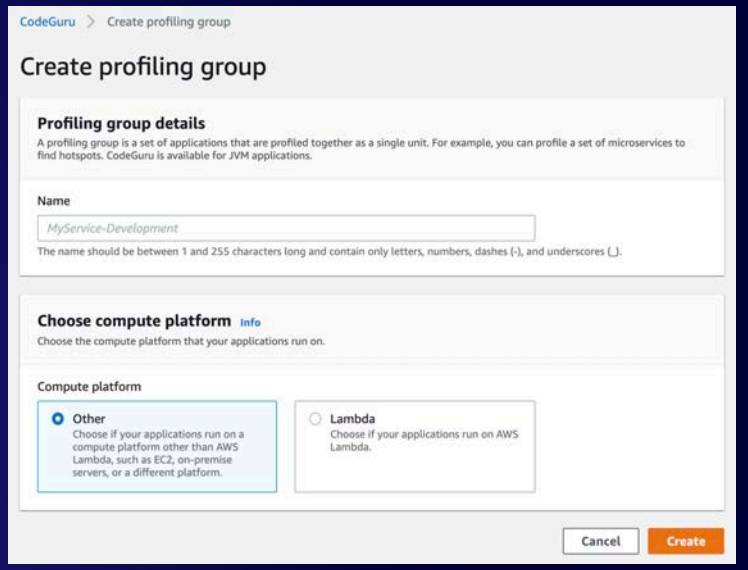
1. CodeGuru Profiler プロファイリンググループの作成

   CreateProfilingGroup API あるいは マネジメントコンソール > Amazon CodeGuru > Profiler より作成

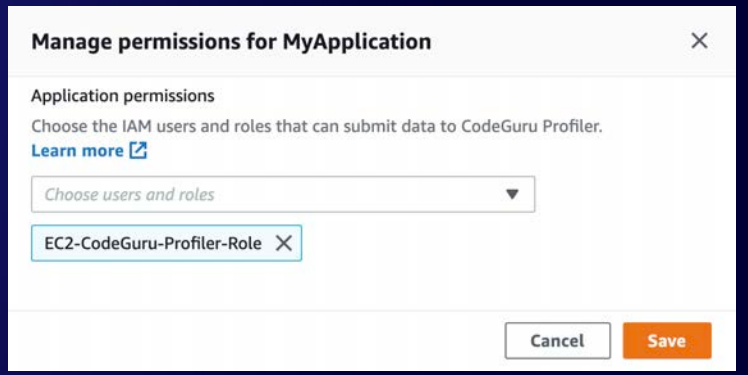
2. IAM権限の設定

   Profilerエージェントが使用する IAM User / Role にプロファイルデータを送信するための権限を付与

3. Profiler エージェントをスタート
   - エージェントは本番環境でアプリケーションを継続的にプロファイルするよう設計されている
   - エージェント起動後 5 - 15分 でアプリケーションデータが送信される。以降は10分間隔で送信



プロファイリンググループの作成



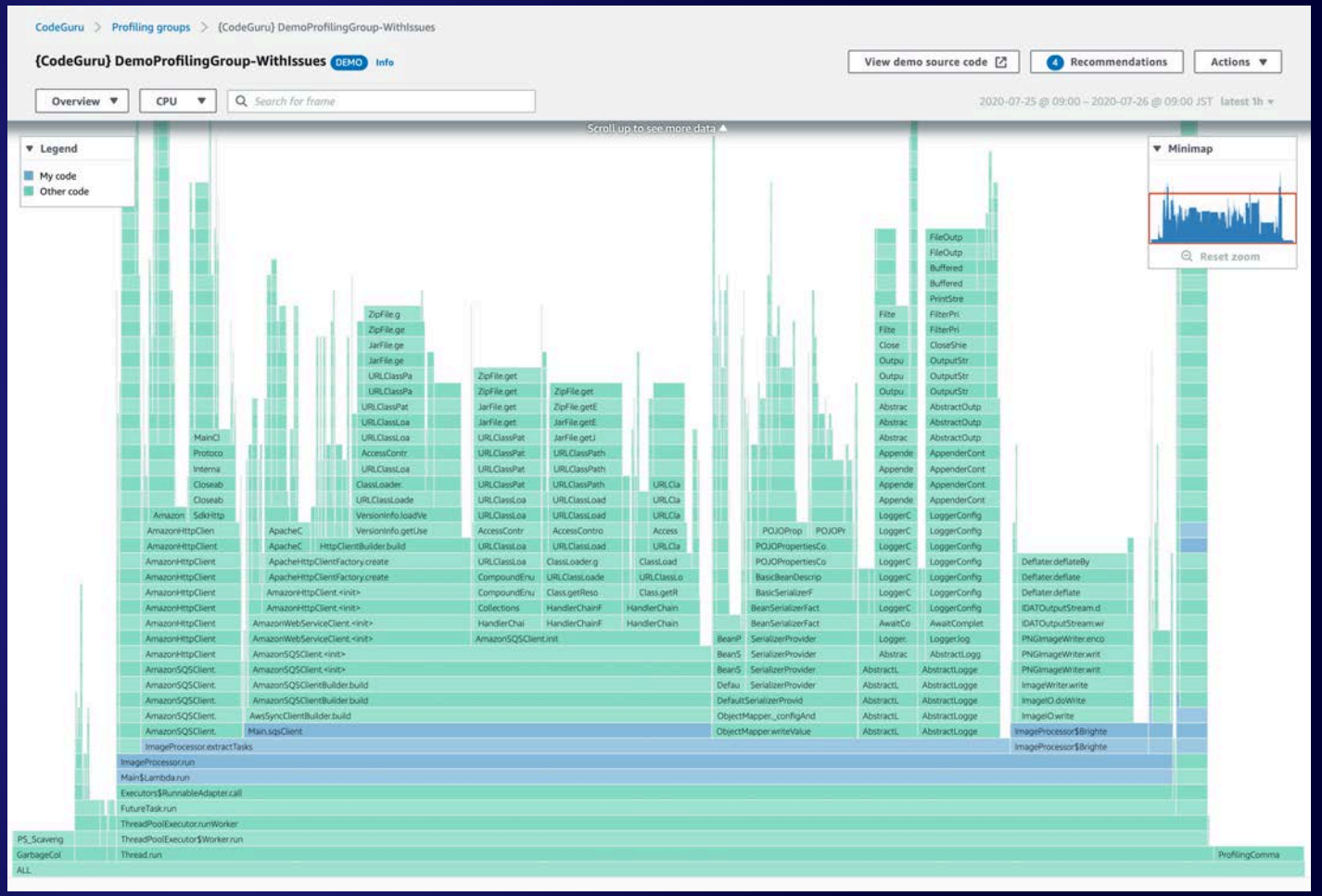
IAM権限の設定

CodeGuru Profiler の有効化

	AWS Lambda		その他プラットフォーム	
Java / JVM	方法1 コードに Profiler を組み込む (全てのJavaランタイム)	方法2 AWS Lambda Layer を使う (Java 8 Corretto とJava 11)	方法1 JVM エージェント (推奨)	方法2 コードに Profiler を組み込む
	• Lambda 環境変数に Profiler 情報を設定し、Maven や Gradle の設定ファイルに依存関係を記述 • コードを変更し Lambda 内でプロファイリングを開始	• Lambda環境変数にProfiler情報を設定 • CodeGuru Profiler Java Agent を AWS Lambda Layer として追加	• CodeGuru Profiler Agent JAR ファイルをダウンロードし配置 • Java アプリケーション起動時に javaagent オプションで Profiler エージェントを指定	• Maven や Gradle の設定ファイルに依存関係を記述 • Mainクラスにて Profiler エージェントを起動
Python (発表時点で Preview 中につき一部制約あり)	方法1 ハンドラー関数をデコレートする	方法2 Lambda Layer を使う	方法1 コードに Profiler を組み込む	方法2 コマンドラインでエージェントを有効化
	• Pip で Profiler エージェントをインストールし Lambda の zip ファイルに含める • プロファイリンググループの情報を含めた形で Lambda ハンドラー関数を @with_llambda_profiler でデコレート	• Lambda環境変数にProfiler情報を設定 • CodeGuru Profiler Python AgentをAWS Lambda Layer として追加 • Lambdaハンドラー関数を codeguru_profiler_agent.aws_lambda.lambda_handler.call_handler に変更	• Pip で Profiler エージェントをインストール • Profiler オブジェクトにプロファイリンググループの情報などを渡し、コード内で start する	• Pip で Profiler エージェントをインストール • プロファイリンググループ情報を環境変数またはコマンドラインの引数で渡し Python スクリプトを起動する

CodeGuru Profiler のビジュアライゼーション

- アプリケーションのスタックトレースのサンプリングを集約したものであり、最も実行コストが高いコード行を把握するために役立つ
- 各フレームには関数やCPU消費時間に関する情報が表示される
- 3種類のビジュアライゼーションを提供: Overview, Hotspots, Inspect
- 2カテゴリ、計4種類のビューを提供
  - Thread States
    - CPU, Latency, Custom
  - Heap Summary (Java/JVMのみ) New!



例: CodeGuru Profiler Overview モード / CPU ビュー

ビジュアライゼーションがどのように生成されるか

ビジュアライゼーションはスタックトレースのサンプリングである

サンプル1

```
Thread main
  java.lang.Thread.State: RUNNABLE
    com.amazon.profiler.demo.Example.doOne()
    com.amazon.profiler.demo.Example.doPlenty()
    com.amazon.profiler.demo.Example.main(String[])
```

サンプル2

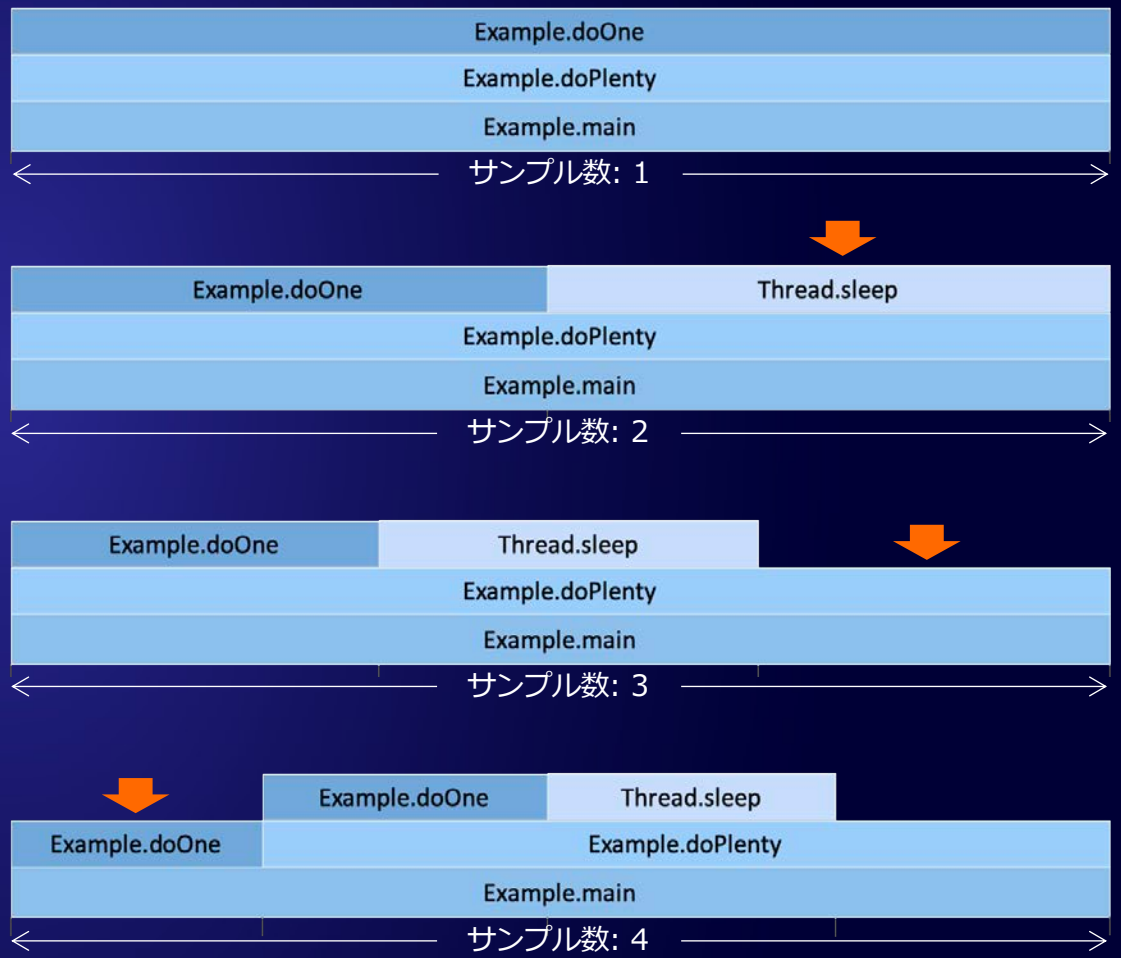
```
Thread main
  java.lang.Thread.State: TIMED_WAITING
    java.lang.Thread.sleep(long)
    com.amazon.profiler.demo.Example.doPlenty()
    com.amazon.profiler.demo.Example.main(String[])
```

サンプル3

```
Thread main
  java.lang.Thread.State: RUNNABLE
    com.amazon.profiler.demo.Example.doPlenty()
    com.amazon.profiler.demo.Example.main(String[])
```

サンプル4

```
Thread main
  java.lang.Thread.State: RUNNABLE
    com.amazon.profiler.demo.Example.doOne()
    com.amazon.profiler.demo.Example.main(String[])
```



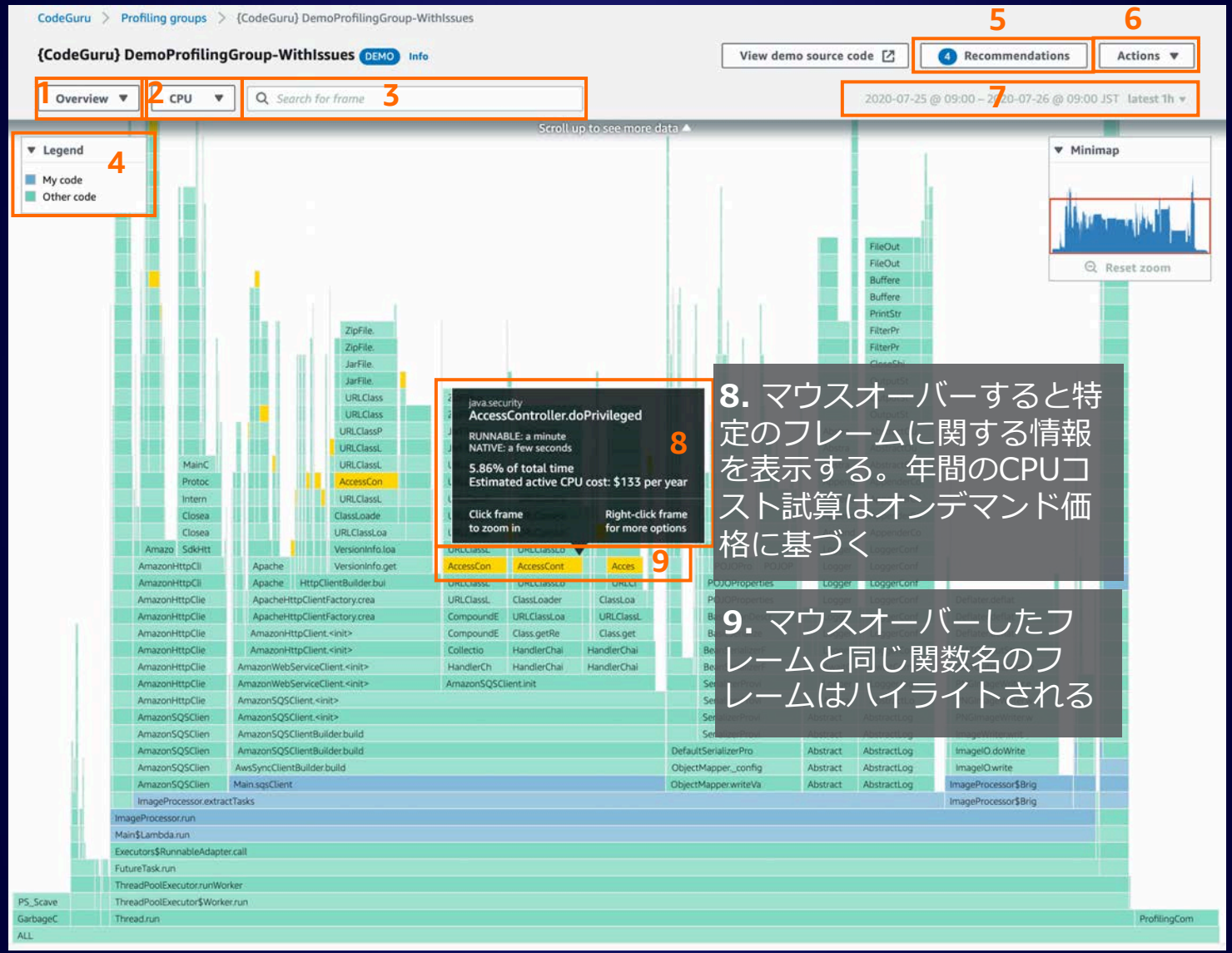
ビジュアライゼーションの解説 (Overview モード / CPU)

1. ビジュアライゼーションモードの切り替え (Overview, Hotspots, Inspect)

2. ビューの切り替え (CPU/Latency/Custom/ Heap Summary)

3. 特定のフレームを検索し、Inspect モードでドリルダウンできる

4. 自身のコードとライブラリ/フレームワークのコードを区別



8. マウスオーバーすると特定のフレームに関する情報を表示する。年間のCPUコスト試算はオンデマンド価格に基づく

9. マウスオーバーしたフレームと同じ関数名のフレームはハイライトされる

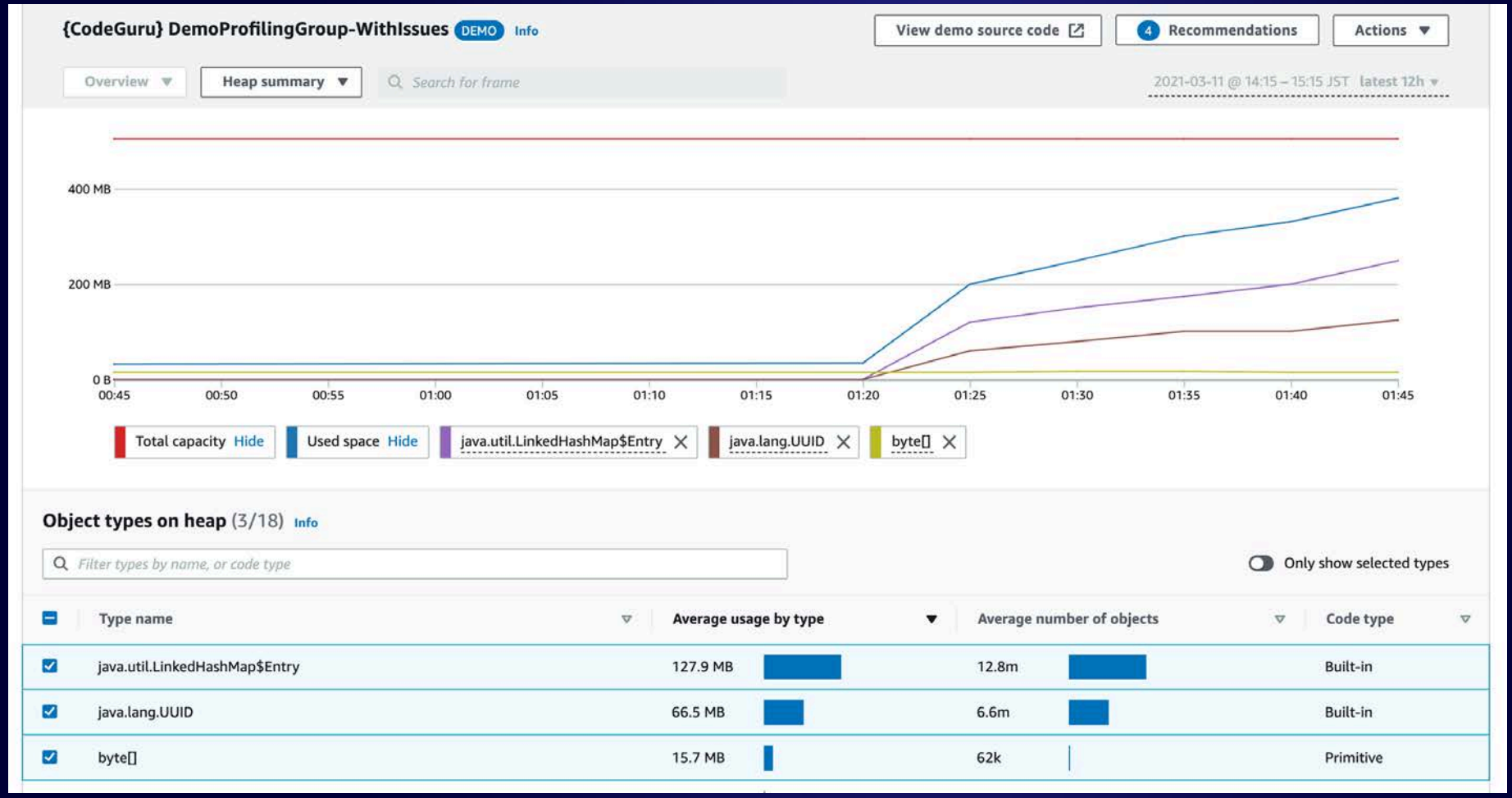
5. CodeGuru Profiler による推奨事項を表示

6. My code とみなすネームスペースの選択、すべての推奨事項レポート表示などのアクション

7. ビジュアライゼーションの時間範囲

メモリプロファイリング New!

JavaのHeapの使用状況を時系列で可視化

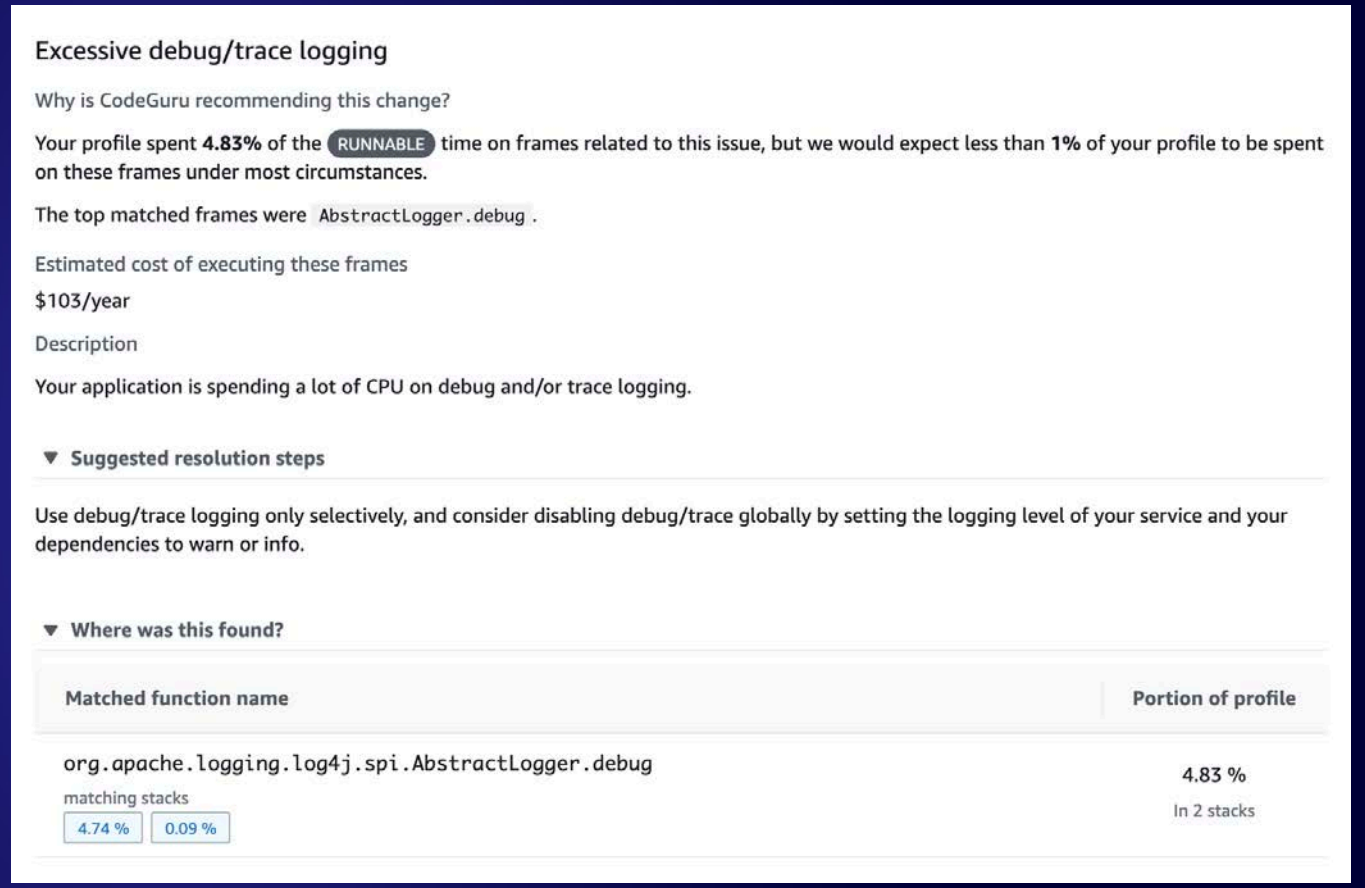


確保されているHeapと使用状況の推移

オブジェクトタイプごとのメモリ使用状況

CodeGuru Profiler 推奨事項

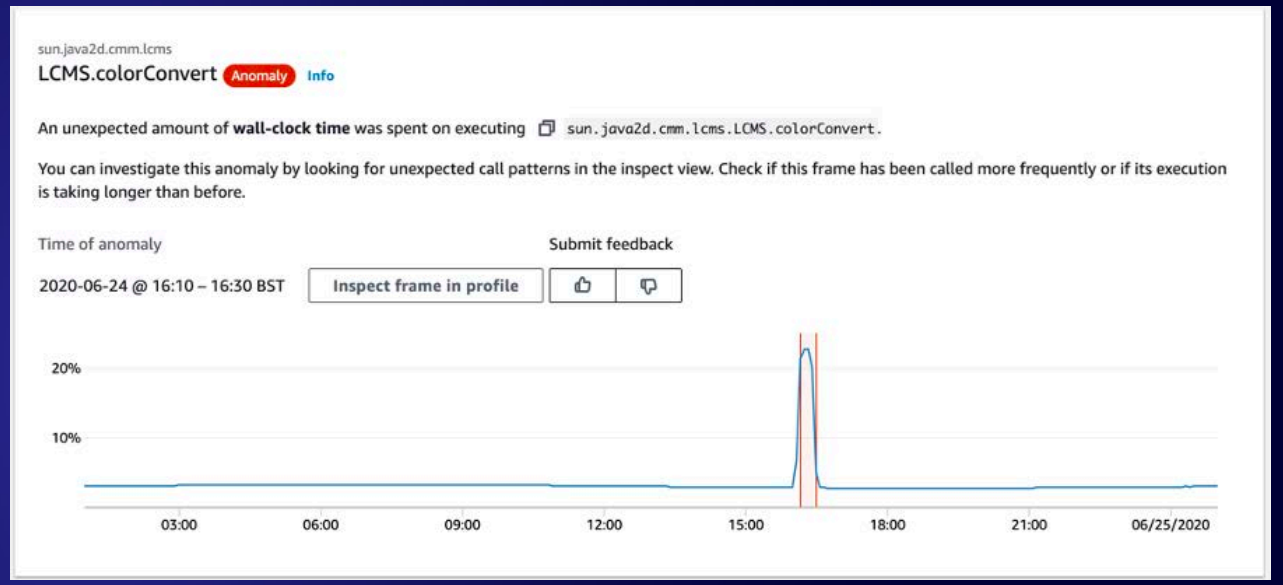
- 推奨事項はCPUリソースを無駄にするようなアンチパターンに対して提示される
- 推奨事項には以下内容が含まれる
  - What/Why: 課題と背景
  - What/Why: オンデマンド価格に基づいた年間推定コスト
  - How: 改善方法及び関連するドキュメントへのリンク
  - Where: 対象となる関数名



例: ロギング処理のCPU使用率が高いことを示す推奨事項

CodeGuru Profiler 異常検知

- 過去のプロファイリングデータと比べ、 CPU 使用率 または wall clock time (処理の実測時間) の乖離が大きい場合、異常として判定しレポートに出力される
- Amazon SNSへの通知をサポート
- 推奨事項には以下内容が含まれる
  - What: 対象のフレーム
  - Why: 課題の背景およびグラフ
  - Where: Inspectモードでの確認
  - フィードバックの提出
- 異常検知は機械学習をベースとしており、フィードバックすることで CodeGuru Profiler の精度向上に繋がる
- Java/JVMのみサポート



例: とある関数の実測時間が異常に長かった場合

まとめ

本セッションのまとめ

開発における俊敏性
機械学習の活用によって開発サイクルを加速

コード品質とアプリケーション可用性の高い基準
CodeGuruはすぐに取りかかれる推奨事項を生成

Amazon CodeGuru Reviewer

- コードの欠陥を特定し、改善方法を含めた推奨事項を提示
- Pull Request、リポジトリ解析、セキュリティ解析の3つのワークフローでコードレビューを開始
- 既存CI/CDパイプラインと簡単に統合

Amazon CodeGuru Profiler

- アプリケーションのパフォーマンス改善に向けた推奨事項や異常検知をレポート
- 既存CI/CDパイプラインと簡単に統合
- リッチなビジュアライゼーションを提供

参考リソース

CALL TO ACTION/NEXT STEPS

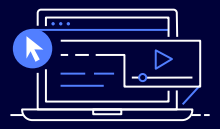
- CodeGuruは90日間の無償期間を提供しています。以下の料金ページをご参考ください
  aws.amazon.com/codeguru/pricing
- サンプルのアプリケーションコード
  github.com/aws-samples/amazon-codeguru-reviewer-sample-app
- サービスページ
  aws.amazon.com/codeguru

Thank you!

Yumiko Kanasugi
yumikan@amazon.co.jp

AWS トレーニングと認定

AWS クラウドをキャリアに活用してください



デジタルトレーニング

クラウドのスキルを構築する無料のオンデマンドコースを探索する

クラスルームトレーニング

エキスパートインストラクターによるトレーニングに参加する

AWS 認定の取得

業界で認められている認定を取得する

教育プログラム

AWS のスキルと経験を持つ人材に出会える

エンタープライズリソース

学習ニーズ分析とAWSランプアップガイドを活用する

詳細はこちら　https://aws.amazon.com/jp/training/