[AWS Black Belt Online Seminar] Amazon CodeGuru

サービスカットシリーズ

Solutions Architect
Yumiko Kanasugi
2020/8/4

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive

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

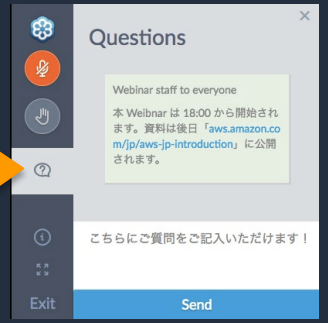
AWS Black Belt Online Seminar とは

「サービス別」「ソリューション別」「業種別」のそれぞれのテーマに分かれて、アマゾン ウェブ サービス ジャパン株式会社が主催するオンラインセミナーシリーズです。

質問を投げることができます！

- 書き込んだ質問は、主催者にしか見えません
- 今後のロードマップに関するご質問は
  お答えできませんのでご了承下さい

① 吹き出しをクリック
② 質問を入力
③ Sendをクリック



Twitter ハッシュタグは以下をご利用ください
#awsblackbelt

内容についての注意点

- 本資料では2020年8月4日現在のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。

- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。

- 価格は税抜表記となっています。日本居住者のお客様には別途消費税をご請求させていただきます。

- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

本セミナーの概要

- 本セミナーで学習できること
  - Amazon CodeGuru の概要
  - Amazon CodeGuru Reviewer と Profiler の詳細
  - Amazon CodeGuru の始め方

- 対象者
  - 技術者の方
  - アプリケーション開発者の方
  - コードレビュープロセスを効率化したい方
  - アプリケーションパフォーマンスを最適化したい方
  - IT 知識レベル:★★☆☆☆
  - AWS 知識レベル:★★☆☆☆

本日のアジェンダ

- 背景
- Amazon CodeGuru とは
- Amazon CodeGuru Reviewer
- Amazon CodeGuru Profiler
- セキュリティ
- サービスクォータ
- 料金体系
- まとめ

本日のアジェンダ

- **背景**
- Amazon CodeGuru とは
- Amazon CodeGuru Reviewer
- Amazon CodeGuru Profiler
- セキュリティ
- サービスクォータ
- 料金体系
- まとめ

開発における一般的なフロー

アプリケーションコードとパフォーマンスを
継続的に改善する必要がある

アプリケーション改善におけるチャレンジ

開発者がコードの問題を特定するのに労力と時間がかかる

コード解析ツールはコード品質と効率に対する標準ベストプラクティスを示さない

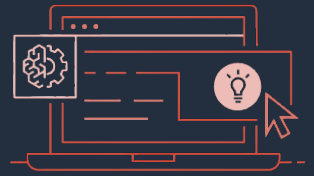
実行コストが高くなっているコード箇所を特定し改修することが困難

本日のアジェンダ

- 背景
- **Amazon CodeGuru とは**
- Amazon CodeGuru Reviewer
- Amazon CodeGuru Profiler
- セキュリティ
- サービスクォータ
- 料金体系
- まとめ

Amazon CodeGuru とは

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

- 機械学習を駆使し、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
- 改善方法を提示することでコード品質の維持に繋がる
- 現時点で Java 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
- 改善方法を提示することでインフラストラクチャ費用の削減に繋がる
- 現時点で JVM ベースの言語に対応

2つの機能は独立しており、単体で使用可能

AWS AI/MLスタック

Amazon CodeGuru はAIサービスとして位置付けられている

2020年6月29日 GA!

AI SERVICES

- VISION: Amazon Rekognition
- SPEECH: Amazon Polly; Amazon Transcribe (+Medical NEW)
- TEXT: Amazon Comprehend (+Medical); Amazon Translate; Amazon Textract
- SEARCH: Amazon Kendra
- CHATBOTS: Amazon Lex
- PERSONALIZATION: Amazon Personalize
- FORECASTING: Amazon Forecast
- NEW! FRAUD: Amazon Fraud Detector
- NEW! DEVELOPMENT: Amazon CodeGuru
- NEW! CONTACT CENTERS: Contact Lens (For Amazon Connect)

ML SERVICES

Amazon SageMaker

Ground Truth | Augmented AI | ML Marketplace

SageMaker Studio IDE

Built-in algorithms | Notebooks | Experiments | Model training & tuning | Debugger | Autopilot | Model hosting | Model Monitor

Neo

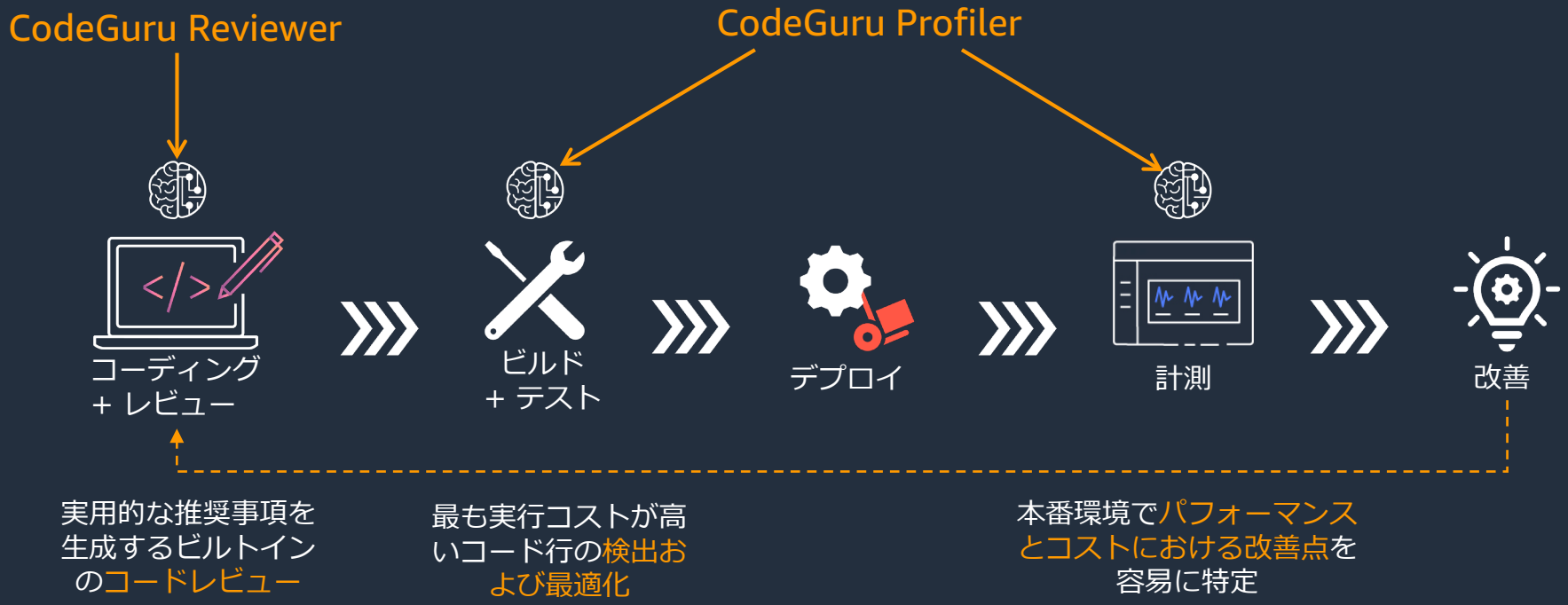
ML FRAMEWORKS & INFRASTRUCTURE

NEW! TensorFlow | NEW! mxnet | NEW! PYTORCH

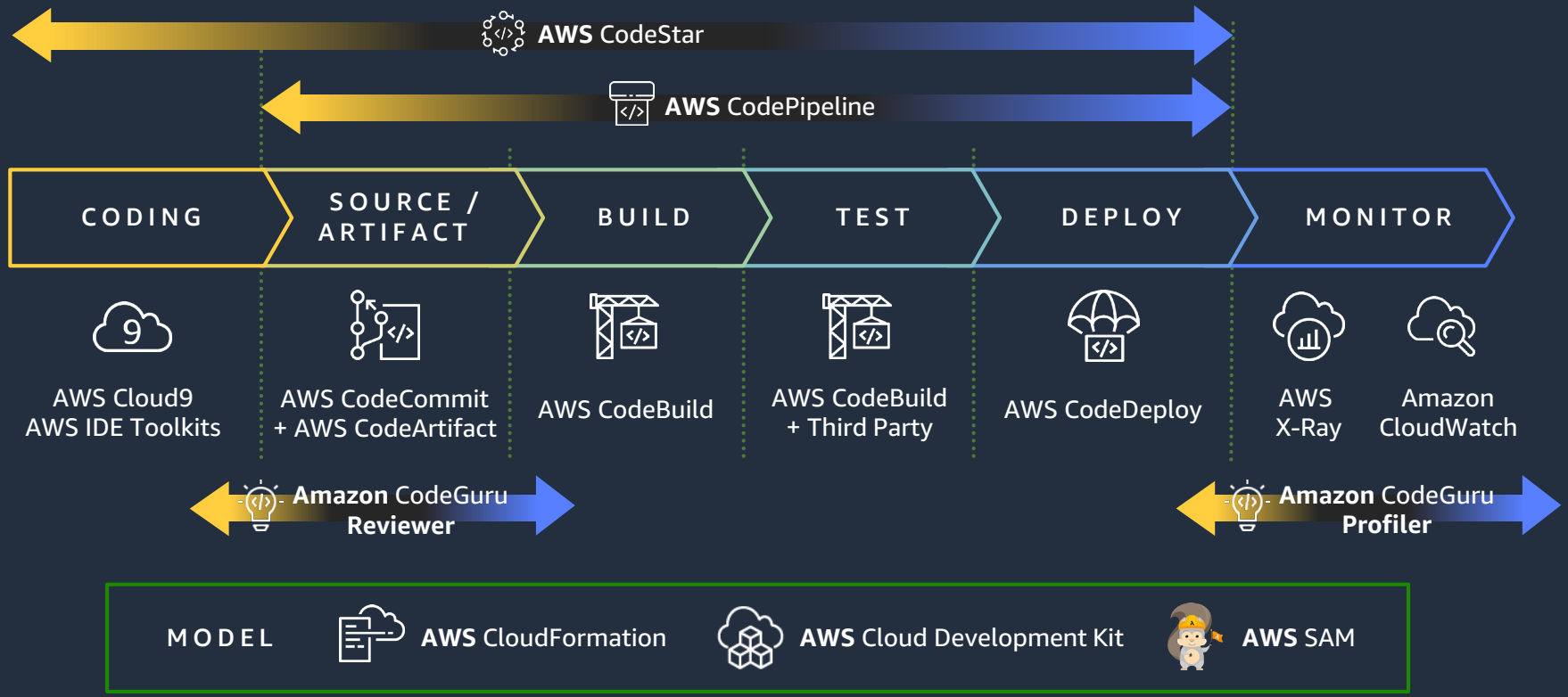
GLUON | Keras

Deep Learning AMIs & Containers | GPUs & CPUs | Elastic Inference | Inferentia | FPGA

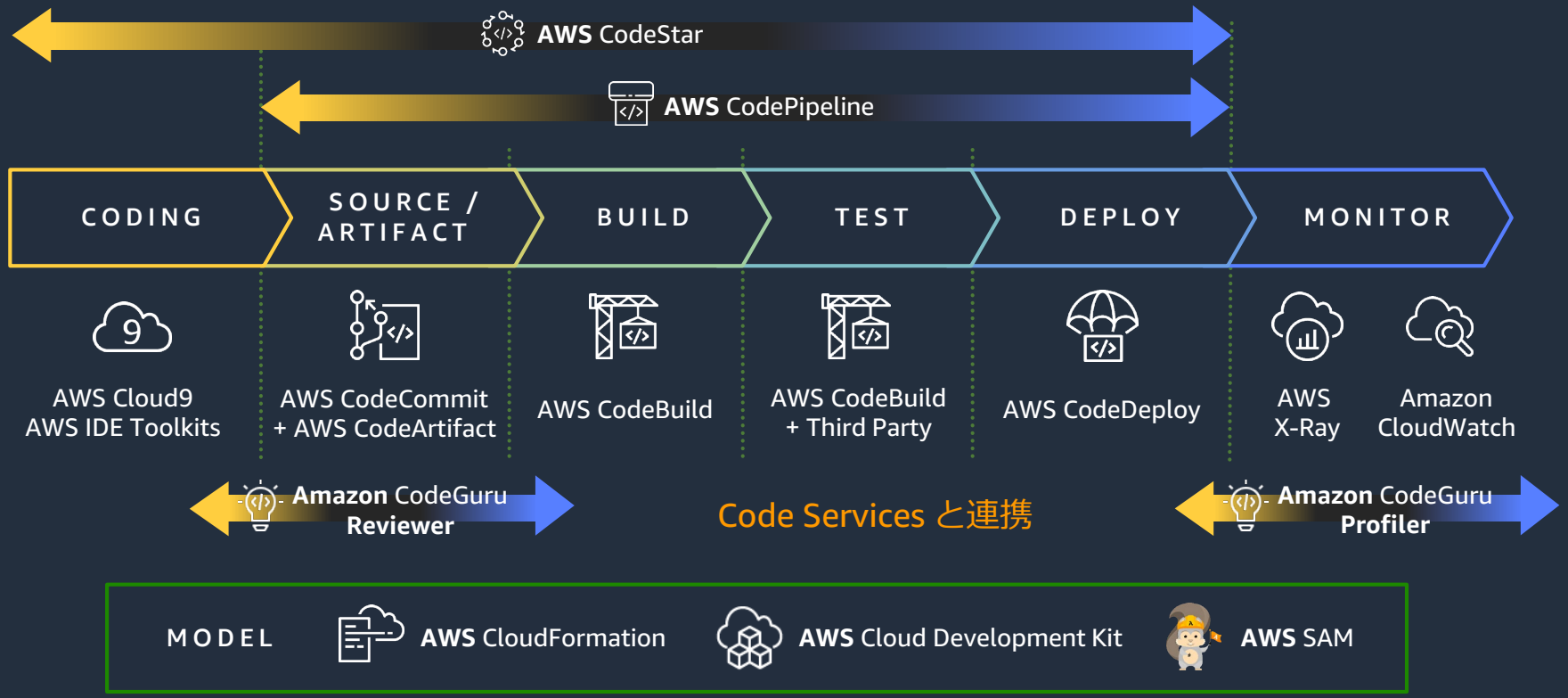
一般的な開発フローにおけるAmazon CodeGuru の立ち位置



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



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 提供リージョン

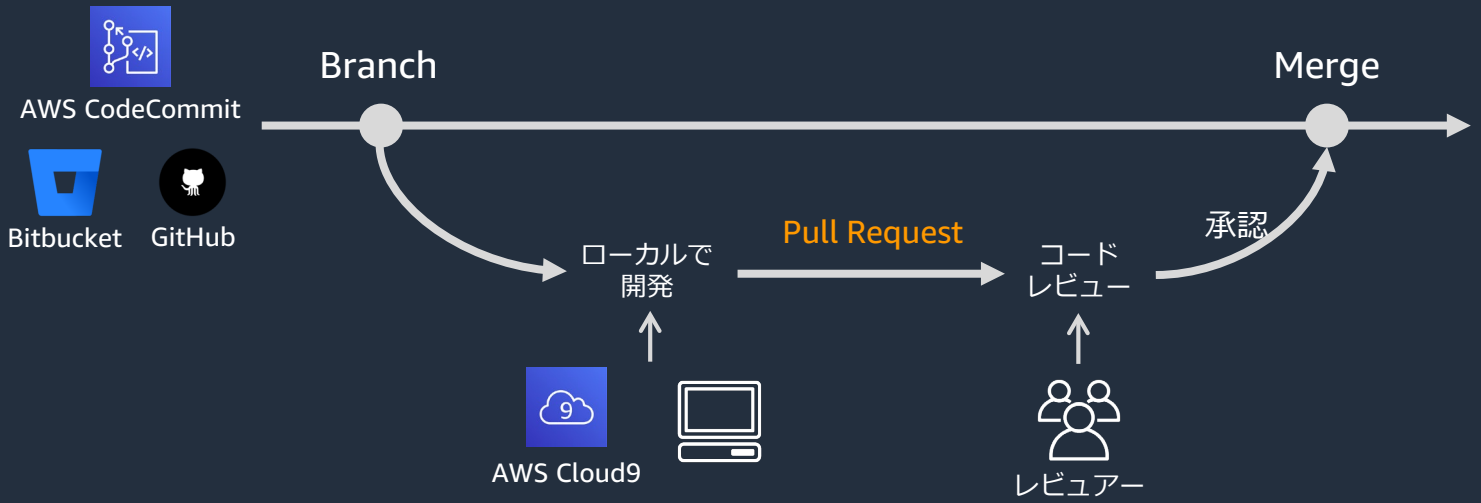
東京リージョンを含む以下リージョンにて利用可能 (2020年8月4日 本日時点)

利用可能なリージョン	リージョンID
米国東部(バージニア北部)	us-east-1
米国東部(オハイオ)	us-east-2
米国西部(オレゴン)	us-west-2
アジアパシフィック(シンガポール)	ap-southeast-1
アジアパシフィック(シドニー)	ap-southeast-2
アジアパシフィック(東京)	**ap-northeast-1**
欧州(フランクフルト)	eu-central-1
欧州(アイルランド)	eu-west-1
欧州(ロンドン)	eu-west-2
欧州(ストックホルム)	eu-north-1

本日のアジェンダ

- 背景
- Amazon CodeGuru とは
- **Amazon CodeGuru Reviewer**
- Amazon CodeGuru Profiler
- セキュリティ
- サービスクォータ
- 料金体系
- まとめ

コードレビュープロセスにおける課題



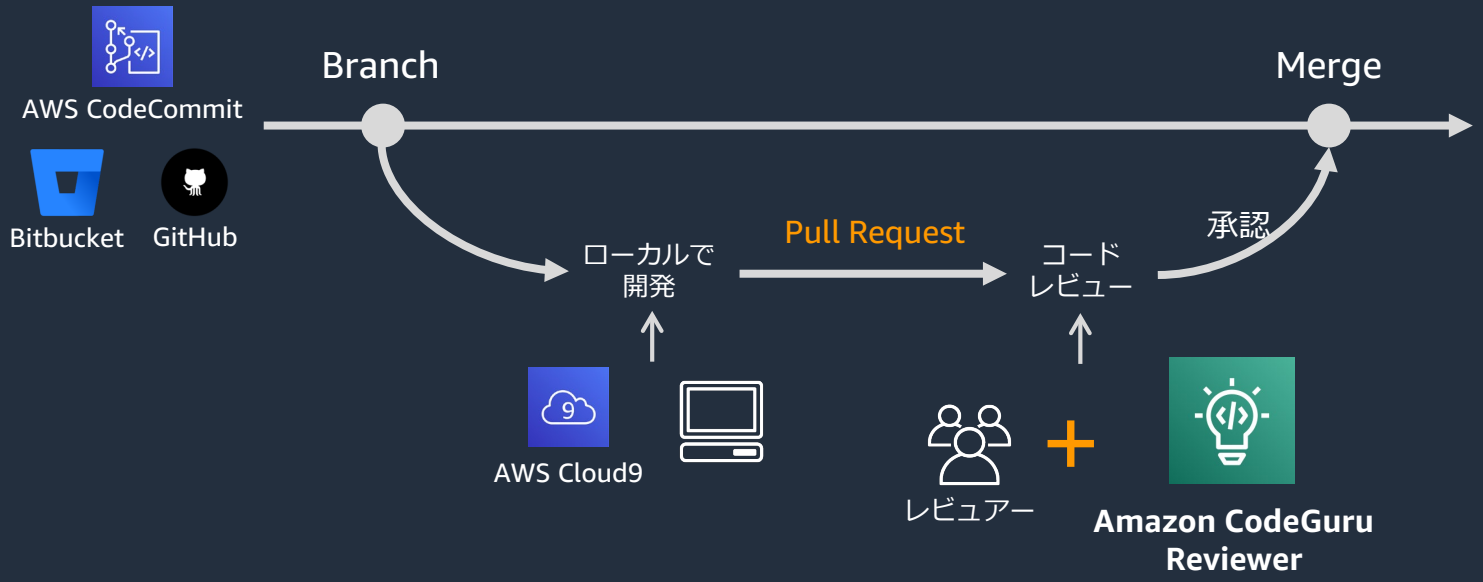
ビジネスロジックの検証

複数領域における専門知識

大量のソースコード

十分なコードレビュー人材の確保は多くの企業にとっての課題である

CodeGuru Reviewer の位置付け



Amazon CodeGuru Reviewer でコードレビューの負担を軽減

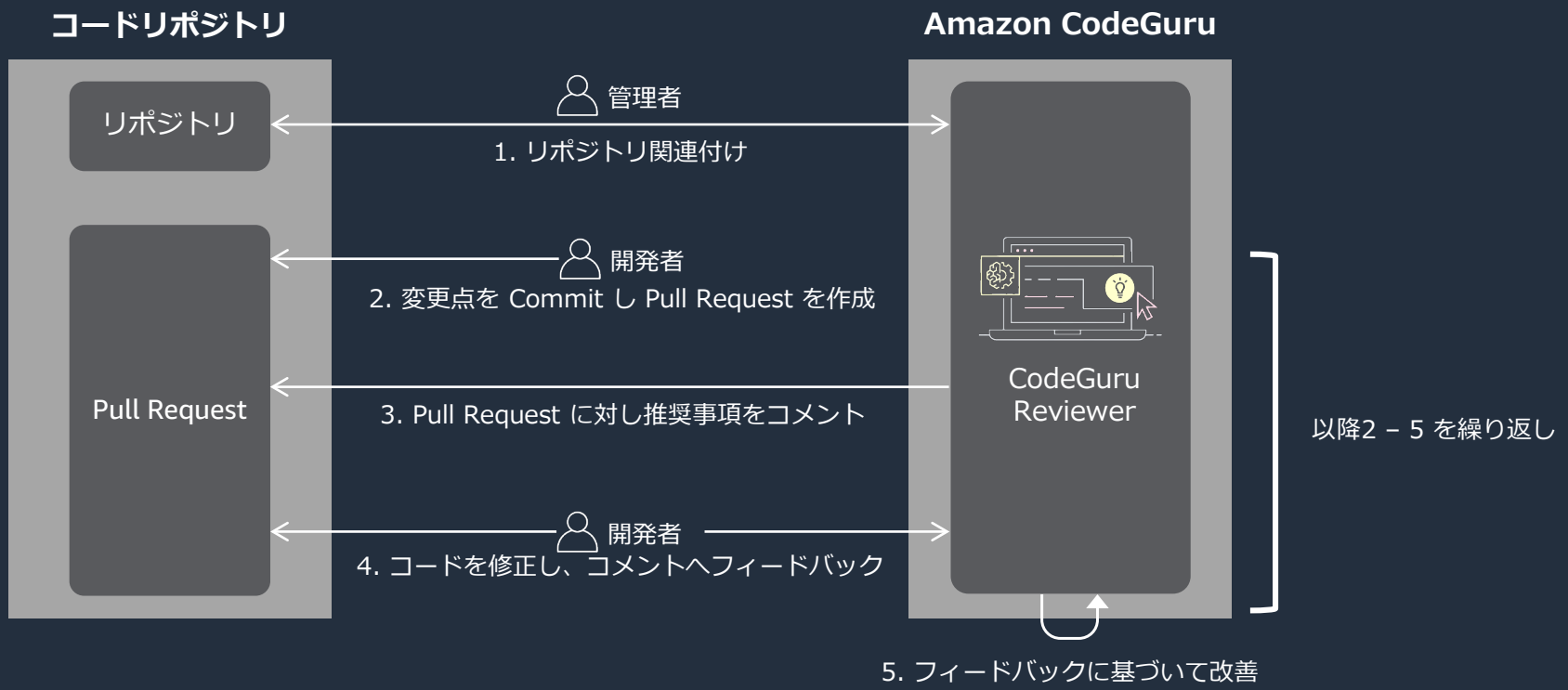
Amazon CodeGuru Reviewer とは

開発者は機械学習のメリットを享受し、自動的にベストプラクティに沿っていない問題箇所を特定することができ、本番環境に影響を与える可能性のある問題を未然に防ぐことができる

コードの具体的な改善方法も推奨事項に含まれており、開発者はコード例や関連ドキュメントを参照することが可能

開発者はAmazon CodeGuru Reviewerをレビュアーの一員としてみなすことができ、追加のソフトウェアインストール不要、通常の開発ワークフローに変更なく簡単に利用を始められる

CodeGuru Reviewer の動作イメージ

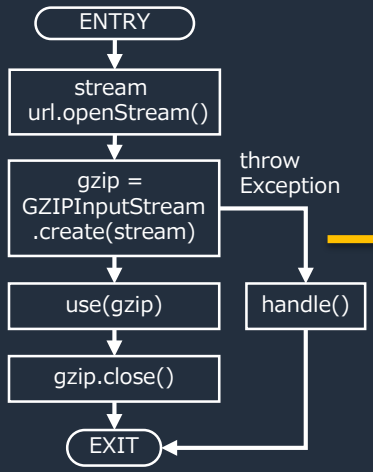


CodeGuru Reviewer の仕組み

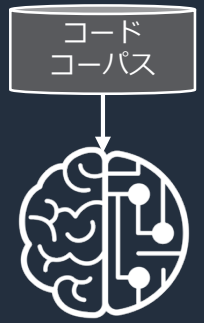
開発者が
Pull Request を作成

```
try (GZip gzip  =
    GZIPInputStream.create(
        url.openStream())) {
    use(gzip);
} catch (Exception e) {
    handle();
}
```

機能やパターンの
意味を抽出



MLアルゴリズム + プログラム
解析 によりコードの欠陥を特定



Pull Requestコメントとして
推奨事項を確認

リソースリークの
可能性

url.openStream()

Use try-with-
resources

インプット:
ソースコード

機能の抽出

機械学習による
処理

アウトプット:
推奨事項
(レコメンデーション)

Amazon CodeGuru Reviewer の対応リポジトリ

以下のリポジトリに対応 (2020年8月4日 本日時点)

AWS CodeCommit

AWSマネージドサービスとしてのGitベースの
ソースコードリポジトリ
IAMによるアクセス制御と高い耐久性を提供

GitHub

GitHub社が提供するGitベースのサービス

GitHub Enterprise Cloud

GitHub Enterprise のクラウド版

GitHub Enterprise Server

GitHub Enterprise のオンプレミス版
(AWS CodeStar Connection 経由)

Bitbucket Cloud

Atlassian社が提供するGitベースのサービス
(AWS CodeStar Connection 経由)

CodeGuru Reviewer のレコメンデーション

- AWSベストプラクティス
    - 正しい AWS API の使い方 (例: ページネーションによる正確性の担保)
- 並列処理
    - マルチスレッド処理の適切な実装 (例: 同期漏れによる意図しないデータ不整合)
- リソースリーク
    - 正しいリソースの扱い (例: DBコネクション解放漏れによる可用性への影響)
- 機密データの漏洩
    - 機密情報の不必要な公開を防止 (例: クレジットカード番号のロギング)
- 一般的なコーディングベストプラクティス
    - コードの欠陥の発見 (例: オブジェクトが null かどうかの確認)
- リファクタリング
    - 冗長なコードの特定 (例: 同じコードとロジックが複数箇所で使われている)
- インプットバリデーション
    - 入力形式の確認 (例: インプットデータにバリデーションロジックが実装されていない)

CodeGuru Reviewer の始め方

AWSマネジメントコンソール > Amazon CodeGuru > Getting Started

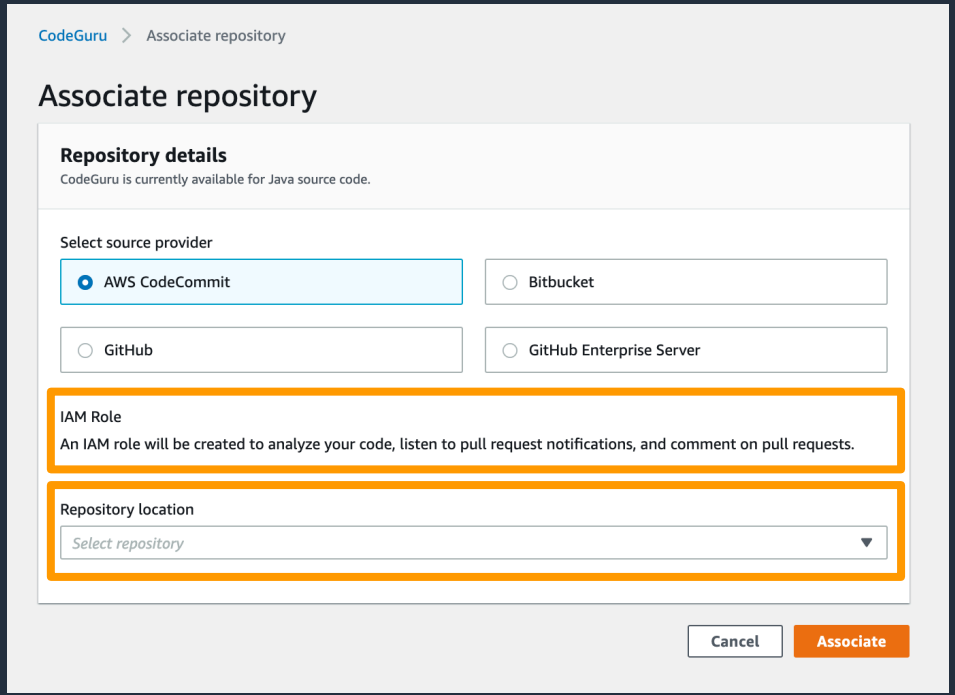
AWS CLI/API AssociateRepository

AWS CodeCommit 及び AWS CodeStar Connection を介した Bitbucket、GitHub Enterprise Server のリポジトリの関連付けに対応

https://docs.aws.amazon.com/codeguru/latest/reviewer-api/API_AssociateRepository.html

CodeGuru Reviewer リポジトリの関連付け

- ソースプロバイダ及び関連付けるリポジトリを選択
  - AWS CodeCommit の場合、対象リポジトリを直接プルダウンから選択可能
- AWS CodeCommit または AWS CodeStar Connection を介した接続の場合 (GitHub Enterprise Server 及び Bitbucket が該当) Amazon CodeGuru 用の IAM リソースが自動的に作成される。IAM Role 名は AWSServiceRoleForAmazonCodeGuruReviewer (Service Linked Role)
- IAM Policy には CodeGuru Reviewer が CodeCommit, CodeStar Connection, CloudWatch へアクセスする際に必要な権限を定義している

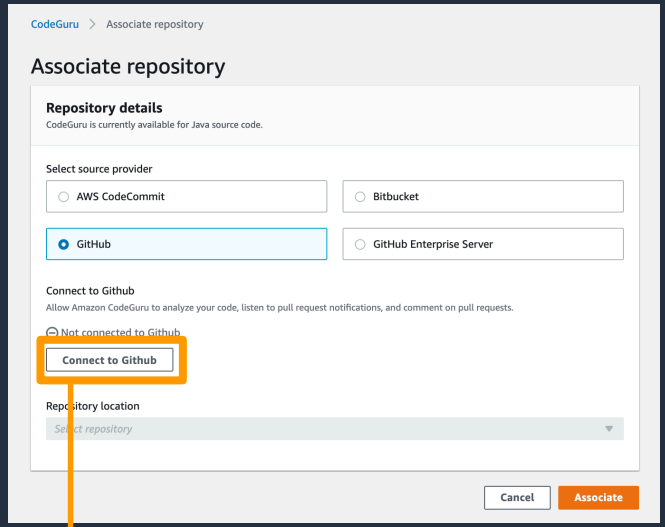


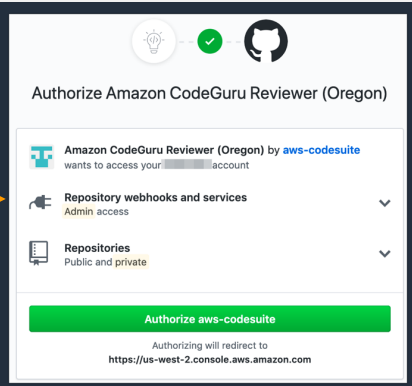
CodeGuru Reviewer - CodeCommitの利用例

CodeCommit の場合、リポジトリ新規作成画面で CodeGuru Reviewer との連携指定が可能 (後からの関連付けも可能)

CodeGuru Reviewer – GitHub & GitHub Enterprise Cloud の利用例

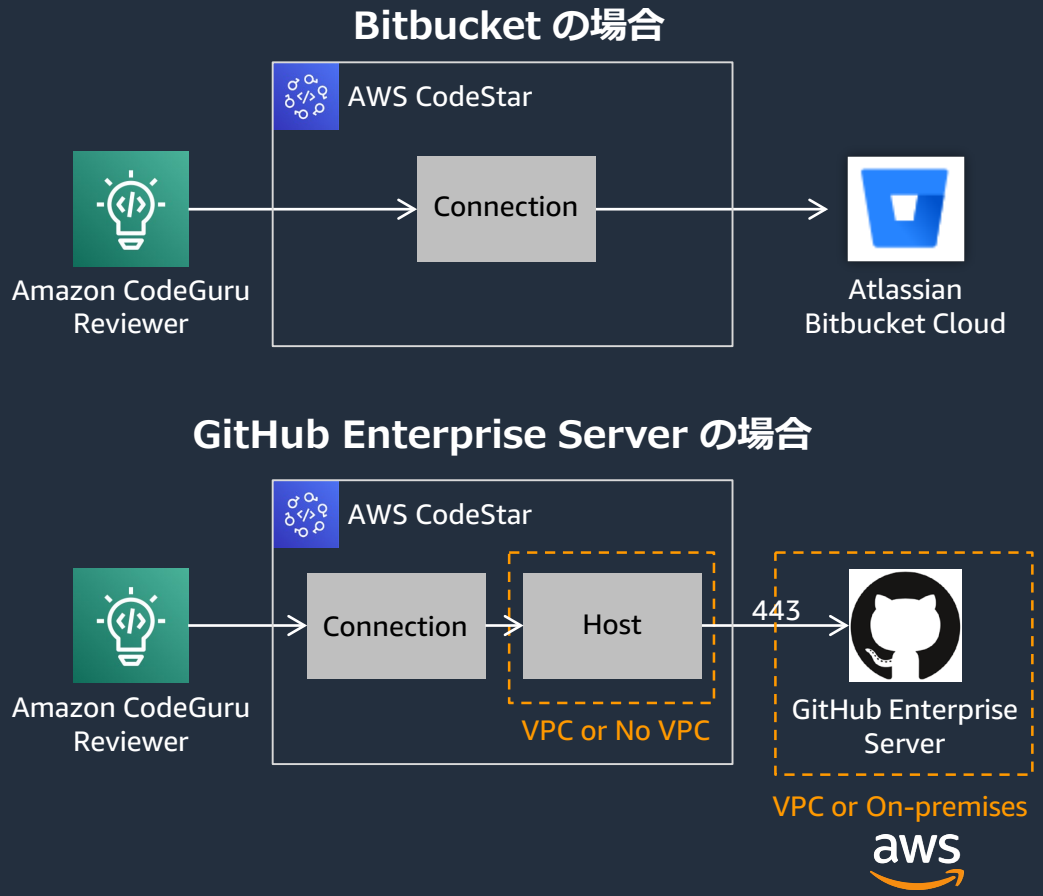
- ソースプロバイダにて GitHub を選択すると GitHub へリダイレクトされ、サインインを要求される
  - CodeGuru Reviewer はこの時サインインしたアカウントを使用して Pull Request にコメントする
  - 区別やメンテナンスがしやすいよう CodeGuru Reviewer 専用の GitHubユーザーを作成することを推奨
- CodeGuru Reviewer アプリケーション (aws-codesuite) からのアクセスを許可
- 接続後、コンソールから対象リポジトリを選択





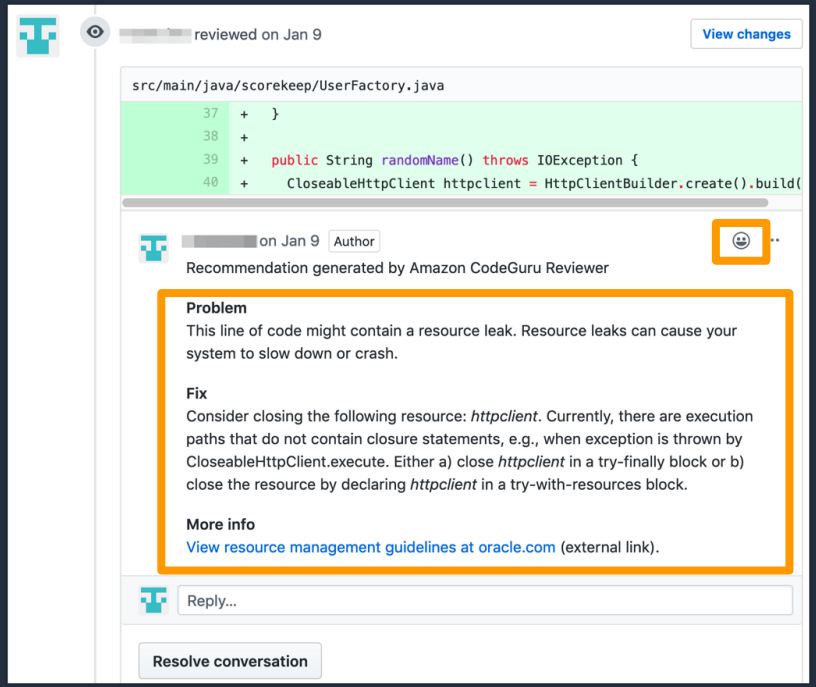
CodeGuru Reviewer – CodeStar Connection の利用例

- ソースプロバイダが Bitbucket 又は GitHub Enterprise Server の場合、AWS CodeStar Connection を介した接続が必要
- Bitbucketの場合、CodeStar Connection 作成時に CodeStar の Bitbucket アカウントへのアクセスを許可し、Amazon CodeGuru Reviewer を Bitbucket Cloud apps としてインストール
- GitHub Enterprise Server の場合、CodeStar Connection 及び Host を作成する必要がある
  - Host は任意でVPC内に作成しセキュリティグループを付与できる。GitHub Enterprise Serever とは別VPCでも可
  - GitHub Enterprise Server 側ではポート443番を許可する必要がある



CodeGuru Reviewer のレコメンデーションサンプル

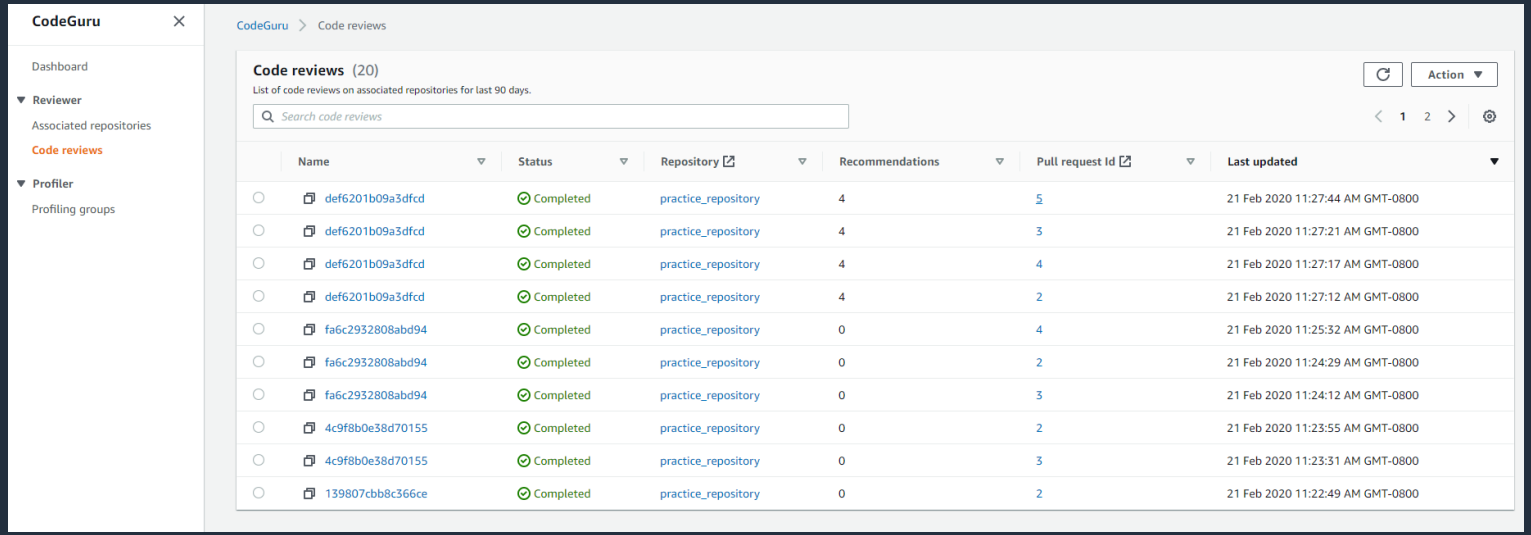
- リポジトリを関連付けた後は Pull Request に対して CodeGuru Reviewer が自動的にコメントを残す
  - 通常 Pull Request が作成されてから15分以内に完了
- 各レコメンデーションには Problem, Fix, More info などといった改善方法や関連するドキュメントへのリンクも含まれる
- コメントあるいはEmojiリアクションを通してレコメンデーションへフィードバックすることで CodeGuru Reviewer の精度向上に繋がる
- コードレビューやレコメンデーション一覧はマネジメントコンソールあるいはAPIで取得可能



例: リソースリークの可能性があるコードを指摘

CodeGuru Reviewer コードレビュー結果

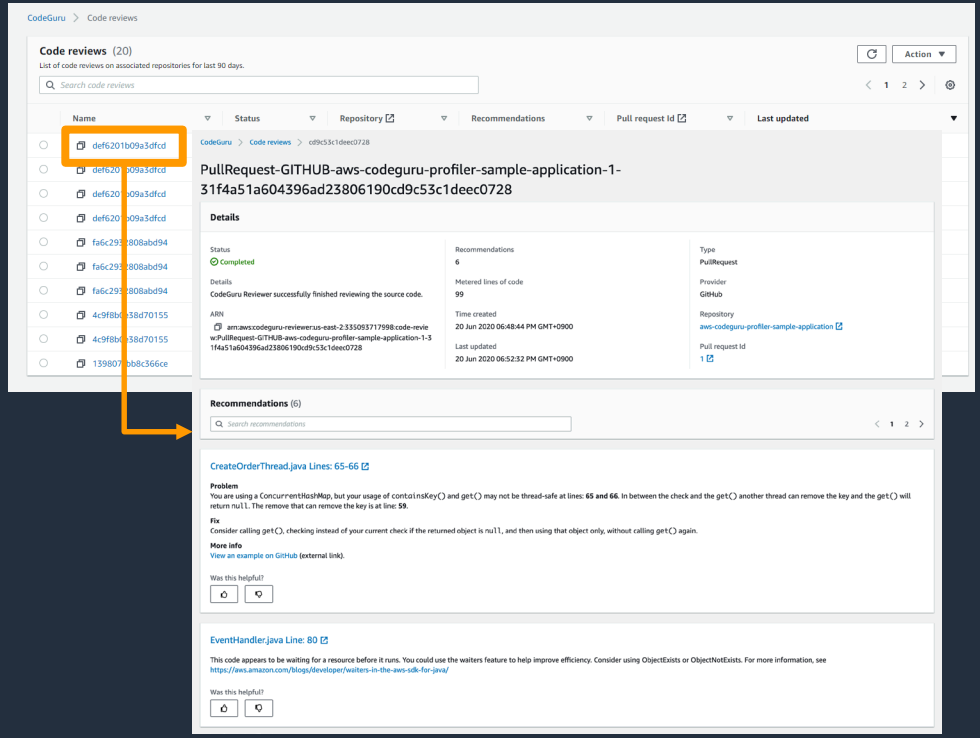
Pull Request のコメントだけでなく、コンソールの Code Reviews から過去のレビュー結果を確認できる



マネジメントコンソール > Amazon CodeGuru > Reviewer > Code reviews

CodeGuru Reviewer Code reviews

- コードレビューは Pull Request と1対1で紐づいている
- マネジメントコンソールからレコメンデーションに対し ポジティブ/ネガティブ フィードバックが可能
- マネジメントコンソールから過去90日間のコードレビュー結果を確認可能 (ListCodeReviews, DescribeCodeReview API)
- コードレビューには以下3つのステータスが存在する
  - Pending
  - Completed
  - Failed



Code Reviewの詳細画面

CodeGuru Reviewer の特徴

- ルールベースおよび機械学習ベースのモデルの両方を使用してインテリジェントなレコメンデーションを提供
  - モデルはオープンソースプロジェクトや Amazon.com のアプリケーションコード及びコードレビューによってトレーニングされている
- 並列処理などの発見が困難なバグやAWSベストプラクティスを含む幅広いレコメンデーションを生成可能
- 精度の高い (即ちFalse Positive が少ない) レコメンデーションを提供
- 既存のデベロッパーワークフローに大きな変更なく利用を開始できる

本日のアジェンダ

- 背景
- Amazon CodeGuru とは
- Amazon CodeGuru Reviewer
- **Amazon CodeGuru Profiler**
- セキュリティ
- サービスクォータ
- 料金体系
- まとめ

アプリケーションパフォーマンス解析における課題

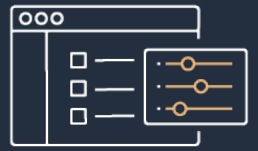
パフォーマンスエンジニアリングの深い知見が必要

ベストプラクティスを学習する必要がある

継続的にパフォーマンス解析を行う必要がある

パフォーマンスの課題点の特定が困難

実用的な推奨事項が欲しい

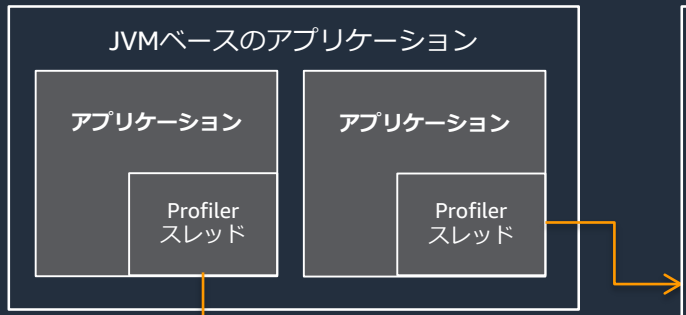


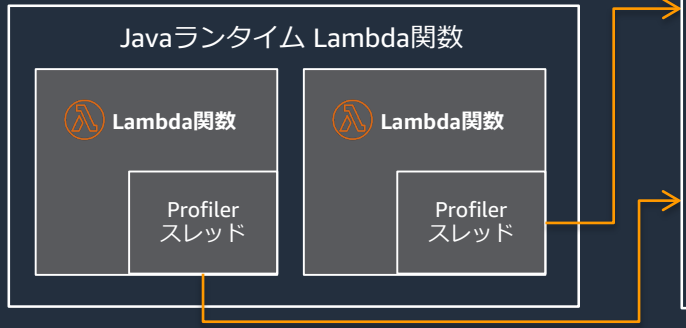
コード修正の優先順位を付けたい

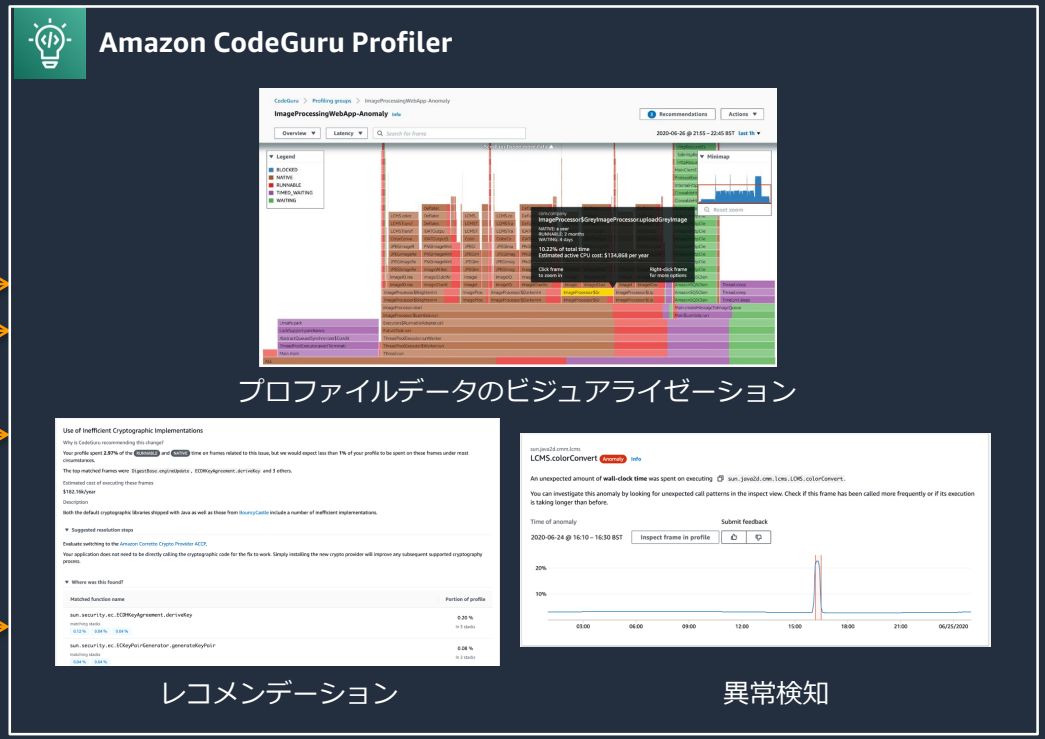
aws

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







Amazon CodeGuru Profiler とは

Amazon CodeGuru Profilerにより、開発者は機械学習をベースとしたアプリケーションプロファイラを使用して最も実行コストが高いコード行を特定することができる

JVMアプリケーションのプロファイルデータを継続的に収集し、ランタイムのパフォーマンス解析のためのビジュアライゼーション及びインサイトを提供する

よりCPU使用効率を向上するための改善方法を含む推奨事項が提供される他、過去のプロファイリングデータと乖離があった際に異常検知を行う

aws

CodeGuru Profiler の始め方

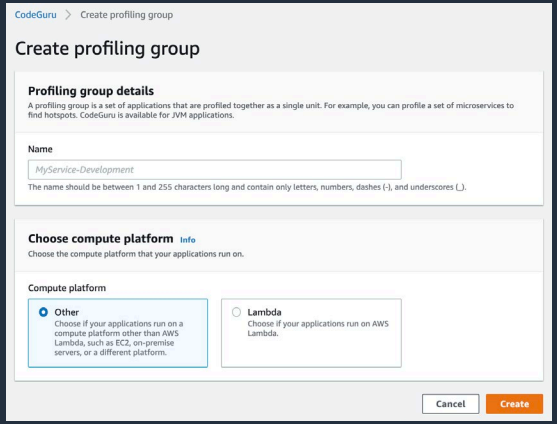
1. CodeGuru Profiler プロファイリンググループの作成

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

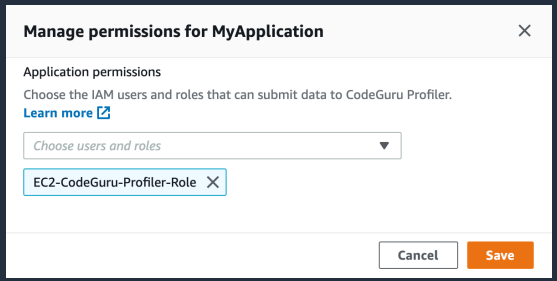
2. IAM権限の設定

   Profilerエージェントが使用する IAM User/Role に CodeGuru へ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
   - 2つの方法で起動:
     1. JVMエージェントを使用 (推奨)
     2. コードにProfilerを組み込む



プロファイリンググループの作成



IAM権限の設定

CodeGuru Profiler エージェント

方法1 JVM エージェントを使用 (推奨)

- コードの書き換えやコンパイルが不要
- 手順
  - 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

```
java -javaagent:/path/to/codeguru-profiler-java-agent-standalone-
1.0.0.jar=profilingGroupName:<MyProfilingGroup>,region:<region-code> -jar MyApplication.jar
```

方法2 コードに Profiler を組み込む

- プロファイリングを柔軟に制御できる
- 手順
  - 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

```
import software.amazon.codeguruprofilerjavaagent.Profiler;

class MyApplication {
    public static void main(String[] args) {
        Profiler.builder()
            .profilingGroupName("MyProfilingGroup")
            .build()
            .start();
        ...
    }
}
```

AWS Lambda における CodeGuru Profiler の使用 (1/2)

- 設定項目
  - Lambda 環境変数に以下を追加
    - Key = AWS_CODEGURU_PROFILER_GROUP_ARN, Value = <プロファイリンググループの*ARN*>
    - Key = AWS_CODEGURU_PROFILER_ENABLED, Value = TRUE
  - 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
  - 次ページで以下3パターンを紹介
    - AWS Lambda が提供する RequestHandler を使用している場合
    - AWS Lambda が提供する RequestStreamHandler を使用している場合
    - AWS Lambda が提供するハンドラーを使用していない場合

AWS Lambda における CodeGuru Profiler の使用 (2/2)

AWS Lambda が提供する RequestHandler を使用している場合

```
package example;

import java.util.Map;

import com.amazonaws.services.lambda.runtime.Context;

import software.amazon.codeguruprofilerjavaagent.RequestHandlerWithProfiling;

public class Handler extends RequestHandlerWithProfiling<Map<String,String>, String> {
    @Override
    public String requestHandler(Map<String, String> input, Context context) {
        // Your function code here
    }
}
```

AWS Lambda が提供するハンドラーを使用していない場合

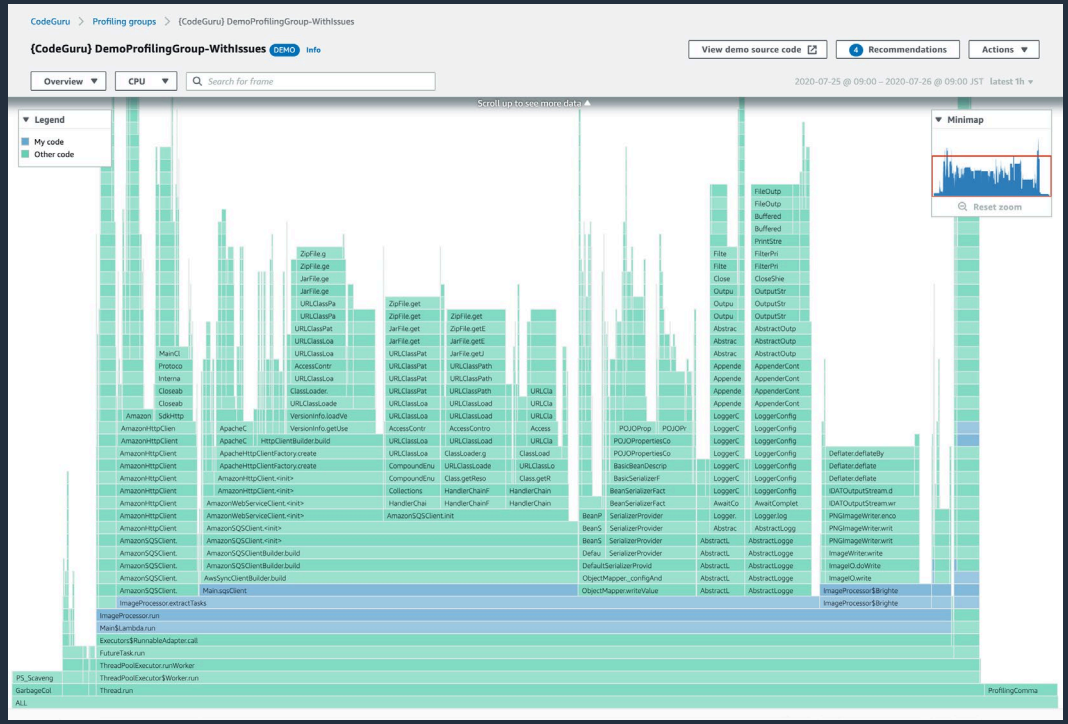
```
import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import software.amazon.codeguruprofilerjavaagent.LambdaProfiler;

public class MyHandler implements RequestHandler<Input, Output>{

    @Override
    public Output handleRequest(Input input, Context context) {
        return LambdaProfiler.profile(input, context, this::myHandlerFunction);
    }

    public Output myHandlerFunction(Input input, Context context) {
        // your function code here
    }
}
```

AWS Lambda が提供する RequestStreamHandler を使用している場合

```
package example;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import com.amazonaws.services.lambda.runtime.Context;

import software.amazon.codeguruprofilerjavaagent.RequestStreamHandlerWithProfiling;

public class StreamHandler extends RequestStreamHandlerWithProfiling {

    @Override
    public void requestHandler(InputStream input, OutputStream output, Context context)
throws IOException {
        // Your function code here
    }
}
```

CodeGuru Profiler のビジュアライゼーション

- 稼働中の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、どの code path が CPU時間を消費しているか把握するために役立つ (Flame Graph とも呼ぶ)
- 各フレーム (frame) 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
- 3種類のビューを提供: CPU, Latency, Custom



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

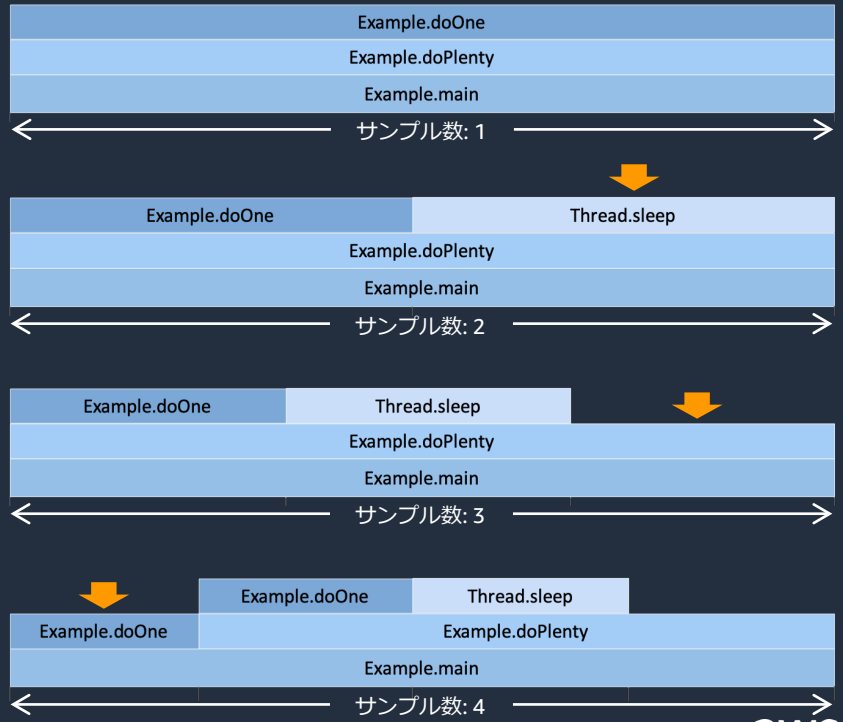
```
Thread main
   java.lang.Thread.State: TIMED_WAITING
     java.lang.Thread.sleep(long)
     com.amazon.profiler.demo.Example.doPlenty()
     com.amazon.profiler.demo.Example.main(String[])
```

サンプル3

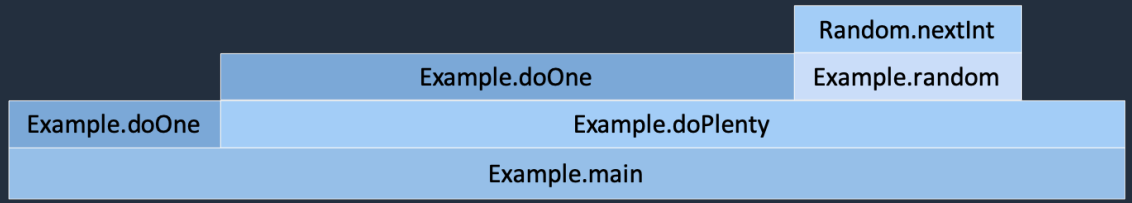
```
Thread main
   java.lang.Thread.State: RUNNABLE
     com.amazon.profiler.demo.Example.doPlenty()
     com.amazon.profiler.demo.Example.main(String[])
```

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から得られる情報



正しい情報

- ○ doOne 関数は main 及び doPlenty 関数で呼び出されている (doOne は main と doPlenty 両方のフレームの上に現れている)
- ○ doPlenty 関数の半分以上のCPU時間は doOne に消費されている (doOne の幅は doPlenty の幅の半分以上を占めている)
- ○ doPlenty 関数の中でもCPU時間を消費する処理が行われている (doPlenty の上には空白が存在する)

誤った情報

- ✕ main 関数の中で doOne 関数は doPlenty 関数よりも前に呼び出されている (フレームはアルファベット順で表示されており、関数の呼び出し順序とは無関係)
- ✕ doOne 関数は random 関数よりも多く呼び出されている (ビジュアライゼーションはCPU使用時間の割合を表すものであり、関数の呼び出し頻度とは無関係)
- ✕ doPlenty 関数は実行に XX 秒時間がかかっている (ビジュアライゼーションはCPU使用時間の割合を表すものであり、関数単体の実行時間は計測できない)

aws

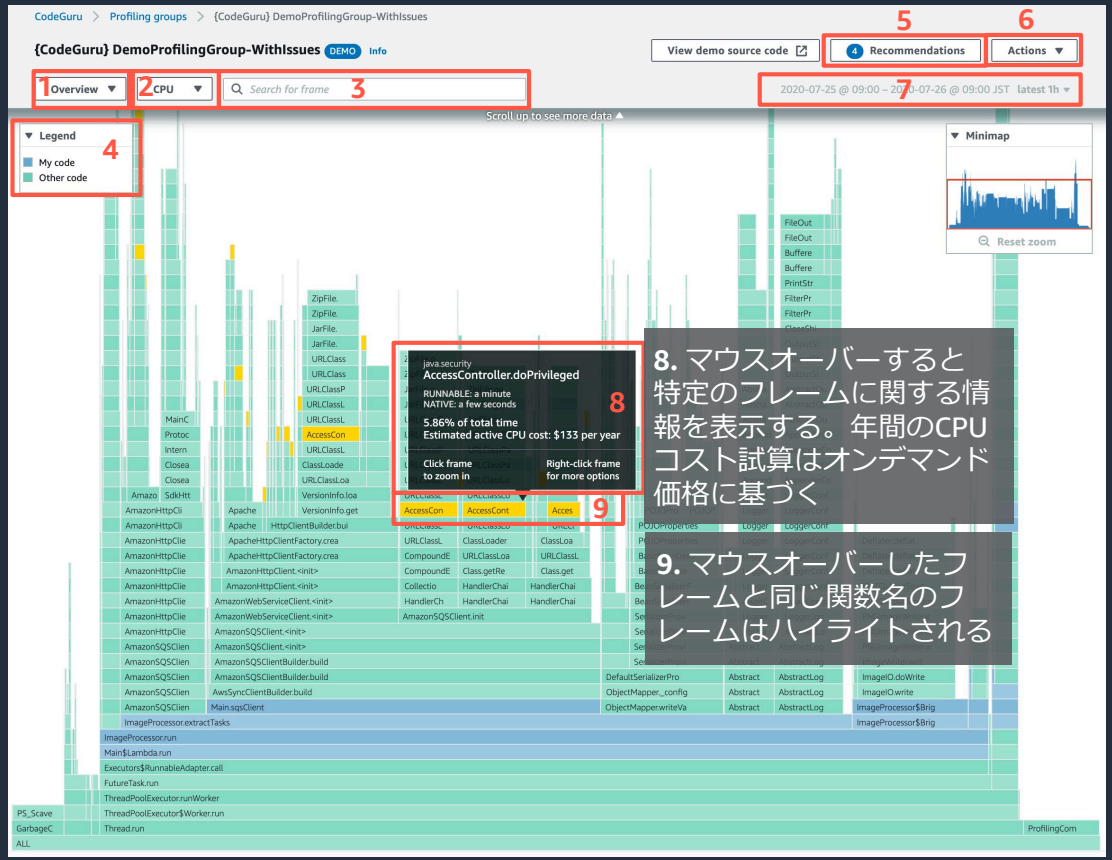
ビジュアライゼーション UI の解説 (Overview モード)

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)

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



5. CodeGuru Profiler によるレコメンデーションを表示

6. My code とみなすネームスペースの選択、すべてのレコメンデーションレポート表示などのアクション

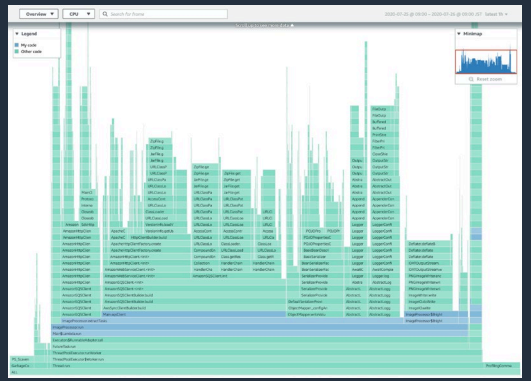
7. ビジュアライゼーションの時間範囲

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

CodeGuru Profiler 3種類のモード

1. Overview モード

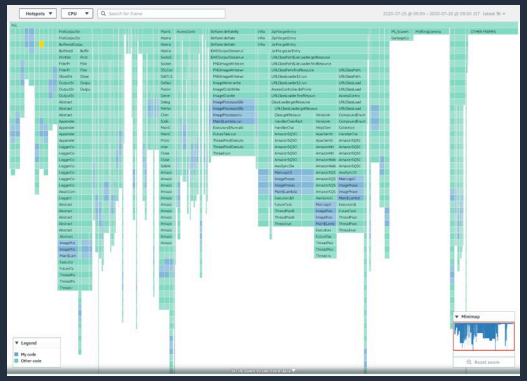


プロファイリングデータをボトムアップで表示

多くのIDEのスタックトレース同様に、最下位がエントリポイントとなる

アプリケーション全体を理解したり、各処理のCPU使用時間を把握するのに有用

2. Hotspots モード

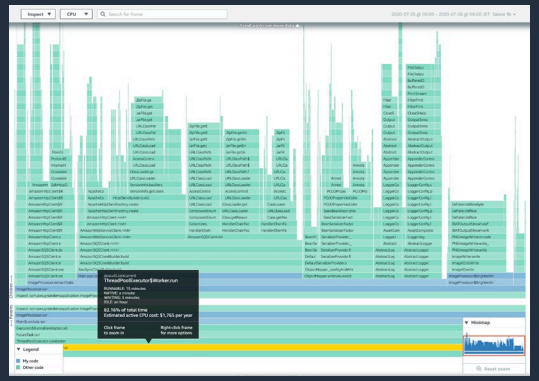


プロファイリングデータをトップダウンで表示

最もCPU使用時間が長い関数が上位に表示される

CPU使用時間が長い (=実行コストが高い) 処理を特定するのに有用。ただしCPU使用時間が長いことが悪いこととは限らない

3. Inspect モード

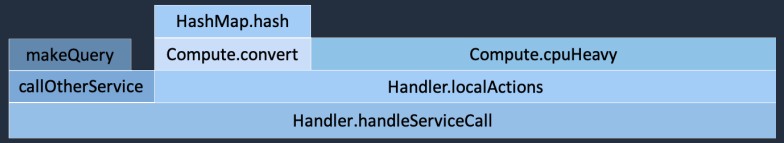


ビジュアライゼーション全体で複数箇所に存在するフレームを集約し表示

呼び出し元 (callers/親) は対象フレームの下部、呼び出し先 (callees/子) は対象フレームの上部に集約される

CodeGuru Profiler 3種類のビュー

1. CPU View

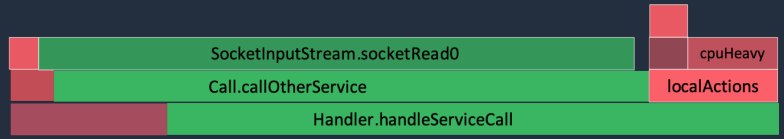


スレッドステートが 'RUNNABLE', 'BLOCKED' または 'NATIVE' のフレームを表示するデフォルトのビュー

アプリケーションのCPU使用率の解析に役立つ

色は視覚的にフレームを区別しやすくするため

2. Latency View



スレッドステートが 'IDLE' 以外のフレームを表示するビュー (つまり、 'BLOCKED', 'WAITING' や 'TIMED_WAITING' も含まれる)

アプリケーションのレイテンシー (実行時間) に影響を与えている処理の特定に役立つ

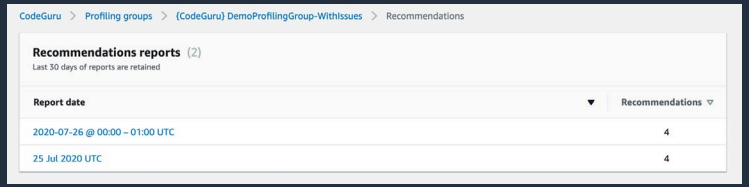
色は異なるスレッドステートを区別するため

3. Custom View

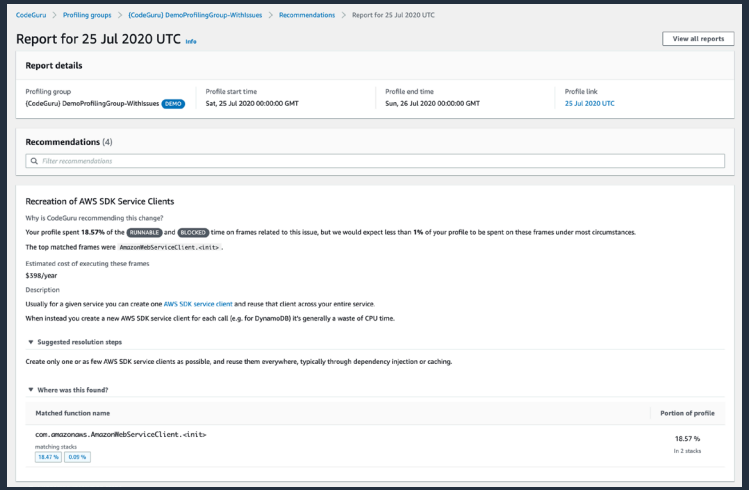
表示対象のスレッドステート及び色をカスタマイズできるビュー

CodeGuru Profiler レコメンデーションレポート

- 収集されたプロファイリングデータを元に1時間おきにレコメンデーションレポートを自動的に生成
- UIからは直近30日分のレポートを閲覧可能 (CLI/APIからは更に古いレポート/レコメンデーションを取得可能)
- レポートにはパフォーマンス改善に役立つレコメンデーションと (検知された場合) 異常事項が含まれる



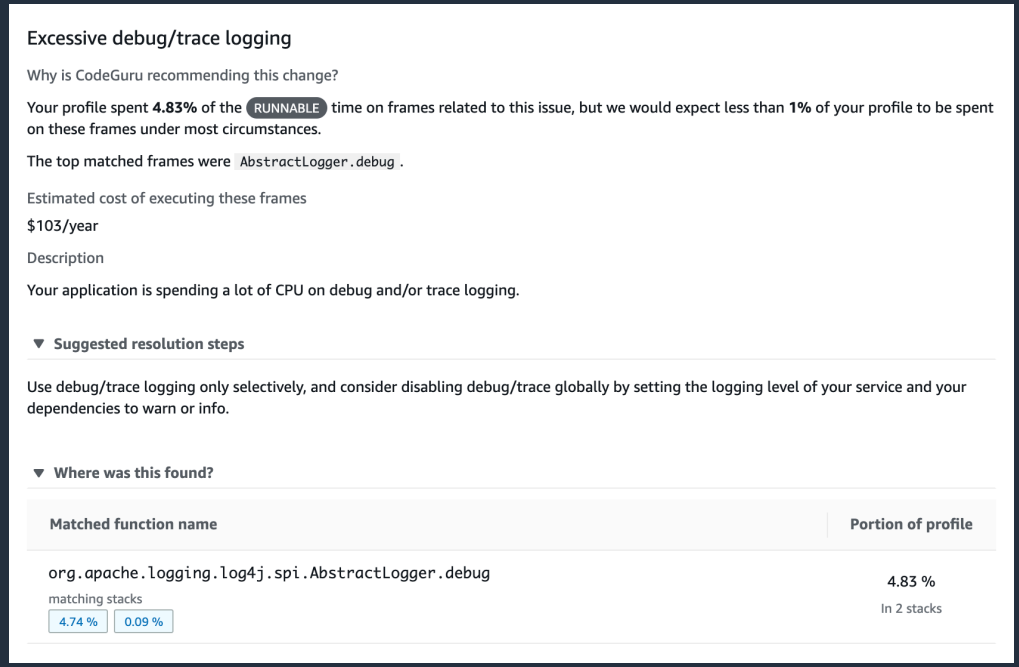
レコメンデーションレポート一覧



レコメンデーションレポートのサンプル

CodeGuru Profiler レコメンデーション

- レコメンデーションは非効率的な処理やライブラリの使用、不必要なオブジェクト/スレッドの再作成など、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
- レコメンデーション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レコメンデーション

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
- レコメンデーション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

例: とある関数の実測時間が異常に長かった場合

CodeGuru Profiler の特徴

- 既存のアプリケーションに対しても簡単にProfilerエージェントを起動することが可能
- アプリケーションデータを継続的に収集し、柔軟なビジュアライゼーションを提供
- プロファイリングデータから、アクションを含むパフォーマンスを向上するためのレコメンデーションを提供
- 過去のプロファイリングデータと乖離がある場合、機械学習をベースとした異常検知をレポート
- Profilerエージェントは最小限のフットプリントで、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

aws

本日のアジェンダ

- 背景
- Amazon CodeGuru とは
- Amazon CodeGuru Reviewer
- Amazon CodeGuru Profiler
- **セキュリティ**
- サービスクォータ
- 料金体系
- まとめ

Amazon CodeGuru セキュリティ

Amazon CodeGuru

- Amazon CodeGuru により収集されたデータは Amazon S3 及び Amazon DynamoDB (Profiler の場合Amazon Kinesisも含む) に保管されておりサーバーサイド暗号化が行われている
- データ転送には全てTLSを採用しており、全てのエンドポイントでは SHA-256 証明書が利用されている

Amazon CodeGuru Reviewer

- CodeGuru Reviewer はリポジトリメタデータ、レコメンデーション、Pull Request メタデータ、お客様フィードバックを保持するが、ソースコードは保持されない

Amazon CodeGuru Profiler

- CodeGuru Profilerはスタックトレースを定期的に収集し、CodeGuru Profiler バックエンドに送信する。CodeGuru Profiler エージェントはパラメーターの名前や値、変数の値などのアプリケーションデータへのアクセスはできない

https://docs.aws.amazon.com/codeguru/latest/reviewer-ug/security.html
https://docs.aws.amazon.com/codeguru/latest/profiler-ug/security.html

本日のアジェンダ

- 背景
- Amazon CodeGuru とは
- Amazon CodeGuru Reviewer
- Amazon CodeGuru Profiler
- セキュリティ
- **サービスクォータ**
- 料金体系
- まとめ

Amazon CodeGuru サービスクォータ

Amazon CodeGuru Reviewer

AWS CodeCommit リポジトリにおけるリージョンごとの月間最大Pull Request数: 5,000 (デフォルト)

Amazon CodeGuru Profiler

リージョンごとの最大プロファイリンググループ数: 50 (デフォルト)

https://docs.aws.amazon.com/codeguru/latest/reviewer-ug/quotas.html
https://docs.aws.amazon.com/codeguru/latest/profiler-ug/quotas.html

aws

本日のアジェンダ

- 背景
- Amazon CodeGuru とは
- Amazon CodeGuru Reviewer
- Amazon CodeGuru Profiler
- セキュリティ
- サービスクォータ
- **料金体系**
- まとめ

Amazon CodeGuru 料金体系

Amazon CodeGuru Reviewer

- **0.75USD/100行のコード**
  - Pull request 内に過去解析済みのファイルが含まれる場合、差分のみが課金対象となる
  - コメントやインポート文などの非コード行は課金対象外

Amazon CodeGuru Profiler

- **0.005USD/サンプリング時間**
  - プロファイリンググループあたり36,000 サンプリング時間が上限
  - AWS Lambda では Payer アカウントごとに500サンプリング時間が無償利用枠

継続的に評価いただくために90日間の無償期間を提供

2020年8月4日時点 東京リージョンの価格
https://aws.amazon.com/codeguru/pricing/

本日のアジェンダ

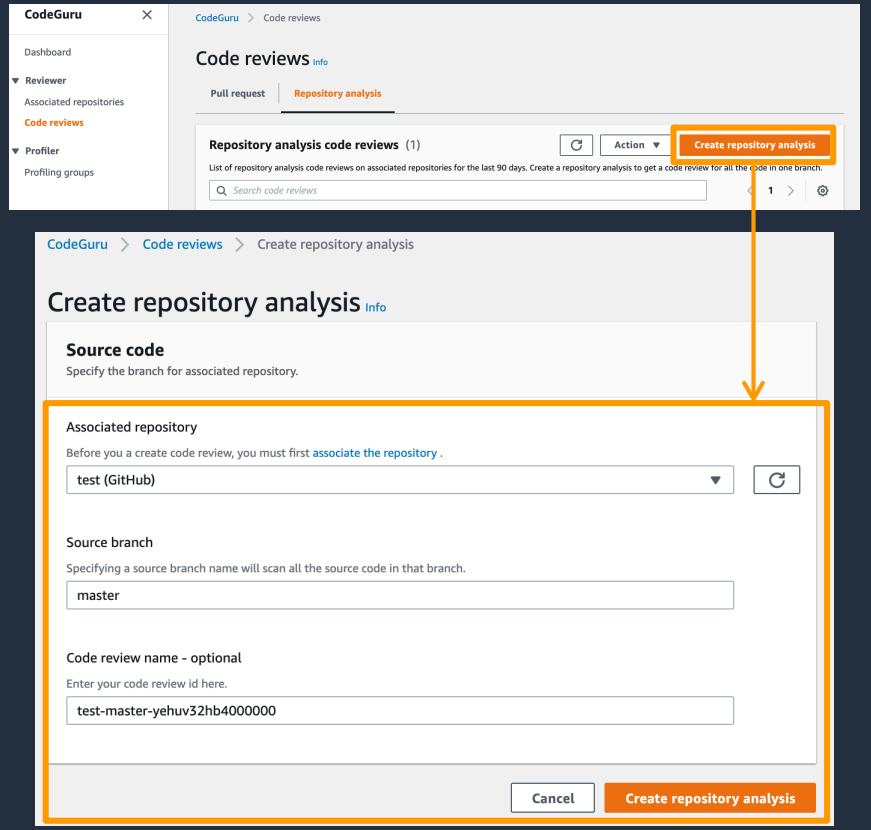
- 背景
- Amazon CodeGuru とは
- Amazon CodeGuru Reviewer
- Amazon CodeGuru Profiler
- セキュリティ
- サービスクォータ
- 料金体系
- **まとめ**

まとめ

- Amazon CodeGuru は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 (レコメンデーション) 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 は現状 AWS CodeCommit、GitHub、 GitHub Enterprise Cloud、GitHub Enterprise Server、Bitbucket に対応しており、Pull Request をトリガーに Java ソースコードに対しレコメンデーションを生成する
- Amazon CodeGuru Profiler は Profiler エージェントによって継続的に JVM ランタイムのデータを収集し、リッチなビジュアライゼーションを提供し、パフォーマンス改善に役立つレコメンデーションや異常検知をレポートする
- お客様のフィードバックを基にモデルを継続的に改善している
- 90日の無償期間を提供

(2020/8/5追記) CodeGuru Reviewer がリポジトリ解析に対応

- Pull Request ベースの解析に加え、リポジトリの特定ブランチに対するフルスキャンをサポート
- 開始方法
  - 対象のリポジトリを関連付ける
  - マネジメントコンソールから、Amazon CodeGuru > Code reviews にて Create repository analysis を選択 (CreateCodeReview API でも可能)
  - リポジトリを選択し、ブランチを指定する
  - レビュー結果はマネジメントコンソールの Code Reviewes 配下あるいはAPIで確認できる
- ユースケース
  - 既存のリポジトリに対し CodeGuru Reviewer を導入する際の評価用途
  - 定期的にフルスキャンを実施し、CodeGuru Reviewer で新たに追加された検知領域に対するレコメンデーションがないかの確認
- 料金体系
  - 本資料の「料金体系」で紹介している Pull Request ベースの料金体系とは異なる
  - 毎月最初の1,500,000行に対して 0.50USD/100行のコード
  - 毎月1,500,000行を超えた分に対し 0.40USD/100行のコード



Q&A

お答えできなかったご質問については
AWS Japan Blog 「https://aws.amazon.com/jp/blogs/news/」にて
後日掲載します。

AWS の日本語資料の場所「AWS 資料」で検索

https://amzn.to/JPArchive

AWS Well-Architected 個別技術相談会

毎週”W-A個別技術相談会”を実施中

- AWSのソリューションアーキテクト(SA)に対策などを相談することも可能
- 申込みはイベント告知サイトから

(https://aws.amazon.com/jp/about-aws/events/)

AWS イベント で[検索]

ご視聴ありがとうございました

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive