aws DEV DAY
ONLINE JAPAN

20.10.2020

DEV DAY

In Partnership with intel

aws DEV DAY
ONLINE JAPAN

20.10.2020

B - 3

やっぱり Java が好き：
～ AWSでJavaを使うなら押さえておきたいツール特選 ～

松原 武司　Matsubara, Takeshi

シニア 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

In Partnership with intel

自己紹介

- 名前
  - 松原 武司
- 所属
  - アマゾン ウェブ サービス ジャパン株式会社
  - 金融ソリューション本部
  - シニア ソリューション アーキテクト

- 好きなAWSサービス
  - Amazon API Gateway　AWS Lambda

このセッションのご参加者は…

Java Developers

In Partnership with intel

さて今月の TIOBE index は…

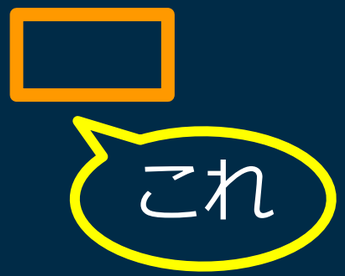
[引用元] https://www.tiobe.com/tiobe-index/

さて今月の TIOBE index は…

[引用元] https://www.tiobe.com/tiobe-index/

In Partnership with intel

しかしこれまでのトレンドも考えると…



[引用元] https://www.tiobe.com/tiobe-index/

In Partnership with

しかしこれまでのトレンドも考えると…

Java

[引用元] https://www.tiobe.com/tiobe-index/

In Partnership with intel

しかしこれまでのトレンドも考えると…

長らくほぼ首位

[引用元] https://www.tiobe.com/tiobe-index/

In Partnership with intel

つまり…

とても大勢

Java Developers

aws

In Partnership with intel

I choose the form of *Java!*

In Partnership with intel

本セッションのテーマ：言語目線で見てみましょう

Javaという切り口

Java Developers

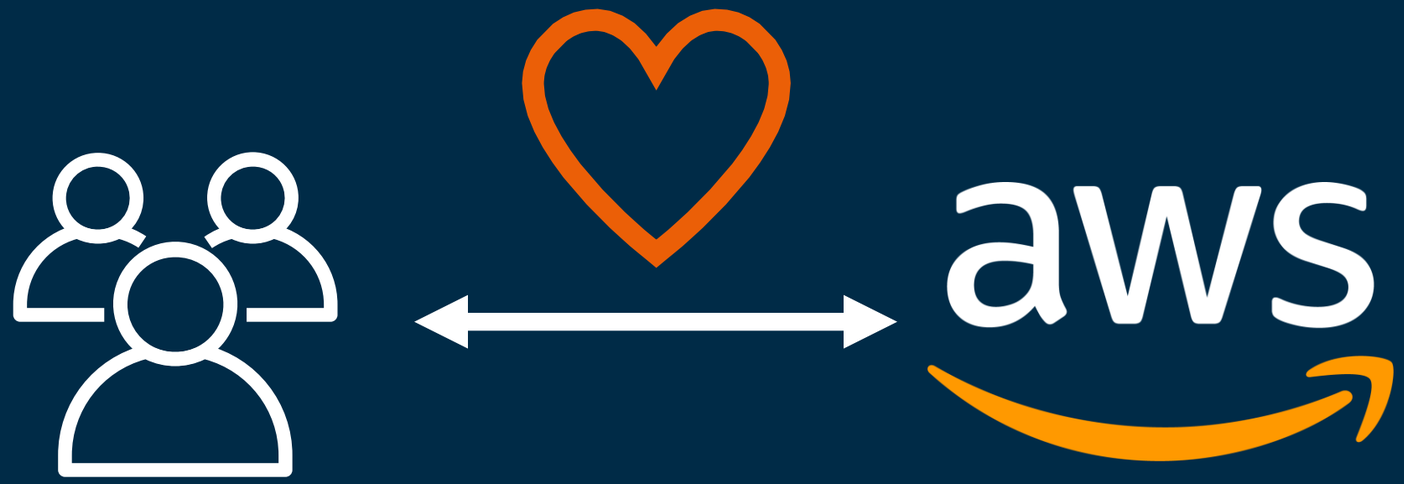
In Partnership with intel

このセッションの目的

Java Developers

In Partnership with intel

このセッションの目的



Java Developers

In Partnership with intel

目線を2つに分けます

AWSに

作る

Javaで

AWSを

操る

In Partnership with intel

目線を2つに分けます

AWSに

AWSを

Javaで

作る

操る

In Partnership with intel

目線を2つに分けます

AWSに

AWSを

Javaで

作る

操る

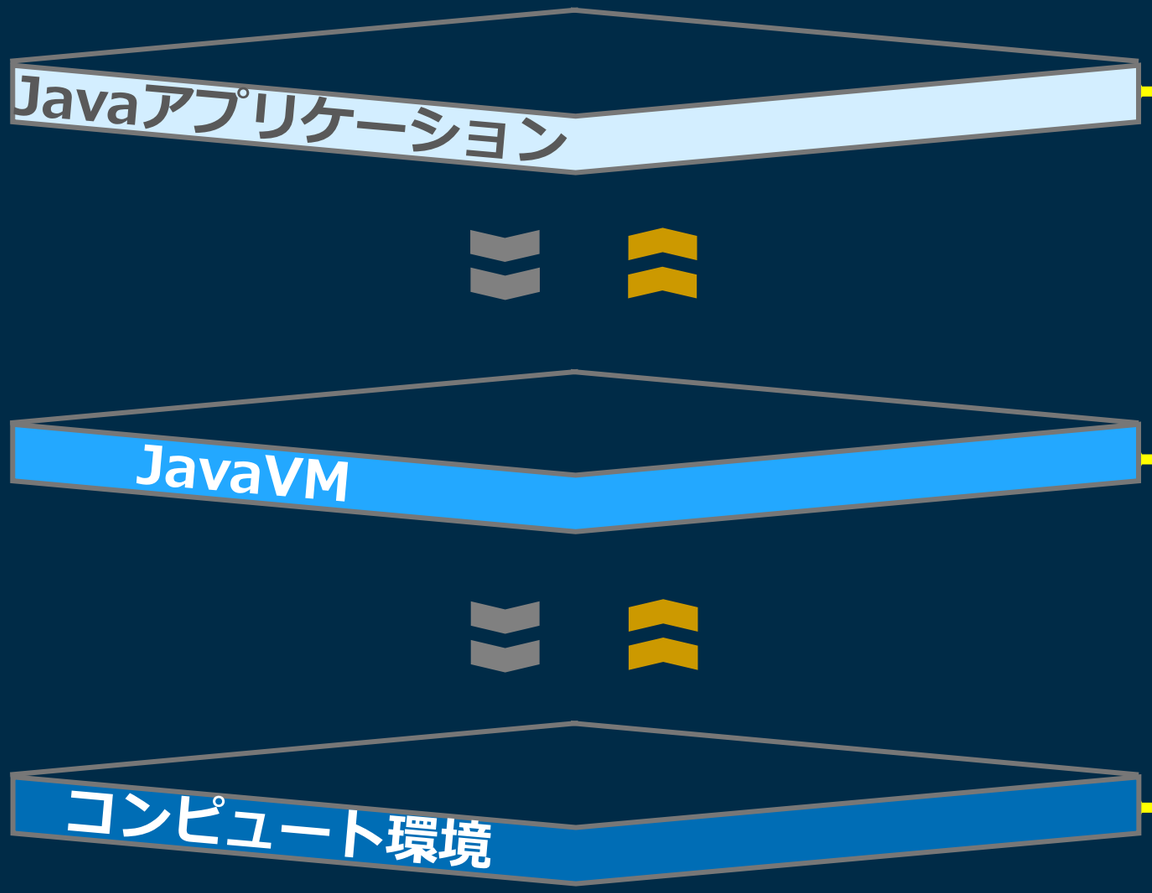
In Partnership with intel

aws DEV DAY
ONLINE JAPAN

AWSに Javaで 作る

In Partnership with intel

3つの階層で考えてみましょう



App with AWS Tools

Java仮想マシン

コンピューティング
ワークロードの実行環境

In Partnership with intel

「AWSに」「Javaで」「つくる」

- **AWS Tools**
- **AWS X-Ray Java Agent**

- **Amazon Corretto**
- **Other JVMs** (OpenJDK, etc…)

※AWS Elastic Beanstalkもありますが本セッションでは省略

- **Amazon EC2** (Elastic Compute Cloud)
- **AWS Lambda**
- **Amazon ECS** (Elastic Container Service)
- **Amazon EKS** (Elastic Kubernetes Service)

In Partnership with intel

コンピューティングレイヤー

- **Amazon EC2** (Elastic Compute Cloud)
- **AWS Lambda**
- **Amazon ECS** (Elastic Container Service)
- **Amazon EKS** (Elastic Kubernetes Service)

In Partnership with intel

コンピューティングレイヤー

- 仮想サーバー(OS)
- コンテナ

In Partnership with intel

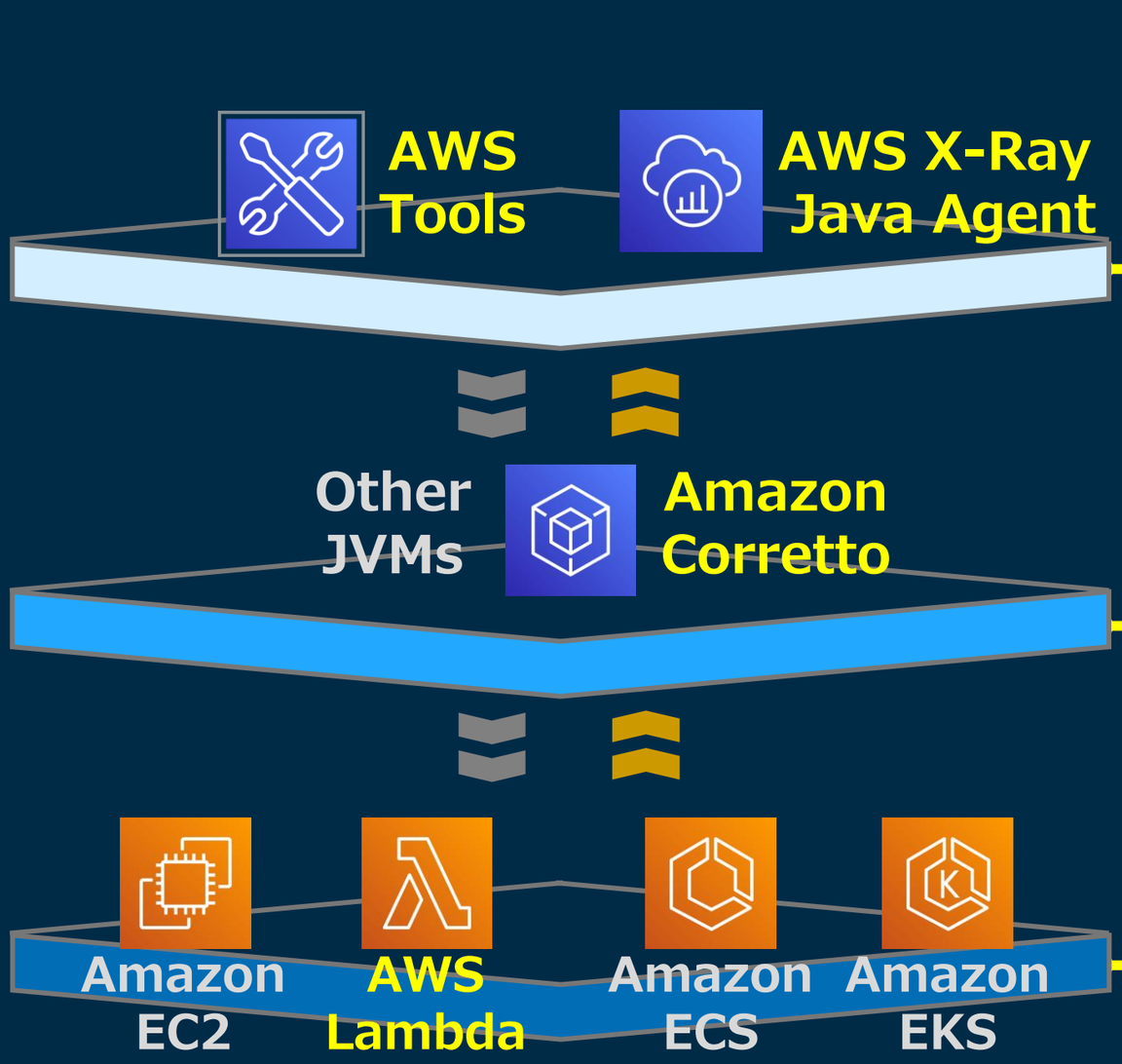
コンピューティングレイヤー

- Java EE(Jakarta EE)アプリケーションサーバーその他を含めて大きく変わるところなし
  - 商用製品 / OSS 利用
  - エンタープライズ開発では“サポート”の有無も念頭に(技術問い合わせ可否含む)

- 仮想サーバー(OS)
- コンテナ

In Partnership with intel

Java目線でピックアップ



- **AWS Tools**
- **AWS X-Ray Java Agent**

- **Amazon Corretto**
- **Other JVMs** (OpenJDK, etc…)

- **Amazon EC2** (Elastic Compute Cloud)
- **AWS Lambda (Java Runtime)**
- **Amazon ECS** (Elastic Container Service)
- **Amazon EKS** (Elastic Kubernetes Service)

In Partnership with intel

押さえておきたいその1: Amazon Corretto

- AWS Tools
- AWS X-Ray Java Agent

- **Amazon Corretto**
- **Other JVMs** (OpenJDK, etc…)

- **Amazon EC2** (Elastic Compute Cloud)
- **AWS Lambda**
- **Amazon ECS** (Elastic Container Service)
- **Amazon EKS** (Elastic Kubernetes Service)

In Partnership with intel

押さえておきたいその1: Amazon Corretto

AWS が提供する、AWS・オンプレミス・PC いずれでも利用可能な
無償のOpenJDK ディストリビューション (ライセンス : GPLv2 with ClassPath Exception)

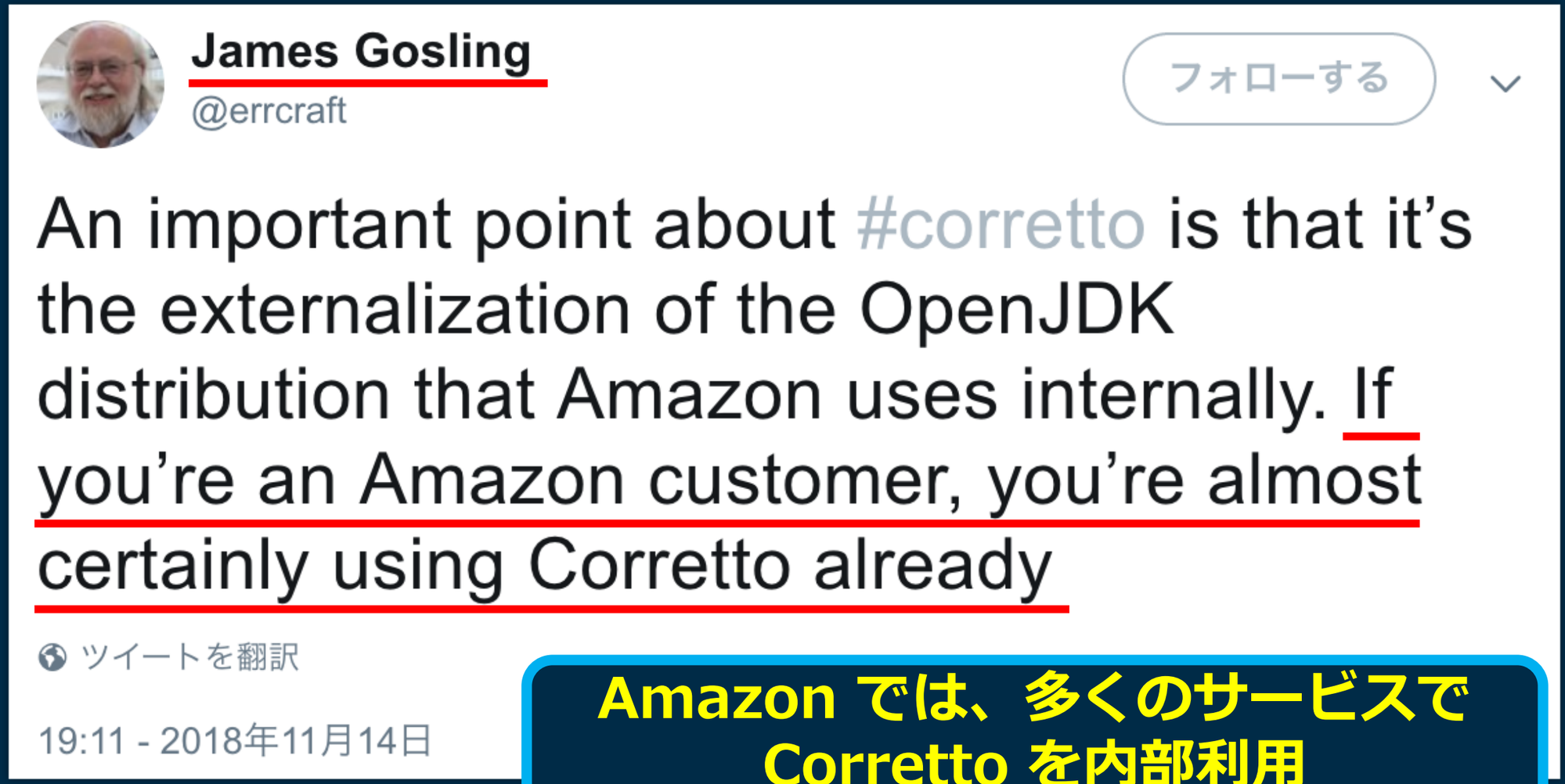
Amazon Corretto = **OpenJDK** + **Amazon提供のパッチ群**

+ **Long-term support (LTS)**

- **Corretto 8** … **OpenJDK 8ベース (一般利用開始 2019/1/31)**
  LTS : 2026年5月まで
- **Corretto 11** … **OpenJDK 11ベース (一般利用開始 2019/5/15)**
  LTS : 2027年9月まで
- **Corretto 15** … **OpenJDK 15ベース (一般利用開始 2020/9/24)** **New!**

In Partnership with intel

2018年 11月 のとあるTweet



Amazon では、多くのサービスで Corretto を内部利用

In Partnership with intel

Javaポリシー変更

※資料引用元：
https://cio.go.jp/sites/default/files/uploads/documents/1006_java_report.pdf

政府CIOポータル 6月 “Javaのサポートポリシー変更等に関する技術レポート”の改訂公開→Java利用システムのクラウド化を見据えた記述

Java のサポートポリシー変更等に関する技術レポート

Oracle JDK/JREのサポートポリシー変更を受け、政府内のJava利用システムに対する方向性をとりまとめたもの

2020 年（令和 2 年）年 6 月 24 日

内閣官房情報通信技術（ＩＴ）総合戦略室

〔標準ガイドライン群ＩＤ〕
1006

〔キーワード〕
Java、JDK、JRE、Java アプレット、ライセンス、ライフサイクルの短期間

1) 現行の情報システムがサーバー環境向け Java を採用している場合

(1) Java9 以降バージョンに移行する場合は、32 ビット版から 64 ビット版への移行を行う必要があります。

(2) サーバー環境向け Java のサポート対応（情報セキュリティ対策を含めたアップデートの提供等）を受けるためには、大きく分けて以下の 5 つの方法から選択する必要があります。

ｱ) サーバー環境向け Java を OpenJDK に移行した上で、半年サイクルにて提供される OpenJDK の最新バージョンに常にバージョンアップします。このサイクルを適用した場合には、無償でサポートが受けられます。

ｲ) サードパーティによる無償提供の Java に移行した上で、無償サポート（セキュリティパッチの提供、各種レポートの提供）を利用します。具体例としては、AdoptOpenJDK は、Java8 を独自に 2026 年 5 月まで無償サポートする予定です。

ｳ) クラウド事業者が提供する Java に移行した上で、クラウドサービス利用を前提としたサポートを利用します（具体例としては、AWS が提供する JavaVM Amazon Corretto、マイクロソフト Azure での Azul Systems JavaVM）。

In Partnership with intel

その2: AWS Lambda & Java

- AWS Tools
- AWS X-Ray Java Agent

- Amazon Corretto
- Other JVMs (OpenJDK, etc...)

- Amazon EC2 (Elastic Compute Cloud)
- AWS Lambda
- Amazon ECS (Elastic Container Service)
- Amazon EKS (Elastic Kubernetes Service)

In Partnership with intel

その2: AWS Lambda & Java

サーバーレスでのイベントベースの関数実行サービス

In Partnership with intel

その2: AWS Lambda & Java

ユーザープログラムを始め、さまざまなAWSサービスから呼び出し可能

ユーザープログラム

AWS SDK

同期 or 非同期呼び出し

AWSサービス(イベント送信型)

Amazon API Gateway ... ... **Amazon S3**

同期 or 非同期呼び出し

AWSサービス(ポーリング型)

Amazon DynamoDB **Amazon Kinesis** **Amazon Simple Queue Service** (SQS)

非同期呼び出し
(Lambdaサービスがポーリング)

AWS リージョン

AWS Lambdaサービス

Lambda関数

In Partnership with intel

その2: AWS Lambda & Java

Lambda関数の開発言語としてJavaに対応
Javaランタイムとして OpenJDK 8, Amazon Corretto 8 & 11 に対応

Lambda関数

言語を選択して、関数を実装

#	対応言語	サポートランタイムVer
1	Node.js (JavaScript)	10.x / 12.x
2	**Java**	**8 (OpenJDK) / 8 (Corretto) 11 (Corretto)**
4	Python	2.7(deprecated) / 3.6 / 3.7 / 3.8
5	Ruby	2.5 / 2.7
6	Go	1.x
7	.NET Core (C#/PowerShell)	2.1 / 3.1
8	カスタムランタイム	(お客様作成のランタイムを利用)

例:JavaでのLambda関数プログラム

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

public class Hello implements RequestHandler<Integer, String>{

    public String myHandler(int myCount, Context context) {
        return String.valueOf(myCount);
    }

}
```

Javaの場合、AWS SDKライブラリが提供するRequestHandlerまたはそのサブインターフェースを実装

In Partnership with intel

AWS Lambda & Java といえば…

Lambda関数におけるライフサイクルの “Cold Start” への配慮と対策
→ インフラ的対処法もあるが、開発者としても「目配り」を

関数(your code)のダウンロード

新規コンテナ(実行環境)の生成と開始

言語ランタイムの起動とアプリ初期化

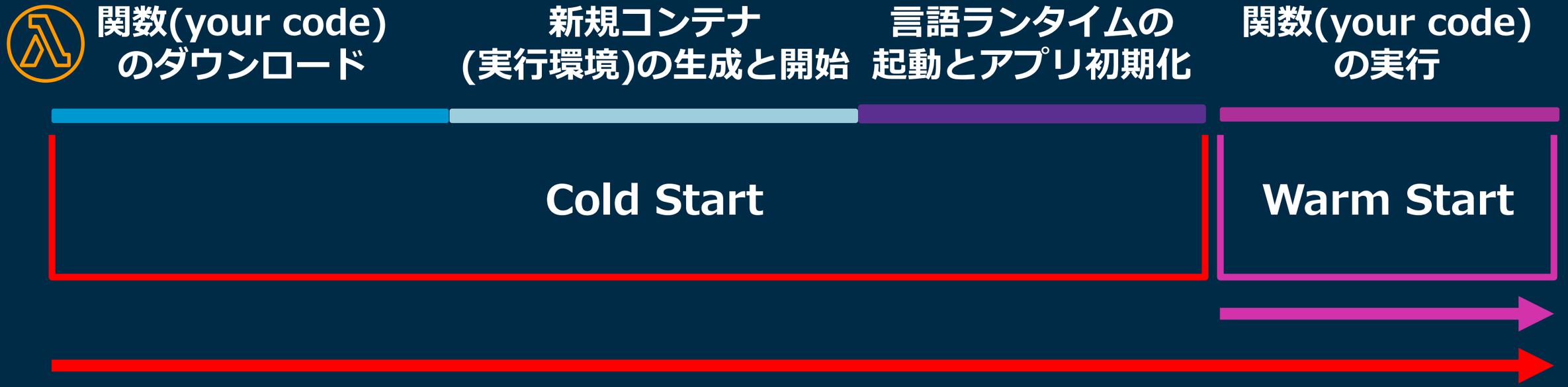
関数(your code)の実行

Cold Start

In Partnership with intel

AWS Lambda & Java といえば…

Lambda関数におけるライフサイクルの “Cold Start” への配慮と対策
→ インフラ的対処法もあるが、開発者としても「目配り」を



In Partnership with intel

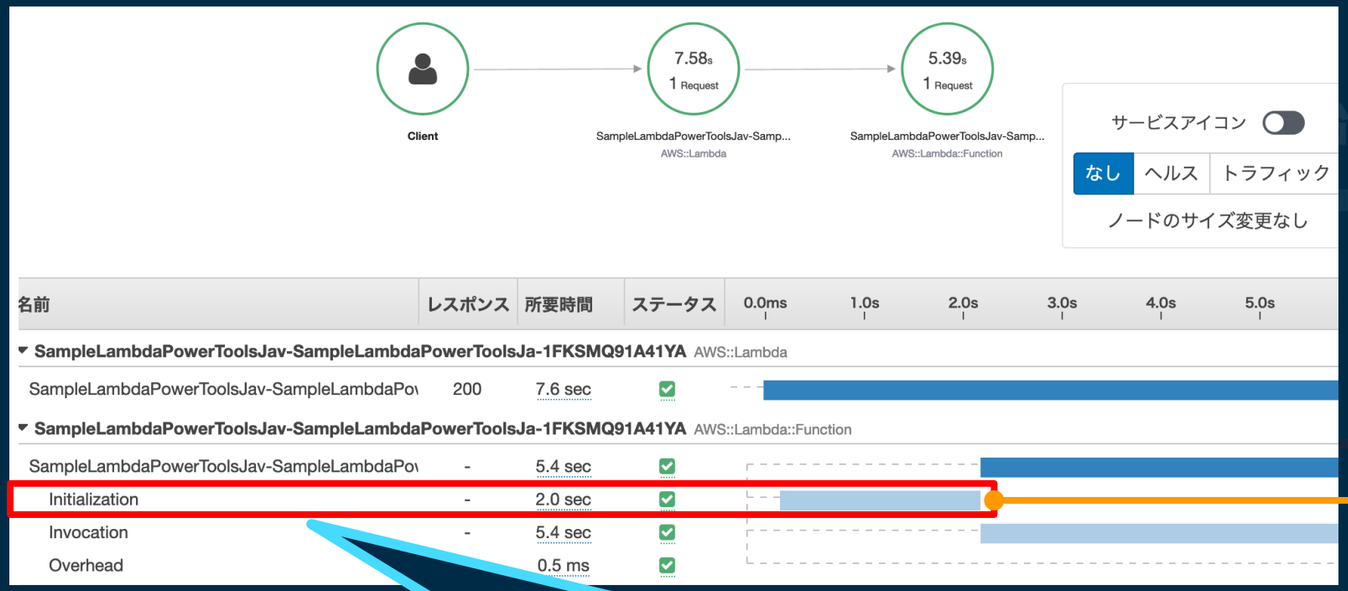
AWS Lambda & Java といえば…

Lambda関数におけるライフサイクルの “Cold Start” への配慮と対策
→ インフラ的対処法もあるが、開発者としても「目配り」を

In Partnership with intel

AWS Lambda & Java といえば…

開発者としてCold Startの時間をより短くするには、
「言語ランタイムの起動と初期化」の時間短縮が鍵



言語ランタイムの起動と初期化

関数(your code)の実行

X-Rayトレースで”Initialization”が表示されている=Cold Start発生を意味

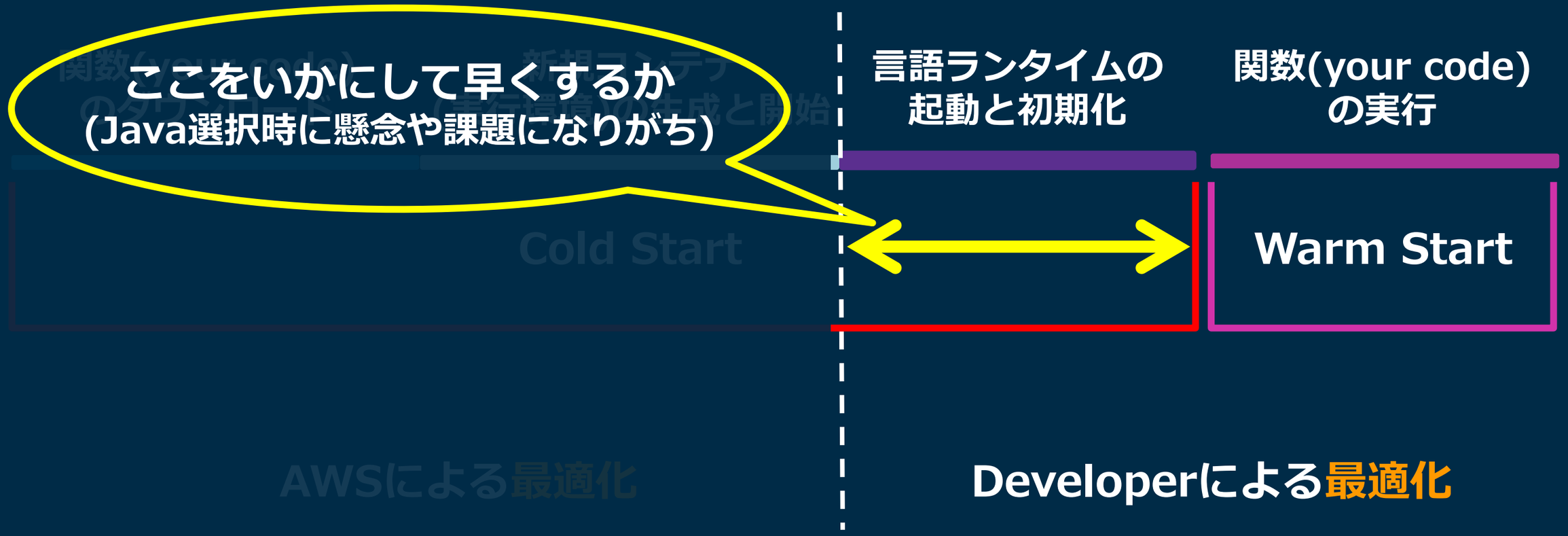
Developerによる最適化

aws

In Partnership with intel

AWS Lambda & Java といえば…

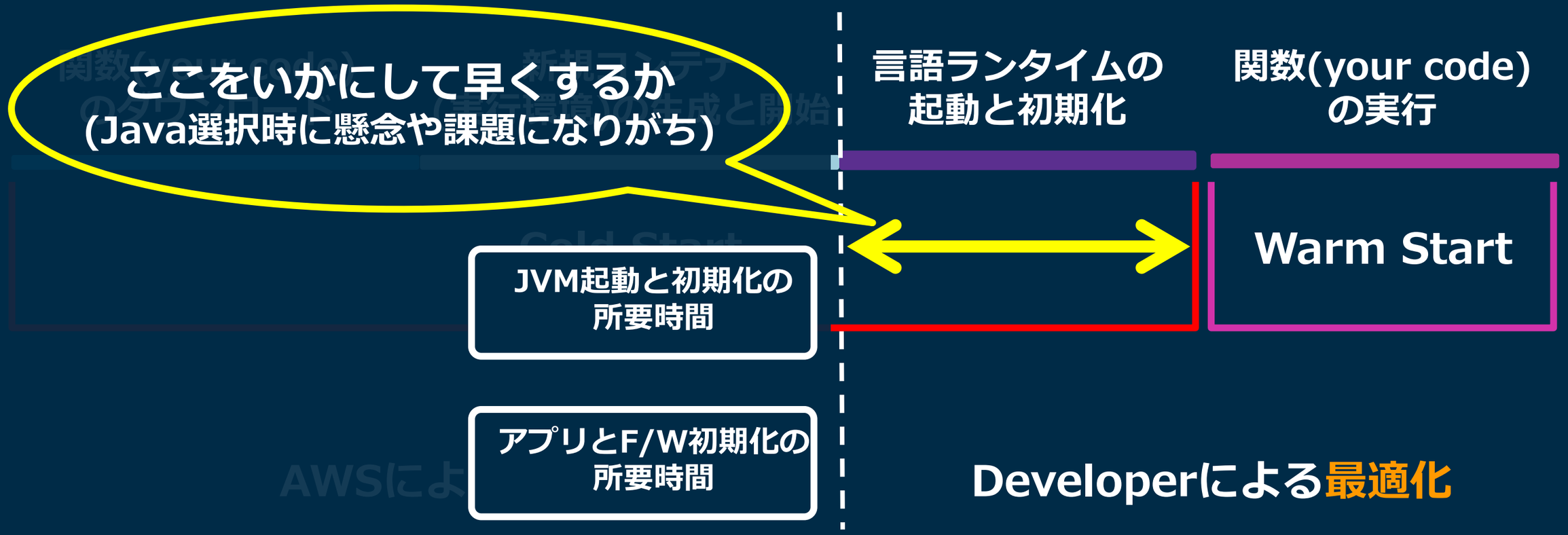
開発者としてCold Startの時間をより短くするには、
「言語ランタイムの起動と初期化」の時間短縮が鍵



In Partnership with intel

AWS Lambda & Java といえば…

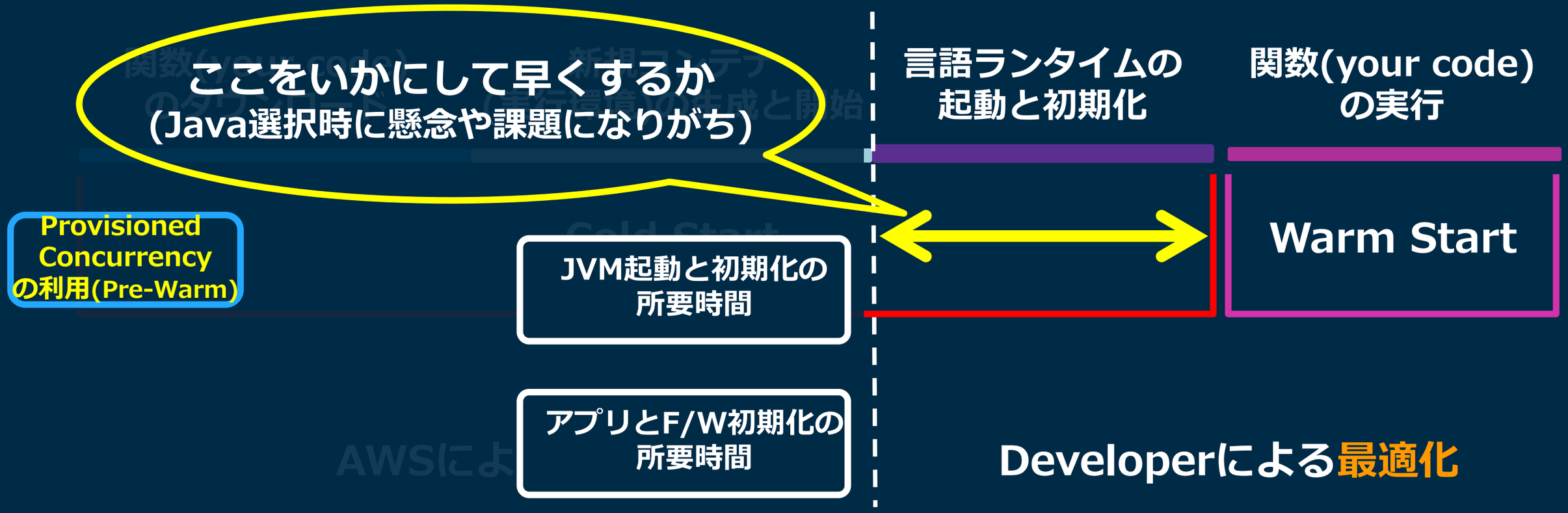
「言語ランタイムの起動と初期化」は大別すると以下の2つに依存
「JVM起動の所要時間」と「アプリとフレームワーク初期化の所要時間」



In Partnership with intel

AWS Lambda & Java といえば…

Lambdaサービス の Provisioned Concurrency機能 の利用
= 利用料金の発生を前提に、コールドスタートの発生自体を減らす



In Partnership with intel

AWS Lambda & Java といえば…

クラスロード時の高コストな処理の発生をできるだけ抑制する

ここをいかにして早くするか
(Java選択時に懸念や課題になりがち)

Provisioned Concurrency の利用(Pre-Warm)

ライブラリ見直し
(不要クラス群のロードを削減)

クラスロード時の処理削減
(特にCDIの挙動は注意)

JVM起動と初期化の所要時間

アプリとF/W初期化の所要時間

言語ランタイムの起動と初期化

関数(your code)の実行

Warm Start

Developerによる最適化

In Partnership with intel

AWS Lambda & Java といえば…

軽量なフレームワークやライブラリを用いる

ここをいかにして早くするか
(Java選択時に懸念や課題になりがち)

Provisioned Concurrency の利用(Pre-Warm)

ライブラリ見直し (不要クラス群のロードを削減)

クラスロード時の処理削減 (特にCDIの挙動は注意)

軽量Framework/ライブラリの利用

JVM起動と初期化の所要時間

アプリとF/W初期化の所要時間

言語ランタイムの起動と初期化

関数(your code)の実行

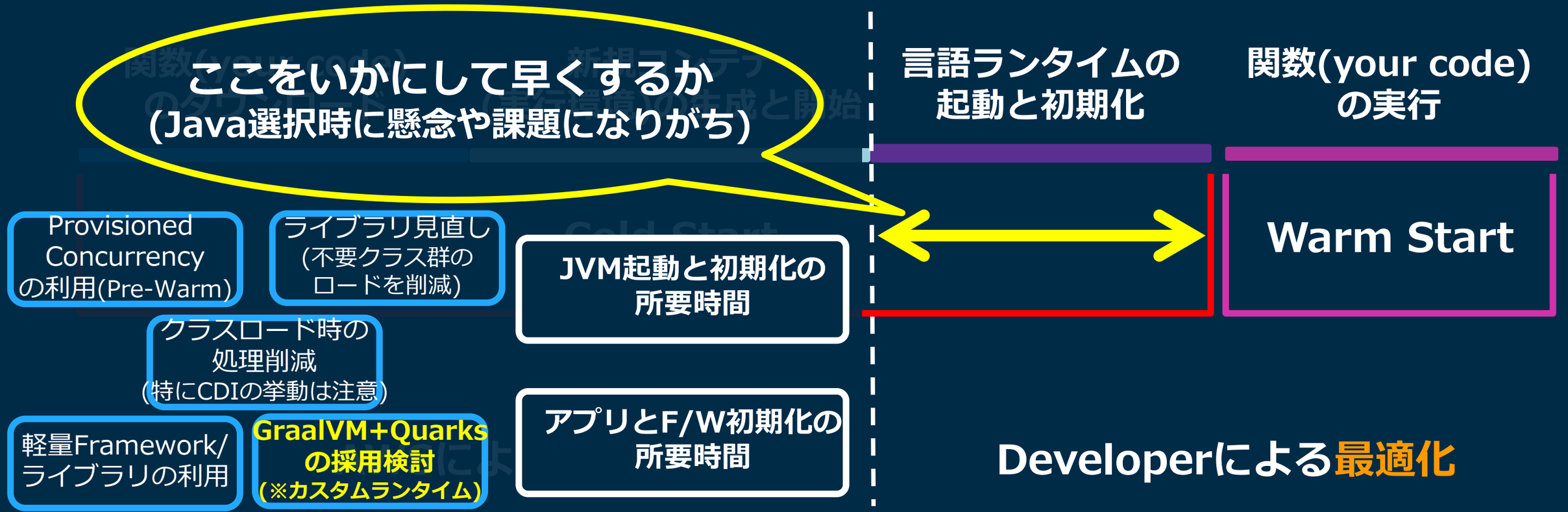
Warm Start

Developerによる最適化

In Partnership with intel

AWS Lambda & Java といえば…

AWS提供の言語ランタイム(OpenJDK8/Corretto8,11)ではなく、
カスタムランタイムの利用も候補になる場合はさらなる選択肢も



In Partnership with intel

AWS Lambda & Java といえば…

つまりは「サーバーサイドJavaアプリ開発」の感覚で考えない
(安易に開発標準や利用フレームワークなどまで ”同じ” にしない)

ここをいかにして早くするか
(Java選択時に懸念や課題になりがち)

言語ランタイムの起動と初期化

関数(your code)の実行

Warm Start

Provisioned Concurrency の利用(Pre-Warm)

ライブラリ見直し (不要クラス群のロードを削減)

クラスロード時の処理削減 (特にCDIの挙動は注意)

軽量Framework/ライブラリの利用

GraalVM+Quarks の採用検討 (※カスタムランタイム)

JVM起動と初期化の所要時間

アプリとF/W初期化の所要時間

Developerによる最適化

In Partnership with intel

その3: Lambda Power Tools for Java (beta)

AWSが提供する「Lambda関数の支援用OSSライブラリ」(License:ASL2)
(Python版に加えて登場)　**8月31日 Preview!**

https://github.com/awslabs/aws-lambda-powertools-java

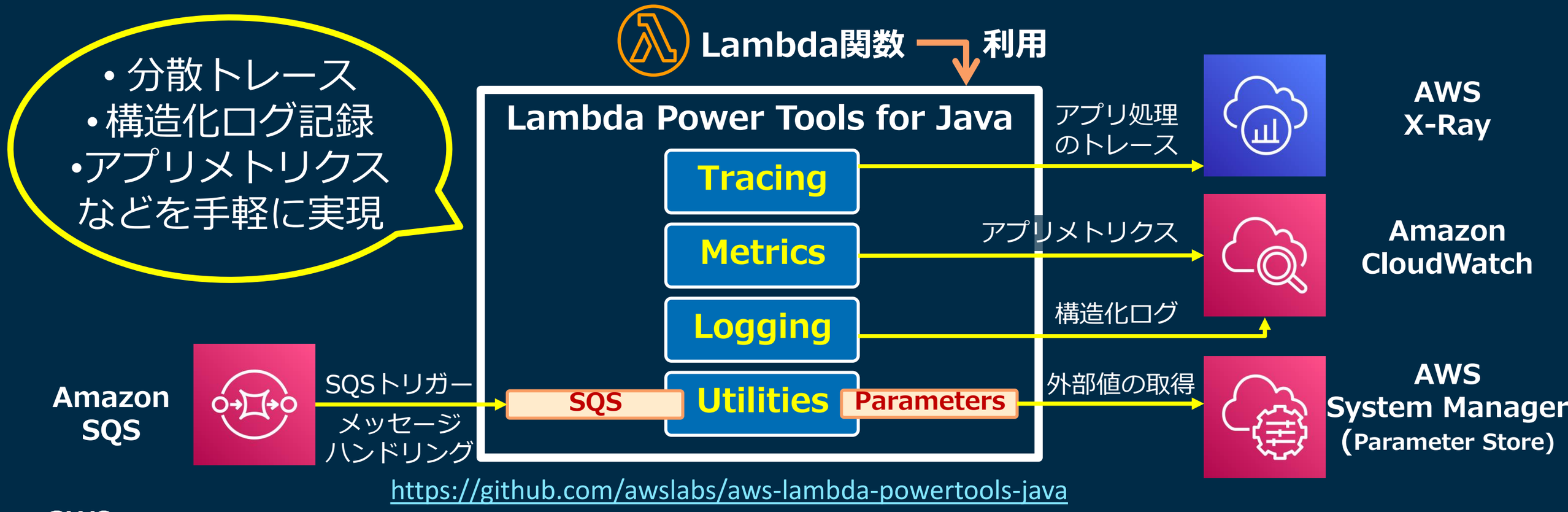
In Partnership with intel

その3: Lambda Power Tools for Java (beta)

AWSが提供する「Lambda関数の支援用OSSライブラリ」(License:ASL2)
(Python版に加えて登場)　**8月31日 Preview!**

- 分散トレース
- 構造化ログ記録
- アプリメトリクス

などを手軽に実現



https://github.com/awslabs/aws-lambda-powertools-java

In Partnership with intel

その3: Lambda Power Tools for Java (beta)

Maven Centralリポジトリから入手 (前提:Java8+)

pom.xml (Maven利用時)

```
<dependencies>
    ...
    <dependency>
        <groupId>software.amazon.lambda</groupId>
        <artifactId>powertools-tracing</artifactId>
        <version>0.5.0-beta</version>
    </dependency>
    <dependency>
        <groupId>software.amazon.lambda</groupId>
        <artifactId>powertools-logging</artifactId>
        <version>0.5.0-beta</version>
    </dependency>
    <dependency>
        <groupId>software.amazon.lambda</groupId>
        <artifactId>powertools-metrics</artifactId>
        <version>0.5.0-beta</version>
    </dependency>
    ...
</dependencies>
```

Tracing

- X-Rayトレース出力の拡張
  - メソッドアノテーションのみでトレースを出力

Logging

- Apaache Log4j2の拡張
  - LambdaJsonLayoutを利用し、構造化ロギングを行う

Metrics

- CloudWatchにカスタムメトリクス(アプリのメトリクス)を送信

In Partnership with intel

その3: Lambda Power Tools for Java (beta)

aspectj-maven-pluginを利用したコンパイル時のAspect定義を実施
(weavingで機能を実現しているため)

pom.xml 抜粋(1/2)

```
  ...
<build>
 <plugins>
     ...
     <plugin>
       <groupId>org.codehaus.mojo</groupId>
         <artifactId>aspectj-maven-plugin</artifactId>
         <version>1.11</version>
         <configuration>
           <source>1.8</source>
           <target>1.8</target>
           <complianceLevel>1.8</complianceLevel>
           <aspectLibraries>
              <aspectLibrary>
                <groupId>software.amazon.lambda</groupId>
                <artifactId>powertools-tracing</artifactId>
              </aspectLibrary>
              <aspectLibrary>
                <groupId>software.amazon.lambda</groupId>
                <artifactId>powertools-logging</artifactId>
              </aspectLibrary>
              <aspectLibrary>
                <groupId>software.amazon.lambda</groupId>
                <artifactId>powertools-metrics</artifactId>
              </aspectLibrary>
           </aspectLibraries>
```

pom.xml 抜粋(2/2)

```
          ...(cont'd)...

          </configuration>
          <executions>
            <execution>
              <goals>
                <goal>compile</goal>
              </goals>
            </execution>
          </executions>
      </plugin>
        ...
  </plugins>
</build>
...
```

※OpenJDK 9以降(Corretto11以降)の
利用時は tools.jar が非同梱になったことの対応が必要
(こちらを参照)

In Partnership with intel

その3: Lambda Power Tools for Java (beta)

メソッドトレース / メソッド呼び出しの自動ログ & アプリログの自動的な構造化 などを容易に実現 (より良い開発体験)

Lambda関数ハンドラ実装クラス

```
package com.myorg;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;
import software.amazon.lambda.powertools.tracing.PowertoolsTracing;
import software.amazon.lambda.powertools.logging.PowertoolsLogger;
import software.amazon.lambda.powertools.logging.PowertoolsLogging;
...
public class App implements RequestHandler<APIGatewayProxyRequestEvent, APIGatewayProxyResponseEvent> {
        Logger log = LogManager.getLogger();

        @PowertoolsTracing(captureError = true, captureResponse = true)    //デフォルトはtrue(両方)
        @PowertoolsLogging(logEvent = true, samplingRate = 1.0)            //デフォルトはfalse
        @PowertoolsMetrics(namespace = "ServerlessAirline", service = "payment", captureColdStart = true)
        public APIGatewayProxyResponseEvent handleRequest(final APIGatewayProxyRequestEvent input, final Context context)
                PowertoolsLogger.appendKey(“test”, “willBeLogged”);       //項目を追加する場合
                log.info(“sample app log message”);                       //アプリケーションとしてのログ出力
                final String pageContents = this.getPageContents("https://checkip.amazonaws.com"); //別メソッド呼び出し
                ...
        }

        @PowertoolsTracing(captureError = true, captureResponse = true)
        private String getPageContents(String address) throws IOException { ... }
}
```

その3: Lambda Power Tools for Java (beta)

Logging機能：Log4j2の拡張実装として提供されている
→ Log4j2ロガーの利用と設定が前提

log4j2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration packages="com.amazonaws.services.lambda.runtime.log4j2">
    <Appenders>
        <Console name="JsonAppender" target="SYSTEM_OUT">
            <LambdaJsonLayout compact="true" eventEol="true"/>
        </Console>
    </Appenders>
    <Loggers>
        <Logger name="JsonLogger" level="INFO" additivity="false">
            <AppenderRef ref="JsonAppender"/>
        </Logger>
        <Root level="info">
            <AppenderRef ref="JsonAppender"/>
        </Root>
    </Loggers>
</Configuration>
```

- 利用には、以下を設定
  - LambdaJsonLayoutを利用して構造化ロギングを行う
  - 標準出力へ出力 (CloudWatch Logsへ)

In Partnership with intel

その3: Lambda Power Tools for Java (beta)

Logging機能1: 指定サンプルレートに基づくメソッドエントリ時の自動ログ (@PowertoolsLoggingアノテーション)

@PowertoolsLogging(logEvent = true)
で CloudWatch Logsに出力された構造化ログレコード例

Lambdaコンテキスト情報の
JSONダンプがmessageとして自動ログ出力


```
{
   "instant":{
      "epochSecond":1602976697,
      "nanoOfSecond":477015000
   },
   "thread":"main",
   "level":"INFO",
   "loggerName":"sample.App",
   "message":"{
            ¥"resource¥":¥"/SampleLambdaPowerToolsJavaFunction¥",
            ¥"path¥":¥"/SampleLambdaPowerToolsJavaFunction¥",
            ¥"httpMethod¥":¥"GET¥",¥"headers¥":null,¥"multiValueHeaders¥":null,¥"queryStringParameters¥":null
            ,¥"multiValueQueryStringParameters¥":null,¥"pathParameters¥":null,¥"stageVariables¥":null,
            ¥"requestContext¥":{
                        ¥"accountId¥":¥"<accountId>¥",¥"stage¥":¥"test-invoke-stage¥",¥"resourceId¥":¥"so6ow9¥",
                        ¥"requestId¥":¥"e928fec9-748c-4b66-a88d-fac15e32a760¥",¥"operationName¥":null,
                        ¥"identity¥":{¥"cognitoIdentityPoolId¥":null,¥"accountId¥":¥"<accountId>¥",
                        ¥"cognitoIdentityId¥":null,¥"caller¥":¥"AIDAIPU5HZ5HWR5F23L3I¥",
                        ¥"apiKey¥":¥"test-invoke-api-key¥",¥"sourceIp¥":¥"test-invoke-source-ip¥",
                        ¥"cognitoAuthenticationType¥":null,¥"cognitoAuthenticationProvider¥":null,
                        ¥"userArn¥":¥"arn:aws:iam::<accountId>:user/<iamUser>¥",
                        ¥"userAgent¥":¥"<userAgent>¥",¥"user¥":¥"AIDAIPU5HZ5HWR5F23L3I¥",¥"accessKey¥":¥"<key>¥"},
                        ¥"resourcePath¥":¥"/SampleLambdaPowerToolsJavaFunction¥",¥"httpMethod¥":¥"GET¥",
                        ¥"apiId¥":¥"<AppId>¥",¥"path¥":¥"/SampleLambdaPowerToolsJavaFunction¥",
                        ¥"authorizer¥":null },
            ¥"body¥":null, ¥"isBase64Encoded¥":false}",
  ...
}
```


In Partnership with intel

その3: Lambda Power Tools for Java (beta)

Logging機能2: 明示的なアプリケーションロギングにおける構造化
→ CloudWatch Logs Insightsで条件指定での検索が容易

log.info(“sample app log messege.”);
で CloudWatch Logsに出力された構造化ログレコード例

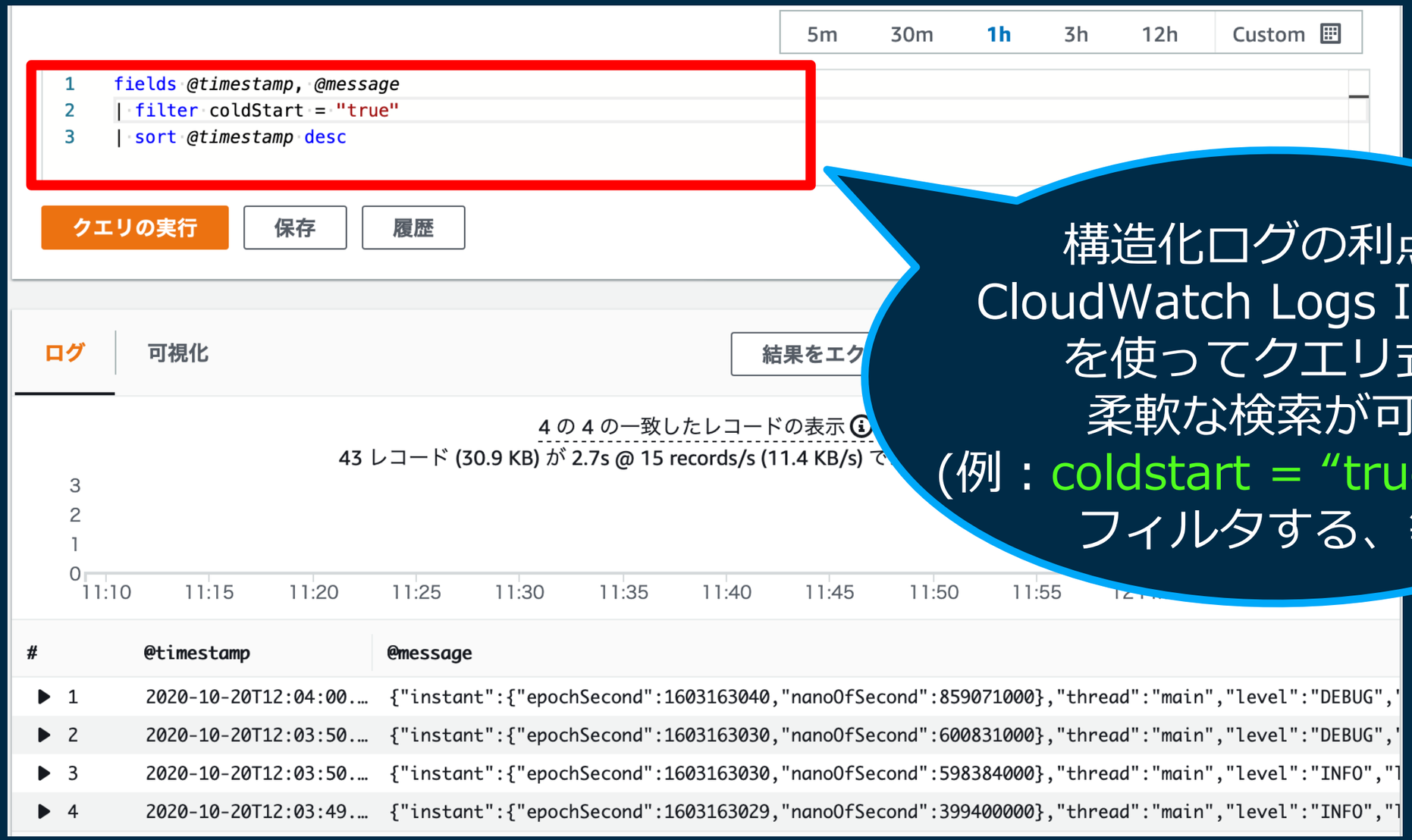
```
{
   "instant":{
      "epochSecond":1602976698,
      "nanoOfSecond":637298000
   },
   "thread":"main",
   "level":"INFO",
   "loggerName":"sample.App",
   "message":"sample app log message.",
   "endOfBatch":false,
   "loggerFqcn":"org.apache.logging.log4j.spi.AbstractLogger",
   "threadId":1,
   "threadPriority":5,
   "timestamp":”2020-10-17T23:18:18.637Z[UTC]",
   "coldStart":"true",
   "functionArn":"arn:aws:lambda:<region>:<account>:function:<functionName>",
   "functionMemorySize":"512",
   "functionName":”<functionName>",
   "functionVersion":"$LATEST",
   "samplingRate":"1.0",
   "service":"payment",
   "test":"willBeLogged",
   "xray_trace_id":”<X-Ray traceId>"
}
```

アプリのログメッセージ

- message以外に
  以下フィールドが自動追加
  - coldstart (ColdStartかどうか)
  - timestamp , level
  - functionArn
  - functionName
  - functionMemorySize
  - functionVersion
  - service, xray_trace_id

In Partnership with intel

その3: Lambda Power Tools for Java (beta)

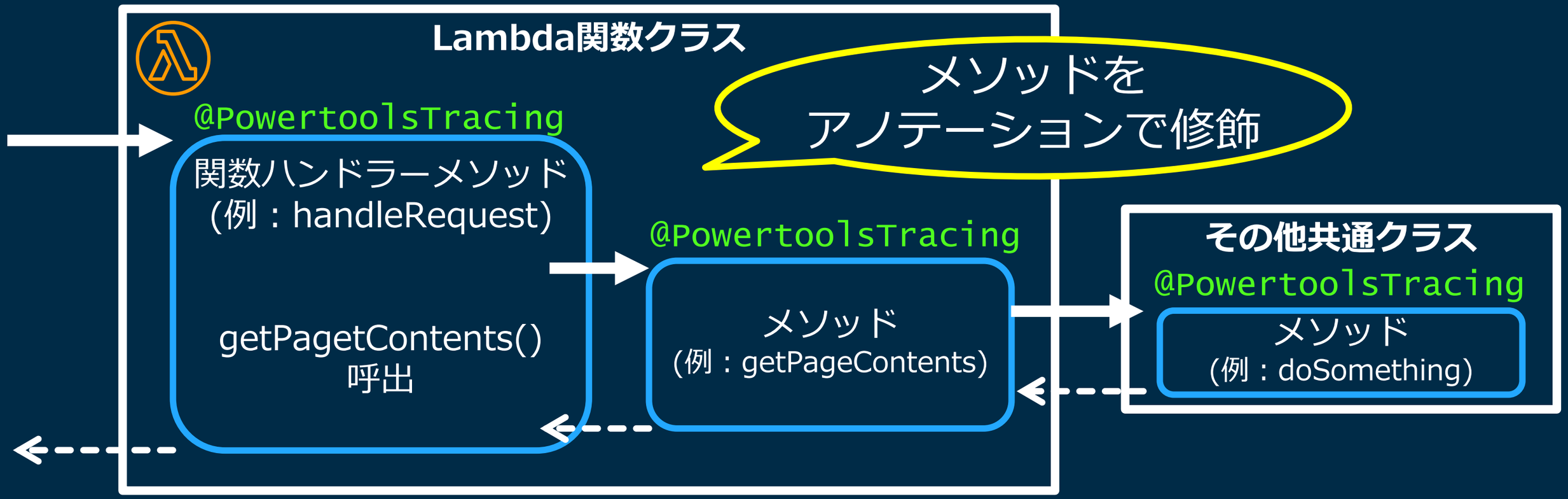


構造化ログの利点：
CloudWatch Logs Insights
を使ってクエリ式で
柔軟な検索が可能
(例：coldstart = “true” のみで
フィルタする、等)

In Partnership with intel

その3: Lambda Power Tools for Java (beta)

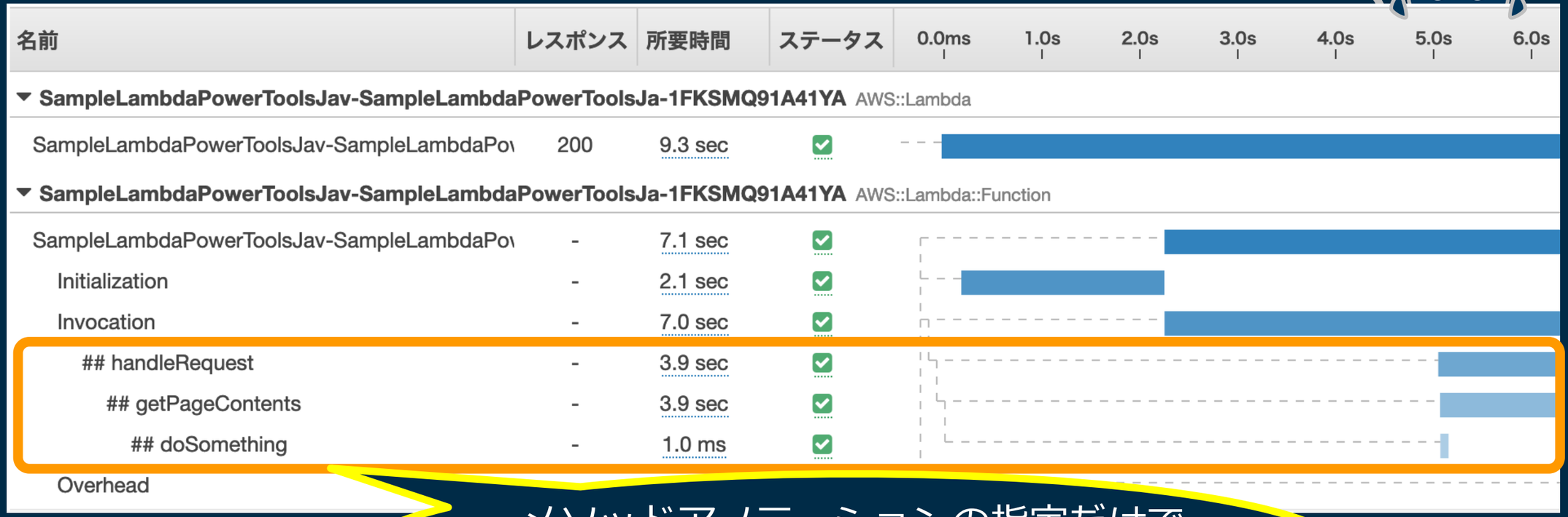
Tracing機能: ハンドラメソッドを含む
@PowertoolsTracingアノテーションの設定だけでX-Rayトレースを送信



In Partnership with intel

その3: Lambda Power Tools for Java (beta)

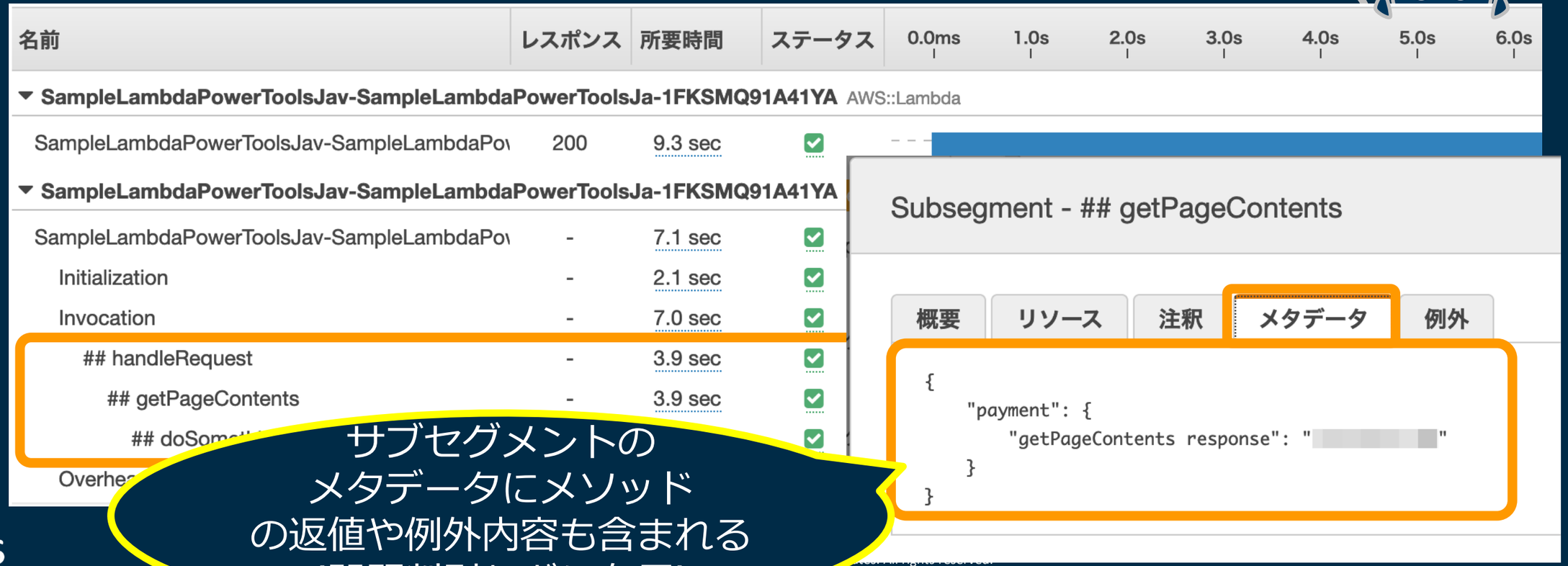
Tracing機能: ハンドラメソッドを含む
@PowertoolsTracingアノテーションの設定だけでX-Rayトレースを出力



メソッドアノテーションの指定だけで
サブセグメントの呼び出し関係トレースが取得

その3: Lambda Power Tools for Java (beta)

Tracing機能: ハンドラメソッドを含む
@PowertoolsTracingアノテーションの設定だけでX-Rayトレースを出力



In Partnership with intel

その4: AWS X-Ray Auto-Instrumentation Agent for Java

- **AWS Toolkit**
- **AWS X-Ray Java Agent**

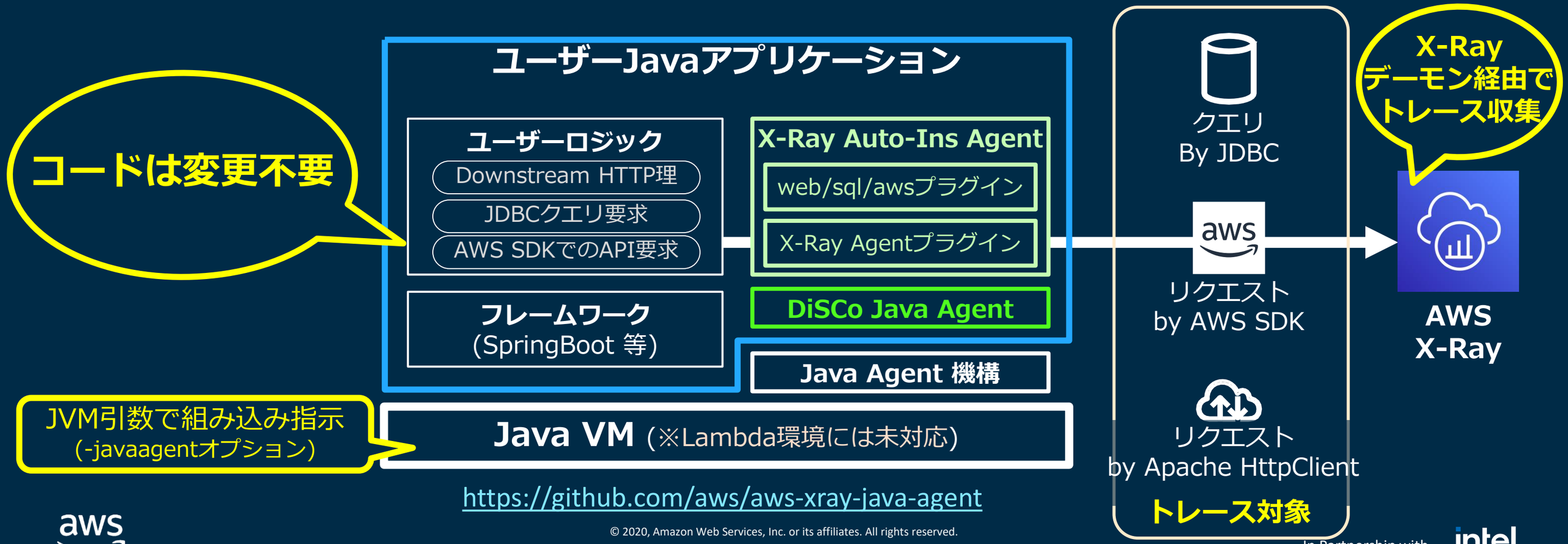
- **Amazon Corretto**
- **Other JVMs** (OpenJDK, etc...)

- **Amazon EC2** (Elastic Compute Cloud)
- **AWS Lambda**
- **Amazon ECS** (Elastic Container Service)
- **Amazon EKS** (Elastic Kubernetes Service)

In Partnership with intel

その4: AWS X-Ray Auto-Instrumentation Agent for Java

Javaエージェントの組み込みにより、Apache HTTP/JDBC/
AWS SDKリクエストのトレースをX-Rayへ送信 **9月4日 New!**

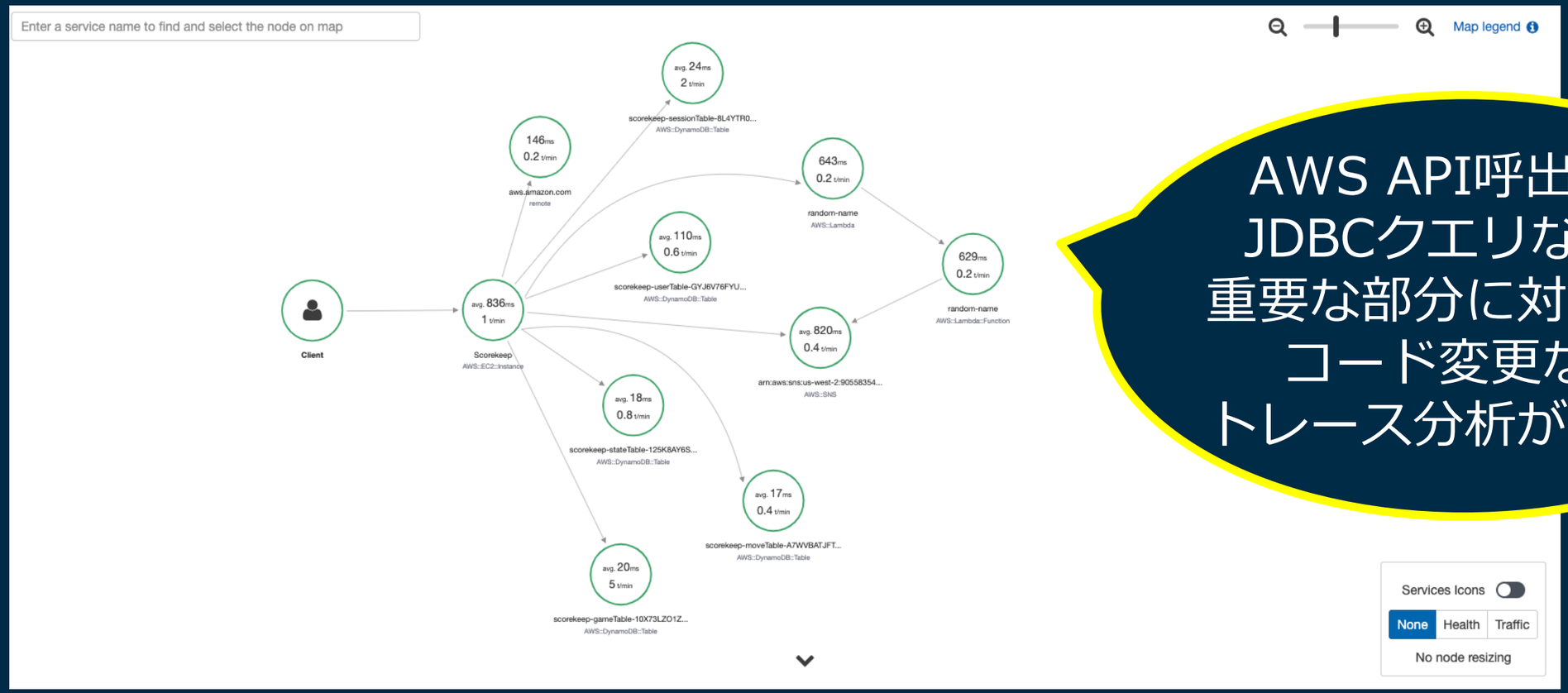


https://github.com/aws/aws-xray-java-agent

In Partnership with intel

その4: AWS X-Ray Auto-Instrumentation Agent for Java

「アプリのLift型更改」であっても、AWS X-Rayを用いた
分散トレーシングをコードへの変更なく、設定やインフラ構成で実現



AWS API呼出しや
JDBCクエリなどの
重要な部分に対して、
コード変更なく
トレース分析が可能に

In Partnership with intel

その5: AWS Toolkit for Eclipse / IntelliJ IDEA

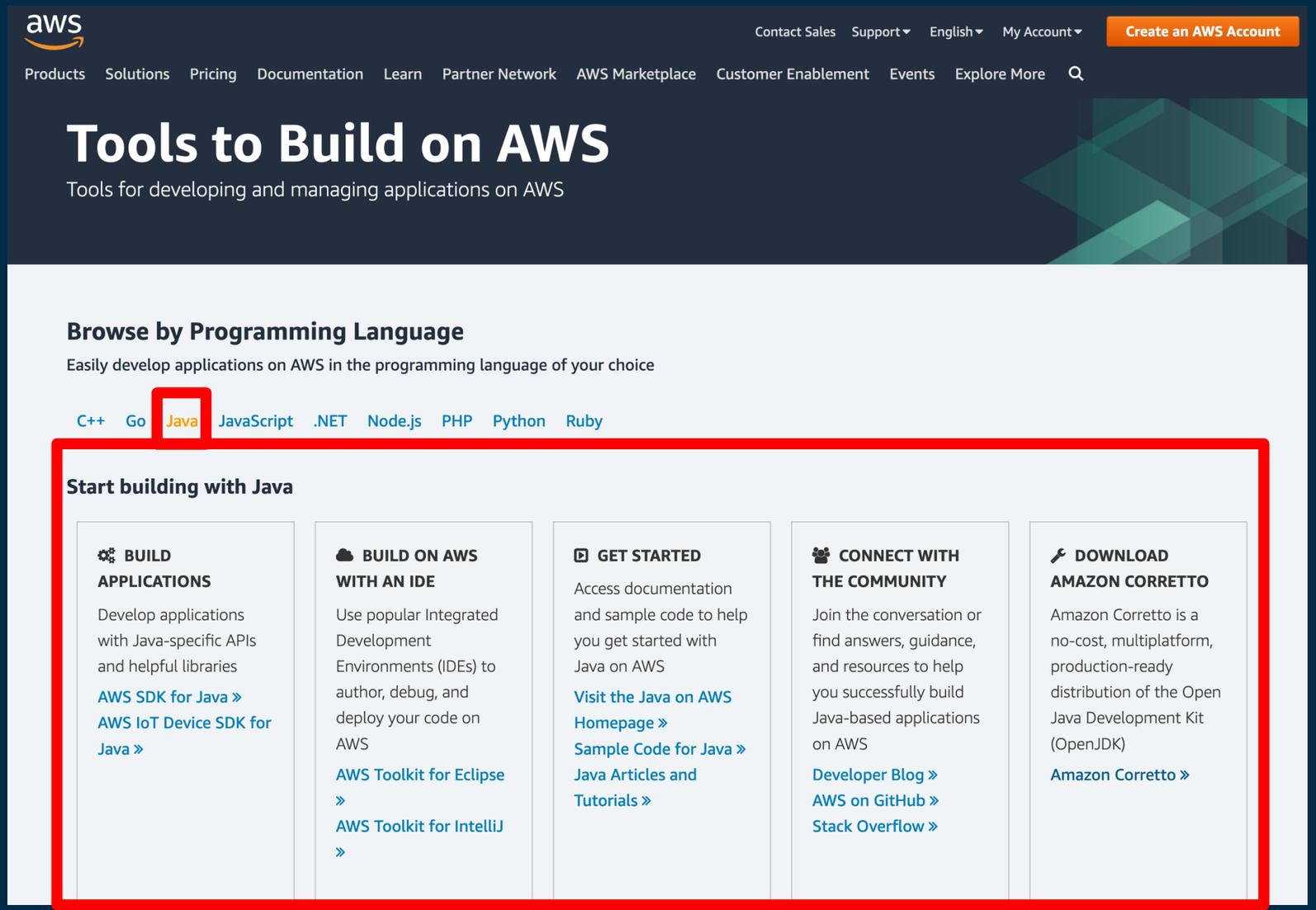
- **AWS Toolkit**
- **AWS X-Ray Java Agent**

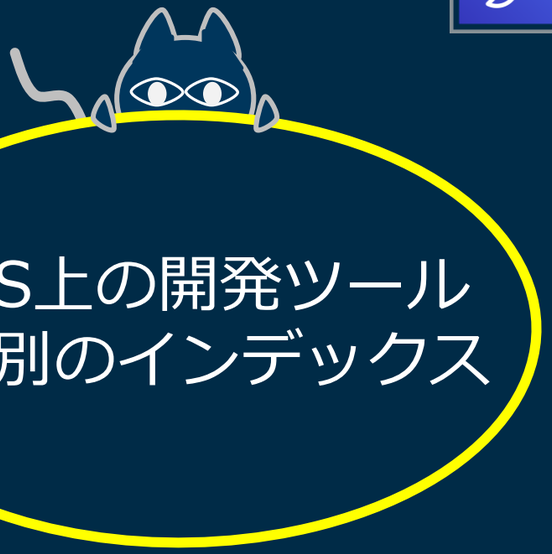
- **Amazon Corretto**
- **Other JVMs** (OpenJDK, etc...)

- **Amazon EC2** (Elastic Compute Cloud)
- **AWS Lambda**
- **Amazon ECS** (Elastic Container Service)
- **Amazon EKS** (Elastic Kubernetes Service)

In Partnership with intel

AWS Tools and Library





https://aws.amazon.com/tools/

In Partnership with intel

その5: AWS Toolkit for Eclipse / IntelliJ IDEA

(※他に VisualStudio Code 用なども提供)

AWS環境へのアクセス・操作を
統合開発環境(IDE)から行いやすくするツールキット

Eclipse

IntelliJ IDEA

https://aws.amazon.com/eclipse/

https://aws.amazon.com/intellij/

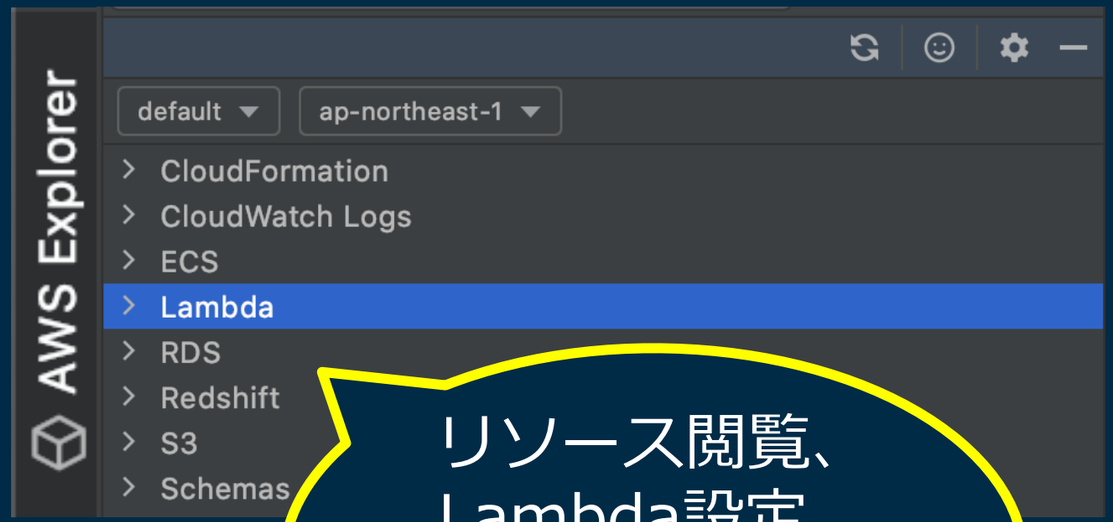
In Partnership with intel

その5: AWS Toolkit for Eclipse / IntelliJ IDEA

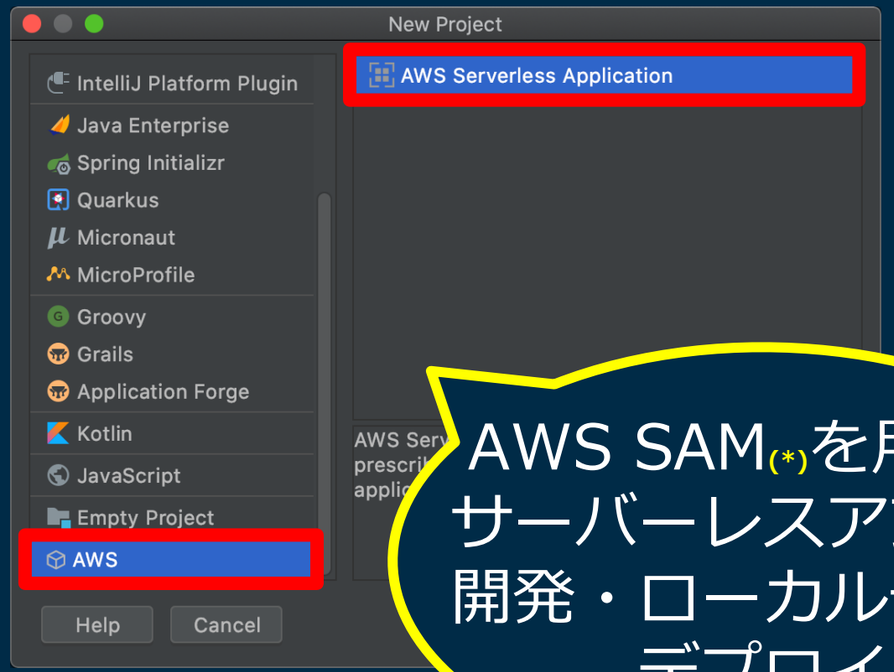
「AWS Explorer」によるサポート対象サービスのリソースを一覧・表示
& SAMを利用したLambdaアプリケーションのローカル開発・デプロイ

主な機能 (画面例: IntelliJ IDEA)

SAM: Serverless Application Model



リソース閲覧、
Lambda設定、
ログストリーム内容の
表示、…



AWS SAM(*)を用いた
サーバーレスアプリの
開発・ローカルテスト
デプロイ

前半のまとめ

AWSに　Javaで　作る

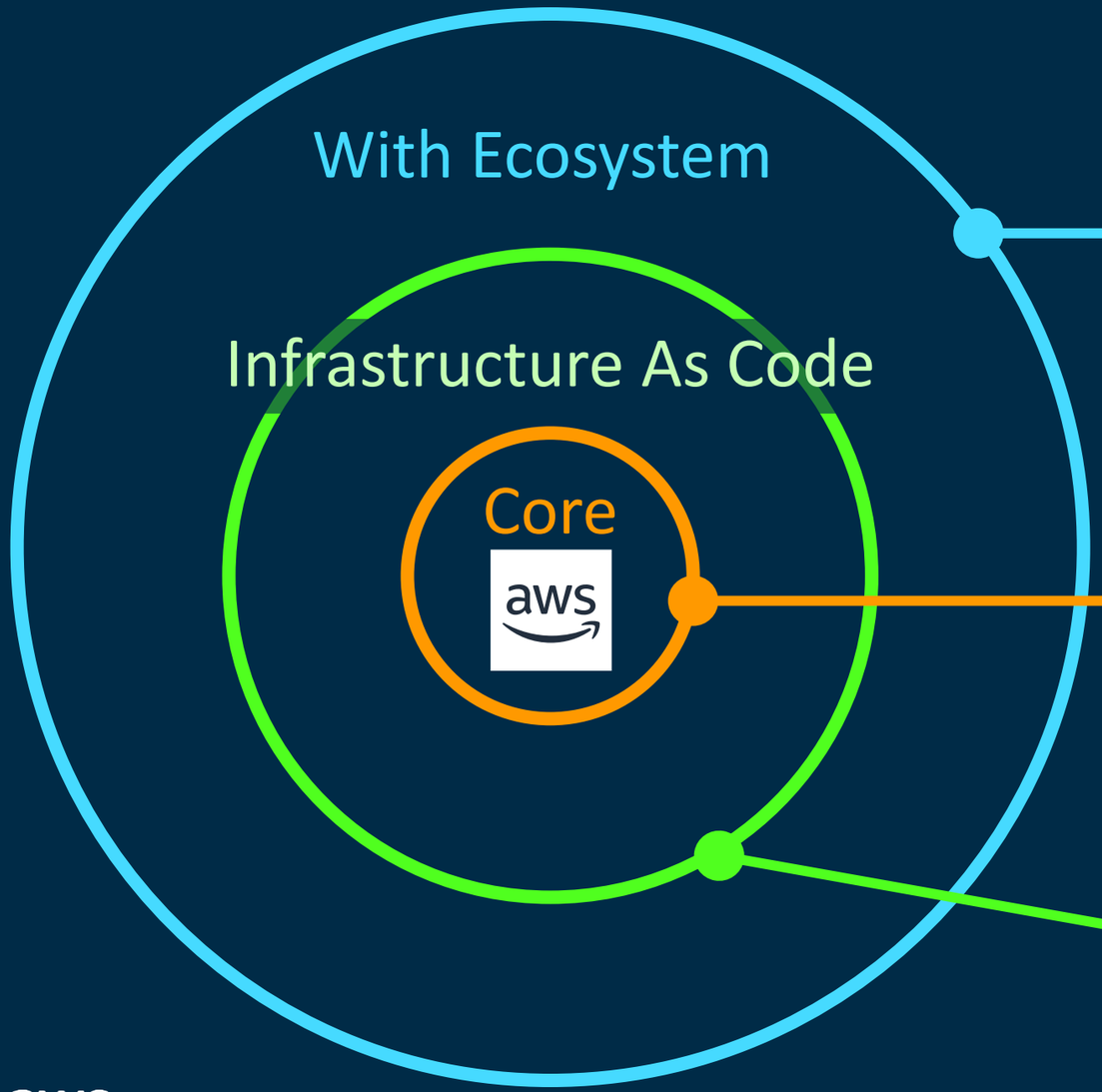
- AWS Toolkit
- X-Ray Auto-Instrumentation Agent for Java
- Lambda Power Tools for Java
- AWS Lambda & Java
- Amazon Corretto

aws DEV DAY
ONLINE JAPAN

AWSを Javaで 操る

In Partnership with intel

「AWSを」「Javaで」「操る」



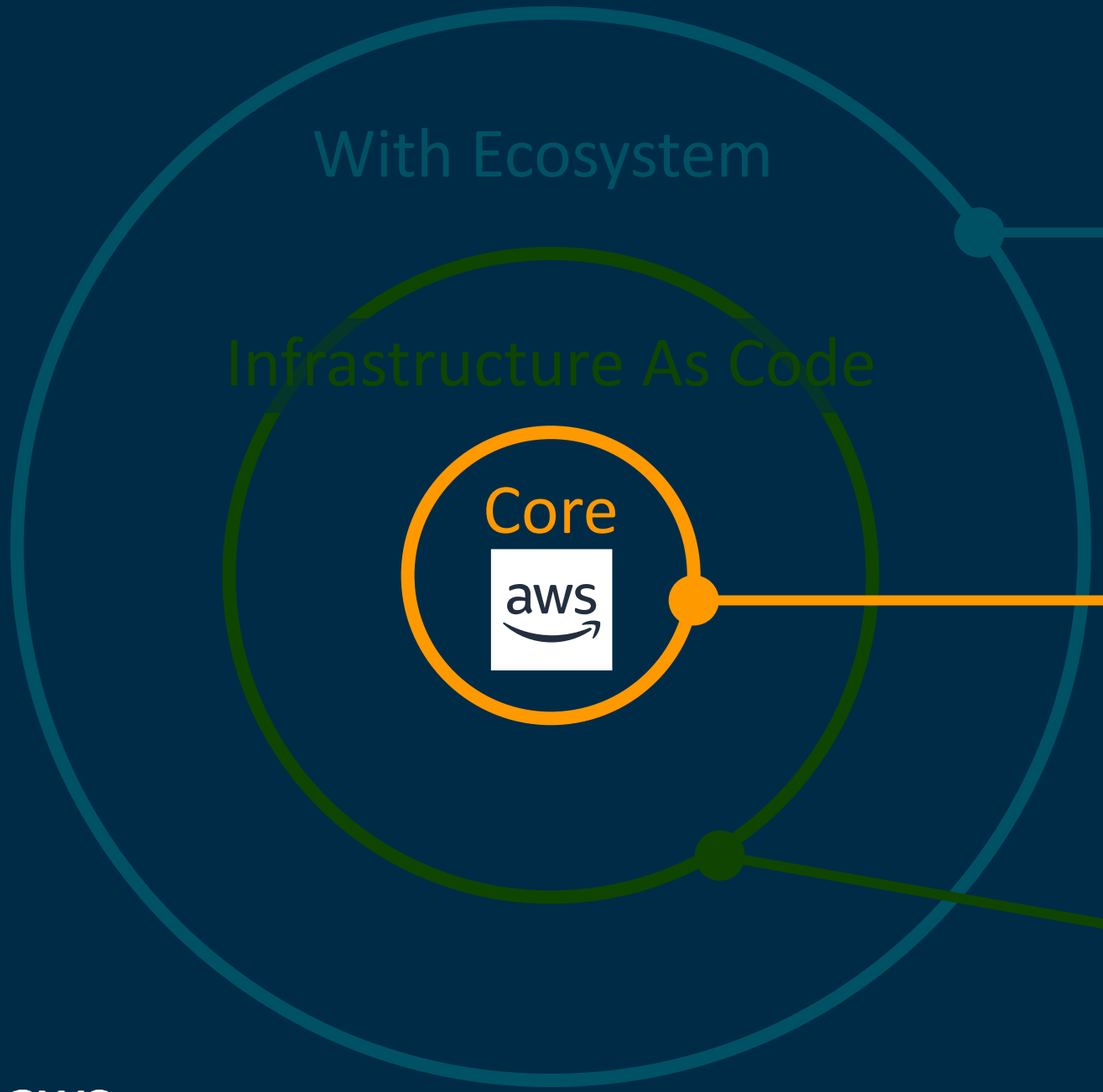
Javaエコシステムの活用

Javaによる AWSリソース の操作

Javaによる IaC の実現

In Partnership with intel

Core：Javaによる AWSリソース の操作



Javaエコシステムの活用

Javaによる AWSリソース の操作

Javaによる IaC の実現

In Partnership with intel

その6: AWS SDK for Java

AWS は API で操作できるもの (APIの集合体と見なせる)

In Partnership with intel

その6: AWS SDK for Java

AWS SDKは、AWSサービスのAPIを利用するSDK

In Partnership with intel

その6: AWS SDK for Java

AWS SDKは、AWSサービスのAPIを利用するSDK
→ Java SDK も提供

In Partnership with intel

その6: AWS SDK for Java v2

AWS SDK for Java 2.0 が現在は利用可能

In Partnership with intel

その6: AWS SDK for Java v2

新規アプリケーション開発では v2 の利用が推奨
(ただし、一部機能は v1 にのみ存在 ※こちらを参照)

AWS SDK for Java 1.0 → **AWS SDK for Java 2.0**

AWS SDK for Java 1.0	比較項目	AWS SDK for Java 2.0
Java 1.7+	**Javaバージョン**	Java 1.8+
com.amazonaws	**Maven groupId**	software.amazon.awssdk
未対応	**非同期I/O**	対応
コンストラクタ,ビルダー	**クライアント生成**	ビルダー(ファクトリメソッド)
例：AWSACMPCAAsyncClient	**クラスネーミング**	例：AcmAsyncClient
-	**Lambda実装観点**	ColdStart時間短縮に配慮

とはいえ… 2.0 と 1.0 の共存は可能

```
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-bom</artifactId>
            <version>1.11.428</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <dependency>
          <groupId>software.amazon.awssdk</groupId>
          <artifactId>bom</artifactId>
          <version>2.1.0</version>
          <type>pom</type>
          <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>

<dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-s3</artifactId>
    </dependency>
    <dependency>
      <groupId>software.amazon.awssdk</groupId>
      <artifactId>dynamodb</artifactId>
    </dependency>
</dependencies>
```

1.0系

2.0系

両方のgroupIdを
依存性管理に指定

その上で必要な
ライブラリ単位で
依存性を定義

その7: 各サービスのJava固有ライブラリ(pickup)

いくつかのサービスは Java 向けに固有のライブラリを提供

Amazon Athena

Athena JDBC Driver

Document Link

JDBC4.1 or JDBC4.2 が提供

Athena(サーバレスのS3上データSQLクエリサービス)に対して、Javaプログラムから JDBC APIによるアクセスを実現するサービス

In Partnership with intel

その7: 各サービスのJava固有ライブラリ(pickup)

いくつかのサービスは Java 向けに固有のライブラリを提供

Amazon Athena

Athena JDBC Driver

Document Link

JDBC4.1 or JDBC4.2 が提供

Athena(サーバレスのS3上データSQL
クエリサービス)に対して、Javaプログ
ラムからJDBC APIによる
アクセスを実現するサービス

Amazon Kinesis

Kinesis Client Library Java 2.x

awslabs / amazon-kinesis-client

現時点では Java と Python のみ 提供

シャードの増減を検知し新規シャード
の開始を自動処理するなどの機能あり
Kinesis Data Analyticsでは
Apache Flinkを利用可能

その7: 各サービスのJava固有ライブラリ(pickup)

いくつかのサービスは Java 向けに固有のライブラリを提供

AWS CloudTrail

CloudTrail Processing Library

aws / aws-cloudtrail-processing-library

AWS SDK for Java 1.10 & Java 1.8+

S3/SNS→SQSのCloudTrailログを読込・解析処理を支援するライブラリ

スケール性に優れ、ビジネスロジックでのレコード処理が可能

その7: 各サービスのJava固有ライブラリ(pickup)

いくつかのサービスは Java 向けに固有のライブラリを提供

AWS CloudTrail

CloudTrail Processing Library

aws / aws-cloudtrail-processing-library

AWS SDK for Java 1.10 & Java 1.8+

S3/SNS→SQSのCloudTrailログを読込・解析処理を支援するライブラリ

スケール性に優れ、ビジネスロジックでのレコード処理が可能

AWS CloudHSM

JCE Provider

Document Link

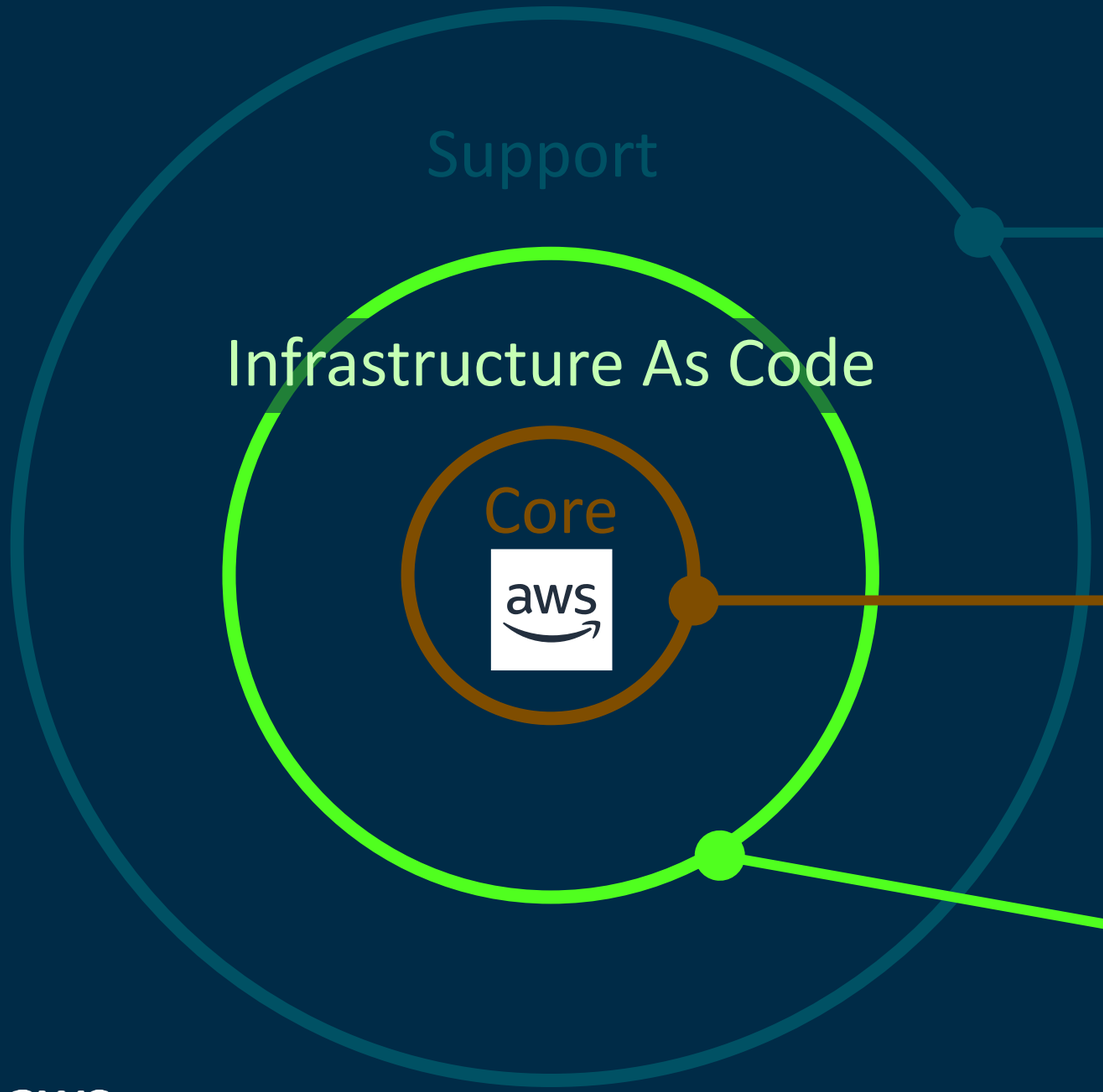
JDK & CloudHSM client

CloudHSM用のJCEプロバイダ実装

JCE/JCAのインターフェイスでCloudHSMの鍵を操作・利用可能

In Partnership with intel

IaC：JavaによるInfrastructure As Codeの実現



Javaエコシステムの活用

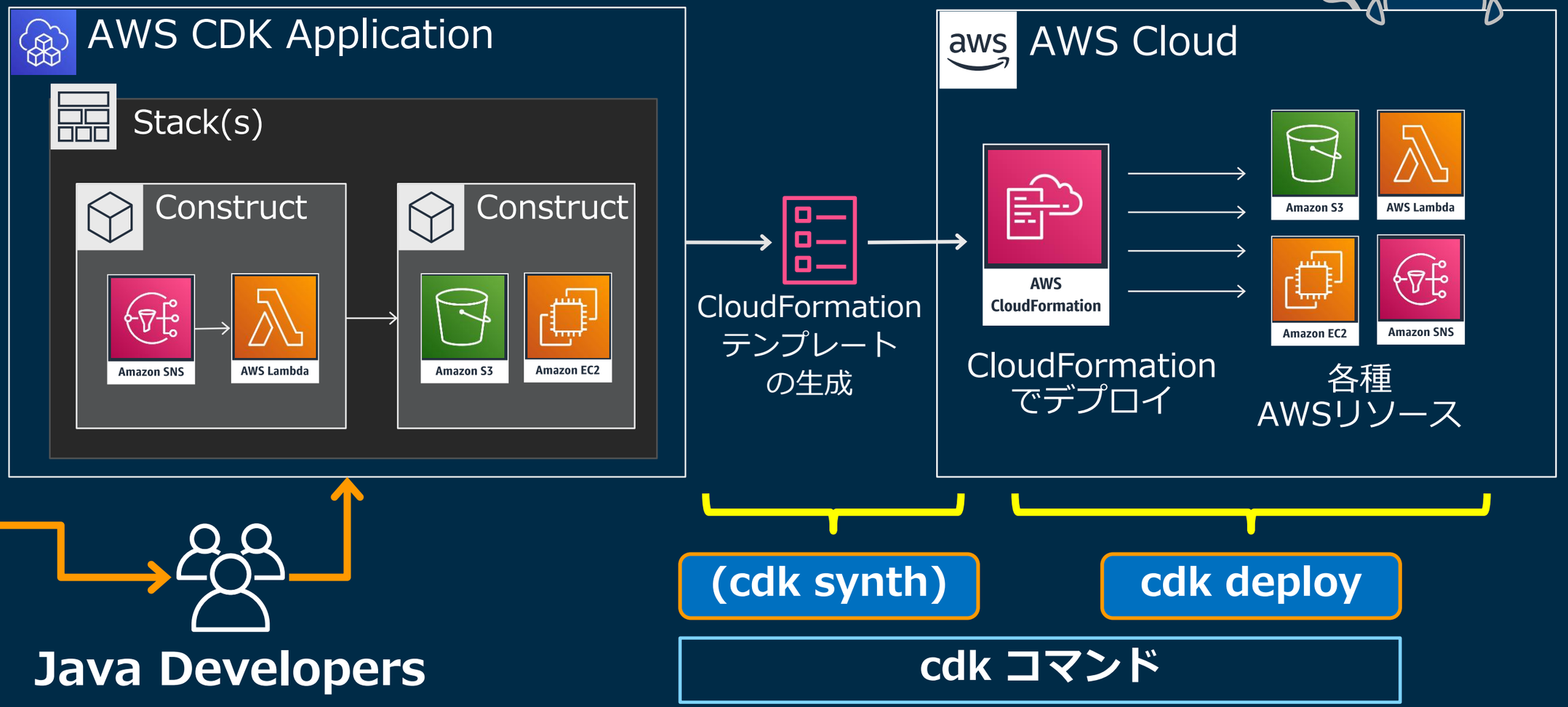
Javaによる AWSリソース の操作

Javaによる IaC の実現

In Partnership with intel

その6: AWS CDK (Cloud Development Kit)

「こうなっていて欲しい」状態をコードで表現



In Partnership with intel

その6: AWS CDK (Cloud Development Kit)

cdk init app --language java でCDKプロジェクトを生成

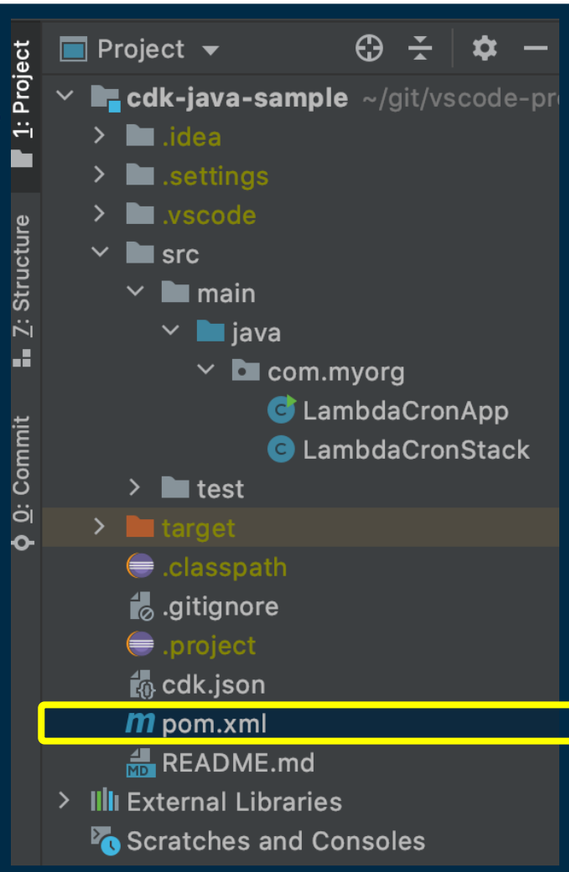
```
$ mkdir cdk-java-sample
$ cd cdk-java-sample
$ cdk init app --language java

Applying project template app for java
# Welcome to your CDK Java project!
This is a blank project for Java development with CDK.
The `cdk.json` file tells the CDK Toolkit how to execute your app.

It is a [Maven](https://maven.apache.org/) based project, so you can open this
project with any Maven compatible Java IDE to build and run tests.

## Useful commands
 * `mvn package`     compile and run tests
 * `cdk ls`          list all stacks in the app
 * `cdk synth`       emits the synthesized CloudFormation template
 * `cdk deploy`      deploy this stack to your default AWS account/region
 * `cdk diff`        compare deployed stack with current state
 * `cdk docs`        open CDK documentation
Enjoy!

Initializing a new git repository...
Executing 'mvn package'
✅ All done!
```

--language javaを
指定してCDK
プロジェクトを作成

その6: AWS CDK (Cloud Development Kit)

pom.xmlにCDK Javaライブラリへの依存性を定義
(扱いたいAWSリソースに合わせて対応ライブラリを依存性に指定)

Maven形式の
Javaプロジェクト構造
が生成
(pom.xml等)




```
 :
<dependencies>
    <!-- AWS Cloud Development Kit -->
    <dependency>
        <groupId>software.amazon.awscdk</groupId>
        <artifactId>core</artifactId>
        <version>${cdk.version}</version>
    </dependency>

    <dependency>
        <groupId>software.amazon.awscdk</groupId>
        <artifactId>lambda</artifactId>
        <version>${cdk.version}</version>
    </dependency>

    <dependency>
        <groupId>software.amazon.awscdk</groupId>
        <artifactId>events-targets</artifactId>
        <version>${cdk.version}</version>
    </dependency>
     :
</dependencies>
```


In Partnership with intel

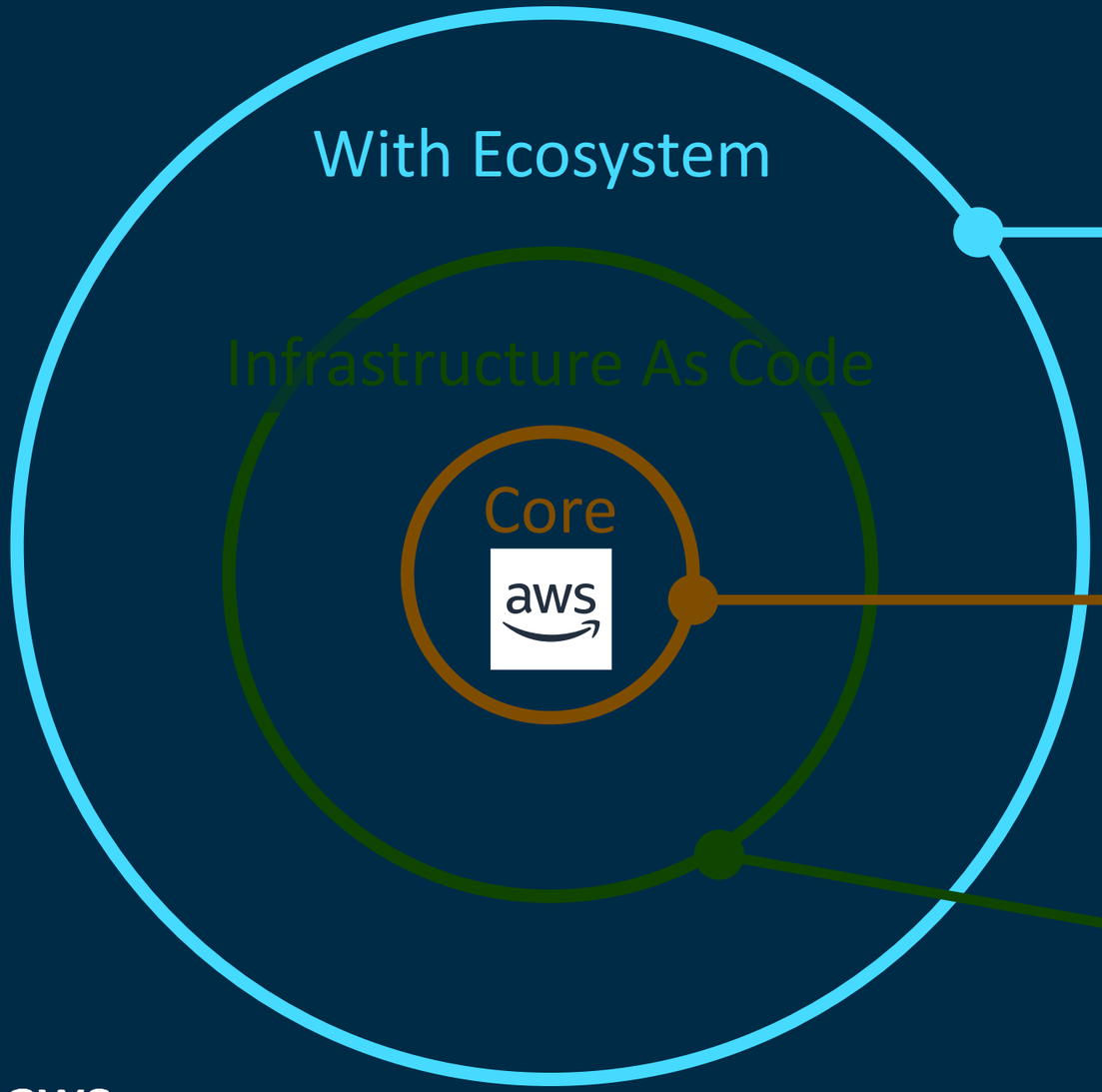
その6: AWS CDK (Cloud Development Kit)

Java型による型安全性やIDEによるクラス/メソッド名補完のメリットを享受しつつ、JavaクラスでAWSリソース定義→cdk deployで生成

```
package com.myorg;

import java.util.UUID;
import software.amazon.awscdk.core.Construct;
import software.amazon.awscdk.core.Duration;
import software.amazon.awscdk.core.Stack;
:
public class LambdaCronStack extends Stack {

    public LambdaCronStack(final Construct scope, final String id) {
        super(scope, id);

        SingletonFunction lambdaFunction = SingletonFunction.Builder.create(this, "cdk-lambda-cron")
            .description("Lambda which prints ¥"I'm running¥"")
            .code(Code.fromInline("def main(event, context):¥n" + "    print(¥"I'm running!¥")¥n"))
            .handler("index.main").timeout(Duration.seconds(300)).runtime(Runtime.PYTHON_2_7)
            .uuid(UUID.randomUUID().toString()).build();

        Rule rule = Rule.Builder.create(this, "cdk-lambda-cron-rule")
                .description("Run every day at 6PM UTC")
                .schedule(Schedule.expression("cron(0 18 ? * MON-FRI *)"))
                .build();
        rule.addTarget(new LambdaFunction(lambdaFunction));
    }
}
```

CDKライブラリのStackクラスを継承し、AWSリソースを属性とともに設定
(左はLambda関数を定義した例)

With Ecosystem：Javaエコシステムの活用



Javaエコシステムの活用

Javaによる AWSリソース の操作

Javaによる IaC の実現

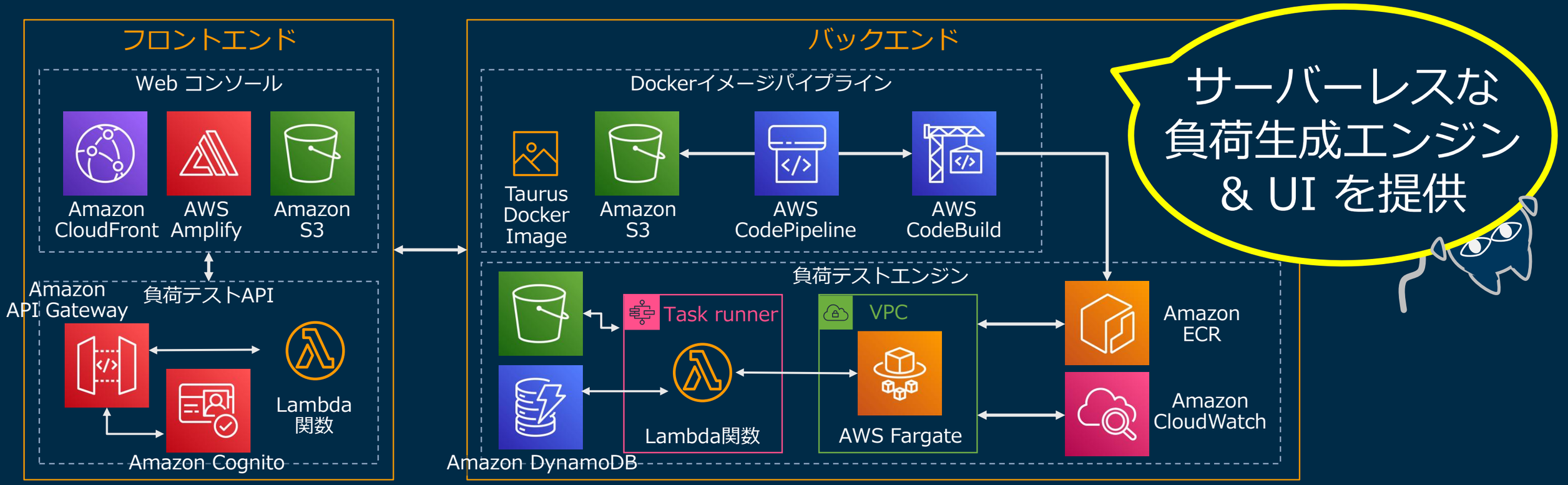
In Partnership with intel

その8: Distributed Load Testing on AWS v1.1

In Partnership with intel

その8: Distributed Load Testing on AWS v1.1

分散負荷テスト用の AWS Solutions の一つ (Solutions自体は無償, 各サービス利用料)
Apache JMeterテストシナリオをサポート **(←10月6日 New!)**

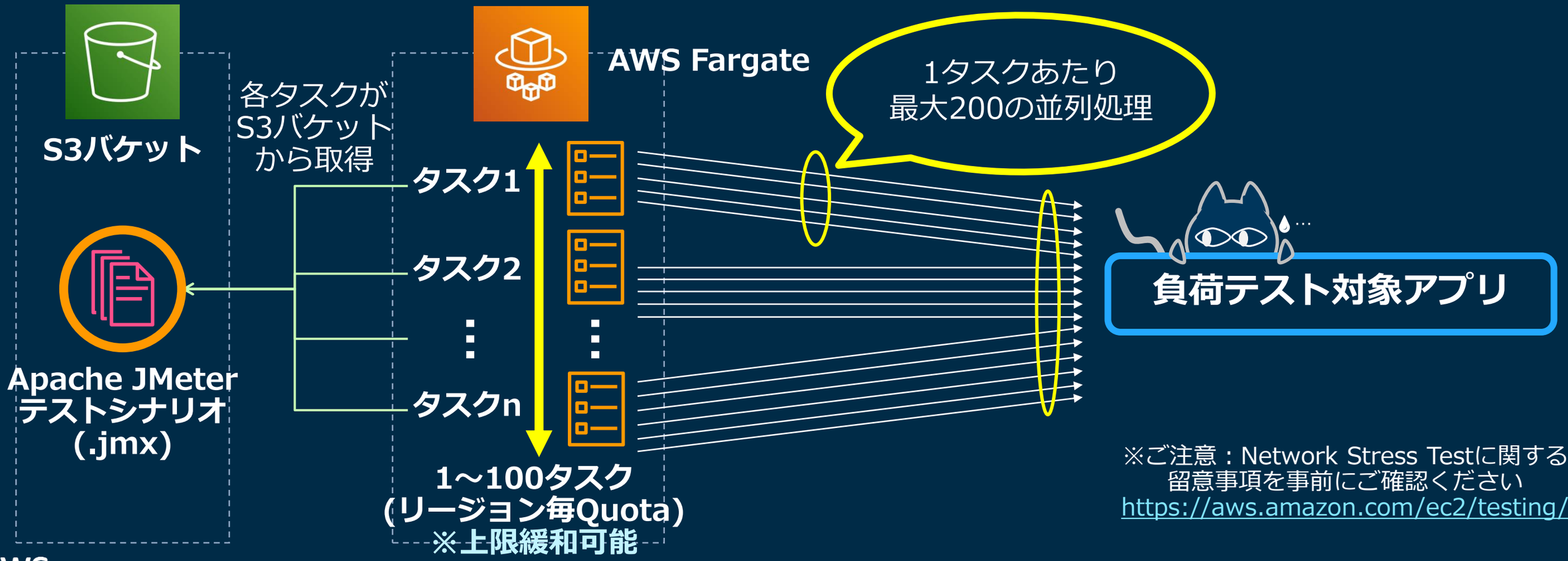


https://aws.amazon.com/solutions/implementations/distributed-load-testing-on-aws/?nc1=h_ls

In Partnership with intel

その8: Distributed Load Testing on AWS v1.1

負荷生成エンジンに OSS の Taurus を利用
Fargateを用いてサーバーレスでスケールし大規模な負荷テストに対応



※ご注意：Network Stress Testに関する
留意事項を事前にご確認ください
https://aws.amazon.com/ec2/testing/

In Partnership with intel

その8: Distributed Load Testing on AWS v1.1

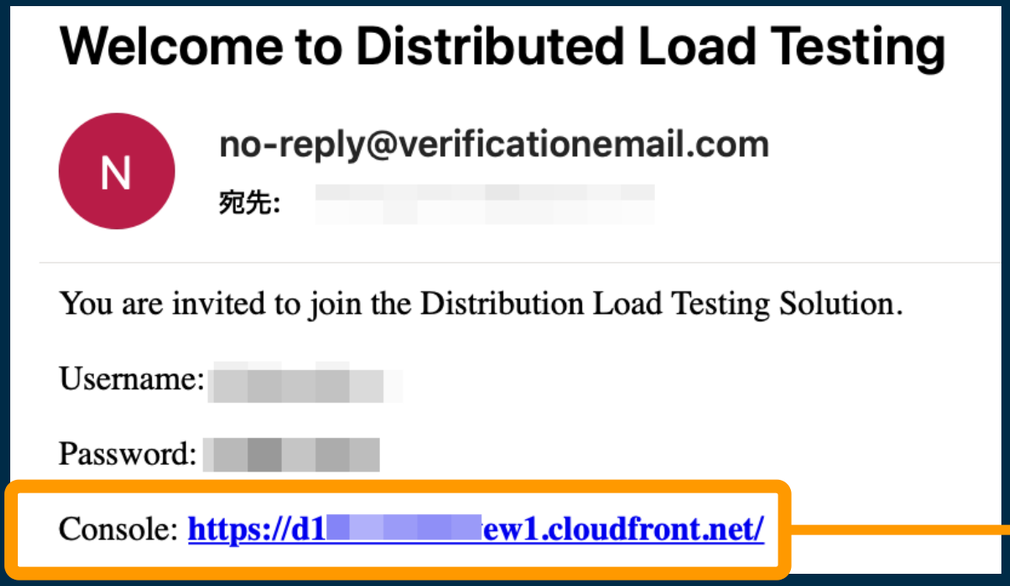
CloudFormationスタックのパラメーター
(→新規にFargateタスク用VPCを作成)

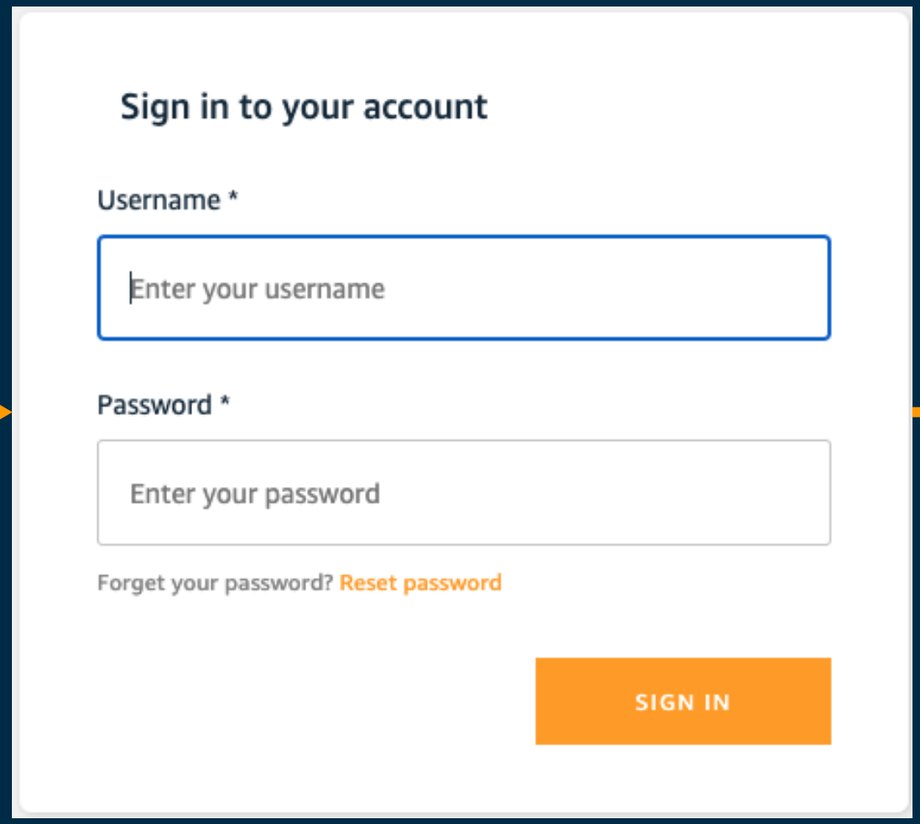
#	パラメーター	説明
1	Console Administrator Name	管理者ユーザー名
2	Console Administrator Email	管理者Eメールアドレス(初期パスワード通知先)
4	AWS Fargate VPC CIDR Block	Fargate用新規VPCのCIDRブロック (デフォルト値：192.168.0.0/16)
5	AWS Fargate Subnet A CIDR Block	Fargate用VPC内のサブネットA用CIDRブロック (デフォルト値：192.168.0.0/20)
6	AWS Fargate Subnet B CIDR Block	Fargate用VPC内のサブネットB用CIDRブロック (デフォルト値：192.168.16.0/20)
7	AWS Fargate SecurityGroup CIDR Block	コンテナのアウトバウンド先許可CIDRブロック (デフォルト値：0.0.0.0/0 ※適宜制限して利用)

その8: Distributed Load Testing on AWS v1.1

デプロイが完了したら、Administrator Emailアドレスに
コンソールUI(CloudFront) の URL と初期パスワードが届く

管理者メールアドレスに届くメール
(コンソールURLと初期パスワードが記載)



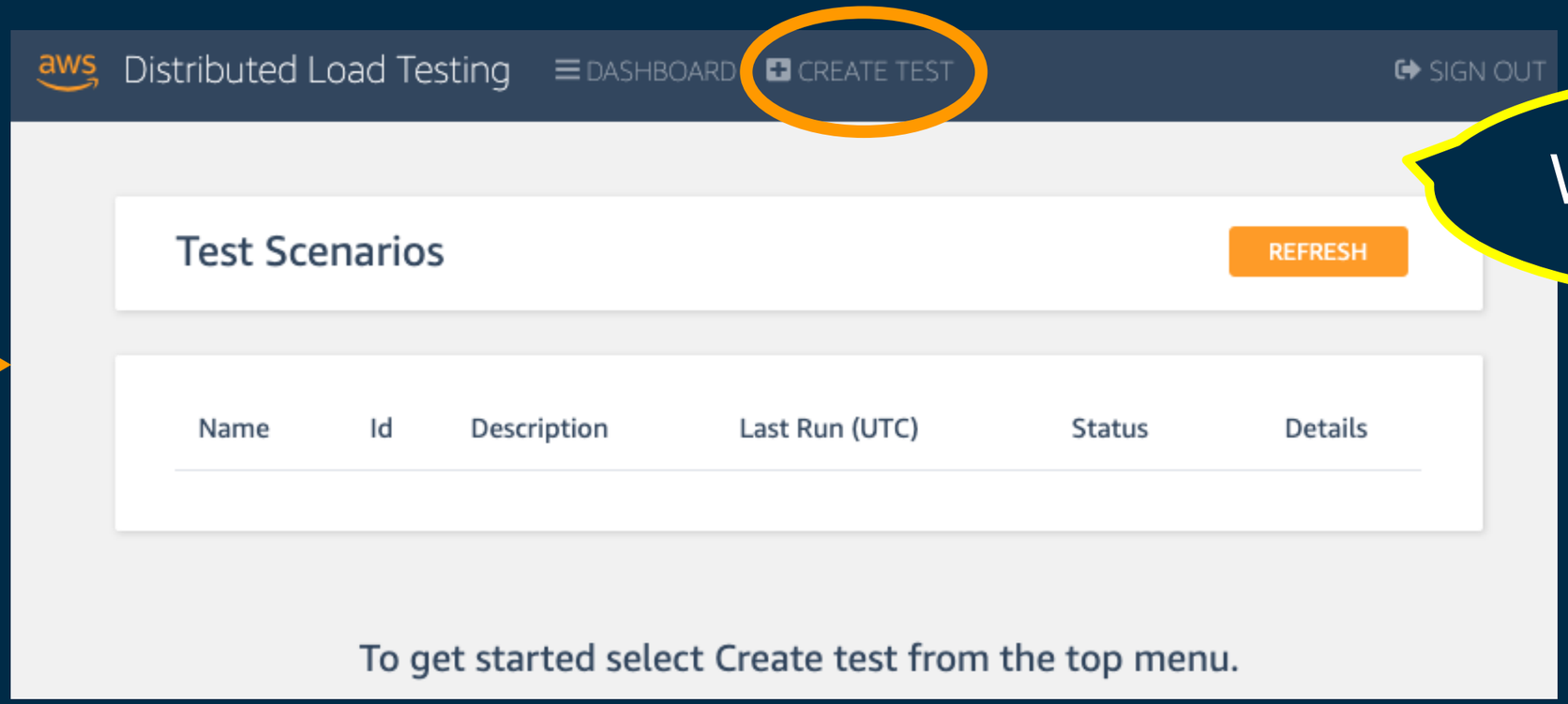


ログイン

In Partnership with intel

その8: Distributed Load Testing on AWS v1.1

テストシナリオ管理ダッシュボード
(テストシナリオの作成や状態一覧)

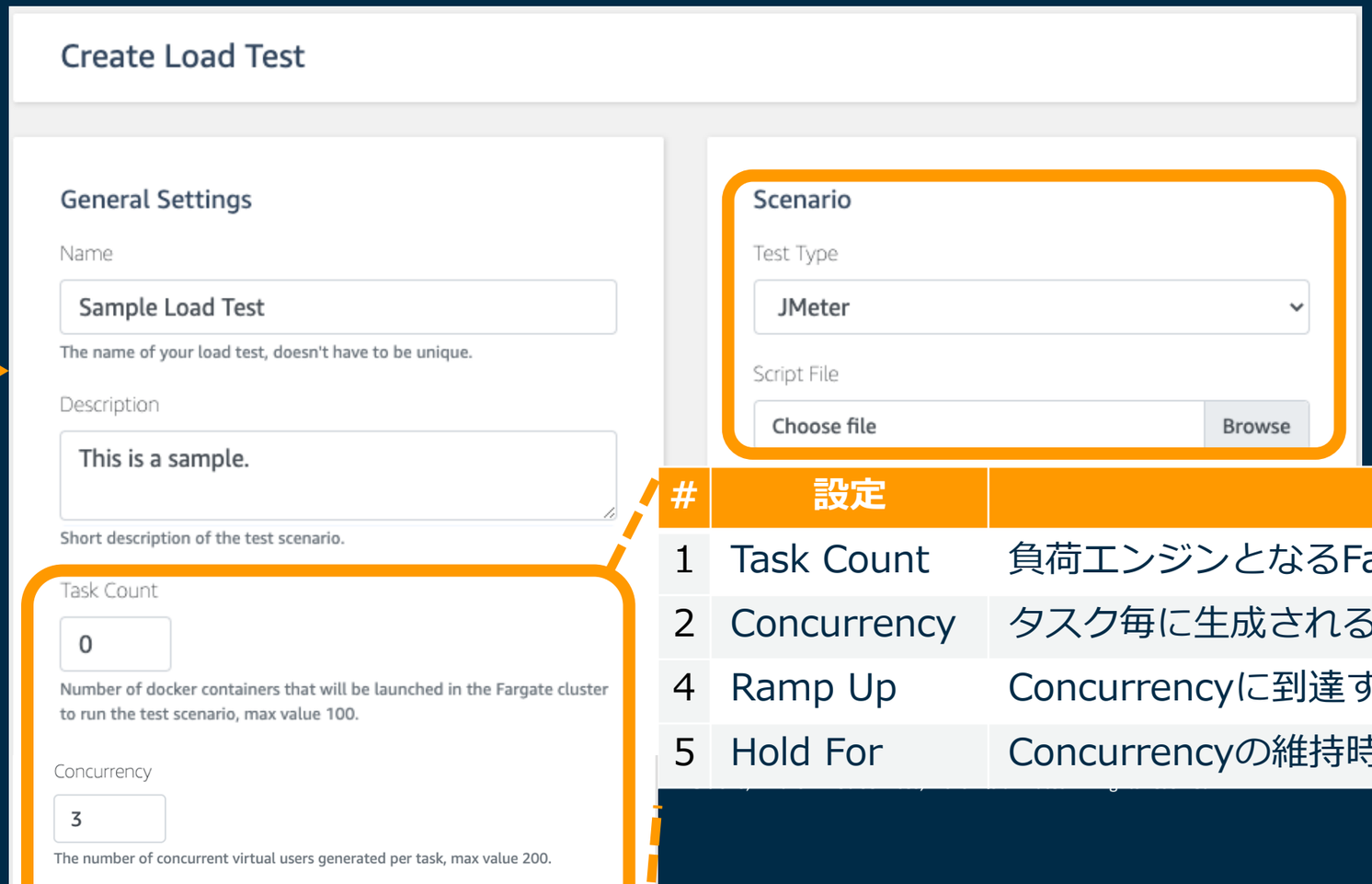


Web UIを提供

In Partnership with intel

その8: Distributed Load Testing on AWS v1.1

新規テストの作成で、シナリオ定義ファイルと負荷設定を行い投入



JMeter
定義(.jmx)を
選択&アップロード
(→S3に保存)

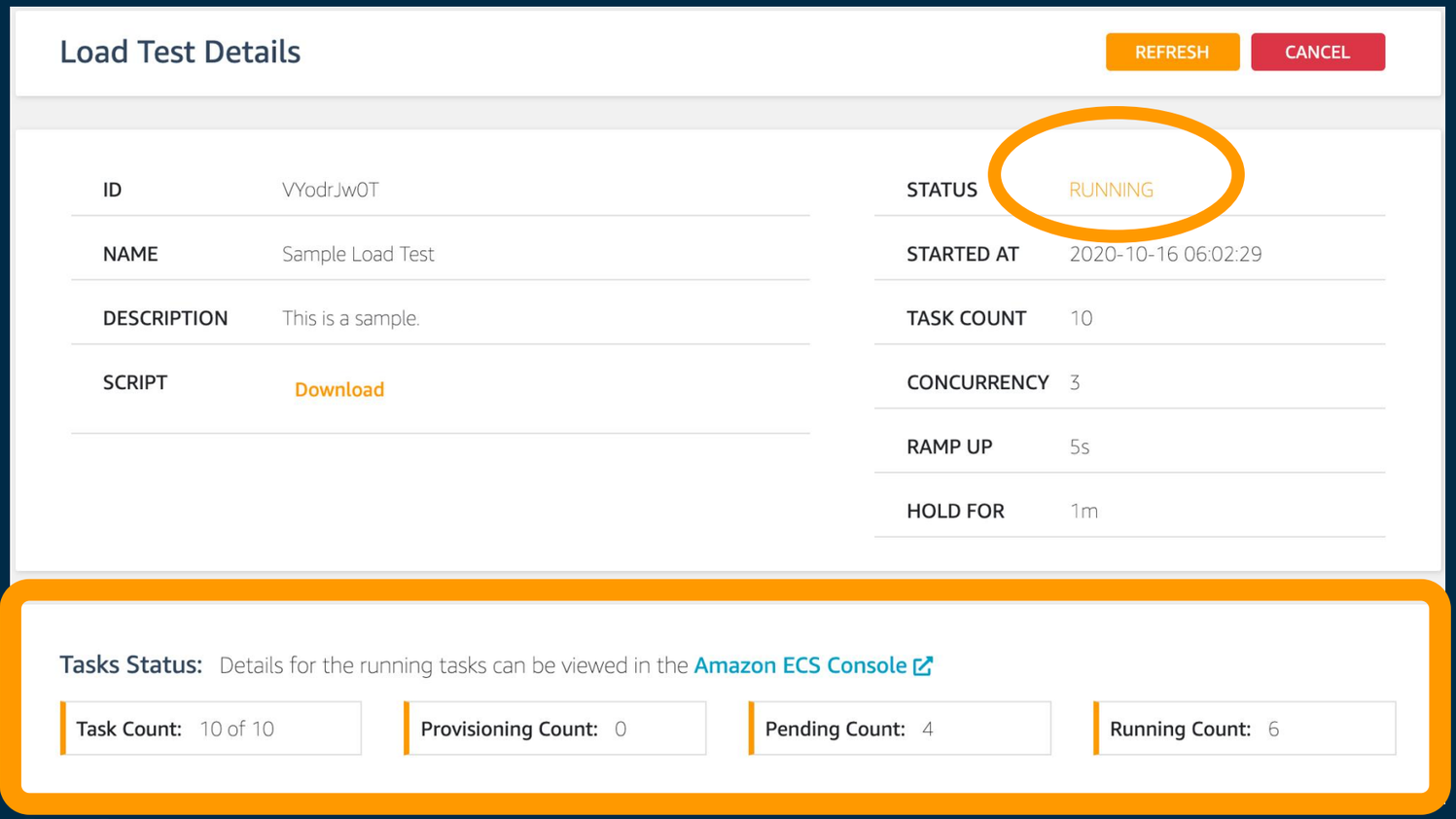
#	設定	説明
1	Task Count	負荷エンジンとなるFargateタスクの数(最大100)
2	Concurrency	タスク毎に生成される仮想ユーザー数(最大200)
4	Ramp Up	Concurrencyに到達する時間(0以上, 単位：秒 or 分)
5	Hold For	Concurrencyの維持時間 (1以上, 単位：秒 or 分)

aws

In Partnership with intel

その8: Distributed Load Testing on AWS v1.1

負荷テストの実行状況や結果は、詳細画面にて確認可能



シナリオ実行状況

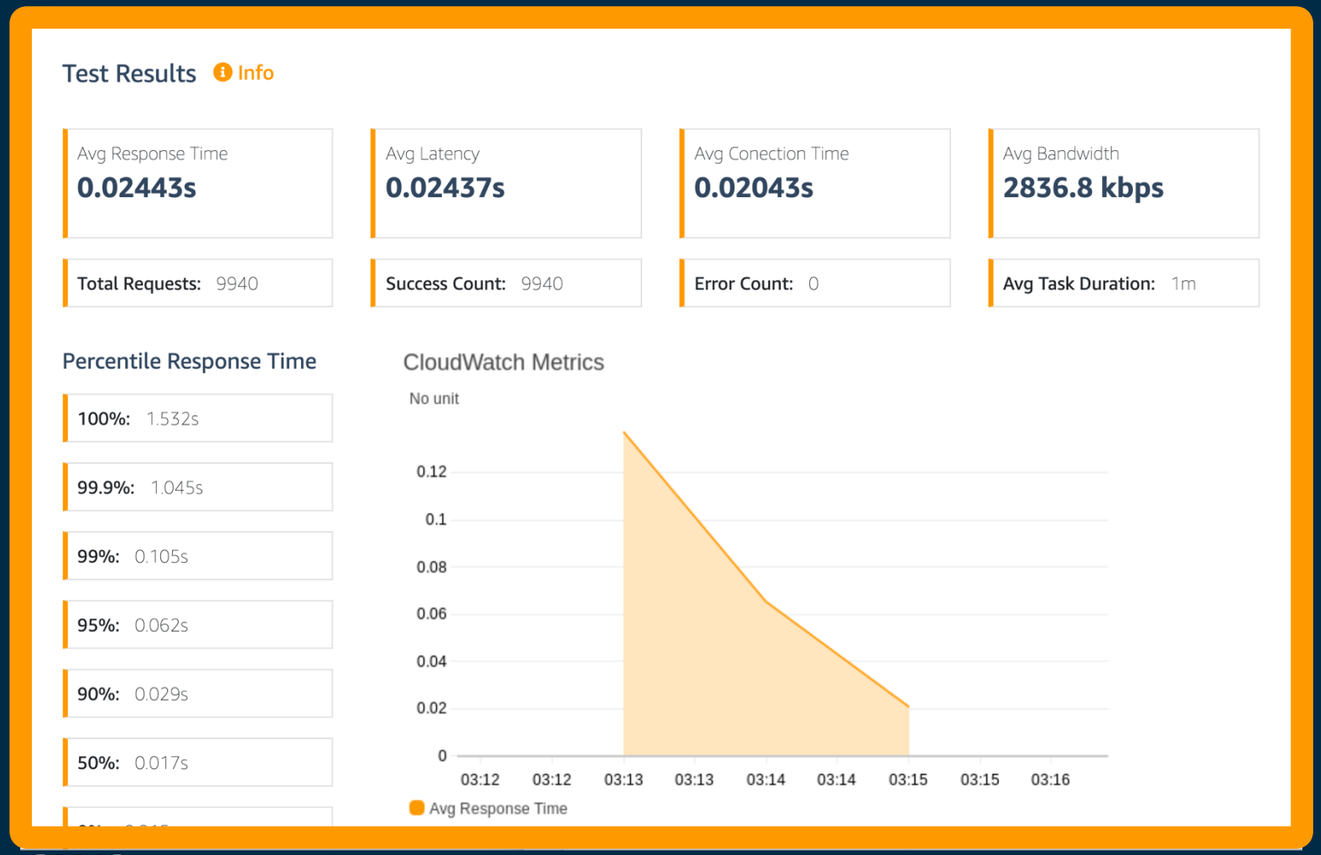
Fargateタスクの状況

TaskCount	シナリオ上タスク数
Provisioning Count	Fargate要求中の数
Pending Count	タスク起動待ち数
Running Count	タスク起動済み数

In Partnership with intel

その8: Distributed Load Testing on AWS v1.1

負荷テストの実行状況や結果サマリーは、詳細画面にて確認可能



Results History

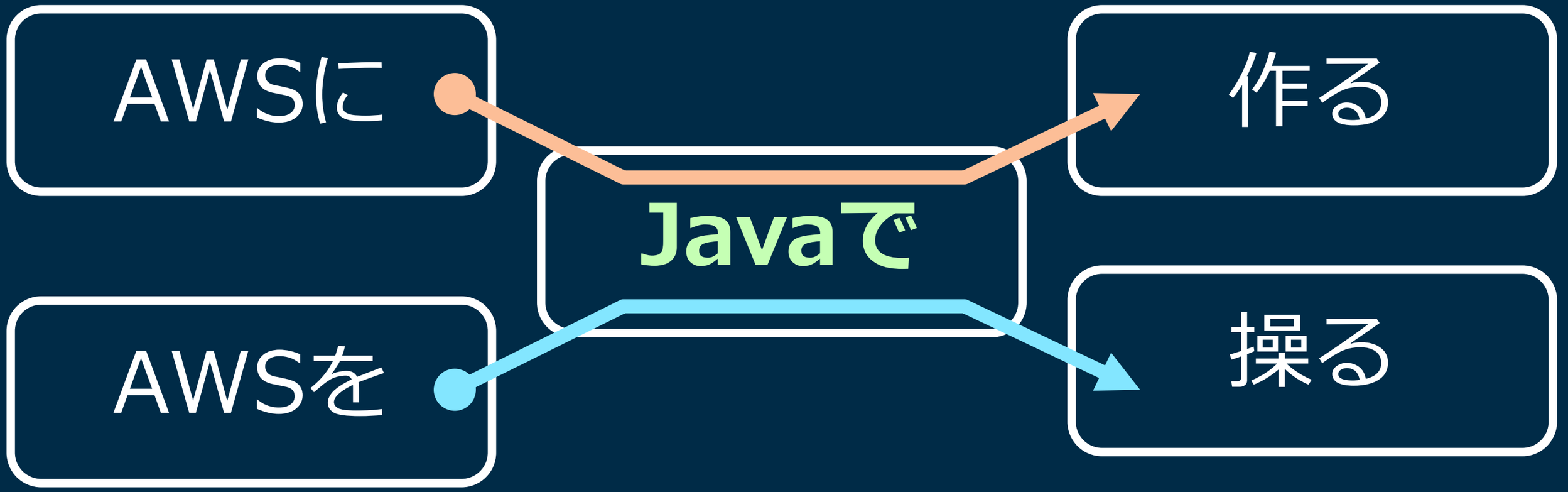
RunTime	AvgRt	AvgLt	AvgCt	100%	99.9%	99.0%	95%	90%	50%	0%
2020-10-16 03:16:24	0.02443	0.02437	0.02043	1.532	1.045	0.105	0.062	0.029	0.017	0.015
2020-10-16 06:06:53	0.02569	0.02565	0.02141	1.377	0.316	0.126	0.078	0.045	0.017	0.015

CloudWatchの
Avg Response Time
メトリクスグラフと共に
パーセンタイルなどの
結果が表示

In Partnership with intel

まとめ

Javaという切り口で、2つの目線から
AWS のサービスやツールをご紹介しました



In Partnership with intel

まとめ

AWSに Javaで 作る

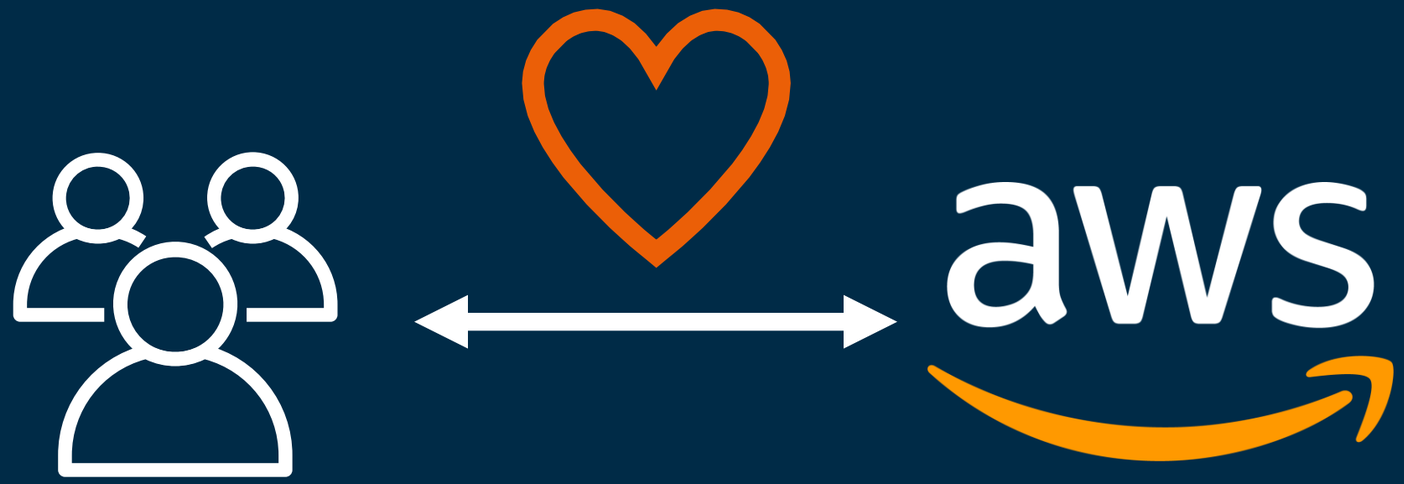
- AWS Toolkit
- X-Ray Auto-Instrumentation Agent for Java
- Lambda Power Tools for Java
- AWS Lambda & Java
- Amazon Corretto

AWSを Javaで 操る

- Distributed Load Testing on AWS
- AWS CDK (Java)
- サービス固有のJavaライブラリ群
- AWS SDK for Java (v2.0)

In Partnership with intel

まとめ



Java Developers

In Partnership with intel

aws DEV DAY
ONLINE JAPAN

Thank you!

In Partnership with intel