イチから理解する
サーバーレスアプリケーション開発

サーバーレス開発環境とテスト

Atsushi Fukui
Senior Solutions Architect, Serverless Specialist

Amazon Web Services Japan K.K.
2020/08/27

自己紹介

- **名前**
  - 福井 厚（ふくい あつし）fatsushi@
- **所属**
  - アマゾン ウェブ サービス ジャパン株式会社
  - 技術統括本部レディネス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
サーバーレス スペシャリスト
- **関心領域**
  - ソフトウェア アーキテクチャ、オブジェクト指向設計、アジャイル開発
- **好きなAWSサービス**
  - サーバーレステクノロジー全般、 AWS Code シリーズ、AWS Amplify

Agenda

- 開発者のためのAWS Developer Tools
- AWS Lambdaのマネージメントコンソール(おさらい)
- Serverless Application Model(SAM)
- AWS Toolkit + SAM
- SAM CLI
- サーバーレスCI/CD
- サーバーレスアプリケーションのテスト
- まとめ

開発者のための
AWS Developer Tools

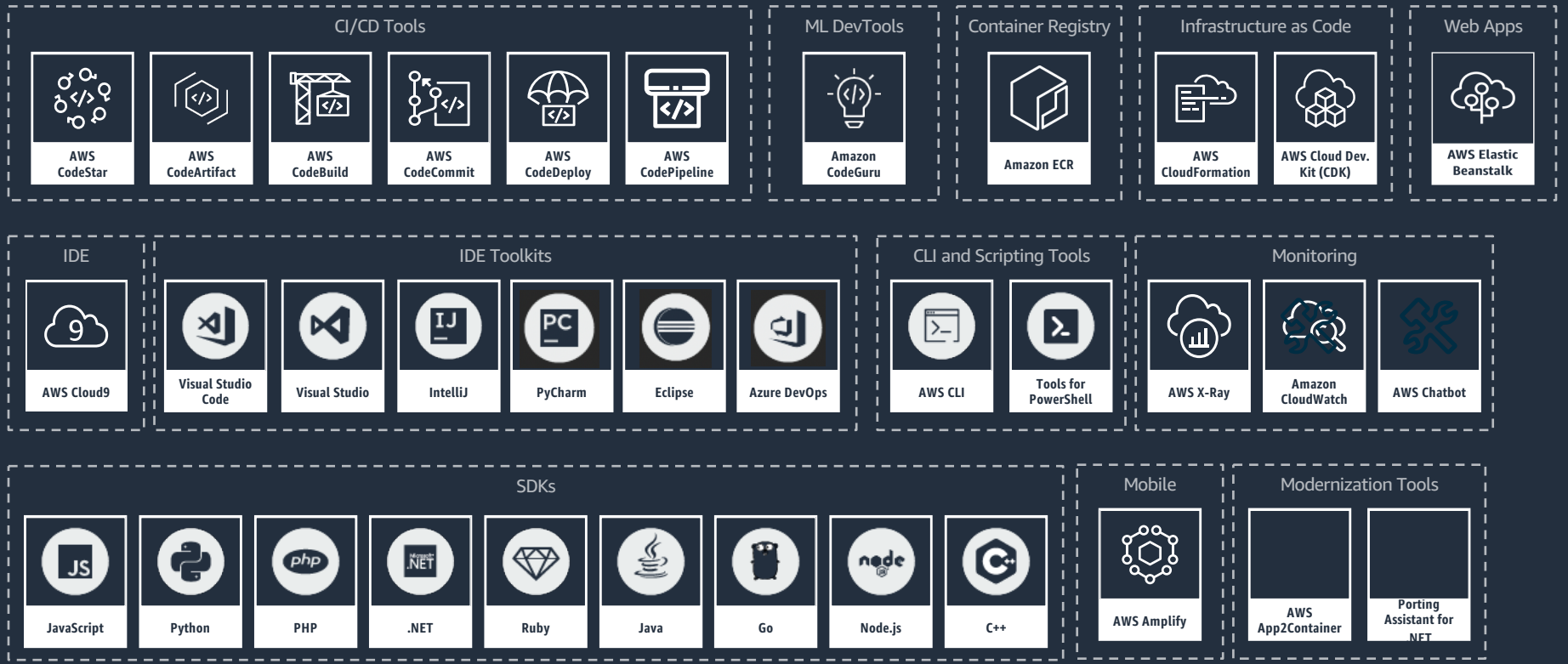
AWS Developer Tools

- **モダンアプリケーション開発をサポート:** 開発チームに対してモダンなソフトウェアやベストプラクティスに従った高品質なソフトウェアをより高速にデリバリーできるように支援

- **学習曲線を短縮:** 開発者が慣れ親しんだ環境、言語、ツールをサポート

- **開発者とチームを繋げる:** ソフトウェア開発におけるチームのコラボレーションを支援

AWS Developer Tools



AWS Lambda
マネージメントコンソール

Demo ー マネジメントコンソールによる操作

AWS Lambda関数の作成と実行デモ

- 関数の作成
- テスト実行
- モニタリング ー CloudWatchのログを表示
- Monitoring tools ー X-Rayの表示

Serverless Application Model (SAM)

SAM：コードによる実行環境の構築

- マネージメントコンソールによる関数作成の課題
  - 手元のソースコードとのマッピングができない
  - 手作業によるミスの可能性
  - 関数の増加に伴い管理が困難に
- Infrastructure as Codeによる自動化のメリット
  - 本番環境と同じ環境を検証環境に構築できる
  - リポジトリによるバージョン管理が可能
  - ミスなく迅速にリリース可能

aws

LambdaのデプロイにはAWS Serverless Application Model (SAM)が利用可能

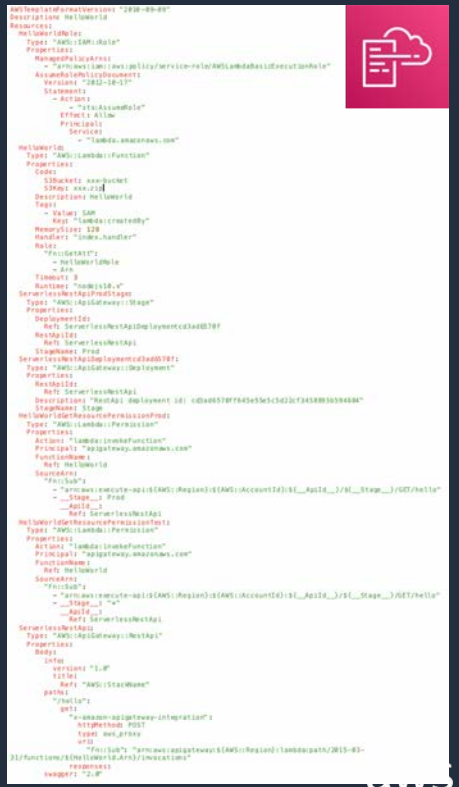
- AWS上のサーバーレスアプリケーションを構築するためのオープンソース フレームワーク
- 関数、API、データベース、イベントソースマッピングを表現する簡易な文法を使用
- デプロイ時にSAMの文法をAWS CloudFormationの文法に変換、展開
- すべてのAWS CloudFormationリソースタイプをサポート

https://aws.amazon.com/serverless/sam/

SAMでより簡潔にサーバーレス アプリを定義できる

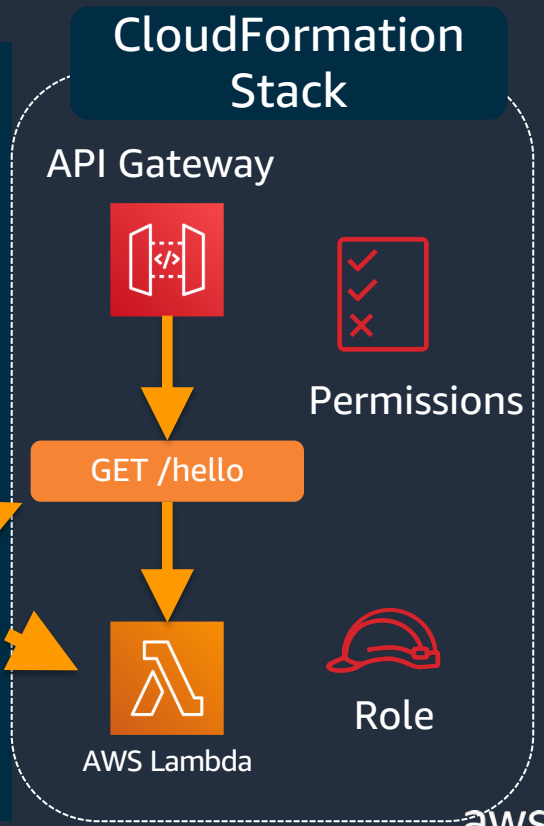

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: HelloWorld
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: python3.8
      CodeUri: src/handlers/func1
      Description: HelloWorld
      MemorySize: 128
      Timeout: 3
      Events:
        GetResource:
          Type: Api
          Properties:
            Path: /hello
            Method: get
```

AWS CloudFormation

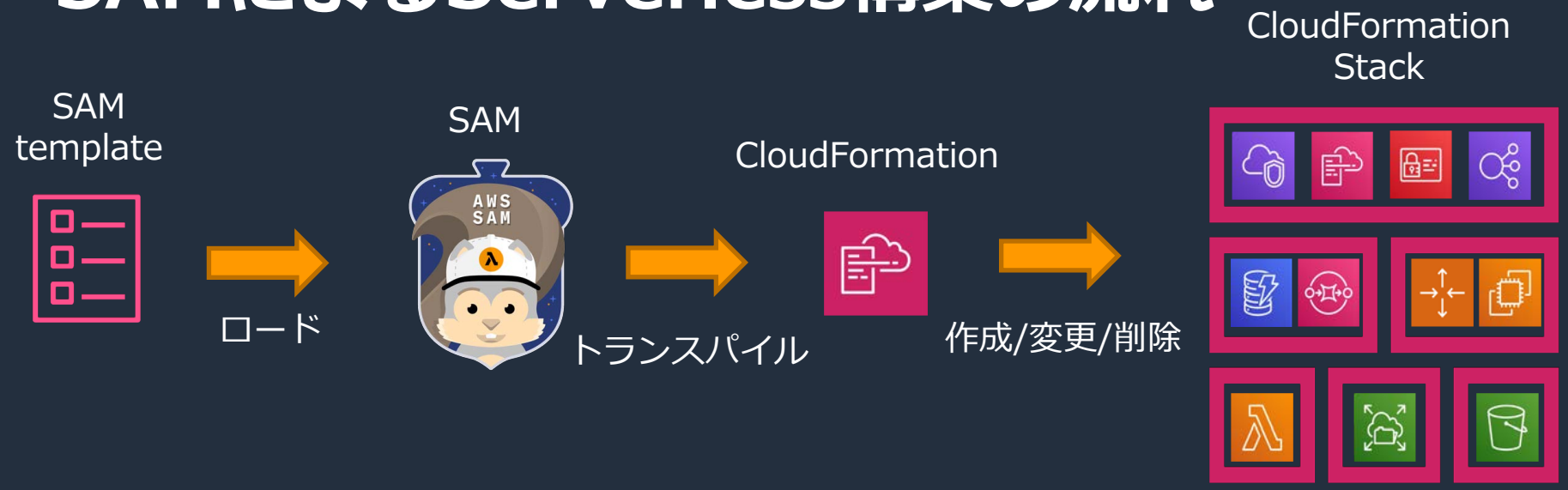


SAMでより簡潔にサーバーレス アプリを定義できる


```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: HelloWorld
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: python3.8
      CodeUri: src/handlers/func1
      Description: HelloWorld
      MemorySize: 128
      Timeout: 3
      Events:
        HelloApi
          Type: Api
          Properties:
            Path: /hello
            Method: get
```



SAMによるServerless構築の流れ



リソースの定義
パラメータの定義

スタックの作成/変更/削除
エラー検知とロールバック

AWSリソース

SAMを使いやすく

IDE + AWS Toolkit

SAM CLI

AWS Toolkit

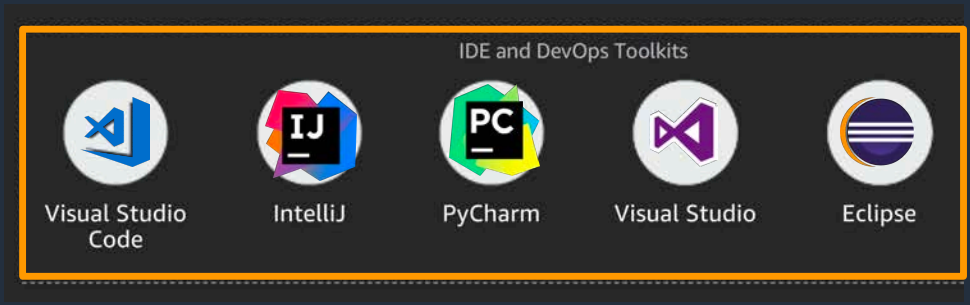
AWS Toolkit

- オープンソースプラグイン
- AWS上でのアプリケーションの作成、デバッグ、デプロイを容易に
- ステップ実行によるデバッグ、およびIDE からのデプロイを含む、サーバーレスアプリケーションの統合開発環境を提供

AWS Toolkit

AWS Toolkit

AWS Toolkit 対応 IDE



- 雛形からプロジェクトを作成
- ステップ実行などのデバッグ支援
- IDE/エディタからのデプロイ

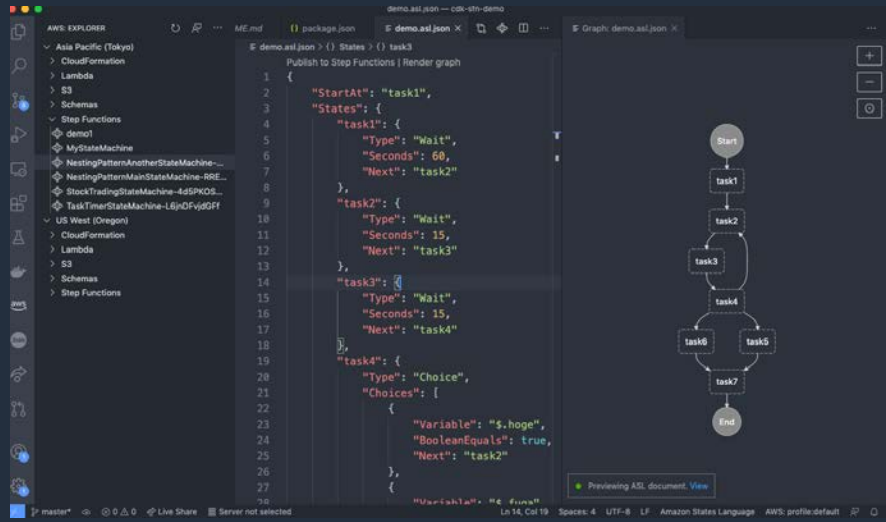
AWS Toolkit for Visual Studio Code

AWS Toolkit for Visual Studio Codeとは

- Visual Studio Code用のオープンソースプラグイン
  https://github.com/aws/aws-toolkit-vscode
- AWS上でのアプリケーションの作成、デバッグ、デプロイを容易に
- サーバーレスアプリケーションテンプレート
- サーバーレスアプリケーションをローカルでデバッグ
- IDEからサーバーレスアプリケーションをデプロイ
- 詳細はUser Guideを参照
  https://docs.aws.amazon.com/ja_jp/toolkit-for-vscode/latest/userguide/welcome.html

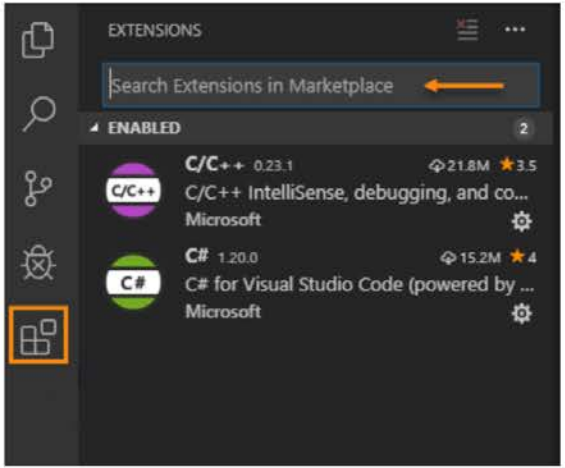
AWS Step Functionsをサポート New!

- AWS Toolkit を使用してStep Functions ステートマシンの開発を加速
- ステートマシンのビルド時にステートマシンを
  リアルタイムで視覚化
- Amazon States Language の言語
  パーサーがステートマシンの定義の
  構文解析、エラーを強調表示
- ステートマシンを作成、更新、
  実行可能



VS Code Toolkit のインストール

1. VS Code エディタを起動します。
2. VS Code エディタの側にある **[Activity Bar (アクティビティっバー)]** で、**[Extensions (拡張機能)]** アイコンを選択します。これにより、**[Extensions (拡張機能)]** ビューが開き、VS Code Marketplace にアクセスできます。



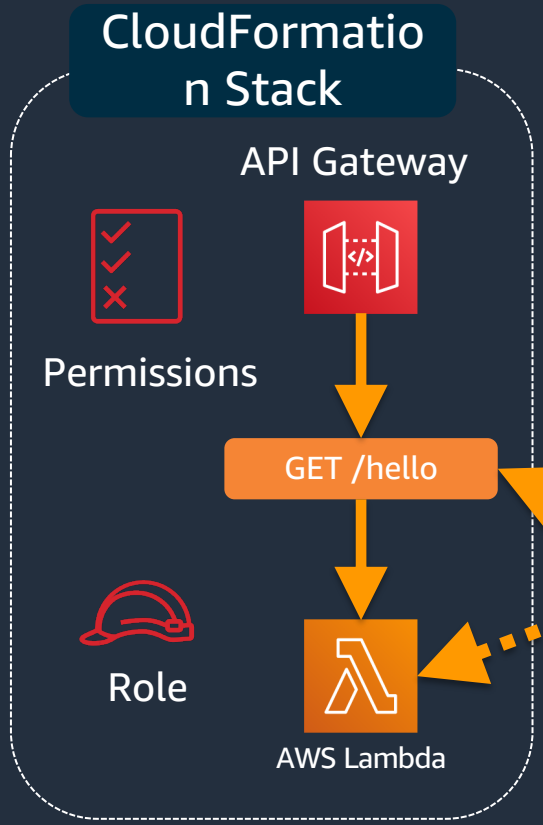
- https://github.com/aws/aws-toolkit-vscode

Demo — AWS Toolkit for Visual Studio Code

AWS Lambda関数のデプロイ デモ

- 新規SAMアプリケーションの作成
- SAMテンプレートの確認
- SAMアプリケーションのデプロイ
- デプロイ結果の確認

Deployされた アーキテクチャとの対比




```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Descript
    Sem

Resources:
    HelloWorldFunction:
        Type: AWS::Serverless::Function
        Properties:
            CodeUri: hello_world/
            Handler: app.lambda_handler
            Runtime: python3.8
            Events:
                HelloWorld:
                    Type: Api
                    Properties:
                        Path: /hello
                        Method: get
```

アーキテクチャとの対応

参考情報：Debug Panel からのデバッグ

- Debugging Node.js Lambda Functions https://github.com/aws/aws-toolkit-vscode/blob/master/docs/debugging-nodejs-lambda-functions.md
- Debugging Python Lambda Functions https://github.com/aws/aws-toolkit-vscode/blob/master/docs/debugging-python-lambda-functions.md
- Debugging .NET Core Lambda Functions https://github.com/aws/aws-toolkit-vscode/blob/master/docs/debugging-dotnet-lambda-functions.md

aws

SAM CLI

SAMを使いやすくしよう！

IDE + AWS Toolkit

SAM CLI

SAM CLI

- SAMをコマンドラインで実行
- スクリプトによる自動化が可能
- 自動化により手作業によるミスを防ぐ

AWS SAM CLI

1) アプリケーションのひな形を生成

```
$ sam init
```

2) ローカル環境でビルド

```
$ cd sam-app
$ sam build
```

3) ビルド後にAWS へデプロイ

```
$ sam deploy --guided
```

※sam deploy の際に必要な S3 バケットは自動生成

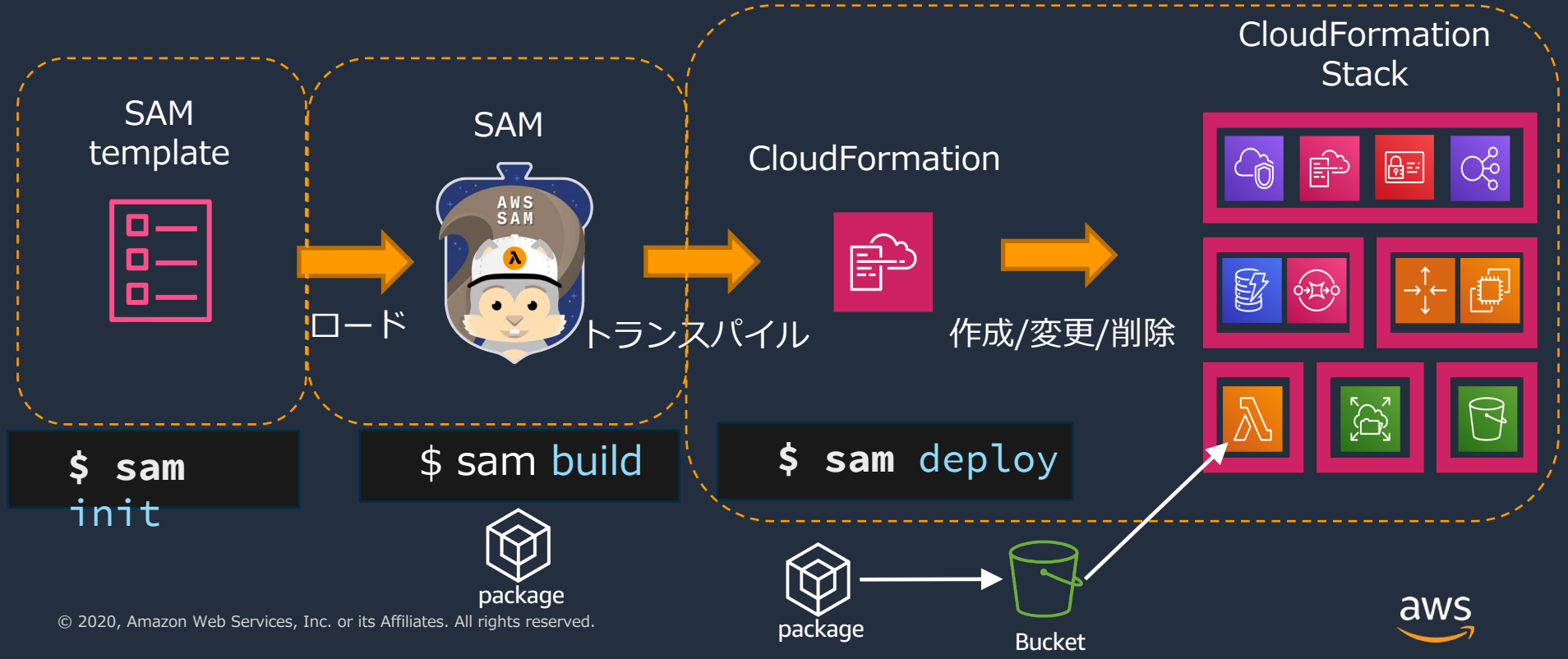
SAM CLI

Demo — SAM CLI

SAM CLIのデモ

- 新規SAMアプリケーションの作成
- プロジェクトファイルの構成確認
- SAMテンプレートの確認
- SAMアプリケーションのデプロイ
- デプロイ結果の確認

SAMによるServerless構築の流れ



CI/CD for Serverless

Serverless Application Pipeline

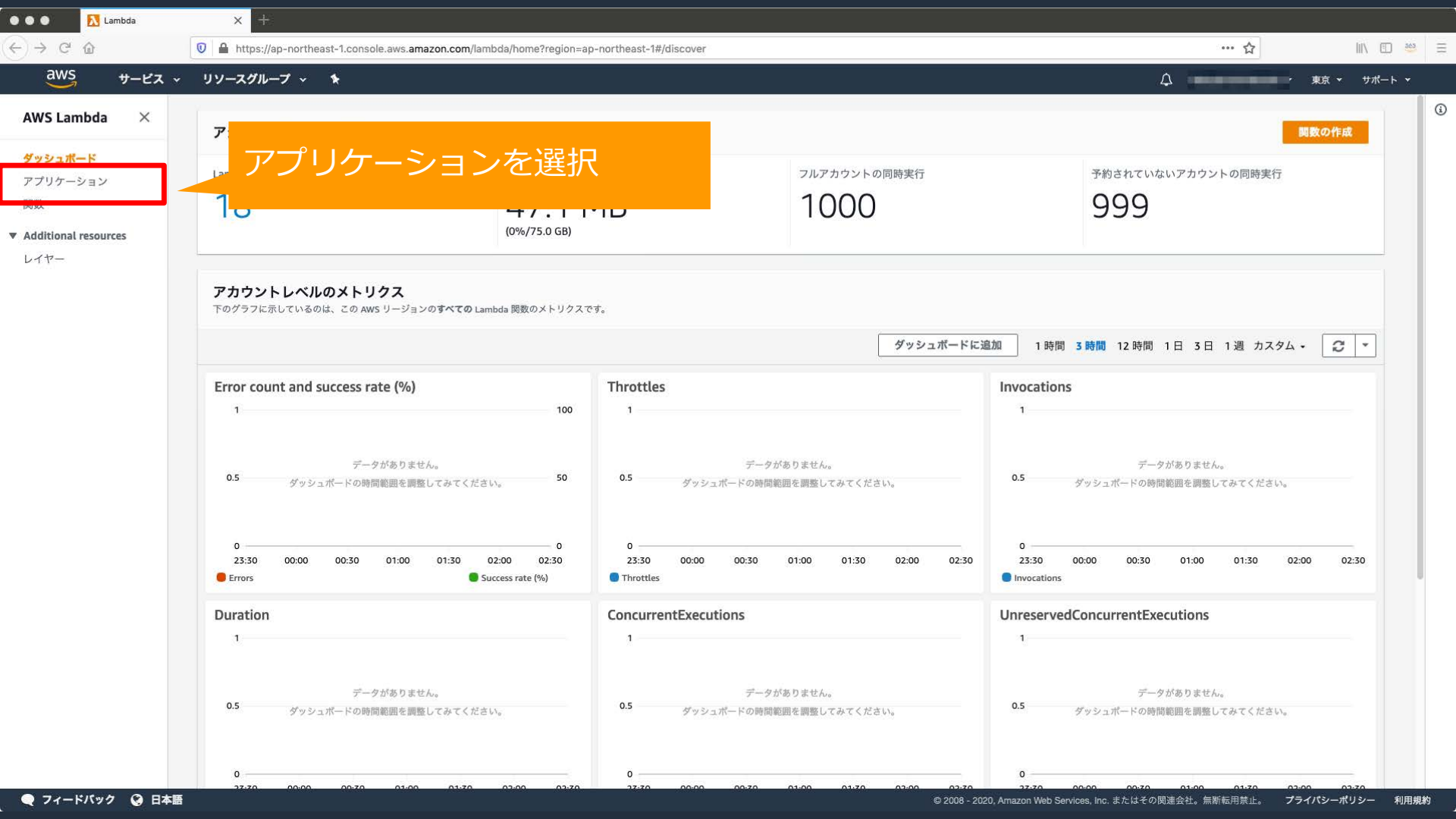
Serverless Application Pipeline

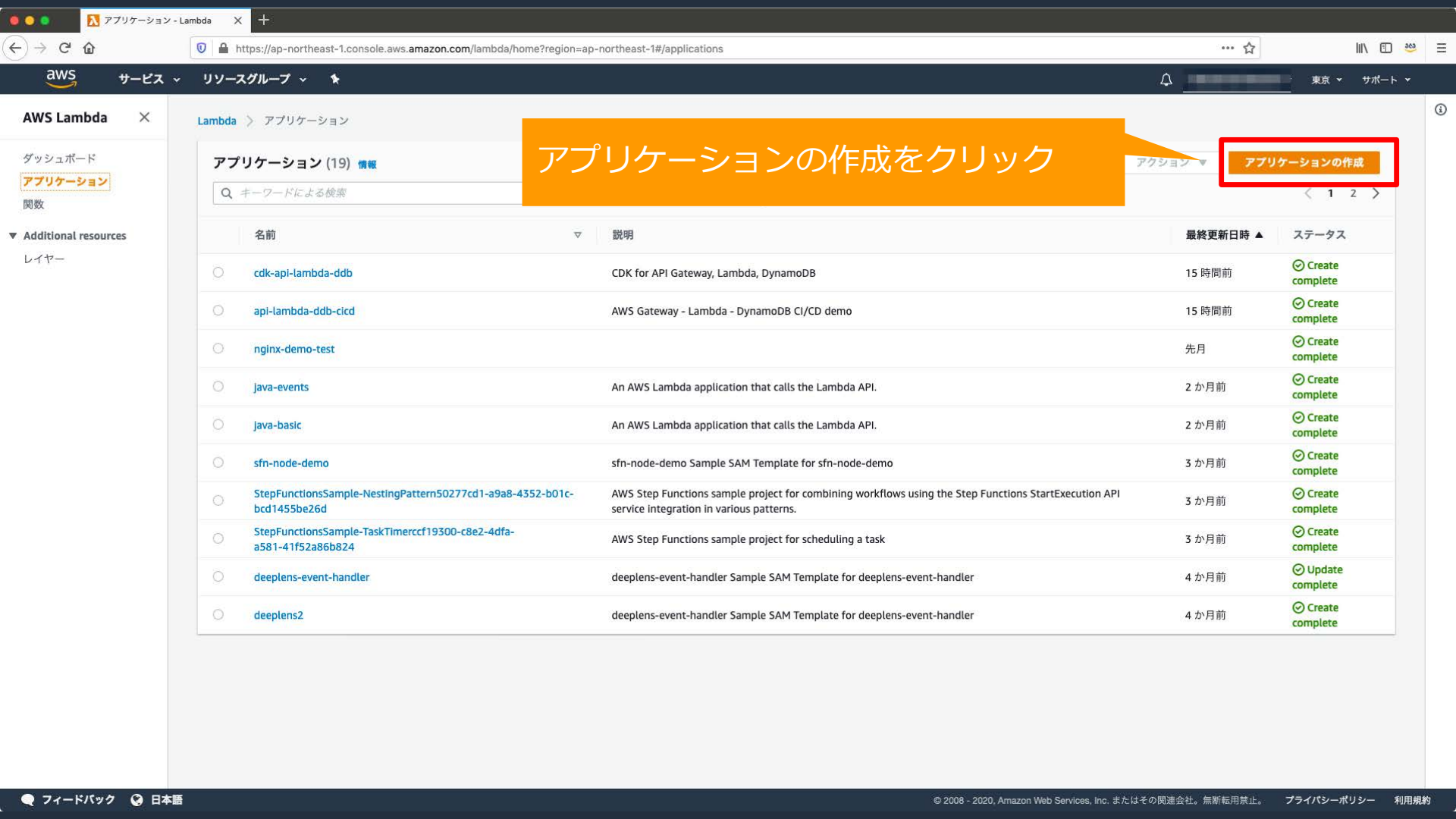
- AWS Lambdaのマネジメントコンソールからアプリケーションの CI/CD環境を構築
  - 構築方法を選択
    - サンプルアプリケーション/AWS Serverless Application Repoitory / 一から作成
  - コードリポジトリを選択
    - CodeCommit / GitHub
- CloudFormationスタックの生成と実行
  - CI/CD環境用スタック
  - Lambda関数デプロイ用スタック
- パイプラインを使用した継続的インテグレーション/デプロイメント

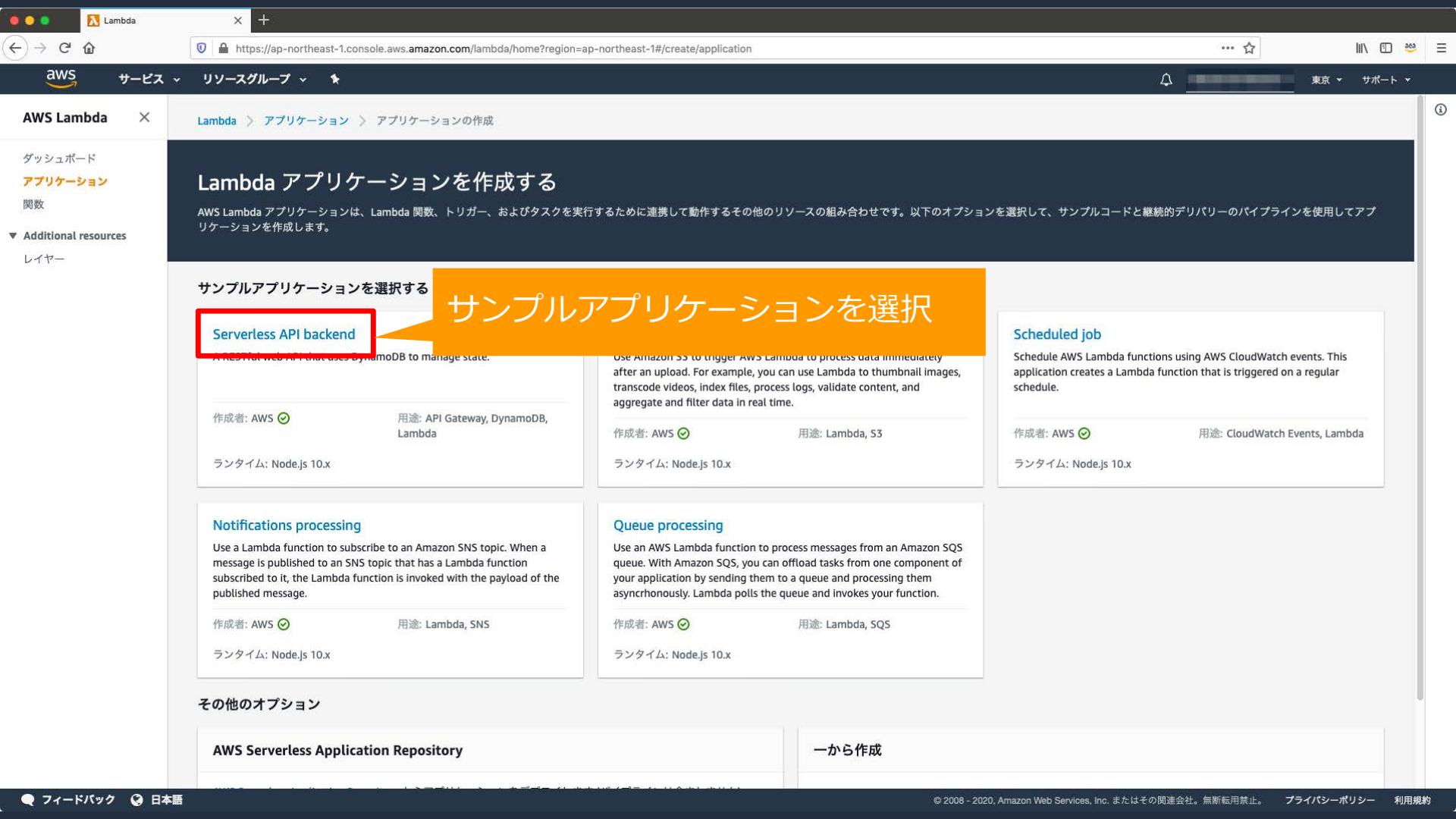
Demo —
Serverless Application Pipeline

Serverless Application Pipelineのデモ

- サンプルアプリケーションの選択
- CI/CD環境の構築
- コードリポジトリの確認
- パイプラインの確認
- モニタリングの確認


Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/discover
aws
サービス
リソースグループ
東京
サポート
AWS Lambda
ダッシュボード
アプリケーション
関数
Additional resources
レイヤー
アプリケーションを選択
関数の作成
フルアカウントの同時実行
1000
予約されていないアカウントの同時実行
999
(0%/75.0 GB)
アカウントレベルのメトリクス
下のグラフに示しているのは、この AWS リージョンのすべての Lambda 関数のメトリクスです。
ダッシュボードに追加
1 時間 3 時間 12 時間 1 日 3 日 1 週 カスタム
Error count and success rate (%)
Throttles
Invocations
Duration
ConcurrentExecutions
UnreservedConcurrentExecutions
データがありません。
ダッシュボードの時間範囲を調整してみてください。
Errors
Success rate (%)
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約


アプリケーション - Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/applications
aws
サービス
リソースグループ
東京
サポート
AWS Lambda
ダッシュボード
アプリケーション
関数
Additional resources
レイヤー
Lambda > アプリケーション
アプリケーションの作成をクリック
アプリケーション (19) 情報
キーワードによる検索
アクション
アプリケーションの作成
1 2
名前
説明
最終更新日時
ステータス
cdk-api-lambda-ddb
CDK for API Gateway, Lambda, DynamoDB
15 時間前
Create complete
api-lambda-ddb-cicd
AWS Gateway - Lambda - DynamoDB CI/CD demo
15 時間前
Create complete
nginx-demo-test
先月
Create complete
java-events
An AWS Lambda application that calls the Lambda API.
2 か月前
Create complete
java-basic
An AWS Lambda application that calls the Lambda API.
2 か月前
Create complete
sfn-node-demo
sfn-node-demo Sample SAM Template for sfn-node-demo
3 か月前
Create complete
StepFunctionsSample-NestingPattern50277cd1-a9a8-4352-b01c-bcd1455be26d
AWS Step Functions sample project for combining workflows using the Step Functions StartExecution API service integration in various patterns.
3 か月前
Create complete
StepFunctionsSample-TaskTimerccf19300-c8e2-4dfa-a581-41f52a86b824
AWS Step Functions sample project for scheduling a task
3 か月前
Create complete
deeplens-event-handler
deeplens-event-handler Sample SAM Template for deeplens-event-handler
4 か月前
Update complete
deeplens2
deeplens-event-handler Sample SAM Template for deeplens-event-handler
4 か月前
Create complete
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約


Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/create/application
aws
サービス
リソースグループ
東京
サポート
AWS Lambda
ダッシュボード
アプリケーション
関数
Additional resources
レイヤー
Lambda > アプリケーション > アプリケーションの作成
Lambda アプリケーションを作成する
AWS Lambda アプリケーションは、Lambda 関数、トリガー、およびタスクを実行するために連携して動作するその他のリソースの組み合わせです。以下のオプションを選択して、サンプルコードと継続的デリバリーのパイプラインを使用してアプリケーションを作成します。
サンプルアプリケーションを選択する
サンプルアプリケーションを選択
Serverless API backend
作成者: AWS
用途: API Gateway, DynamoDB, Lambda
ランタイム: Node.js 10.x
after an upload. For example, you can use Lambda to thumbnail images, transcode videos, index files, process logs, validate content, and aggregate and filter data in real time.
作成者: AWS
用途: Lambda, S3
ランタイム: Node.js 10.x
Scheduled job
Schedule AWS Lambda functions using AWS CloudWatch events. This application creates a Lambda function that is triggered on a regular schedule.
作成者: AWS
用途: CloudWatch Events, Lambda
ランタイム: Node.js 10.x
Notifications processing
Use a Lambda function to subscribe to an Amazon SNS topic. When a message is published to an SNS topic that has a Lambda function subscribed to it, the Lambda function is invoked with the payload of the published message.
作成者: AWS
用途: Lambda, SNS
ランタイム: Node.js 10.x
Queue processing
Use an AWS Lambda function to process messages from an Amazon SQS queue. With Amazon SQS, you can offload tasks from one component of your application by sending them to a queue and processing them asyncrhonously. Lambda polls the queue and invokes your function.
作成者: AWS
用途: Lambda, SQS
ランタイム: Node.js 10.x
その他のオプション
AWS Serverless Application Repository
一から作成
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

Lambda

https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/create/application/view?applicationId=web-backend

aws サービス リソースグループ 東京 サポート

AWS Lambda

- ダッシュボード
- アプリケーション
- 関数
- Additional resources
- レイヤー

Serverless API backend

A RESTful web API that uses DynamoDB to manage state.

作成者: AWS

用途: API Gateway, DynamoDB, Lambda

ランタイム: Node.js 10.x

アーキテクチャ

ソースコード

使用したサービス

ソース管理 CodeCommit または GitHub	継続的デリバリー CodePipeline	アプリケーションのリソース API Gateway DynamoDB Lambda
構築およびテスト CodeBuild	デプロイ AWS CloudFormation	

開発ワークフロー

次のページでアプリケーションを設定して、開始します。

1 選択
このサンプルアプリケーションを使用するか、前のページに戻ります。

2 作成
設定を構成し、アプリケーションのリソースを作成します。

3 クローン作成
ローカル開発用のアプリケーション
アップツールのクローンを作成します。

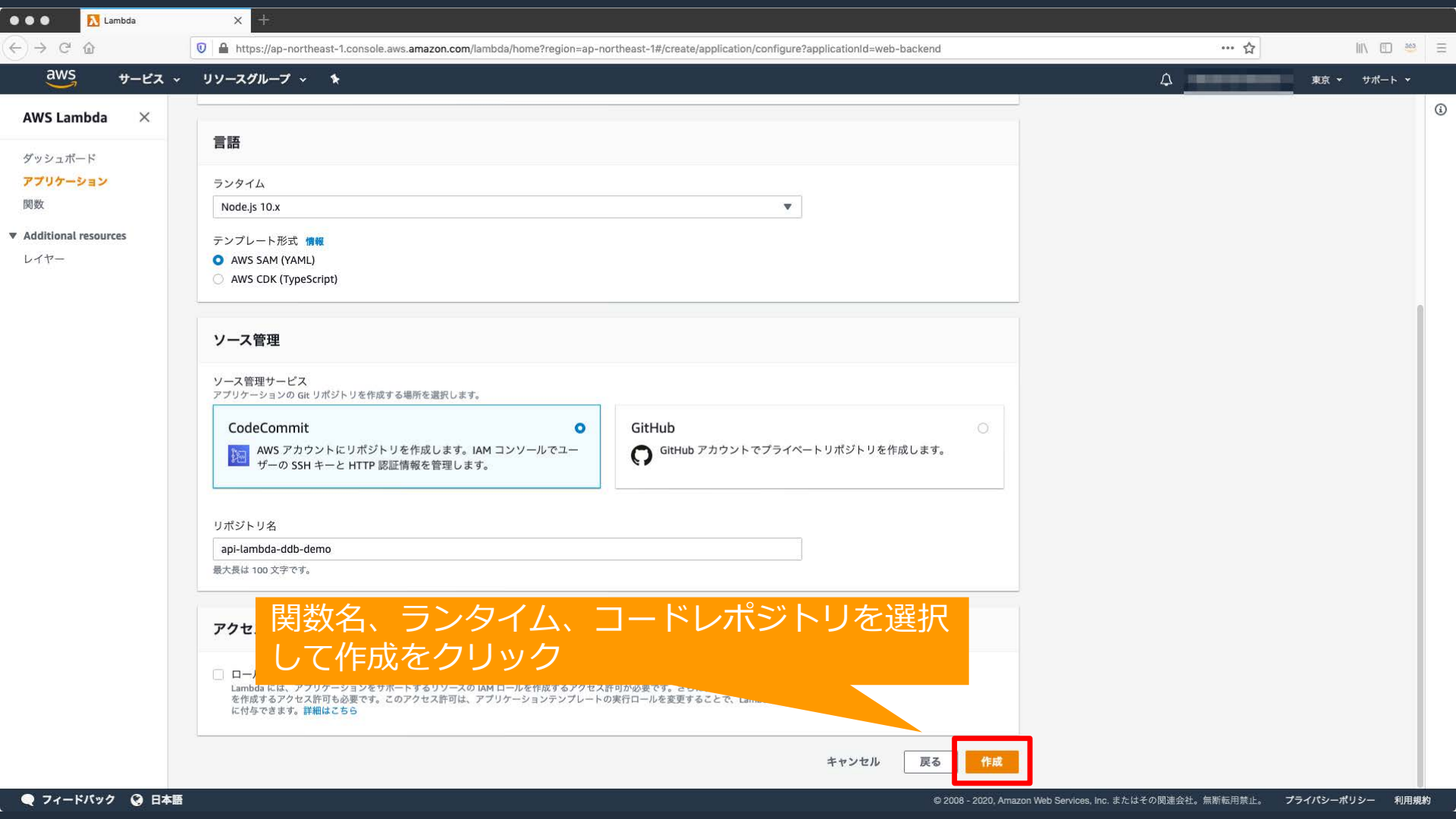
ラインをトリガーします。

次へをクリック

戻る 次へ

フィードバック 日本語

プライバシーポリシー 利用規約


Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/create/application/configure?applicationId=web-backend
aws
サービス
リソースグループ
東京
サポート
AWS Lambda
ダッシュボード
アプリケーション
関数
Additional resources
レイヤー
言語
ランタイム
Node.js 10.x
テンプレート形式 情報
AWS SAM (YAML)
AWS CDK (TypeScript)
ソース管理
ソース管理サービス
アプリケーションの Git リポジトリを作成する場所を選択します。
CodeCommit
AWS アカウントにリポジトリを作成します。IAM コンソールでユーザーの SSH キーと HTTP 認証情報を管理します。
GitHub
GitHub アカウントでプライベートリポジトリを作成します。
リポジトリ名
api-lambda-ddb-demo
最大長は 100 文字です。
関数名、ランタイム、コードレポジトリを選択して作成をクリック
キャンセル
戻る
作成
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

AWS Lambda
ダッシュボード
アプリケーション
関数
Additional resources
レイヤー

api-lambda-ddb-cicd

概要 | コード | デプロイ | モニタリング

▼ 開始方法

サーバーレスアプリケーション

アプリケーションリポジトリには、アプリケーションを定義する関数コード、テンプレート、およびビルド設定が含まれています。その構造の概要と開始手順については、 README を表示します。**Code タブには、ローカル開発およびテスト用のソースリポジトリとツールに関する情報が表示されます。アプリケーションからメトリクスを表示してログを分析するには、 Monitoring を選択します。**

継続的デリバリーインフラストラクチャ

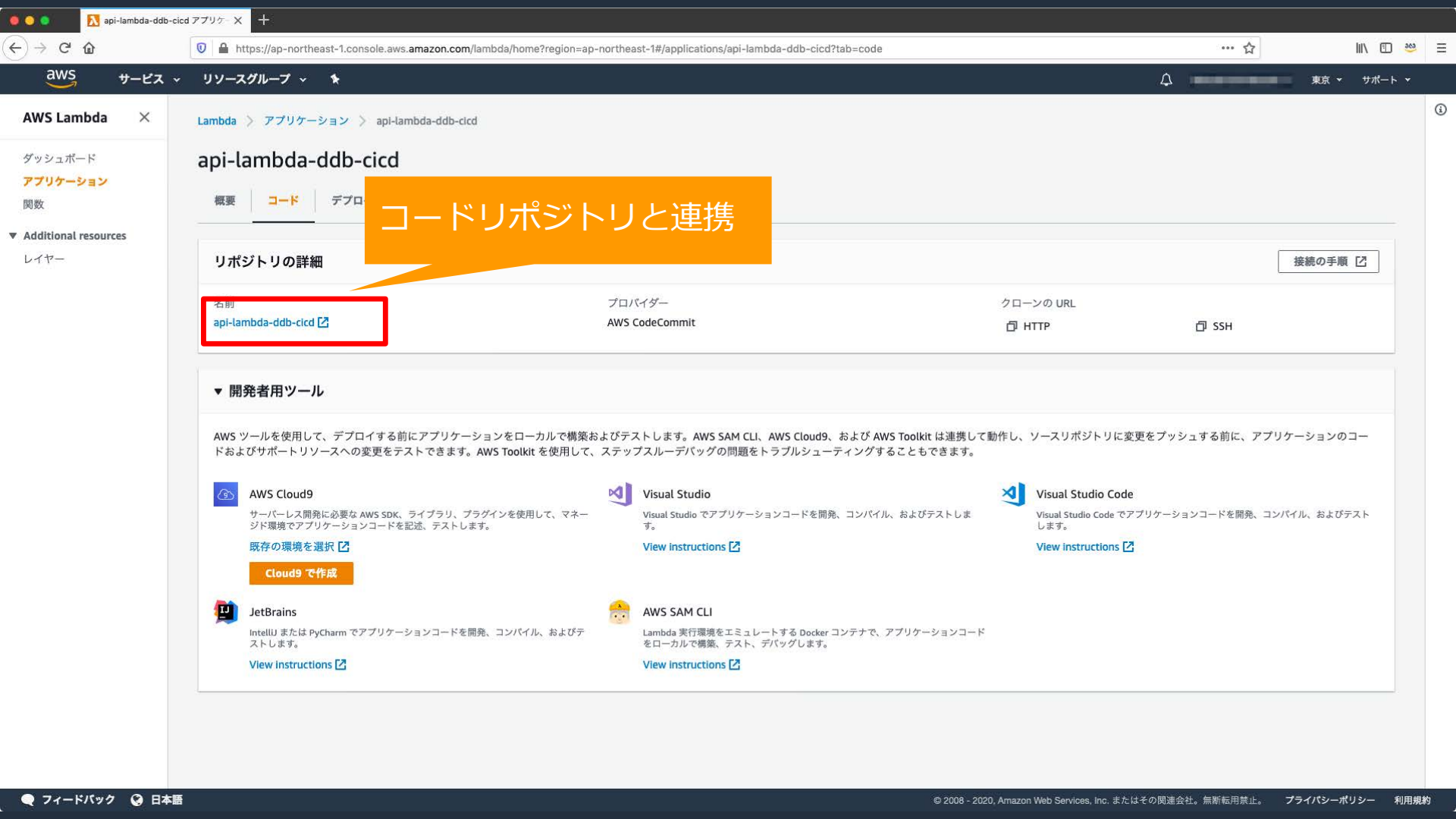
マスターブランチに変更をコミットして AWS CodeCommit または GitHub にプッシュすると、アプリケーションのパイプラインによって変更が自動的にビルド、パッケージ化、デプロイされます。アプリケーションの継続的デリバリーを有効にするサポートリソースのリストについては、以下を参照してください。 **デプロイで**, 、ソース、ビルド、デプロイの各リソースを接続するパイプラインの詳細を確認できます。

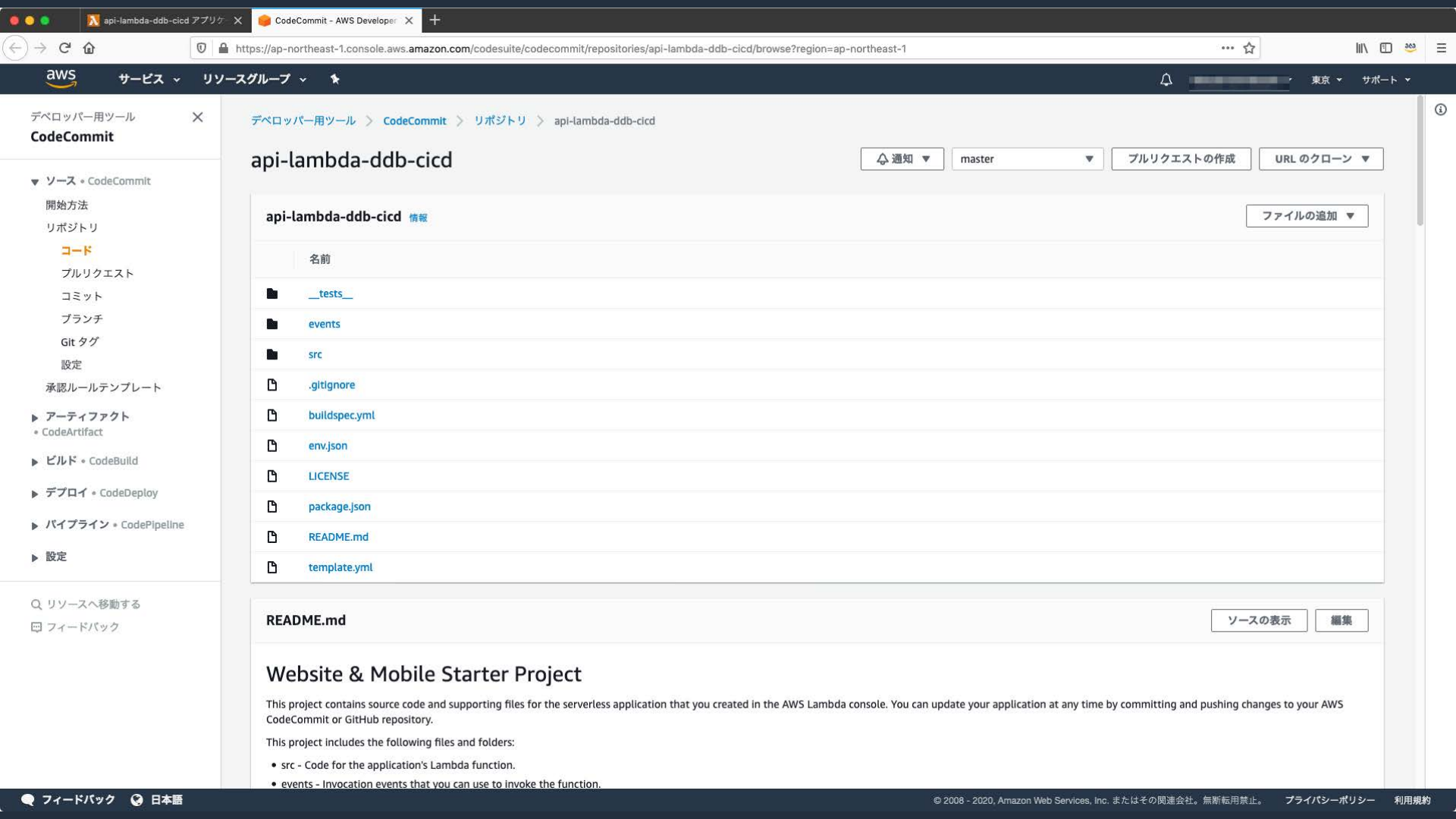
AWS リソースとサービスのアクセス許可を追加する

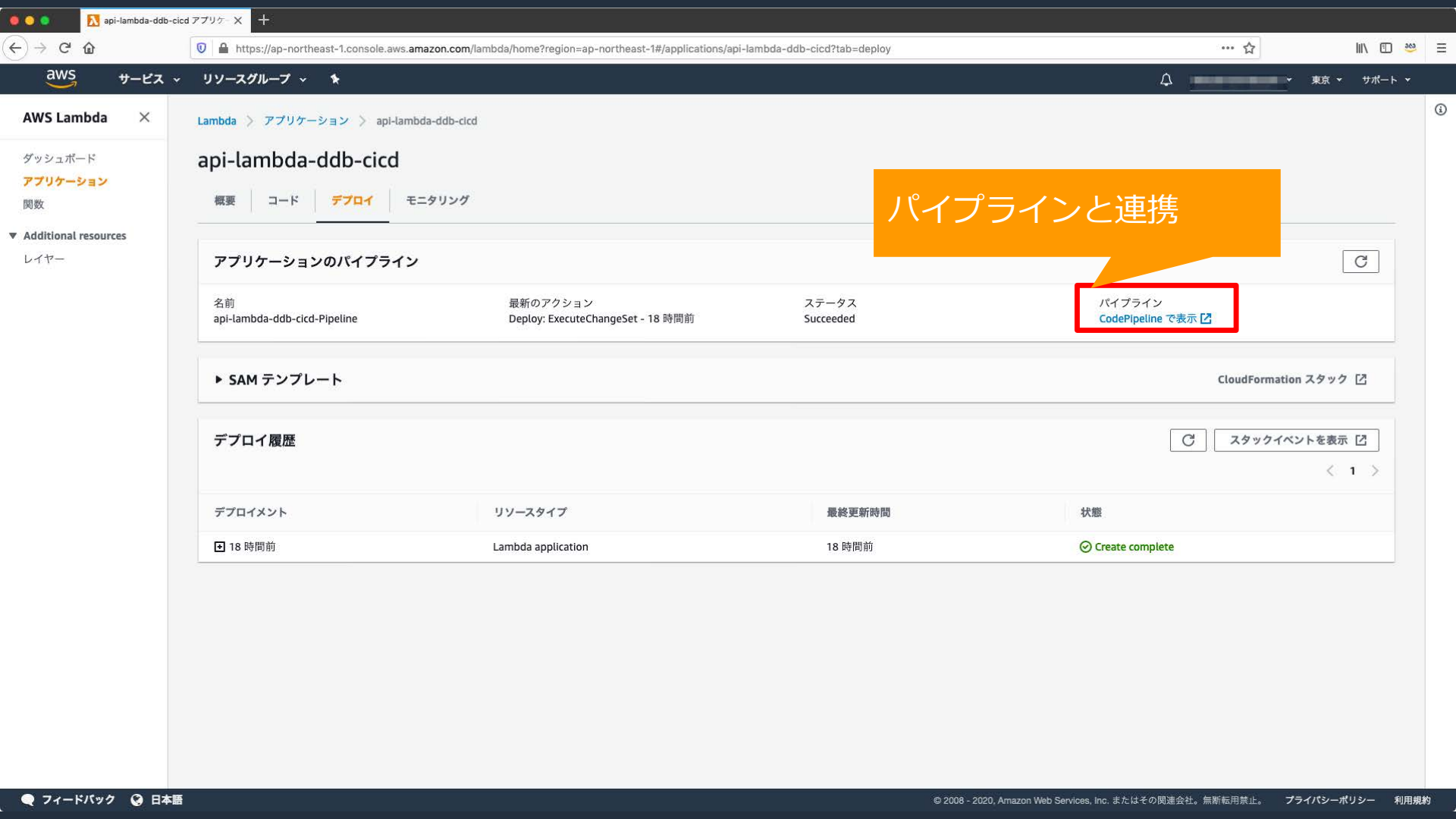
アプリケーションに AWS リソースとトリガーを追加するには、アプリケーションのソースコードに含まれている AWS SAM テンプレートでそれらを定義します。テンプレートは、アプリケーションの関数の実行ロールも定義します。このロールを拡張して、コードからより多くのサービスやリソースにアクセスできます。実行ロールは、Amazon SQS などのサービスから読み取る Lambda イベントソースマッピングへのアクセスも提供します。 詳細

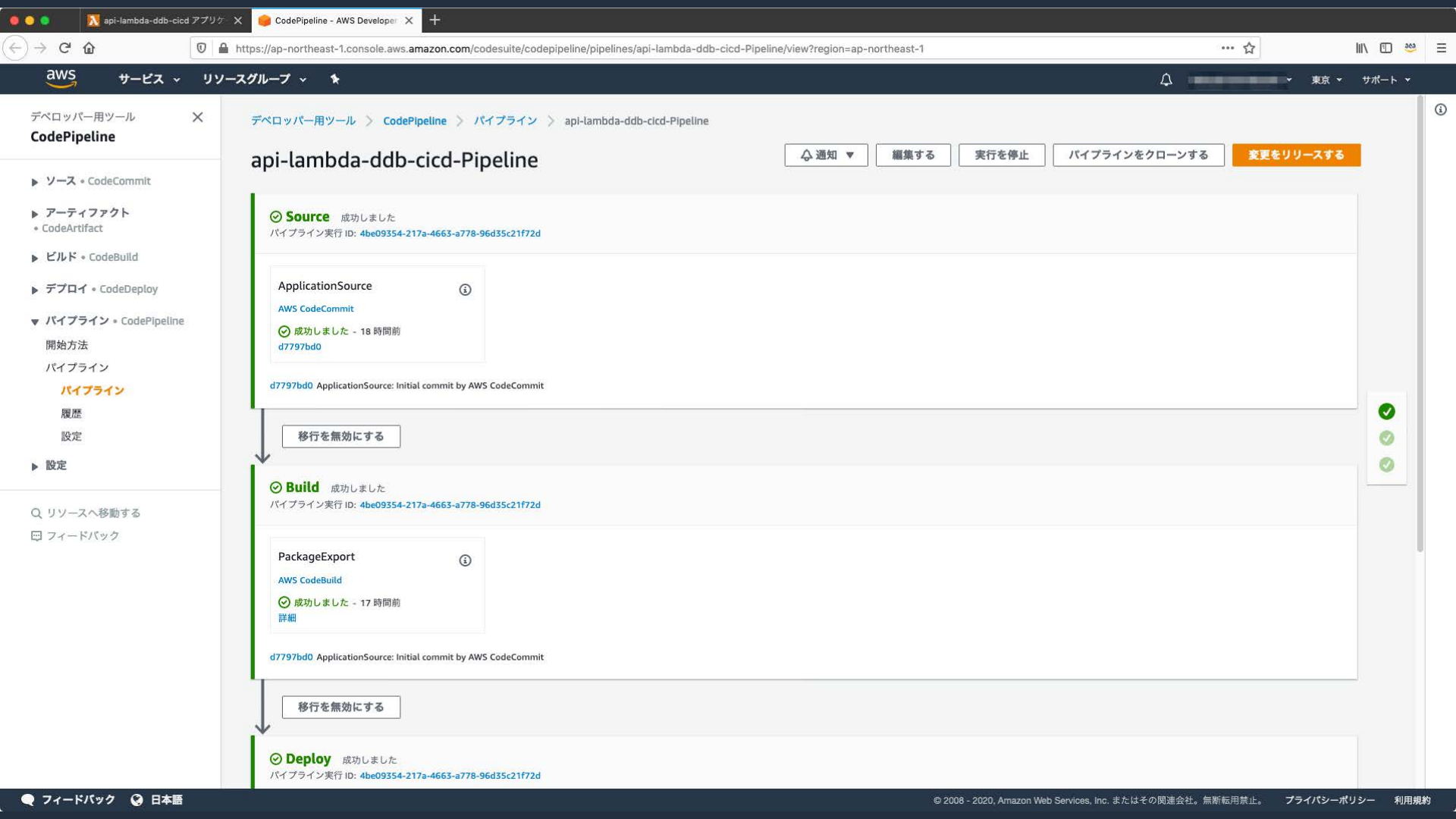
API エンドポイント

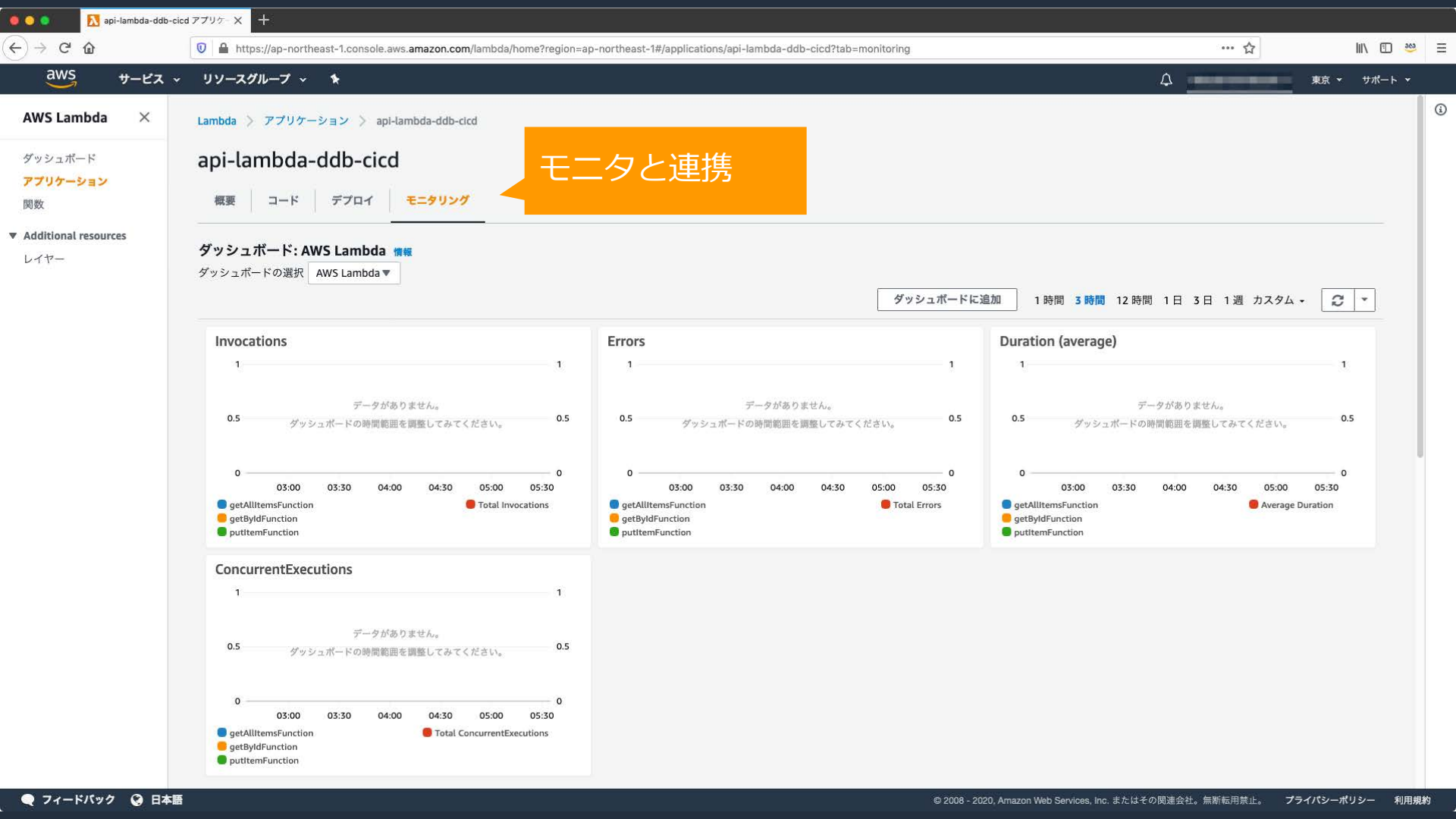
エンドポイント
https://nn6y2e3ay8.execute-api.ap-northeast-1.amazonaws.com/Prod


api-lambda-ddb-cicd アプリケ
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/applications/api-lambda-ddb-cicd?tab=code
aws
サービス
リソースグループ
東京
サポート
AWS Lambda
ダッシュボード
アプリケーション
関数
Additional resources
レイヤー
Lambda > アプリケーション > api-lambda-ddb-cicd
api-lambda-ddb-cicd
概要
コード
デプロ
コードリポジトリと連携
リポジトリの詳細
接続の手順
名前
api-lambda-ddb-cicd
プロバイダー
AWS CodeCommit
クローンの URL
HTTP
SSH
開発者用ツール
AWS ツールを使用して、デプロイする前にアプリケーションをローカルで構築およびテストします。AWS SAM CLI、AWS Cloud9、および AWS Toolkit は連携して動作し、ソースリポジトリに変更をプッシュする前に、アプリケーションのコードおよびサポートリソースへの変更をテストできます。AWS Toolkit を使用して、ステップスルーデバッグの問題をトラブルシューティングすることもできます。
AWS Cloud9
サーバーレス開発に必要な AWS SDK、ライブラリ、プラグインを使用して、マネージド環境でアプリケーションコードを記述、テストします。
既存の環境を選択
Cloud9 で作成
Visual Studio
Visual Studio でアプリケーションコードを開発、コンパイル、およびテストします。
View instructions
Visual Studio Code
Visual Studio Code でアプリケーションコードを開発、コンパイル、およびテストします。
View instructions
JetBrains
IntelliJ または PyCharm でアプリケーションコードを開発、コンパイル、およびテストします。
View instructions
AWS SAM CLI
Lambda 実行環境をエミュレートする Docker コンテナで、アプリケーションコードをローカルで構築、テスト、デバッグします。
View instructions
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約


api-lambda-ddb-cicd アプリケ
CodeCommit - AWS Developer
https://ap-northeast-1.console.aws.amazon.com/codesuite/codecommit/repositories/api-lambda-ddb-cicd/browse?region=ap-northeast-1
aws
サービス
リソースグループ
東京
サポート
デベロッパー用ツール
CodeCommit
ソース • CodeCommit
開始方法
リポジトリ
コード
プルリクエスト
コミット
ブランチ
Git タグ
設定
承認ルールテンプレート
アーティファクト
• CodeArtifact
ビルド • CodeBuild
デプロイ • CodeDeploy
パイプライン • CodePipeline
設定
リソースへ移動する
フィードバック
デベロッパー用ツール > CodeCommit > リポジトリ > api-lambda-ddb-cicd
api-lambda-ddb-cicd
通知
master
プルリクエストの作成
URL のクローン
api-lambda-ddb-cicd 情報
ファイルの追加
名前
__tests__
events
src
.gitignore
buildspec.yml
env.json
LICENSE
package.json
README.md
template.yml
README.md
ソースの表示
編集
Website & Mobile Starter Project
This project contains source code and supporting files for the serverless application that you created in the AWS Lambda console. You can update your application at any time by committing and pushing changes to your AWS CodeCommit or GitHub repository.
This project includes the following files and folders:
• src - Code for the application's Lambda function.
• events - Invocation events that you can use to invoke the function.
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約


api-lambda-ddb-cicd アプリケ
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/applications/api-lambda-ddb-cicd?tab=deploy
aws
サービス
リソースグループ
東京
サポート
AWS Lambda
ダッシュボード
アプリケーション
関数
Additional resources
レイヤー
Lambda > アプリケーション > api-lambda-ddb-cicd
api-lambda-ddb-cicd
概要
コード
デプロイ
モニタリング
パイプラインと連携
アプリケーションのパイプライン
名前
api-lambda-ddb-cicd-Pipeline
最新のアクション
Deploy: ExecuteChangeSet - 18 時間前
ステータス
Succeeded
パイプライン
CodePipeline で表示
SAM テンプレート
CloudFormation スタック
デプロイ履歴
スタックイベントを表示
1
デプロイメント
リソースタイプ
最終更新時間
状態
18 時間前
Lambda application
18 時間前
Create complete
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約


api-lambda-ddb-cicd アプリケ
CodePipeline - AWS Developer
https://ap-northeast-1.console.aws.amazon.com/codesuite/codepipeline/pipelines/api-lambda-ddb-cicd-Pipeline/view?region=ap-northeast-1
aws
サービス
リソースグループ
東京
サポート
デベロッパー用ツール
CodePipeline
ソース • CodeCommit
アーティファクト
• CodeArtifact
ビルド • CodeBuild
デプロイ • CodeDeploy
パイプライン • CodePipeline
開始方法
パイプライン
パイプライン
履歴
設定
設定
リソースへ移動する
フィードバック
デベロッパー用ツール > CodePipeline > パイプライン > api-lambda-ddb-cicd-Pipeline
api-lambda-ddb-cicd-Pipeline
通知
編集する
実行を停止
パイプラインをクローンする
変更をリリースする
Source 成功しました
パイプライン実行 ID: 4be09354-217a-4663-a778-96d35c21f72d
ApplicationSource
AWS CodeCommit
成功しました - 18 時間前
d7797bd0
d7797bd0 ApplicationSource: Initial commit by AWS CodeCommit
移行を無効にする
Build 成功しました
パイプライン実行 ID: 4be09354-217a-4663-a778-96d35c21f72d
PackageExport
AWS CodeBuild
成功しました - 17 時間前
詳細
d7797bd0 ApplicationSource: Initial commit by AWS CodeCommit
移行を無効にする
Deploy 成功しました
パイプライン実行 ID: 4be09354-217a-4663-a778-96d35c21f72d
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約


api-lambda-ddb-cicd アプリケ
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/applications/api-lambda-ddb-cicd?tab=monitoring
aws
サービス
リソースグループ
東京
サポート
AWS Lambda
ダッシュボード
アプリケーション
関数
Additional resources
レイヤー
Lambda > アプリケーション > api-lambda-ddb-cicd
api-lambda-ddb-cicd
モニタと連携
概要
コード
デプロイ
モニタリング
ダッシュボード: AWS Lambda 情報
ダッシュボードの選択
AWS Lambda
ダッシュボードに追加
1 時間 3 時間 12 時間 1 日 3 日 1 週 カスタム
Invocations
Errors
Duration (average)
ConcurrentExecutions
データがありません。
ダッシュボードの時間範囲を調整してみてください。
1
0.5
0
03:00
03:30
04:00
04:30
05:00
05:30
getAllItemsFunction
getByIdFunction
putItemFunction
Total Invocations
Total Errors
Average Duration
Total ConcurrentExecutions
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

api-lambda-ddb-cicd アプリケ

AWS CloudFormation Designer

https://ap-northeast-1.console.aws.amazon.com/cloudformation/designer/home?region=ap-northeast-1&stackId=arn:aws:cloudformation:ap-northeast-1:786032344772:stack/api-lambda-ddb-cicd/a4f

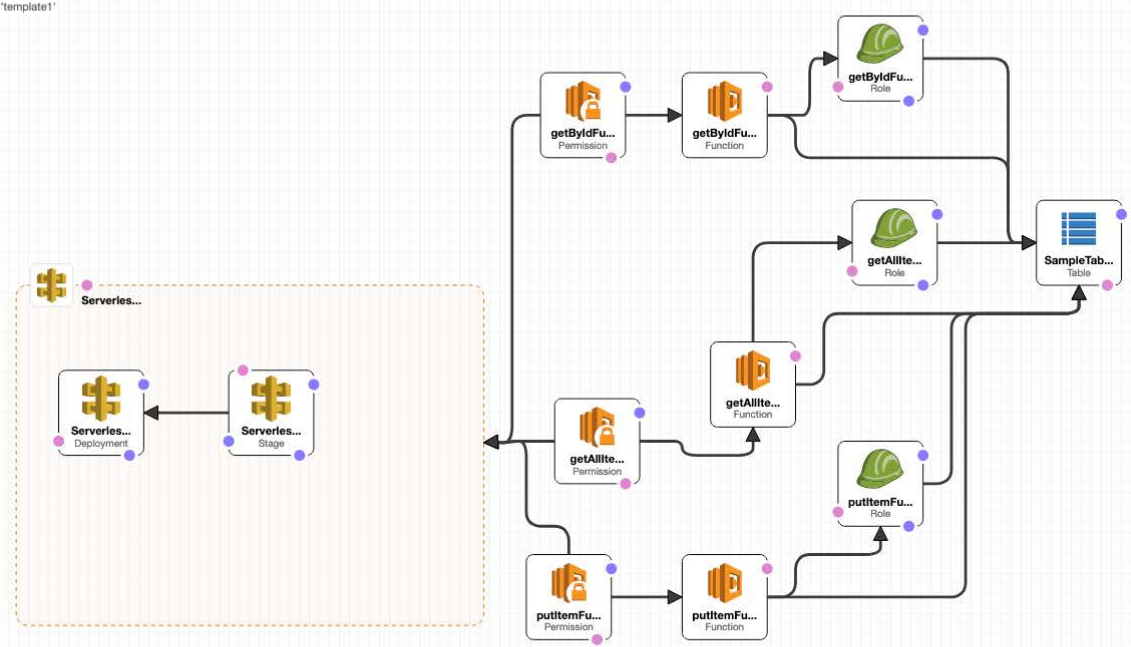
aws サービス リソースグループ 東京 サポート

閉じる

リソースタイプ

- ACMPCA
- AccessAnalyzer
- AmazonMQ
- Amplify
- ApiGateway
- ApiGatewayV2
- AppConfig
- AppMesh
- AppStream
- AppSync
- ApplicationAutoScaling
- ApplicationInsights
- Athena
- AutoScaling
- AutoScalingPlans
- Backup
- Batch
- Budgets
- Cassandra
- CertificateManager
- Chatbot
- CloudFormation
- CloudFront
- CloudTrail
- CloudWatch
- CodeBuild
- CodeDeploy
- CodeGuruProfiler
- CodePipeline

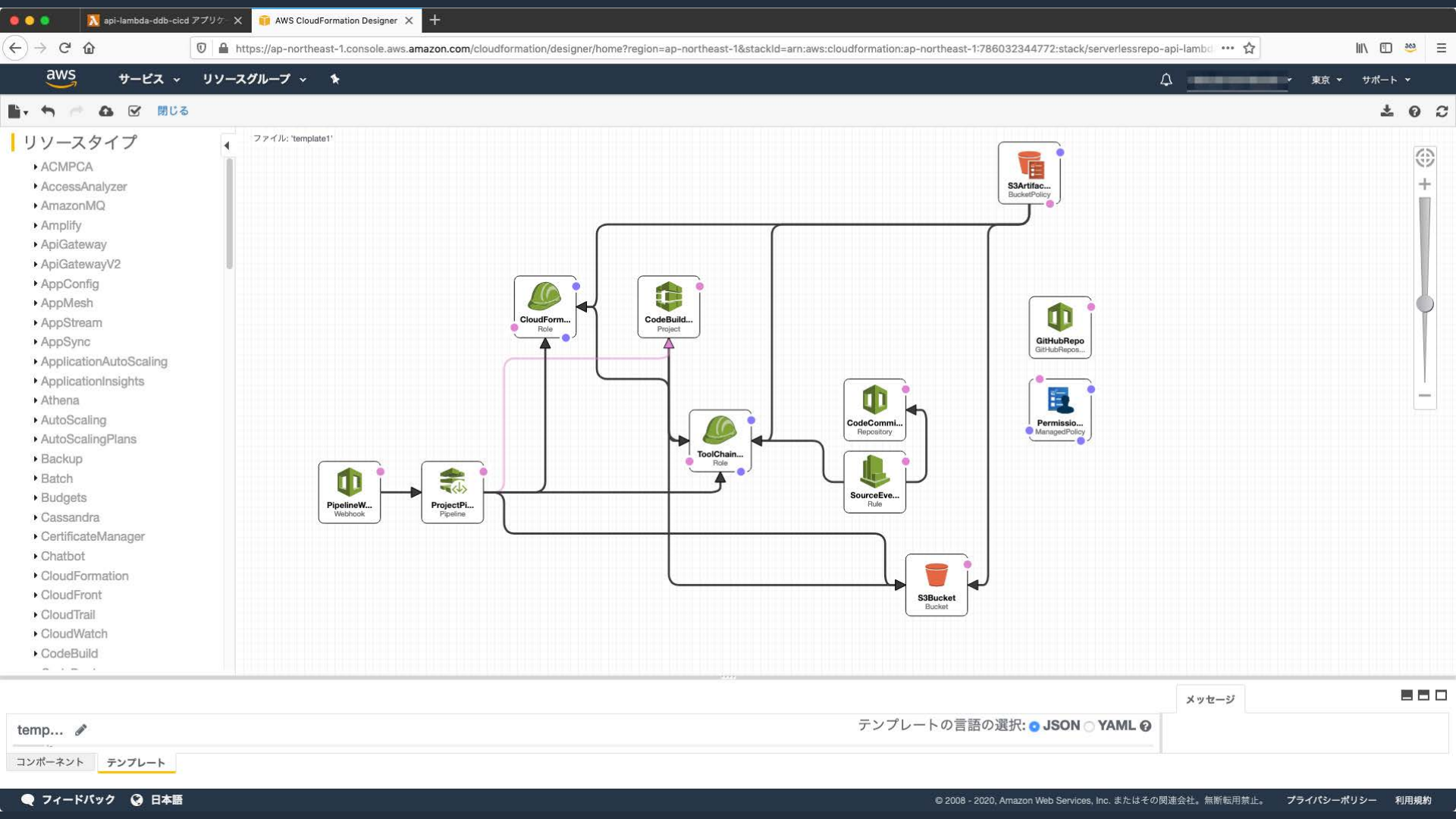
ファイル: 'template1'



フィードバック 日本語

 プライバシーポリシー 利用規約

codepipeline Highlight All Match Case Match Diacritics Whole Words 5 of 11 matches


api-lambda-ddb-cicd アプリケ
AWS CloudFormation Designer
https://ap-northeast-1.console.aws.amazon.com/cloudformation/designer/home?region=ap-northeast-1&stackId=arn:aws:cloudformation:ap-northeast-1:786032344772:stack/serverlessrepo-api-lambd
aws
サービス
リソースグループ
東京
サポート
閉じる
リソースタイプ
ACMPCA
AccessAnalyzer
AmazonMQ
Amplify
ApiGateway
ApiGatewayV2
AppConfig
AppMesh
AppStream
AppSync
ApplicationAutoScaling
ApplicationInsights
Athena
AutoScaling
AutoScalingPlans
Backup
Batch
Budgets
Cassandra
CertificateManager
Chatbot
CloudFormation
CloudFront
CloudTrail
CloudWatch
CodeBuild
ファイル: 'template1'
S3Artifac...
BucketPolicy
CloudForm...
Role
CodeBuild...
Project
GitHubRepo
GitHubRepos...
CodeCommi...
Repository
Permissio...
ManagedPolicy
ToolChain...
Role
SourceEve...
Rule
PipelineW...
Webhook
ProjectPi...
Pipeline
S3Bucket
Bucket
メッセージ
temp...
テンプレートの言語の選択: JSON YAML
コンポーネント
テンプレート
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

CloudFormation Stack の確認

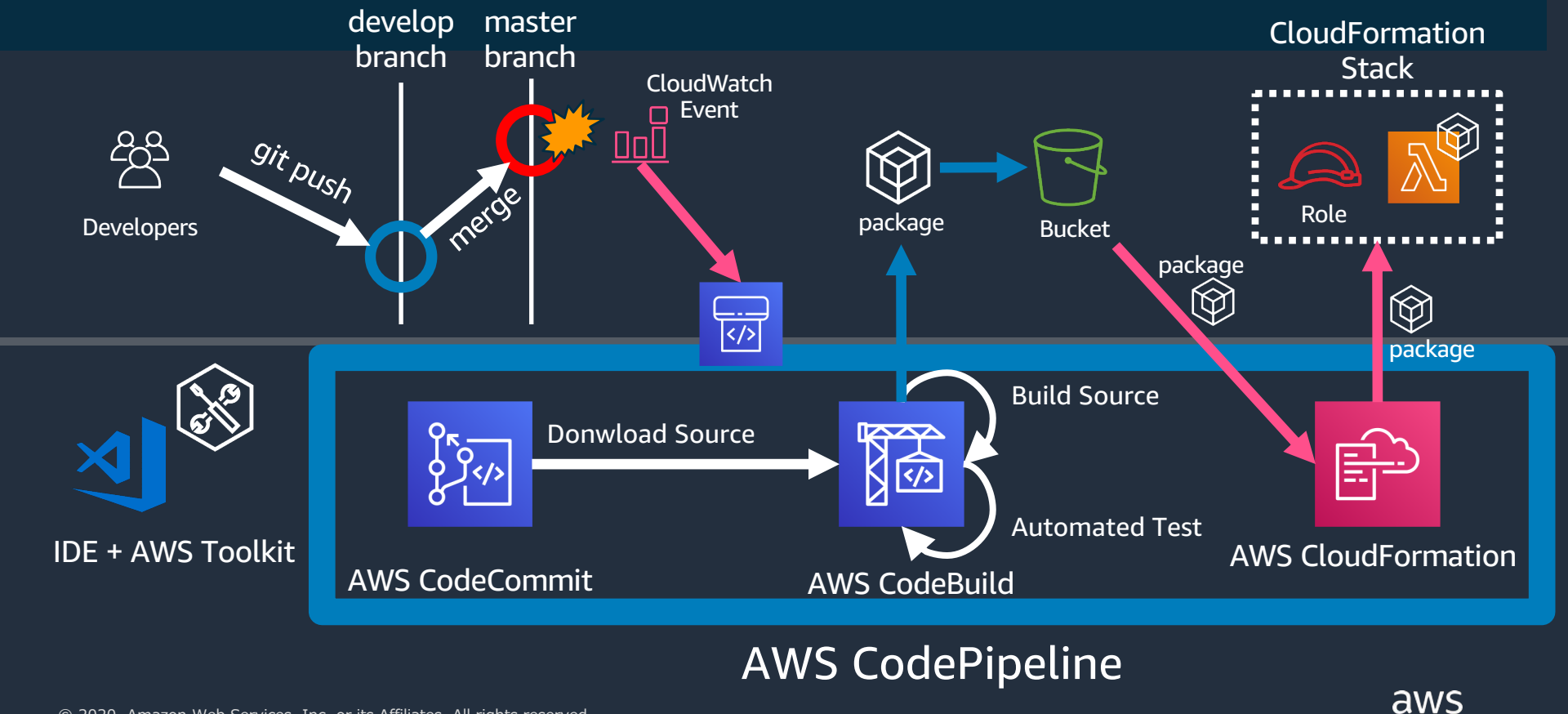
上位Stack

seminar-app
アプリケーションに必要なリソースを生成

下位Stack

serverlessrepo-seminar-app-toolchain
パイプラインに必要なツールリソースを生成

Lambda関数修正後のmergeでパイプライン起動！

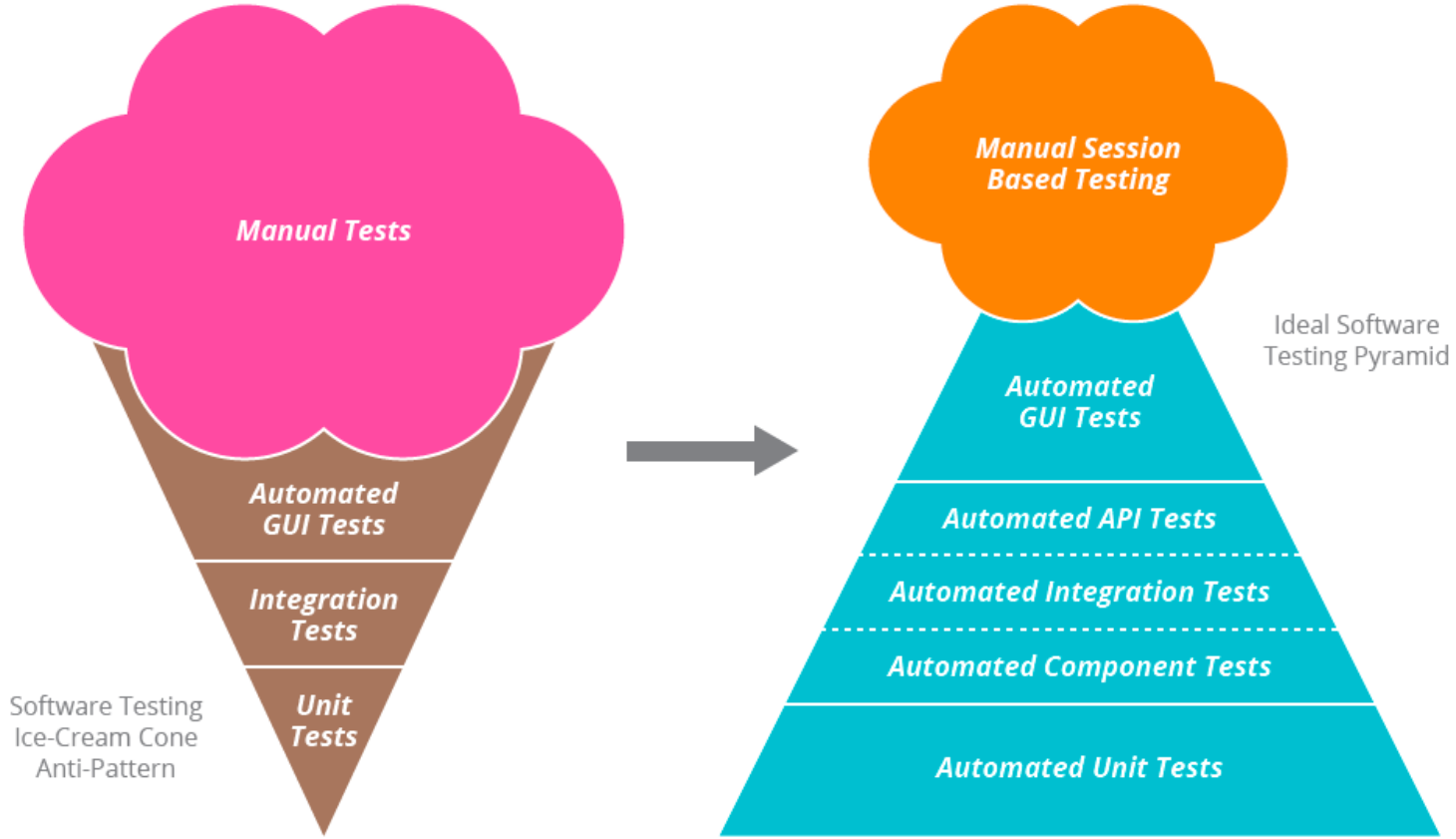


ここまでのまとめ

- AWS Lambdaの開発にはServerless Application Model (SAM) の利用が可能
- SAM CLIを利用して手作業によるミスを防ぎ自動化を促進
- Serverless Application Pipelineを利用してCI/CD環境を容易に構築可能

サーバーレスアプリケーションのテスト

Testing Pyramid



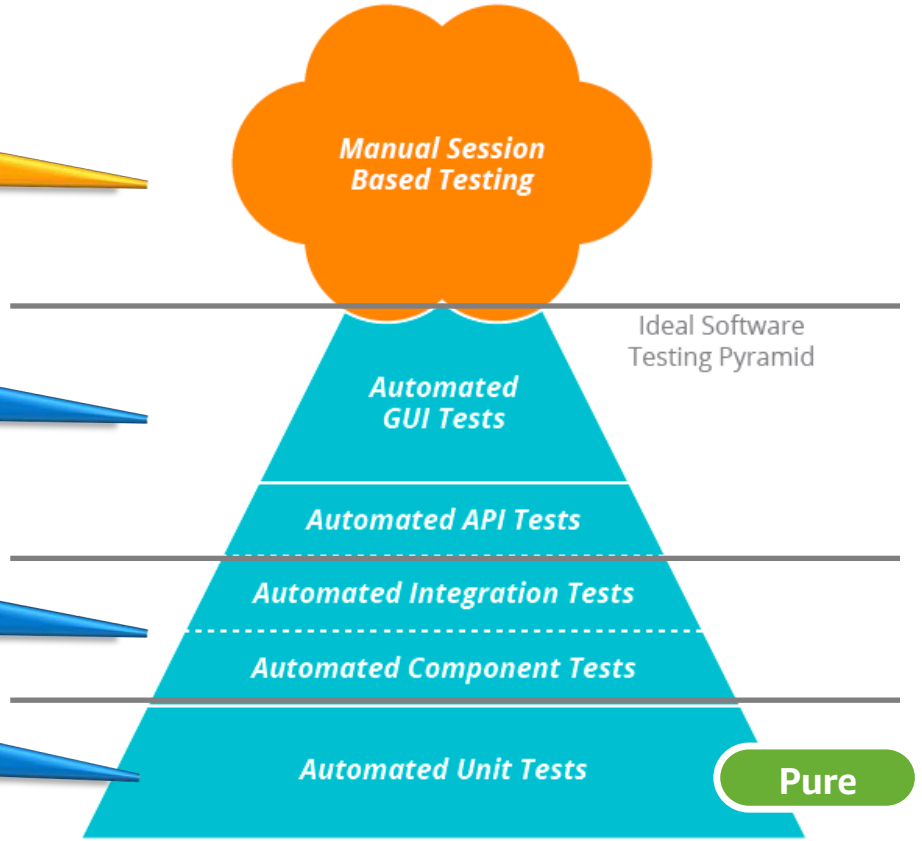
Testing Pyramid

極力減らす！！

E2E TEST、外形監視、
自動受入TEST

Lambdaが接する
世界とのTEST

Lambdaのことも知ら
ないTEST



サーバーレスのUnit Testは純粋なロジックに対して実施

具体例で考える

テイクアウト可能なドリンクショップ

軽減税率の適用
持ち帰り or イートイン？
8% or 10%

商品代金＋消費税計算

シンプルなアーキテクチャ図

Mobile client

Amazon API Gateway

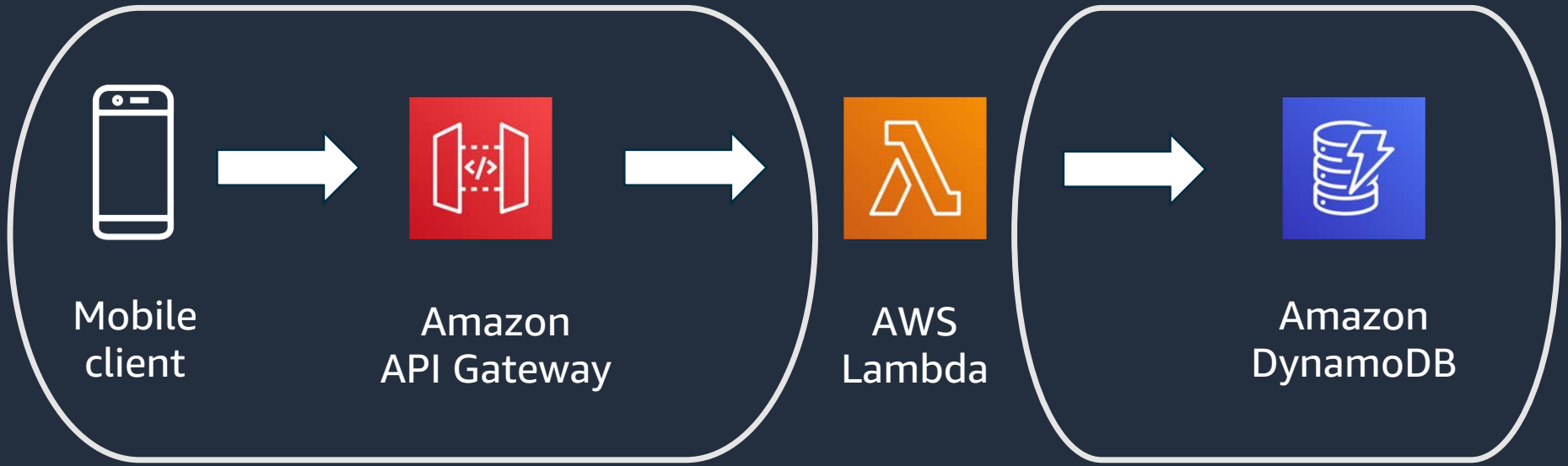
AWS Lambda

Amazon DynamoDB

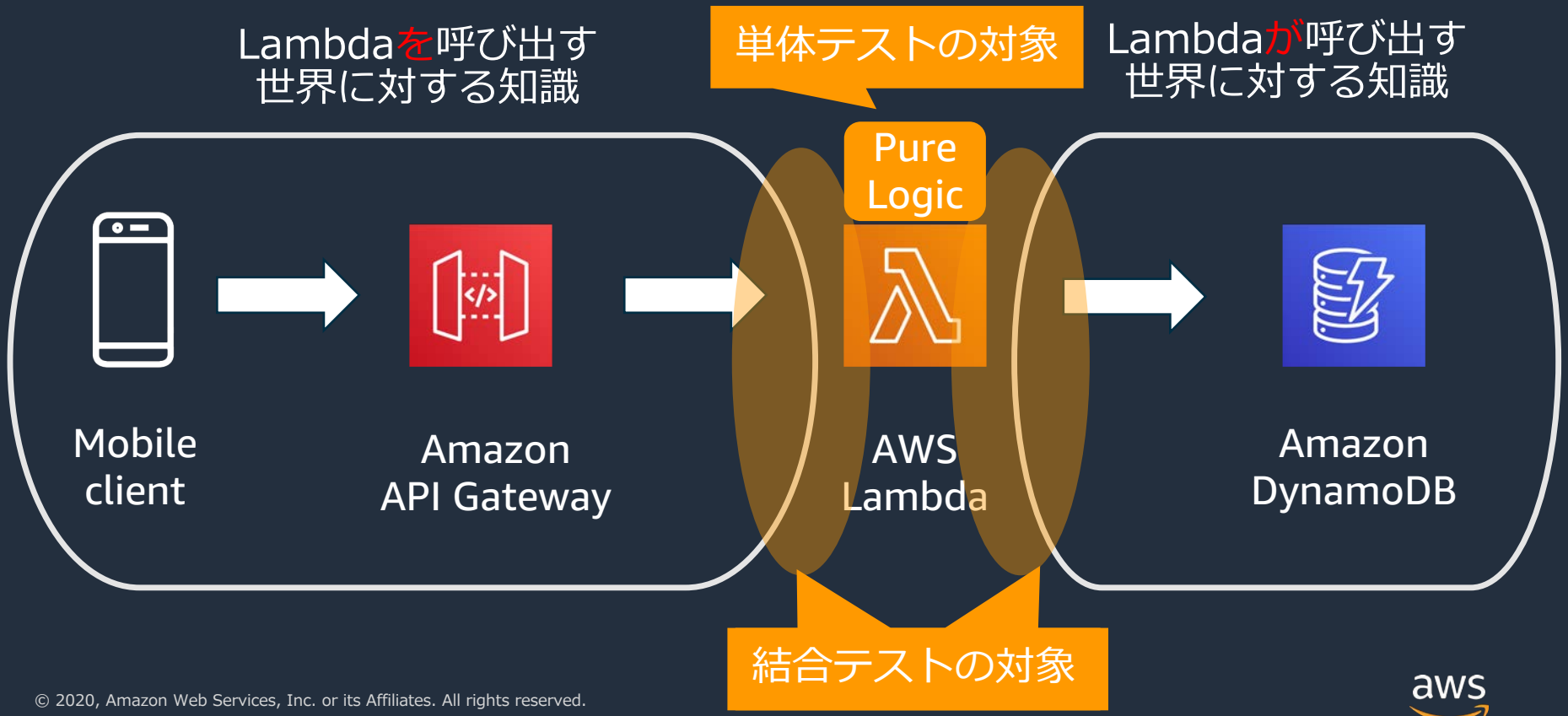
シンプルなアーキテクチャ図

Lambdaを呼び出す
世界に対する知識

Lambdaが呼び出す
世界に対する知識



シンプルなアーキテクチャ図



DynamoDB 商品価格マスタ

Amazon DynamoDB

id	item_name	price
10	Tea	120
20	Cola	150
30	Coffee	300
	…	

Lambda関数実装例

```
def lambda_handler(event, context):
    """ Unit TestしやすいLambda関数のサンプル
    """
    id = get_parameter(event, "id")
    is_takeout = get_parameter(event, "is_takeout")

    item = get_item(id)

    tax = calc_tax(item, is_takeout)

    return {
        "statusCode": 200,
        "body": json.dumps({
            'id': item['id'],
            'item_name': item['item_name'],
            'price': item['price'],
            'tax': tax
        }),
    }
```


aws

Lambda関数実装例

```
def lambda_handler(event, context):
    """ Unit TestしやすいLambda関数のサンプル
    """
    id = get_parameter(event, "id")
    is_takeout = get_parameter(event, "is_takeout")

    item = get_item(id)

    tax = calc_tax(item, is_takeout)

    return {
        "statusCode": 200,
        "body": json.dumps({
            'id': item['id'],
            'item_name': item['item_name'],
            'price': item['price'],
            'tax': tax
        }),
    }
```

Lambdaを呼び出すAPI Gatewayから必要なパラメータを取得する処理

DynamoDBからデータを取得する処理

消費税計算を行う処理（AWSに関する知識が不要な処理）

API Gatewayに結果を返す処理

Lambda関数実装例

```
def lambda_handler(event, context):
    """ Unit TestしやすいLambda関数のサンプル
    """
    id = get_parameter(event, "id")
    is_takeout = get_parameter(event, "is_takeout")


    item = get_item(id)


    tax = calc_tax(item, is_takeout)


    return {
        "statusCode": 200,
        "body": json.dumps({
            'id': item['id'],
            'item_name': item['item_name'],
            'price': item['price'],
            'tax': tax
        }),
    }
```

Pureなロジックとして Unit Testを実施可能

Unit Test対象の関数をTestableにする

- 関数をTestable(テスト可能)にするとは？
  - y = f(x) の形にする
  - xに特定の値を与えるとyの値が決まる

```
# add関数
def add(a, b):
    return a + b

def test_add():
    #unit test
    expected = 10  #期待する値
    assert expected == add(5, 5)
```


aws

Unit Testの例

テスト対象メソッド

```
def calc_tax(item, is_takeout):
    price = Decimal(item['price'])
    tax_rate = Decimal(0.1)
    if is_takeout is not None and is_takeout.lower() == "true":
        tax_rate = Decimal(0.08)
    return int(math.floor(price * tax_rate))
```

```
def test_calc_tax():
    item = {'id': '10', 'item_name': 'caffee', 'price': Decimal(100)}
    is_takeout = 'true'
    expected = 8

    assert expected == app.calc_tax(item, is_takeout)
```

期待する結果と比較する

aws

Pytest parametrizeの例

parametrizeの利用

```
@pytest.mark.parametrize(('item', 'is_takeout', 'expected'), [
    ({'id': '10', 'item_name': 'caffee', 'price': Decimal(100)}, 'false', 10),
    ({'id': '10', 'item_name': 'caffee', 'price': Decimal(100)}, 'true', 8),
    ({'id': '10', 'item_name': 'caffee', 'price': Decimal(90)}, 'false', 9),
    ({'id': '10', 'item_name': 'caffee', 'price': Decimal(90)}, 'true', 7),
    ({'id': '10', 'item_name': 'caffee', 'price': Decimal(110)}, 'false', 11),
    ({'id': '10', 'item_name': 'caffee', 'price': Decimal(110)}, 'true', 8),
    ({'id': '10', 'item_name': 'caffee', 'price': Decimal(200)}, 'false', 20),
    ({'id': '10', 'item_name': 'caffee', 'price': Decimal(200)}, 'true', 16)
])
def test_parameter_calc_tax(item, is_takeout, expected):
    assert expected == app.calc_tax(item, is_takeout)
```


aws

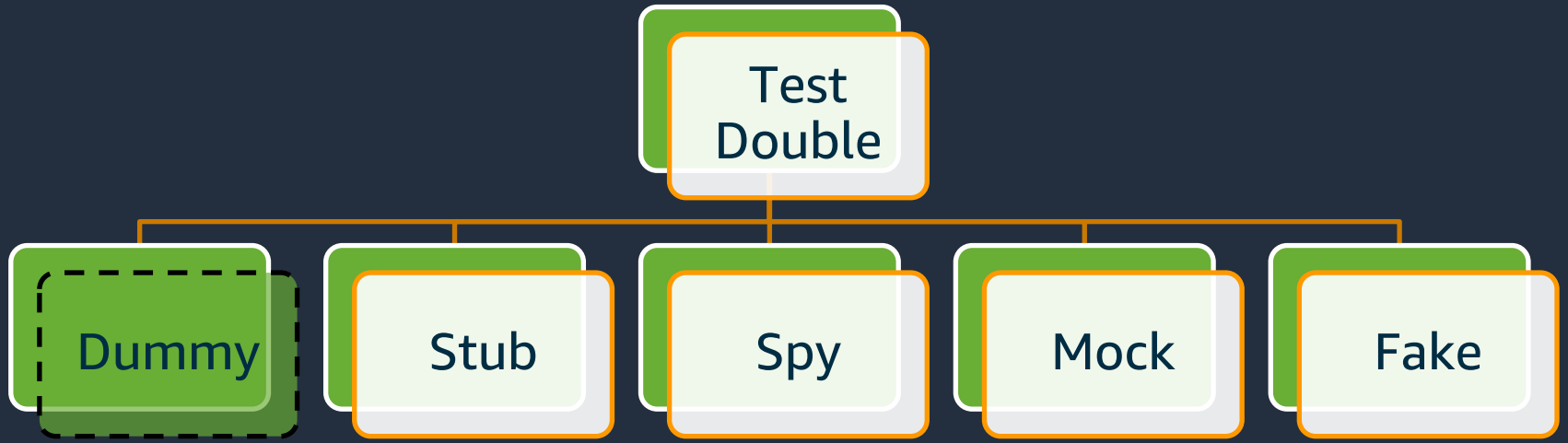
Pure Logicに対するUnite Test

- 純粋な要件に対してテストケースを書く
  - 他のAWSサービスとの結合方法に対する知識を含まない
- テストが軽くなるため、開発やデリバリーを高速化できる
- 外部のアーキテクチャが変更された時にも、テストケースが影響を受けにくい
  - テストの陳腐化が防げる
  - テストのメンテナンスコストの軽減
- Pure Logicなので意図が伝わりやすい、理解しやすい
  - テストケースから仕様を理解出来る

aws

AWSのサービスを利用するコードのUnit Test

Test Double
システムの一部を差し替えてテストする手法



Test Doubleは、テスト目的で本番オブジェクトを
置き換える場合の置き換えデザインパターンを表す用語
さまざまな種類がある

https://martinfowler.com/bliki/TestDouble.html

aws

システムの一部を差し替えてテストする手法

ダミーオブジェクト	パラメータとして受け渡しするが利用されることのないオブジェクト
スタブ	テスト時に呼び出され、あらかじめ容易した結果を返すオブジェクト
スパイ	テスト対象からの出力を記録しておくことでテストコード実行後に値を取り出して検証する
モック	期待した通りに呼び出されることを確認するためのオブジェクト。期待されない呼び出しが行われた場合は例外を返す。
フェイク	動作するように実装されているがエラー処理などを手抜きしたテスト用オブジェクト

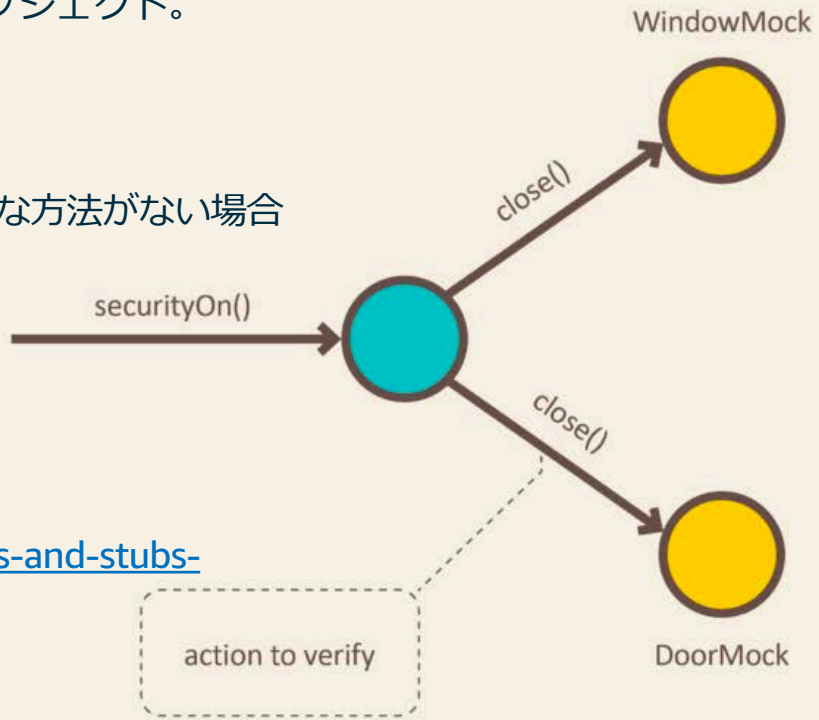
Mockを利用したテスト

Mock の概念

期待した通りに呼び出されることを確認するためのオブジェクト。
期待されない呼び出しが行われた場合は例外を返す。

- Mock は以下の場合に利用
  - 実動コードを呼び出したくない場合
  - 意図したコードが実行されたことを確認する簡単な方法がない場合

例）securityOnメソッドが呼ばれたらWindowオブジェクトとDoorオブジェクトの両方のcloseメソッドが呼ばれることをテストで確認したい。



https://blog.pragmatists.com/test-doubles-fakes-mocks-and-stubs-1a7491dfa3da

Moto - Mock AWS Services

https://github.com/spulec/moto

In a nutshell

Moto is a library that allows your tests to easily mock out AWS Services.

Imagine you have the following python code that you want to test:

```
import boto3

class MyModel(object):
    def __init__(self, name, value):
        self.name = name
        self.value = value

    def save(self):
        s3 = boto3.client('s3', region_name='us-east-1')
        s3.put_object(Bucket='mybucket', Key=self.name, Body=self.value)
```

motoを利用したテストコード例

```
@mock_dynamodb2
def test_mock_ddb():
    ddb_client = boto3.client('dynamodb)
    ddb_client.create_table(
       AttributeDefinitions=[{'AttributeName': 'id','AttributeType': 'S'}],
       TableName='Items',
       KeySchema=[{'AttributeName': 'id','KeyType': 'HASH'}],
       BillingMode='PAY_PER_REQUEST')

    (続く…)
```

motoが提供するデコレータ

boto3によるAPIの呼び出しを自動的にモック化

motoを利用したテストコード例(続き)

```
ddb_client.batch_write_item(
  RequestItems={'Items': [
  {'PutRequest': {'Item':
     {'id': {'S': '10'},'item_name': {'S': 'Tea'},'price': {'N': '120'}}}},
   {'PutRequest': {'Item':
     {'id': {'S': '20'},'item_name': {'S': 'Cola'},'price': {'N': '150'}}}},
   {'PutRequest': {'Item':
     {'id': {'S': '30'},'item_name': {'S': 'Coffee'},'price': {'N': '300'}}}}
    ]
 })
```

（続く…）

テスト用にモックの中にテーブルとアイテムを作成

aws

motoを利用したテストコード例(続き)

```
item = app.get_item('10')
assert '10' == item['id']
assert 'Tea' == item['item_name']
assert Decimal(120) == item['price']

item = app.get_item('20')
assert '20' == item['id']
assert 'Cola' == item['item_name']
assert Decimal(150) == item['price']

item = app.get_item('30')
assert '30' == item['id']
assert 'Coffee' == item['item_name']
assert Decimal(300) == item['price']
```

関数がDynamoDBから正しく値を取得できていることを確認

aws

Mock による Unit Test

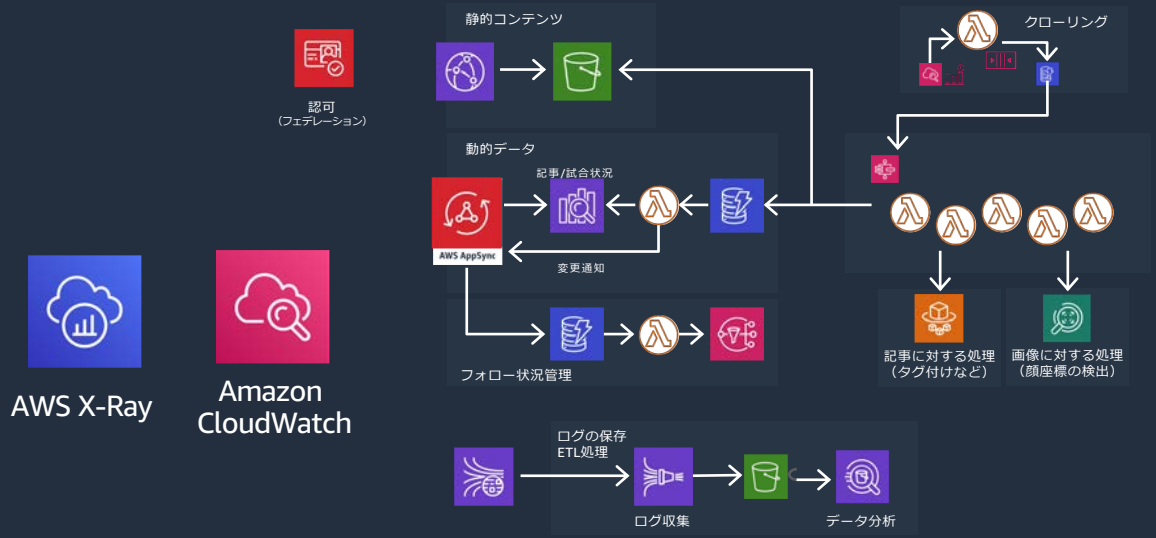
- 他のサービスとの連携を含む単体テストケースが書ける
- AWSのサービスを構築することなくローカル環境で実行可能
- テストは軽く開発やデリバリーを高速化できる
- 外部のアーキテクチャが変更された時には、テストケースが影響を受ける

aws

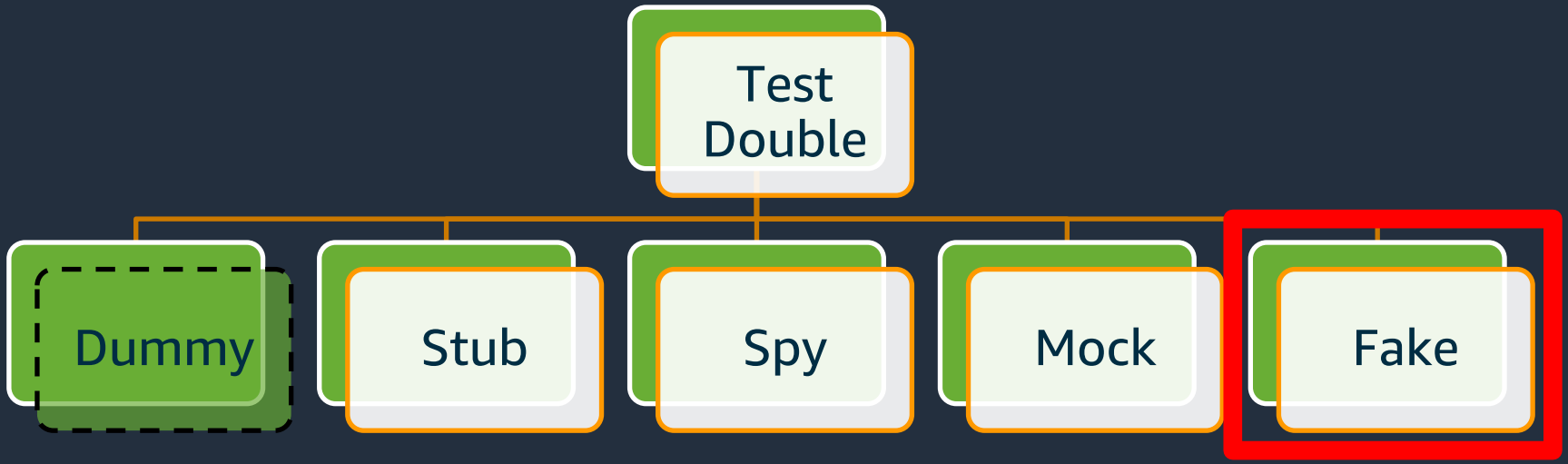
Fakeを利用したテスト

Serverless は分散アーキテクチャ

複数サービスにまたがるロジックをどのようにテストするか



Fakeの利用

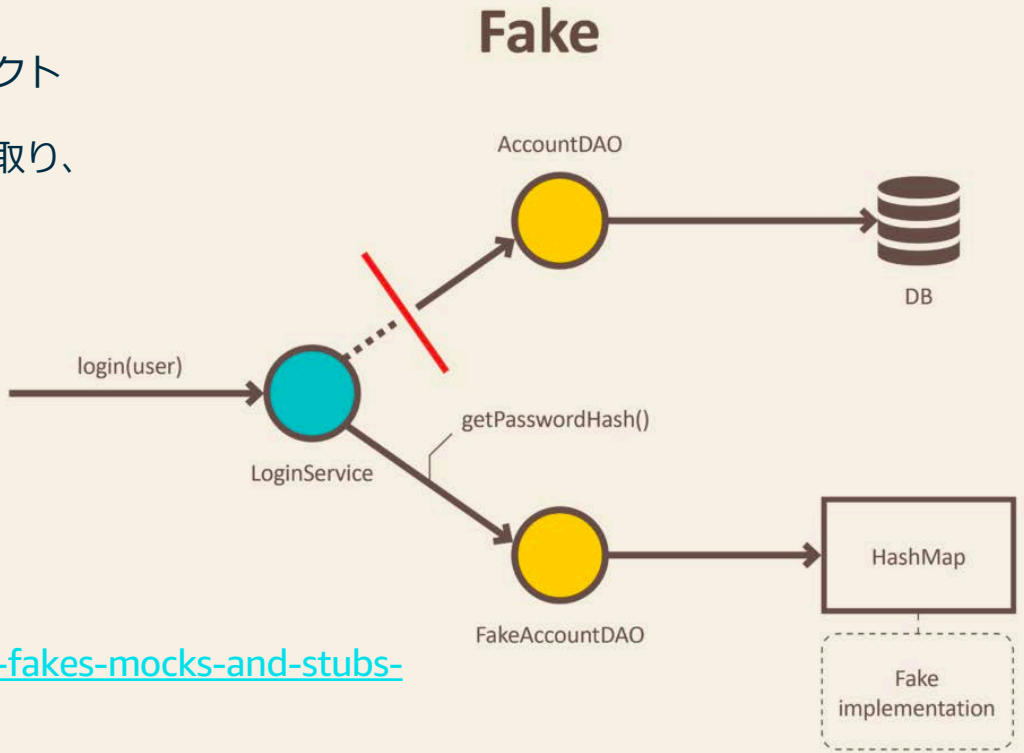


https://martinfowler.com/bliki/TestDouble.html

Fake の概念

- Fakeとは、
  - 実際に機能する実装を備えたオブジェクト
  - 実稼働のものとは実装が異なる
  - いくつかの実装上のショートカットを取り、プロダクション実装より簡素化される

例）実際のDBへのアクセスを避けて
ダミーの値を返すなど



https://blog.pragmatists.com/test-doubles-fakes-mocks-and-stubs-1a7491dfa3da

AWS が提供するFake

Step Functions Local

ローカル開発環境で
実行されている
StepFunctionsを使用して
アプリケーションを開発お
よびローカル
テストできます

ECS Local Container
Endpoints

Fargate＆ECSに展開
する前にアプリケーシ
ョンをローカルでテス
トするのに役立ちます

DynamoDB Local

ローカルバージョンを使用
することでスループットや
データストレージ、データ
転送料金を節約すること
できます

CodeBuild Local Builds

CodeBuild環境をローカルで
シミュレートして、BuildSpecファ
イルにあるコマンドと設定のトラブ
ルシューティングをすぐに行えるよ
うになります

https://hub.docker.com/u/amazon

aws

AWS が提供するFake

Step Functions Local

ローカル開発環境で実行されているStepFunctionsを使用してアプリケーションを開発およびローカルテストできます

ECS Local Container Endpoints

Fargate＆ECSに展開する前にアプリケーションをローカルでテストするのに役立ちます

DynamoDB Local

ローカルバージョンを使用することでスループットやデータストレージ、データ転送料金を節約することができます

CodeBuild Local Builds

CodeBuild環境をローカルでシミュレートして、BuildSpecファイルにあるコマンドと設定のトラブルシューティングをすぐに行えるようになります

https://hub.docker.com/u/amazon

aws

DynamoDB Local Dockerの利用

docker-compose.yml

```yaml
version: '3.7'
services:
  dynamodb-local:
    image: amazon/dynamodb-local
    container_name: dynamodb-local
    ports:
      - "8000:8000"
    networks:
      - ddb-local
networks:
  ddb-local:
    external: true
```

DynamoDB Localコンテナイメージ

ローカル実行のLambdaとの通信用

DynamoDB Local Dockerの利用

ddb-local-up.sh

```
#!/bin/sh
docker network create ddb-local
sleep 1
docker-compose up -d
```

Docker Networkの作成

DynamoDB Localの起動

aws

DynamoDB Local Dockerの利用

create_ddb_table_data.sh

```
#!/bin/sh
export DDB_ENDPOINT_URL=http://localhost:8000
cd awscli/
./create_ddb_table.sh
sleep 2
./add_items.sh
```

DynamoDBのEndpoint URLの指定

DynamoDB Localにテスト用のテーブルとデータを作成

DynamoDB Local Dockerの利用

create_ddb_table.sh

```
#!/bin/sh
 aws dynamodb create-table --endpoint-url $DDB_ENDPOINT_URL \
  --table-name Items \
  --attribute-definitions \
    AttributeName=id,AttributeType=S \
  --key-schema \
    AttributeName=id,KeyType=HASH \
  --billing-mode=PAY_PER_REQUEST
```


DynamoDB Local Dockerの利用

add_items.sh

```
#!/bin/sh
aws dynamodb batch-write-item --endpoint-url $DDB_ENDPOINT_URL \
  --request-items file://items.json \
  --return-item-collection-metrics SIZE
```


aws

DynamoDB Local Dockerの利用

items.json

```
{
  "Items" : [
    {
        "PutRequest": {
           "Item": {
               "id": { "S": "10"},
               "item_name": { "S":"Tea"},
               "price": {"N": "120" }
           }
       }
   }, …
```

DynamoDB Local Dockerの利用

test_handler.py

```
@pytest.fixture()
def apigw_event():
    subprocess.run(["./ddb-local-up.sh"])
    time.sleep(2)
    subprocess.run(["./create_ddb_table_data.sh"])
    os.environ["DDB_ENDPOINT_URL"] = "http://localhost:8000"
    event = {
        "resource": "/price/{id}/{is_takeout}",
        "requestContext": {
    … 省略
    }
    yield event
    time.sleep(2)
    subprocess.run(["./ddb-local-down.sh"])
```

fixture でテストの前後に実施する処理を記述

API Gatewayから渡されるeventをシミュレート

テスト実施後の処理

DynamoDB Local Dockerの利用

test_handler.py 続き

```python
def test_lambda_handler(apigw_event, mocker):
    ret = app.lambda_handler(apigw_event, "")
    data = json.loads(ret["body"])

    assert ret["statusCode"] == 200
    assert "id" in ret["body"]
    assert "item_name" in ret["body"]
    assert "price" in ret["body"]
    assert "tax" in ret["body"]
    assert data["id"] == "10"
    assert data["item_name"] == "cola"
    assert data["price"] == 120
    assert data["tax"] == 12
```

fixture でテストの前後に実施する処理を記述

AWS SAM CLI での Fake

1. ローカル起動用の Dummy Event データ生成

```
$ sam local generate-event ¥
    apigateway aws-proxy ¥
    --path datadog_report ¥
    --method GET > event.json
```

2. ローカルでLambdaを eventデータを指定して実行

```
$ sam local invoke -e
event.json
```

SAM CLI

AWS SAM CLI での Fake

1. ローカルでAPI Gatewayから、Lambda呼び出し実行

```
$ sam local start-api
```

2. http://127.0.0.1:3000 で起動
3. ローカルブラウザ、 curl などでアクセスし実行

```
$ curl
http://127.0.0.1:3000/hello
```

SAM CLI

aws

sam local invokeの利用

test-lambda-invoke-local.sh

```
#!/bin/sh
./ddb-local-up.sh
sleep 2
./create_ddb_table_data.sh
sam local invoke --docker-network ddb-local \
-e events/test.json -n env.json
sleep 1
./ddb-local-down.sh
```

DynamoDB Localの起動

テーブルの作成とデータの登録

sam local invokeコマンドでLambda関数をローカルに実行

aws

sam localへの環境変数の提供

env.json

```
{
  "CalcPriceFunction": {
     "DDB_ENDPOINT_URL": "http://dynamodb-local:8000"
    }
}
```

コンテナ間の通信で必要

aws

sam local start-apiの利用

test-api-start-local.sh

```
#!/bin/sh
./ddb-local-up.sh
sleep 2
./create_ddb_table_data.sh
sleep 3
sam local start-api --docker-network ddb-local -n env.json
sleep 1
./ddb-local-down.sh
```

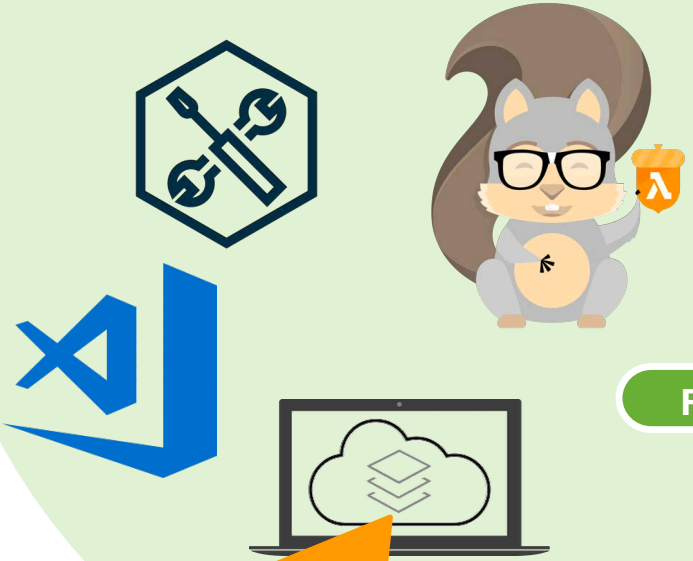
DynamoDB Localの起動

テーブルの作成とデータの登録

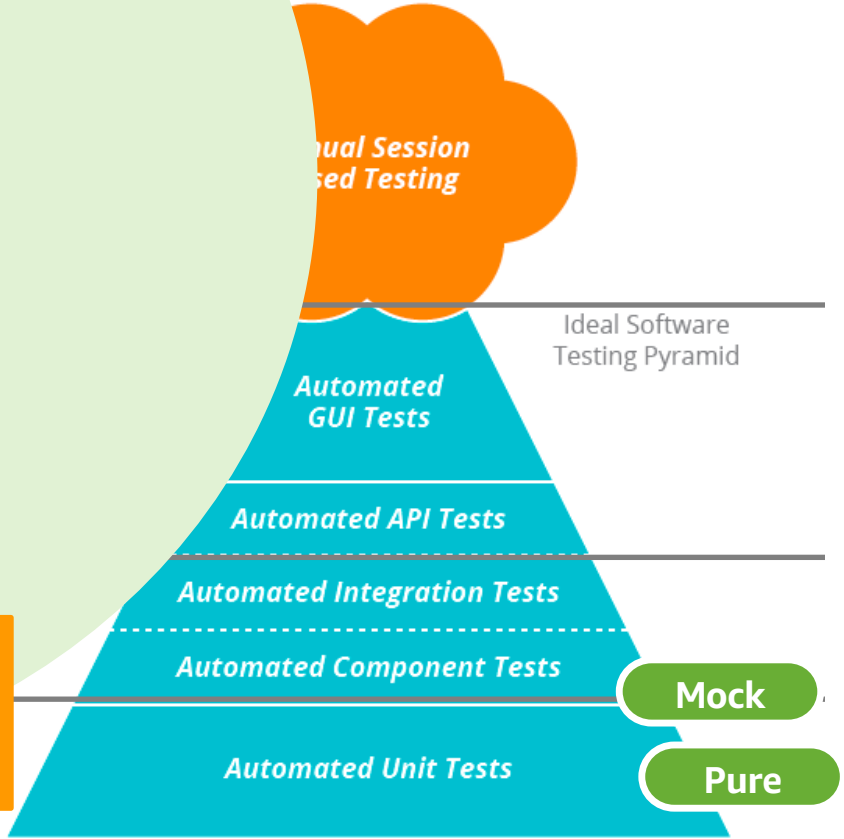
ここで別ターミナルから以下を実行
http://localhost:3000/price/10/true

aws

Fake は開発時に利用すると便利 (補足)



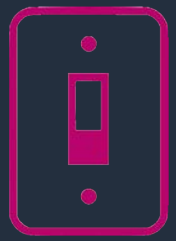
Fake



コーディング中、Fakeを常駐させておき、対話させるなど便利

サーバーレスアプリケーションの統合テスト

サーバーレスは使った分だけのお支払い

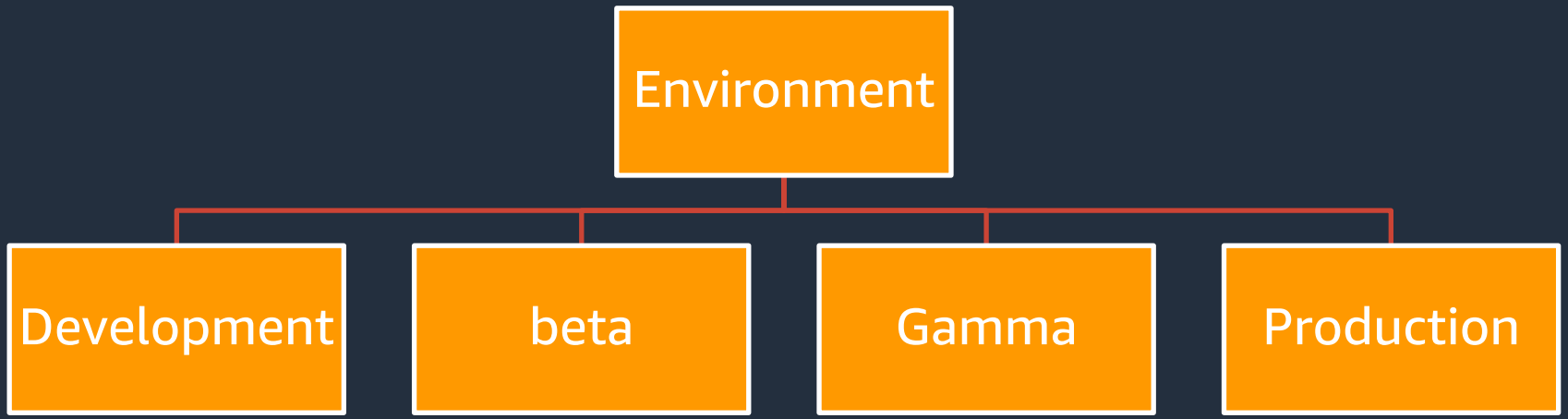


価値に対する支払い

リクエスト数、ならびに処理の実行時間に応じた課金となり、実際の処理量と金額が比例する

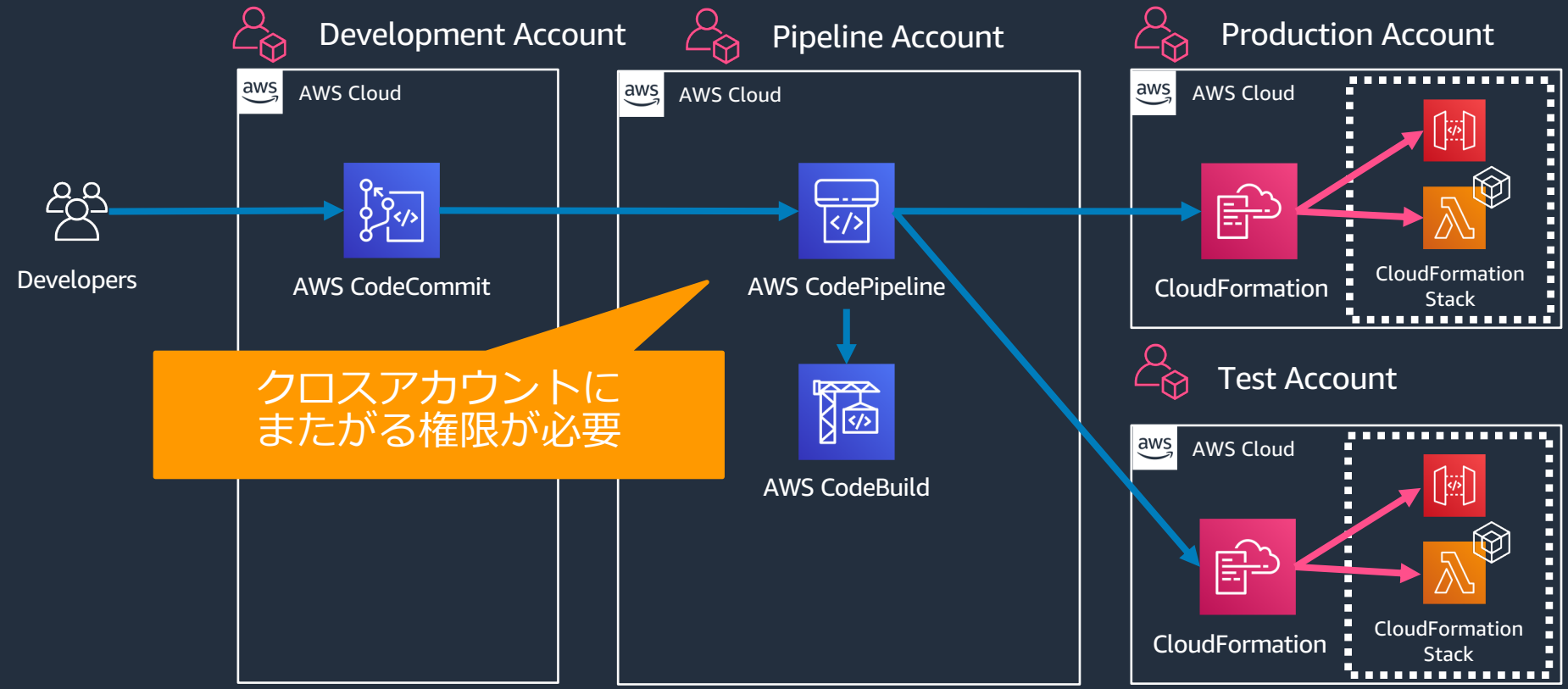
事前の初期投資が不要なので、コストゼロからのスタートが可能

デプロイ環境戦略

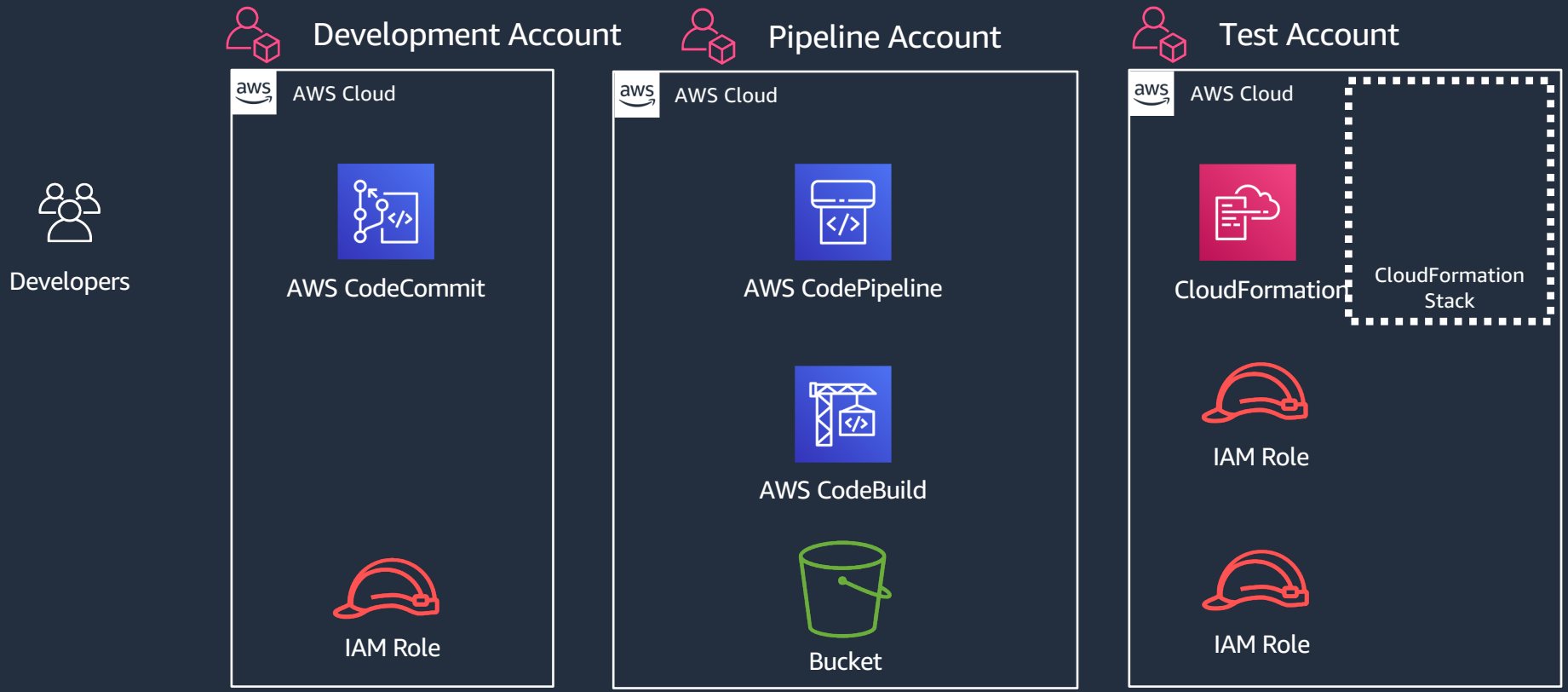


デリバリーライフサイクルに応じて AWSアカウントを個別に用意する

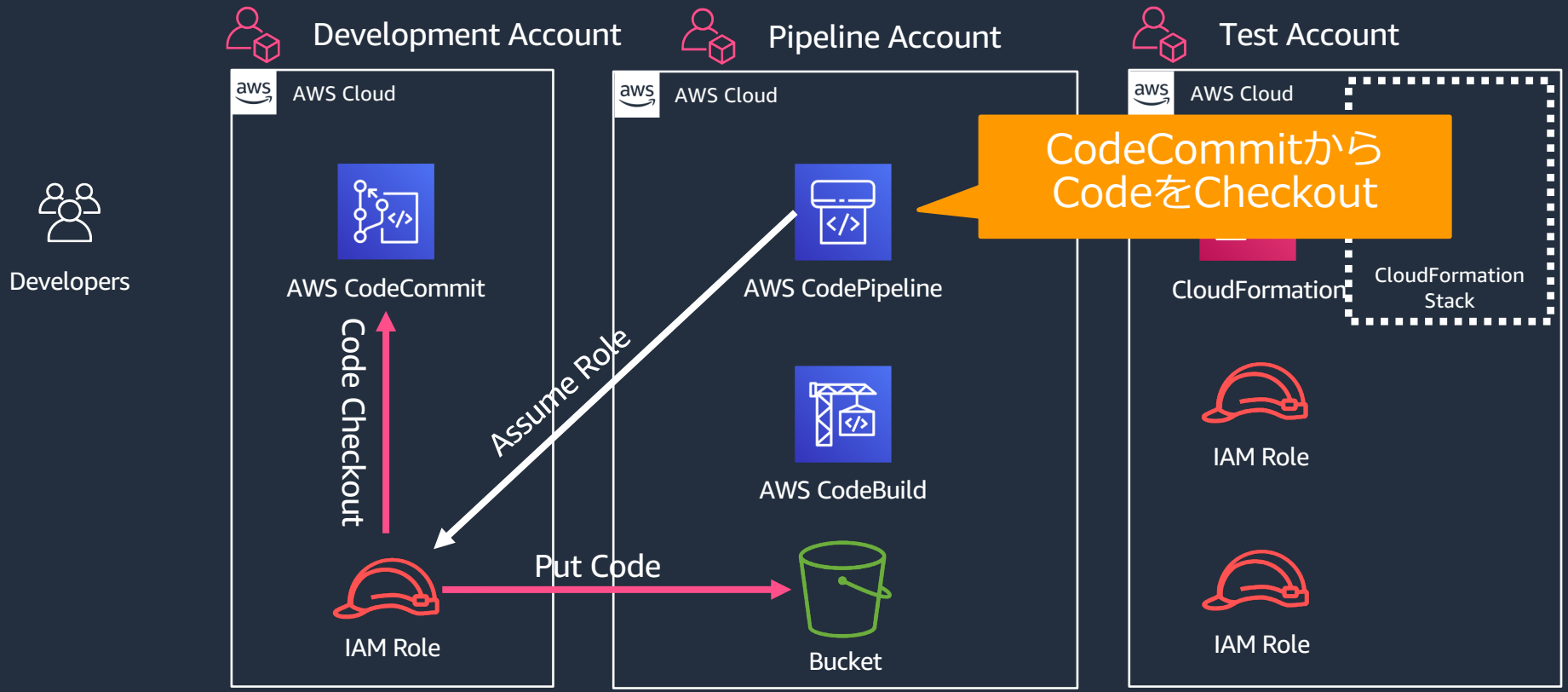
クロスアカウントデリバリーパイプライン



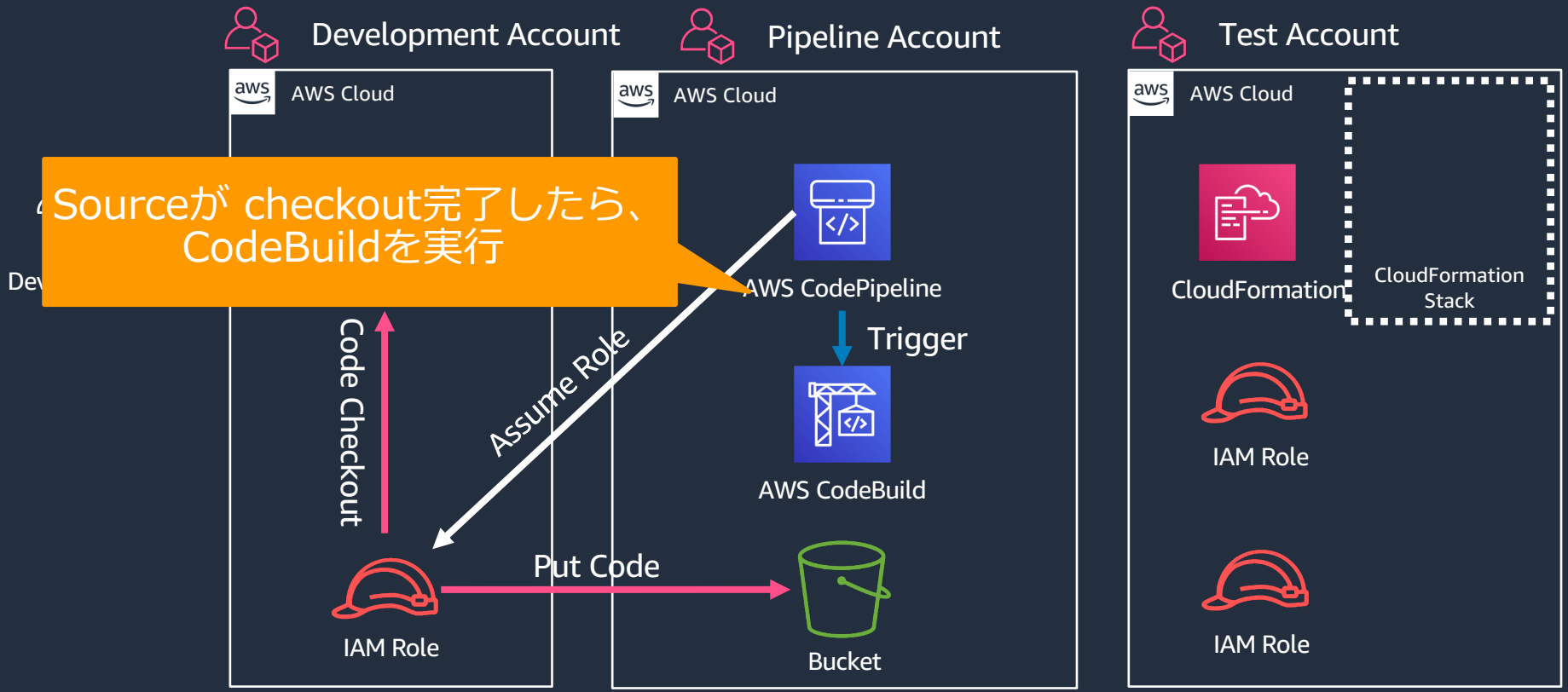
クロスアカウントデリバリーパイプライン



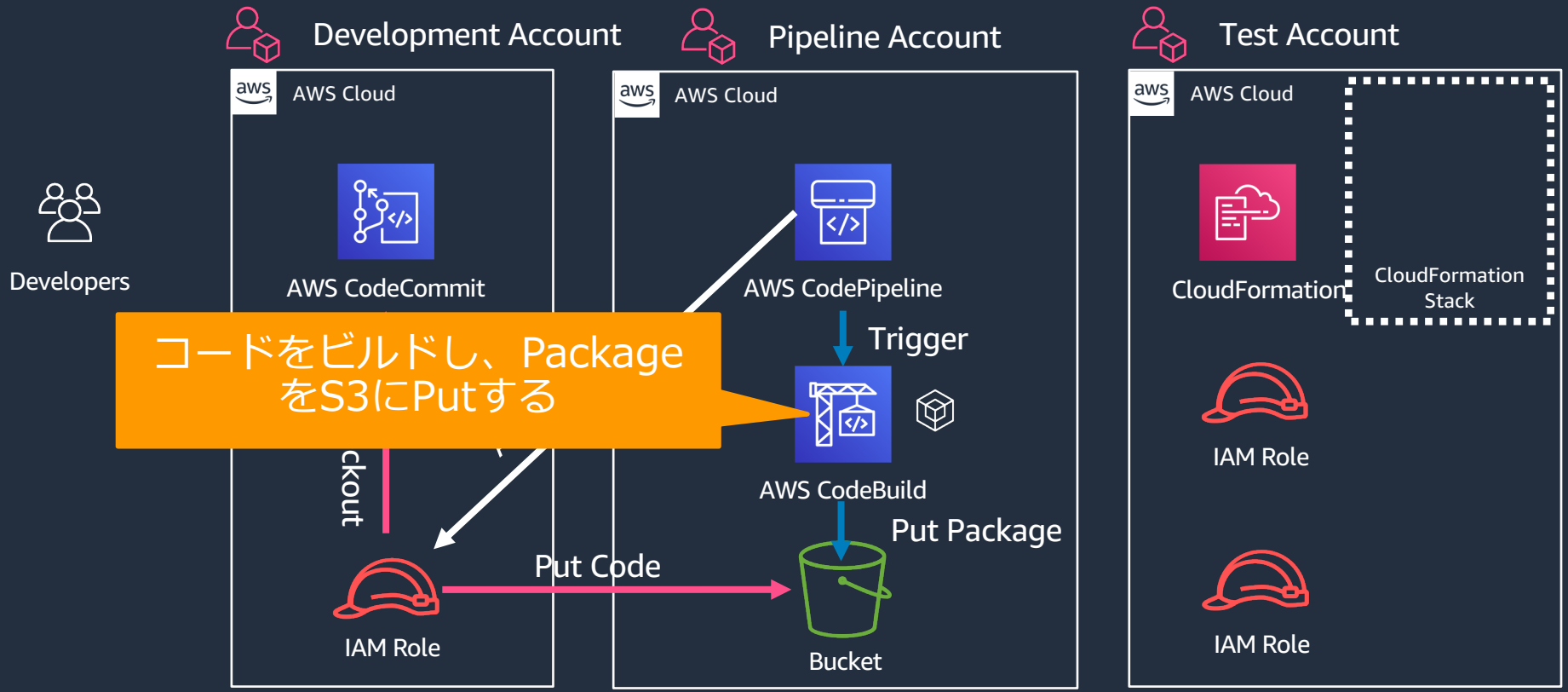
Cross Account Delivery Pipeline



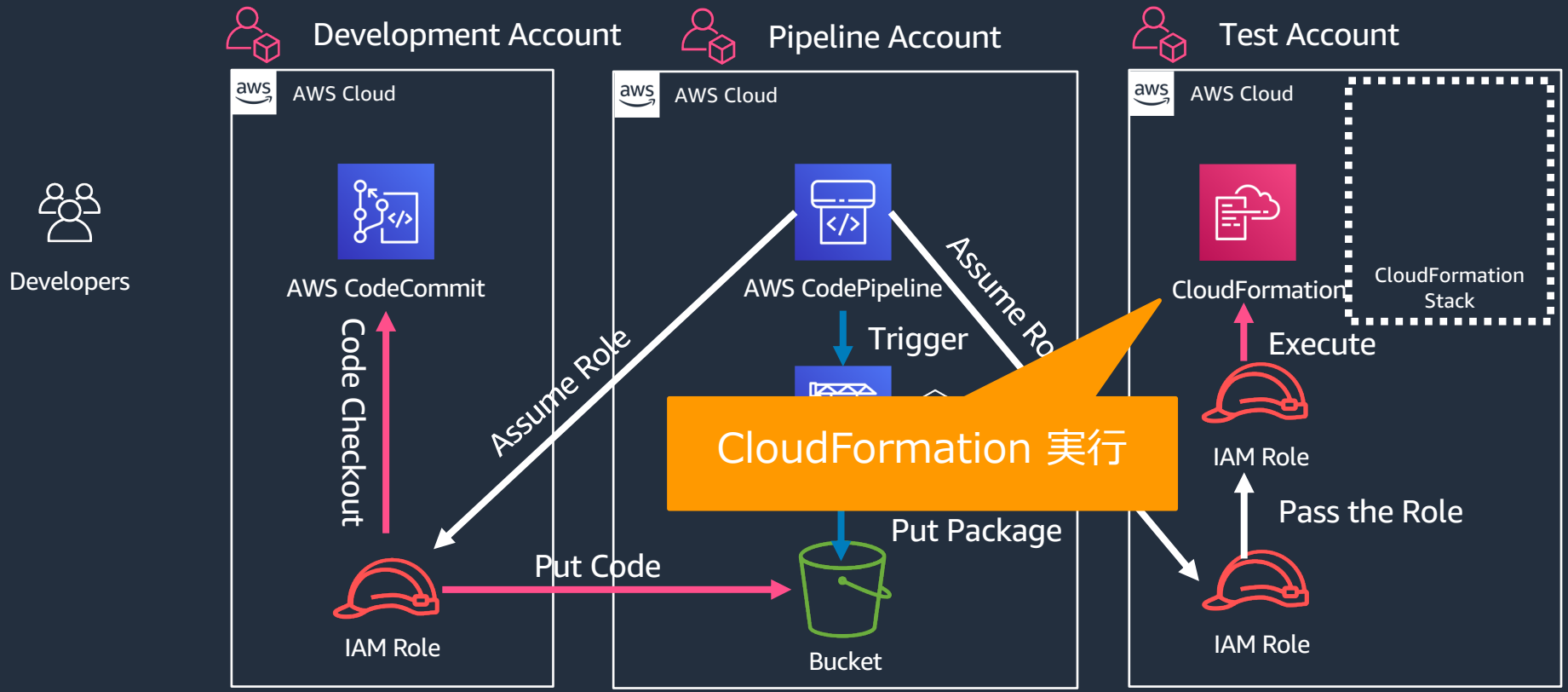
Cross Account Delivery Pipeline



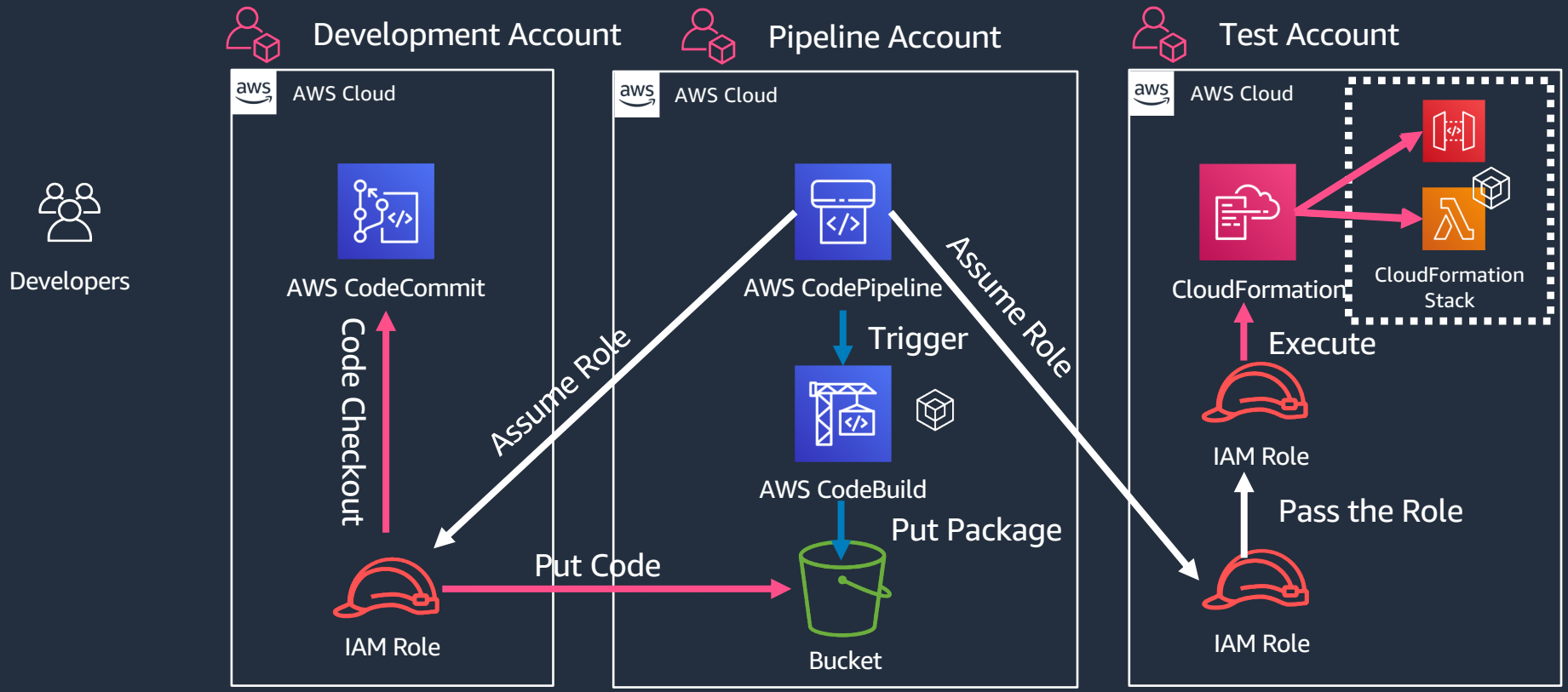
Cross Account Delivery Pipeline



Cross Account Delivery Pipeline



Cross Account Delivery Pipeline



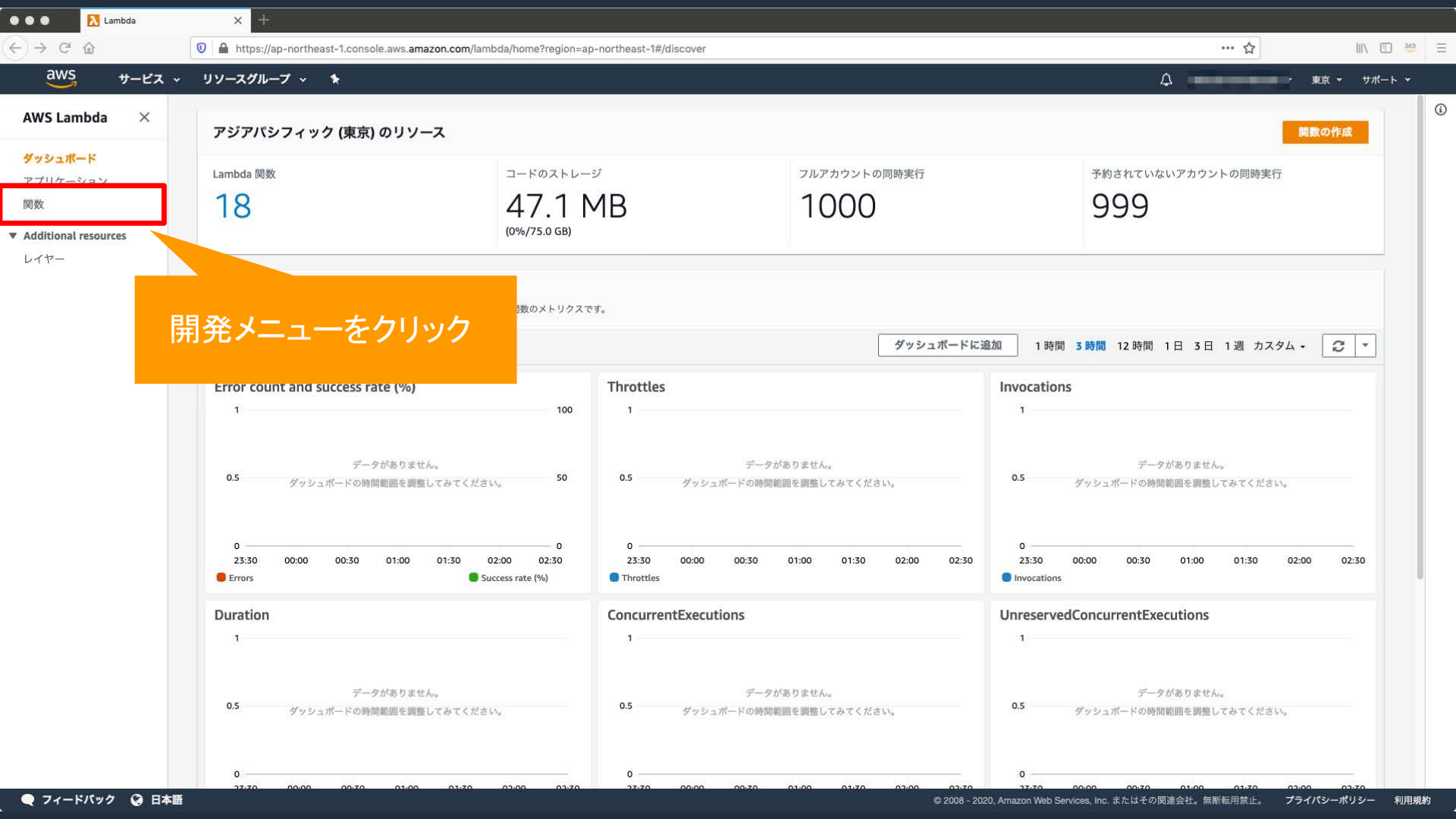
https://aws.amazon.com/jp/blogs/devops/aws-building-a-secure-cross-account-continuous-delivery-pipeline/

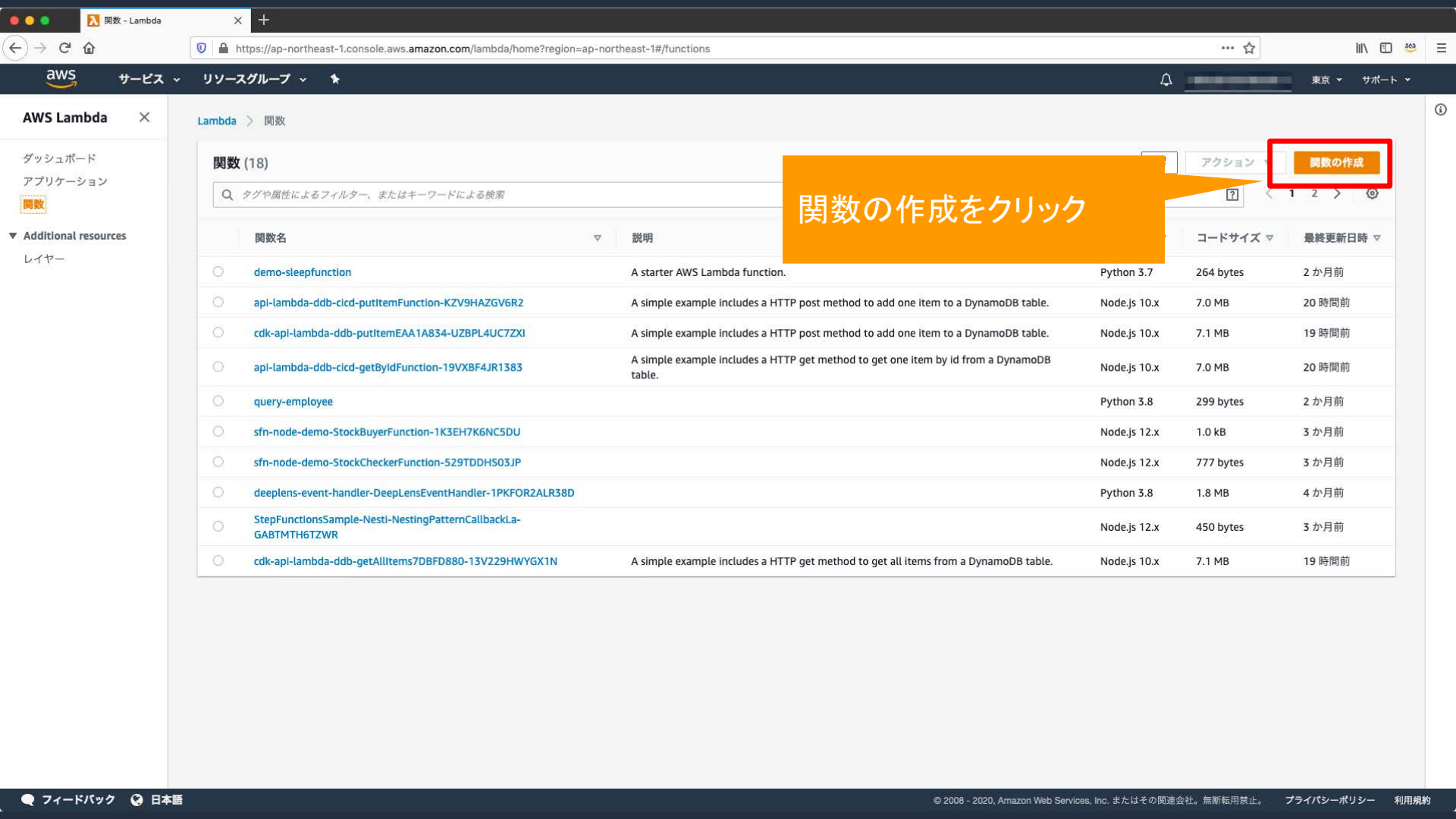
まとめ

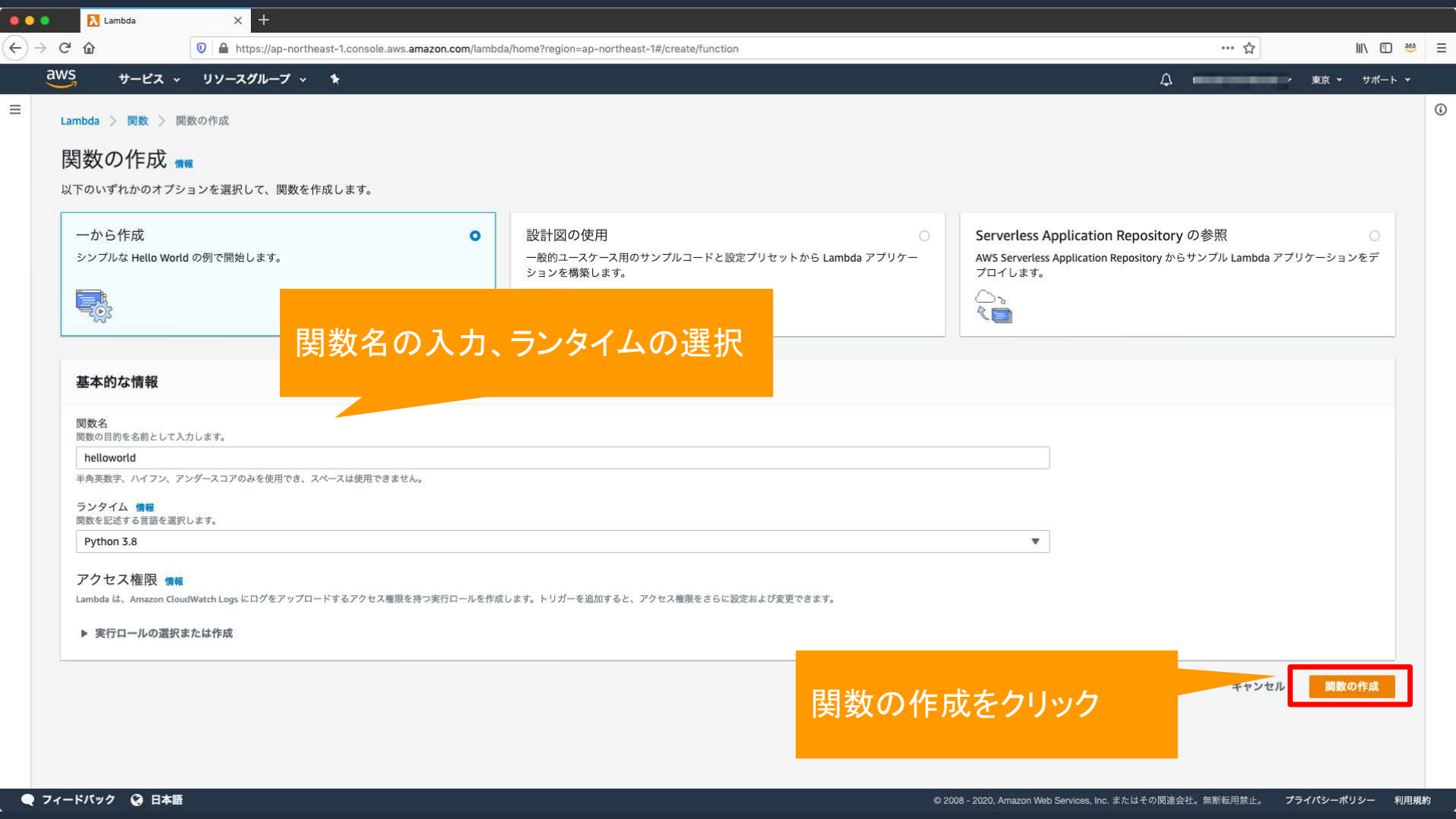
- Unit TestはPureなロジックを関数に切り出しテストしやすい形に
- MockやFakeを活用して開発効率をアップ
- 結合テスト以降はマルチアカウント戦略を活用、安全に必要なときに必要なリソースを作成しテストを実行
- 可能な限りテストを自動化することでサーバーレス開発を効率化

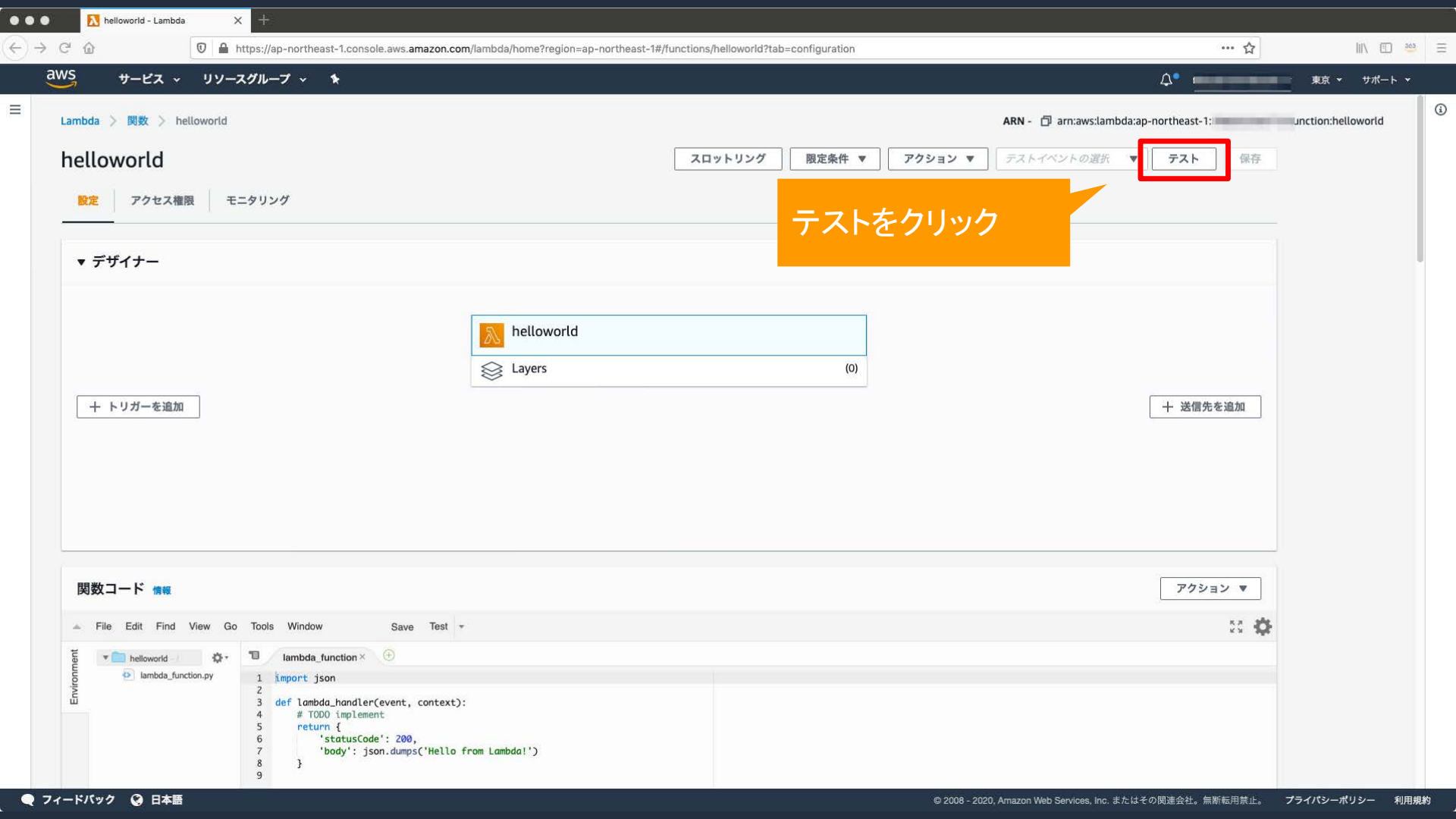
Appendix

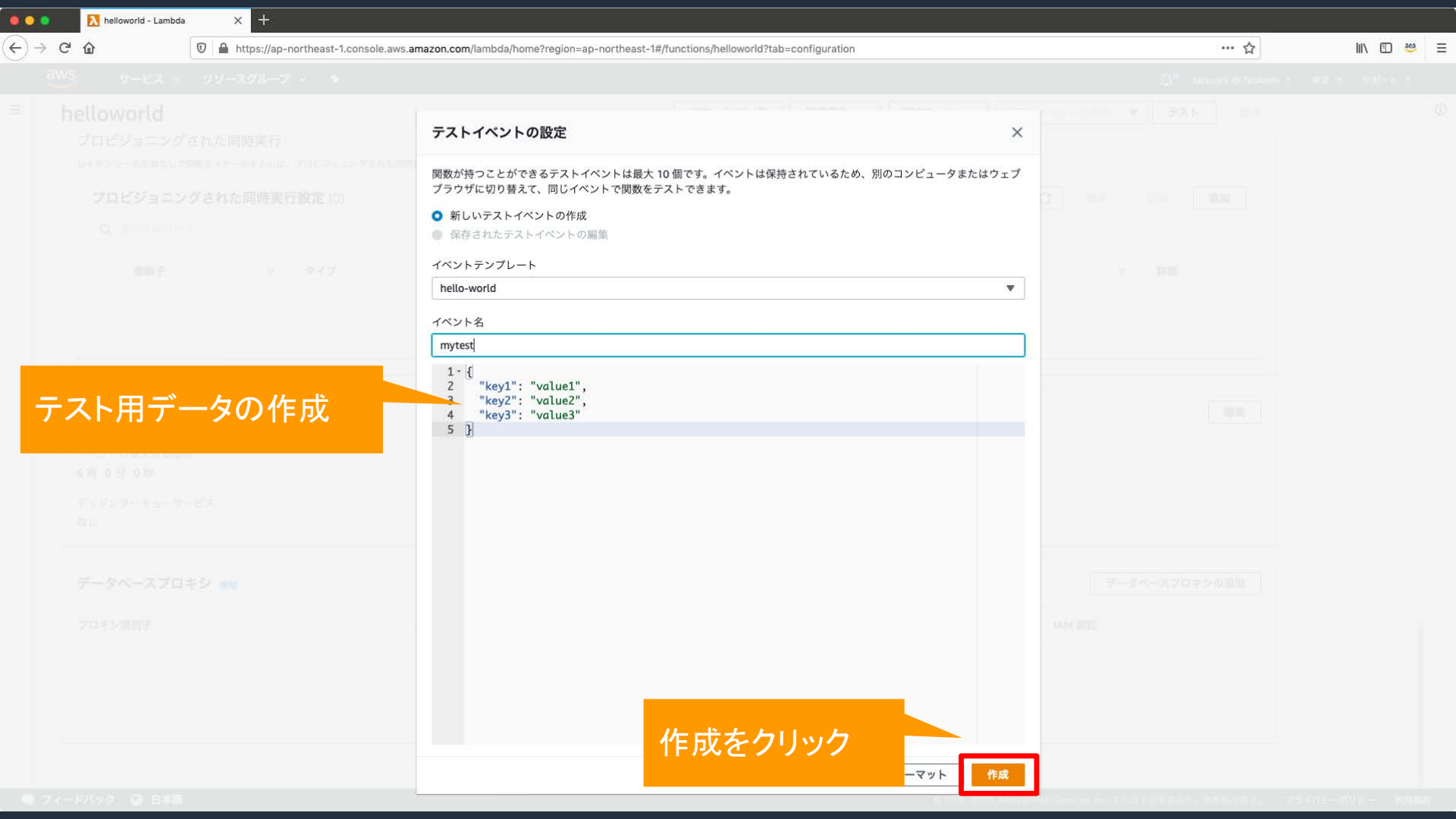
マネージメントコンソールの操作

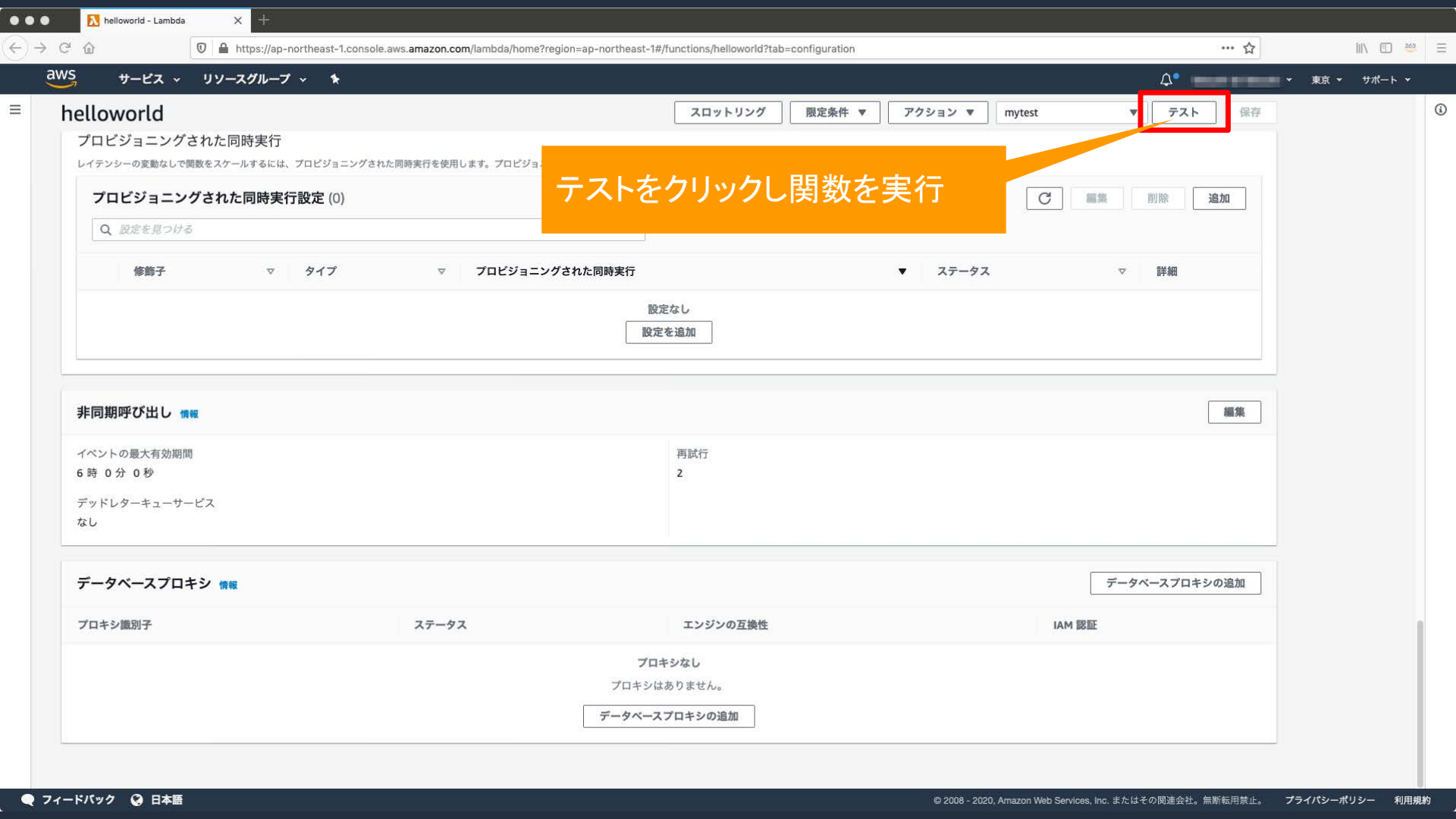

Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/discover
aws
サービス
リソースグループ
東京
サポート
AWS Lambda
ダッシュボード
アプリケーション
関数
Additional resources
レイヤー
開発メニューをクリック
アジアパシフィック (東京) のリソース
関数の作成
Lambda 関数
18
コードのストレージ
47.1 MB
(0%/75.0 GB)
フルアカウントの同時実行
1000
予約されていないアカウントの同時実行
999
ダッシュボードに追加
1 時間
3 時間
12 時間
1 日
3 日
1 週
カスタム
Error count and success rate (%)
Throttles
Invocations
Duration
ConcurrentExecutions
UnreservedConcurrentExecutions
データがありません。
ダッシュボードの時間範囲を調整してみてください。
Errors
Success rate (%)
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

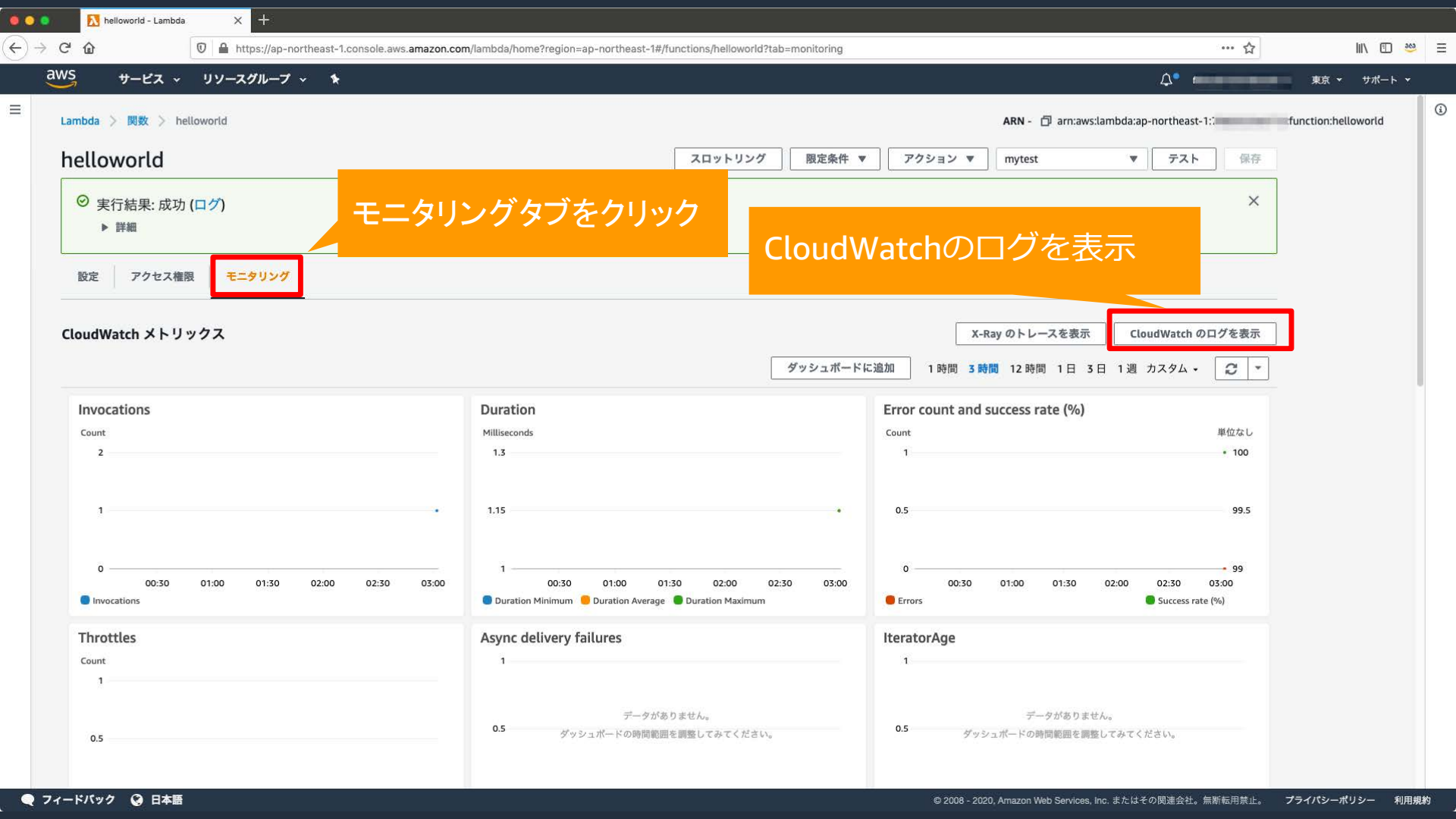

関数 - Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/functions
aws
サービス
リソースグループ
東京
サポート
AWS Lambda
ダッシュボード
アプリケーション
関数
Additional resources
レイヤー
Lambda
関数
関数 (18)
アクション
関数の作成
タグや属性によるフィルター、またはキーワードによる検索
関数の作成をクリック
関数名
説明
コードサイズ
最終更新日時
demo-sleepfunction
A starter AWS Lambda function.
Python 3.7
264 bytes
2 か月前
api-lambda-ddb-cicd-putItemFunction-KZV9HAZGV6R2
A simple example includes a HTTP post method to add one item to a DynamoDB table.
Node.js 10.x
7.0 MB
20 時間前
cdk-api-lambda-ddb-putItemEAA1A834-UZBPL4UC7ZXI
A simple example includes a HTTP post method to add one item to a DynamoDB table.
Node.js 10.x
7.1 MB
19 時間前
api-lambda-ddb-cicd-getByIdFunction-19VXBF4JR1383
A simple example includes a HTTP get method to get one item by id from a DynamoDB table.
Node.js 10.x
7.0 MB
20 時間前
query-employee
Python 3.8
299 bytes
2 か月前
sfn-node-demo-StockBuyerFunction-1K3EH7K6NC5DU
Node.js 12.x
1.0 kB
3 か月前
sfn-node-demo-StockCheckerFunction-529TDDHS03JP
Node.js 12.x
777 bytes
3 か月前
deeplens-event-handler-DeepLensEventHandler-1PKFOR2ALR38D
Python 3.8
1.8 MB
4 か月前
StepFunctionsSample-Nesti-NestingPatternCallbackLa-GABTMTH6TZWR
Node.js 12.x
450 bytes
3 か月前
cdk-api-lambda-ddb-getAllItems7DBFD880-13V229HWYGX1N
A simple example includes a HTTP get method to get all items from a DynamoDB table.
Node.js 10.x
7.1 MB
19 時間前
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

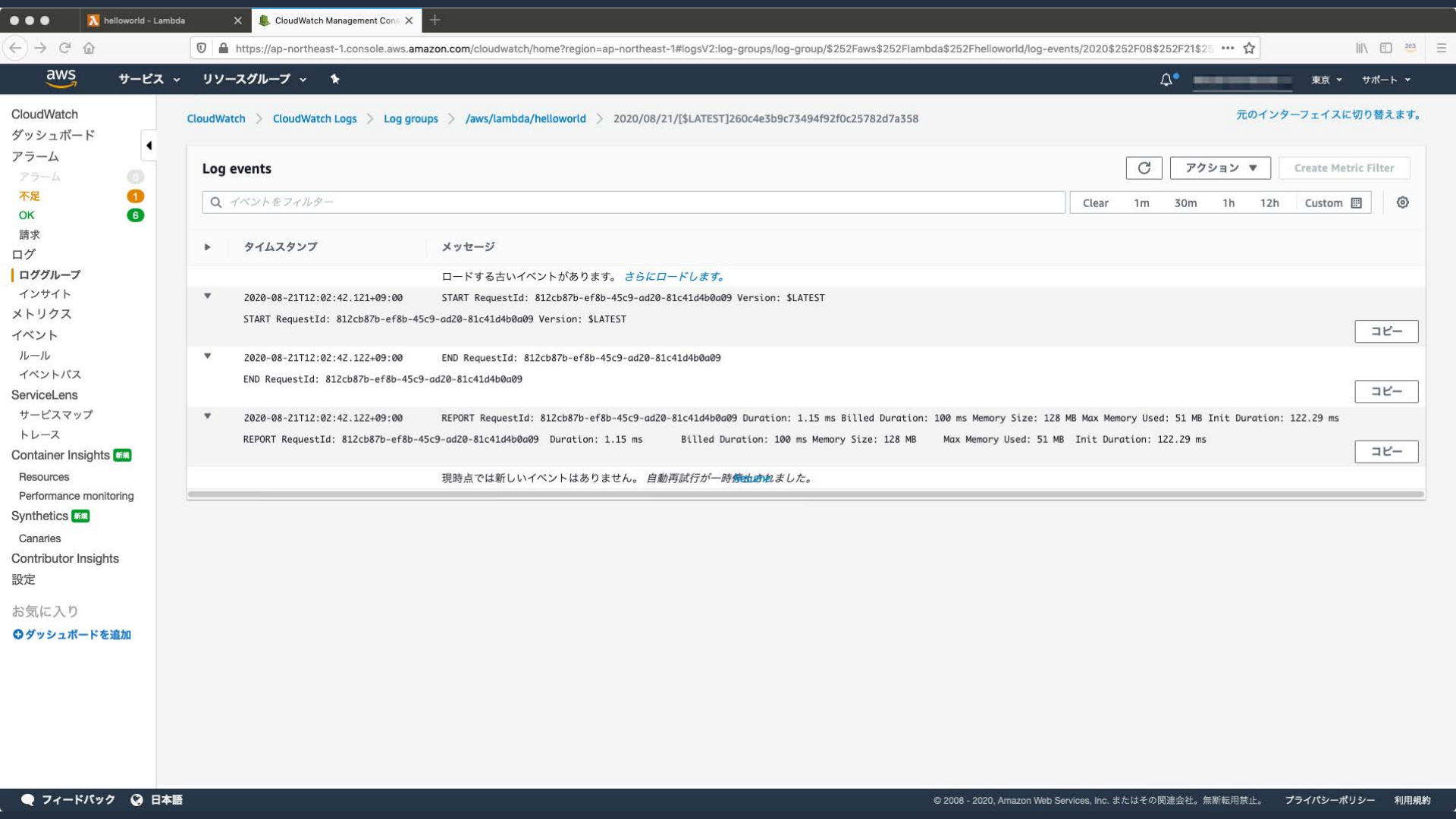

Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/create/function
aws
サービス
リソースグループ
東京
サポート
Lambda > 関数 > 関数の作成
関数の作成 情報
以下のいずれかのオプションを選択して、関数を作成します。
一から作成
シンプルな Hello World の例で開始します。
設計図の使用
一般的ユースケース用のサンプルコードと設定プリセットから Lambda アプリケーションを構築します。
Serverless Application Repository の参照
AWS Serverless Application Repository からサンプル Lambda アプリケーションをデプロイします。
関数名の入力、ランタイムの選択
基本的な情報
関数名
関数の目的を名前として入力します。
helloworld
半角英数字、ハイフン、アンダースコアのみを使用でき、スペースは使用できません。
ランタイム 情報
関数を記述する言語を選択します。
Python 3.8
アクセス権限 情報
Lambda は、Amazon CloudWatch Logs にログをアップロードするアクセス権限を持つ実行ロールを作成します。トリガーを追加すると、アクセス権限をさらに設定および変更できます。
▶ 実行ロールの選択または作成
関数の作成をクリック
キャンセル
関数の作成
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

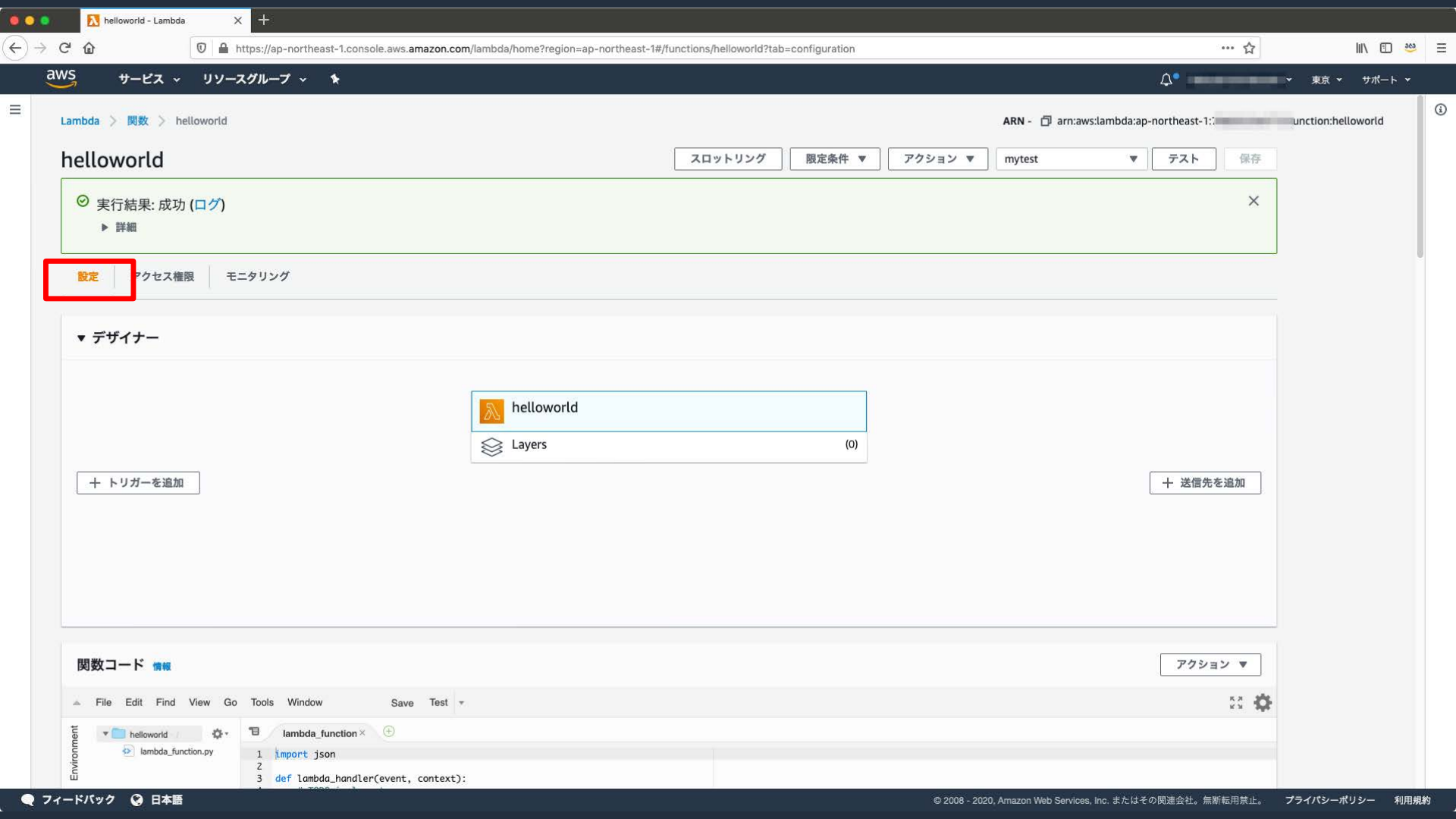

helloworld - Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/functions/helloworld?tab=configuration
aws
サービス
リソースグループ
東京
サポート
Lambda > 関数 > helloworld
ARN - arn:aws:lambda:ap-northeast-1:
unction:helloworld
helloworld
スロットリング
限定条件
アクション
テストイベントの選択
テスト
保存
テストをクリック
設定
アクセス権限
モニタリング
デザイナー
helloworld
Layers
(0)
トリガーを追加
送信先を追加
関数コード 情報
アクション
File Edit Find View Go Tools Window Save Test
Environment
helloworld
lambda_function.py
lambda_function
import json
def lambda_handler(event, context):
# TODO implement
return {
'statusCode': 200,
'body': json.dumps('Hello from Lambda!')
}
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

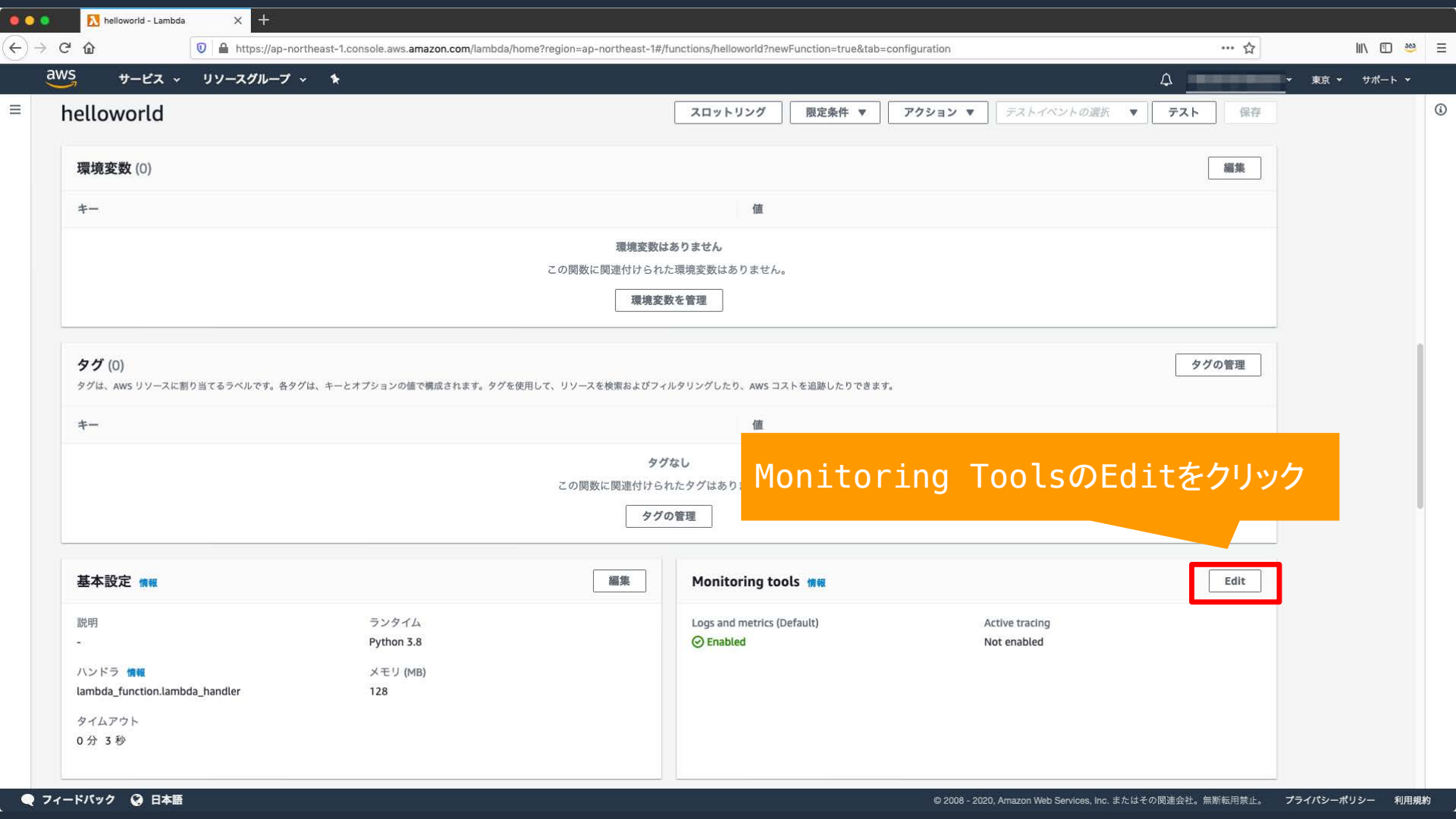

helloworld - Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/functions/helloworld?tab=configuration
helloworld
テストイベントの設定
関数が持つことができるテストイベントは最大 10 個です。イベントは保持されているため、別のコンピュータまたはウェブブラウザに切り替えて、同じイベントで関数をテストできます。
新しいテストイベントの作成
保存されたテストイベントの編集
イベントテンプレート
hello-world
イベント名
mytest
{
"key1": "value1",
"key2": "value2",
"key3": "value3"
}
テスト用データの作成
作成をクリック
作成

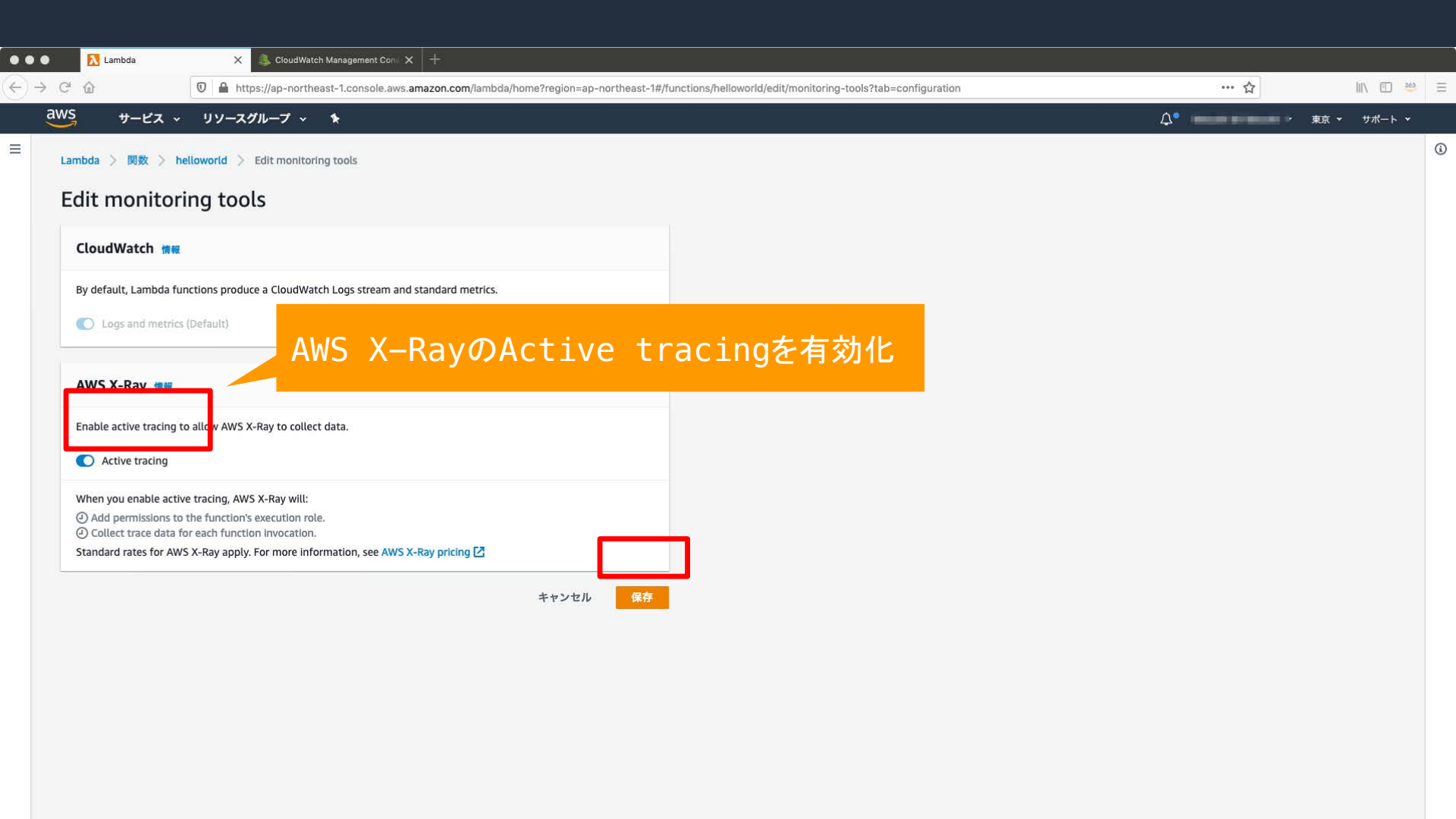

helloworld - Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/functions/helloworld?tab=configuration
aws
サービス
リソースグループ
東京
サポート
helloworld
スロットリング
限定条件
アクション
mytest
テスト
保存
プロビジョニングされた同時実行
レイテンシーの変動なしで関数をスケールするには、プロビジョニングされた同時実行を使用します。プロビジョ
テストをクリックし関数を実行
プロビジョニングされた同時実行設定 (0)
編集
削除
追加
設定を見つける
修飾子
タイプ
プロビジョニングされた同時実行
ステータス
詳細
設定なし
設定を追加
非同期呼び出し 情報
編集
イベントの最大有効期間
6 時 0 分 0 秒
再試行
2
デッドレターキューサービス
なし
データベースプロキシ 情報
データベースプロキシの追加
プロキシ識別子
ステータス
エンジンの互換性
IAM 認証
プロキシなし
プロキシはありません。
データベースプロキシの追加
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

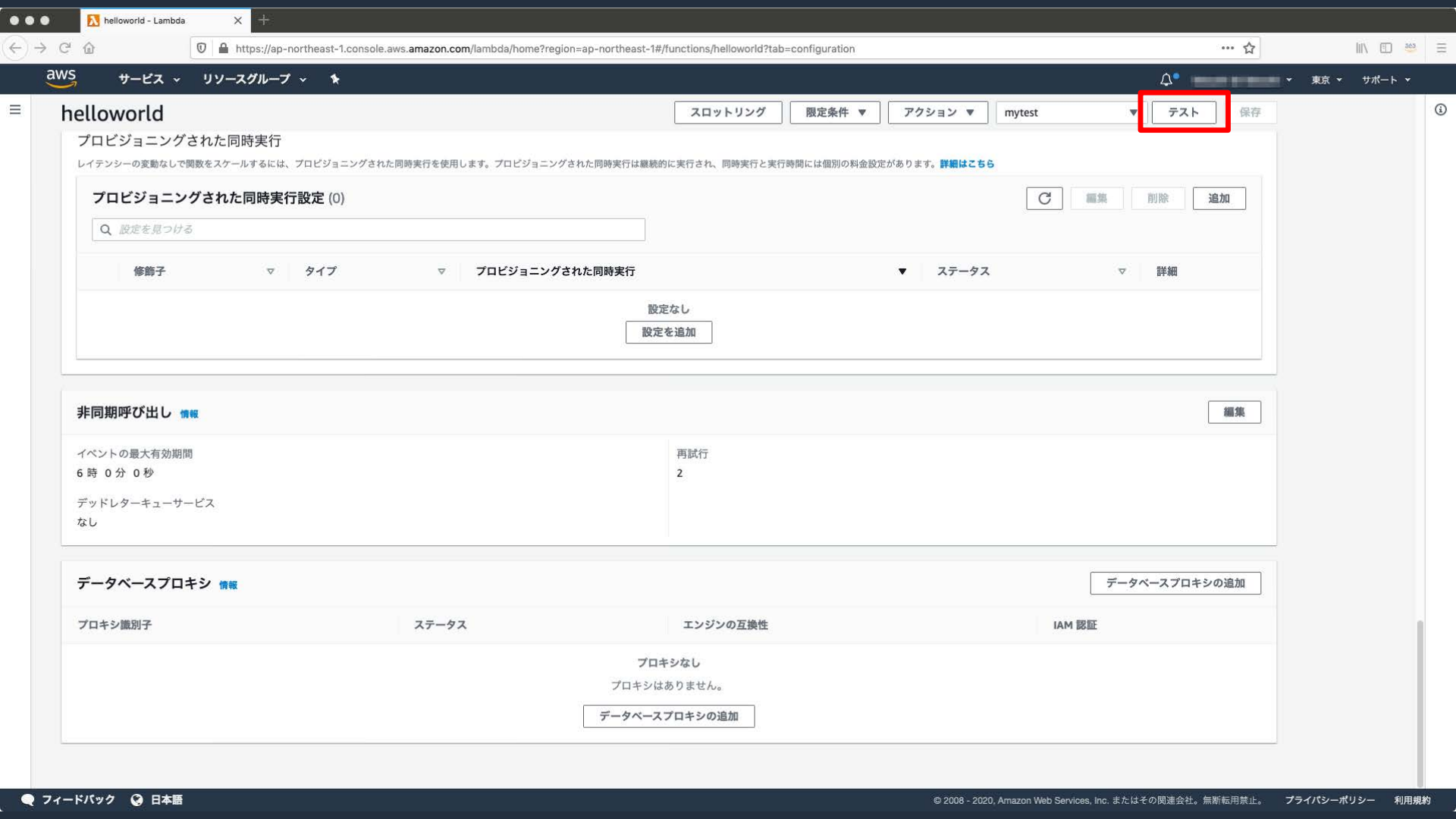

helloworld - Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/functions/helloworld?tab=monitoring
サービス
リソースグループ
東京
サポート
Lambda > 関数 > helloworld
ARN - arn:aws:lambda:ap-northeast-1:[redacted]function:helloworld
helloworld
スロットリング
限定条件
アクション
mytest
テスト
保存
実行結果: 成功 (ログ)
詳細
モニタリングタブをクリック
CloudWatchのログを表示
設定
アクセス権限
モニタリング
CloudWatch メトリックス
X-Ray のトレースを表示
CloudWatch のログを表示
ダッシュボードに追加
1 時間 3 時間 12 時間 1 日 3 日 1 週 カスタム
Invocations
Count
2
1
0
00:30 01:00 01:30 02:00 02:30 03:00
Invocations
Duration
Milliseconds
1.3
1.15
1
00:30 01:00 01:30 02:00 02:30 03:00
Duration Minimum
Duration Average
Duration Maximum
Error count and success rate (%)
Count
単位なし
1
0.5
0
100
99.5
99
00:30 01:00 01:30 02:00 02:30 03:00
Errors
Success rate (%)
Throttles
Count
1
0.5
Async delivery failures
1
0.5
データがありません。
ダッシュボードの時間範囲を調整してみてください。
IteratorAge
1
0.5
データがありません。
ダッシュボードの時間範囲を調整してみてください。
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

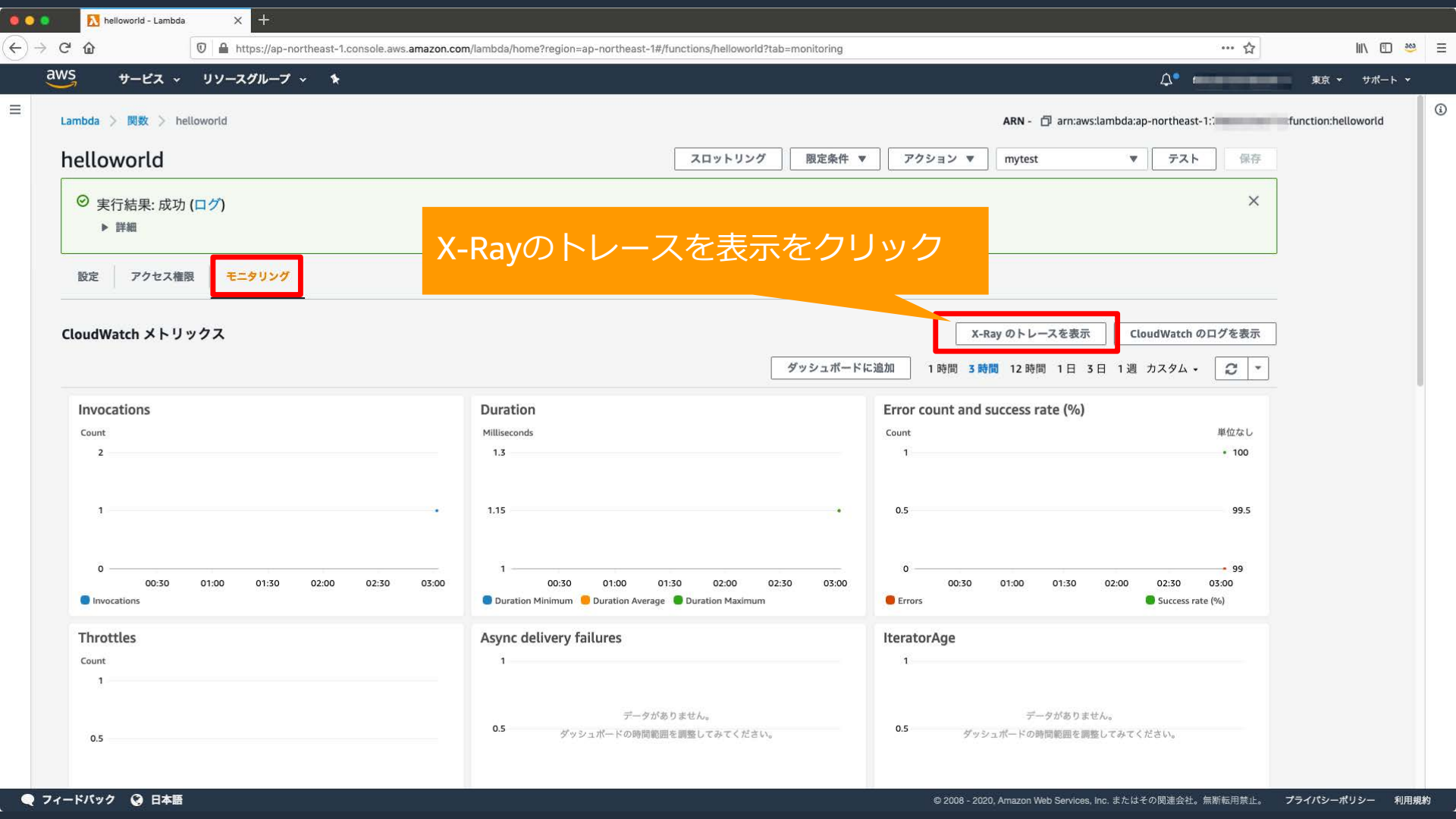

helloworld - Lambda
CloudWatch Management Cons
https://ap-northeast-1.console.aws.amazon.com/cloudwatch/home?region=ap-northeast-1#logsV2:log-groups/log-group/$252Faws$252Flambda$252Fhelloworld/log-events/2020$252F08$252F21$25
aws
サービス
リソースグループ
東京
サポート
CloudWatch
ダッシュボード
アラーム
アラーム 0
不足 1
OK 6
請求
ログ
ロググループ
インサイト
メトリクス
イベント
ルール
イベントバス
ServiceLens
サービスマップ
トレース
Container Insights 新規
Resources
Performance monitoring
Synthetics 新規
Canaries
Contributor Insights
設定
お気に入り
ダッシュボードを追加
CloudWatch > CloudWatch Logs > Log groups > /aws/lambda/helloworld > 2020/08/21/[$LATEST]260c4e3b9c73494f92f0c25782d7a358
元のインターフェイスに切り替えます。
Log events
アクション
Create Metric Filter
イベントをフィルター
Clear
1m
30m
1h
12h
Custom
タイムスタンプ
メッセージ
ロードする古いイベントがあります。 さらにロードします。
2020-08-21T12:02:42.121+09:00 START RequestId: 812cb87b-ef8b-45c9-ad20-81c41d4b0a09 Version: $LATEST
START RequestId: 812cb87b-ef8b-45c9-ad20-81c41d4b0a09 Version: $LATEST
コピー
2020-08-21T12:02:42.122+09:00 END RequestId: 812cb87b-ef8b-45c9-ad20-81c41d4b0a09
END RequestId: 812cb87b-ef8b-45c9-ad20-81c41d4b0a09
コピー
2020-08-21T12:02:42.122+09:00 REPORT RequestId: 812cb87b-ef8b-45c9-ad20-81c41d4b0a09 Duration: 1.15 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 51 MB Init Duration: 122.29 ms
REPORT RequestId: 812cb87b-ef8b-45c9-ad20-81c41d4b0a09 Duration: 1.15 ms Billed Duration: 100 ms Memory Size: 128 MB Max Memory Used: 51 MB Init Duration: 122.29 ms
コピー
現時点では新しいイベントはありません。 自動再試行が一時停止されました。
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

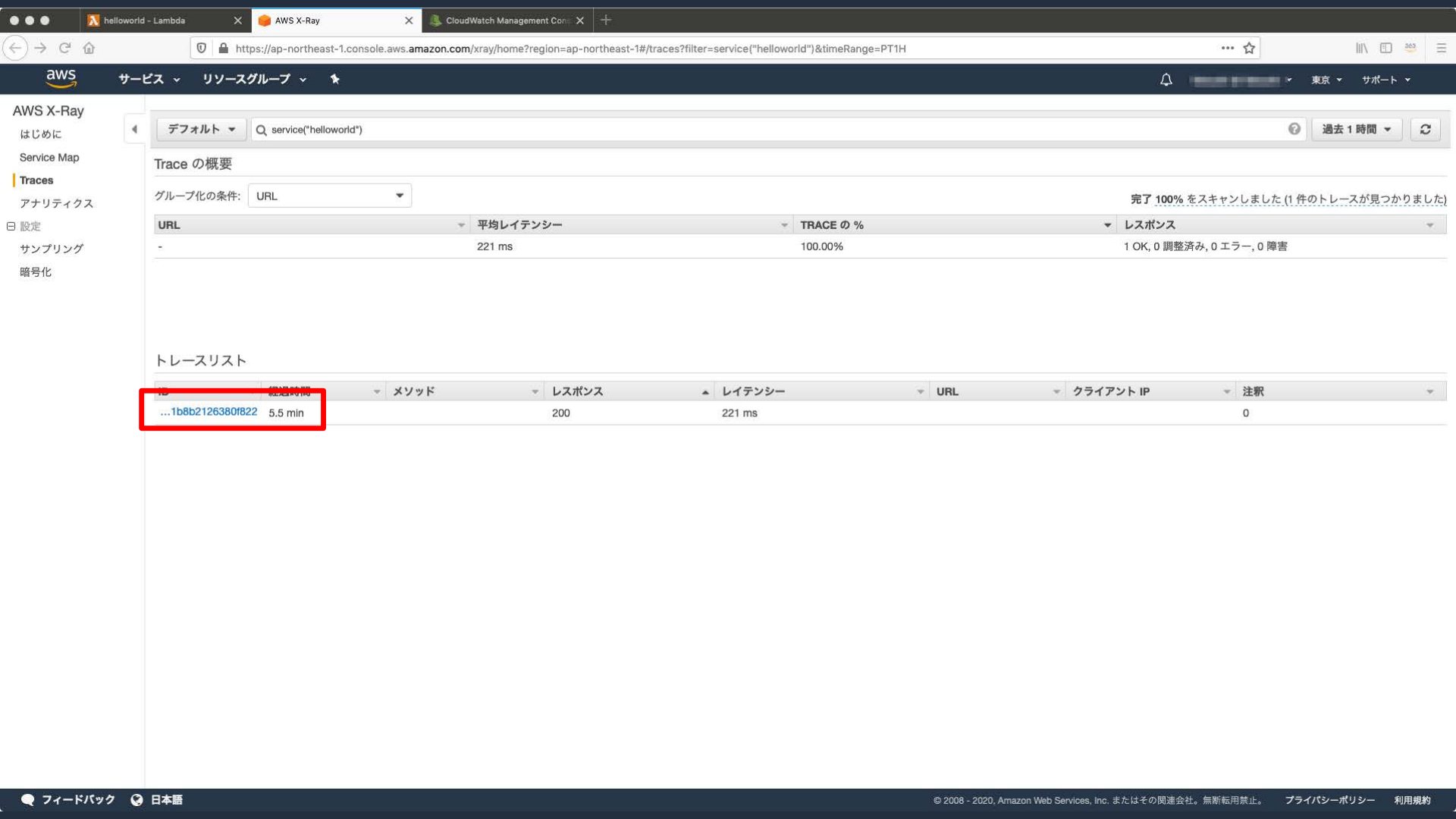

helloworld - Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/functions/helloworld?tab=configuration
aws
サービス
リソースグループ
東京
サポート
Lambda > 関数 > helloworld
ARN - arn:aws:lambda:ap-northeast-1:
unction:helloworld
helloworld
スロットリング
限定条件
アクション
mytest
テスト
保存
実行結果: 成功 (ログ)
詳細
設定
アクセス権限
モニタリング
デザイナー
helloworld
Layers
(0)
トリガーを追加
送信先を追加
関数コード 情報
アクション
File Edit Find View Go Tools Window
Save
Test
Environment
helloworld
lambda_function.py
lambda_function
1 import json
2
3 def lambda_handler(event, context):
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約


helloworld - Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/functions/helloworld?newFunction=true&tab=configuration
aws
サービス
リソースグループ
東京
サポート
helloworld
スロットリング
限定条件
アクション
テストイベントの選択
テスト
保存
環境変数 (0)
編集
キー
値
環境変数はありません
この関数に関連付けられた環境変数はありません。
環境変数を管理
タグ (0)
タグは、AWS リソースに割り当てるラベルです。各タグは、キーとオプションの値で構成されます。タグを使用して、リソースを検索およびフィルタリングしたり、AWS コストを追跡したりできます。
タグの管理
キー
値
タグなし
この関数に関連付けられたタグはあり
タグの管理
Monitoring ToolsのEditをクリック
基本設定 情報
編集
説明
-
ランタイム
Python 3.8
ハンドラ 情報
lambda_function.lambda_handler
メモリ (MB)
128
タイムアウト
0 分 3 秒
Monitoring tools 情報
Edit
Logs and metrics (Default)
Enabled
Active tracing
Not enabled
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約


Lambda
CloudWatch Management Con
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/functions/helloworld/edit/monitoring-tools?tab=configuration
aws
サービス
リソースグループ
東京
サポート
Lambda > 関数 > helloworld > Edit monitoring tools
Edit monitoring tools
CloudWatch 情報
By default, Lambda functions produce a CloudWatch Logs stream and standard metrics.
Logs and metrics (Default)
AWS X-Ray 情報
Enable active tracing to allow AWS X-Ray to collect data.
Active tracing
When you enable active tracing, AWS X-Ray will:
Add permissions to the function's execution role.
Collect trace data for each function invocation.
Standard rates for AWS X-Ray apply. For more information, see AWS X-Ray pricing
キャンセル
保存
AWS X-RayのActive tracingを有効化


helloworld - Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/functions/helloworld?tab=configuration
aws
サービス
リソースグループ
東京
サポート
helloworld
スロットリング
限定条件
アクション
mytest
テスト
保存
プロビジョニングされた同時実行
レイテンシーの変動なしで関数をスケールするには、プロビジョニングされた同時実行を使用します。プロビジョニングされた同時実行は継続的に実行され、同時実行と実行時間には個別の料金設定があります。詳細はこちら
プロビジョニングされた同時実行設定 (0)
編集
削除
追加
設定を見つける
修飾子
タイプ
プロビジョニングされた同時実行
ステータス
詳細
設定なし
設定を追加
非同期呼び出し 情報
編集
イベントの最大有効期間
6 時 0 分 0 秒
再試行
2
デッドレターキューサービス
なし
データベースプロキシ 情報
データベースプロキシの追加
プロキシ識別子
ステータス
エンジンの互換性
IAM 認証
プロキシなし
プロキシはありません。
データベースプロキシの追加
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約


helloworld - Lambda
https://ap-northeast-1.console.aws.amazon.com/lambda/home?region=ap-northeast-1#/functions/helloworld?tab=monitoring
サービス
リソースグループ
東京
サポート
Lambda > 関数 > helloworld
ARN - arn:aws:lambda:ap-northeast-1:[redacted]function:helloworld
helloworld
スロットリング
限定条件
アクション
mytest
テスト
保存
実行結果: 成功 (ログ)
詳細
X-Rayのトレースを表示をクリック
設定
アクセス権限
モニタリング
CloudWatch メトリックス
X-Ray のトレースを表示
CloudWatch のログを表示
ダッシュボードに追加
1 時間 3 時間 12 時間 1 日 3 日 1 週 カスタム
Invocations
Count
2
1
0
00:30 01:00 01:30 02:00 02:30 03:00
Invocations
Duration
Milliseconds
1.3
1.15
1
00:30 01:00 01:30 02:00 02:30 03:00
Duration Minimum Duration Average Duration Maximum
Error count and success rate (%)
Count
単位なし
1
0.5
0
100
99.5
99
00:30 01:00 01:30 02:00 02:30 03:00
Errors
Success rate (%)
Throttles
Count
1
0.5
Async delivery failures
1
0.5
データがありません。
ダッシュボードの時間範囲を調整してみてください。
IteratorAge
1
0.5
データがありません。
ダッシュボードの時間範囲を調整してみてください。
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約


helloworld - Lambda
AWS X-Ray
CloudWatch Management Con
https://ap-northeast-1.console.aws.amazon.com/xray/home?region=ap-northeast-1#/traces?filter=service("helloworld")&timeRange=PT1H
aws
サービス
リソースグループ
東京
サポート
AWS X-Ray
はじめに
Service Map
Traces
アナリティクス
設定
サンプリング
暗号化
デフォルト
service("helloworld")
過去 1 時間
Trace の概要
グループ化の条件:
URL
完了 100% をスキャンしました (1 件のトレースが見つかりました)
URL
平均レイテンシー
TRACE の %
レスポンス
-
221 ms
100.00%
1 OK, 0 調整済み, 0 エラー, 0 障害
トレースリスト
ID
メソッド
レスポンス
レイテンシー
URL
クライアント IP
注釈
...1b8b2126380f822
5.5 min
200
221 ms
0
フィードバック
日本語
© 2008 - 2020, Amazon Web Services, Inc. またはその関連会社。無断転用禁止。
プライバシーポリシー
利用規約

helloworld - Lambda | AWS X-Ray | CloudWatch Management Con

https://ap-northeast-1.console.aws.amazon.com/xray/home?region=ap-northeast-1#/traces/1-5f3f3bde-2ea593d011b8b2126380f822

aws サービス リソースグループ 東京 サポート

AWS X-Ray
- はじめに
- Service Map
- Traces
- アナリティクス
- 設定
- サンプリング
- 暗号化

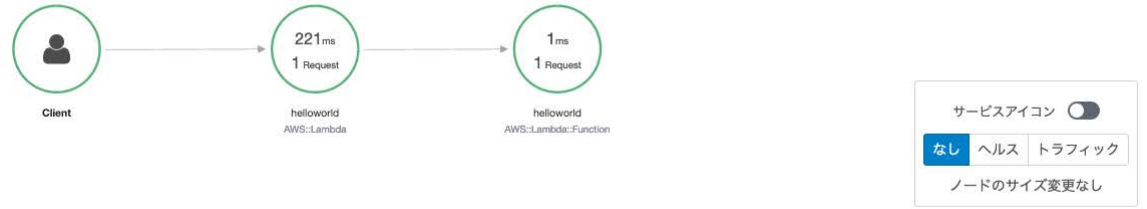
1-5f3f3bde-2ea593d011b8b2126380f822

Traces > 詳細

タイムライン | Raw データ

メソッド	レスポンス	所要時間	期間	ID
--	200	221 ms	6.5 min (2020-08-21 03:13:34 UTC)	1-5f3f3bde-2ea593d011b8b2126380f822

Trace Map



名前	レスポンス	所要時間	ステータス	0.0ms 20ms 40ms 60ms 80ms 100ms 120ms 140ms 160ms 180ms 200ms 220ms 240ms
helloworld AWS::Lambda				
helloworld	200	221 ms	✓	
helloworld AWS::Lambda::Function				
helloworld	-	1.2 ms	✓	
Initialization	-	120 ms	✓	
Invocation	-	0.6 ms	✓	
Overhead	-	0.1 ms	✓	

フィードバック 日本語

プライバシーポリシー 利用規約