サーバーレス開発:環境準備・CI/CD

Beyond Beginner

@_kensh

Who am I?

- Name
  - Kensuke Shimokawa
- Company
  - Amazon Web Services Japan K.K.
- Role
  - Serverless Specialist Solutions Architect

@_kensh

Agenda

- Serverlessの開発環境を整える
- Serverlessのデプロイ環境を整える
- ServerlessのCI/CDパイプラインを整える
- まとめ

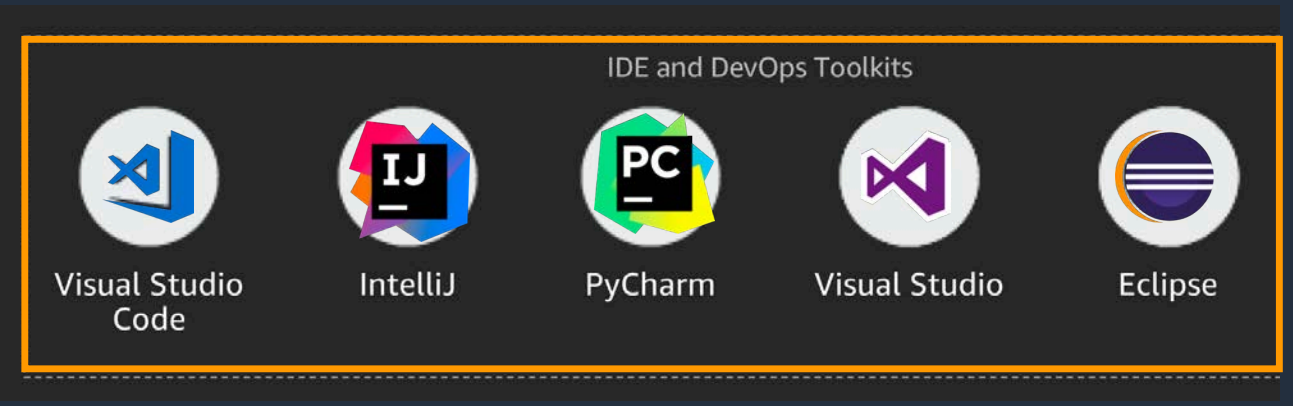
@_kensh

Serverlessの開発環境を整える

AWS提供の総合的なツールセット

AWS提供の総合的なツールセット

統合開発環境(IDE)



+

AWS Toolkit

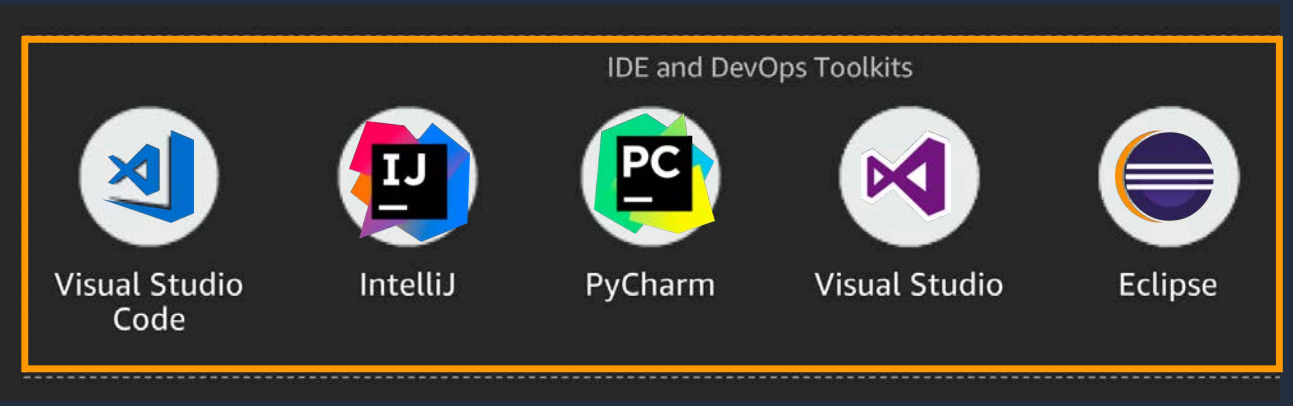
AWS Toolkit

- 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、デプロイを容易にします。
- 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します。

AWS Toolkit

AWS提供の総合的なツールセット

AWS Toolkit 対応 IDE



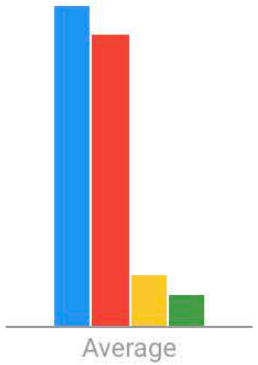
- 雛形からプロジェクトを作成
- ステップ実行などのデバッグ支援
- IDE/エディタからのデプロイ

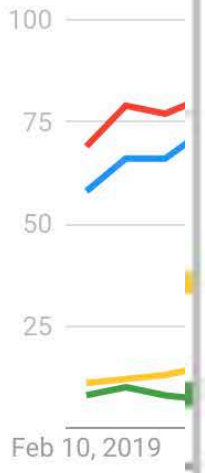
Visual Studio Code
Software

Japan

Past 12 month

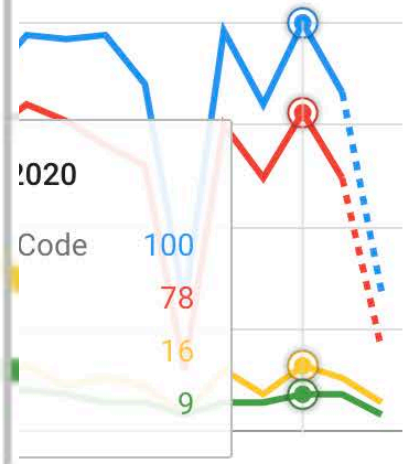
Interest over time





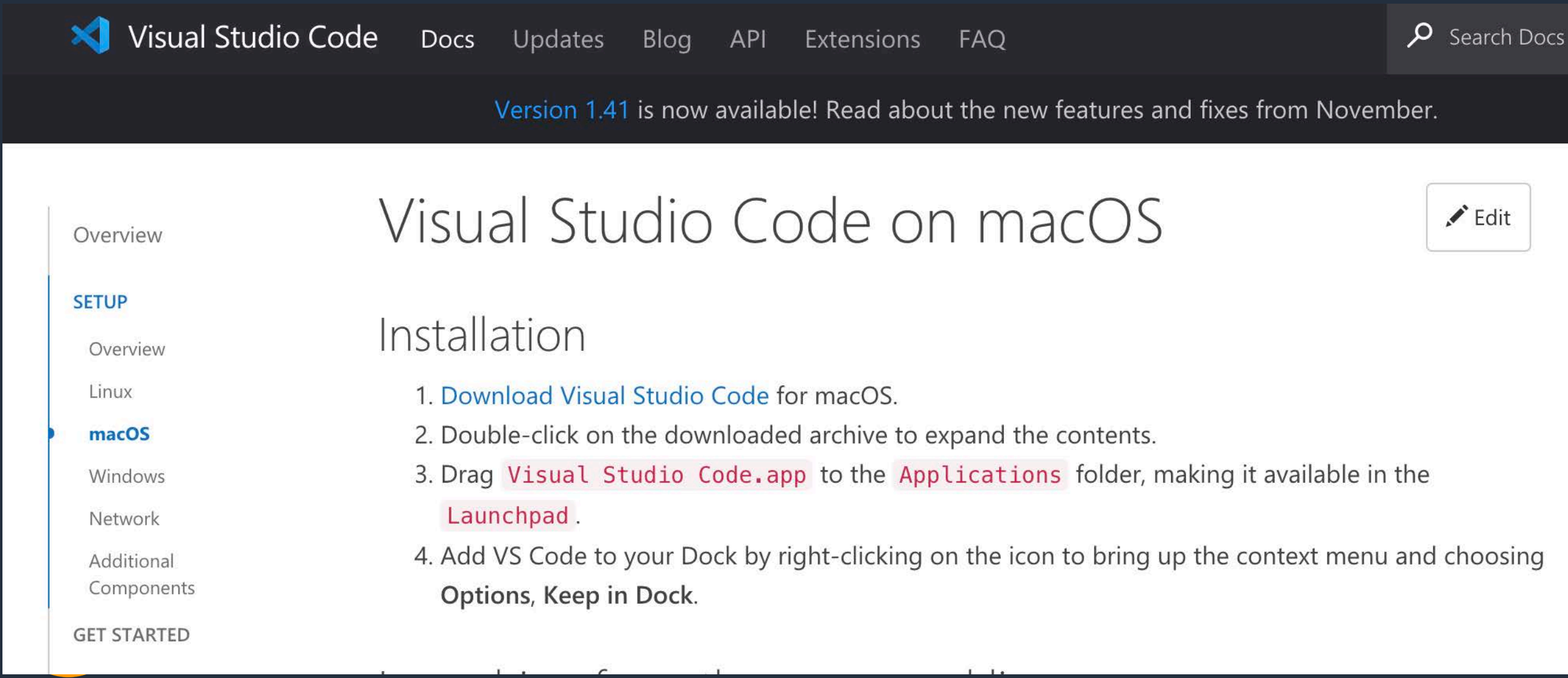
Jan 19 – 25, 2020

Visual Studio Code	100
Eclipse	78
IntelliJ IDEA	16
PyCharm	9

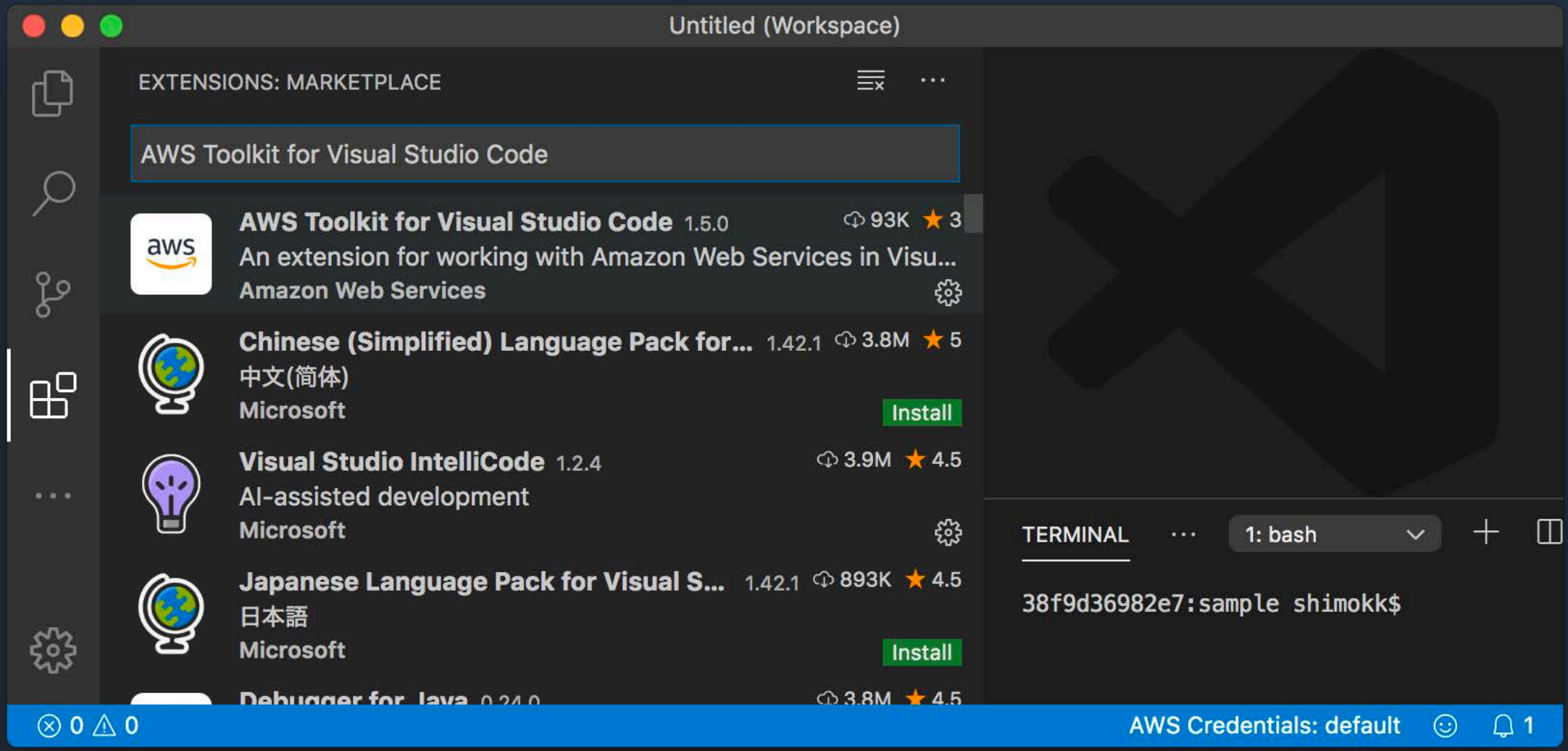


How to install Visual Studio Code

https://code.visualstudio.com/docs/setup/mac

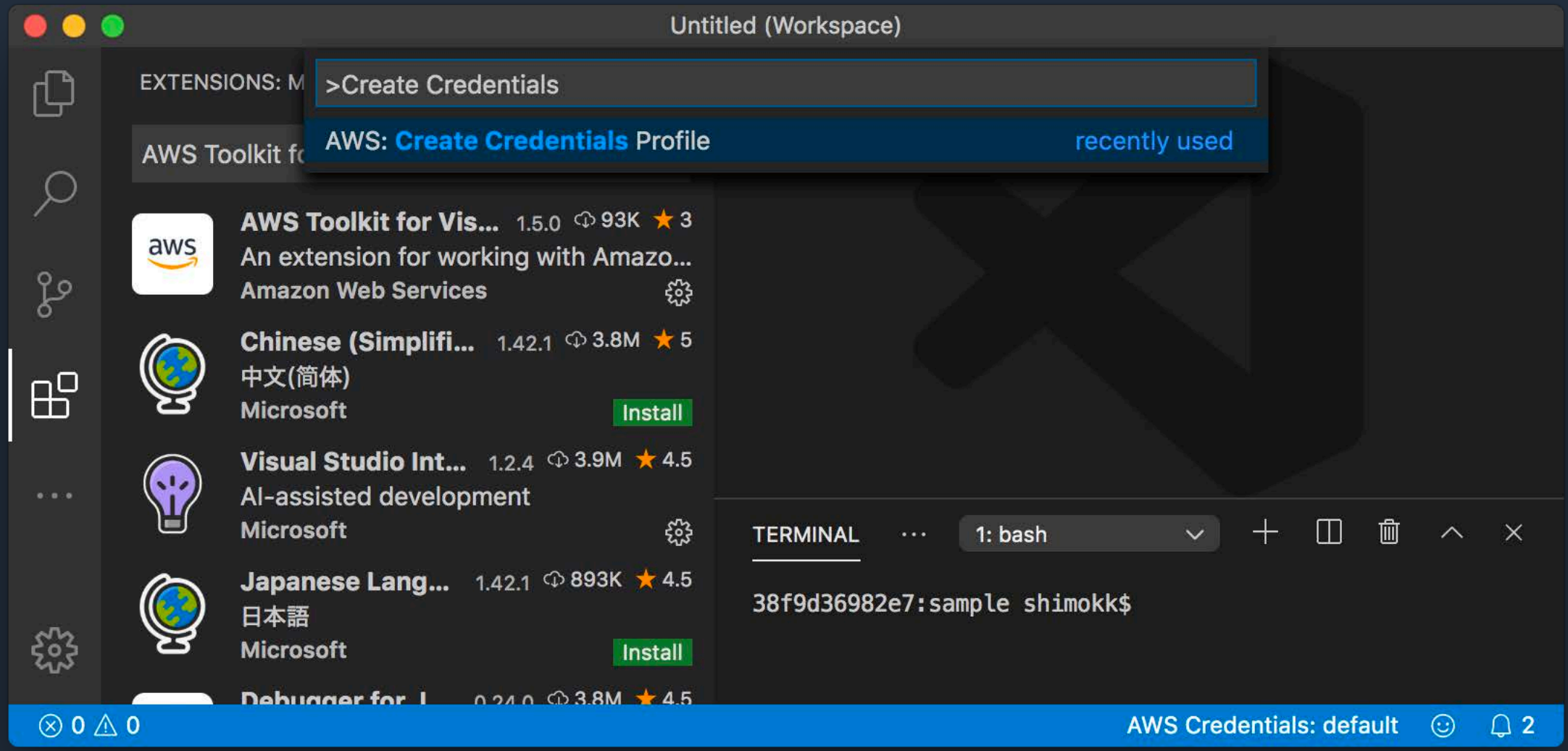


「AWS Toolkit for Visual Studio Code」を検索してInstall

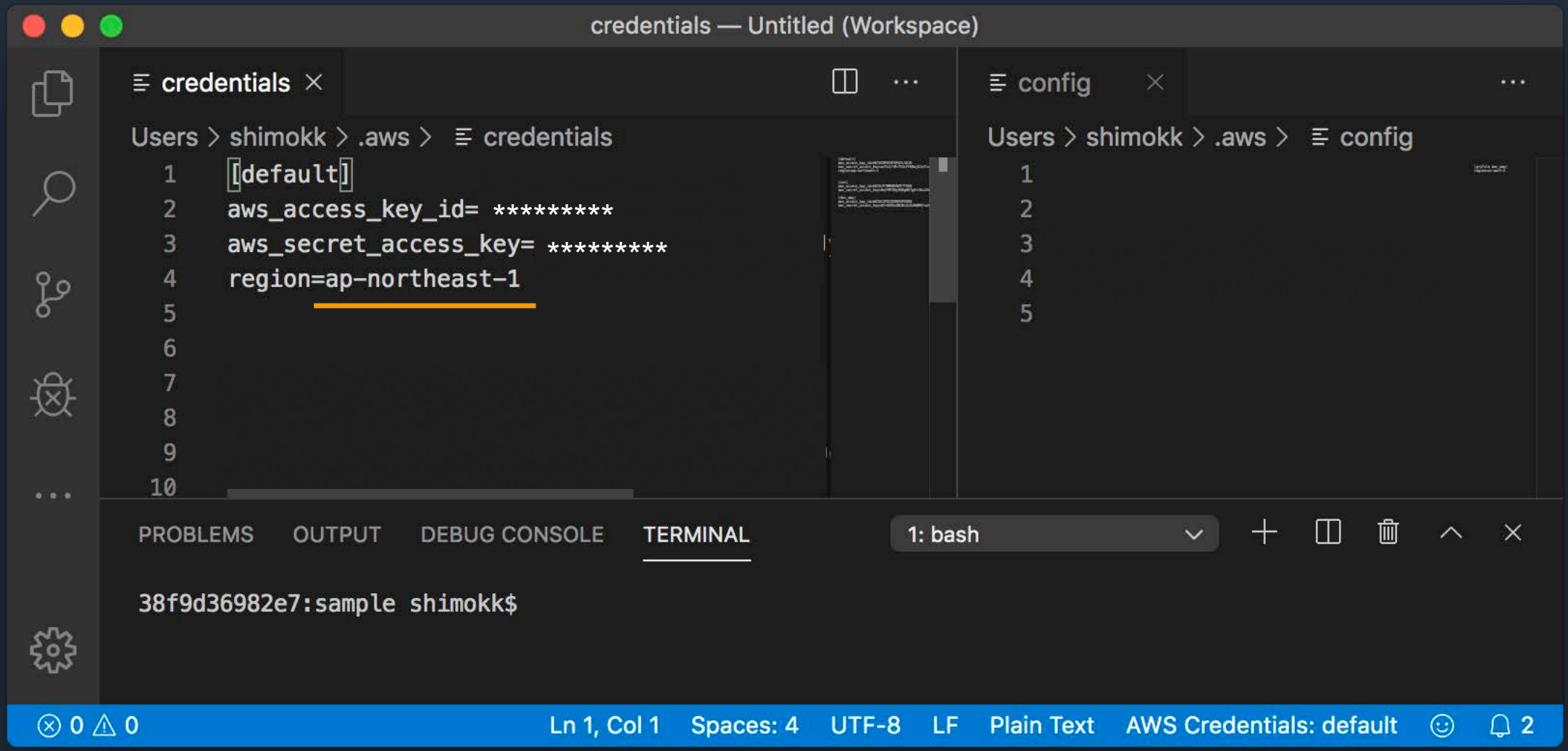


Credentials の設定

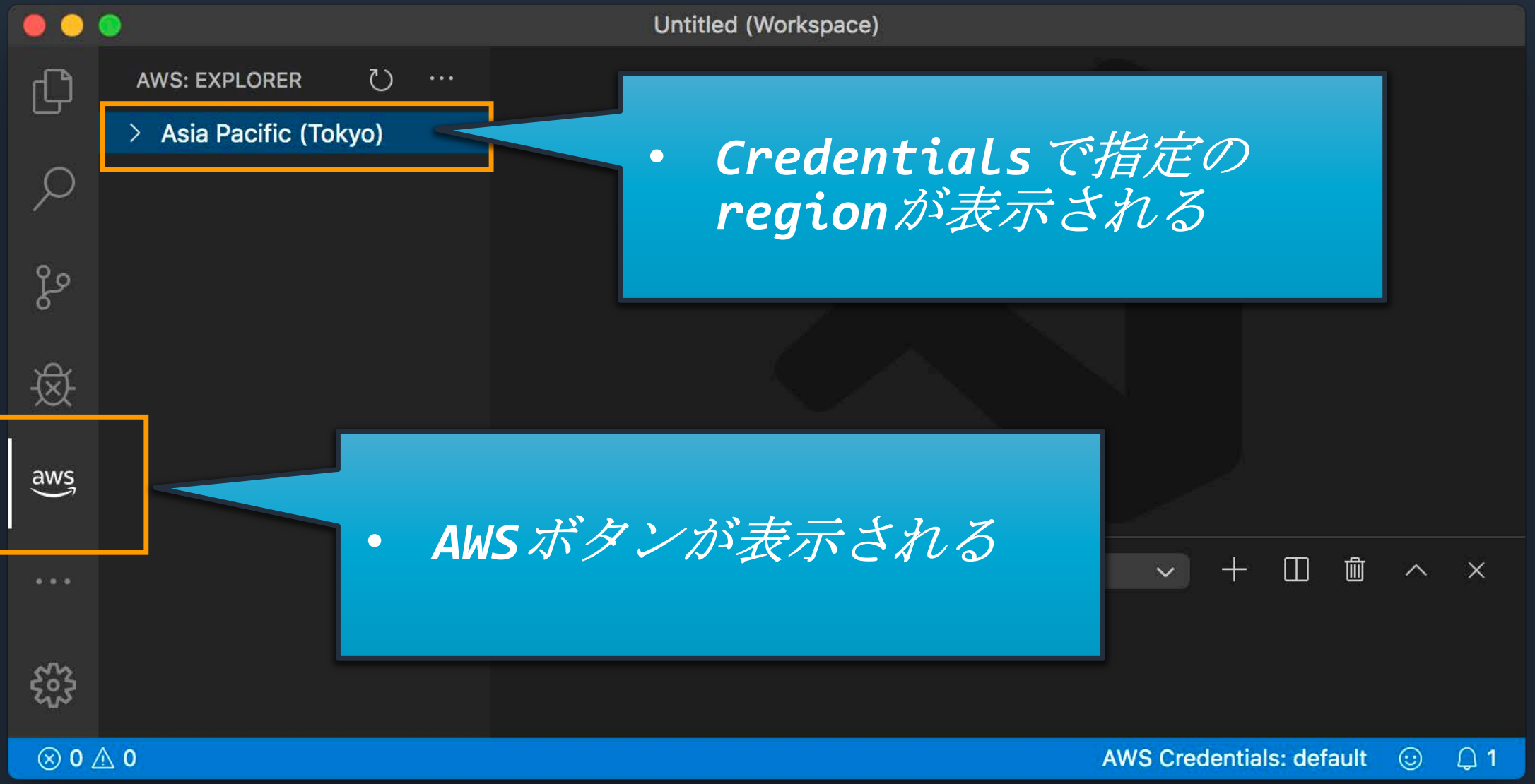
Command+ Shift + P



Credentials の設定



AWS Toolkit 対応 IDE/エディタ

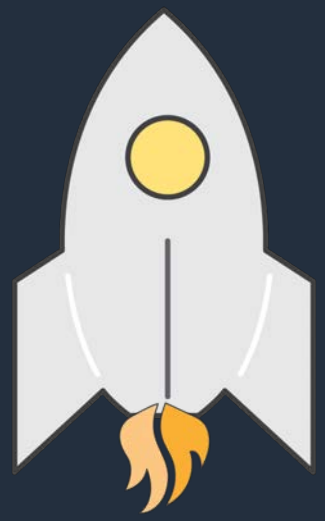


アカウント上のリソースを確認

- *Tokyo-Region の Lambda* リソースが確認できる。
- 他にも*Cloud Formation Stack*や、*EventBridge*の*Schemas*も確認できる。

Serverlessのデプロイ環境を整える

従来のインフラ構築/管理の課題

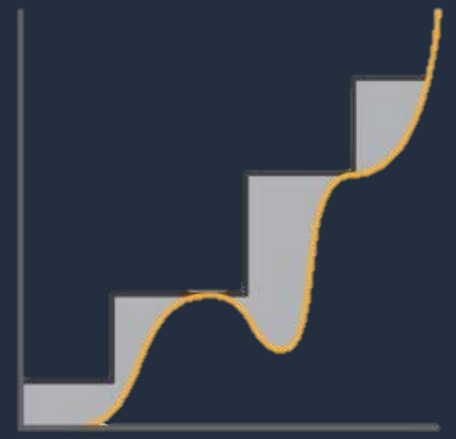


スピード

調達、構築に時間がかかる

安全性

ヒューマンエラーによる構成の不一致



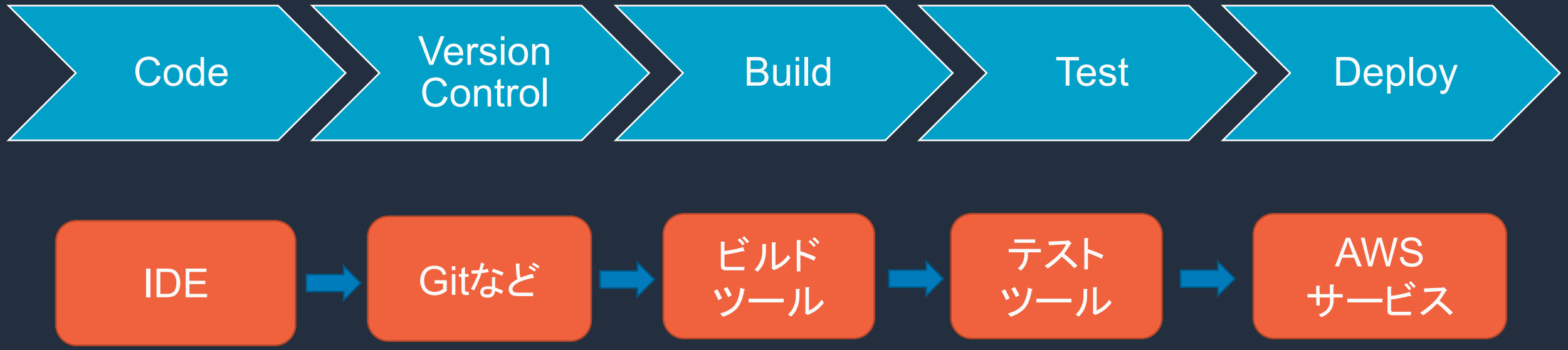
効率性

必要に応じた設定変更ができない

Infrastructure-as-Code

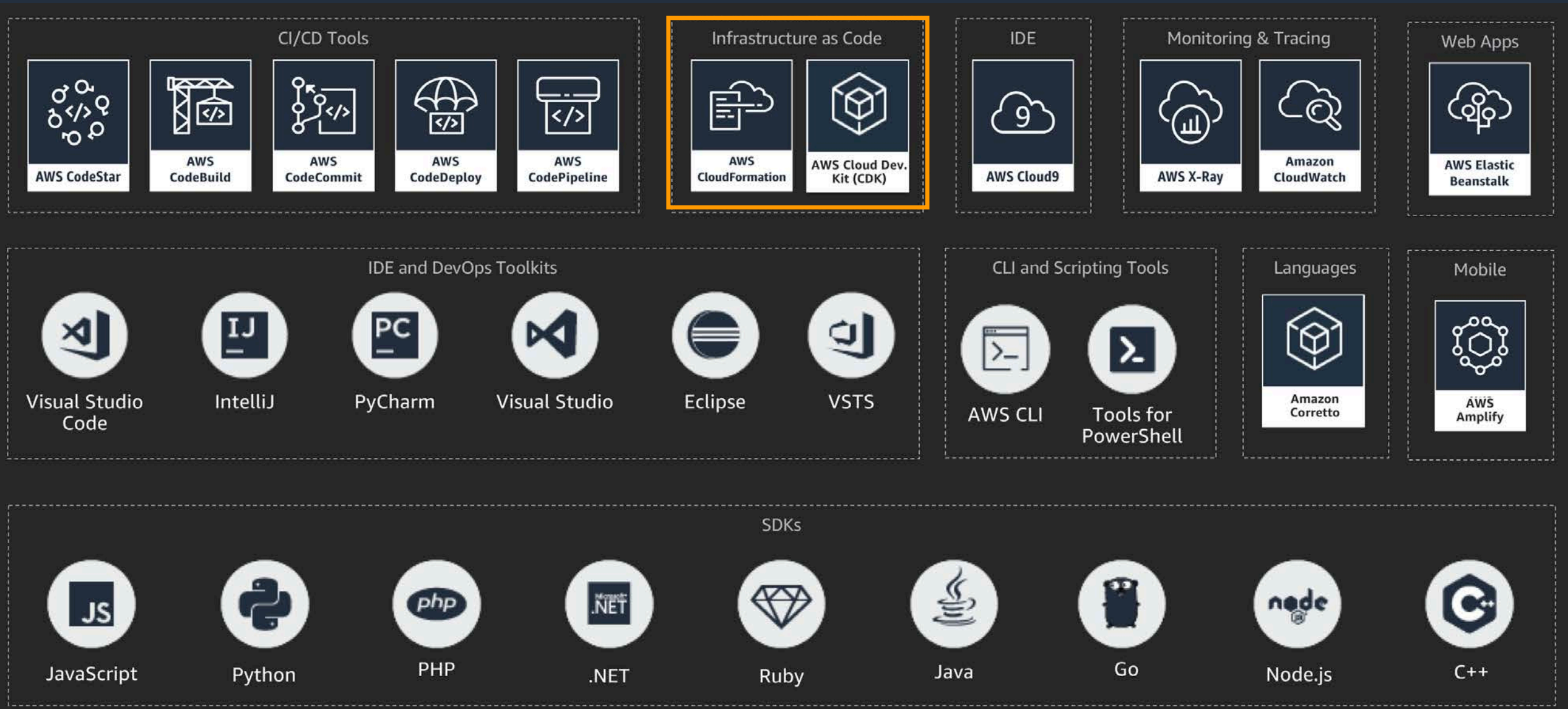
“インフラをコード化、ソフトウェアのように管理”

Infrastructure-as-Code ワークフロー



“It’s all software”

AWS提供の総合的なツールセット



AWS Serverless Application Model

- AWS Serverless Application Model は、サーバーレス構築用のInfrastructure as Codeフレームワークです。

略して SAM と呼ばれます

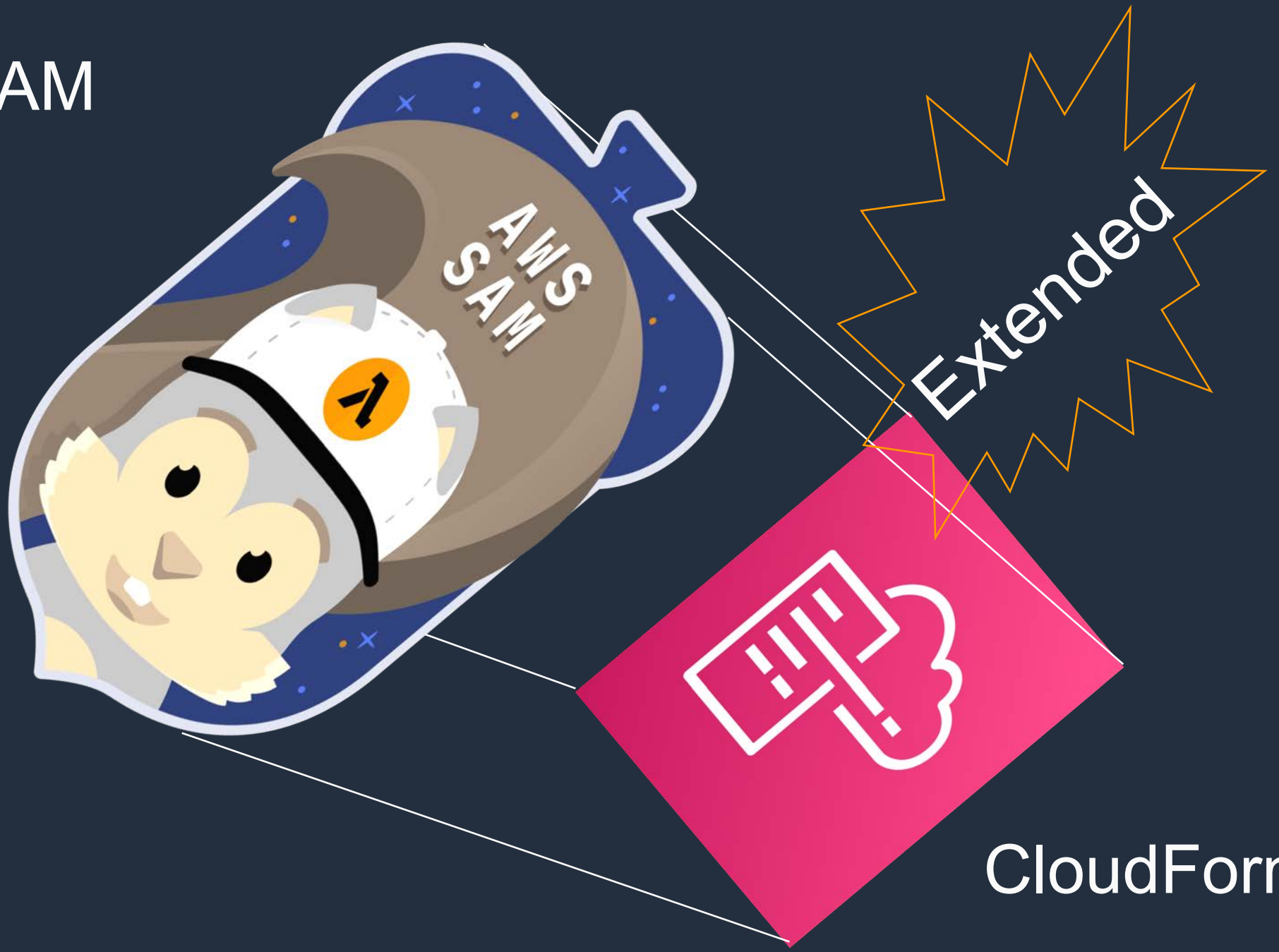
AWSのIaCって、AWS CloudFormationがあったよね？

AWS CloudFormation

おさらい

- AWSの環境構築を、設定ファイル(テンプレート) で自動化するサービス
- システム構成をJSONやYAMLフォーマットのテキストで記述
- ベストプラクティスが盛り込まれた各種サンプルテンプレートを利用可能

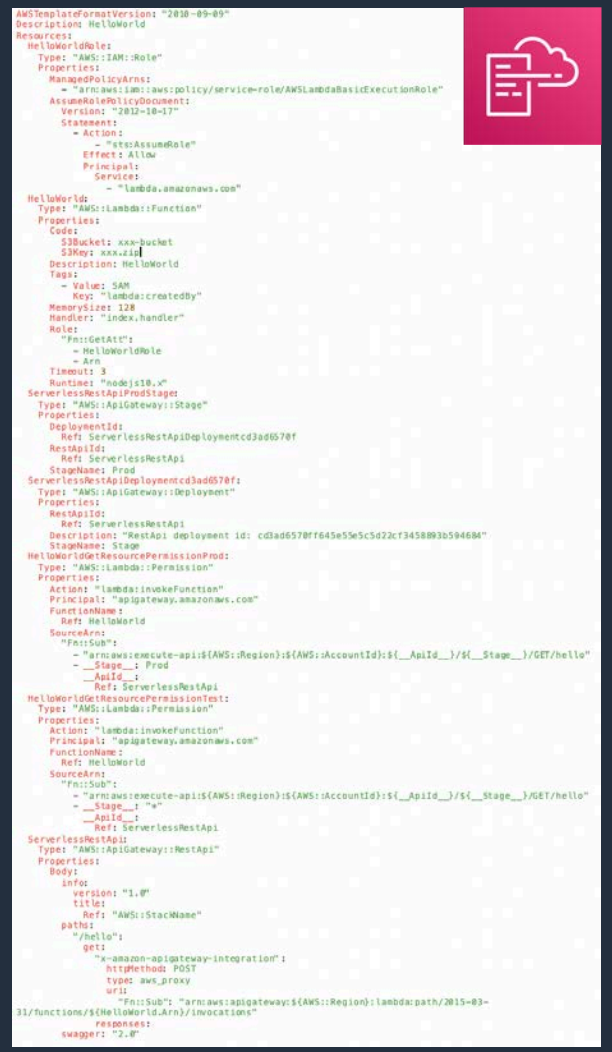
AWS SAM



SAMで、より簡潔にサーバーレス アプリを定義できる

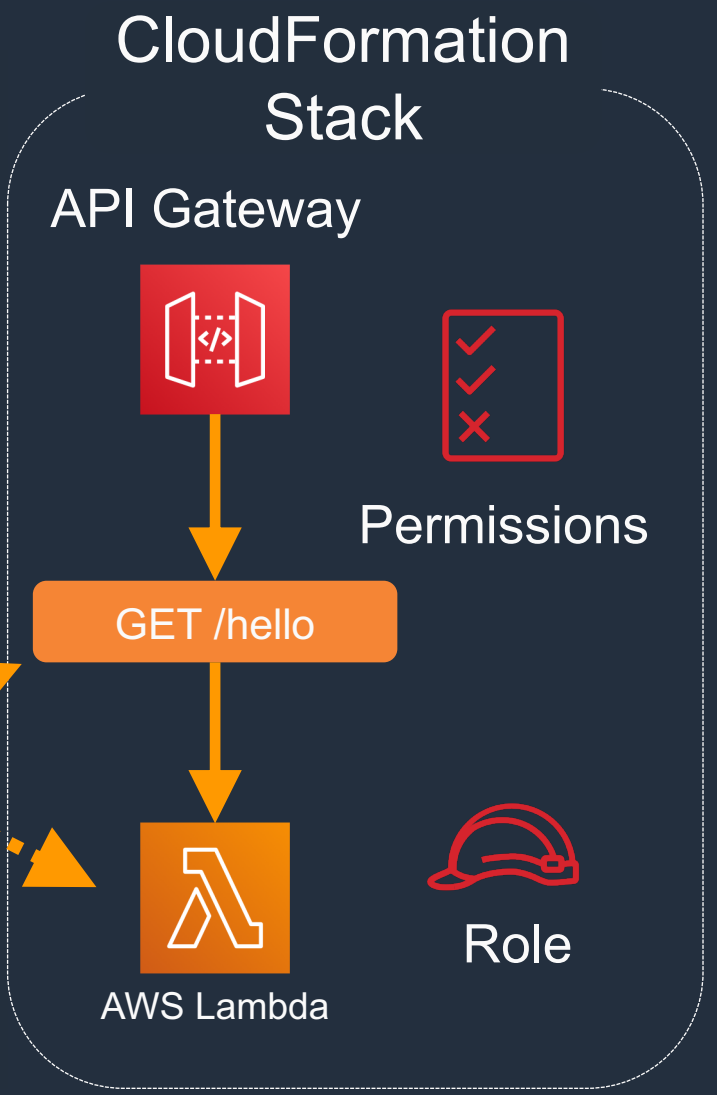
```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: HelloWorld
Resources:
  HelloWorld:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: index.handler
      Runtime: python3.8
      CodeUri: src/handlers/func1
      Description: HelloWorld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

AWS CloudFormation

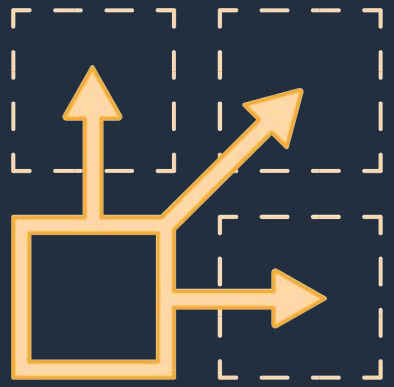


SAMで、より簡潔にサーバーレス アプリを定義できる

```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:
        GetResource:
          Type: Api
          Properties:
            Path: /hello
            Method: get
```

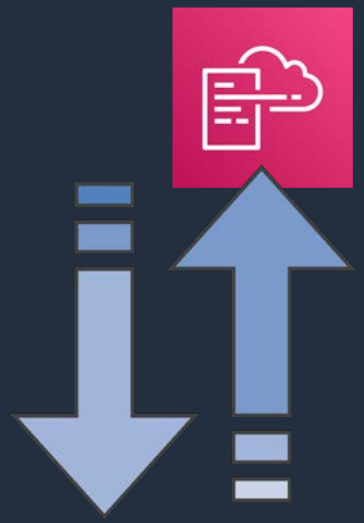


AWS SAM

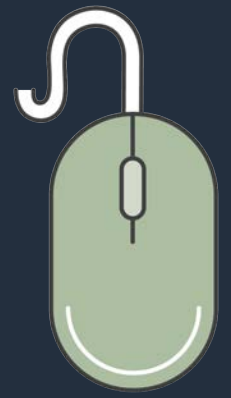


リソース参照による
効率良い記述

ローカルでのテスト
およびデバッグ



CloudFormation
文法での構築も可能



組み込みの
ベストプラクティス
での簡単な構築

Amazon Web Services ブログ

https://go.aws/3bzerVZ

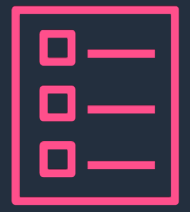
[AWS Black Belt Online Seminar] AWS Serverless Application Model 資料及び QA 公開

by AWS Japan Staff | on 23 AUG 2019 | in AWS Lambda, AWS Serverless Application Model, Webinars | Permalink | Share

先日 (2019/08/14) 開催しました AWS Black Belt Online Seminar「AWS Serverless Application Model」の資料を公開しました。当日、参加者の皆様から頂いた QA の一部についても共有しております。

SAMによるServerless構築の流れ

SAM
template



ロード

SAM

CloudFormation

トランスパイル

作成/変更/削除

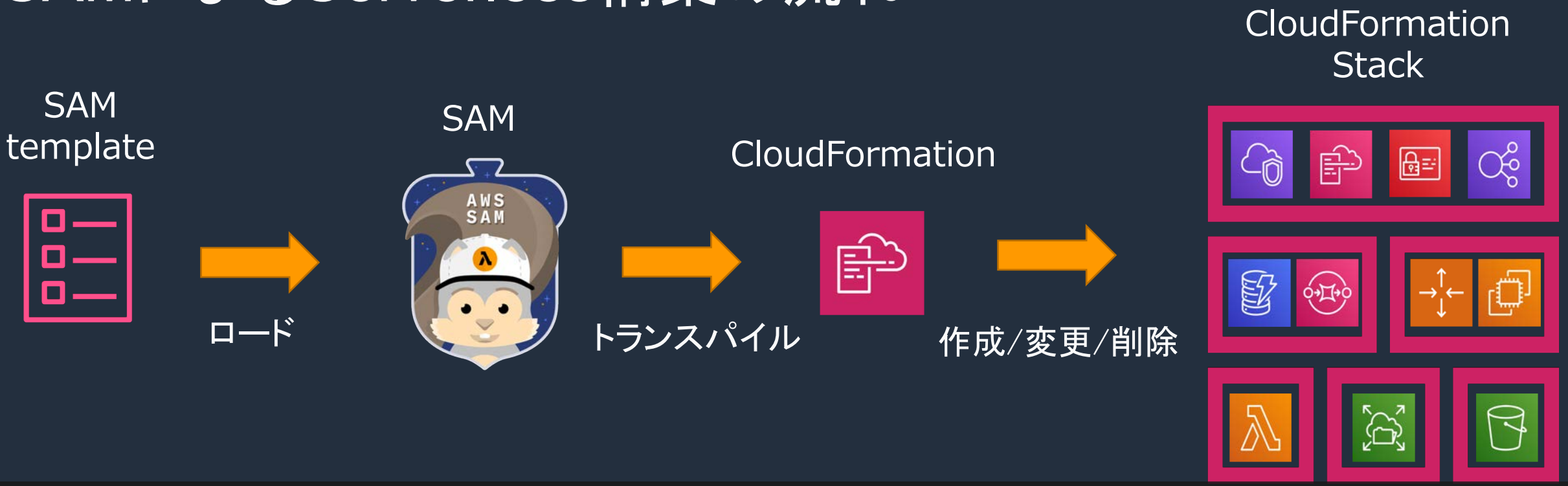
CloudFormation
Stack

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を効率よく操作したい！

SAMを使いやすくしよう！

IDE + AWS Toolkit

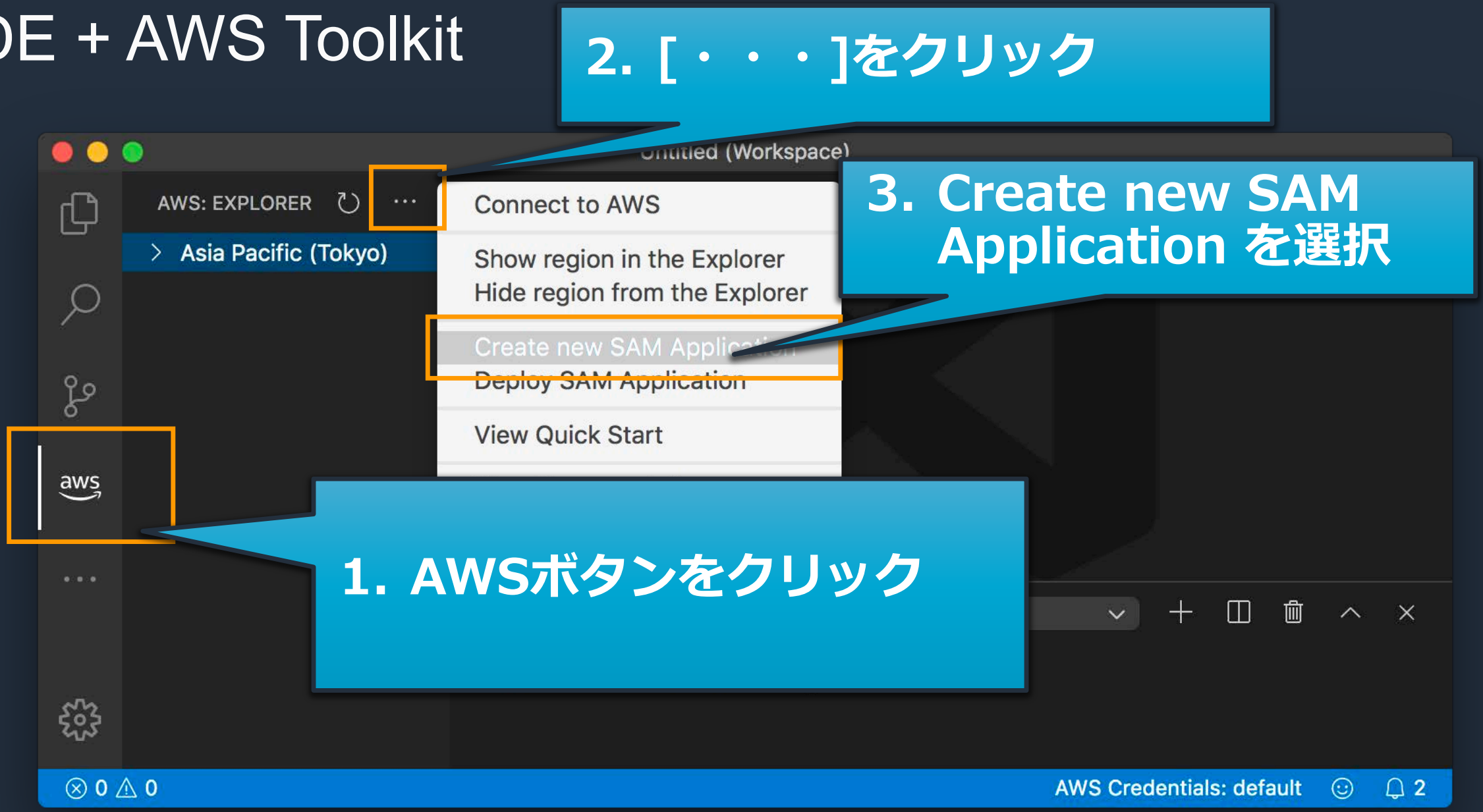
SAM CLI

SAMを使いやすくしよう！

IDE + AWS Toolkit

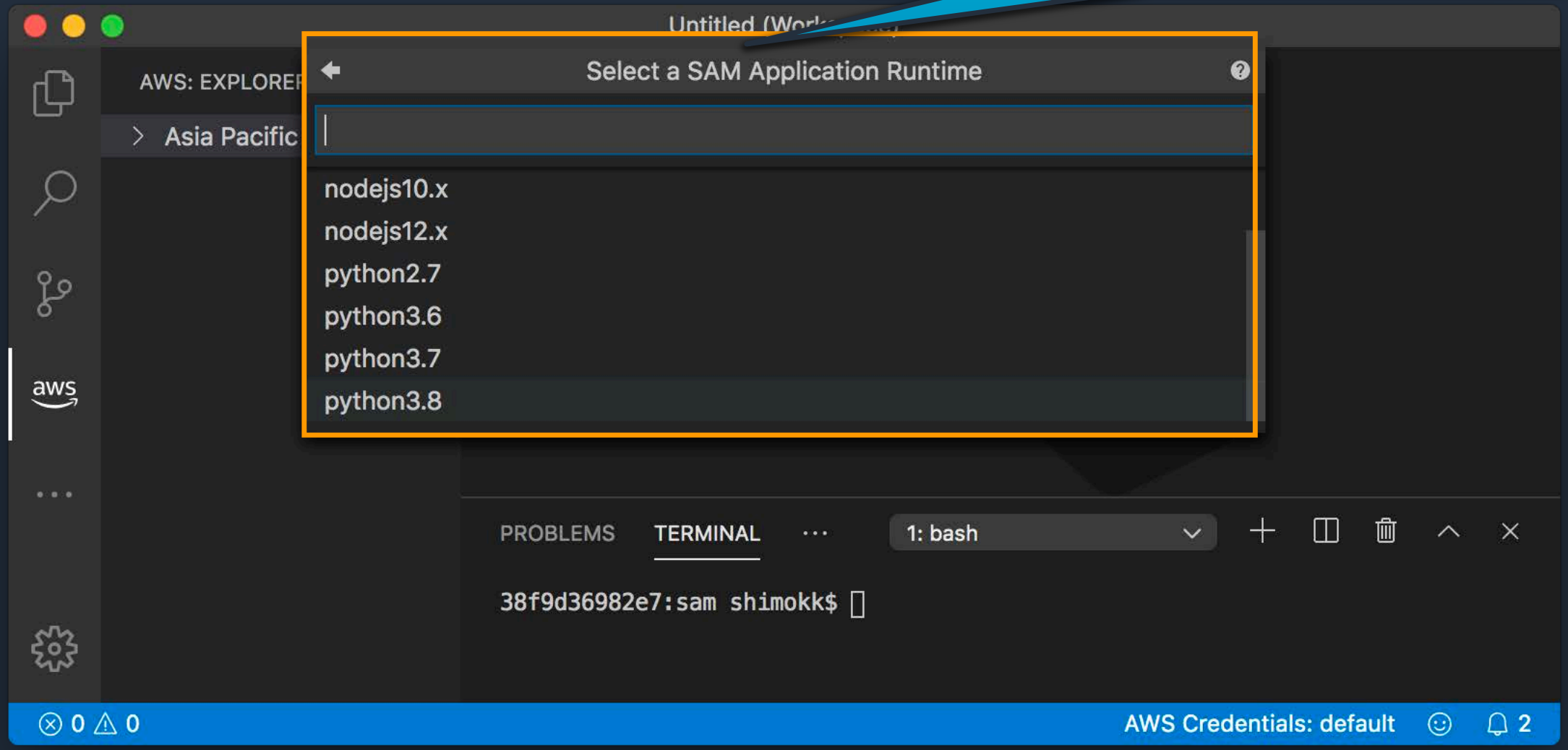
SAM CLI

IDE + AWS Toolkit



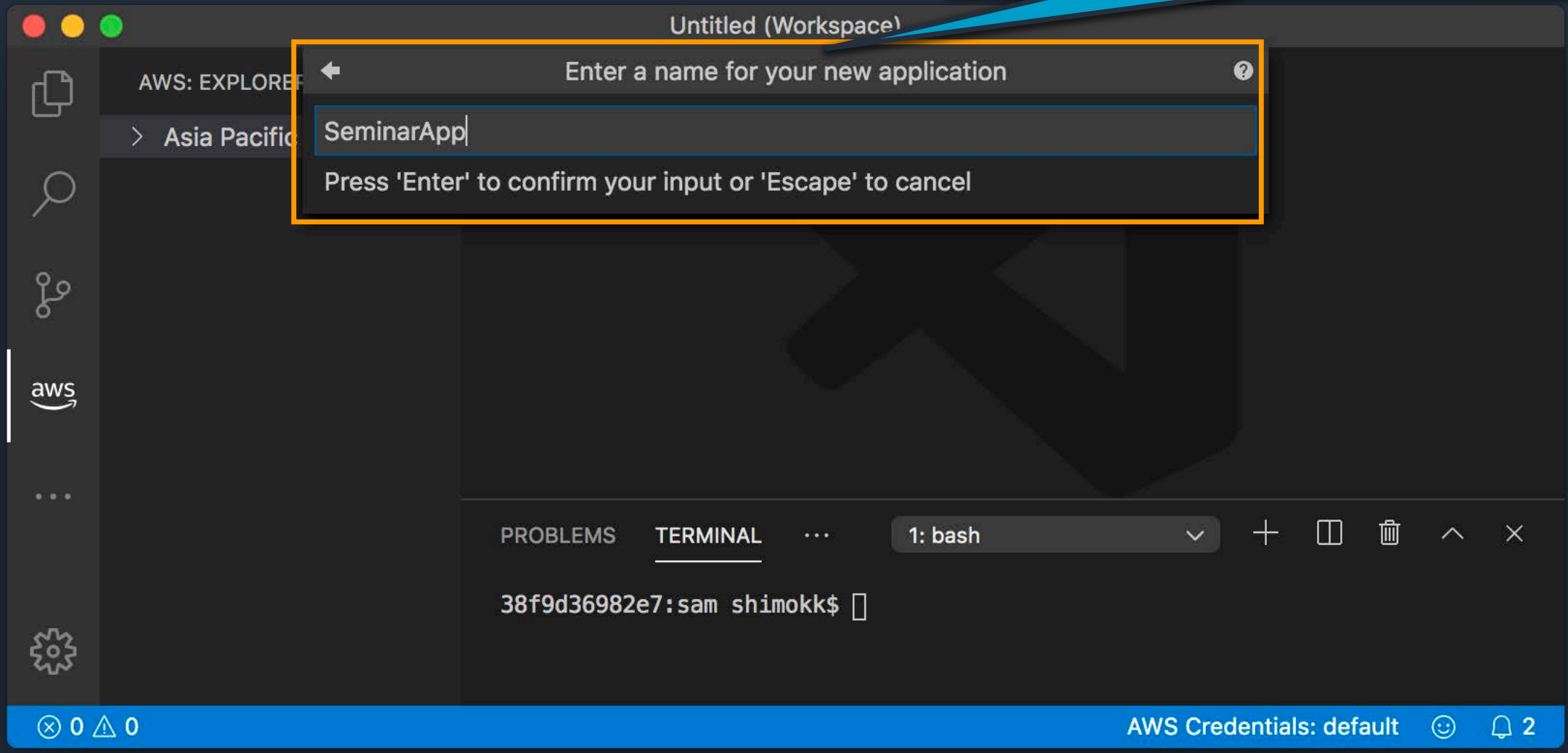
IDE + AWS Toolkit

runtimeを選択



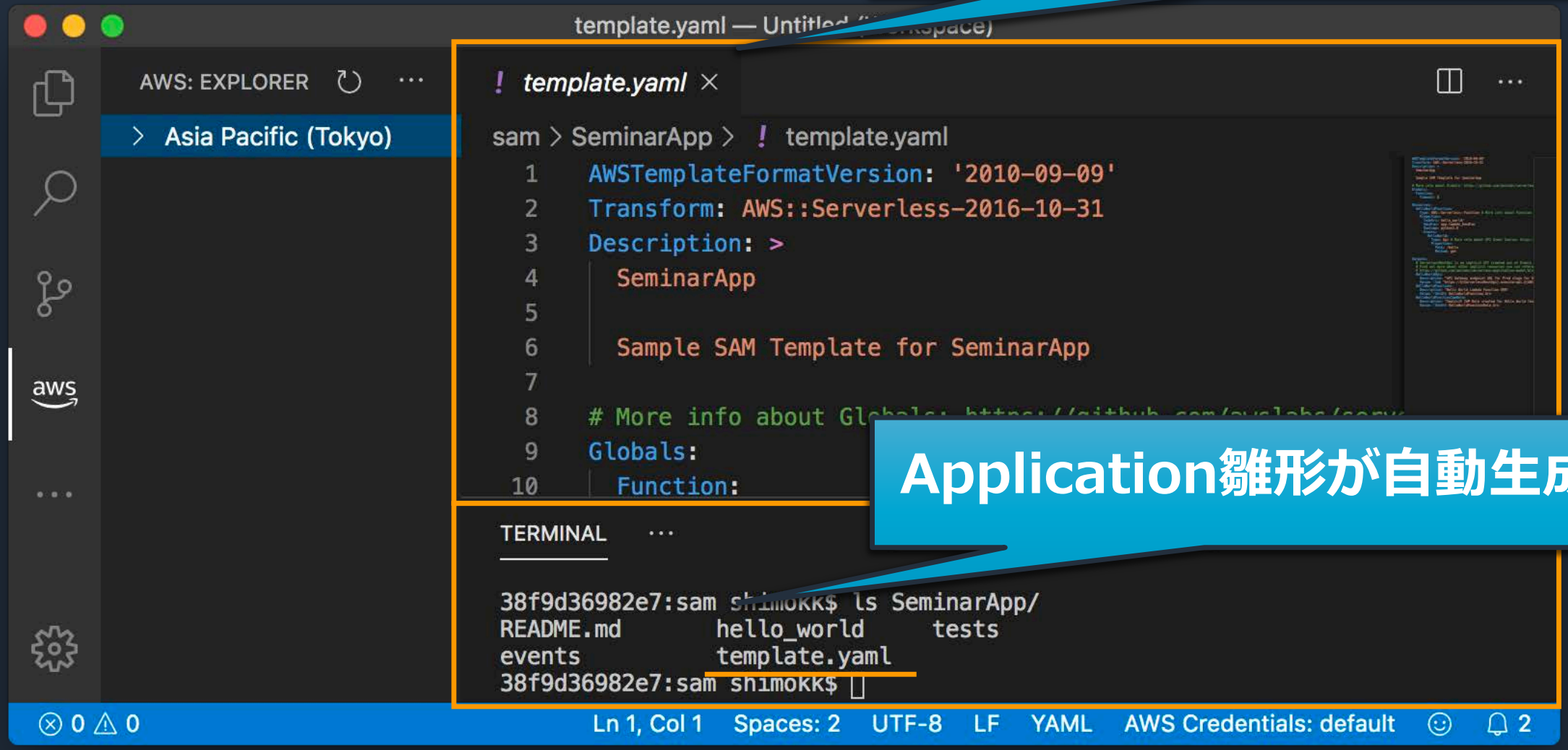
IDE + AWS Toolkit

Applicationの名前を入力



IDE + AWS Toolkit

Template.yamlが自動生成



Application雛形が自動生成

自動生成された SAM の構造

```
SeminarApp
├── README.md
├── events
│   └── event.json
├── hello_world
│   ├── app.py
│   └── requirements.txt
├── template.yaml
└── tests
    └── unit
        └── test_handler.py
```

```python
import json

def lambda_handler(event, context):

    return {
        "statusCode": 200,
        "body": json.dumps({
                "message": "hello world",
            }),
    }
```

シンプルなLambda関数を生成

自動生成された SAM の構造

```
SeminarApp
├── README.md
├── events
│   └── event.json
├── hello_world
│   ├── app.py
│   └── requirements.txt
├── template.yaml
└── tests
    └── unit
        └── test_handler.py
```

```python
import json
import pytest
from hello_world import app

def test_lambda_handler(apigw_event, mocker):

    ret = app.lambda_handler(apigw_event, "")
    data = json.loads(ret["body"])

    assert ret["statusCode"] == 200
    assert "message" in ret["body"]
    assert data["message"] == "hello world”
```

シンプルな assertsを生成

自動生成された SAM の構造

```
SeminarApp
├── README.md
├── events
│   └── event.json
├── hello_world
│   ├── app.py
│   └── requirements.txt
├── template.yaml
└── tests
    └── unit
        └── test_handler.py
```

```
{
    "body": "{¥"message¥": ¥"hello world¥"}",
    "resource": "/{proxy+}",
    "path": "/path/to/resource",
    "httpMethod": "POST",
    "isBase64Encoded": false,
    "queryStringParameters": {
        "foo": "bar"
    },
    "pathParameters": {
        "proxy": "/path/to/resource"
    },
    "stageVariables": {
    "baz": "qux"
    },
    "header                    }
}
```

API GatewayからのEventを生成

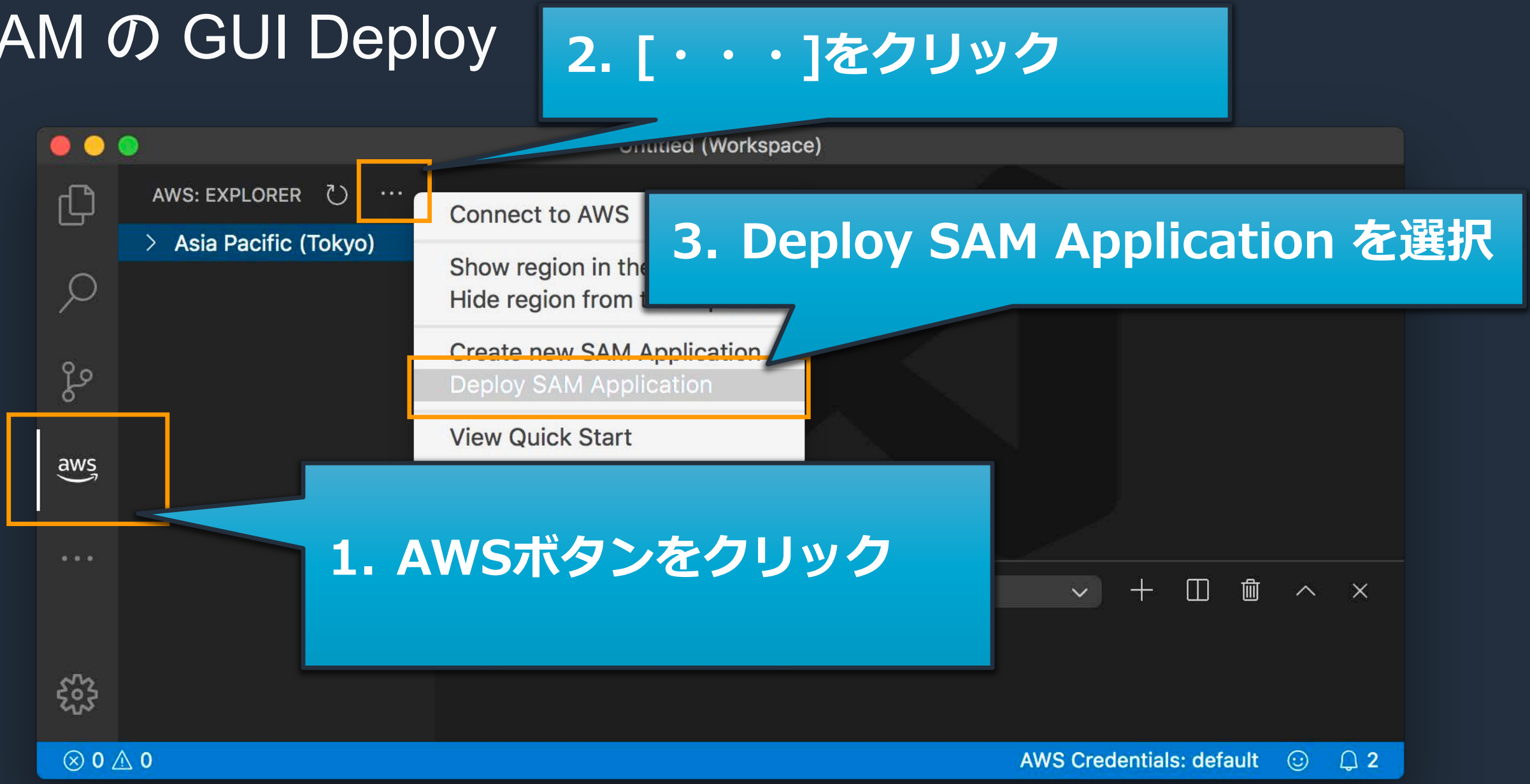
自動生成された SAM の構造

```
SeminarApp
├── README.md
├── events
│   └── event.json
├── hello_world
│   ├── app.py
│   └── requirements.txt
├── template.yaml
└── tests
    └── unit
        └── te
```

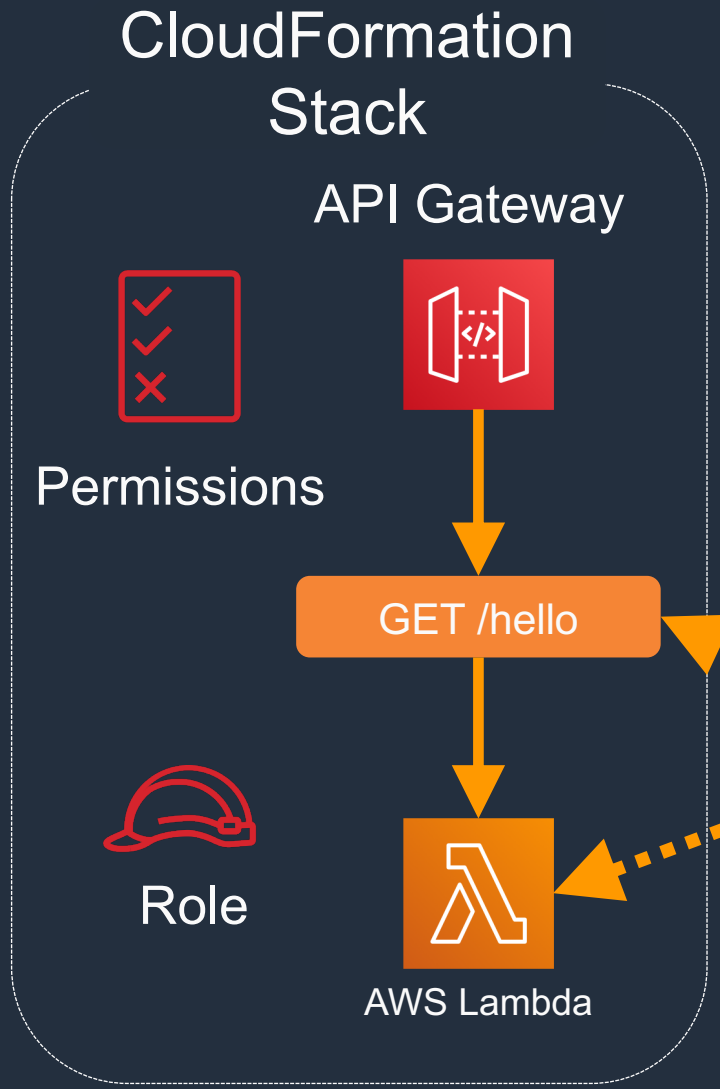
```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: >
    SeminarApp

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
```

SAM Templateを生成

SAM の GUI Deploy



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
    Semin

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

Close

Resource types

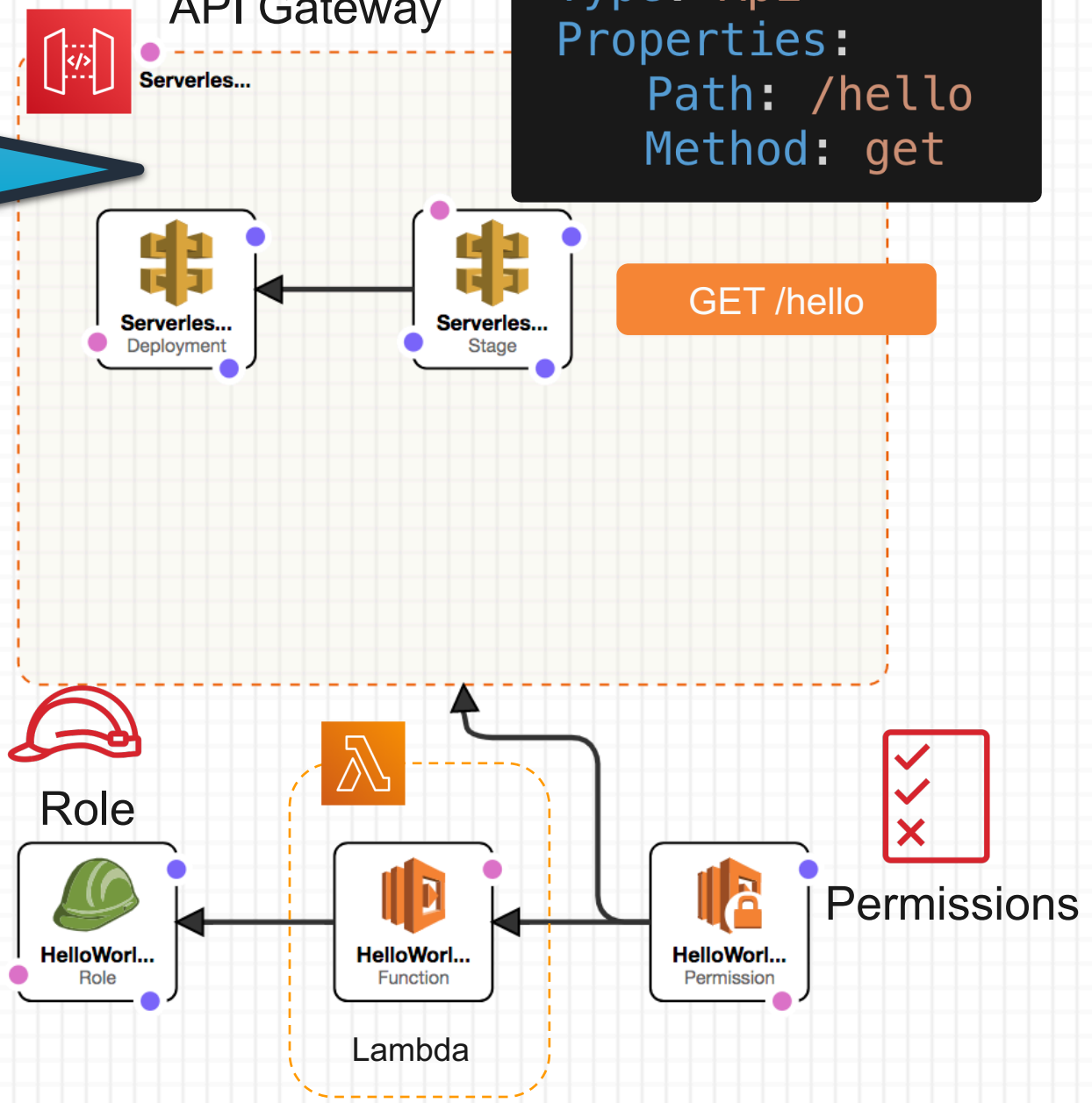
- ACMPCA
- ApiGatewayV2
- AppMesh
- AppStream
- AppSync
- ApplicationAutoScaling
- Athena
- AutoScaling
- CloudFormation

DeployしたStackは Management Console上のDesignerで確認できる。

```
Type: AWS::Serverless::Function
Properties:
    CodeUri: hello_world/
    Handler: app.lambda_handler
    Runtime: python3.8
```

File: 'template1'

```
Type: Api
Properties:
    Path: /hello
    Method: get
```



SAM の GUI Deploy

手作業はミスが増える？

SAM の GUI Deploy

SAMを使いやすくしよう！

IDE + AWS Toolkit

SAM CLI

AWS SAM CLI

1）アプリケーションのひな形を生成

```
$ sam init --runtime python3.8
```

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 CLI

※sam deploy の際に必要な S3 バケットは自動生成

AWS SAM CLI

1）アプリケーションのひな形を生成

```
$ sam init --runtime python3.8
```

2）ローカル環境でビ

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

3）ビルド後にAWS へデプロイ

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

- ***sam init* コマンドでプロジェクトの雛形を作成**
- ***--runtime* オプションで言語を指定**

SAM CLI

※sam deploy の際に必要な S3 バケットは自動生成

AWS SAM CLI

1） アプリケーションのひな形を生成

```
$ sam init --runtime python3.8
```

2） ローカル環境でビルド

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

3） ビルド後にAWS

```
$ sam deploy
```

- ***--use-container* オプションで *Docker Container*内でビルドすることも可能**

SAM CLI

※sam deploy の際に必要な S3 バケットは自動生成

AWS SAM CLI

1）アプリケーションのひな形を生成

```
$ sam init --runtime python3.8
```

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
```

- ***--guided* オプションで*change set* を確認後にデプロイ可能**

※sam deploy の際に必要な S3 バケットは自動生

--guided オプション

デプロイ先のリージョンや、*AWS CloudFormation* のスタック名を *CLI* のプロンプトから指定可能

```
sam-deploy-test ) sam deploy --guided

Configuring SAM deploy
======================

        Looking for samconfig.toml :  Not found

        Setting default arguments for 'sam deploy'
        =========================================
        Stack Name [sam-app]: sam-deploy-test
        AWS Region [us-east-1]: ap-northeast-1
        #Shows you resources changes to be deployed and require a 'Y' to initiate deploy
        Confirm changes before deploy [y/N]: y
        #SAM needs permission to be able to create roles to connect to the resources in your template
        Allow SAM CLI IAM role creation [Y/n]: y
        Save arguments to samconfig.toml [Y/n]: y

        Looking for resources needed for deployment: Not found.
        Creating the required resources...
        Successfully created!

                Managed S3 bucket: aws-sam-cli-managed-default-samclisourcebucket-sykucxjohevd
                A different default S3 bucket can be set in samconfig.toml
```

--guided オプション

```
sam-deploy-test ) sam deploy --guided

Configuring SAM deploy
======================

        Looking for samconfig.toml :  Not found

        Setting default arguments for 'sam
        =================================
        Stack Name [sam-app]: sam-deploy-t
        AWS Region [us-east-1]: ap-northea
        #Shows you resources changes to be
        Confirm changes before deploy [y/N
        #SAM needs permission to be able t                                    template
        Allow SAM CLI IAM role creation [Y/n]: y
        Save arguments to samconfig.toml [Y/n]: y

        Looking for resources needed for deployme        t found.
        Creating the required resources...
        Successfully created!

            Managed S3 bucket: aws-sam-cli-managed-default-samclisourcebucket-sykucxjohevd
            A different default S3 bucket can be set in samconfig.toml
```

自動生成された、*package* 格納用の *S3 Bucket* 情報

--guided オプション

Operation	LogicalResourceId	ResourceType
+ Add	ServerlessRestApiDeployment8cf30ed3cd	AWS::ApiGateway::Deployment
* Modify	HelloWorldFunctionHelloWorldPermissionProd	AWS::Lambda::Permission
* Modify	ServerlessRestApiProdStage	AWS::ApiGateway::Stage
* Modify	ServerlessRestApi	AWS::ApiGateway::RestApi
- Delete	ServerlessRestApiDeployment47fc2d5f9d	AWS::ApiGateway::Deployment

前回デプロイからの変更差分を表示

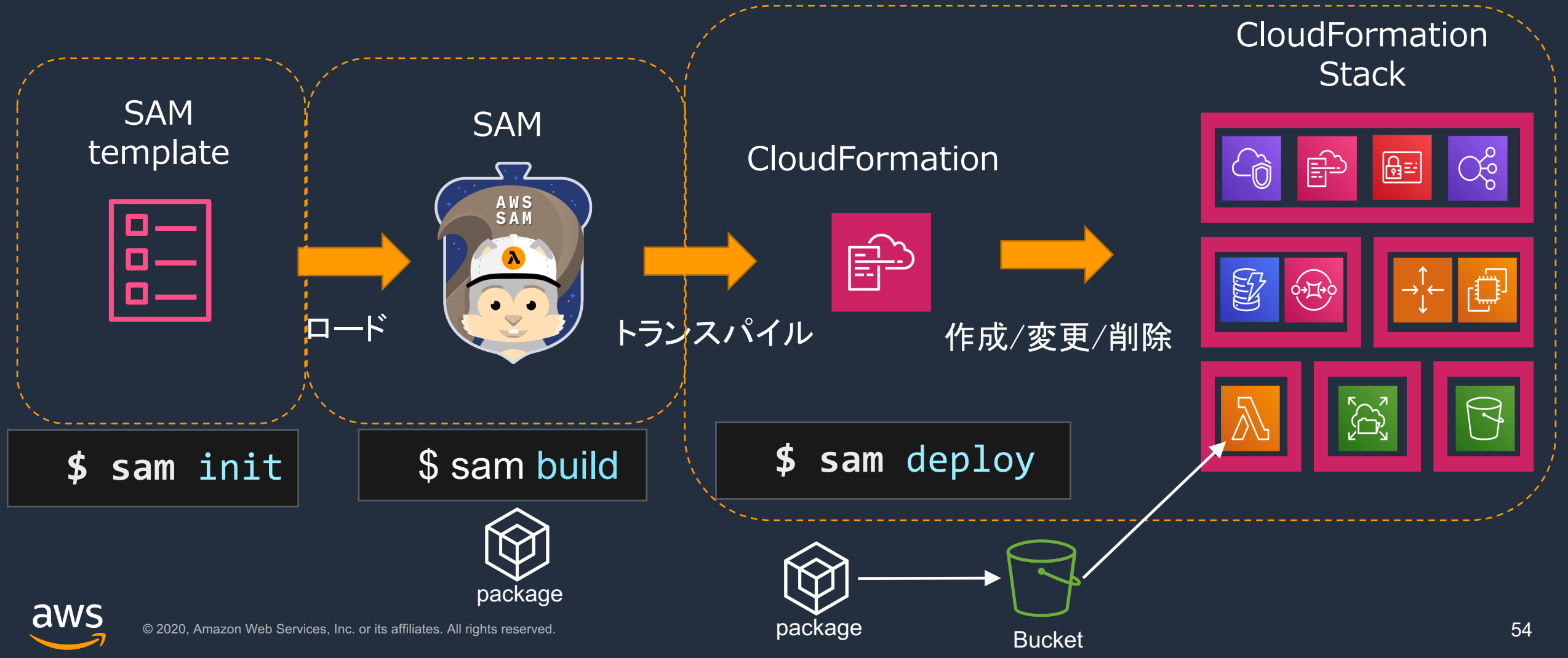
→ この段階でデプロイをキャンセルできる

--guided オプション

CloudFormation 実行後のイベント発生履歴

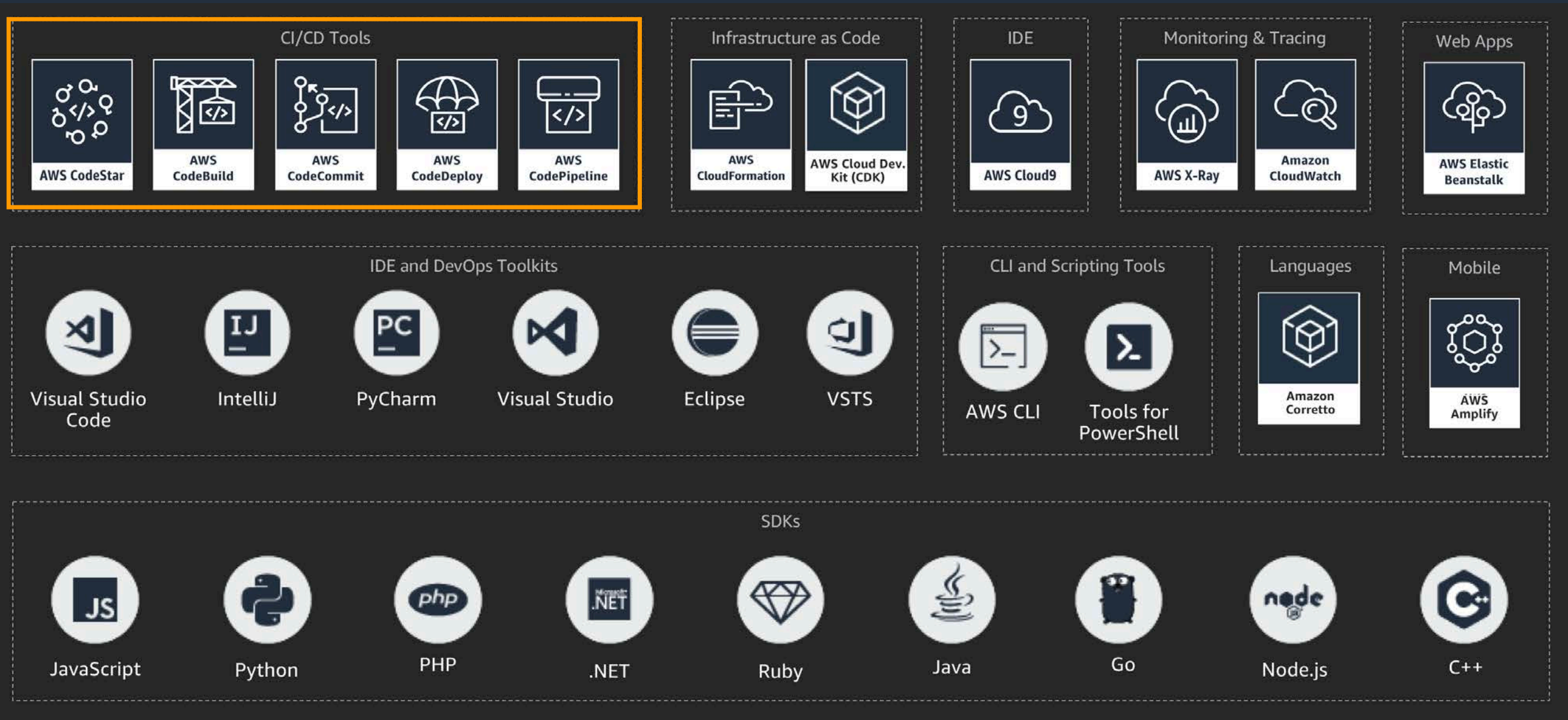
ResourceStatus	ResourceType	LogicalResourceId	ResourceStatusReason
UPDATE_IN_PROGRESS	AWS::ApiGateway::RestApi	ServerlessRestApi	-
UPDATE_COMPLETE	AWS::ApiGateway::RestApi	ServerlessRestApi	-
CREATE_IN_PROGRESS	AWS::ApiGateway::Deployment	ServerlessRestApiDeployment8cf30ed3cd	-
CREATE_IN_PROGRESS	AWS::ApiGateway::Deployment	ServerlessRestApiDeployment8cf30ed3cd	Resource creation Initiated
CREATE_COMPLETE	AWS::ApiGateway::Deployment	ServerlessRestApiDeployment8cf30ed3cd	-
UPDATE_IN_PROGRESS	AWS::Lambda::Permission	HelloWorldFunctionHelloWorldPermissionProd	Requested update requires the creation of a new physical resource; hence creating one.
UPDATE_IN_PROGRESS	AWS::ApiGateway::Stage	ServerlessRestApiProdStage	-
UPDATE_COMPLETE	AWS::ApiGateway::Stage	ServerlessRestApiProdStage	-
UPDATE_IN_PROGRESS	AWS::Lambda::Permission	HelloWorldFunctionHelloWorldPermissionProd	Resource creation Initiated
UPDATE_COMPLETE	AWS::Lambda::Permission	HelloWorldFunctionHelloWorldPermissionProd	-
UPDATE_COMPLETE_CLEANUP_I	AWS::CloudFormation::Stac	sam-deploy-test	-

SAMによるServerless構築の流れ

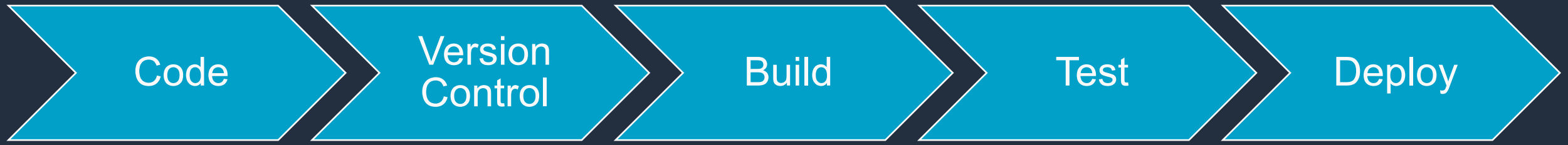


ServerlessのCI/CDパイプラインを整える

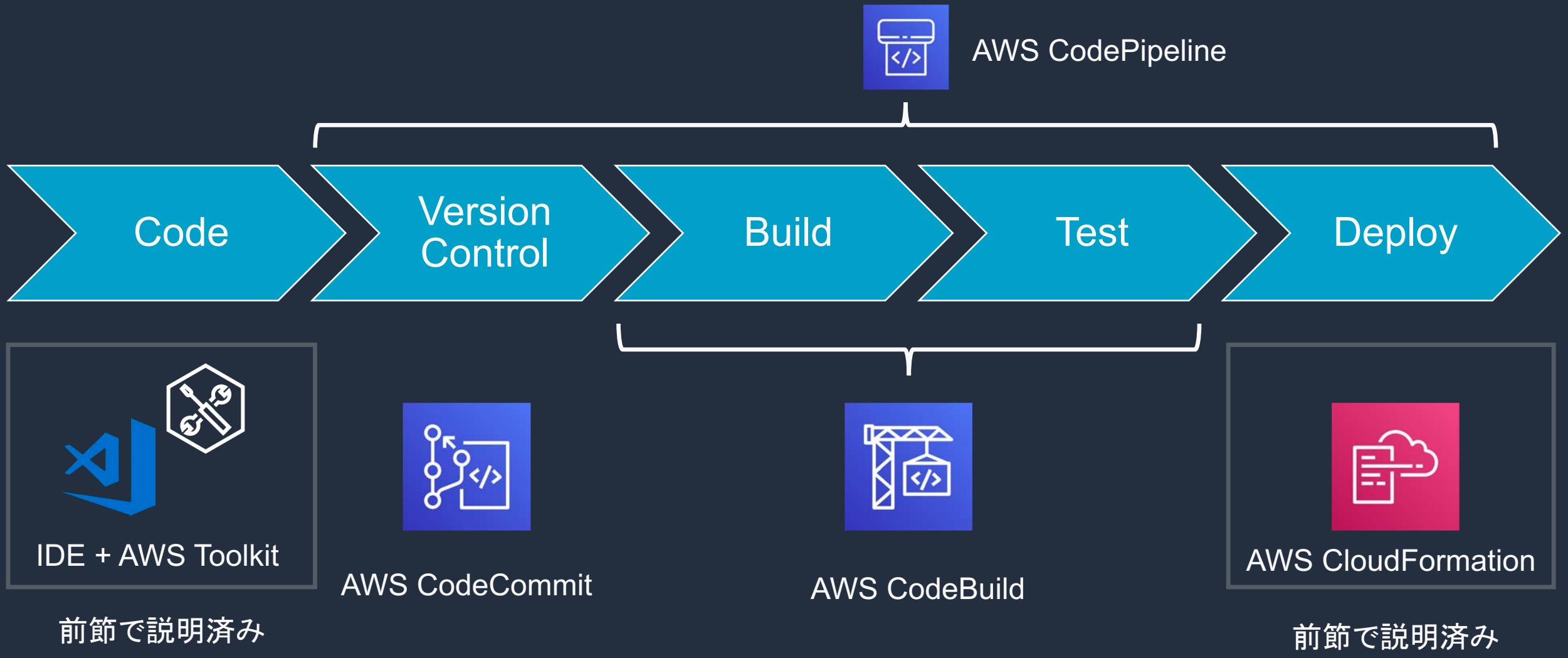
AWS提供の総合的なツールセット



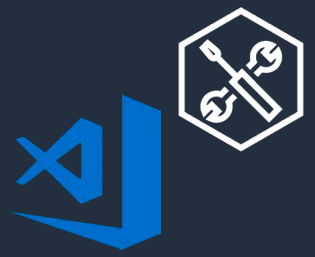
CI/CD for Serverless のワークフロー (復習）



CI/CD for Serverless のワークフロー に利用できるツール



CI/CD for Serverless のよくあるパイプライン化



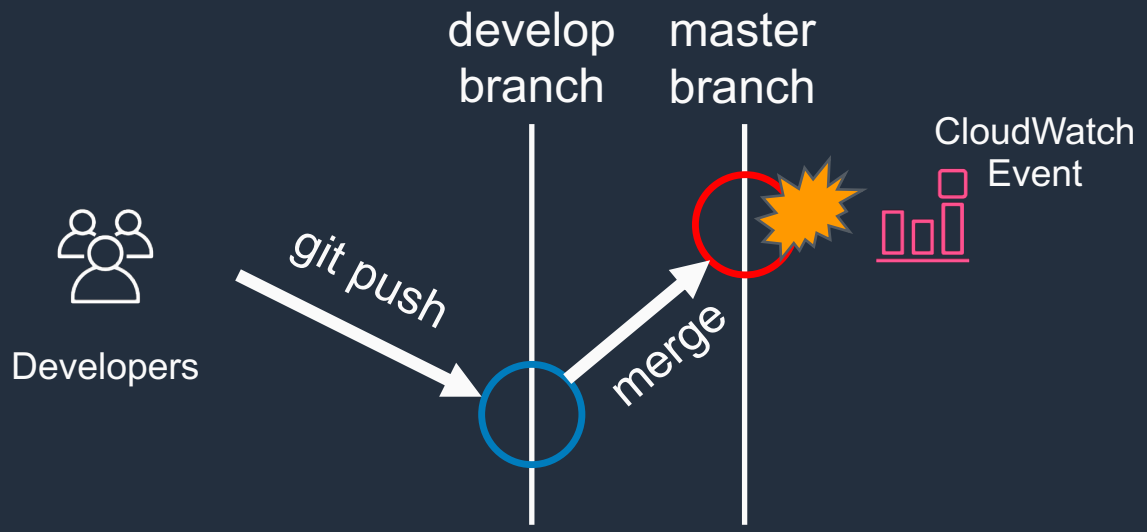
IDE + AWS Toolkit

AWS CodeCommit

AWS CodeBuild

AWS CloudFormation

CI/CD for Serverless のよくあるパイプライン化



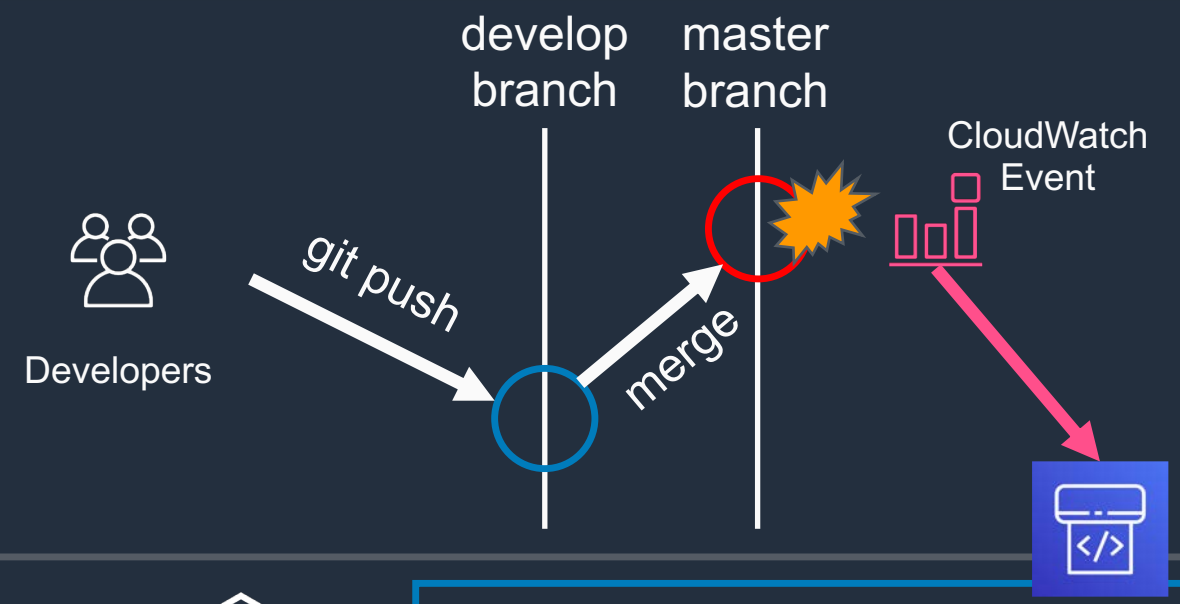
IDE + AWS Toolkit

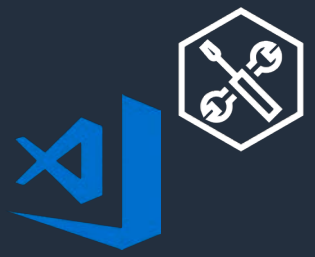
AWS CodeCommit

AWS CodeBuild

AWS CloudFormation

CI/CD for Serverless のよくあるパイプライン化

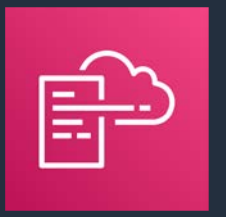




IDE + AWS Toolkit

AWS CodeCommit

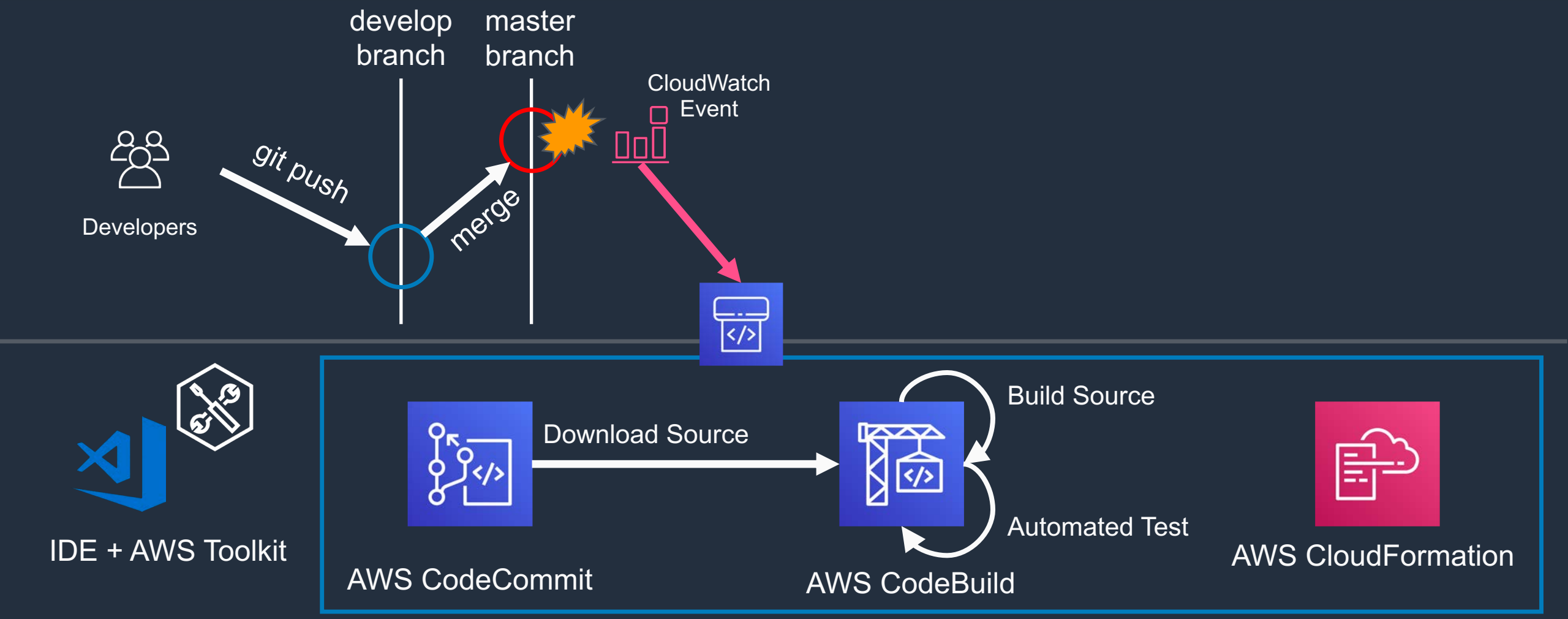
AWS CodeBuild



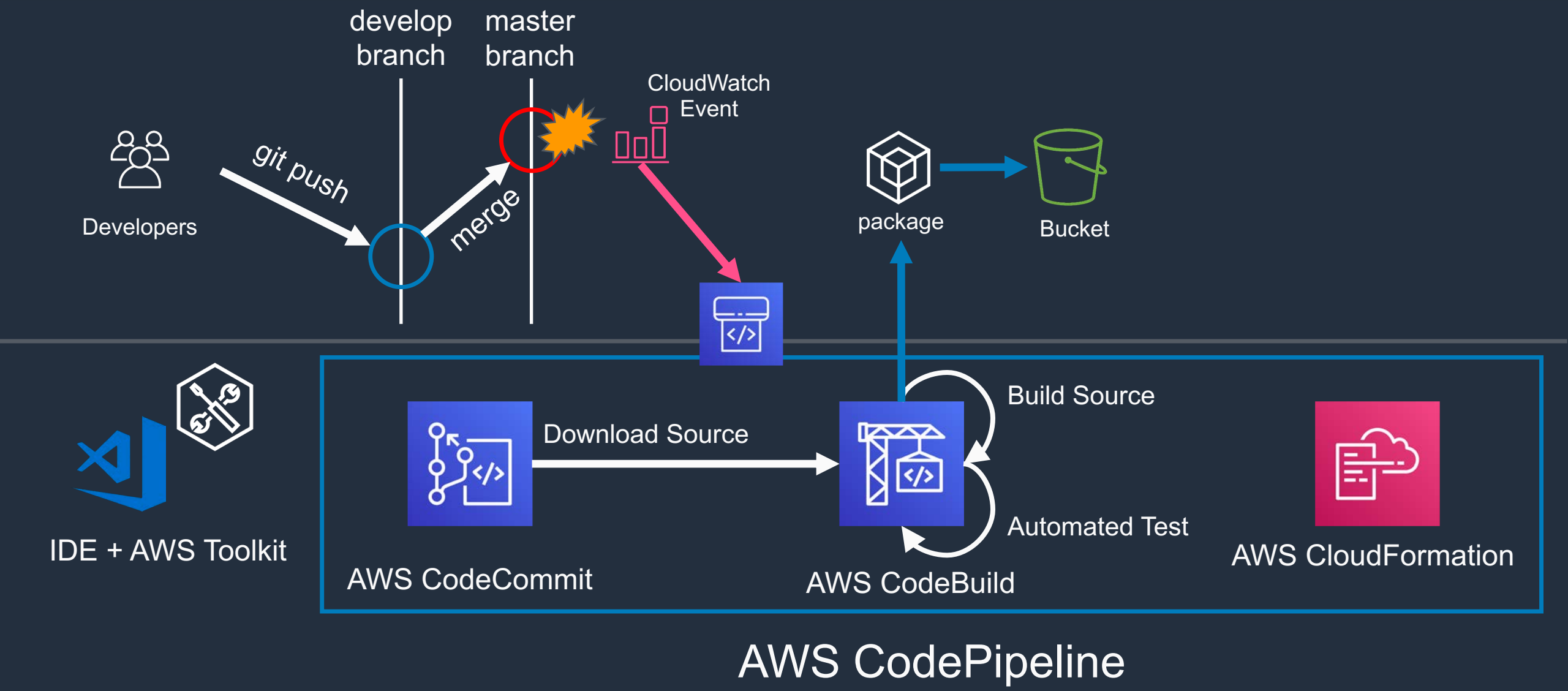
AWS CloudFormation

AWS CodePipeline

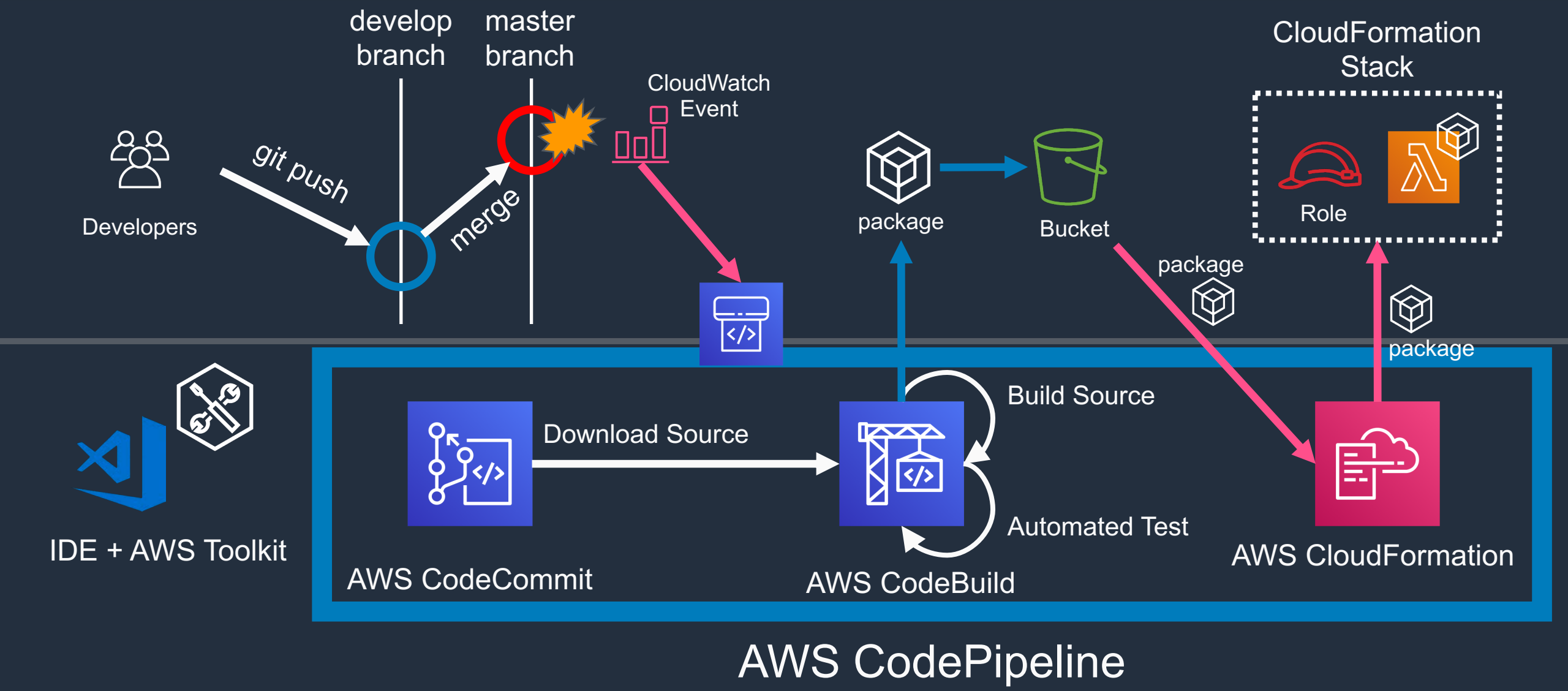
CI/CD for Serverless のよくあるパイプライン化



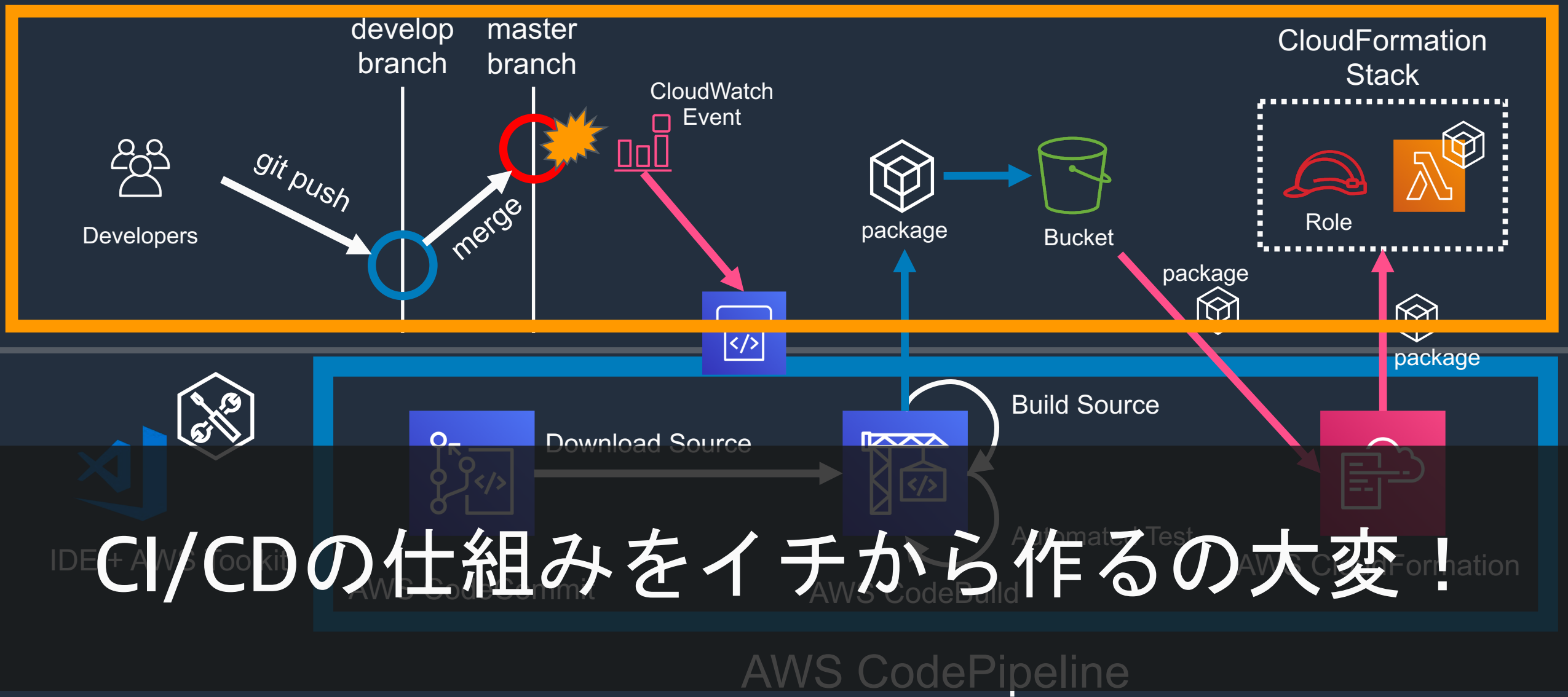
CI/CD for Serverless のよくあるパイプライン化



CI/CD for Serverless のよくあるパイプライン化

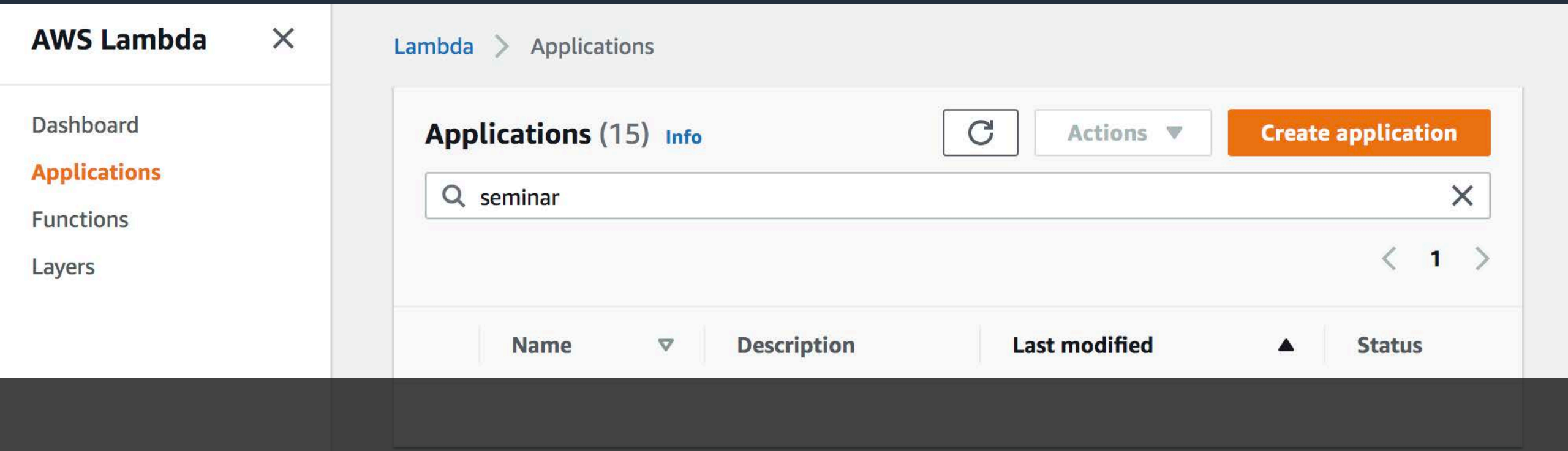


CI/CD for Serverless のよくあるパイプライン化



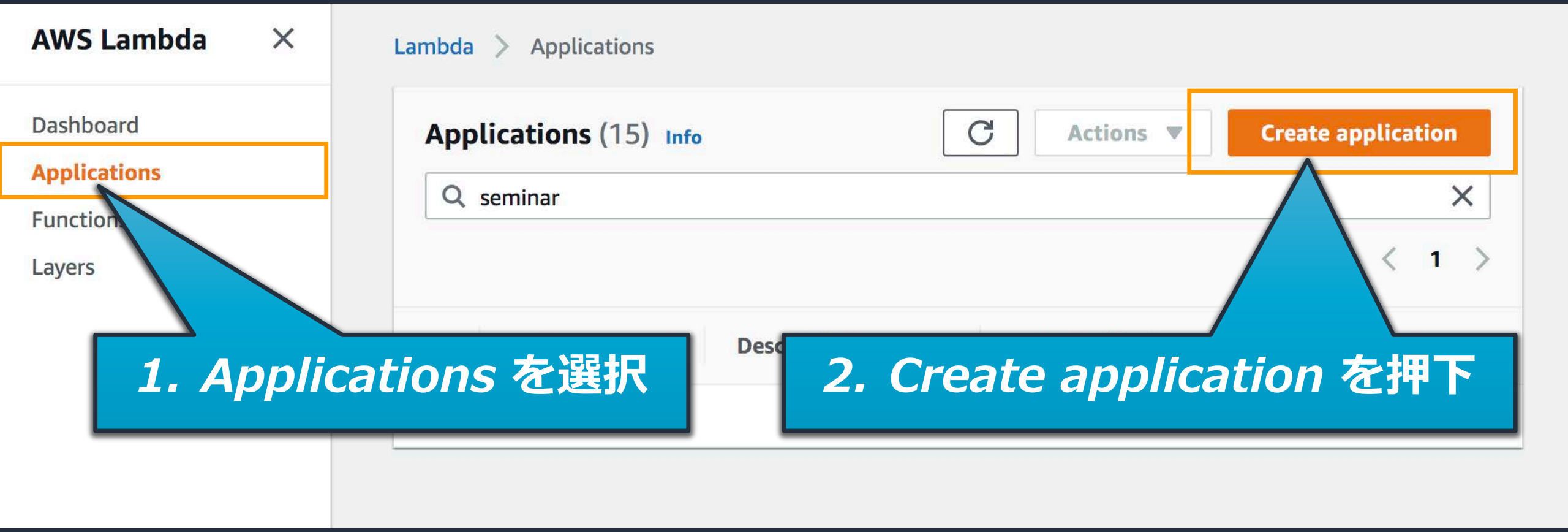
CI/CDの仕組みをイチから作るの大変！

Serverless Applications Pipeline

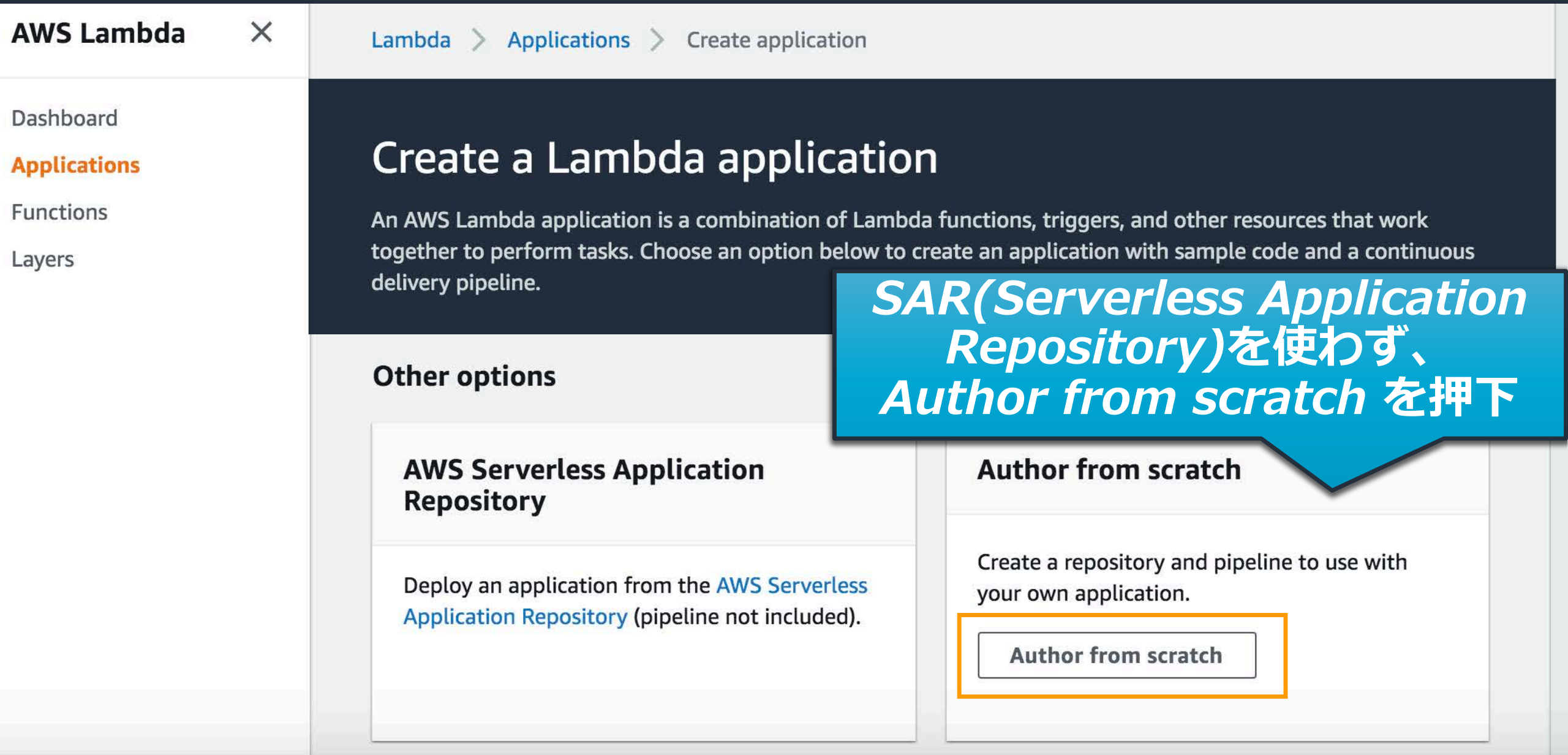


Serverless Applications の機能でCI/CD！

Serverless Applications Pipeline



Serverless Applications Pipeline



Serverless Applications Pipeline

Dashboard

Applications

Functions

Layers

Configure your application

Application details

Application name

seminar-app

Use only lowercase letters, numbers, or hyphens. The maximum length is 20 characters.

Application description

How to create Serverless CICD

The maximum length is 1000 characters.

Function configuration

Runtime

Node.js 10.x

Node.js 10.x

現在は、*Node.js 10.x*のみが選択可能

Serverless Applications Pipeline

Dashboard

Applications

Functions

Layers

Source control

Source control service

Choose where to create your application's Git repository.

CodeCommit

Create a repository in your AWS account. Manage SSH keys and HTTP credentials for users in the IAM console.

GitHub

Create a private repository in yo

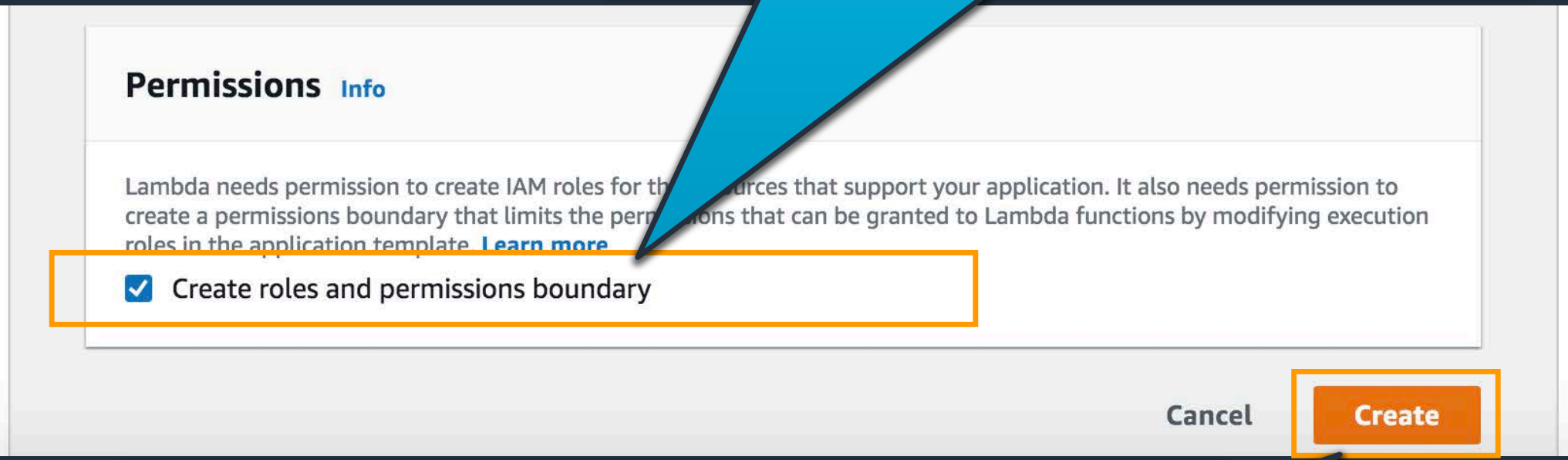
Repository name

seminar-app

The maximum length is 100 characters.

***Default* では、*Application*名が自動的に入る**

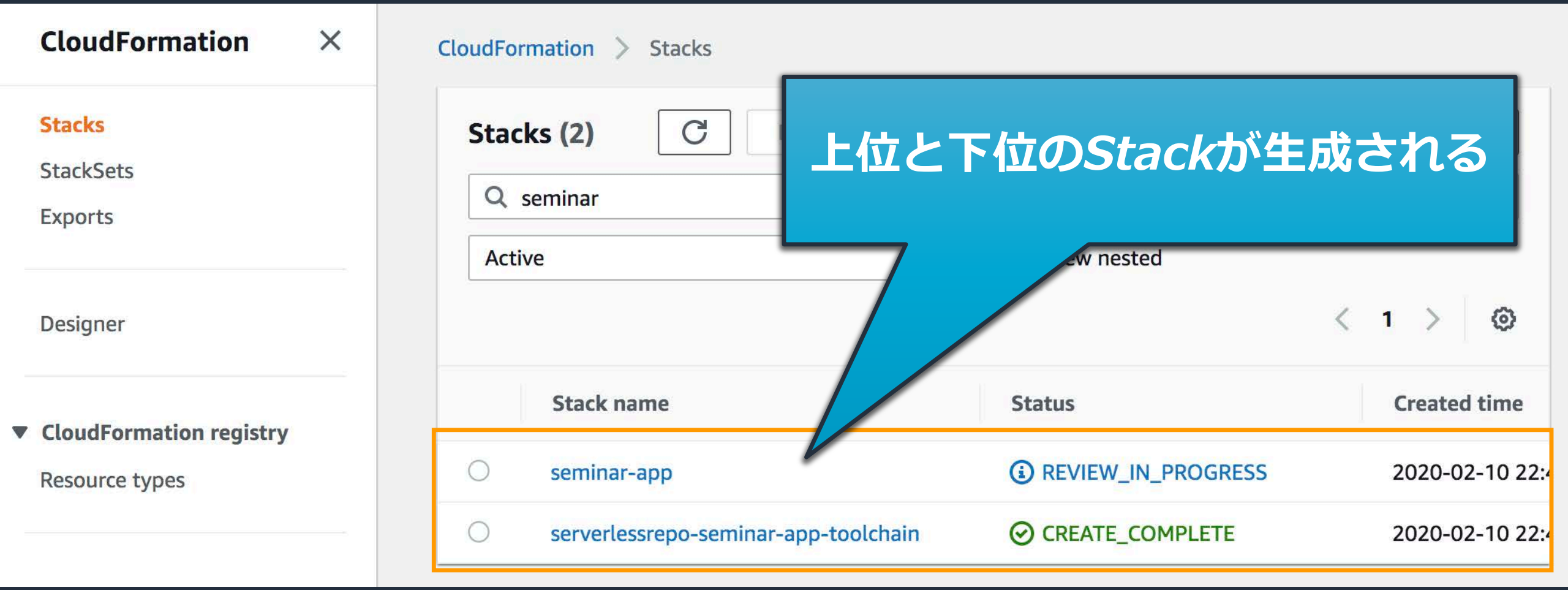
Serverless Applications Pipeline



Permission Boundaryの作成
チェックボックスが必須

Create 押下するとCloudFormation
による生成が始まる

CloudFormation Stack の確認



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

下位 CloudFormation Stackでパイプラインリソース生成

master branch

master branch を作成

CloudWatch Event

master branch を監視

Bucket

Package格納用 bucketを作成

buildプロジェクト作成

Git リポジトリ 作成

AWS CodeCommit

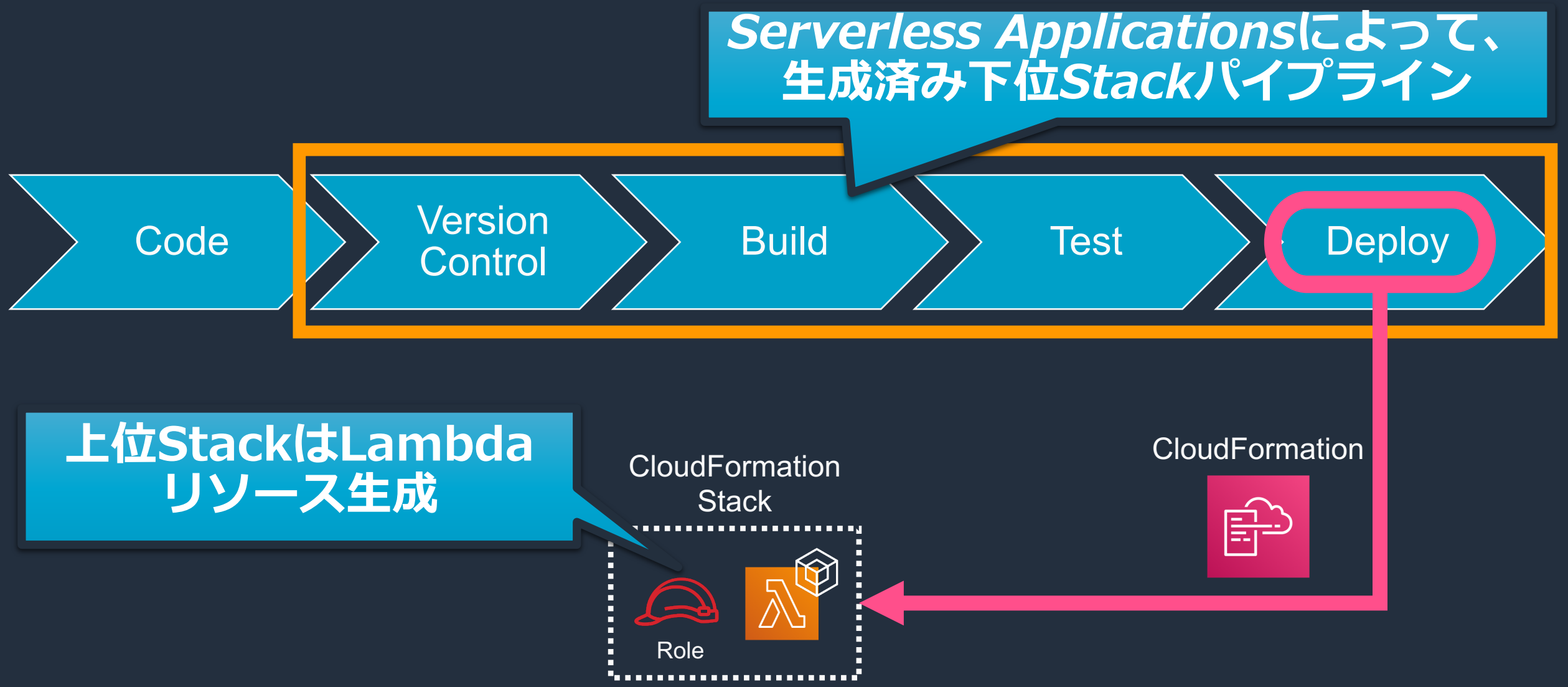
AWS CodeBuild

AWS CloudFormation

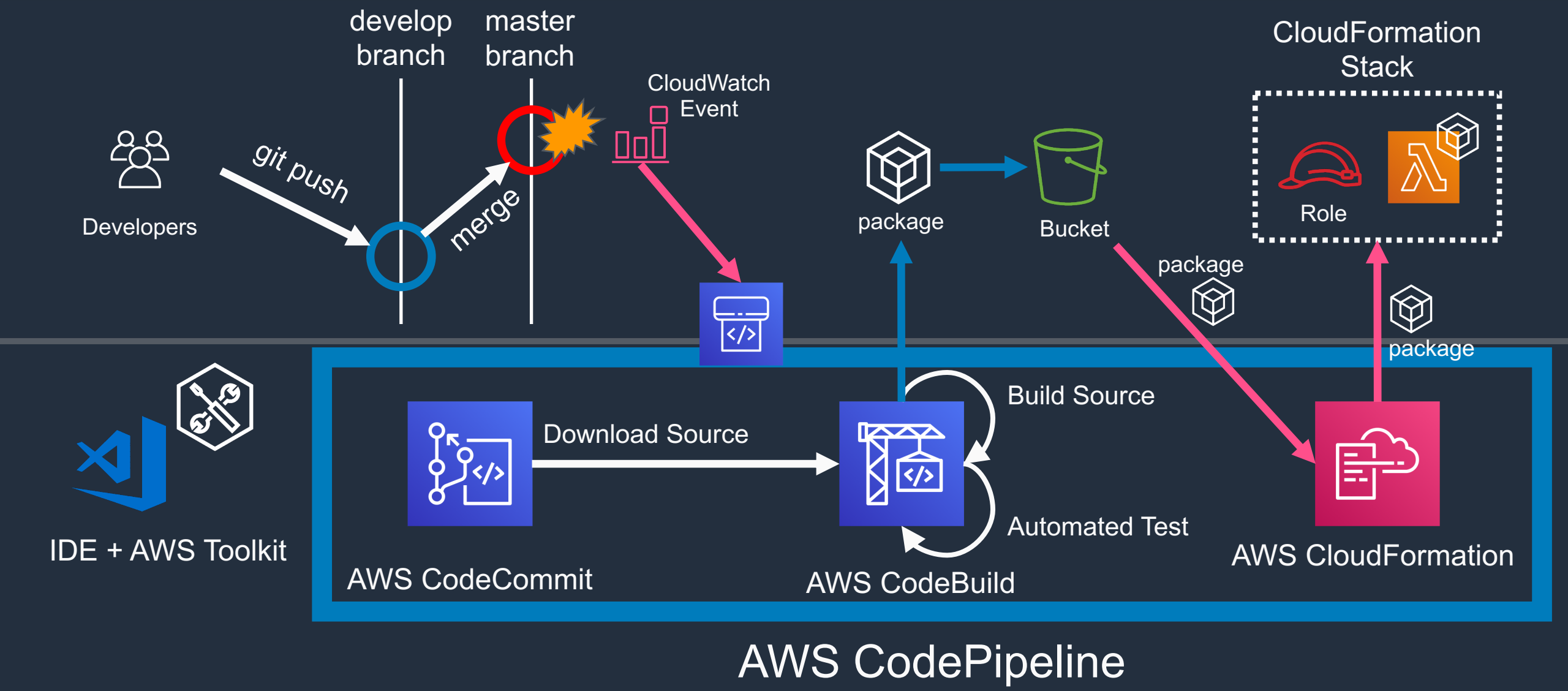
AWS CodePipeline

パイプライン 作成

上位 CloudFormation Stackでのリソース生成



Lambda関数修正後のmaster mergeでパイプライン起動！



Serverless Applications Pipeline の補足

Serverless Applications Pipelineは Node.jsだけ？

Applications

Functions

Layers

Application details

Application name

seminar-app

Use only lowercase letters, numbers, or hyphens. The maximum length is 20 characters.

Application description

How to create Serverless CICD

The maximum length is 1000 characters.

Function configuration

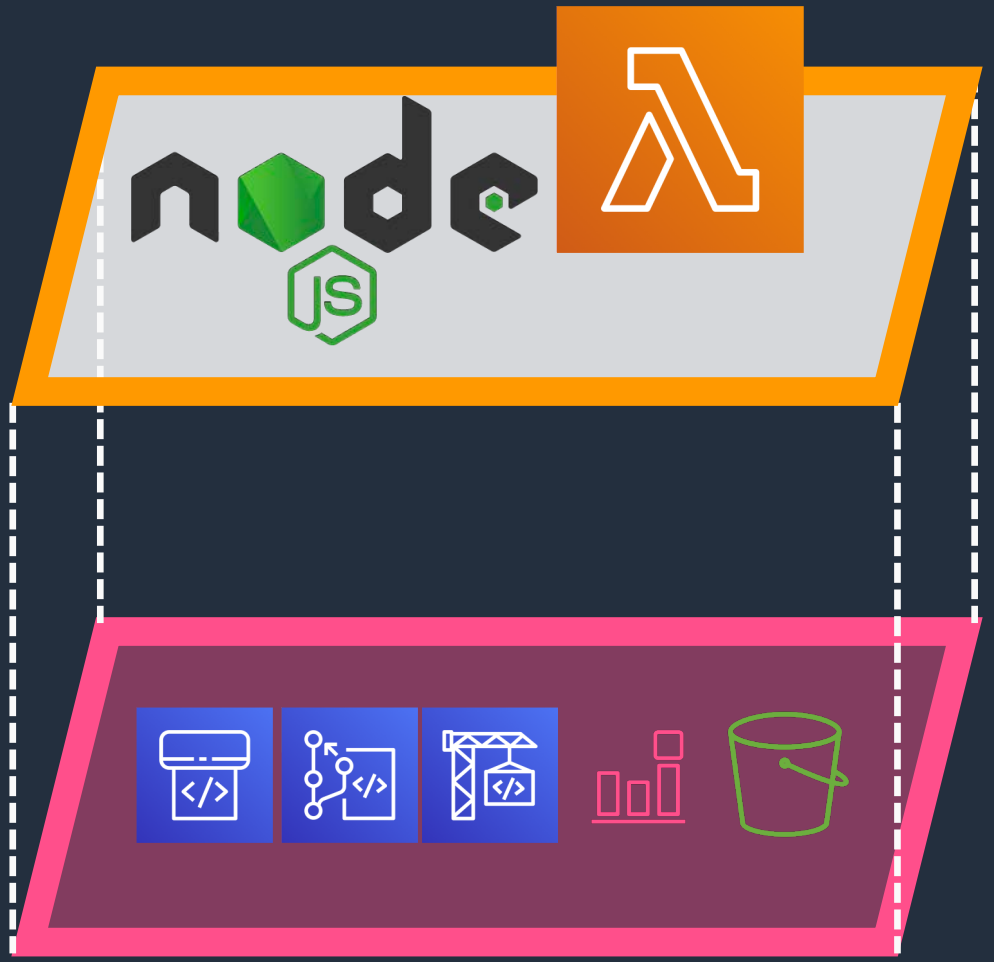
Runtime

Node.js 10.x

Node.js 10.x

現在は、*Node.js 10.x*のみが選択可能

CloudFormation Stack の上位Stackについて



上位Stack

seminar-app
Node.js アプリケーションの雛形Stack

下位Stack

serverlessrepo-seminar-app-toolchain
アプリに依存しないパイプラインStack

言語置き換え

Serverless Applicationsによって生成

```
SeminarApp
├── README.md
├── __tests__
│   └── unit
│       └── handlers
│           └── hello-from-lambda.test.js
├── buildspec.yml
├── package.json
├── src
│   └── handlers
│       └── hello-from-lambda.js
└── template.yml
```

```
SeminarApp
├── README.md
├── events
│   └── event.json
├── hello_world
│   ├── app.py
│   └── requirements.txt
├── template.yml
└── tests
    └── unit
        └── test_handler.py
```

Python

IDE + AWS Toolkit によって生成

言語置き換え

***Serverless Applications*によって生成**

```
SeminarApp
├── README.md
├── __tests__
│   └── unit
│       └── handlers
│           └── hello-from-lambda.te
├── buildspec.yml
├── package.json
├── src
│   └── handlers
│       └── hello-from-lambda.js
└── template.yml
```

```
SeminarApp
├── README.md
├── events
├── template.yml
└── tests
    └── unit
        └── test_handler.py
```

（置き換え時の注意点）
***buildspec.yml* だけは、保持したい**

AWS CodeBui

***IDE + AWS Toolkit* によって生成**

言語置き換え

Serverless Applicationsによって生成

```
SeminarApp
├── README.md
├── __tests__
│   └── unit
│       └── handlers
│           └── hello-from-lambda.test.js
├── buildspec.yml
├── package.json
├── src
│   └── handlers
│       └── hello-from-lambda.js
└── template.yml
```

ローカル端末上で置き換え

```
SeminarApp
├── README.md
├── events
│   └── event.json
├── hello_world
│   ├── app.py
│   └── requirements.txt
├── template.yml
└── tests
    └── unit
        └── test_handler.py
```

Python

IDE + AWS Toolkit によって生成

言語置き換え

Serverless Applicationsによって生成

SeminarApp
- README.md
- **buildspec.yml**
- events
  - event.json
- hello_world
  - app.py
  - requirements.txt
- template.yml
- tests
  - unit
    - test_handler.py

Python

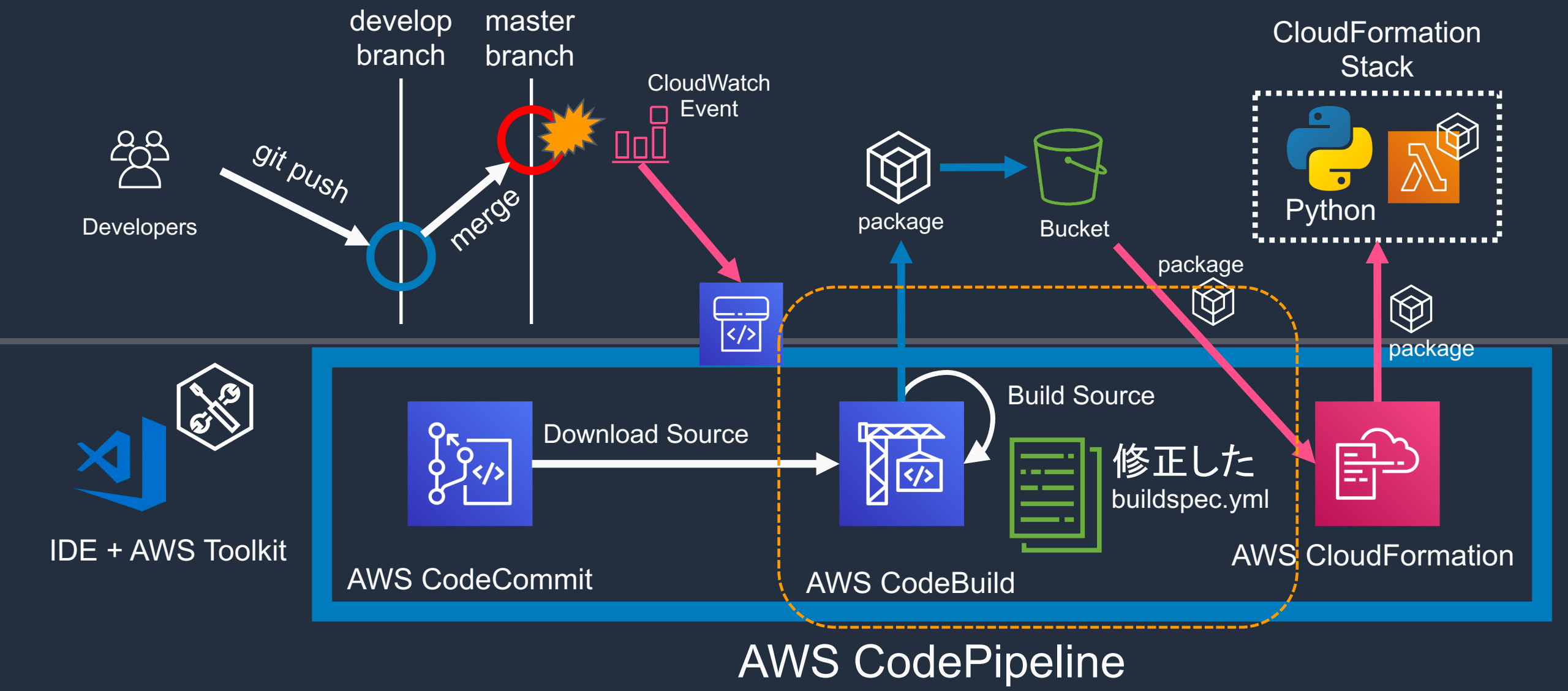
buildspec.yml のPython化編集 (抜粋)

```
version: 0.2
phases:
  install:
    commands:
      - npm install
  build:
    commands:
      - aws cloudformation package –template
        template.yml --s3-bucket $S3_BUCKET
        --output-template template-export.yml
artifacts:
  type: zip
  files:
    - template-export.yml
```

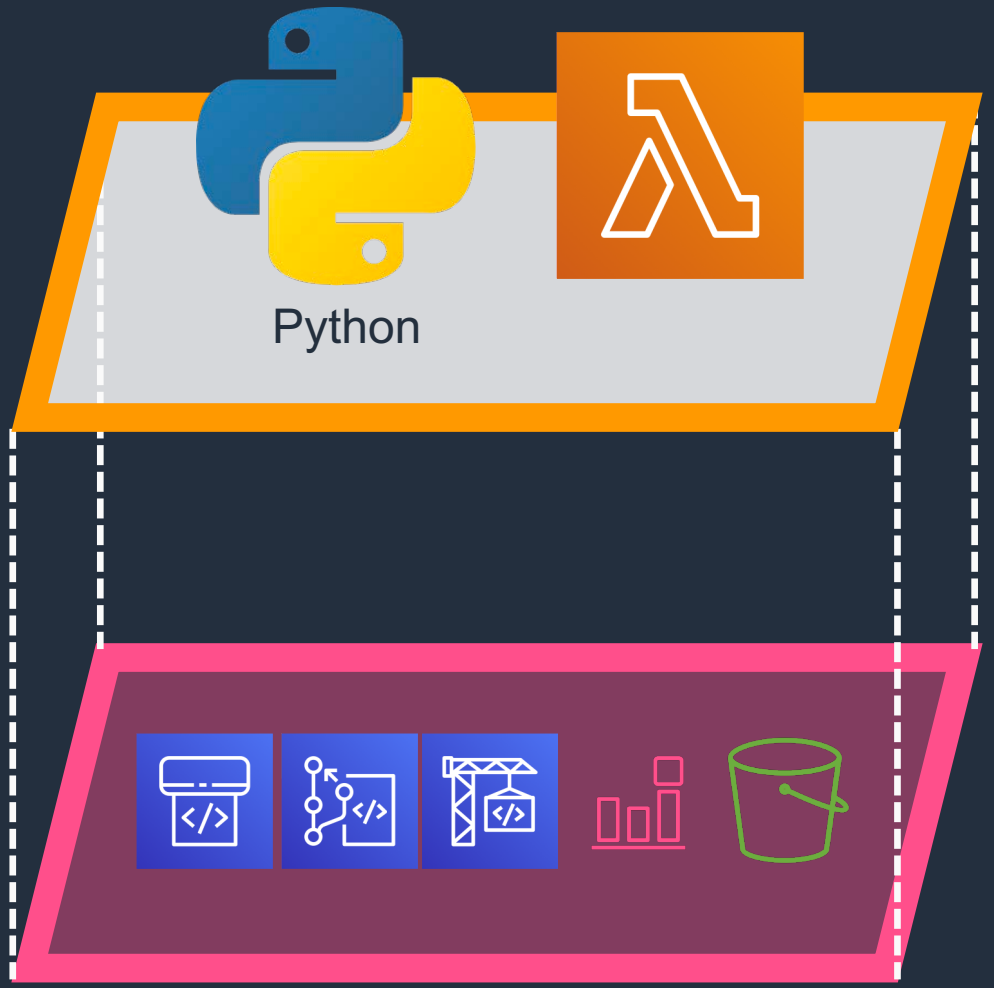
書き換え


```
version: 0.2
phases:
  install:
    commands:
      - pip install -r
        requirements.txt
  build:
    commands:
      - aws cloudformation package –template
        template.yml --s3-bucket $S3_BUCKET
        --output-template template-export.yml
artifacts:
  type: zip
  files:
    - template-export.yml
```

修正後にmaster mergeでパイプライン起動！



CloudFormation Stack の確認



上位Stack

seminar-app
Node.js を Pythonに書き換え済み

下位Stack

serverlessrepo-seminar-app-toolchain
とくに修正なし

まとめ

- サーバーレスの開発も、開発者のお気に入りのIDEで行うことができる
- AWS SAMを使って、CLIによるサーバーレス開発を高速化できる
- CI/CDパイプラインを作成して、Git Pushから自動テストの流れまでを作っておき、属人化する開発、デリバリーから脱却できる

@_kensh