
aws
DEVDAY
GLOBAL SERIES

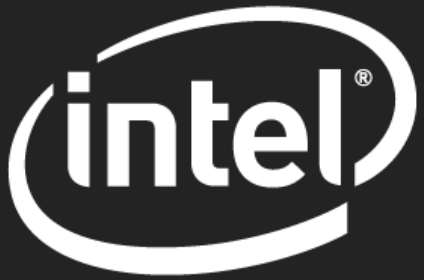
サーバーレスアプリケーションのためのCI/CD パイプライン構築

Solution Architect Takashi Koyanagawa

2017/6/2

amazon web services

THANKS TO OUR FRIENDS AT:

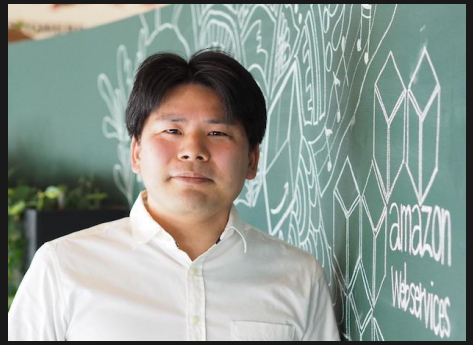


本セッションのFeedbackをお願いします

受付でお配りしたアンケートに本セッションの満足度やご感想などをご記入ください
アンケートをご提出いただきました方には、もれなく素敵なAWSオリジナルグッズをプレゼントさせていただきます

アンケートは各会場出口、パミール3FのEXPO展示会場内にて回収させて頂きます

自己紹介



小梁川　貴史(こやながわ　たかし)

パートナーソリューションアーキテクト

- APNさまへの技術支援とくにIoT向け
  - アーキテクチャの検討支援やレビューなど

経歴

- 電機メーカーにて、自社Webサービスの設計から運用まで経験
  - AWSのユーザとして4年半、開発・運用を経験

好きなサービス

- AWS IoT
- Amazon Kinesis
- AWS Lambda

本セッションの対象

- CI/CDなど基本的の用語はご理解頂けている方
- サーバレス環境の構築に興味がある方

本セッションでご理解いただきたい点

- サーバレス環境でのデプロイ手法
- SAMを使ったデプロイ方法
- SAMとAWS Code系サービスの連携方法

継続的なインテグレーション/デプロイの必要性

リリースプロセスの4つの主なフェーズ

ソース → ビルド → テスト → 運用

- .javaファイルなどのソースコードをチェックイン
- 新しいコードのピアレビュー

- コードのコンパイル
- ユニットテスト
- スタイルチェッカー
- コードメトリック
- コンテナイメージの作成

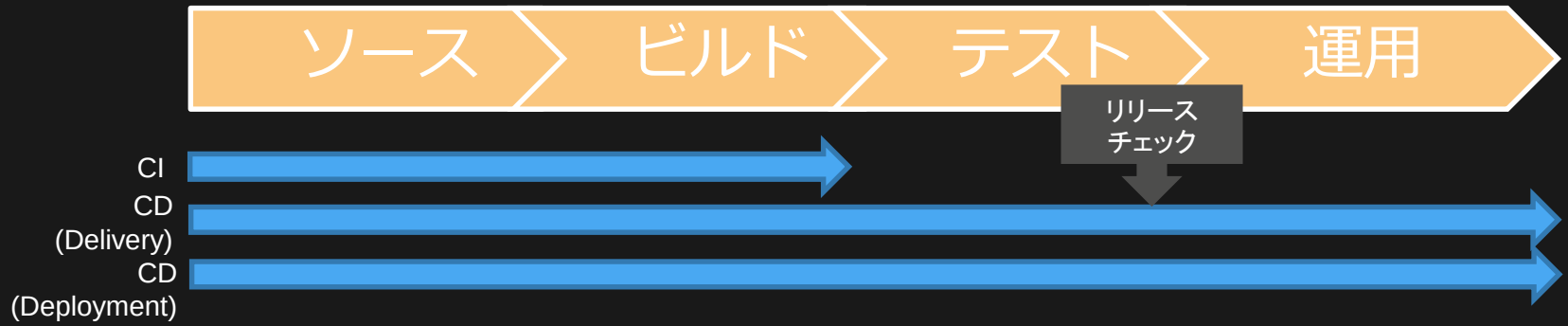
- 他のシステムとの統合テスト
- ロードテスト
- UIテスト
- 侵入テスト

- 本番環境にデプロイ

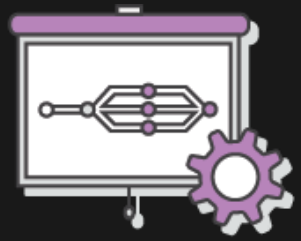



CI / CDとは

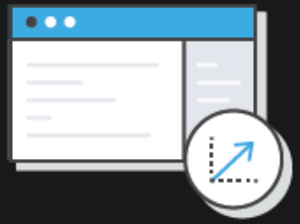
- CI (Continuous Integration)
  - コード変更を定期的にmasterへマージし、ビルド/テストを自動で実行する手法。
- CD(Continuous Delivery/Continuous Deployment)
  - Continuous Delivery:自動化されたテストを通過した後に、Productionへのリリース判断をした後にリリースする手法
  - Continuous Deployment：自動化されたテストを通過した後、自動でプロダクションへリリースを実施する手法



継続的デリバリのメリット



ソフトウェアの
リリースプロセスを
自動化



開発者の
生産性を改善

バグをすばやく
検出して対処

アップデートの
配信を高速化

サーバレス環境構築のフレームワークの紹介

サーバレス環境構築エコシステム

SAM

(Serverless Application Model)

本日のお話

Chalice
Framework

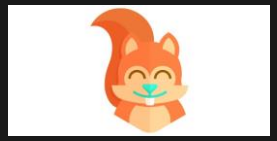
SERVERLESS

Claudia.js

APEX
serverless architecture

λ Gordon

Serverless Application Model



- サーバレスアプリに最適化されたAWS CloudFormationの拡張
- CloudFormtionでサポートされているものは利用可能
- 既存のファンクションをSAMテンプレートとしてエクスポート可能
- apache 2.0ライセンス
    - ユーザがSAMを利用して、エコシステムを作成することも可能
- サーバレス用のリソースタイプ

SAMで指定できるサーバレスのリソース

AWS
Lambda

AWS::Serverless::Function

- AWS Lambdaファンクション
- メモリ設定など、ファンクション自体の設定が可能
- イベントソースの設定が可能

Amazon API
Gateway

AWS::Serverless::Api

- Amazon API GatewayのAPI作成
- エンドポイント経由で呼び出されるリソースとメソッドの定義
- Swaggerを利用して管理す場合は必須定義

Amazon
DynamoDB

AWS::Serverless::SimpleTable

- DynamoDBのシンプルなテーブルを作成
- 詳細な設定が必要な場合は、通常のAWS::DyanmoDB::Tableを使用する

SAMのリリースに伴い以下のコマンドが追加

$ aws cloudformation pakage

- zip fileとして、デプロイパッケージを作成
- Amazon S3バケットへのパッケージアップロード
- S3 URIによるCodeUriプロパティの追加

$ aws cloudformation deploy

- CloudFormationの CreateChangeSet APIをコール
- CloudFormationの ExecuteChangeset APIをコール

Severless用のTempaleteの説明

SAM templeteのyaml

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: Test Environment create by SAM
Resources:
  SAMtestFunction:
    # Lambda Setup
    Type: AWS::Serverless::Function
    Properties:
      Handler: DynamoPut.lambda_handler
      FunctionName: EasySAM_DynamoPut
      Timeout: 10
      Role: arn:aws:iam::[redacted]:role/Lambda_roleToDynamo@Write
      Runtime: python2.7
      Environment:
        Variables:
          dynamoTableName: !Ref Table

  Table:
    Type: AWS::Serverless::SimpleTable
    Properties:
      PrimaryKey:
        Name: test
        Type: String
```

Serverless用のtemplateである宣言

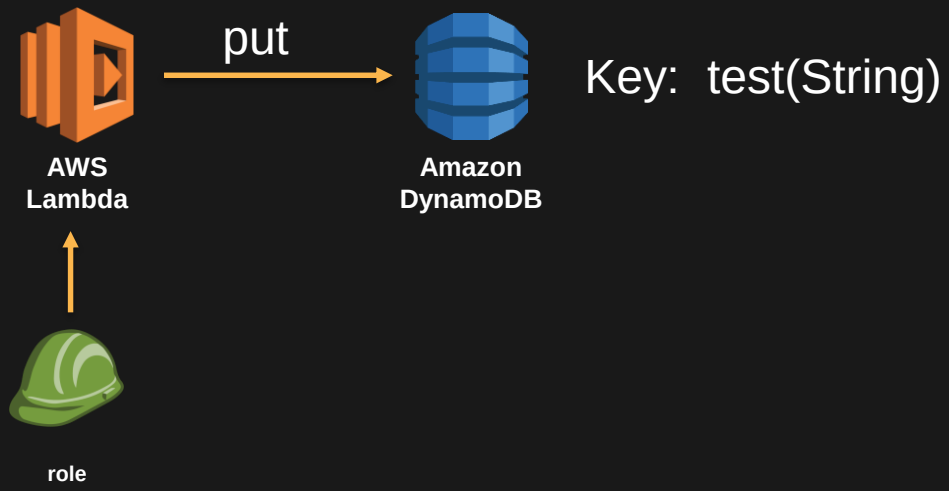
packageコマンド実行

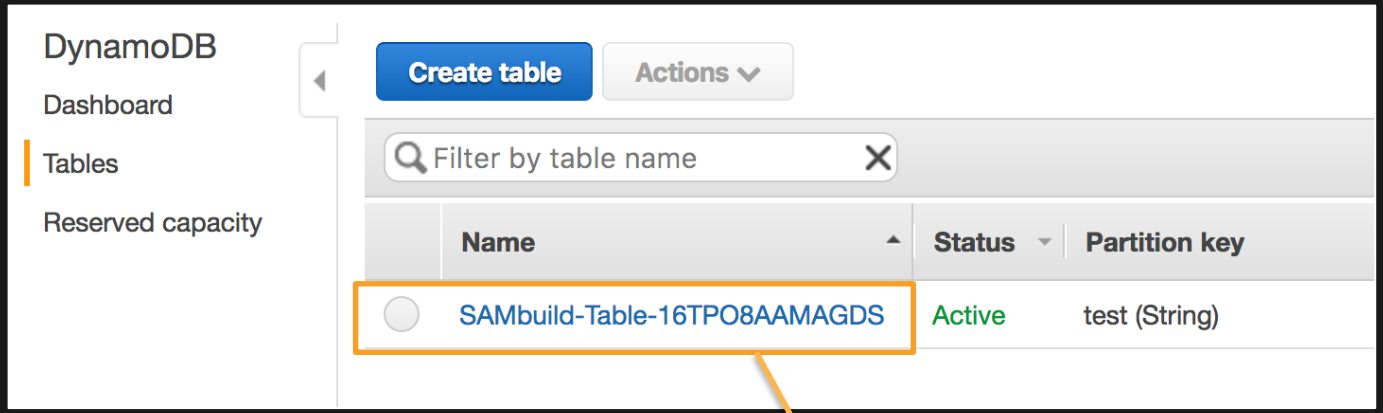
Cloudformation用のファイルへ変換された yaml

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: Test Environment create by SAM
Resources:
  SAMtestFunction:
    Properties:
      CodeUri: s3://[redacted]/573d156b394a0e77e20046566f9a6e0f
      Environment:
        Variables:
          dynamoTableName:
            Ref: Table
      FunctionName: EasySAM_DynamoPut
      Handler: DynamoPut.lambda_handler
      Role: arn:aws:iam::[redacted]:role/Lambda_roleToDynamo@Write
      Runtime: python2.7
      Timeout: 10
    Type: AWS::Serverless::Function
  Table:
    Properties:
      PrimaryKey:
        Name: test
        Type: String
    Type: AWS::Serverless::SimpleTable
Transform: AWS::Serverless-2016-10-31
```

引数で指定したS3バケットへのupload情報が作成される

このテンプレートのアーキテクチャ






```
32          dynamoPut(uuid, data)
33
34          return
35      except Exception as e:
36          print e.message
37          raise e
38
39
```

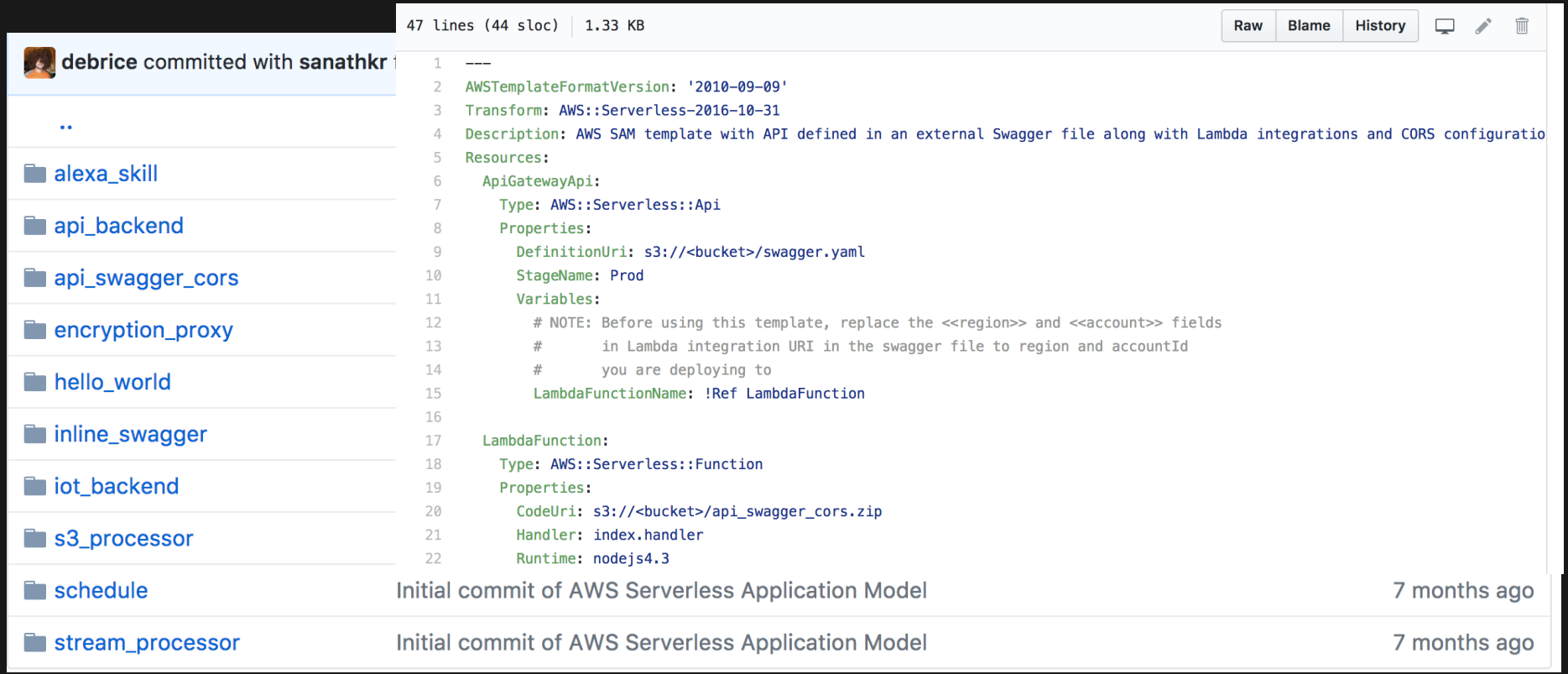
DynamoDBのテーブル名を指定しないで作成、
自動で作成されたテーブル名をtemplateからRefで
参照しているので、Lambdaの環境変数にも適用さ
れている

You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings with
code. Learn more. For storing sensitive information, we recommend encrypting values using KMS and the console's encryption helpers.

Enable encryption helpers

Environment variables dynamoTableName SAMbuild-Table-16TPO8AAMAGDS

SAMが提供するサンプル



debrice committed with sanathkr

..

- alexa_skill
- api_backend
- api_swagger_cors
- encryption_proxy
- hello_world
- inline_swagger
- iot_backend
- s3_processor
- schedule — Initial commit of AWS Serverless Application Model — 7 months ago
- stream_processor — Initial commit of AWS Serverless Application Model — 7 months ago

47 lines (44 sloc) | 1.33 KB — Raw | Blame | History

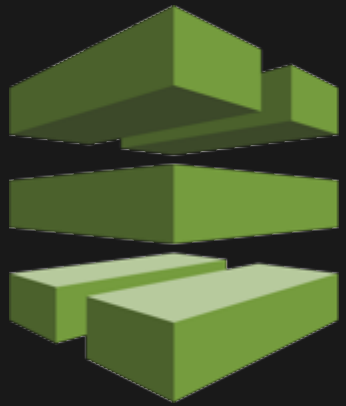
```yaml
---
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Description: AWS SAM template with API defined in an external Swagger file along with Lambda integrations and CORS configuratio
Resources:
  ApiGatewayApi:
    Type: AWS::Serverless::Api
    Properties:
      DefinitionUri: s3://<bucket>/swagger.yaml
      StageName: Prod
      Variables:
        # NOTE: Before using this template, replace the <<region>> and <<account>> fields
        #       in Lambda integration URI in the swagger file to region and accountId
        #       you are deploying to
        LambdaFunctionName: !Ref LambdaFunction

  LambdaFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: s3://<bucket>/api_swagger_cors.zip
      Handler: index.handler
      Runtime: nodejs4.3
```

https://github.com/awslabs/serverless-application-model/tree/master/examples/2016-10-31

SAMとAWS Code関係サービスとの連携

AWS CodePipeline

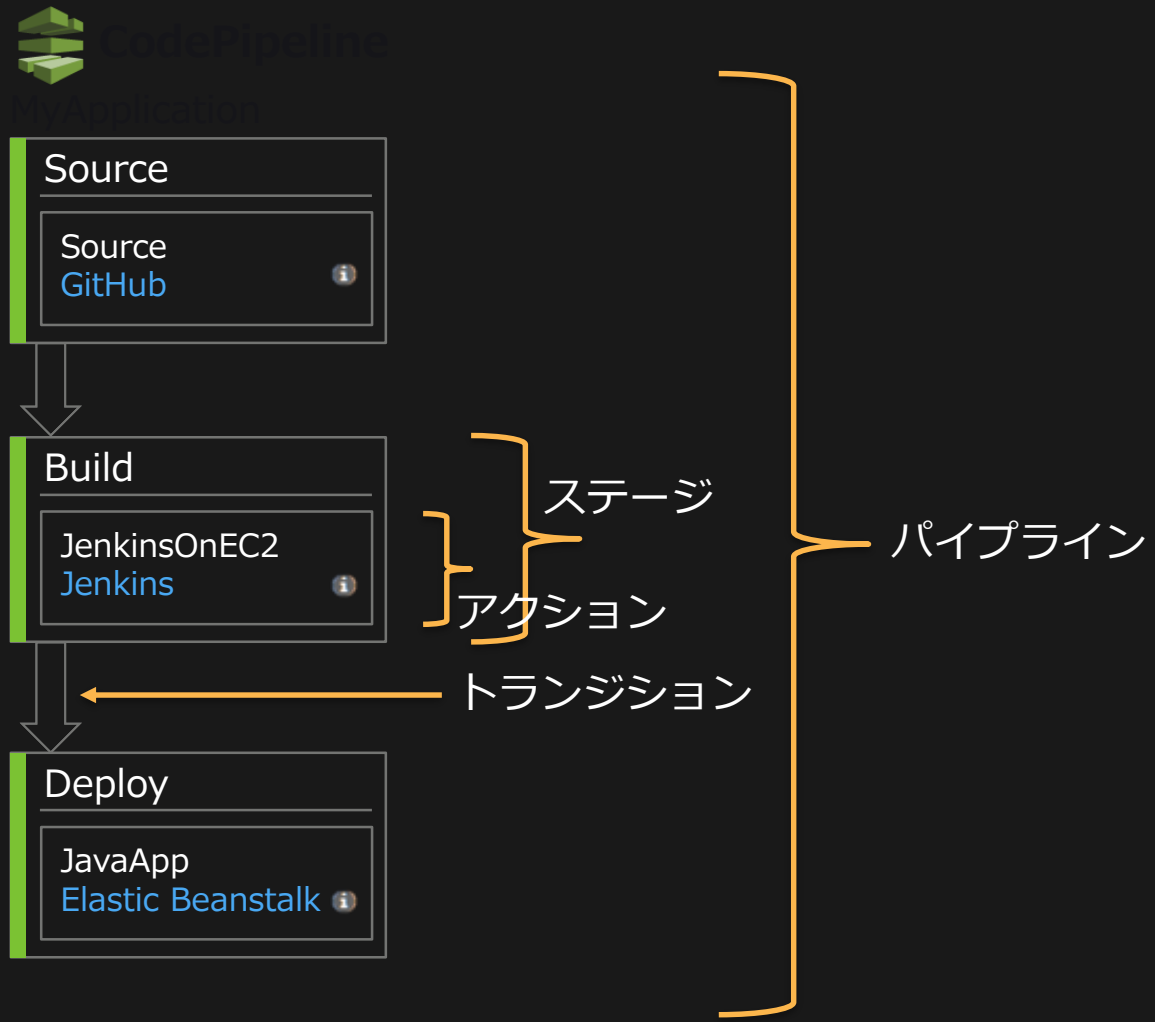


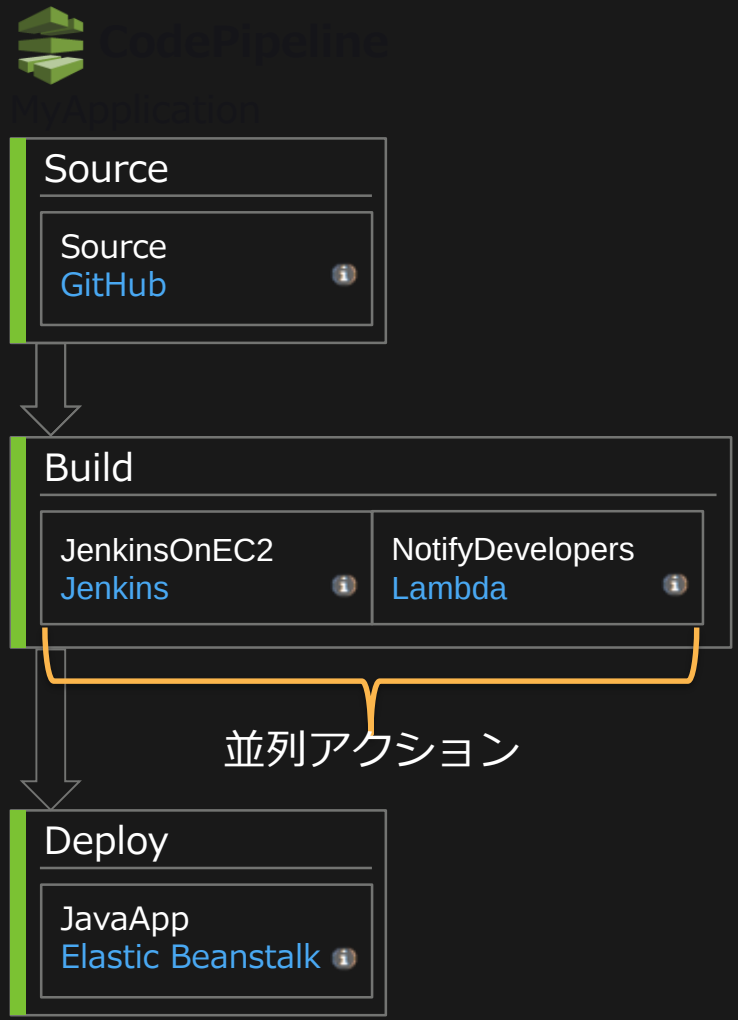
アプリケーションのすばやく信頼できるアップデートを可能にする継続的デリバリサービス

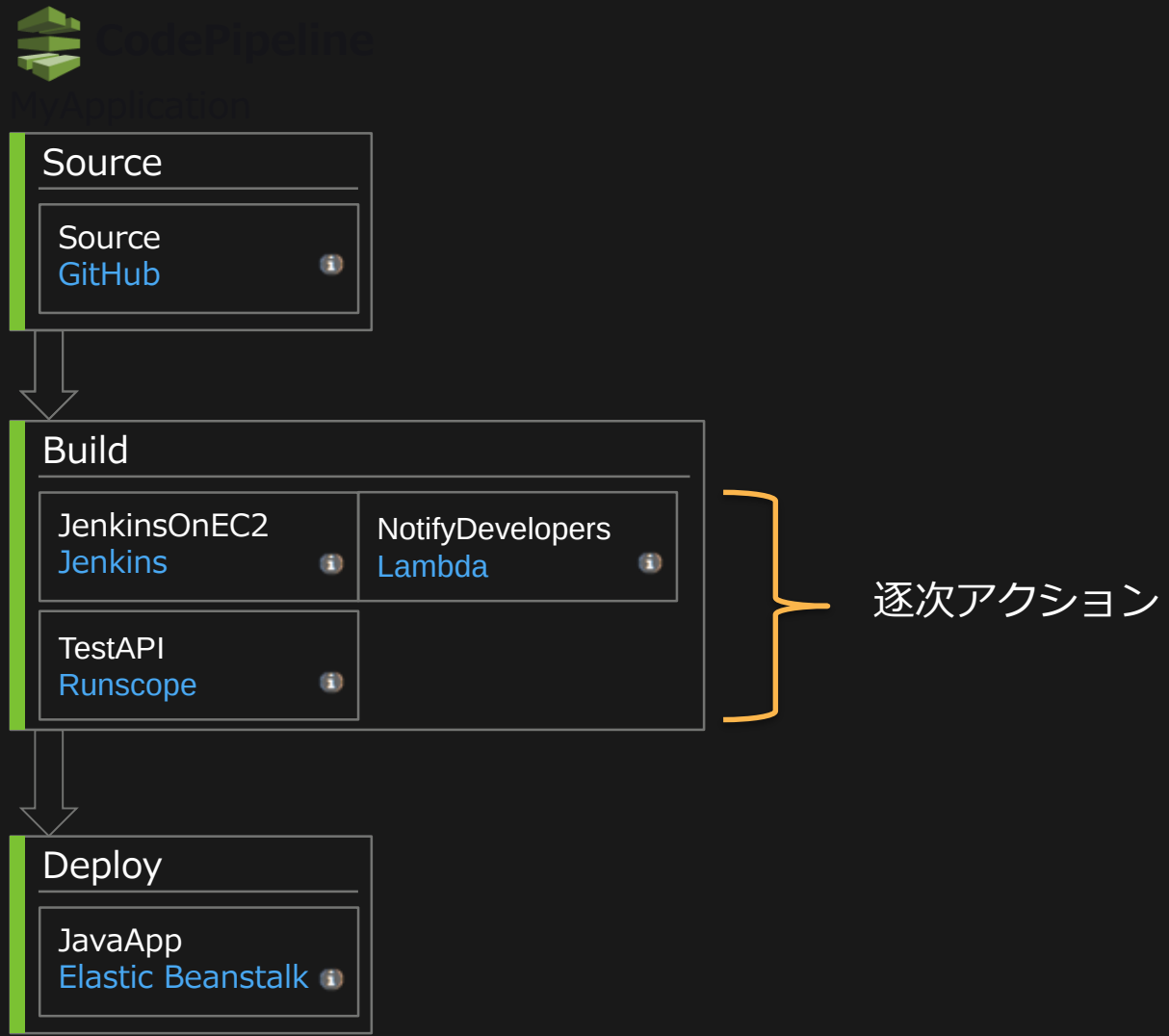
ソフトウェアリリースプロセスのモデル化と見える化

コードが変更されるたびにコードをビルド、テスト、デプロイ

サードパーティツールやAWSとの統合


CodePipeline
MyApplication
Source
Source
GitHub
Build
JenkinsOnEC2
Jenkins
Deploy
JavaApp
Elastic Beanstalk
ステージ
アクション
トランジション
パイプライン


CodePipeline
MyApplication
Source
Source
GitHub
Build
JenkinsOnEC2
Jenkins
NotifyDevelopers
Lambda
並列アクション
Deploy
JavaApp
Elastic Beanstalk


CodePipeline
MyApplication
Source
Source
GitHub
Build
JenkinsOnEC2
Jenkins
NotifyDevelopers
Lambda
TestAPI
Runscope
逐次アクション
Deploy
JavaApp
Elastic Beanstalk

AWS CodeCommit

Secure, scalable, managed Git source control

スターンダードなGit toolが利用可能

Amazon S3のScalability, availability, durability なストレージを利用

Encryption at rest with customer-specific keys

レポジトリサイズの上限なし

Post commit hooks で SNS/Lambdaを呼び出せる

5/27 Tokyo region launch!

AWS CodeBuild

完全なマネージドのビルドサービスでソースコードのコンパイル、実行、テスト、ソフトウェア パッケージの生成をサポート

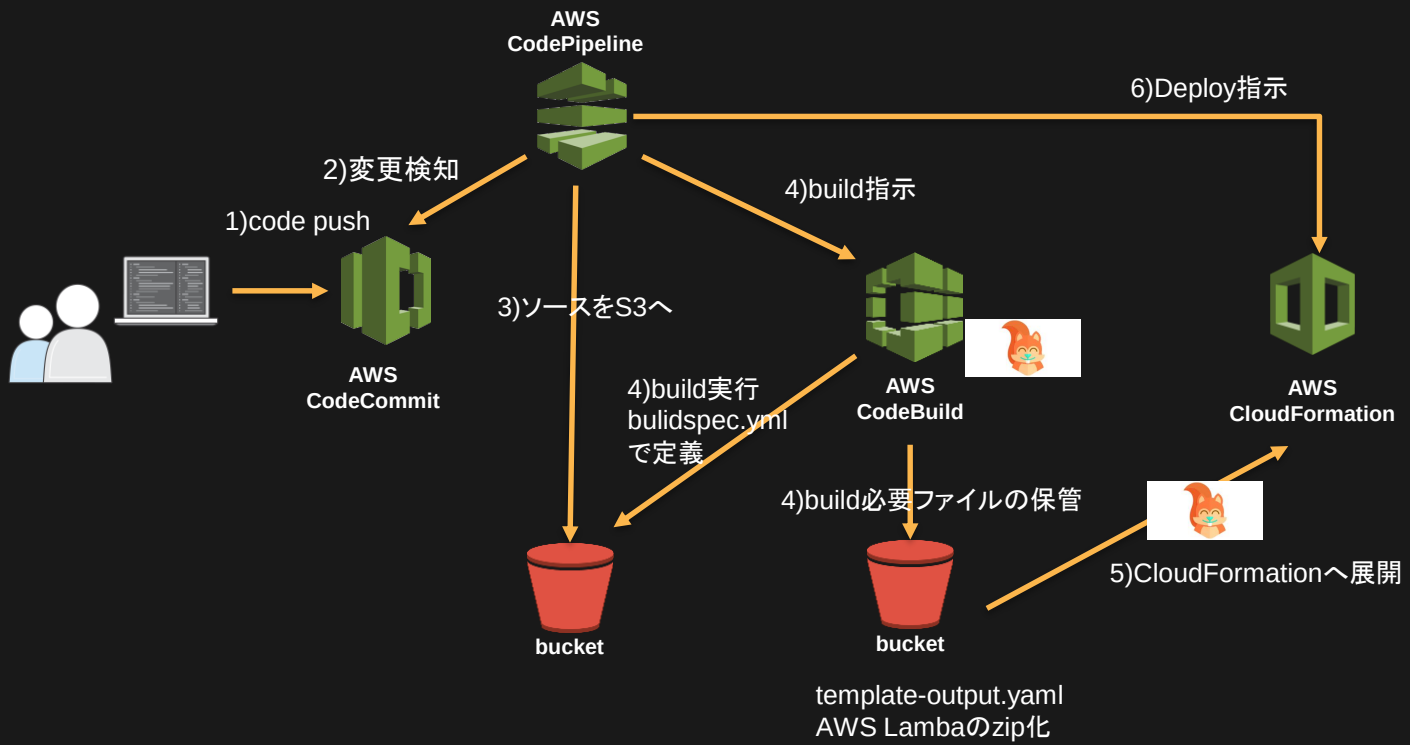
継続的なスケールと同時複数ビルドの実行

Dockerイメージによってニーズにマッチするカスタムなビルド環境を構築可能

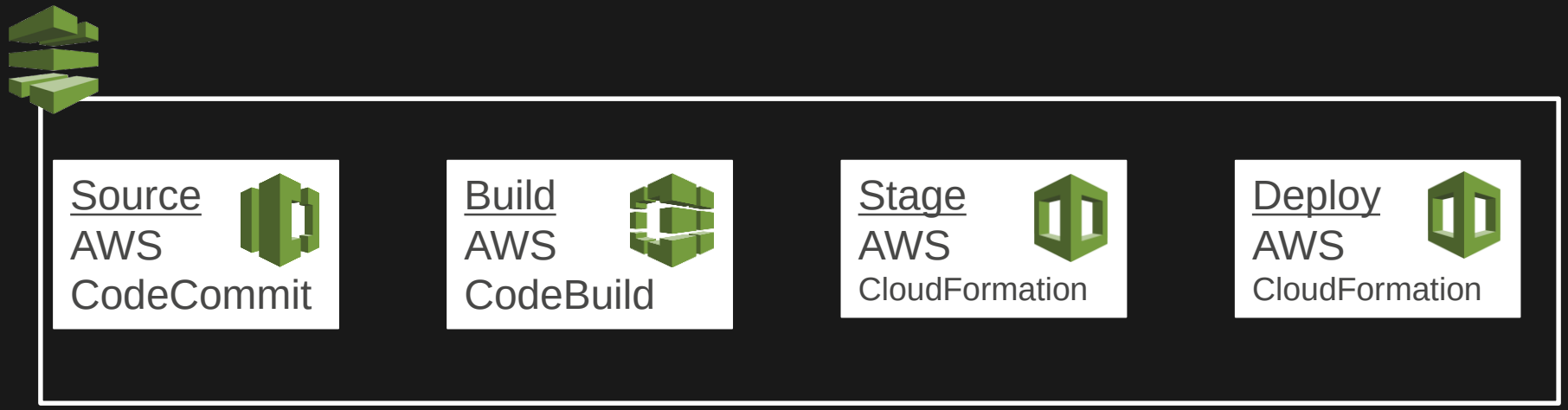
利用したコンピュータ リソース/分のみの支払い

CodePipelineやJenkinsとの統合が可能

AWS CodePipeLineとの連携



AWS CodePipelineでデプロイフローを作る



レポジトリから
ソース取得

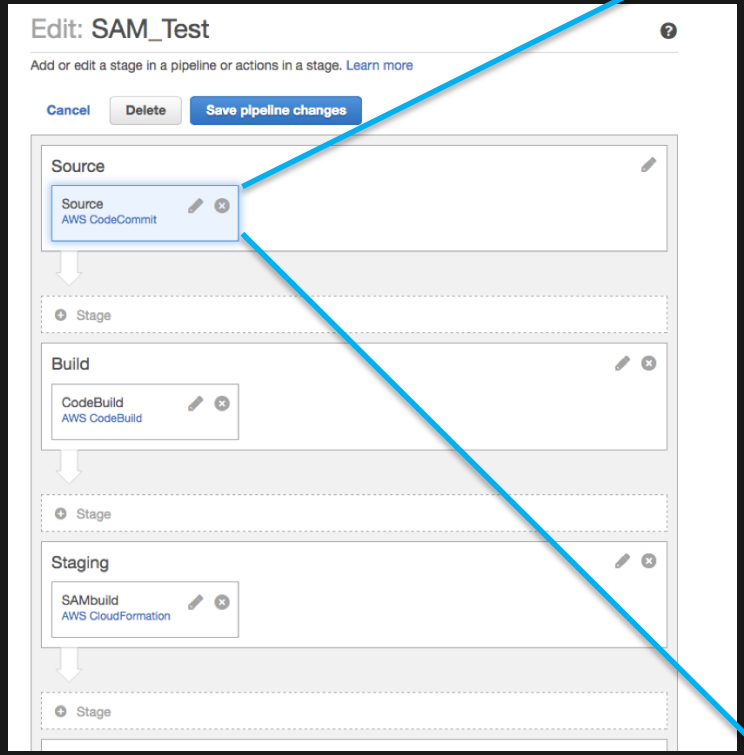
buildspec.ymlの実行
CFn templateの作成

CFnスタックの変更

CFnスタックの実行

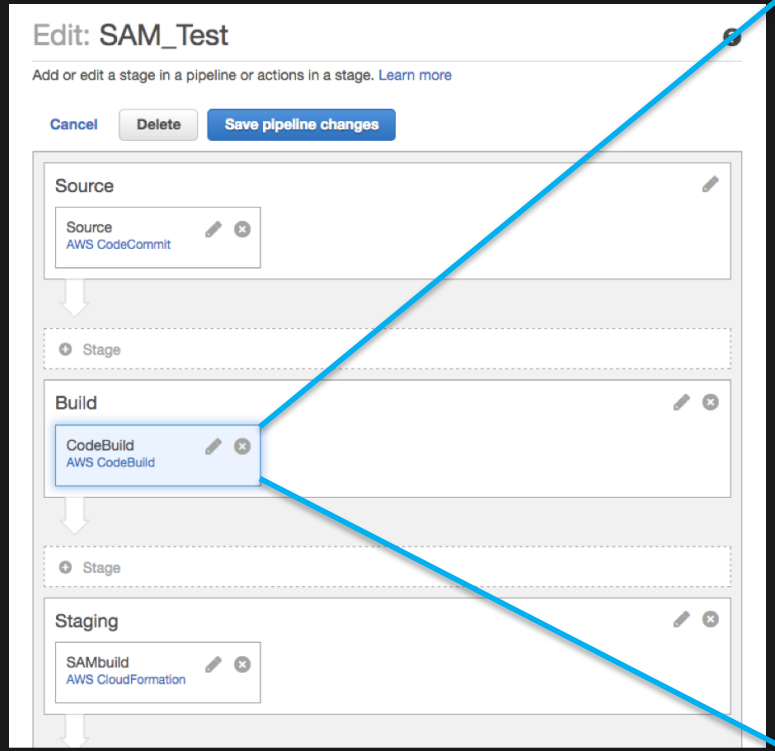
http://docs.aws.amazon.com/ja_jp/lambda/latest/dg/automating-deployment.html

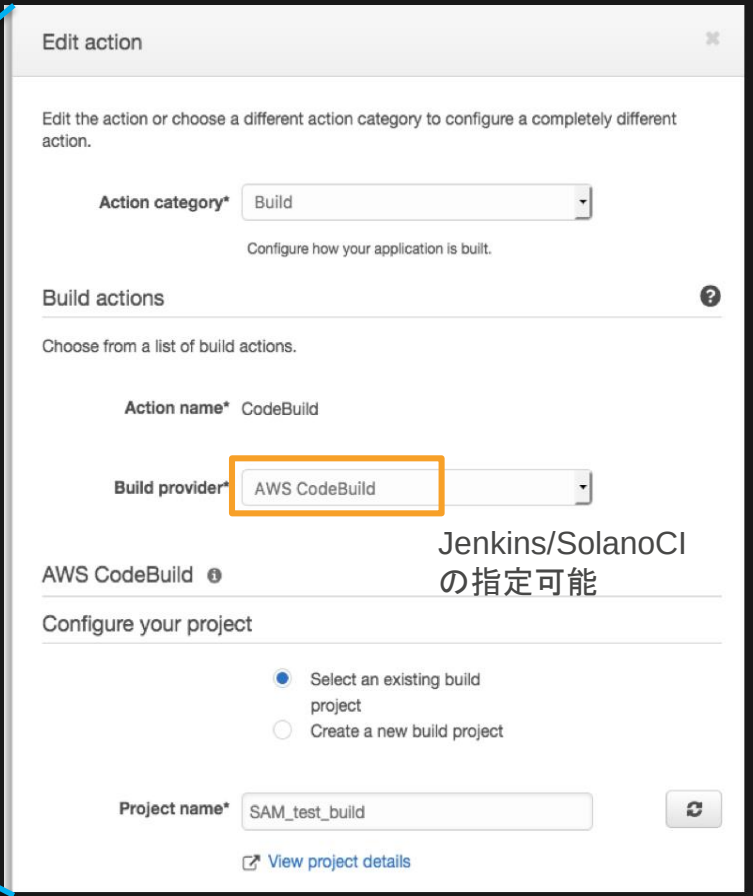
Sourceの設定


Edit: SAM_Test
Add or edit a stage in a pipeline or actions in a stage. Learn more
Cancel
Delete
Save pipeline changes
Source
Source
AWS CodeCommit
Stage
Build
CodeBuild
AWS CodeBuild
Stage
Staging
SAMbuild
AWS CloudFormation
Stage

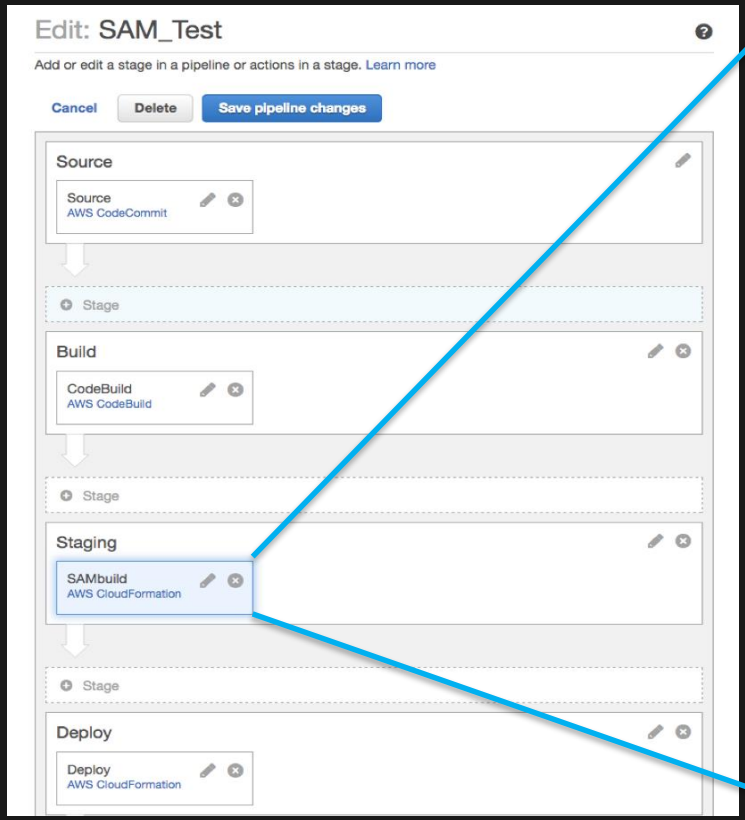

Edit action
Edit the action or choose a different action category to configure a completely different action.
Action category*
Source
Configure where your application is stored.
Source actions
Choose the location and system you use to store code.
Action name*
Source
Github/S3の登録も可能
Source provider*
AWS CodeCommit
AWS CodeCommit
Choose a repository and a branch to use as the source location.
Repository name*
SAM_Test
Branch name*
master
Output artifacts
repository/branch
の指定
Choose a name for the output of this action. Learn more
Output artifact #1
MyApp

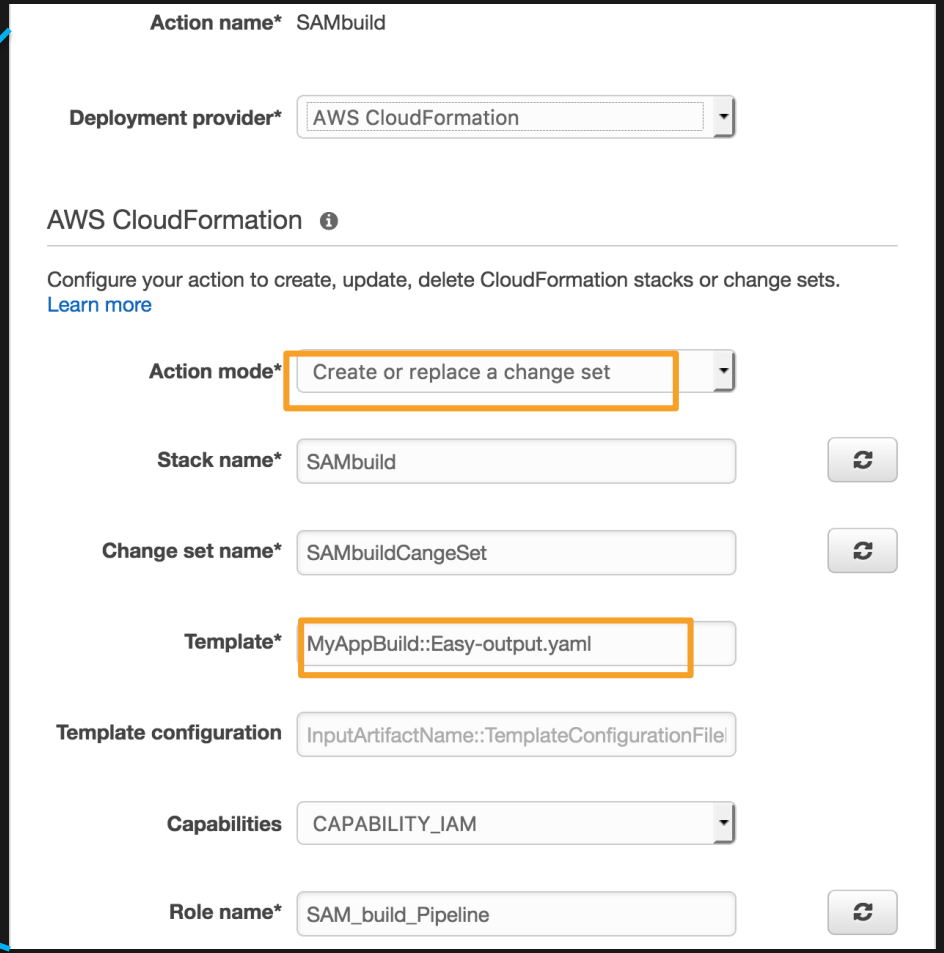
buildの設定


Edit: SAM_Test
Add or edit a stage in a pipeline or actions in a stage. Learn more
Cancel
Delete
Save pipeline changes
Source
Source
AWS CodeCommit
Stage
Build
CodeBuild
AWS CodeBuild
Stage
Staging
SAMbuild
AWS CloudFormation

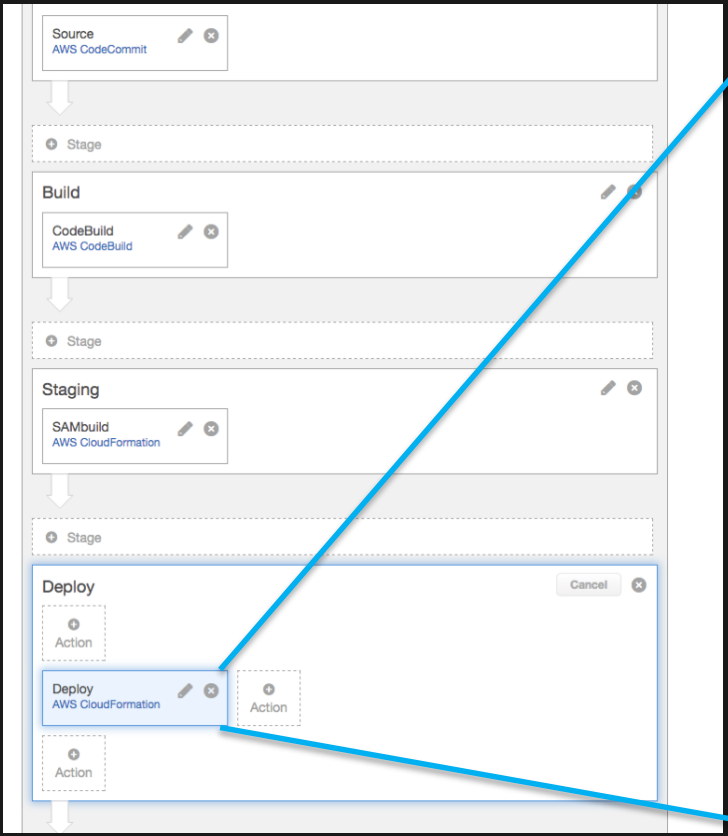

Edit action
Edit the action or choose a different action category to configure a completely different action.
Action category*
Build
Configure how your application is built.
Build actions
Choose from a list of build actions.
Action name*
CodeBuild
Build provider*
AWS CodeBuild
Jenkins/SolanoCIの指定可能
AWS CodeBuild
Configure your project
Select an existing build project
Create a new build project
Project name*
SAM_test_build
View project details

CloudFormation更新





CloudFormation実行設定



Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

Action name* Deploy

Deployment provider* AWS CloudFormation

AWS CloudFormation

Configure your action to create, update, delete CloudFormation stacks or change sets.
Learn more

Action mode* Execute a change set

Stack name* SAMbuild

Change set name* SAMbuildCangeSet

▸ Advanced

本例のソースのスタック

SAMText
- DynamoPut.py
- template.yaml ← SAM用
- buildspec.yml ← ファイル名固定

buildspecはCodebuildの制約でファイル名固定かつ、repostory-rootディレクトリに配置が必須(codebuildの制約)

http://docs.aws.amazon.com/ja_jp/codebuild/latest/userguide/build-spec-ref.html

bulidspec.yml

Cloudformationコマンドを書くのみ

```
1  version: 0.1
2  phases:
3    install:
4      commands:
5        - aws cloudformation package --template-file 入力template --s3-bucket バケット名
6                                     --output-template-file 出力template
7  artifacts:
8    type: zip
9    files:
10     - 出力template
```

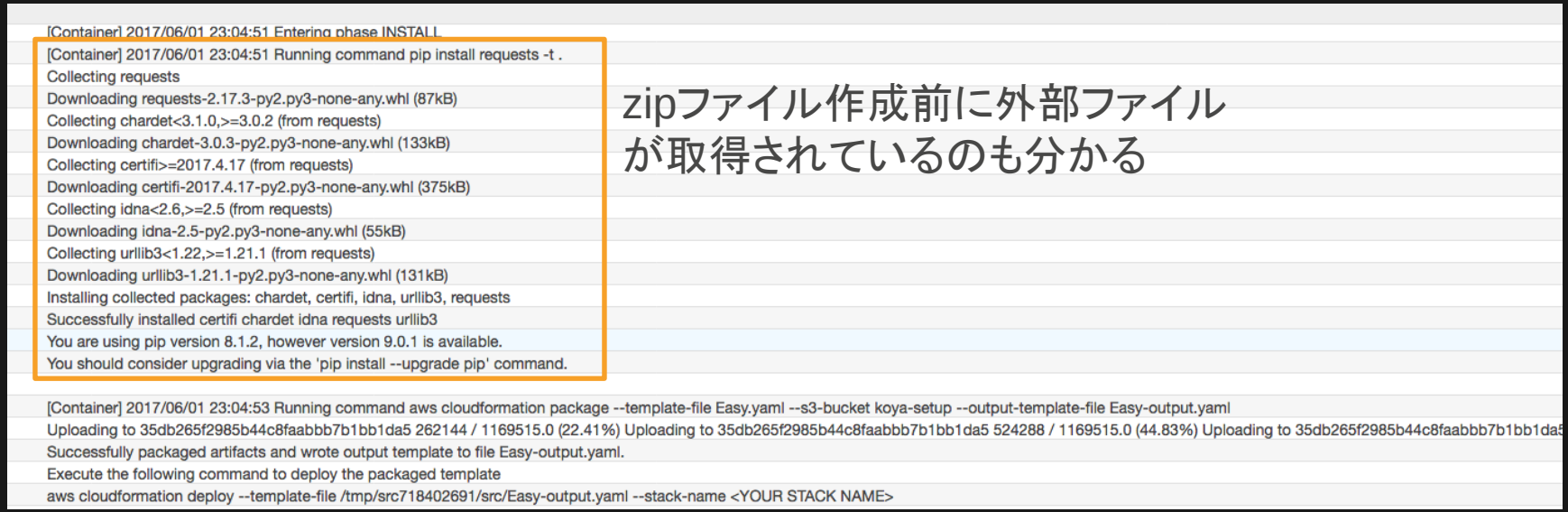
CodePipelineの実行確認

13:40:31	[Container] 2017/06/01 13:40:20 Waiting for agent
13:40:35	[Container] 2017/06/01 13:40:34 Phase is DOWNLOAD_SOURCE
13:40:35	[Container] 2017/06/01 13:40:34 Source is located at /tmp/src142911830/src
13:40:35	[Container] 2017/06/01 13:40:34 YAML location is /tmp/src142911830/src/buildspec.yml
13:40:35	[Container] 2017/06/01 13:40:34 Registering with agent
13:40:35	[Container] 2017/06/01 13:40:34 Phases found in YAML: 1
13:40:35	[Container] 2017/06/01 13:40:34 INSTALL: 1 commands
13:40:35	[Container] 2017/06/01 13:40:34 Phase complete: DOWNLOAD_SOURCE Success: true
13:40:35	[Container] 2017/06/01 13:40:34 Phase context status code: Message:
13:40:35	[Container] 2017/06/01 13:40:34 Processing plaintext environment variables
13:40:35	[Container] 2017/06/01 13:40:34 Processing build-level environment variables
13:40:35	[Container] 2017/06/01 13:40:34 {"Name":"SAM test","CODEBUILD_RESOLVED_SOURCE_VERSION":"73c98f80ce68487129a3cf25f89cd741ef6975db"}
13:40:35	[Container] 2017/06/01 13:40:34 Name = SAM test
13:40:35	[Container] 2017/06/01 13:40:34 CODEBUILD_RESOLVED_SOURCE_VERSION = 73c98f80ce68487129a3cf25f89cd741ef6975db
13:40:35	[Container] 2017/06/01 13:40:34 Processing builtin environment variables
13:40:35	[Container] 2017/06/01 13:40:34 Moving to directory /tmp/src142911830/src
13:40:35	[Container] 2017/06/01 13:40:34 Entering phase INSTALL
13:40:35	[Container] 2017/06/01 13:40:34 Running command aws cloudformation package --template-file l --s3-bucket l p --output-template-file
13:40:45	Uploading to 58bf2dba02f1717783d8ea62f26fb704 1124 / 1124.0 (100.00%)
13:40:45	Successfully packaged artifacts and wrote output template to file E
13:40:45	Execute the following command to deploy the packaged template
13:40:45	aws cloudformation deploy --template-file /tmp/src142911830/src/Easy-output.yaml --stack-name <YOUR STACK NAME>
13:40:45	
13:40:45	[Container] 2017/06/01 13:40:43 Phase complete: INSTALL Success: true
13:40:45	[Container] 2017/06/01 13:40:43 Phase context status code: Message:
13:40:45	[Container] 2017/06/01 13:40:44 Preparing to copy artifacts
13:40:45	[Container] 2017/06/01 13:40:44 Expanding base directory path
13:40:45	[Container] 2017/06/01 13:40:44 Assembling file list
13:40:45	[Container] 2017/06/01 13:40:44 Expanding .
13:40:45	[Container] 2017/06/01 13:40:44 Expanding artifact file paths for base directory .
13:40:45	[Container] 2017/06/01 13:40:44 Assembling file list
13:40:45	[Container] 2017/06/01 13:40:44 Expanding Easy-output.yaml
13:40:45	[Container] 2017/06/01 13:40:44 Found 1 file(s)
13:40:45	[Container] 2017/06/01 13:40:44 Phase complete: UPLOAD_ARTIFACTS Success: true
13:40:45	[Container] 2017/06/01 13:40:44 Phase context status code: Message:

外部ライブラリを利用したいケース

```
1  version: 0.1
2  phases:
3    install:
4      commands:
5        - pip install requests -t .
6        - aws cloudformation package --template-file 入力template --s3-bucket バケット名
7                                     --output-template-file 出力template
8  artifacts:
9    type: zip
10   files:
11     - 出力template
```

Cloudwachlogsで確認



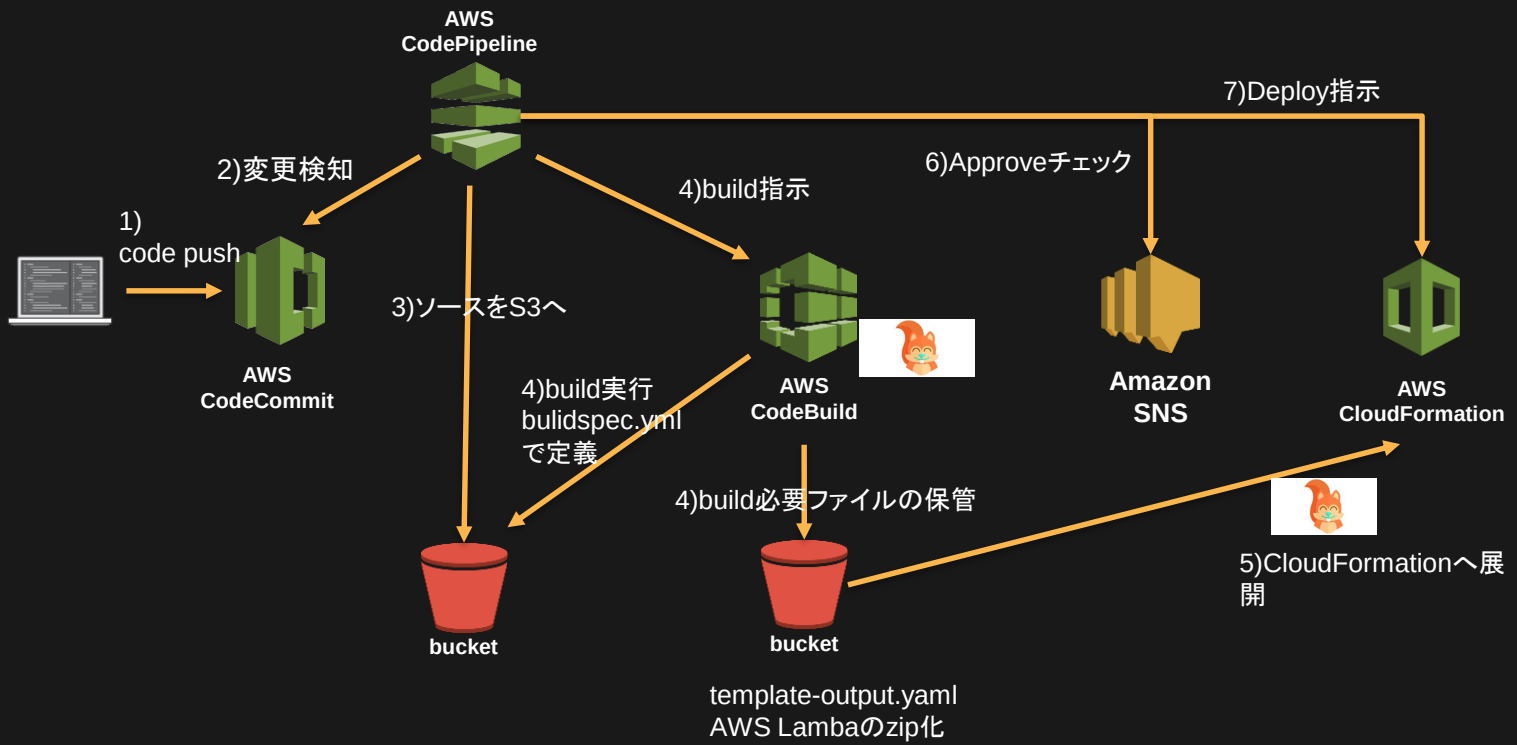
zipファイル作成前に外部ファイル
が取得されているのも分かる

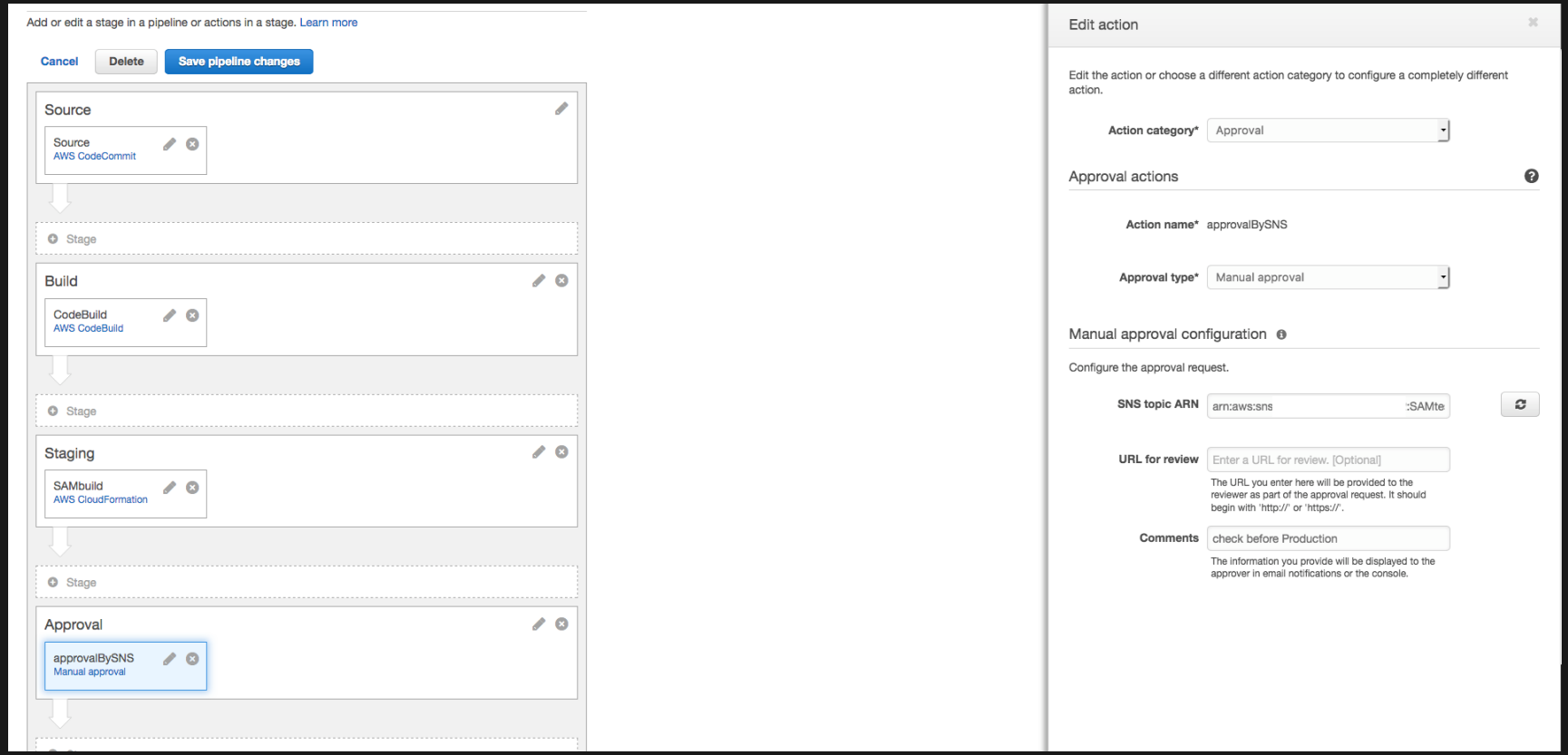
初回構築後にAWS Lambdaを修正してpush

差分はLambdaのみなので、Lambdaのdeployのみ

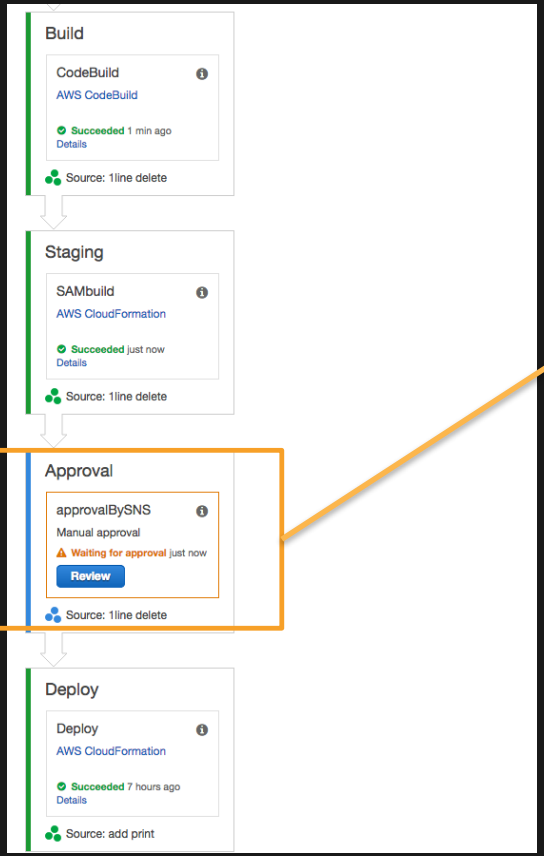
23:17:03 UTC+0900	UPDATE_COMPLETE	AWS::CloudFormation::Stack	SAMbuild	
23:17:02 UTC+0900	UPDATE_COMPLETE_CLEANUP_IN_PROGRESS	AWS::CloudFormation::Stack	SAMbuild	
23:17:00 UTC+0900	UPDATE_COMPLETE	AWS::Lambda::Function	SAMtestFunction	
23:17:00 UTC+0900	UPDATE_IN_PROGRESS	AWS::Lambda::Function	SAMtestFunction	Failed to set tags on AWS::Lambda::Function. IAM permissions lambda:ListTags, lambda:TagResource and lambda:UntagResource are required to tag AWS::Lambda::Function resources and to propagate stack level tags.
23:17:00 UTC+0900	UPDATE_IN_PROGRESS	AWS::Lambda::Function	SAMtestFunction	
23:16:55 UTC+0900	UPDATE_IN_PROGRESS	AWS::CloudFormation::Stack	SAMbuild	User Initiated
22:43:25 UTC+0900	CREATE_COMPLETE	AWS::CloudFormation::Stack	SAMbuild	
22:43:23 UTC+0900	CREATE_COMPLETE	AWS::Lambda::Function	SAMtestFunction	
22:43:23 UTC+0900	CREATE_IN_PROGRESS	AWS::Lambda::Function	SAMtestFunction	Resource creation Initiated
22:43:22 UTC+0900	CREATE_IN_PROGRESS	AWS::Lambda::Function	SAMtestFunction	
22:43:19 UTC+0900	CREATE_COMPLETE	AWS::DynamoDB::Table	Table	
22:42:49 UTC+0900	CREATE_IN_PROGRESS	AWS::DynamoDB::Table	Table	Resource creation Initiated
22:42:48 UTC+0900	CREATE_IN_PROGRESS	AWS::DynamoDB::Table	Table	
22:42:45 UTC+0900	CREATE_IN_PROGRESS	AWS::CloudFormation::Stack	SAMbuild	User Initiated
22:41:39 UTC+0900	REVIEW_IN_PROGRESS	AWS::CloudFormation::Stack	SAMbuild	User Initiated

承認フローを追加

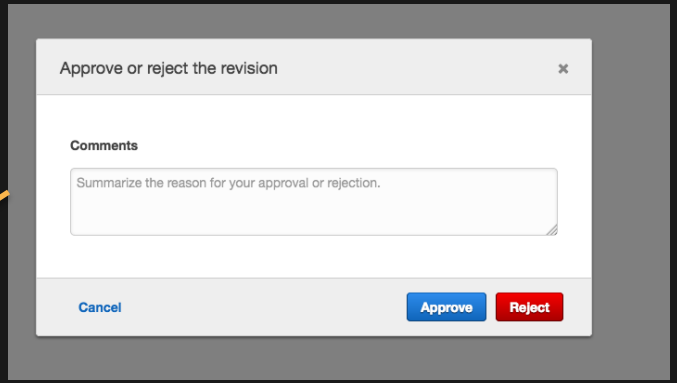

AWS
CodePipeline
7)Deploy指示
2)変更検知
6)Approveチェック
4)build指示
1)
code push
3)ソースをS3へ
AWS
CodeCommit
AWS
CodeBuild
Amazon
SNS
AWS
CloudFormation
4)build実行
bulidspec.yml
で定義
4)build必要ファイルの保管
5)CloudFormationへ展開
bucket
bucket
template-output.yaml
AWS Lambaのzip化


Add or edit a stage in a pipeline or actions in a stage. Learn more
Cancel
Delete
Save pipeline changes
Source
Source
AWS CodeCommit
Stage
Build
CodeBuild
AWS CodeBuild
Stage
Staging
SAMbuild
AWS CloudFormation
Stage
Approval
approvalBySNS
Manual approval
Edit action
Edit the action or choose a different action category to configure a completely different action.
Action category*
Approval
Approval actions
Action name*
approvalBySNS
Approval type*
Manual approval
Manual approval configuration
Configure the approval request.
SNS topic ARN
arn:aws:sns
:SAMte
URL for review
Enter a URL for review. [Optional]
The URL you enter here will be provided to the reviewer as part of the approval request. It should begin with 'http://' or 'https://'.
Comments
check before Production
The information you provide will be displayed to the approver in email notifications or the console.

承認フローを追加

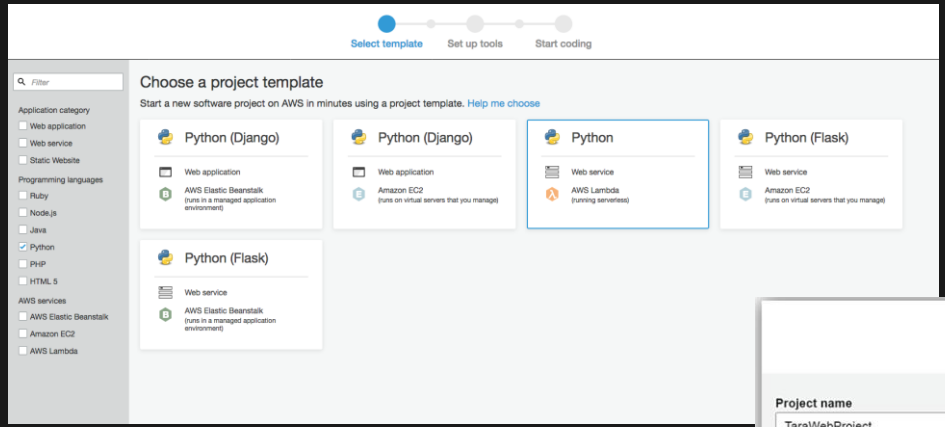


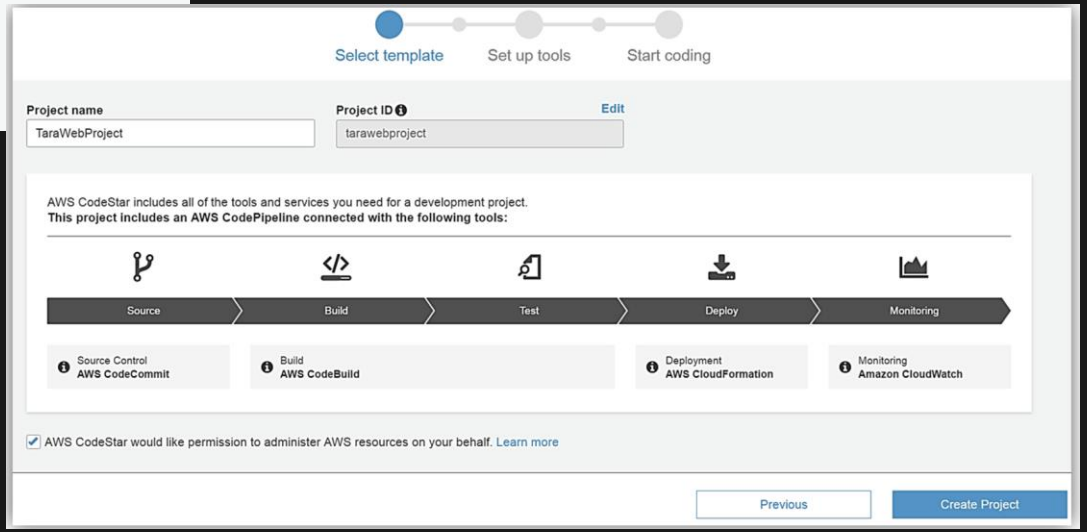
SNSを作成し、新しいアクション=>Action category=>
approvalでSNSを関連付けるのみで承認フローが作成できる



AWS CodeStar

幾つかのガイドに従うだけ



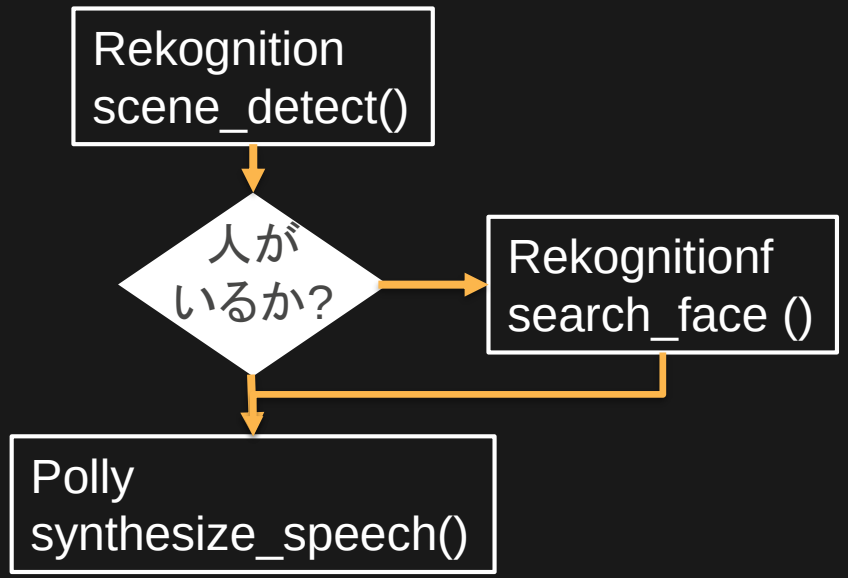


https://aws.amazon.com/jp/blogs/news/new-aws-codestar/

CI/CDのためのプログラミング

AWS Lambdaの設計

テストし易い/管理しやすい単位で設計をする
例えば、uploadされたphotoの解析のユースケース



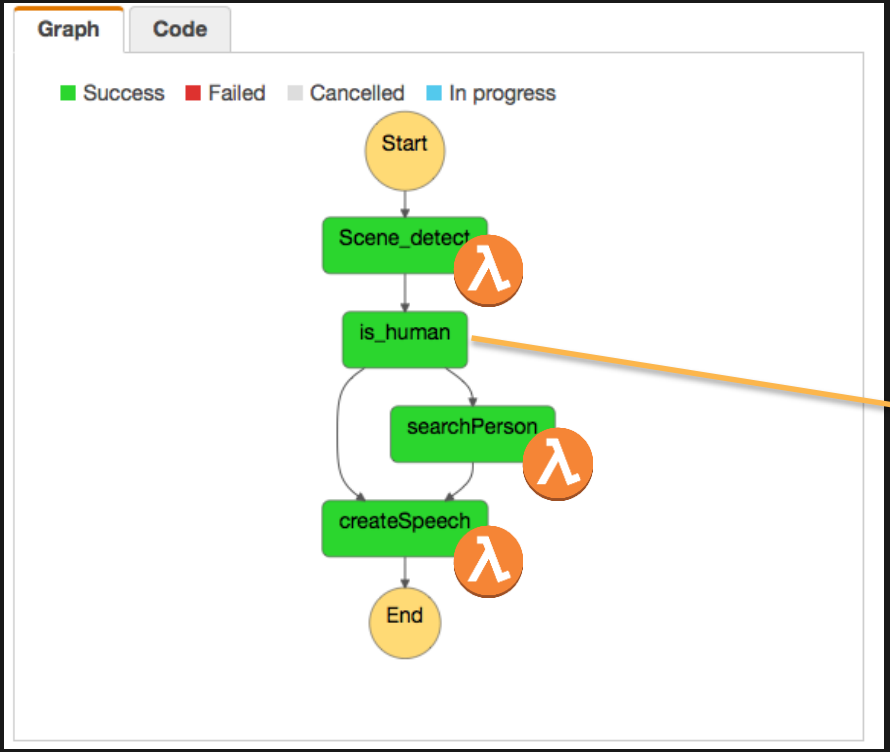
1つのLambdaの中でも表現可能だが、変更するとテスト対象の範囲は全体になる
モノリシックなロジック

AWS Step Functions

- 視覚的なワークフローの提供
- 分散アプリケーション/マイクロサービスの設定が可能
- JSONでのワークフロー定義（プログラム開発不要)

- Amazon EC2 / Amazon ECSとのコラボレーション設定も可能

AWS Step Functions




```
"States": {
  "Scene_detect": {
    "Type": "Task",
    "Resource": "LAMBDA-ARN-1",
    "Next": "is_human"
  },
  "is_human":{
    "Type":"Choice",
    "Choices":[
      {
        "Variable": "$.HUMAN",
        "BooleanEquals": true,
        "Next": "searchPerson"
      }],
      "Default": "createSpeech"
    },
```

テスト/運用しやすいようなプログラミングへ

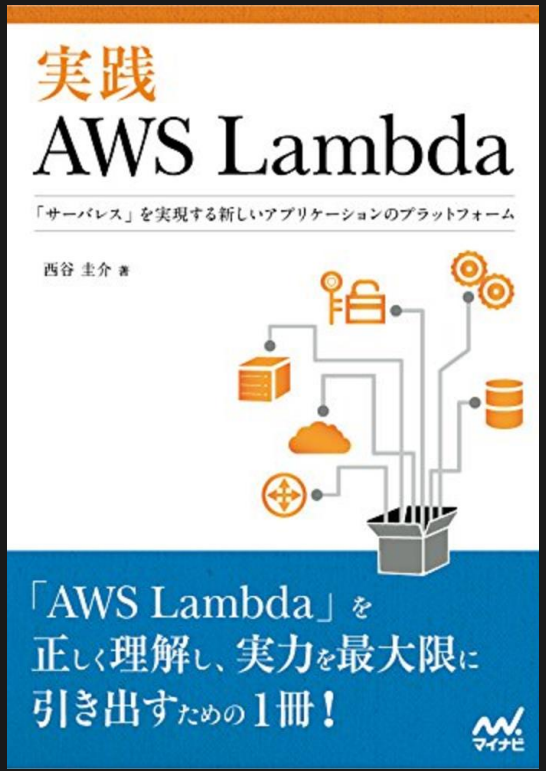
- AWS Lambda環境変数の利用
  - 例えば、DB接続し、パスワード、URLのような環境情報をハードコーディングをせずに、環境変数から取得する
  - 環境変数でlogの出力レベルを変更できるようにする
  など

まとめ

- サーバレス環境でのデプロイ手法
  - SAMを利用した継続的なデプロイワークフロー手法
- CI/CDを支えるAWSサービス
  - SAMを使うことで従来の知識との組み合わせのレバレッジが生まれる
  - ログの一元管理も一つのメリット

サーバレスだからといって特別なことはない。通常のアプリケーションと同じように開発/運用が出来る
一方でテストの品質は人依存、浮いた工数をテストレビューやテスト開発にあてることで品質を向上させることが可能

AWS Lambdaの本が出ます



AWS Lambdaを網羅した本を出します

2017年6月9日マイナビ出版より出版予定

3,240円（税込）

Amazonで予約受け付け中
http://amzn.asia/ew2WWPm

Thank You!

Don’t Forget Evaluations!