



このコンテンツは公開から3年以上経過しており内容が古い可能性があります  
最新情報については[サービス別資料](#)もしくはサービスのドキュメントをご確認ください

# [AWS Black Belt Online Seminar]

## Dive Deep into AWS Chalice

ソリューションカットシリーズ

アマゾンウェブサービスジャパン株式会社

ソリューションアーキテクト 鈴木 哲詩

2019/06/19

AWS 公式 Webinar

<https://amzn.to/JPWebinar>



過去資料

<https://amzn.to/JPArchive>



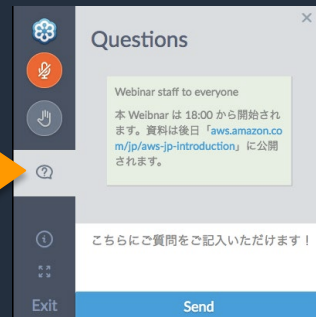
# AWS Black Belt Online Seminar とは

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

質問を投げることができます！

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

- ① 吹き出しをクリック
- ② 質問を入力
- ③ Sendをクリック



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

# 内容についての注意点

- 本資料では2019年6月19日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(<http://aws.amazon.com>)にてご確認ください。
- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。
- 価格は税抜表記となっています。日本居住者のお客様が東京リージョンを使用する場合、別途消費税をご請求させていただきます。
- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at <http://aws.amazon.com/agreement/>. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

# アジェンダ

- Chalice とは
- WEB API 実装のための基本機能
- 一歩進んだ API 実装のための機能
- Amazon API Gateway 以外と連携する Lambda 関数
- さらにもう一歩先に
- 実践的開発プロセス

# Chalice とは

# Chalice とは

- Python 製 OSS サーバーレスアプリケーションフレームワーク
  - Apache License 2.0
  - GitHub の aws org で公開されている
  - 2017/7/31 に 正式リリース
  - boto3, botocore の開発者による継続的なメンテナンス
- “AWS Chalice を使用すると、Amazon API Gateway と AWS Lambda を使用するアプリケーションを迅速に作成してデプロイすることができます:
  - アプリケーションを作成・デプロイ・管理するためのコマンドラインツール
  - Python コードでビューを宣言するための使いやすいAPI
  - 自動 IAM ポリシー生成” \*

\*原文

<https://chalice.readthedocs.io/en/latest>



# 注意

- 本資料は **Chalice** 1.8.0 時点での情報で構成されています
- 後方互換性を非常に重要視していますが
  - 1.x 代に限定されているのでメジャーバージョンアップの際には API がガラリと変わっている可能性もあります
- スライドは文字での説明が控えめです
  - 公開される(されている)ビデオと併せてご参照いただくことを推奨
- 想定オーディエンス
  - Python についてある程度ご理解がある
  - AWS Lambda を使ったことがある

# 注意

- 本資料は **Chalice** 1.8.0 時点での情報で構成されています
- 後方互換性を非常に重要視していますが
  - 1.x 代に限定されているのでメジャーバージョンアップの際には API がガラリと変わっている可能性もあります
- スライドは文字での説明が控えめです
  - 公開される(されている)ビデオと併せてご参照いただくことを推奨
- ~~• 想定オーディエンス~~
  - ~~• Python についてある程度ご理解がある~~
  - ~~• AWS Lambda を使ったことがある~~

後半以降！



# Chalice とは

試してみましよう！

# スケルトンプロジェクトを作成

```
$ pip install chalice
```

```
$ chalice --help
```

```
Usage: chalice [OPTIONS] COMMAND [ARGS]...
```

```
▪  
▪  
▪
```

- *pip* で *chalice* をインストール
- *easy\_install* でも OK

# スケルトンプロジェクトを作成

```
$ pip install chalice
```

- `--help` オプションをつけて実行

```
$ chalice --help
```

```
Usage: chalice [OPTIONS] COMMAND [ARGS]...
```

- 
- 
- 

- パスが通っていることを確認

# スケルトンプロジェクトを作成

```
$ chalice new-project example
```

```
$ tree -aC example/  
example/
```

```
|— .chalice  
|   |— config.json  
|— .gitignore  
|— app.py  
|— requirements.txt
```

1 directory, 4 files

- *new-project* サブコマンドによってプロジェクトの雛形を作成
- 任意のプロジェクト名を引数に

# スケルトンプロジェクトを作成

```
$ chalice new-project example
```

```
$ tree -aC example/
```

```
example/
```

```
|— .chalice
|— config.json
|— .gitignore
|— app.py
|— requirements.txt
```

- 設定ファイルなどが設置されるディレクトリ

- 設定ファイル
- デプロイパッケージアーカイブ
- デプロイ済みリソース情報
- IAM ポリシー定義

1 directory, 4 files

# スケルトンプロジェクトを作成

```
$ chalice new-project example
```

```
$ tree -aC example/  
example/
```

```
|— .chalice  
|   |— config.json  
|— .gitignore  
|— app.py  
|— requirements.txt
```

1 directory, 4 files

- 依存ライブラリを記述
- デフォルトでは空ファイル
- 記述された依存ライブラリをデプロイパッケージアーカイブに含む

# スケルトンプロジェクトを作成

```
$ chalice new-project example
```

```
$ tree -aC example/  
example/
```

```
├── .chalice  
│   └── config.json  
├── .gitignore  
├── app.py  
└── requirements.txt
```

1 directory, 4 files

```
from chalice import Chalice  
  
app = Chalice(app_name='example')  
  
@app.route('/')  
def index():  
    return {'hello': 'world'}  
  
# The view function above will return {"hello": "world"}  
# whenever you make an HTTP GET request to '/'.  
#  
# Here are a few more examples:  
#  
# @app.route('/hello/{name}')  
# def hello_name(name):  
#     # '/hello/james' -> {"hello": "james"}  
#     return {'hello': name}  
#  
# @app.route('/users', methods=['POST'])  
# def create_user():  
#     # This is the JSON body the user sent in their POST request.  
#     user_as_json = app.current_request.json_body  
#     # We'll echo the json body back to the user in a 'user' key.  
#     return {'user': user_as_json}  
#  
# See the README documentation for more examples.  
#
```

# 自動生成されたサンプルアプリケーション

```
from chalice import Chalice  
  
app = Chalice(app_name='example')
```

```
@app.route('/')  
def index():  
    return {'hello': 'world'}
```

- ライブラリをインポート
- アプリケーションオブジェクト生成



# 自動生成されたサンプルアプリケーション

```
from chalice import Chalice

app = Chalice(app_name='example')
```

```
@app.route('/')
def index():
    return {'hello': 'world'}
```

- エンドポイントを設定
- /に対するアクセスの処理であることが明白
- パス以外にも様々な設定が可能
- このデコレータへ渡した情報が API Gateway に設定される

# 自動生成されたサンプルアプリケーション

```
from chalice import Chalice

app = Chalice(app_name='example')

@app.route('/')
def index():
    return {'hello': 'world'}
```

- 実際の処理を記述
- レスポンスボディ (明示的)  
-> {"hello": "world"}
- ステータスコード (暗黙的)  
-> 200 OK
- レスポンスヘッダの一例 (暗黙的)  
-> Content-Type: application/json
- "暗黙的"要素も明示的に記述可能

# サンプルアプリケーションをデプロイ

```
$ chalice deploy
Creating deployment package.
...
```

- デプロイパッケージを作成
- IAM ロールを設定
- Lambda Function をデプロイ
- API Gateway を設定

```
$ http $(chalice url)
HTTP/1.1 200 OK
...
```

```
$ chalice delete
Deleting Rest API: *****
...
```

# サンプルアプリケーションをデプロイ

```
$ chalice deploy
Creating deployment package.
...
```

```
$ http $(chalice url)
HTTP/1.1 200 OK
...
```

```
$ chalice delete
Deleting Rest API: *****
...
```

```
Creating deployment package.
Creating IAM role: example-dev
Creating lambda function: example-dev
Creating Rest API
Resources deployed:
  - Lambda ARN: arn:aws:lambda:us-west-2:*****:function:example-dev
  - Rest API URL:
    https://*****.execute-api.us-west-2.amazonaws.com/api/
```

# サンプルアプリケーションをデプロイ

```
$ chalice deploy
```

```
Creating deployment package.
```

```
...
```

```
$ http $(chalice url)
```

```
HTTP/1.1 200 OK
```

```
...
```

```
$ chalice delete
```

```
Deleting Rest API: *****
```

```
...
```

```
HTTP/1.1 200 OK
```

```
Connection: keep-alive
```

```
Content-Length: 17
```

```
Content-Type: application/json
```

```
Date: Sun, 26 May 2019 14:47:45 GMT
```

```
Via: 1.1 *****.cloudfront.net
```

```
(CloudFront)
```

```
X-Amz-Cf-Id: *****
```

```
X-Amzn-Trace-Id: Root=*****;Sampled=0
```

```
X-Cache: Miss from cloudfront
```

```
x-amz-apigw-id: *****
```

```
x-amzn-RequestId: *****
```

```
{
```

```
  "hello": "world"
```

```
}
```

# サンプルアプリケーションをデプロイ

```
$ chalice deploy
```

```
Creating deployment package.
```

```
...
```

```
$ http $(chalice url)
```

```
HTTP/1.1 200 OK
```

```
...
```

```
$ chalice delete
```

```
Deleting Rest API: *****
```

```
...
```

- API Gateway を削除
- Lambda Function を削除
- IAM ロールを削除

# サンプルアプリケーションをデプロイ

```
$ chalice deploy
Creating deployment package.
...
```

```
Deleting Rest API: *****
Deleting function: arn:aws:lambda:us-
west-2:*****:function:example-dev
Deleting IAM role: example-dev
```

```
$ http $(chalice url)
HTTP/1.1 200 OK
...
```

```
$ chalice delete
Deleting Rest API: *****
...
```

# Chalice とは

AWS 上ではどのように構成されるのか



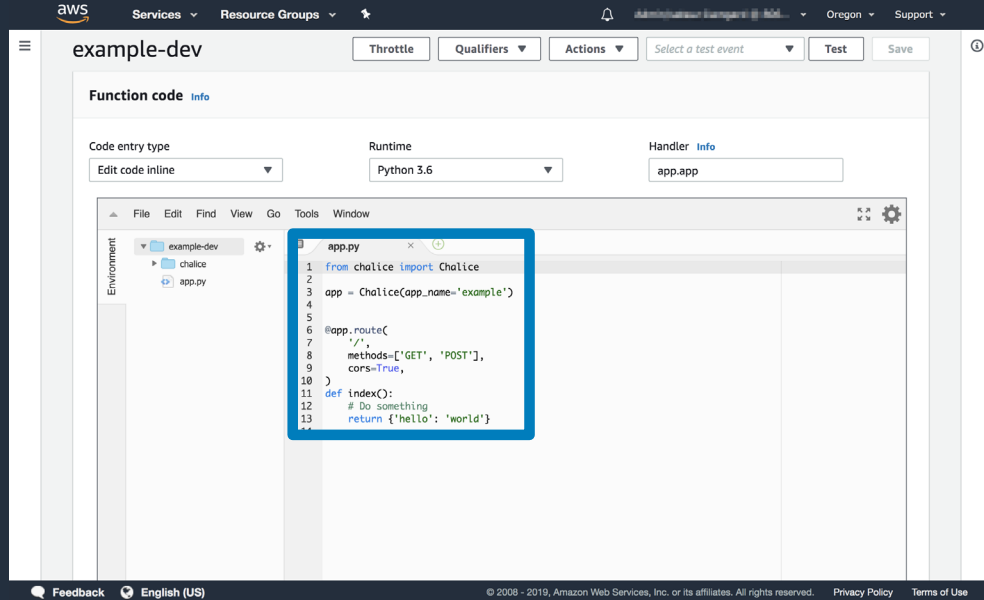
# AWS 上ではどのように構成されるの

か

```
from chalice import Chalice

app = Chalice(app_name='example')

@app.route(
    '/',
    methods=['GET', 'POST'],
    cors=True,
)
def index():
    # Do something
    return {'hello': 'world'}
```



The screenshot shows the AWS Lambda console interface for a function named 'example-dev'. The 'Function code' section is active, displaying the Python code for the function. The code is as follows:

```
1 from chalice import Chalice
2
3 app = Chalice(app_name='example')
4
5
6 @app.route(
7     '/',
8     methods=['GET', 'POST'],
9     cors=True,
10 )
11 def index():
12     # Do something
13     return {'hello': 'world'}
```

The code is displayed in a code editor window titled 'app.py'. The editor shows a file tree on the left with 'example-dev' containing 'chalice' and 'app.py'. The code is highlighted with a blue border. The console also shows the 'Code entry type' as 'Edit code inline', the 'Runtime' as 'Python 3.6', and the 'Handler' as 'app.app'.

# AWS 上ではどのように構成されるの

か

```
from chalice import chalice
```

```
app = chalice(app_name='example')
```

```
@app.route(
```

```
    '/',
```

```
    methods=['GET', 'POST'],
```

```
    cors=True,
```

```
)
```

```
def index():
```

```
    # Do something
```

```
    return {'hello': 'world'}
```

The screenshot displays the AWS Lambda console interface for a function named 'example-dev'. The 'Designer' tab is active, showing a visual representation of the function's configuration. A blue box highlights the 'API Gateway' resource, which is currently disabled. Below this, a detailed view of the API Gateway configuration is shown, including the endpoint URL, authorization method (NONE), and the HTTP method (POST). The console also shows other services like CloudWatch Logs, CodeCommit, and Amazon CloudWatch Logs.

# AWS 上ではどのように構成されるの

か

```
from chalice import chalice
```

```
app = chalice(app_name='example')
```

```
@app.route(
    '/',
    methods=['GET', 'POST'],
    cors=True,
)
```

```
def index():
    # Do something
    return {'hello': 'world'}
```

The screenshot displays the AWS API Gateway console interface. At the top, the navigation bar includes the AWS logo, 'Services', 'Resource Groups', and user information. The main content area shows the configuration for a REST API. The breadcrumb navigation indicates the path: Amazon API Gateway > APIs > example (rtvtnc81aa) > Resources > / (fvdkijpl37). The 'Methods' tab is selected, and a dropdown menu is open, showing the available methods: GET, OPTIONS, and POST. The 'POST' method is highlighted in green. The console shows the configuration for both the 'GET' and 'POST' methods. For both methods, the 'Authorization' is set to 'None' and the 'API Key' is set to 'Not required'. The 'Mock Endpoint' section is also visible, showing 'Authorization: None' and 'API Key: Not required'. At the bottom of the console, there are links for 'Feedback', 'English (US)', and a copyright notice: '© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use'.

# AWS 上ではどのように構成されるの

か

```
from chalice import chalice
```

```
app = chalice(app_name='example')
```

```
@app.route(
```

```
    '/',  
    methods=['GET', 'POST'],  
    cors=True,
```

```
)
```

```
def index():  
    # Do something  
    return {'hello': 'world'}
```

The screenshot shows the AWS API Gateway console interface. The breadcrumb navigation at the top indicates the path: Amazon API Gateway > APIs > example (rtvnc81aa) > Resources > / (fvdkijp37). The main content area is titled '/ Methods' and shows a list of methods for the resource '/'. The methods listed are GET, OPTIONS, and POST. The GET method is selected, and its configuration is displayed in the main panel. The configuration shows 'Authorization: None' and 'API Key: Not required'. The 'Mock Endpoint' section is also visible, showing 'Authorization: None' and 'API Key: Not required'. The POST method is also visible in the list, with its configuration showing 'Authorization: None' and 'API Key: Not required'. The footer of the console shows 'Feedback', 'English (US)', and '© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use'.

# AWS 上ではどのように構成されるの

か

```
from chalice import chalice
```

```
app = chalice(app_name='example')
```

```
@app.route(
```

```
    '/',
```

```
    methods=['GET', 'POST']
```

```
    cors=True,
```

```
)
```

```
def index():
```

```
    # Do something
```

```
    return {'hello': 'world'}
```

The screenshot shows the AWS API Gateway console for a resource named '/'. The resource is configured with two methods: GET and POST. Both methods are configured with 'Authorization: None' and 'API Key: Not required'. The GET method is highlighted in blue, and the POST method is highlighted in green. The console also shows an 'OPTIONS' method and a 'Mock Endpoint' section.

# AWS 上ではどのように構成されるの

か

```
from chalice import chalice
```

```
app = chalice(app_name='example')
```

```
@app.route(
    '/',
    methods=['GET', 'POST'],
    cors=True,
)
```

```
def index():
    # Do something
    return {'hello': 'world'}
```

The screenshot shows the AWS API Gateway console interface. The breadcrumb navigation indicates the path: Amazon API Gateway > APIs > example (rtvnc81aa) > Resources > / (fvdkijp37). The 'Actions' dropdown is open, and the 'Methods' tab is selected. On the left, a list of methods (GET, OPTIONS, POST) is shown for the resource '/'. The main area displays the configuration for the selected method. The 'OPTIONS' method configuration is highlighted with a blue border. It shows 'Authorization: None' and 'API Key: Not required'. Below this, the 'Mock Endpoint' section is also highlighted with a blue border, showing 'Authorization: None' and 'API Key: Not required'. The bottom of the console features a footer with 'Feedback', 'English (US)', and copyright information: '© 2008 - 2019, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use'.

# IAM ポリシーを自動で作成

```
from chalice import Chalice
```

```
app = Chalice(app_name='example')
```

```
@app.route('/')
```

```
def index():  
    return {'hello': 'world'}
```

サンプルアプリケーションから  
自動生成される IAM Policy

The screenshot shows the AWS IAM console interface. The left sidebar contains navigation options: Search IAM, Dashboard, Groups, Users, Roles, Policies, Identity providers, Account settings, Credential report, and Encryption keys. The main content area is titled 'Permissions' and shows 'Permissions policies (1 policy applied)'. A table lists the policy 'example-dev' as an 'Inline policy'. Below the table, the 'Policy summary' section displays the JSON policy document:

```
1 - {  
2   "Version": "2012-10-17",  
3   "Statement": [  
4     {  
5       "Effect": "Allow",  
6       "Action": [  
7         "logs:CreateLogGroup",  
8         "logs:CreateLogStream",  
9         "logs:PutLogEvents"  
10      ],  
11      "Resource": "arn:aws:logs:*:*:*"  
12    }  
13  ]  
14 }
```

# IAM ポリシーを自動で作成

```
import boto3
from chalice import Chalice

app = chalice(app_name='example')
ddb = boto3.client('dynamodb')
```

```
DDB_TABLE_NAME = 'example_table'
```

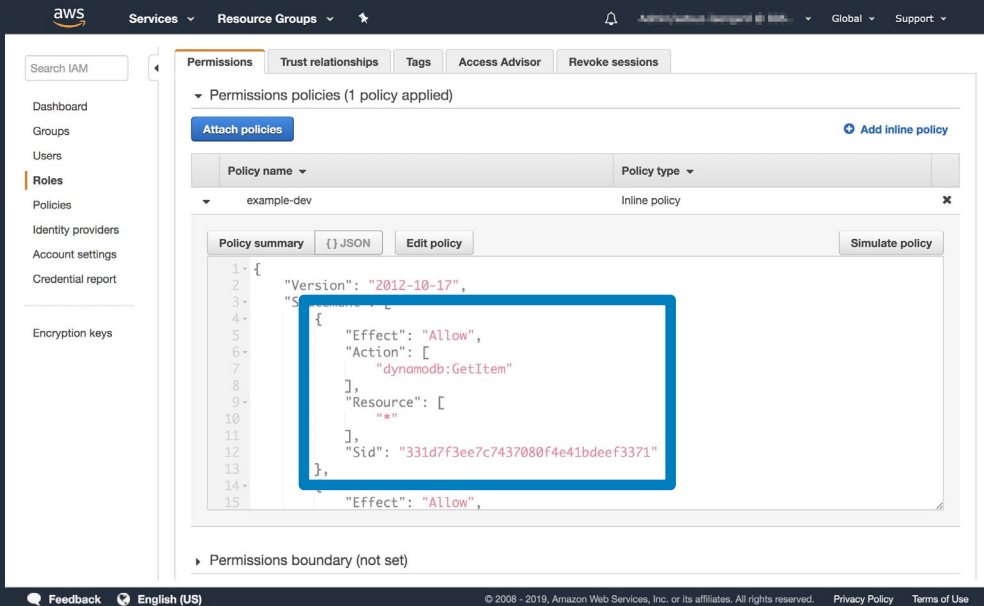
```
@app.route('/')
```

```
def index():
```

```
    item = ddb.get_item(
        TableName=DDB_TABLE_NAME,
        Key={'Id': 1},
    )
```

```
    return item
```

Amazon DynamoDB の GetItem の実装を追加後に自動追加される IAM Policy



The screenshot shows the AWS IAM console interface. On the left, there is a navigation menu with options like Dashboard, Groups, Users, Roles, Policies, Identity providers, Account settings, Credential report, and Encryption keys. The main area is titled 'Permissions' and shows 'Permissions policies (1 policy applied)'. There is a table with columns for 'Policy name' and 'Policy type'. The table contains one entry: 'example-dev' with 'Inline policy'. Below the table, there is a 'Policy summary' section with a blue box highlighting the policy details. The policy is a JSON document with the following structure:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "331d7f3ee7c7437080f4e41bdeef3371"
    }
  ],
  "Effect": "Allow",
}
```

At the bottom of the console, there is a section for 'Permissions boundary (not set)'. The footer of the console includes 'Feedback', 'English (US)', and copyright information for Amazon Web Services, Inc. or its affiliates.



# 現実的な IAM Policy 管理

```
import boto3
from chalice import Chalice

app = Chalice(app_name='example')
ddb = boto3.resource('dynamodb')
table = ddb.Table('example_table')
```

```
@app.route('/')
def index():
    item = table.get_item(
        Key={'Id': 1},
    )
    return item
```

- `boto3.resource()` は非対応
- 3rd Party の定義も検出されない
- 自動生成機能を無効にして自前で `policy.json` を管理することが現実的
- *“The automatic policy generation is still in the early stages, it should be considered experimental. You can always disable policy generation with `--no-autogen-policy` for complete control.”*

\* <https://chalice.readthedocs.io/en/latest/quickstart.html#experimental-status>

# Chalice とは

ここまでのまとめ

# ここまでのまとめ

- シンプルな実装で API Gateway + Lambda の API を定義可能
  - エンドポイントの定義と実際のロジックの関連性がひと目でわかる
    - `@app.route()` デコレータの定義が素直に API Gateway に設定される
    - デコレータが付けられた関数が該当エンドポイントの処理であることが自明
- 簡単デプロイ
  - 自前での `zip` アーカイブのパッケージングが不要
  - ソースコードに実装された通りに連携設定が全自動で行われる
- デプロイした環境はすぐに利用可能
  - 実際に可動する WEB API の構成をマネジメントコンソール上で確認可能

# WEB API 実装のための基本機能

# WEB API 実装のための基本機能

- リクエストハンドリング
  - URL Parameters
  - HTTP Methods
  - Request Metadata
  - Request Content Types
- レスポンスハンドリング
  - Custom HTTP Response
  - Error HTTP Response

# WEB API 実装のための基本機能

## URL Parameters

# URL Parameters

```
from chalice import chalice

app = Chalice(app_name=__name__)

@app.route('/hello/{name}')
def greet(name):
    return {'hello': name}
```

- 可変的な URL パスを定義可能
- パス定義で `{}` で括られた文字列がデコレートされる関数で同名の仮引数にマッピングされる
- 文字列型の変数
- 複数定義も可能

# URL Parameters

```
from chalice import chalice

app = Chalice(app_name=__name__)

@app.route('/hello/{name}')
def greet(name):
    return {'hello': name}
```

- 可変的な URL パスを定義可能
- GET /hello/kuwano  
--> greet('kuwano')
- GET /hello/galaxy  
--> greet('galaxy')



# URL Parameters

```
from chalice import Chalice
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/hello/{name}')
```

```
def greet(name):
```

```
    return {'hello': name}
```

```
$ http $(chalice url)/hello/akihiro
```

```
HTTP/1.1 200 OK
```

```
Connection: keep-alive
```

```
Content-Length: 19
```

```
Content-Type: application/json
```

```
Date: Mon, 27 May 2019 12:12:27 GMT
```

```
Via: 1.1 *****.cloudfront.net (CloudFront)
```

```
X-Amz-Cf-Id: *****
```

```
X-Amzn-Trace-Id: Root=*****;Sampled=0
```

```
X-Cache: Miss from cloudfront
```

```
x-amz-apigw-id: *****
```

```
x-amzn-RequestId: *****
```

```
{
```

```
  "hello": "akihiro"
```

```
}
```

# WEB API 実装のための基本機能

HTTP Methods

# HTTP Methods

```
from chalice import Chalice  
  
app = Chalice(app_name=__name__)
```

```
@app.route('/hello', methods=['POST'])  
def hello():  
    return {'hello': 'world'}
```

- HTTP メソッドを明示してエンドポイントを定義可能

# HTTP Methods

```
from chalice import Chalice
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/hello', methods=['POST', 'PUT'])
```

```
def hello():
```

```
    return {'hello': 'world'}
```

- HTTP メソッドを明示してエンドポイントを定義可能
- 複数の HTTP メソッドをまとめて指定することも可能

# HTTP Methods

```
from chalice import Chalice
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/hello', methods='POST')  
def hello_post():  
    return {'hello': 'world'}
```

```
@app.route('/hello', methods='PUT')  
def hello_put():  
    return {'hello': 'world'}
```

- HTTP メソッドを明示してエンドポイントを定義可能
- 異なる関数として HTTP メソッドごとに同一 URL パスを定義可能
- **POST** /hello --> hello\_post()
- **PUT** /hello --> hello\_put()

# HTTP Methods

```
from chalice import Chalice
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/hello', methods=['POST'])
```

```
def hello_post():  
    return {'hello': 'world'}
```

```
@app.route('/hello', methods=['PUT'])
```

```
def hello_put():  
    return {'hello': 'world'}
```

- HTTP メソッドを明示してエンドポイントを定義可能
- 異なる関数として HTTP メソッドごとに同一 URL パスを定義可能
- POST /hello --> `hello_post()`
- PUT /hello --> `hello_put()`
- 関数に同名を使用することは出来ない

# HTTP Methods

```
from chalice import Chalice
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/hello', methods=['POST', 'PUT'])
def hello_world():
    return {'hello': 'world'}
```

```
@app.route('/hello', methods=['PUT'])
def hello_put():
    return {'hello': 'world'}
```

NOTE: このような定義は不可

- 同じ URL パスを持つ複数の関数で重複する HTTP メソッドを指定することは出来ない
- `hello_world` 関数は `PUT` を受け付ける `/hello` にマッピングされたエンドポイントとして定義済み
- `/hello` に対する `PUT` を処理する `hello_put` 関数は定義不可

# WEB API 実装のための基本機能

## Request Metadata



# Request Metadata

## 実装内で取り扱うことができるリクエストメタデータ

- HTTP Method
- HTTP Headers
- Query Parameters
- URI Parameters
- Request Body
  - Raw
  - JSON
- Additional Request Context
- API Gateway Stage

# Request Metadata - HTTP Method

```
from chalice import Chalice
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/hello', methods=['POST', 'PUT'])
```

```
def hello():
```

```
    if app.current_request.method == 'POST':  
        return {'hello': 'POST world'}
```

```
    if app.current_request.method == 'PUT':  
        return {'hello': 'PUT world'}
```

- `app.current_request.method` に文字列として格納される
- 関数内で HTTP メソッドによって処理を分岐させる場合に有用

# Request Metadata - HTTP Method

```
from chalice import Chalice

app = Chalice(app_name=__name__)

@app.route('/hello', methods=['POST', 'PUT'])
def hello():
    if app.current_request.method == 'POST':
        return {'hello': 'POST world'}

    if app.current_request.method == 'PUT':
        return {'hello': 'PUT world'}
```

```
$ http POST $(chalice url)/hello
HTTP/1.1 200 OK
...
{
  "hello": "POST world"
}
```

# Request Metadata - HTTP Method

```
from chalice import Chalice

app = Chalice(app_name=__name__)

@app.route('/hello', methods=['POST', 'PUT'])
def hello():
    if app.current_request.method == 'POST':
        return {'hello': 'POST world'}

    if app.current_request.method == 'PUT':
        return {'hello': 'PUT world'}
```

```
$ http PUT $(chalice url)/hello
HTTP/1.1 200 OK
...
{
  "hello": "PUT world"
}
```

# Request Metadata - HTTP Headers

```
from chalice import Chalice, Response
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/')
```

```
def index ():
```

```
    headers = app.current_request.headers
```

```
    # Do something...
```

- `app.current_request.headers` に `chalice.app.CaseInsensitiveMapping` として格納される
- `collections.abc.Mapping` を継承して独自拡張したクラス
- `dict` 互換
- 各キーに対して大文字小文字の区別なしにアクセス可能
- 参照のみ

# Request Metadata - HTTP Headers

```
from chalice import Chalice, Response
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/')
```

```
def index():
```

```
    headers = app.current_request.headers
```

```
    # Do something...
```

オブジェクトをダンプすると

```
CaseInsensitiveMapping({'accept': '*/*',  
'accept-encoding': 'gzip, deflate',  
'cloudfront-forwarded-proto': 'https',  
'cloudfront-is-desktop-viewer': 'true',  
'cloudfront-is-mobile-viewer': 'false',  
'cloudfront-is-smarttv-viewer': 'false',  
'cloudfront-is-tablet-viewer': 'false',  
'cloudfront-viewer-country': 'JP', 'host':  
'*****.execute-api.us-west-  
2.amazonaws.com', 'user-agent':  
'HTTPie/1.0.2', 'via': '1.1  
*****.cloudfront.net (CloudFront)',  
'x-amz-cf-id': '*****', 'x-amzn-trace-  
id': 'Root=*****', 'x-forwarded-for':  
'27.0.***.***, 52.46.***.***', 'x-  
forwarded-port': '443', 'x-forwarded-  
proto': 'https'})
```

# Request Metadata - Query Parameters

```
from chalice import Chalice, Response
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/')
```

```
def index():
```

```
    queries = app.current_request.query_params
```

```
    # Do something
```

- `app.current_request.query_params` に `dict` として格納される
- GET `/?hoge=123`  
--> `{'hoge': 123}`
- GET `/?foo=123&bar=123&bar=456`  
--> `{'foo': 123, 'bar': 456}`

# Request Metadata - URI Parameters

```
from chalice import Chalice, Response
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/hello/{name}')
```

```
def greet(name):
```

```
    params = app.current_request.uri_params
```

```
    # Do something
```

- `app.current_request.uri_params` に `dict` として格納される
- GET /hello/kuwano  
--> {'name': 'kuwano'}
- `name` 仮引数をそのまま扱えばよいので一見すると無意味そうだが



# Request Metadata - URI Param

EXAMPLE

```
import boto3
from chalice import Chalice, Response

import functools

app = chalice(app_name=__name__)
ddb = boto3.resource('dynamodb')
user_table = ddb.Table('User')
```

```
def retrieve_user(f):
    @functools.wraps(f)
    def wrapper(*args, **kwargs):
        user_id = kwargs.pop('user_id')
        user = user_table.get_item(key={'Id': user_id})
        kwargs['user'] = user
        return f(*args, **kwargs)
    return wrapper
```

```
@app.route('/user/{user_id}')
```

```
@retrieve_user
```

```
def do_something(user):
    uri_params = app.current_request.uri_params
    # Do something
```

- まったく同じ処理を必要とするエンドポイントが複数ある場合に各関数で同じ処理を書くことは視認性やメンテナンス性に欠く
- そのような場合にはデコレータを実装して処理を一箇所にまとめるテクニックを用いることがある

# Request Metadata - URI Parameters

```
import boto3
from chalice import Chalice, Response

import functools

app = chalice(app_name=__name__)
ddb = boto3.resource('dynamodb')
user_table = ddb.Table('User')

def retrieve_user(f):
    @functools.wraps(f)
    def wrapper(*args, **kwargs):
        user_id = kwargs.pop('user_id')
        user = user_table.get_item(Key={'Id': user_id})
        kwargs['user'] = user
        return f(*args, **kwargs)
    return wrapper

@app.route('/user/{user_id}')
@retrieve_user
def do_something(user):
    uri_params = app.current_request.uri_params
    # Do something
```

- デコレータは `user_id` 変数を後続の処理に渡していない
- `do_something` 関数内でリクエスト時 `user_id` に何が格納されていたかを確認する必要がある場合に有用

# Request Metadata - URI Param

NOTE

```
import boto3
from chalice import Chalice, Response

import functools

app = chalice(app_name=__name__)
ddb = boto3.resource('dynamodb')
user_table = ddb.Table('User')
```

```
def retrieve_user(f):
    @functools.wraps(f)
    def wrapper(*args, **kwargs):
        user_id = kwargs.pop('user_id')
        user = user_table.get_item(Key={'Id': user_id})
        kwargs['user'] = user
        return f(*args, **kwargs)
    return wrapper
```

```
@app.route('/user/{user_id}')
@retrieve_user
def do_something(user):
    uri_params = app.current_request.uri_params
    # Do something
```

例示では2行だけの共通化だが  
実際のアプリケーションでは他にも  
様々な処理をさせる必要がある

たとえば

- ユーザが存在しなかったら 404
  - DDB アクセスのエラーハンドリング
  - ロギング
- などなど

# Request Metadata - Request Body

```
from chalice import Chalice, Response
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/', methods=['POST'])
```

```
def index():
```

```
    raw_body = app.current_request.raw_body
```

```
    json_body = app.current_request.json_body
```

```
    # Do something...
```

- `app.current_request.raw_body` に bytes として格納される

```
$ curl -i -X POST $(chalice url) \  
  -H 'Content-Type: application/json' \  
  --data '{"hello": "world"}'
```

```
--> b'{"hello": "world"}'
```

# Request Metadata - Request Body

```
from chalice import Chalice, Response
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/', methods=['POST'])
```

```
def index ():
```

```
    raw_body = app.current_request.raw_body
```

```
    json_body = app.current_request.json_body
```

```
    # Do something...
```

- Content-Type: application/json なら
- `app.current_request.json_body` に `dict` として格納される
- `== json.dumps(raw_body)`

```
$ curl -i -X POST $(chalice url) \
  -H 'Content-Type: application/json' \
  --data '{"hello": "world"}
```

```
--> {"hello": "world"}
```

# Request Metadata - Additional Request

## Context

```
from chalice import Chalice, Response
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/')
```

```
def index ():
```

```
    context = app.current_request.context
```

```
    # Do something...
```

- `app.current_request.context` に `dict` として格納される

```
{'resourceId': 'eyfxof', 'resourcePath': '/qs',  
'httpMethod': 'GET', 'extendedRequestId':  
'ae3uHEgyvHcFTXg=', 'requestTime':  
'30/May/2019:05:40:16 +0000', 'path': '/api//qs',  
'accountId': '*****', 'protocol': 'HTTP/1.1', 'stage':  
'api', 'domainPrefix': '*****', 'requestTimeEpoch':  
1559194816732, 'requestId': '675fd24c-829d-11e9-  
a474-b736a1da8816', 'identity':  
{'cognitoIdentityPoolId': None, 'accountId': None,  
'cognitoIdentityId': None, 'caller': None, 'sourceIp':  
'27.0.3.146', 'principalOrgId': None, 'accessKey':  
None, 'cognitoAuthenticationType': None,  
'cognitoAuthenticationProvider': None, 'userArn':  
None, 'userAgent': 'HTTPIe/1.0.2', 'user': None},  
'domainName': '*****.execute-api.us-west-  
2.amazonaws.com', 'apiId': '*****'}
```

# WEB API 実装のための基本機能

Custom HTTP Response

# Custom HTTP Response

```
from chalice import chalice

app = Chalice(app_name='example')

@app.route('/')
def index():
    return {'hello': 'world'}
```

200 OK で JSON レスポンスを返す  
ために必要な実装のおさらい

- JSON serializable オブジェクトを  
*return* するだけ



# Custom HTTP Response

現実的にアプリケーション実装に必要なもの/こと

- 任意のレスポンスステータスコード
- 任意のレスポンスヘッダ
- JSON 以外のレスポンスコンテンツ

# Custom HTTP Response

```
from chalice import Chalice, Response
import json
```

```
app = chalice(app_name=__name__)
```

```
@app.route('/')
def index():
```

```
    return Response(
        body=json.dumps({'hello': 'world'}),
        headers={'Content-Type': 'application/json'},
        status_code=200,
    )
```

レスポンスをカスタマイズするには

- *Response* クラスをインポート
- 必要な情報を含めた *Response* オブジェクトを返す

※ このサンプルは Hello, world のサンプルと同じ結果を返す

# Custom HTTP Response

```
from chalice import Chalice, Response
import json

app = chalice(app_name=__name__)

@app.route('/put_item', methods=['PUT'])
def put_item():
    return Response(
        body=json.dumps({'Message': 'Accepted'}),
        headers={'Content-Type': 'application/json'},
        status_code=202,
    )
```

任意のレスポンスステータスコード

- 任意のステータスコードで *Response* オブジェクトを返す

# Custom HTTP Response

```
from chalice import Chalice, Response
```

```
app = chalice(app_name=__name__)
```

```
@app.route('/')
```

```
def index():
```

```
    return Response(  
        body='Hello, world!',  
        headers={'Content-Type': 'text/plain'},  
        status_code=200,  
    )
```

```
        body='Hello, world!',  
        headers={'Content-Type': 'text/plain'},  
        status_code=200,
```

## JSON 以外のレスポンスコンテンツ

- 任意のオブジェクトを *body* に指定
- 適切な Content-Type を指定

# WEB API 実装のための基本機能

Error HTTP Response

# Error HTTP Response

```
from chalice import (  
    Chalice,  
    ForbiddenError,  
)
```

```
app = Chalice(app_name=__name__)
```

```
@app.route('/forbidden')
```

```
def forbidden():
```

```
    raise ForbiddenError(  
        'You are NOT allowed to '  
        'access here!'  
    )
```

- 任意のステータスコードに対応する例外クラスで例外を上げる
- この例では 403 Forbidden
- エラーメッセージは任意文字列

# Error HTTP Response

## バンドルされているエラーレスポンスクラス

- 400 - BadRequestError
- 401 - UnauthorizedError
- 403 - ForbiddenError
- 404 - NotFoundError
- 409 - ConflictError
- 429 - TooManyRequestsError
- 500 - ChaliceViewError

<https://chalice.readthedocs.io/en/latest/quickstart.html#tutorial-error-messages>

# Error HTTP Response

```
from chalice import Chalice, Response
import json

app = chalice(app_name=__name__)

@app.route('/make_coffee')
def make_coffee():
    return Response(
        body=json.dumps({
            'Code': "I'm a teapot",
            'Message': 'Tip me over and pour me out',
        }),
        headers={'Content-Type': 'application/json'},
        status_code=418,
    )
```

例外クラスが実装されていない  
ステータスコードでレスポンスする

- 任意のステータスコードで  
*Response* オブジェクトを返す



# Error HTTP Response

```
from chalice import Chalice, Response
```

```
app = chalice(app_name=__name__)
```

```
@app.route('/forbidden')
def forbidden():
    return Response(
        body= 'You are NOT allowed to access here!',
        headers={'Content-Type': 'text/plain'},
        status_code=403,
    )
```

例外クラスが実装されている  
ステータスコードでレスポンスする

- レスポンスボディに含む  
エラーメッセージは任意の構造
- JSON 以外も許容

# 一歩進んだ API 実装のための機能

# 一歩進んだ API 実装のための機能

- 認証、認可
  - IAM, Cognito との連携が容易
  - 独自認証実装との連携も
- CORS (Cross Origin Resource Sharing) サポート
- GZIP 圧縮によるパフォーマンス効率的なレスポンス
- Blueprints

# Amazon API Gateway 以外と連携 する **Lambda** 関数

# その他の Lambda Functions

- 他サービスと連携しない Lambda Functions
- Lambda イベントソース
  - Amazon CloudWatch Events
  - Amazon S3
  - Amazon SQS
  - Amazon SNS

# その他の Lambda Functions - SQS

```
from chalice import Chalice  
  
app = Chalice(app_name=__name__)
```

```
@app.on_sqs_message(queue='exampleQueue')  
def handle_sqs_message(event):  
    for record in event:  
        # Do something...
```

- `on_sqs_message` デコレータで SQS と連携する関数であることを指定
- SQS と連携するための設定は自動で

さらにもう一歩先に

# さらにもう一步先に

- Python Versions
- App Packaging
- Logging
- SDK Generation
- Chalice Stages
- Configuration File
- Policy Generation
- Local Mode
- Lambda Layers
- Experimental APIs
- WebSocket Support (Coming soon?)



# 実践的開発プロセス

# 実践的開発プロセス

- ユニットテスト
- 継続的インテグレーション、継続的デプロイメント

# 実践的開発プロセス

## ユニットテスト

# ユニットテスト?

- 単体テストとも呼ばれる

## 特長\*

- 早期に問題を発見する
- 変更を容易にする
- 統合の簡素化
- ドキュメント
- 設計



\* <https://ja.wikipedia.org/wiki/%E5%8D%98%E4%BD%93%E3%83%86%E3%82%B9%E3%83%88>

# ユニットテスト

```
$ chalice new-project example && cd example  
$ mkdir tests && cd tests  
$ touch __init__.py conftest.py test_app.py  
$ pip install pytest-chalice  
$ echo pytest-chalice > ../test_requirements.txt  
$ tree -ac ..
```

```
..  
├── .chalice  
│   └── config.json  
├── .gitignore  
├── app.py  
├── requirements.txt  
├── test_requirements.txt  
└── tests  
    ├── __init__.py  
    ├── conftest.py  
    └── test_app.py
```

2 directories, 8 files  
© 2019, Amazon Web Services, Inc. or its Affiliates.

スケルトンプロジェクトを対象にして  
ユニットテストを書く

- *new-project* サブコマンドによってプロジェクトの雛形を作成
- 任意のプロジェクト名を引数に
- *tests* ディレクトリを作成

# ユニットテスト

```
$ chalice new-project example && cd example  
$ mkdir tests && cd tests  
$ touch __init__.py conftest.py test_app.py  
$ pip install pytest-chalice  
$ echo pytest-chalice > ../test_requirements.txt  
$ tree -aC ..
```

```
..  
├── .chalice  
│   └── config.json  
├── .gitignore  
├── app.py  
├── requirements.txt  
├── test_requirements.txt  
└── tests  
    ├── __init__.py  
    ├── conftest.py  
    └── test_app.py
```

2 directories, 8 files

© 2019, Amazon Web Services, Inc. or its Affiliates.

スケルトンプロジェクトを対象にして  
ユニットテストを書く

- `__init__.py` を設置
- `test_app.py` を設置
- `conftest.py` を設置

# ユニットテスト

```
$ chalice new-project example && cd example  
$ mkdir tests && cd tests  
$ touch __init__.py conftest.py test_app.py  
$ pip install pytest-chalice
```

```
$ echo pytest-chalice > ../test_requirements.txt
```

```
$ tree -ac ..  
..  
├── .chalice  
│   └── config.json  
├── .gitignore  
├── app.py  
├── requirements.txt  
├── test_requirements.txt  
└── tests  
    ├── __init__.py  
    ├── conftest.py  
    └── test_app.py
```

スケルトンプロジェクトを対象にして  
ユニットテストを書く

- *pytest-chalice* をインストール
- *test\_requirements.txt* を設置

2 directories, 8 files

© 2019, Amazon Web Services, Inc. or its Affiliates.



# ユニットテスト

```
tests/conftest.py
```

```
import pytest

from app import app as chalice_app

@pytest.fixture
def app():
    return chalice_app
```

スケルトンプロジェクトを対象にして  
ユニットテストを書く

- *py.test* のテストのコンテキスト内で  
実際のロジックにアクセスするため



# ユニットテスト

`app.py`

```
from chalice import Chalice

app = Chalice(app_name='example')
```

```
@app.route('/')
def index():
    return {'hello': 'world'}
```

テストを検討

- /に対してのリクエストが成功する
- ステータスコードが 200 OK
- レスポンスボディが {"hello": "world"}

# ユニットテスト

`app.py`

```
from chalice import Chalice

app = Chalice(app_name='example')
```

```
@app.route('/')
def index():
    return {'hello': 'world'}
```

## テストを検討

- /に対してのリクエストが成功する
- ステータスコードが 200 OK
- レスポンスボディが `{"hello": "world"}`

# ユニットテスト

`tests/test_app.py`

```
from http import HTTPStatus
```

```
def test_index(client):  
    response = client.get('/')  
    assert response.status_code \\  
        == HTTPStatus.OK  
    assert response.json \\  
        == {'hello': 'world'}
```

テストを**実装**

- /に対してのリクエストが成功する
- ステータスコードが 200 OK
- レスポンスボディが `{"hello": "world"}`

# ユニットテスト

```
tests/test_app.py
```

```
from http import HTTPStatus
```

```
def test_index(client):  
    response = client.get('/')  
    assert response.status_code \  
        == HTTPStatus.OK  
    assert response.json \  
        == {'hello': 'world'}
```

テストを**実装**

- /に対してのリクエストが成功する
- ステータスコードが 200 OK
- レスポンスボディが {"hello": "world"}

# ユニットテスト

```
tests/test_app.py
```

```
from http import HTTPStatus
```

```
def test_index(client):  
    response = client.get('/')  
    assert response.status_code \  
        == HTTPStatus.OK  
    assert response.json \  
        == {'hello': 'world'}
```

テストを**実装**

- /に対してのリクエストが成功する
- ステータスコードが 200 OK
- レスポンスボディが {"hello": "world"}

# ユニットテスト

```
$ pytest -vv
===== test session starts =====
platform darwin -- Python 3.6.8, pytest-4.4.0,
py-1.8.0, pluggy-0.9.0 --
/Users/satsuz/.pyenv/versions/3.6.8/bin/python3.6
cachedir: .pytest_cache
hypothesis profile 'default' ->
database=DirectoryBasedExampleDatabase('/private/
tmp/example/tests/.hypothesis/examples')
rootdir: /private/tmp/example/tests
plugins: ordering-0.6, chalice-0.0.4, hypothesis-
4.14.2
collected 1 item
```

```
test_app.py::test_index PASSED [100%]
```

```
===== 1 passed in 0.01 seconds =====
```

## テストを**実行**

- *pytest* コマンドを実行
- テスト関数 *test\_index* が成功している

# 実践的開発プロセス

CI/CD

# CI/CD?

Source

Test

Package

Deploy

継続的インテグレーション (CI)

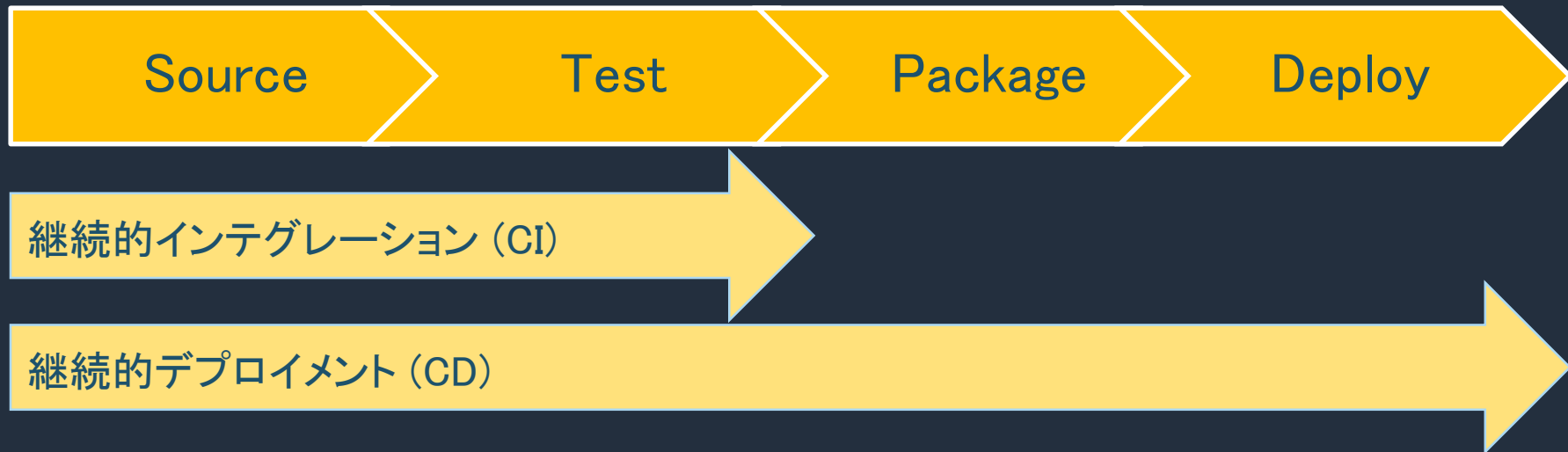
継続的デプロイメント (CD)



<https://aws.amazon.com/jp/devops/continuous-delivery/>

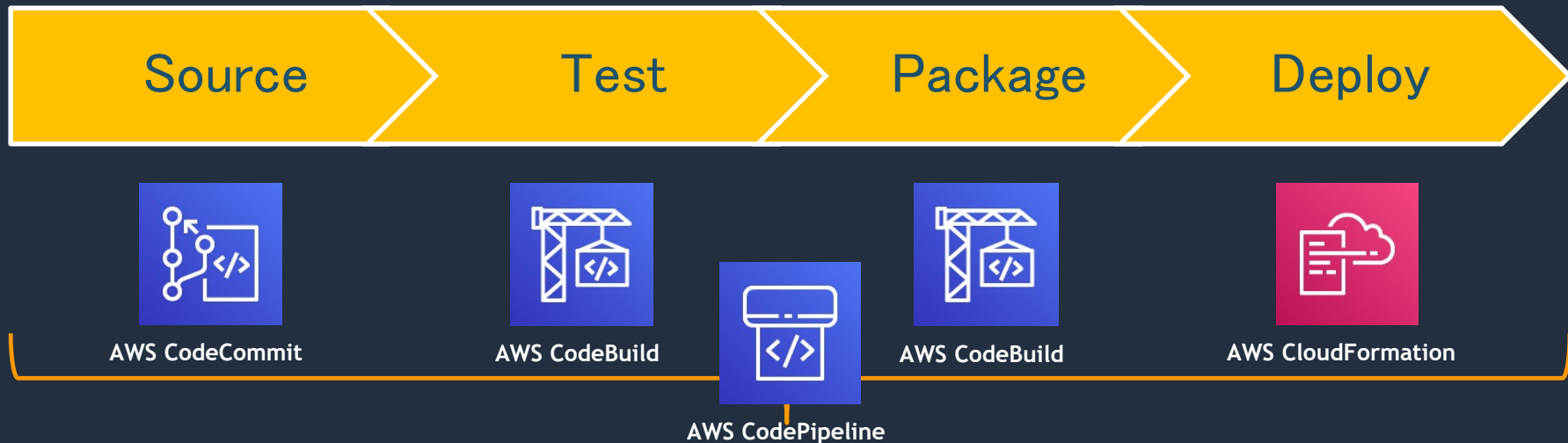


# Chalice のための CI/CD 例



- ロジック実装
- テスト実装
- ブランチ管理
  - ローカル
  - リモート
- プルリクエスト
- ピアレビュー
- py.test でユニットテスト
- Flake8 で Lint チェック
- mypy で静的型チェック
- デプロイパッケージ作成
- S3 にアップロード
- AWS CloudFormation
  - ChangeSet 作成
  - ChangeSet 実行

# Chalice のための CI/CD 例



- ロジック実装
- テスト実装
- ブランチ管理
  - ローカル
  - リモート
- プルリクエスト
- ピアレビュー

- `py.test` でユニットテスト
- Flake8 で Lint チェック
- mypy で静的型チェック

- デプロイパッケージ作成
- S3 にアップロード

- AWS CloudFormation
  - ChangeSet 作成
  - ChangeSet 実行

# *chalice deploy? chalice package?*

## *chalice deploy*

- 開発したアプリケーションを即座にデプロイ出来るため非常に便利
- ただしチーム開発には適さない
  - *deployed.json* を共有することは現実的ではない
  - いつでもだれでもどこからでもデプロイ出来てはならない

# *chalice deploy? chalice package?*

## *chalice package*

- アプリケーションをデプロイするためのアーティファクトを生成するコマンド
  - Serverless Application Model (SAM) の JSON ファイル
  - 依存ライブラリを含めたアプリケーションの zip アーカイブ

# Chalice のための CI/CD テンプレート

```
$ chalice generate-pipeline \  
-b buildspec.yml \  
pipeline.json
```

```
$ aws cloudformation deploy \  
--stack-name exampleAppPipeline \  
--template-file pipeline.json \  
--capabilities CAPABILITY_IAM
```

- *generate-pipeline* サブコマンドで以下のような構成を AWS 上に作成するための CFn template を生成可能
- *-b* オプションで *buildspec.yml* を単独ファイルとして出力させられる

\*

<https://chalice.readthedocs.io/en/latest/topics/cd.html#extending>

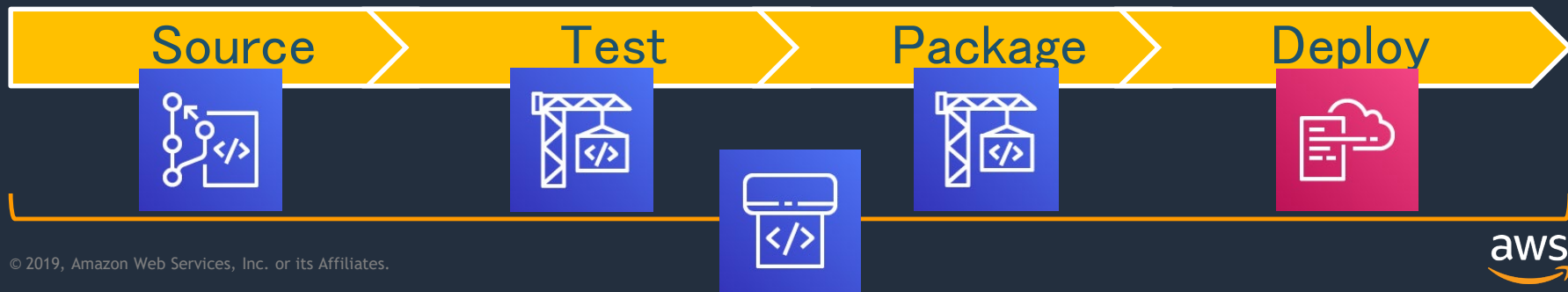


# Chalice のための CI/CD テンプレート

```
$ chalice generate-pipeline \  
  -b buildspec.yml \  
  pipeline.json
```

```
$ aws cloudformation deploy \  
  --stack-name exampleAppPipeline \  
  --template-file pipeline.json \  
  --capabilities CAPABILITY_IAM
```

- 生成された CFn template を適用
- 各リソースが生成される
  - AWS CodePipeline Pipeline
  - AWS CodeCommit Repo
  - AWS CodeBuild Project
  - Amazon S3 Bucket

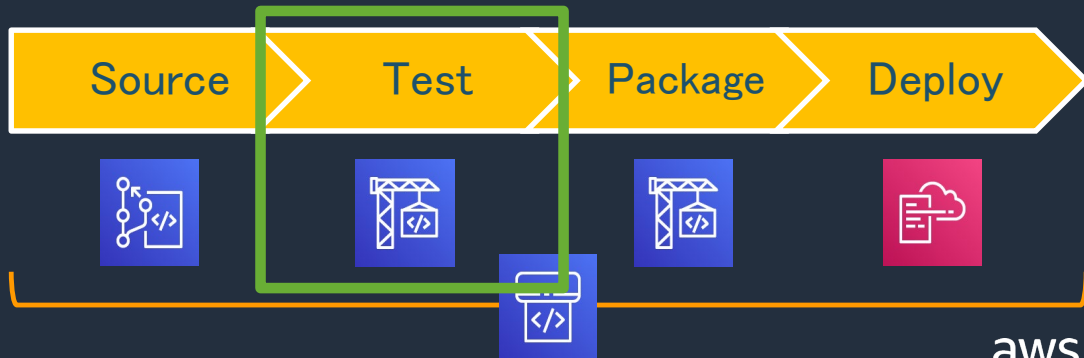


# buildspec.yml を編集してみよう

```
version: 0.1
phases:
  install:
    commands:
      - pip install -U awscli pip
      - aws --version
  pre_build:
    commands:
      - pip install -r requirements.txt
      - pip install -r test_requirements.txt
  build:
    commands:
      - flake8
      - py.test -vv
  post_build:
    commands:
      - pip install chalice
      - chalice package /tmp/package
      - aws cloudformation package \
        --template-file /tmp/package/sam.json \
        --s3-bucket ${APP_S3_BUCKET} \
        --output-template-file transformed.yaml
artifacts:
  type: zip
  files:
    - transformed.yaml
```

*buildspec.yml* を書き換えてユニットテストもさせる

- 依存ライブラリをインストール
  - *requirements.txt*
  - *test\_requirements.txt*

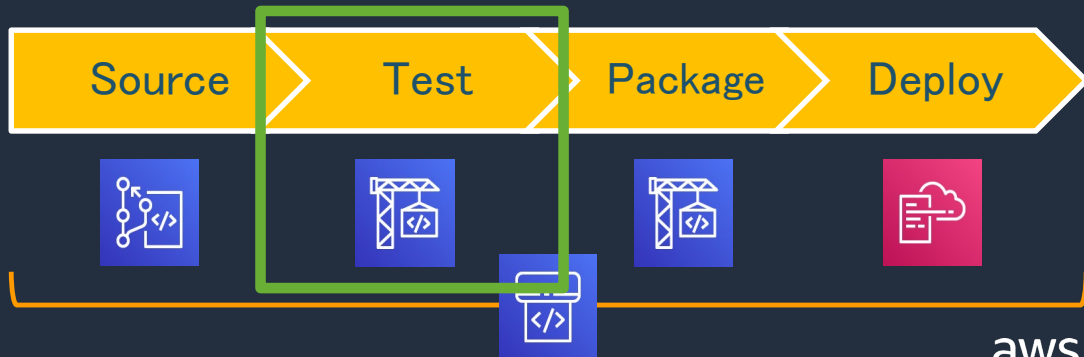


# buildspec.yml を編集してみよう

```
version: 0.1
phases:
  install:
    commands:
      - pip install -U awscli pip
      - aws --version
  pre_build:
    commands:
      - pip install -r requirements.txt
      - pip install -r test_requirements.txt
  build:
    commands:
      - flake8
      - py.test -vv
  post_build:
    commands:
      - pip install chalice
      - chalice package /tmp/packaged
      - aws cloudformation package \
        --template-file /tmp/packaged/sam.json \
        --s3-bucket ${APP_S3_BUCKET} \
        --output-template-file transformed.yaml
artifacts:
  type: zip
  files:
    - transformed.yaml
```

*buildspec.yml* を書き換えてユニットテストもさせる

- *flake8* コマンドで Lint チェック
- *py.test* コマンドでユニットテスト実行



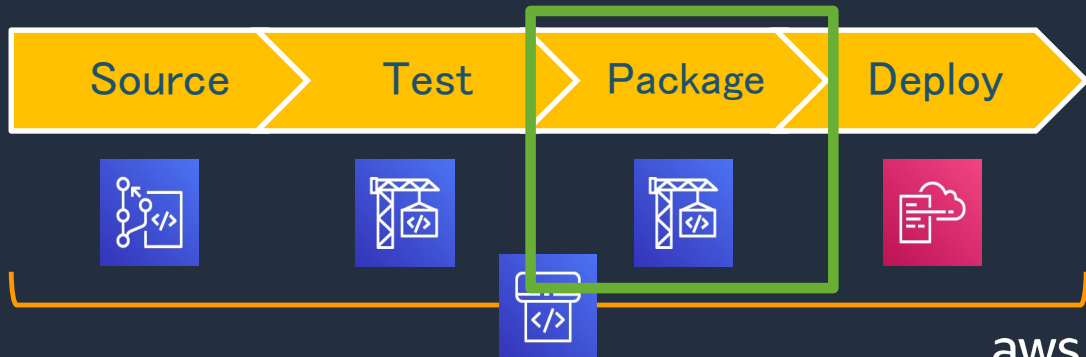


# buildspec.yml を編集してみよう

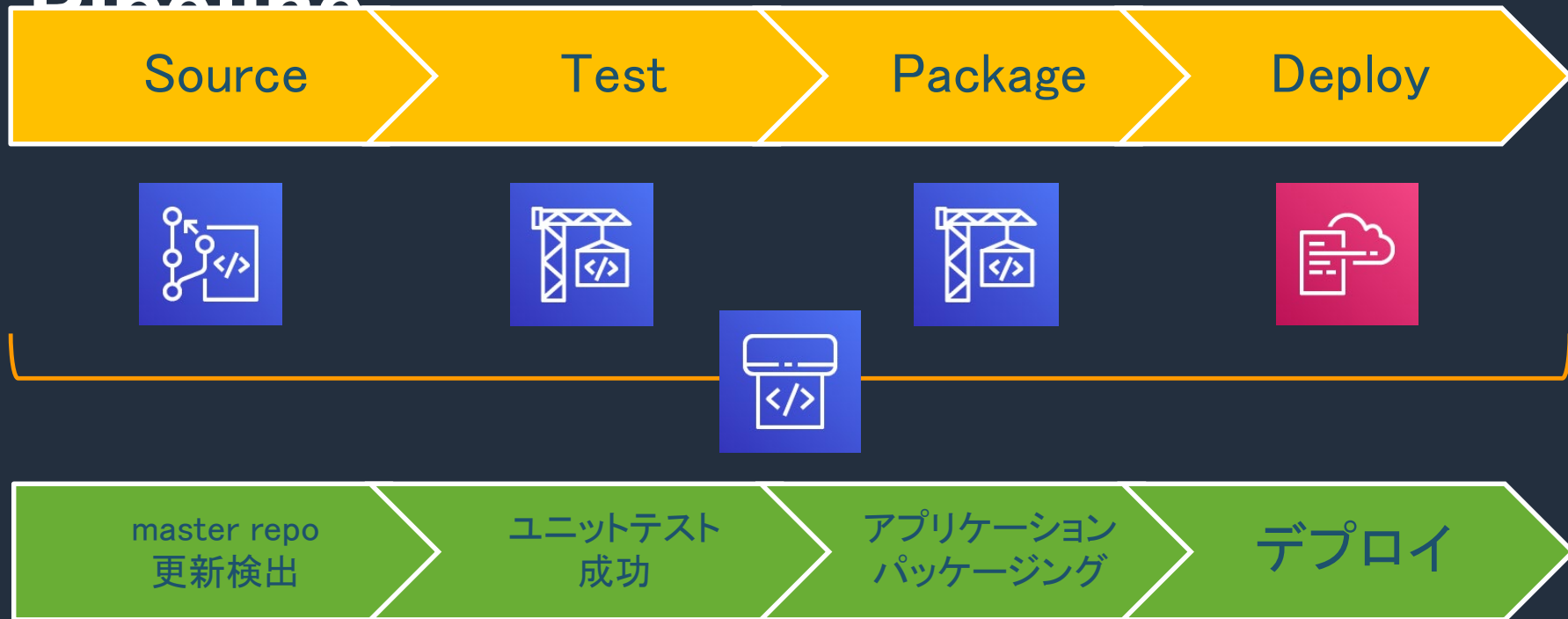
```
version: 0.1
phases:
  install:
    commands:
      - pip install -U awscli pip
      - aws --version
  pre_build:
    commands:
      - pip install -r requirements.txt
      - pip install -r test_requirements.txt
  build:
    commands:
      - flake8
      - py.test -vv
  post_build:
    commands:
      - pip install chalice
      - chalice package /tmp/packaged
      - aws cloudformation package \
        --template-file /tmp/packaged/sam.json \
        --s3-bucket ${APP_S3_BUCKET} \
        --output-template-file transformed.yaml
artifacts:
  type: zip
  files:
    - transformed.yaml
```

Build フェーズはデフォルトと同じ

- *chalice package* コマンド deploy artifacts 生成
- S3 にアップロード
- *sam.json* を *transformed.yml* に変換



# Continuous Deployment via the Pipeline



**That's it!**  
**Enjoy the Serverless World with**  
**Chalice!**

# Q&A

お答えできなかったご質問については

AWS Japan Blog 「<https://aws.amazon.com/jp/blogs/news/>」にて  
後日掲載します。

# AWS の日本語資料の場所「AWS 資料」で検索



日本担当チームへお問い合わせ サポート 日本語 ▾ アカウント ▾

コンソールにサインイン

製品 ソリューション 料金 ドキュメント 学習 パートナー AWS Marketplace その他 🔍

## AWS クラウドサービス活用資料集トップ

アマゾン ウェブ サービス (AWS) は安全なクラウドサービスプラットフォームで、ビジネスのスケールと成長をサポートする処理能力、データベースストレージ、およびその他多種多様な機能を提供します。お客様は必要なサービスを選択し、必要な分だけご利用いただけます。それらを活用するために役立つ日本語資料、動画コンテンツを多数ご提供しております。(本サイトは主に、AWS Webinar で使用した資料およびオンデマンドセミナー情報を掲載しています。)

[AWS Webinar お申込 »](#)

[AWS 初心者向け »](#)

[業種・ソリューション別資料 »](#)

[サービス別資料 »](#)

<https://amzn.to/JPArchive>



# ご視聴ありがとうございました

AWS 公式 Webinar

<https://amzn.to/JPWebinar>



過去資料

<https://amzn.to/JPArchive>

