BETTER TOGETHER WITH intel

aws DEV DAY

JAPAN | SEPTEMBER 28, 2021

A - 3

AWS Chalice 再入門 ～RESTful APIはやっぱり最高～

Koya Kimura (@kimyan_udon2)
アマゾン ウェブ サービス ジャパン株式会社
2021/09/28

木村 公哉（きむら こうや）

- 所属

  アマゾン ウェブ サービス ジャパン（株）

  技術統括本部 ISV/SaaSソリューション本部

  ソリューションアーキテクト

- 好きなAWSサービス

AWS Amplify

AWS Lambda

Amazon Kinesis

想定視聴者とゴール

- 想定視聴者
  - これからWebアプリを作ってみたい！と**意気込んでいる方**
  - Webアプリを作ろうと思ったが、**何から始めればいいのだろうと悩んでいる方**
- ゴール
  - RESTful APIとAWS Chaliceの関係性を知って「**お、すぐにAPI作れそう！**」と感じる
  - この講演を参考にしながら手を動かすことで、**Webアプリ作成の第一歩を踏み出す**

はじめに

みなさん、Web API作ってますか！

API : Application Programing Interface

- APIがあるとき・ないときってやろうと思ったけど、
  もはや何かを実行する時の入り口として当たり前になっている
  - プログラミング言語のメソッドなんかもAPI
- 今回はWebアプリケーション向けのAPI「**Web API**」にフォーカスする
- とりあえずAPIを作っておけば、フロントエンドから呼び出して
  よしなにしやすい（諸説あります）
- **千里の道もWeb APIから**ですね

Web APIの実装スタイルはいろいろある

- REST、GraphQL（、gRPC、SOAP）
- どれを使えばいいんだろう？やっぱり流行のやつ？

好きなのでいい！

むしろ最初はよく使われている方法を
使ったほうが資料がたくさんあっていい

Chaliceの前にRESTのおさらい

- Web APIを作るときに用いられる**アーキテクチャースタイル**
- 2000年、Roy Fieldingが博士論文で定義した
- RESTの原則に従って実装されているAPIを**RESTful API**と呼ぶ
- RESTの主な特徴
  - **単一HTTPメッセージで1つの操作**に関する情報を（理想では）含む
  - 扱う情報をURIで表現する「**リソース**」として定義し、それらを**HTTPメソッド**（PUT, GET, POST, DELETE, …）の表現で操作

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

RESTのことが好きな理由（個人の意見）

- いろんなところで見ていて**親しみがある**
  - AWSのAPI、Slack API、などなど
- 単一のHTTPメッセージでやりとりするので**分かりやすい**
- サクッと作るには、**自分にとって分かりやすいか、手になじむかどうか**が重要
  - 組織で技術選定をするときはこの「自分」が「みんな」になる

自分にとっていいものを見つけよう！

AWS Chaliceとは

- **Python**製サーバーレスアプリケーションフレームワーク
- AWS Lambdaを用いて、アプリケーションを**簡単に作成、デプロイ**
- Chaliceで提供される機能
  - アプリの作成、デプロイ、管理ができる**コマンドラインツール**
  - Amazon API Gateway、Amazon S3、Amazon SNS、Amazon SQS、そのほかAWSサービスと統合するための**デコレーターベースのAPI**

Chaliceのいいところ

- めちゃくちゃ**サクッ**とRESTful APIを作ることができる
  - これに尽きる！

百聞は一見にしかず！
とりあえずChaliceでAPIを
作ってみよう！

とりあえずChaliceでAPI作ってみる

AWS Chaliceの始め方 by 公式ドキュメント

まずはPython3系が入っているかを確認して、仮想環境を作成

```
$ python3 --version
Python 3.7.3
$ python3 -m venv venv37
$ . venv37/bin/activate
```

https://aws.github.io/chalice/quickstart.html

AWS Chaliceの始め方 by 公式ドキュメント

Pipを使ってChaliceをインストール、動作確認

- デプロイの前にはクレデンシャルが設定されているかを確認！
  - ※「クレデンシャル？」となった方はこちら↓
    https://pages.awscloud.com/event_JAPAN_Ondemand_Hands-on-for-Beginners-1st-Step_LP.html

```
$ python3 -m pip install chalice
$ chalice --help
Usage: chalice [OPTIONS] COMMAND [ARGS]...
...
```

https://aws.github.io/chalice/quickstart.html

AWS Chaliceの始め方 by 公式ドキュメント

新しいプロジェクトを作ろう！

```
$ chalice new-project helloworld
$ cd helloworld
$ ls -la
drwxr-xr-x    .chalice
-rw-r--r--    app.py
-rw-r--r--    requirements.txt
```

https://aws.github.io/chalice/quickstart.html

AWS Chaliceの始め方 by 公式ドキュメント

とりあえずデプロイしてみる

```
$ chalice deploy
Creating deployment package.
Creating IAM role: helloworld-dev
Creating lambda function: helloworld-dev
Creating Rest API
Resources deployed:
  - Lambda ARN: arn:aws:lambda:us-west-2:12345:function:helloworld-dev
  - Rest API URL: https://restapiid.execute-api.region.amazonaws.com/api/
```

https://aws.github.io/chalice/quickstart.html

AWS Chaliceの始め方 by 公式ドキュメント

「Rest API URL」にアクセスしてみる

```
$ curl https://restapiid.execute-api.region.amazonaws.com/api/
{"hello": "world"}
```

APIができてる！

https://aws.github.io/chalice/quickstart.html

AWS Chaliceの始め方 by 公式ドキュメント

Chaliceの心臓部、app.pyの中身を見てみる

```
from chalice import Chalice

app = Chalice(app_name='helloworld')


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

ここに注目

https://aws.github.io/chalice/quickstart.html

AWS Chaliceの始め方 by 公式ドキュメント

Chaliceの心臓部、app.pyの中身を見てみる

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

ここととここが
対応している

```
$ curl https://restapiid.execute-api.region.amazonaws.com/api/
{"hello": "world"}
```

https://aws.github.io/chalice/quickstart.html

AWS Chaliceの始め方 by 公式ドキュメント

Chaliceの心臓部、app.pyの中身を見てみる

- 下の囲っている部分を「デコレーター」という（Pythonの機能）
- ここではデコレーターについて踏み込まない
  - 詳しく知りたい方は、後で検索してみてください！

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

https://aws.github.io/chalice/quickstart.html

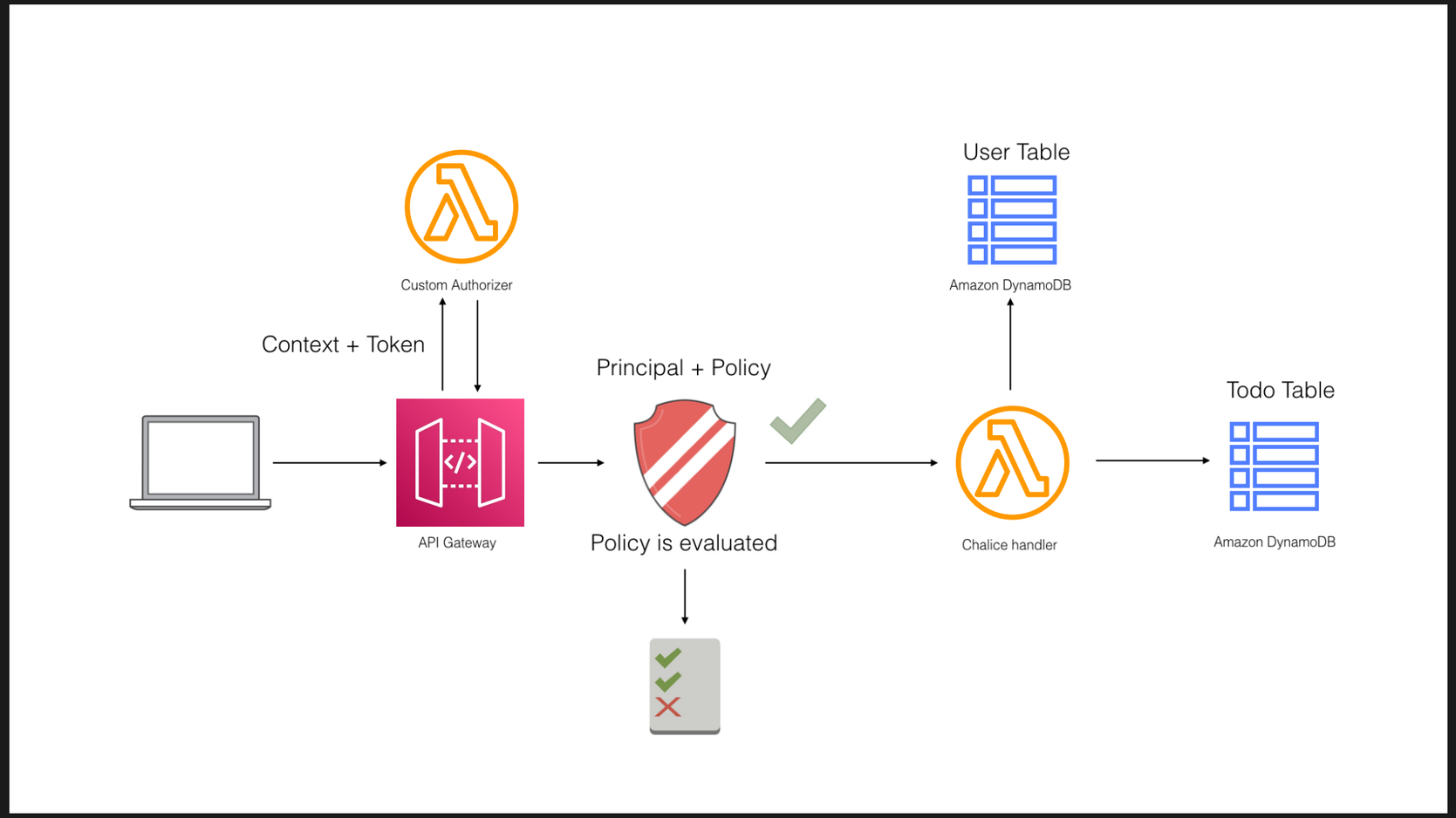
Todoアプリを眺めながら AWS ChaliceとRESTを知る

チュートリアル：Todoアプリ

- 「Todoを作成、更新、取得、削除、管理するためのサーバーレスWeb APIの作成、データベース内のTodo管理、JWTによる認可の追加」を体験することができる
- 使うAWSサービス
  - **AWS Lambda**, **Amazon API Gateway**, **Amazon DynamoDB**, AWS CodeBuild, AWS Systems Manager
- ここではRESTful API作成パートにフォーカスする

https://aws.github.io/chalice/samples/todo-app/index.html

Todoアプリのアーキテクチャー



https://aws.github.io/chalice/samples/todo-app/index.html

ここで作るAPIのリソースとHTTPメソッド

GET	/todos/	すべてのTodoを取得する
POST	/todos/	新しいTodoを作成する
GET	/todos/{id}	特定のTodoを取得する
DELETE	/todos/{id}	特定のTodoを削除する
PUT	/todos/{id}	特定のTodoの状態を更新する

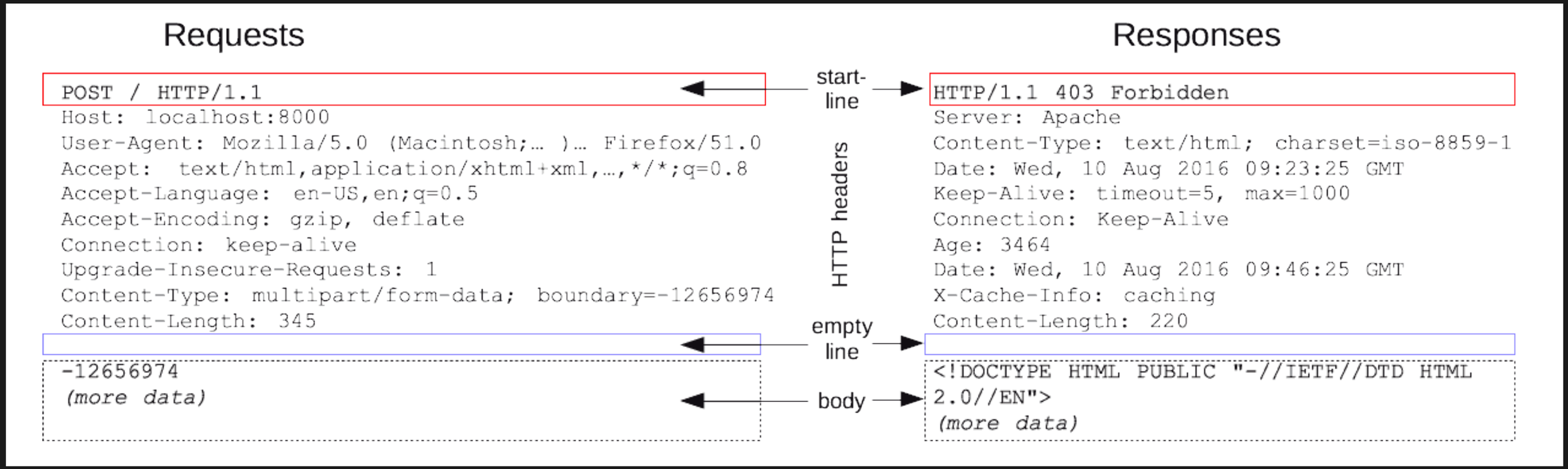
https://aws.github.io/chalice/samples/todo-app/index.html

Chaliceの前にRESTのおさらい

- Web APIを作るときに用いられるアーキテクチャースタイル
- 2000年、Roy Fieldingが博士論文で定義した
- RESTの原則に従って実装されているAPIをRESTful APIと呼ぶ
- RESTの主な特徴
  - 単一HTTPメッセージで1つの操作に関する情報を（理想では）含む
  - 扱う情報をURIで表現する「リソース」として定義し、それらをHTTPメソッド（PUT, GET, POST, DELETE, …）の表現で操作

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

（参考）HTTPメッセージの具体例



https://developer.mozilla.org/ja/docs/Web/HTTP/Messages

（参考）HTTPメッセージの具体例

https://developer.mozilla.org/ja/docs/Web/HTTP/Messages

Chaliceの前にRESTのおさらい

- Web APIを作るときに用いられるアーキテクチャースタイル
- 2000年、Roy Fieldingが博士論文で定義した
- RESTの原則に従って実装されているAPIをRESTful APIと呼ぶ
- RESTの主な特徴
  - 単一HTTPメッセージで1つの操作に関する情報を（理想では）含む
  - 扱う情報をURIで表現する「リソース」として定義し、それらをHTTPメソッド（PUT, GET, POST, DELETE, …）の表現で操作

https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

ここで作るAPIのリソースとHTTPメソッド

HTTPメソッド	リソース	
GET	/todos/	すべてのTodoを**取得する**
POST	/todos/	新しいTodoを**作成する**
GET	/todos/{id}	特定のTodoを**取得する**
DELETE	/todos/{id}	特定のTodoを**削除する**
PUT	/todos/{id}	特定のTodoの状態を**更新する**

https://aws.github.io/chalice/samples/todo-app/index.html

ここで作るAPIのリソースとHTTPメソッド

HTTPメソッド	リソース	
GET	/todos/	**すべてのTodo**を**取得する**
POST	/todos/	**新しいTodo**を**作成する**
GET	/todos/{id}	**特定のTodo**を**取得する**
DELETE	/todos/{id}	**特定のTodo**を**削除する**
PUT	/todos/{id}	**特定のTodo**の状態を**更新する**

https://aws.github.io/chalice/samples/todo-app/index.html

ここで作るAPIのリソースとHTTPメソッド

GET	/todos/	**すべてのTodo**を**取得する**
POST	/todos/	**新しいTodo**を**作成する**
G		
D		
PUT	/todos/{id}	**特定のTodo**の状態を**更新する**

HTTP メソッド

リソース

ここからは、具体的に
ソースコードとの対応を見ていく

https://aws.github.io/chalice/samples/todo-app/index.html

GET - /todos/ - すべてのTodoを取得

```
@app.route('/todos', methods=['GET'], authorizer=jwt_auth)
def list_todos():
    username = get_authorized_username(app.current_request)
    return get_app_db().list_items(username=username)
```

- サンプルアプリ内で独自に定義している関数
  - `get_authorized_username()` - リクエストの中身から認可済みのユーザー名を取得する関数
  - `get_app_db().list_items()` - Todoの入っているテーブルからTodoをリストする関数
    - そのほかに `add_item()`, `get_item()`, `delete_item()`, `update_item()` を定義している

https://aws.github.io/chalice/samples/todo-app/index.html

GET - /todos/ - すべてのTodoを取得

- methods
  - この関数で処理するHTTPメソッドを指定する
  - この場合は /todos/ へのGETメソッドにのみ反応する

```
@app.route('/todos', methods=['GET'], authorizer=jwt_auth)
def list_todos():
    username = get_authorized_username(app.current_request)
    return get_app_db().list_items(username=username)
```

https://aws.github.io/chalice/samples/todo-app/index.html

GET - /todos/ - すべてのTodoを取得

- authorizer
  - このリソースを認可するための関数を指定する
  - 今回はBuilt-in Authorizerを使った簡易認可を一例として実装
  - 実際には IAMAuthorizer や CognitoUserPoolAuthorizer を使うと楽

```
@app.route('/todos', methods=['GET'], authorizer=jwt_auth)
def list_todos():
    username = get_authorized_username(app.current_request)
    return get_app_db().list_items(username=username)
```

https://aws.github.io/chalice/samples/todo-app/index.html

GET - /todos/ - すべてのTodoを取得

- app.current_request
  - 現在のリソースに対するリクエストの情報が入っている
  - 入っている情報の例：header, query parameter, bodyなど

```
@app.route('/todos', methods=['GET'], authorizer=jwt_auth)
def list_todos():
    username = get_authorized_username(app.current_request)
    return get_app_db().list_items(username=username)
```

https://aws.github.io/chalice/samples/todo-app/index.html

POST - /todos/ - 新しいTodoを作成

```
@app.route('/todos', methods=['POST'], authorizer=jwt_auth)
def create_todo():
    body = app.current_request.json_body
    username = get_authorized_username(app.current_request)
    return get_app_db().add_item(
        username=username,
        description=body['description'],
        metadata=body.get('metadata'),
    )
```

https://aws.github.io/chalice/samples/todo-app/index.html

GET - /todos/{id} – 特定のTodoを取得

```
@app.route('/todos/{uid}', methods=['GET'], authorizer=jwt_auth)
def get_todo(uid):
    username = get_authorized_username(app.current_request)
    return get_app_db().get_item(uid, username=username)
```

https://aws.github.io/chalice/samples/todo-app/index.html

DELETE - /todos/{id} – 特定のTodoを削除

```
@app.route('/todos/{uid}', methods=['DELETE'], authorizer=jwt_auth)
def delete_todo(uid):
    username = get_authorized_username(app.current_request)
    return get_app_db().delete_item(uid, username=username)
```

https://aws.github.io/chalice/samples/todo-app/index.html

PUT - /todos/{id} – 特定のTodoの状態を更新

```
@app.route('/todos/{uid}', methods=['PUT'], authorizer=jwt_auth)
def update_todo(uid):
    body = app.current_request.json_body
    username = get_authorized_username(app.current_request)
    get_app_db().update_item(
        uid,
        description=body.get('description'),
        state=body.get('state'),
        metadata=body.get('metadata'),
        username=username)
```

https://aws.github.io/chalice/samples/todo-app/index.html

まとめ

- 「@app.route()にリソース、HTTPメソッドなどを書けば、それにあわせてその下の関数が実行される」という感じで、素直にLambdaとAPI Gatewayを連携してRESTful APIを作成することができる
- 簡単にリクエストの中身を取得しながら、データベースに書き込んだりレスポンスとして加工したりできる

```
@app.route('/todos', methods=['GET'], authorizer=jwt_auth)
def list_todos():
    username = get_authorized_username(app.current_request)
    return get_app_db().list_items(username=username)
```

AWS Chaliceの便利機能

関数を定期的に実行する

- 書き方はCloudWatch Events（EventBridge）に準じる

```
from chalice import Chalice, Rate

app = Chalice(app_name="helloworld")

# Automatically runs every 5 minutes
@app.schedule(Rate(5, unit=Rate.MINUTES))
def periodic_task(event):
    return {"hello": "world"}
```

https://aws.github.io/chalice/topics/events.html

Amazon S3 Event連携

- 先にS3バケットは作成しておく

```
from chalice import Chalice

app = Chalice(app_name="helloworld")

# Whenever an object is uploaded to 'mybucket'
# this lambda function will be invoked.

@app.on_s3_event(bucket='mybucket')
def handler(event):
    print("Object uploaded for bucket: %s, key: %s"
          % (event.bucket, event.key))
```

https://aws.github.io/chalice/topics/events.html

Amazon SNS連携

- 先にSNSのトピックは作っておく

```
from chalice import Chalice

app = Chalice(app_name='chalice-sns-demo')
app.debug = True

@app.on_sns_message(topic='my-demo-topic')
def handle_sns_message(event):
    app.log.debug("Received message with subject: %s, message: %s",
                  event.subject, event.message)
```

https://aws.github.io/chalice/topics/events.html

Amazon SQS連携

- 先にSQSのキューは作っておく

```
from chalice import Chalice

app = Chalice(app_name="helloworld")

# Invoke this lambda function whenever a message
# is sent to the ``my-queue-name`` SQS queue.

@app.on_sqs_message(queue='my-queue-name')
def handler(event):
    for record in event:
        print("Message body: %s" % record.body)
```

https://aws.github.io/chalice/topics/events.html

Amazon Kinesis連携

- 先にKinesis Data Streamのストリームは作っておく

```
from chalice import Chalice

app = chalice.Chalice(app_name='kinesiseventdemo')
app.debug = True

@app.on_kinesis_record(stream='mystream')
def handle_kinesis_message(event):
    for record in event:
        # The .data attribute is automatically base64 decoded for you.
        app.log.debug("Received message with contents: %s", record.data)
```

https://aws.github.io/chalice/topics/events.html

DynamoDB Streams連携

- ストリームのARNを指定する必要あり

```
from chalice import Chalice

app = chalice.Chalice(app_name='ddb-event-demo')
app.debug = True

@app.on_dynamodb_record(stream_arn='arn:aws:dynamodb:.../stream/2020')
def handle_ddb_message(event):
    for record in event:
        app.log.debug("New: %s", record.new_image)
```

https://aws.github.io/chalice/topics/events.html

Pure Lambda Function

- 2つめと3つめの関数が
  Pure Lambda Function
- どちらも他のAWSサービスから
  明示的に呼び出さないと
  実行されない
- 他のAWSサービスからLambda
  関数を呼び出したい場合に、
  コードをChaliceで完結できる

```
app = chalice.Chalice(app_name='foo')

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

@app.lambda_function()
def custom_lambda_function(event, context):
    # Anything you want here.
    return {}

@app.lambda_function(name='MyFunction')
def other_lambda_function(event, context):
    # Anything you want here.
    return {}
```

https://aws.github.io/chalice/topics/purelambda.html

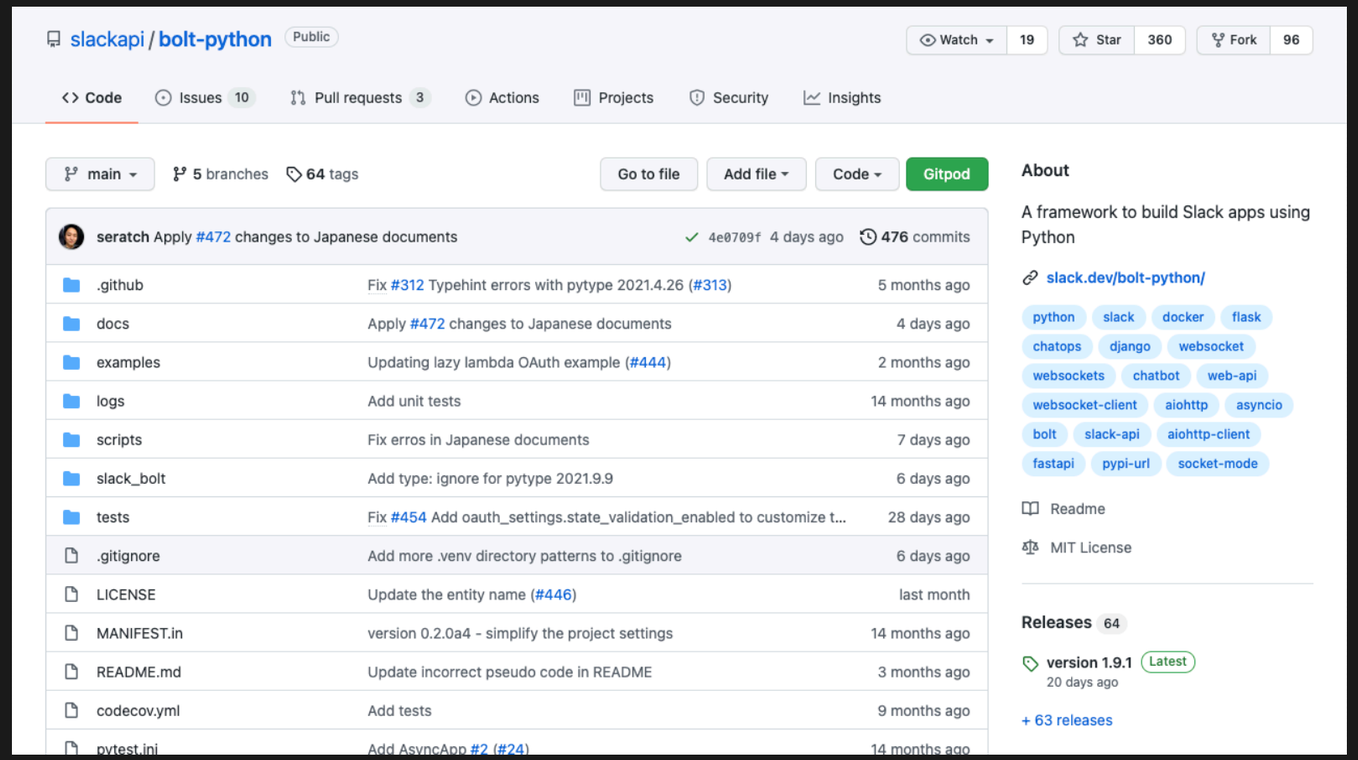
お悩み相談と参考情報

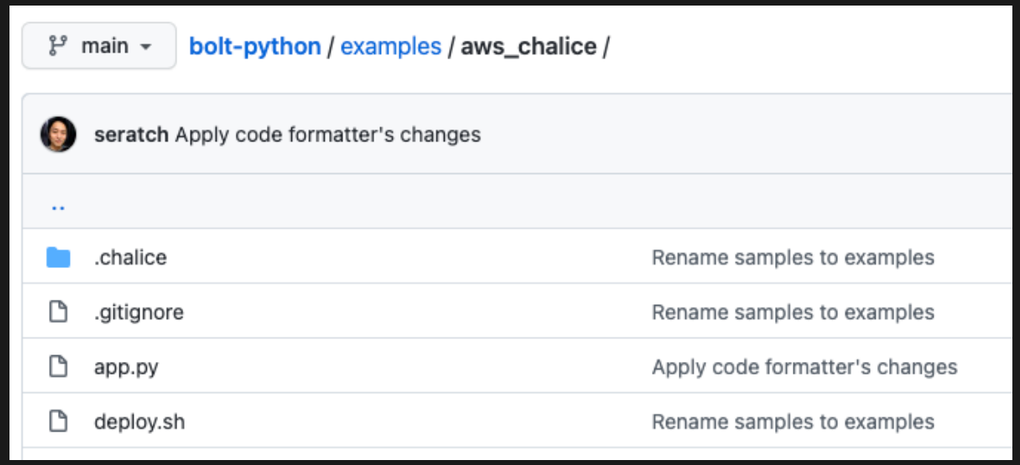
何から作ってみたらいい？

- 個人的には**チャットボット**がおすすめ
- いいところ
  - フロントエンドを作り込まなくていい
  - すぐに役に立つものができて**モチベーションが湧く**
  - API連携の勉強になる

Slack Bolt for PythonのChalice実装サンプル

- Slack Bolt: Slackアプリを作るためのフレームワーク
- /examplesの中にAWS Chaliceとの連携例がある





https://github.com/slackapi/bolt-python

設計とかに自信がない。。。

- いい設計ももちろん大事だけど、**とりあえず作り始めること**が重要！
  - 自戒もこめて。。。
- CI/CDとかどうしよう？というのも同じ、**まずは作り始めよう**！
  - こちらも自戒をこめて。。。
- 重要なこと：**トライアンドエラーを繰り返すこと**！最初から完璧を目指すのではなく、だんだんよくしていく！
  - 自戒をこめて（しつこい）

Chaliceの情報ってどこにある？

- まずは公式リポジトリとドキュメント
  - https://github.com/aws/chalice
  - https://aws.github.io/chalice/
- [AWS Black Belt Online Seminar] Dive Deep into AWS Chalice
  - https://d1.awsstatic.com/webinars/jp/pdf/services/20190619_AWS-BlackBeltOnlineSeminar_DiveDeepIntoAWSChalice.pdf
- そのほか検索するといろいろ出てきます！

まとめ

まとめ

- **AWS Chalice**は**Python**製サーバーレスアプリケーションフレームワーク
  - めちゃくちゃ**サクッ**とRESTful APIを作ることができる
  - 便利機能もたくさん！
  - デコレーターを活用して素直に**LambdaとAPI Gatewayを連携した RESTful API**を作成することができる
- AWS Chaliceは「**とりあえず作ってみよう！**」を強力に支援
  - トライアンドエラーを素早く繰り返すことができる

Let’s Chalice & RESTful Life!

Thank you!

Please complete the session survey