サーバーレスでAPIを提供する際のアプリケーション"以外"の話

ServerlessDays Fukuoka 2019
Shimpei Takamatsu
(@shimpeiws)

Shimpei Takamatsu
(@shimpeiws)

01

ソフトウェアエンジニア at Tokyo

Webアプリケーションを中心に

JavaScript / Ruby / React / Node.js ….

02

アジャイル開発

Certified Scrum Master

03

技術支援

アーリーステージのスタートアップを中心に、

アーキテクチャ/設計・実装のサポート

04

ジョギング

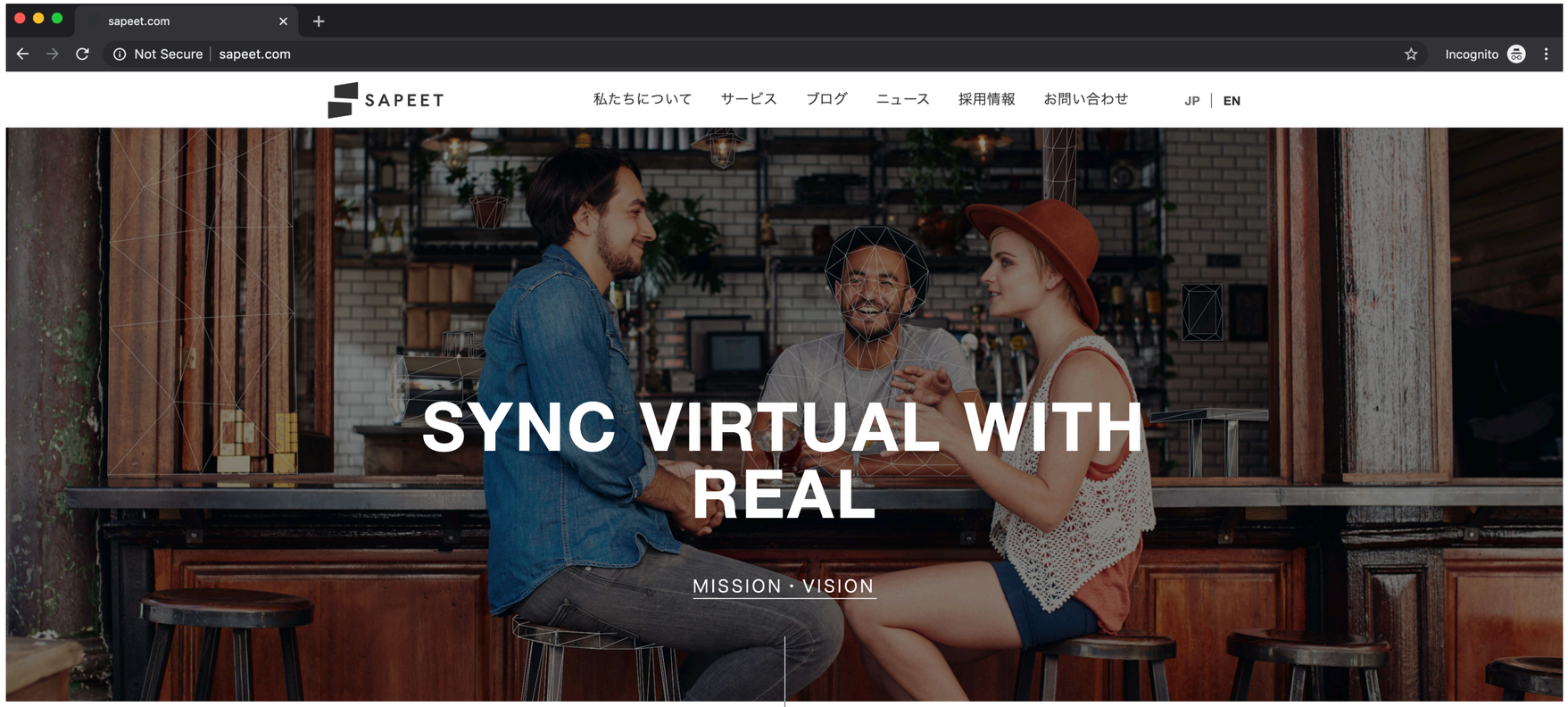
冬は寒いし暗い、サボりがち

今日はこちらのお話です

目次

01 サービスの概要とアーキテクチャ

01 サービスの概要とアーキテクチャ



WHAT'S SAPEET

Sapeetは、Computer GraphicsとComputer VisionとDeep Learningを組み合わせた3Dアルゴリズムで、モノの形状把握及び3Dデータ化、デジタル空間上でシミュレーションや設計の最適化を実現します。

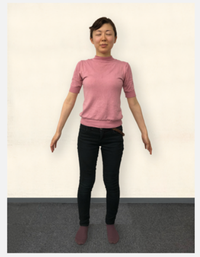
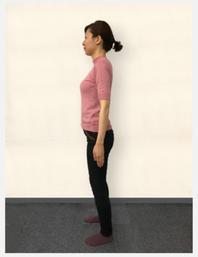
http://sapeet.com/

01 サービスの概要とアーキテクチャ

体型サイジングモジュール（スマホサイジング）

スマホで撮影した全身写真数枚とユーザーの基本情報（4項目）から、ユーザーの体型を模した3Dアバターと任意の体型寸法項目を推定します

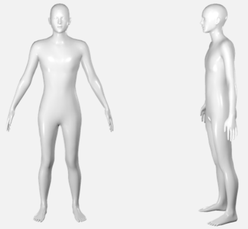
全身写真（数枚）

ユーザー基本情報（4項目）

- 性別
- 年齢
- 身長
- 体重

3Dアバター



任意の体型寸法項目

- 首囲
- 肩幅
- バスト
- ウエスト
- 着丈
- 袖丈
- 裄丈
- 腕周り
- ヒップ
- 股下

※ご要望に応じた測定部位の追加/変更が可能

人々の生活をスムーズで面白く

私たちは人間を中心とした形あるものをそのままの姿でデータ化することによって、計算機の処理の対象となり、人々がより自分に合うモノ・コトにアクセスでき、スムーズでワクワクする生活を実現します.

強みを持つ技術

- 3次元関連技術
- 物理演算
- 機械学習/深層学習など

01 サービスの概要とアーキテクチャ

システム構成の特徴

- APIサービング部分とアルゴリズムモジュールに分かれる
- アルゴリズム部分の応答スピードはモジュールによって様々
- アルゴリズムモジュールは言語/ワークロード/レスポンスタイム/IOが様々

APIサービング

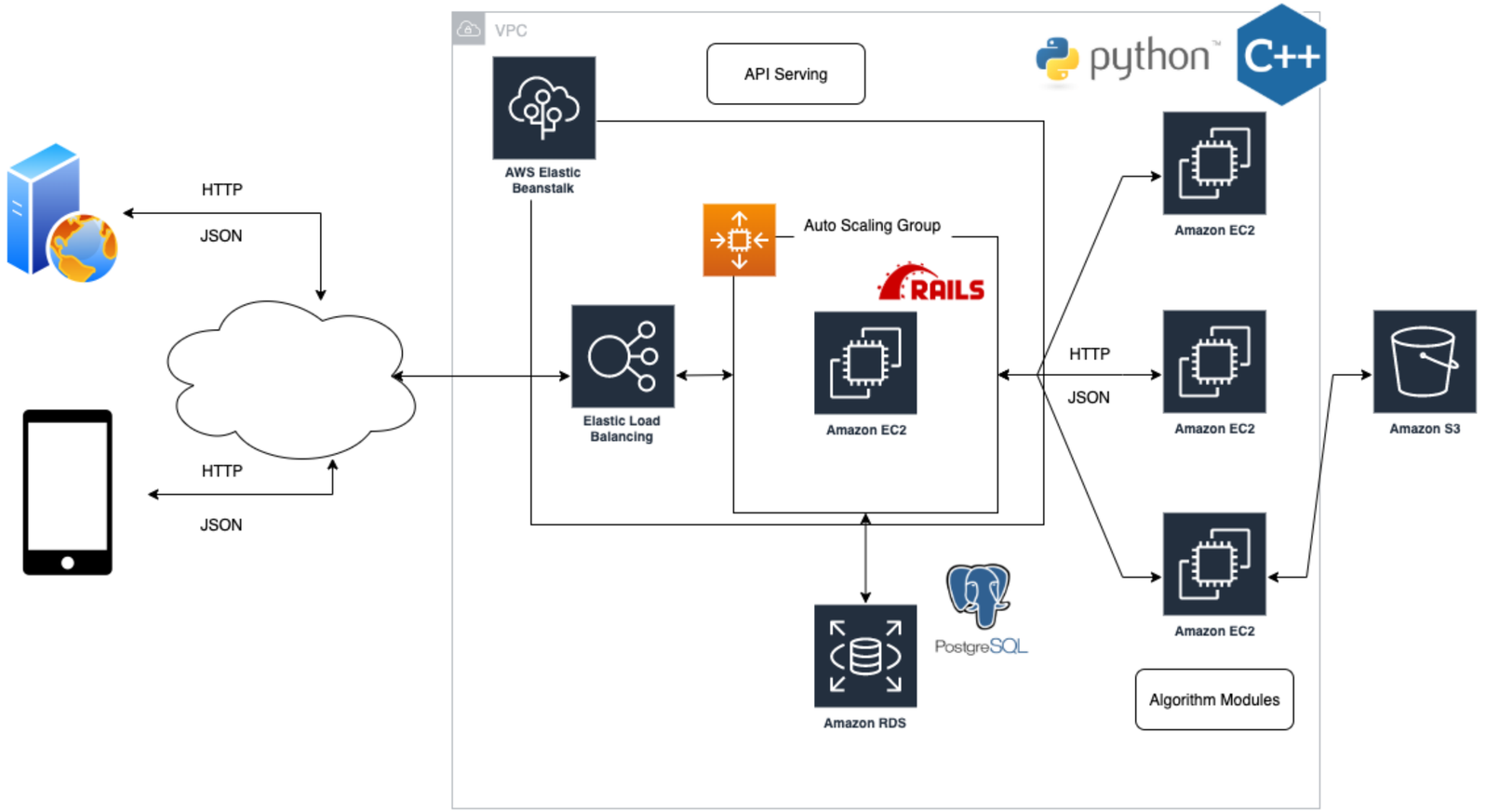
- ユーザーからのリクエストを受けレスポンスを返却
  - HTTP <> JSON
- 後続のアルゴリズムモジュールにリクエストし、結果をアグリゲーション
- 認証・認可など

アルゴリズムモジュール

- 画像認識、機械学習・深層学習、アバター生成 etc…
- 使用言語がアルゴリズムによって異なる
- ワークロードがアルゴリズムによって異なる
- IOがアルゴリズムによって異なる
  - IN: 採寸データ、画像 etc…
  - OUT: データ、画像、3Dファイル etc…

01 サービスの概要とアーキテクチャ

Before



01 APIサーバー

Rails + PostgreSQL

Elastic Beanstalkでオートスケーリング

02 Algorithm Modules

Python, C++ etc…

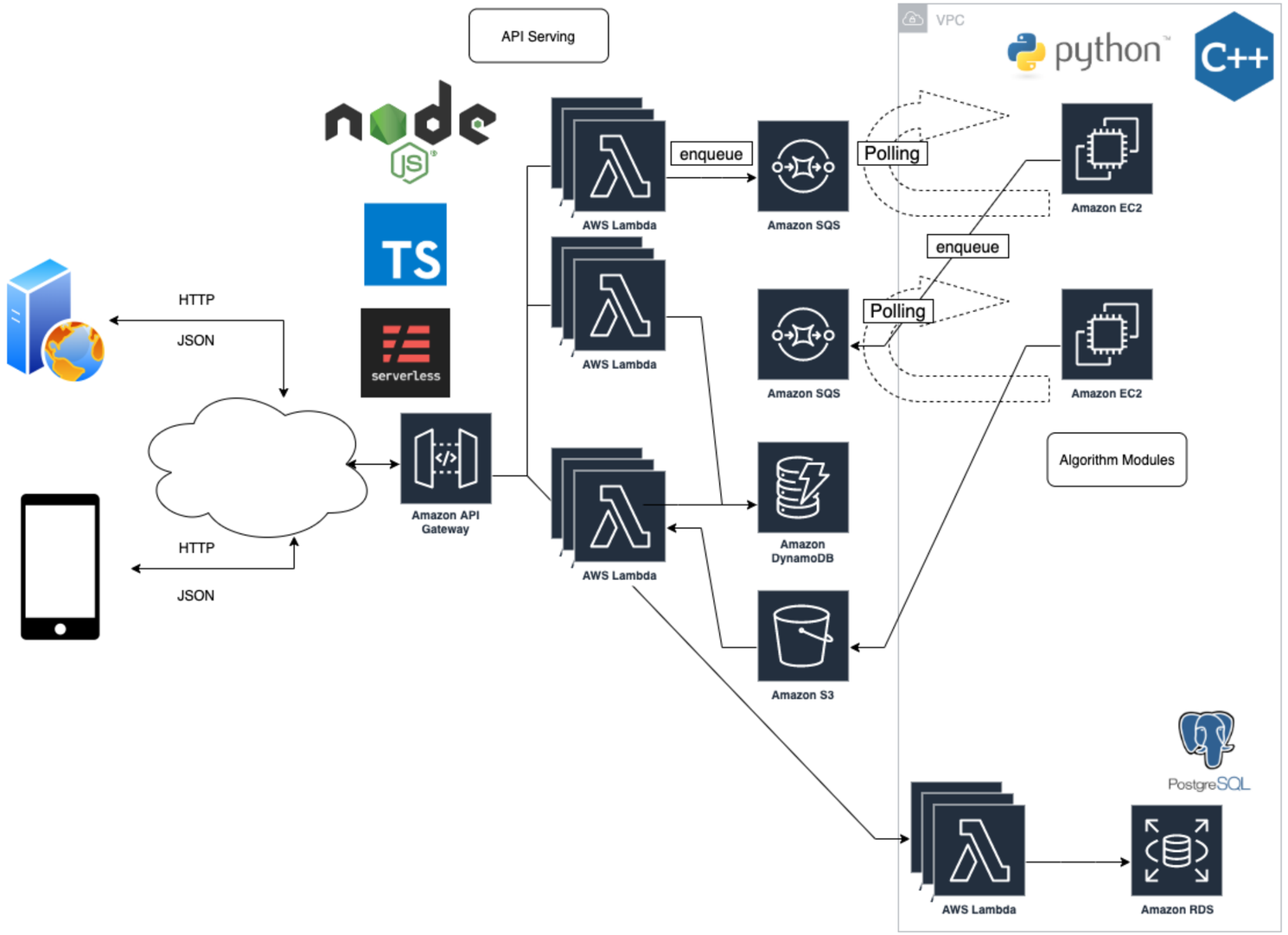
EC2インスタンス(CPU/GPU)

各モジュールがHTTPインタフェース提供

03 通信(API <> アルゴリズム)

HTTP/JSONですべて同期的に通信

01 サービスの概要とアーキテクチャ



After

01

APIサーバー

Serverless + DynamoDB + 一部postgres

スケーリングはLambda

02

Algorithm Modules

Python, C++ etc…

EC2インスタンス(CPU/GPU)

各モジュールはSQSをポーリング

03

通信(API <> アルゴリズム)

SQSを中心としたインタフェース

一部S3やDynamoDBのトリガー

02

サーバーレスはどうだった?

よかった 😀😀😀

02 サーバーレスはどうだった?

開発前の事前準備

- 開発者は~3人程度、機能開発・テストで2ヶ月程度の期間
- 事前に自身が担当で仕様を満たすことが可能かのPoCと開発環境の整備をおこなった

PoC

- Serverless Framework(Node.js + TypeScript) / AWS
- 単純なAPI、ストリームからのLambda起動
- ミドルウェアの使用 (DynamoDB、API Gatewayでのwebsocket etc…)
- プロダクションでは使用しなかったものもある

開発環境の整備

- レイヤリング/ディレクトリ構成の決定
- 共通モジュールの作成(レスポンスやエラーハンドリング)
- ユニットテスト(jest)の環境とテスト対象・テストの書き方のパターンを整備
- Lint

02 サーバーレスはどうだった?

Good / More

- 開発を通じて良かった部分(Good)と工夫した or もっと改善したい(More)な部分
- アプリケーション開発は総じてスムーズだった
- Moreはアプリケーション”以外”の部分

Good

- 開発の立ち上がりが早かった
- アプリケーションを書く分には学習コストが低い
  - 基本的にLambdaのFunctionを書くだけ
- ユニットテストで品質は安定した

More

- CIやリリースフロー
- local開発環境
- ネットワーク設定
  - ドメイン、VPC、外部ネットワーク接続 etc…

02 サーバーレスはどうだった?

まとめ

- Serverless Frameworkは開発に道筋を与えてくれた。アプリケーション開発に集中できる
  - ある程度の準備をすればチーム開発での立ち上がりも早い
- アプリケーション開発に集中するためにはアプリケーション”以外”の部分を整えておく必要がある

03 開発フロー/開発環境

02 サーバーレスはどうだった?

Good / More

- 開発を通じて良かった部分(Good)と工夫した or もっと改善したい(More)な部分
- アプリケーション開発は総じてスムーズだった
- Moreはアプリケーション”以外”の部分

Good

- 開発の立ち上がりが早かった
- アプリケーションを書く分には学習コストが低い
  - 基本的にLambdaのFunctionを書くだけ
- ユニットテストで品質は安定した

More

- CIやリリースフロー
- local開発環境
- ネットワーク設定
  - ドメイン、VPC、外部ネットワーク接続 etc…

03 開発環境/開発フロー

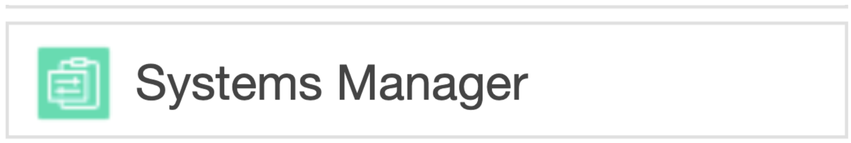
CIの導入

- Circle CI/GitHub Actionsをセットアップ
- ユニットテスト(Jest)とLintをpush時に実行
- PRマージのタイミングで社内検証環境にデプロイ

03 開発環境/開発フロー

デプロイ

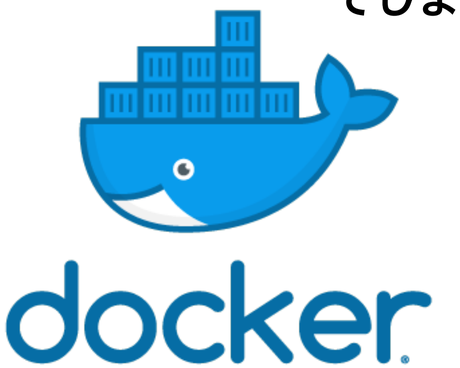
- AWS CodeBuildを使用、sls deployしているだけ
  - CodeBuildとテンプレートの作成はserverlessとは別に CloudFormationで書いている
  - 環境変数はSystem ManagerのParameter Storeを使用
- CodeBuildの実行はCIからaws-cliで実行

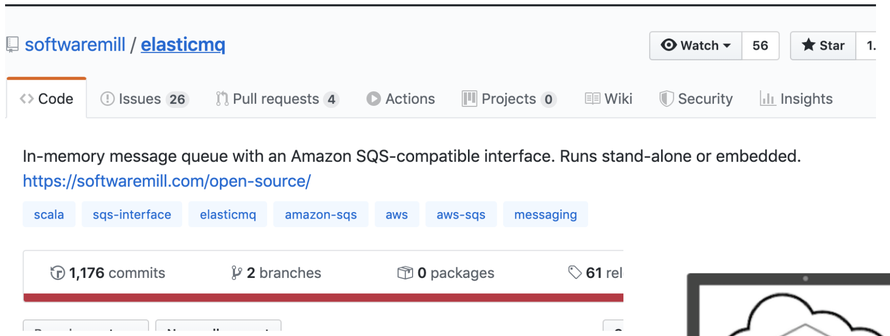


03 開発環境/開発フロー

ローカル開発環境

- docker-composeでミドルウェアを揃える
  - DynamoDB、ElasticMQ、LocalStack(S3用)
- ストリーム経由起動が多いのでserverless-offlineやinvokeでのローカル動作確認がしづらい
- ユニットテストは結構しっかり書いている (EventやContextさえmockしてしまえば書きやすい)





LocalStack

03 開発環境/開発フロー

まだ課題に感じている部分

- 開発環境
  - やっぱりローカルでも動作確認したい、クライアントとあわせて動作確認した
    - ローカルで動かせないならば、feature単位のテスト環境を作る?
- デプロイ
  - RDBの場合、migrationをどう管理していつデプロイする?
  - ワンショットで流したいスクリプト的なものをどう流す?
    - serverless deployに乗り切らないものをどう扱う?

03 開発環境/開発フロー

まとめ

- 開発環境
  - ミドルウェアが多くなりがちで、動作確認まで出来るローカル環境が作りづらい
  - dockerイメージでミドルウェアを揃えて、テストを書くことで品質担保した
- CIの整備
  - CodeBuildをserverlessとは別のCloudFormationで準備、CIから実行してデプロイ
  - serverless deployに乗り切らないものをどう扱うか? みなさんmakeやshell、作り込んでますよね?

04

サーバーレスと非機能要件(ネットワークなど)

02 サーバーレスはどうだった?

Good / More

- 開発を通じて良かった部分(Good)と工夫した or もっと改善したい(More)な部分
- アプリケーション開発は総じてスムーズだった
- Moreはアプリケーション”以外”の部分

Good

- 開発の立ち上がりが早かった
- アプリケーションを書く分には学習コストが低い
  - 基本的にLambdaのFunctionを書くだけ
- ユニットテストで品質は安定した

More

- CIやリリースフロー
- local開発環境
- ネットワーク設定
  - ドメイン、VPC、外部ネットワーク接続 etc…

04

サーバーレスと非機能要件(ネットワークなど)

Q1: APIサーバーを
独自ドメインで運用したいです

04 サーバーレスと非機能要件(ネットワークなど)

Q1: APIサーバーを独自ドメインで運用したいです

- URLのデフォルトはAWSが発行するFQDN
  - 独自ドメイン使用のためにはRoute 53でドメインに紐付ける + ssl証明書の準備
- “amplify-education/serverless-domain-manager” が便利
  - 事前にCertificate ManagerでSSL/TLS証明書を発行しておく
  - デプロイ前に“serverless create_domain” でCustom Domainを作成する

```
custom:
  customDomain:
    domainName: ${self:provider.stage}-your-domain.${env:CUSTOM_DOMAIN_NAME}
    certificateName: ${env:CERTIFICATE_NAME}
    basePath: ''
    stage: ${self:provider.stage}
    createRoute53Record: true
    endpointType: 'regional'
```

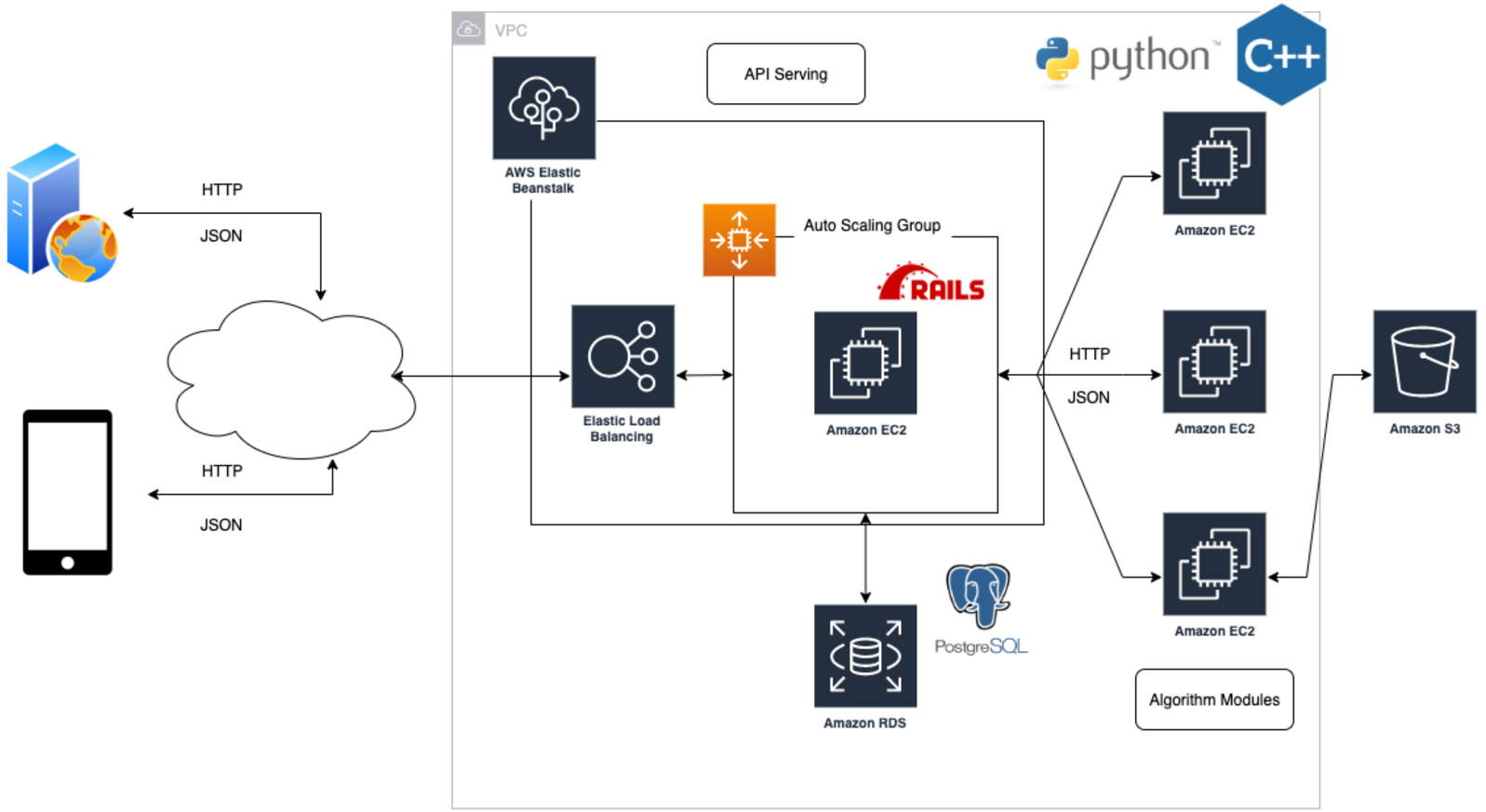
04

サーバーレスと非機能要件(ネットワークなど)

Q2: アルゴリズムモジュールは
インターネットに公開したくないです

01 サービスの概要とアーキテクチャ

Before



01 APIサーバー

Rails + PostgreSQL

Elastic Beanstalkでオートスケーリング

02 Algorithm Modules

Python, C++ etc…

EC2インスタンス(CPU/GPU)

各モジュールがHTTPインタフェース提供

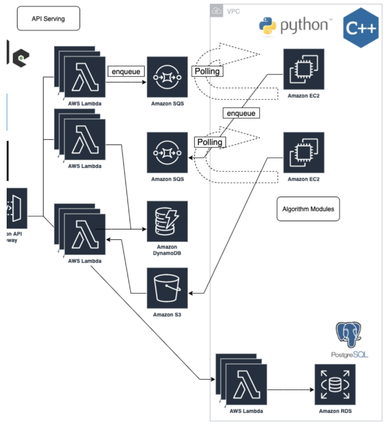
03 通信(API <> アルゴリズム)

HTTP/JSONですべて同期的に通信

04 サーバーレスと非機能要件(ネットワークなど)

Q2: アルゴリズムモジュールはインターネットに公開したくないです

- Beforeの構成では全てが同じVPC内にいた
  - サーバーレスは基本publicなのでアルゴリズムモジュールもパブリックになってしまう?という懸念が当初あった
- API <> アルゴリズムモジュールのインタフェースをSQSにしたことで自然と解決
  - アルゴリズムモジュールからSQSの取得ができればいいのでOUTBOUNDだけ空いてればよい



04

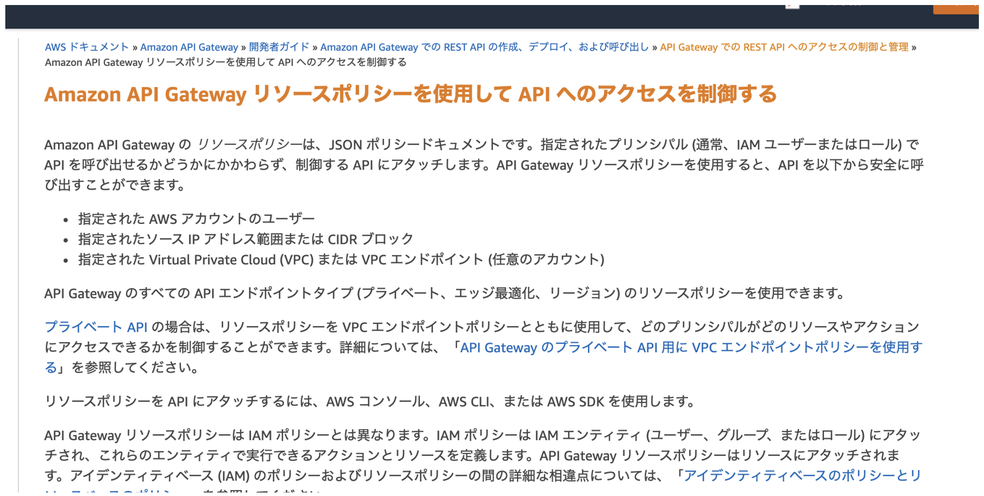
サーバーレスと非機能要件(ネットワークなど)

Q3: APIへの接続元のIPを制限したいです

04 サーバーレスと非機能要件(ネットワークなど)

Q3: APIへの接続元のIPを制限したいです

- エンタープライズな案件 & 接続元がサーバーの場合に発生する
  - 認証を作り込むよりも双方対応が簡単なので選択肢にあがる
- API GatewayのResource PolicyでIP制限可能
  - 知らないとどこのレイヤーでかけるの?となりますね



AWS ドキュメント » Amazon API Gateway » 開発者ガイド » Amazon API Gateway での REST API の作成、デプロイ、および呼び出し » API Gateway での REST API へのアクセスの制御と管理 » Amazon API Gateway リソースポリシーを使用して API へのアクセスを制御する

Amazon API Gateway リソースポリシーを使用して API へのアクセスを制御する

Amazon API Gateway の *リソースポリシー*は、JSON ポリシードキュメントです。指定されたプリンシパル (通常、IAM ユーザーまたはロール) で API を呼び出せるかどうかにかかわらず、制御する API にアタッチします。API Gateway リソースポリシーを使用すると、API を以下から安全に呼び出すことができます。

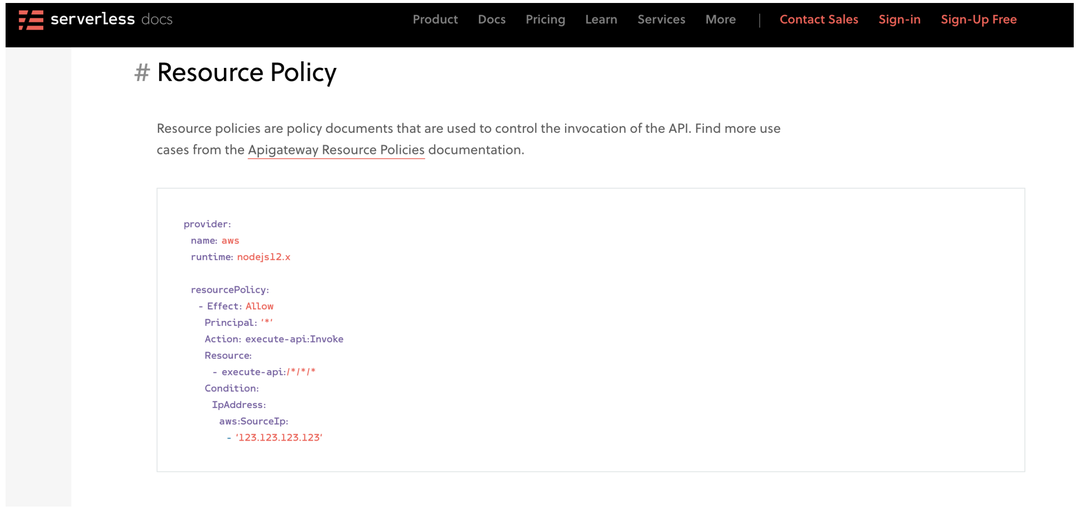
- 指定された AWS アカウントのユーザー
- 指定されたソース IP アドレス範囲または CIDR ブロック
- 指定された Virtual Private Cloud (VPC) または VPC エンドポイント (任意のアカウント)

API Gateway のすべての API エンドポイントタイプ (プライベート、エッジ最適化、リージョン) のリソースポリシーを使用できます。

プライベート API の場合は、リソースポリシーを VPC エンドポイントポリシーとともに使用して、どのプリンシパルがどのリソースやアクションにアクセスできるかを制御することができます。詳細については、「API Gateway のプライベート API 用に VPC エンドポイントポリシーを使用する」を参照してください。

リソースポリシーを API にアタッチするには、AWS コンソール、AWS CLI、または AWS SDK を使用します。

API Gateway リソースポリシーは IAM ポリシーとは異なります。IAM ポリシーは IAM エンティティ (ユーザー、グループ、またはロール) にアタッチされ、これらのエンティティで実行できるアクションとリソースを定義します。API Gateway リソースポリシーはリソースにアタッチされます。アイデンティティベース (IAM) のポリシーおよびリソースポリシーの間の詳細な相違点については、「アイデンティティベースのポリシーとリ



serverless docs　Product　Docs　Pricing　Learn　Services　More　Contact Sales　Sign-in　Sign-Up Free

Resource Policy

Resource policies are policy documents that are used to control the invocation of the API. Find more use cases from the Apigateway Resource Policies documentation.

```
provider:
  name: aws
  runtime: nodejs12.x

  resourcePolicy:
    - Effect: Allow
      Principal: '*'
      Action: execute-api:Invoke
      Resource:
        - execute-api:/*/*/*
      Condition:
        IpAddress:
          aws:SourceIp:
            - '123.123.123.123'
```

04

サーバーレスと非機能要件(ネットワークなど)

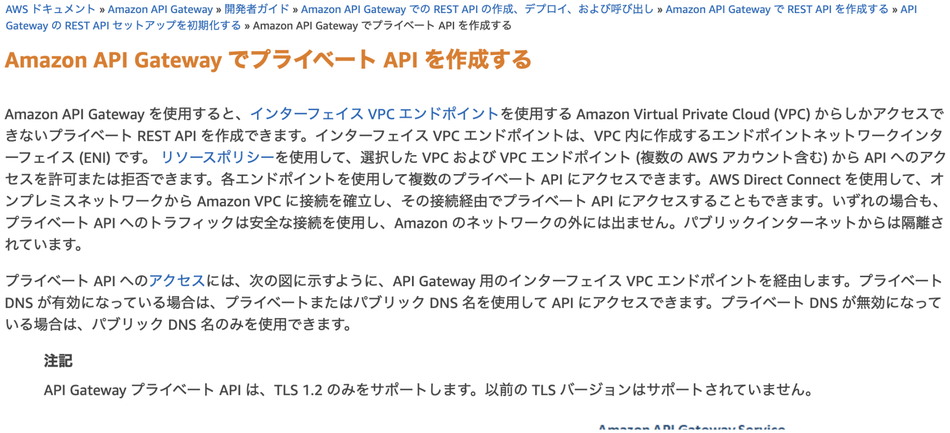
Q4: Admin用のAPIを
社内限定に公開したいです

04

サーバーレスと非機能要件(ネットワークなど)

Q4: Admin用のAPIを社内限定に公開したいです

- 外部に提供しているAPIとは別に管理機能を作りたい!けれど、画面や認証を作る工数が…
  - CLIツールを作り、踏み台サーバーからのみアクセス可能にしよう
- API GatewayのPrivate EndpointsでVPCエンドポイント経由でAPI Gatewayにアクセス可能
  - serverlessでもendpointTypeというパタメータでサポートされている



AWS ドキュメント » Amazon API Gateway » 開発者ガイド » Amazon API Gateway での REST API の作成、デプロイ、および呼び出し » Amazon API Gateway で REST API を作成する » API Gateway の REST API セットアップを初期化する » Amazon API Gateway でプライベート API を作成する

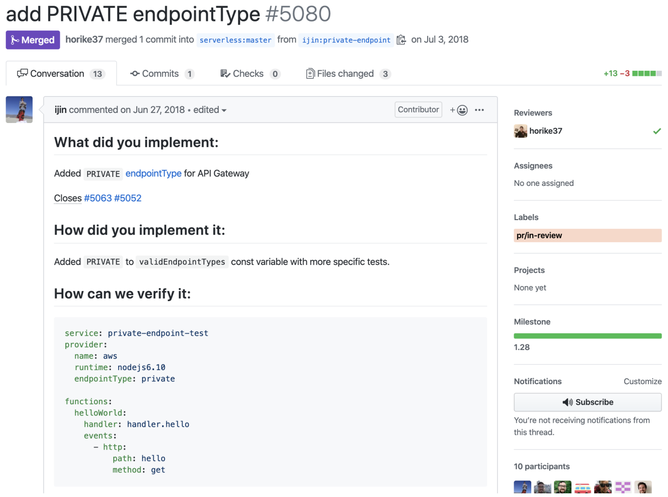
Amazon API Gateway でプライベート API を作成する

Amazon API Gateway を使用すると、インターフェイス VPC エンドポイントを使用する Amazon Virtual Private Cloud (VPC) からしかアクセスできないプライベート REST API を作成できます。インターフェイス VPC エンドポイントは、VPC 内に作成するエンドポイントネットワークインターフェイス (ENI) です。 リソースポリシーを使用して、選択した VPC および VPC エンドポイント (複数の AWS アカウント含む) から API へのアクセスを許可または拒否できます。各エンドポイントを使用して複数のプライベート API にアクセスできます。AWS Direct Connect を使用して、オンプレミスネットワークから Amazon VPC に接続を確立し、その接続経由でプライベート API にアクセスすることもできます。いずれの場合も、プライベート API へのトラフィックは安全な接続を使用し、Amazon のネットワークの外には出ません。パブリックインターネットからは隔離されています。

プライベート API へのアクセスには、次の図に示すように、API Gateway 用のインターフェイス VPC エンドポイントを経由します。プライベート DNS が有効になっている場合は、プライベートまたはパブリック DNS 名を使用して API にアクセスできます。プライベート DNS が無効になっている場合は、パブリック DNS 名のみを使用できます。

注記

API Gateway プライベート API は、TLS 1.2 のみをサポートします。以前の TLS バージョンはサポートされていません。



add PRIVATE endpointType #5080

Merged horike37 merged 1 commit into serverless:master from ijin:private-endpoint on Jul 3, 2018

Conversation 13 Commits 1 Checks 0 Files changed 3 +13 −3

ijin commented on Jun 27, 2018 • edited Contributor

What did you implement:

Added `PRIVATE` endpointType for API Gateway

Closes #5063 #5052

How did you implement it:

Added `PRIVATE` to `validEndpointTypes` const variable with more specific tests.

How can we verify it:

```
service: private-endpoint-test
provider:
  name: aws
  runtime: nodejs6.10
  endpointType: private

functions:
  helloWorld:
    handler: handler.hello
    events:
      - http:
          path: hello
          method: get
```

Reviewers: horike37

Assignees: No one assigned

Labels: pr/in-review

Projects: None yet

Milestone: 1.28

Notifications Customize

Subscribe

You're not receiving notifications from this thread.

10 participants

04

サーバーレスと非機能要件(ネットワークなど)

Q5: RDBが使いたくなりました

04 サーバーレスと非機能要件(ネットワークなど)

Q5: RDBが使いたくなりました

- 基本的にDynamoDBを使う方針ですが、リレーショナルなデータを扱うシーンが出てきた
- RDB(RDS)に限らずVPC内のリソースにアクセスする時
  - LambdaをVPC内に配置する典型的なパターン (serverlessでのサポートもあるので比較的簡単)

AWS ドキュメント » AWS Lambda » 開発者ガイド » 関数の AWS Lambda 管理 » VPC 内のリソースにアクセスするように Lambda 関数を設定する

VPC 内のリソースにアクセスするように Lambda 関数を設定する

アカウントの Virtual Private Cloud (VPC) のプライベートサブネットに接続するように関数を設定することができます。Amazon Virtual Private Cloud (Amazon VPC) を使用して、データベース、キャッシュインスタンス、または内部サービスなどのリソースのプライベートネットワークを作成します。実行中にプライベートリソースにアクセスするには、関数を VPC に接続します。

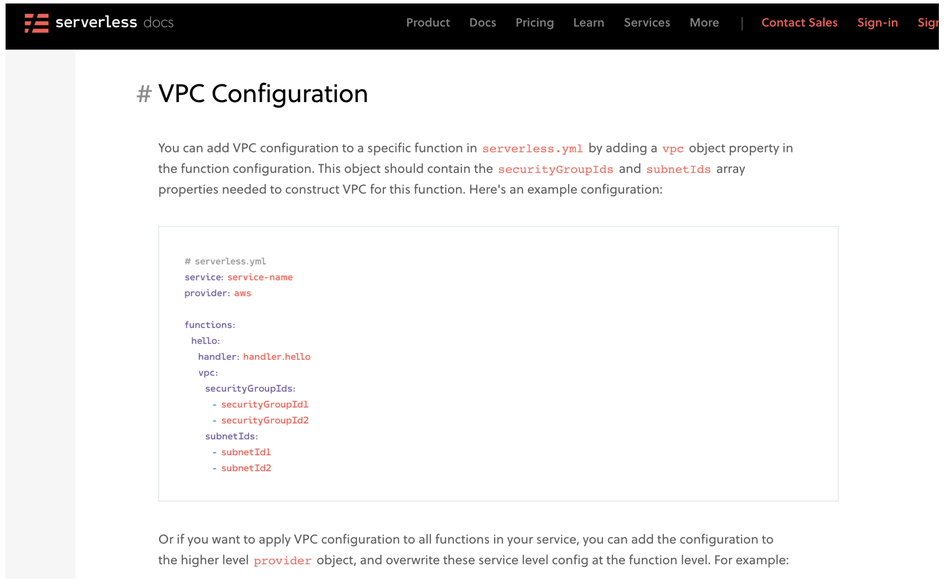
Lambda は、関数の VPC 設定内のセキュリティグループとサブネットの組み合わせごとに Elastic Network Interface を作成します。このプロセスには 1 分ほどかかります。 同じサブネットに複数の関数が接続されている場合、ネットワークインターフェイスは共有されるため、Lambda で管理されたネットワークインターフェイスが既に存在するサブネットに追加関数を接続する方がはるかに高速です。関数の数が多い場合や、非常に使用率が高い関数がある場合は、Lambda で追加のネットワークインターフェイスが作成されることがあります。

Lambda 関数は、専有インスタンスのテナンシーを使用して VPC に直接接続することはできません。専有 VPC のリソースに接続するには、デフォルトのテナンシーで 2 番目の VPC にピア接続します。

VPC チュートリアル

- チュートリアル: Amazon VPC で Amazon RDS にアクセスできるように Lambda 関数を設定する
- チュートリアル: Amazon VPC で Amazon ElastiCache にアクセスできるように Lambda 関数を設定する

実行ロールおよびユーザーアクセス許可



serverless docs　Product　Docs　Pricing　Learn　Services　More　|　Contact Sales　Sign-in　Sign

VPC Configuration

You can add VPC configuration to a specific function in `serverless.yml` by adding a `vpc` object property in the function configuration. This object should contain the `securityGroupIds` and `subnetIds` array properties needed to construct VPC for this function. Here's an example configuration:

```
# serverless.yml
service: service-name
provider: aws

functions:
  hello:
    handler: handler.hello
    vpc:
      securityGroupIds:
        - securityGroupId1
        - securityGroupId2
      subnetIds:
        - subnetId1
        - subnetId2
```

Or if you want to apply VPC configuration to all functions in your service, you can add the configuration to the higher level `provider` object, and overwrite these service level config at the function level. For example:

04 サーバーレスと非機能要件(ネットワークなど)

Q6: RDBを使うLambdaから
インターネットにも接続したいです

04

サーバーレスと非機能要件(ネットワークなど)

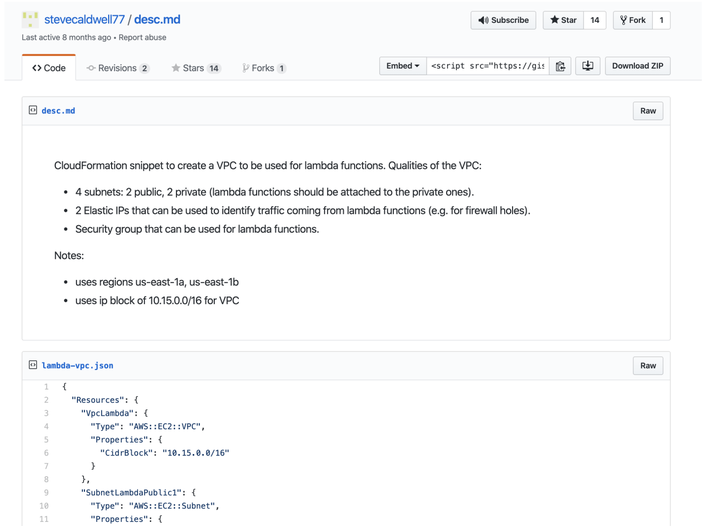
Q6: RDBを使うLambdaからインターネットにも接続したいです

- 一つのLamda内でRDB内のデータを参照/更新しつつ外部APIを叩くようなパターン
- NAT Gateway が必要になりこれがなかなか面倒、費用も気になる…
  - AWS Blogの以下記事で手順は網羅的に解説されているのであとは実装

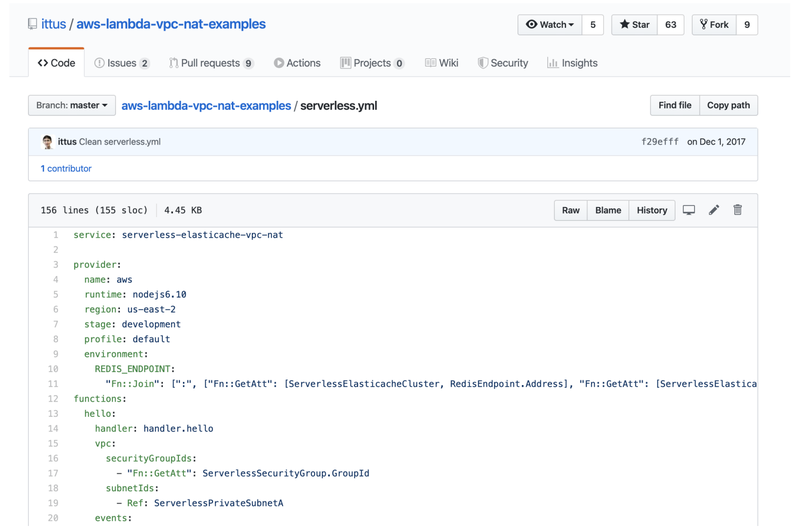
04 サーバーレスと非機能要件(ネットワークなど)

Q6: RDBを使うLambdaからインターネットにも接続したいです

- serverless経由でふんわりと使っていたCloudFormationに真摯に向き合う時が来た
- 以下のスニペット(CFnとserverless)を参考に最小構成を作成



https://gist.github.com/stevecaldwell77/b2e9f9179b4e52235fcee24cf37c8dc9



https://github.com/ittus/aws-lambda-vpc-nat-examples/blob/master/serverless.yml

04

サーバーレスと非機能要件(ネットワークなど)

Q7: 夜間停止可能な
EC2インスタンスを停止したいです

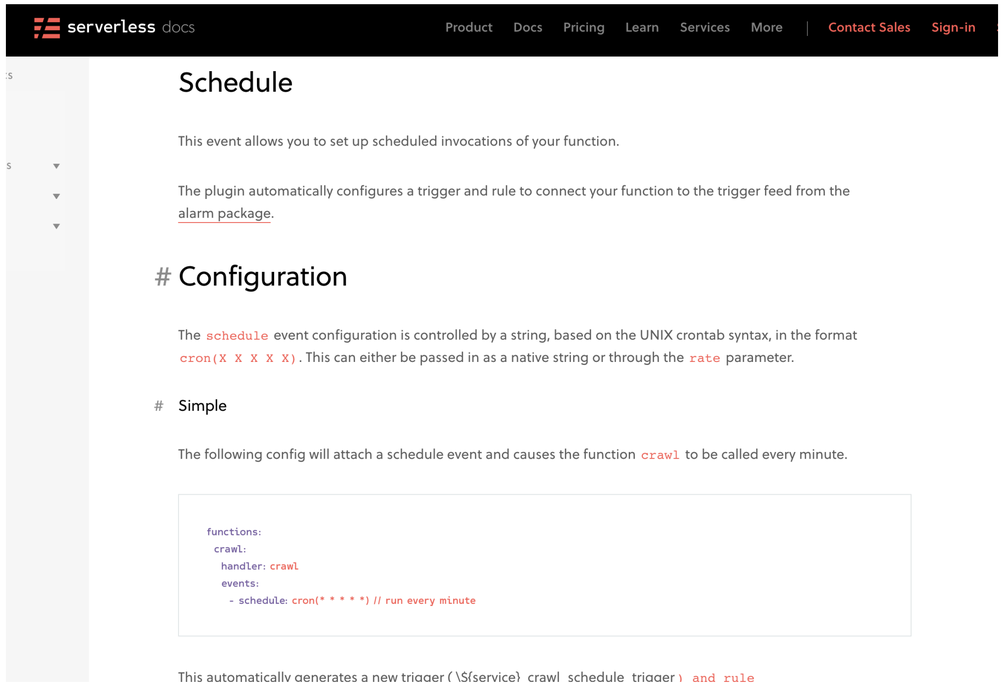
04

サーバーレスと非機能要件(ネットワークなど)

Q7: 夜間停止可能な EC2インスタンスを停止したいです

- 費用削減が目的、特にGPUインスタンスなど高価なものを対象にしたい
- serverlessではschedule起動のイベントをcronで記述する
  - AWSではCloudwatch Eventsが作成されてLambdaをトリガーしている

```
  startEC2:
    handler: src/functions/EC2Start.index
    events:
      - schedule: cron(0 1 * * ? *)
  stopEC2:
    handler: src/functions/EC2Stop.index
    events:
      - schedule: cron(0 2 * * ? *)
```



04

サーバーレスと非機能要件(ネットワークなど)

その他の小ネタ

- 環境名のsuffixが原因でproduction環境構築のタイミングでLambdaの文字数制限を超えた
  - リリース間際になり、production環境を準備して…
- API Gateway、Lambdaはそれぞれにペイロード上限を持っている
  - API Gateway(10MB)、Lambda(6MB)…
- ミドルウェアを多用するためUnit Test用Dockerコンテナのメモリ消費が大きく、Circle CIの標準メモリサイズを超えた
  - 無念の一部test skip…
- [PR]"終わらないLambda実行" というLambdaでハマったパターンネタをアドベントカレンダーで書きます

04 サーバーレスと非機能要件(ネットワークなど)

まとめ

- 実際にアプリケーション開発していくと非機能的な要件が出てくる
  - 知らないと”これはどこのレイヤーで行うもの?”となりがち
  - クラウドインフラの知識と、実際にserverlessで設定するためのフレームワークの知識、両方が必要
  - ドキュメント読み込み業が多くなりがち

まとめ

まとめ

- サーバーレス、良かった
  - 開発者がアプリケーションコードを書くのに集中できた
  - 下準備をすればチーム開発もできる
  - 非機能的な要件を満たすためにはクラウドインフラ、フレームワーク両方の知識を貯めて取り組む必要があった
- 今後の課題感
  - serverless deployに乗り切らない部分はグルーコード的に各チームで作り込んでいる状況では?
  - 本当はアルゴリズムモジュールもサーバーレスにのせたいな…