サーバレス時代の負荷テスト戦略

～CircleCIで実現する継続的負荷テストとチューニングTips～

2020.01.22 Serverless Meetup Tokyo #15

淡路大輔 / INTEC

@gee0awa

好きな技術

Serverless / React Native

aws certified Solutions Architect Associate

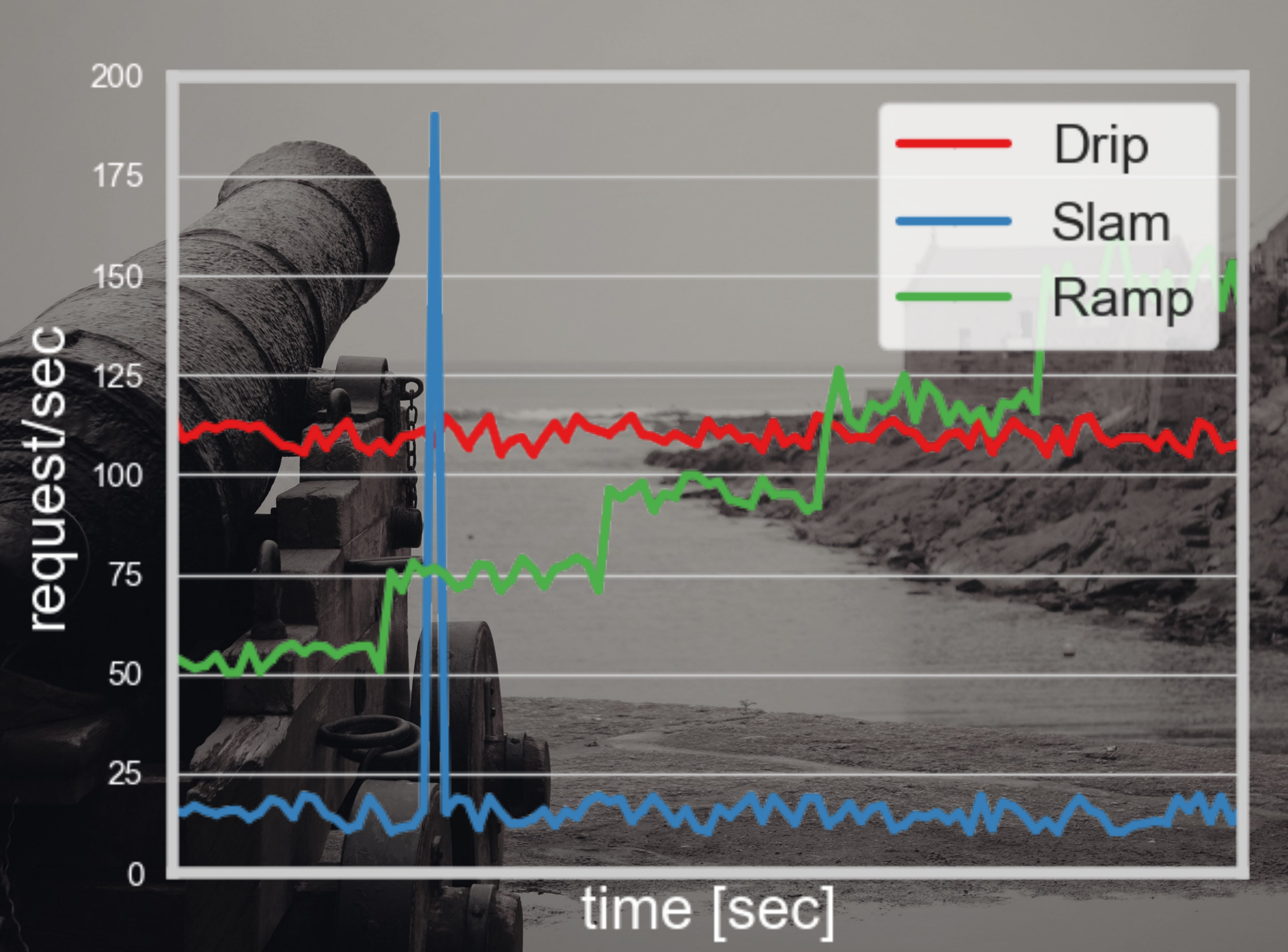
Licensed Scrum Master LSM

scruminc.

負荷テストとサーバレス

Elasticity

Performance, Scalability, Resilience


200
175
150
125
100
75
50
25
0
requests/sec
time [sec]
Drip
Slam
Ramp


App

EC2

App

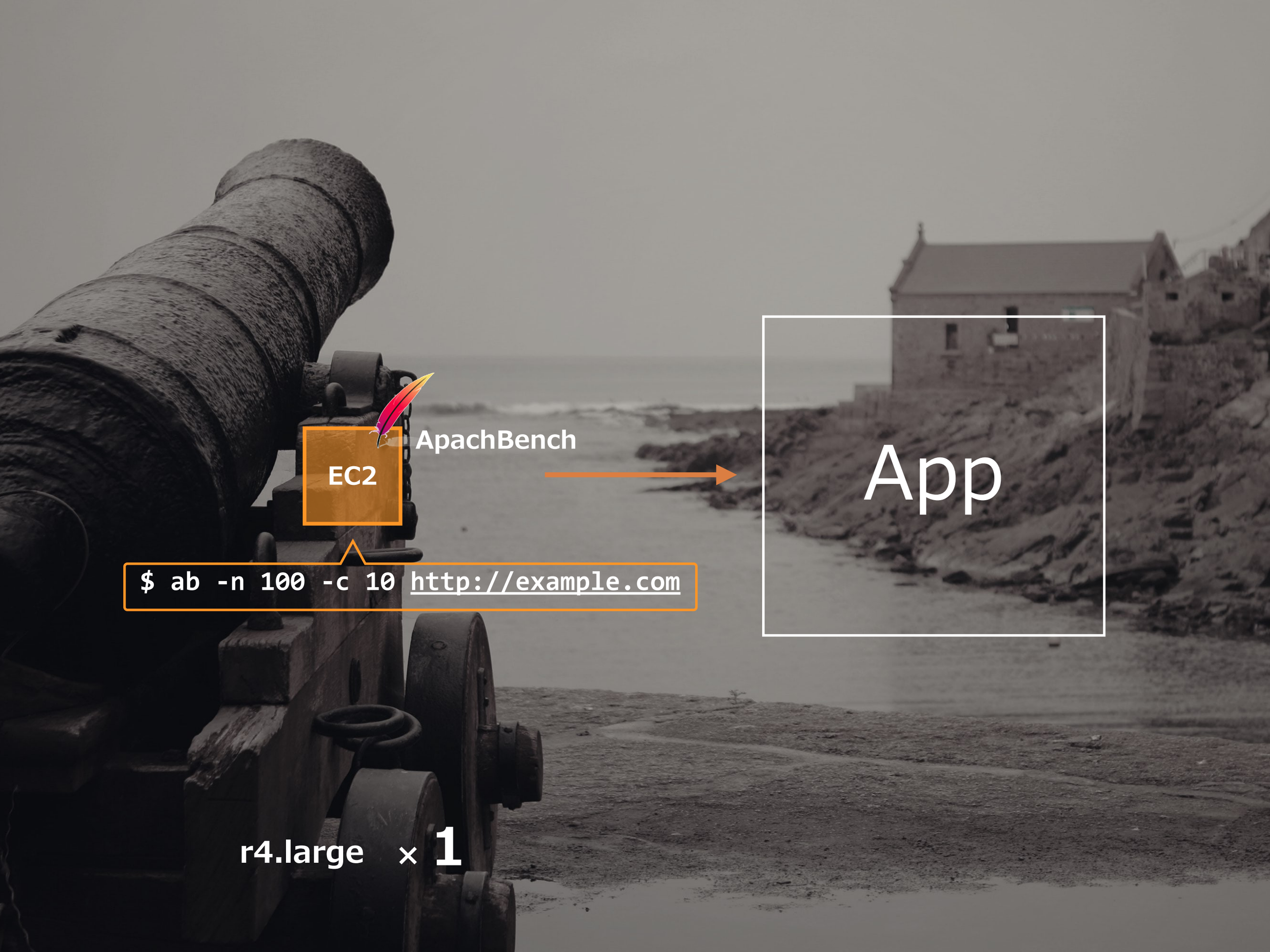
r4.large × 1

ApachBench

EC2

App

r4.large × 1


ApachBench
EC2
$ ab -n 100 -c 10 http://example.com
App
r4.large × 1

EC2

EC2

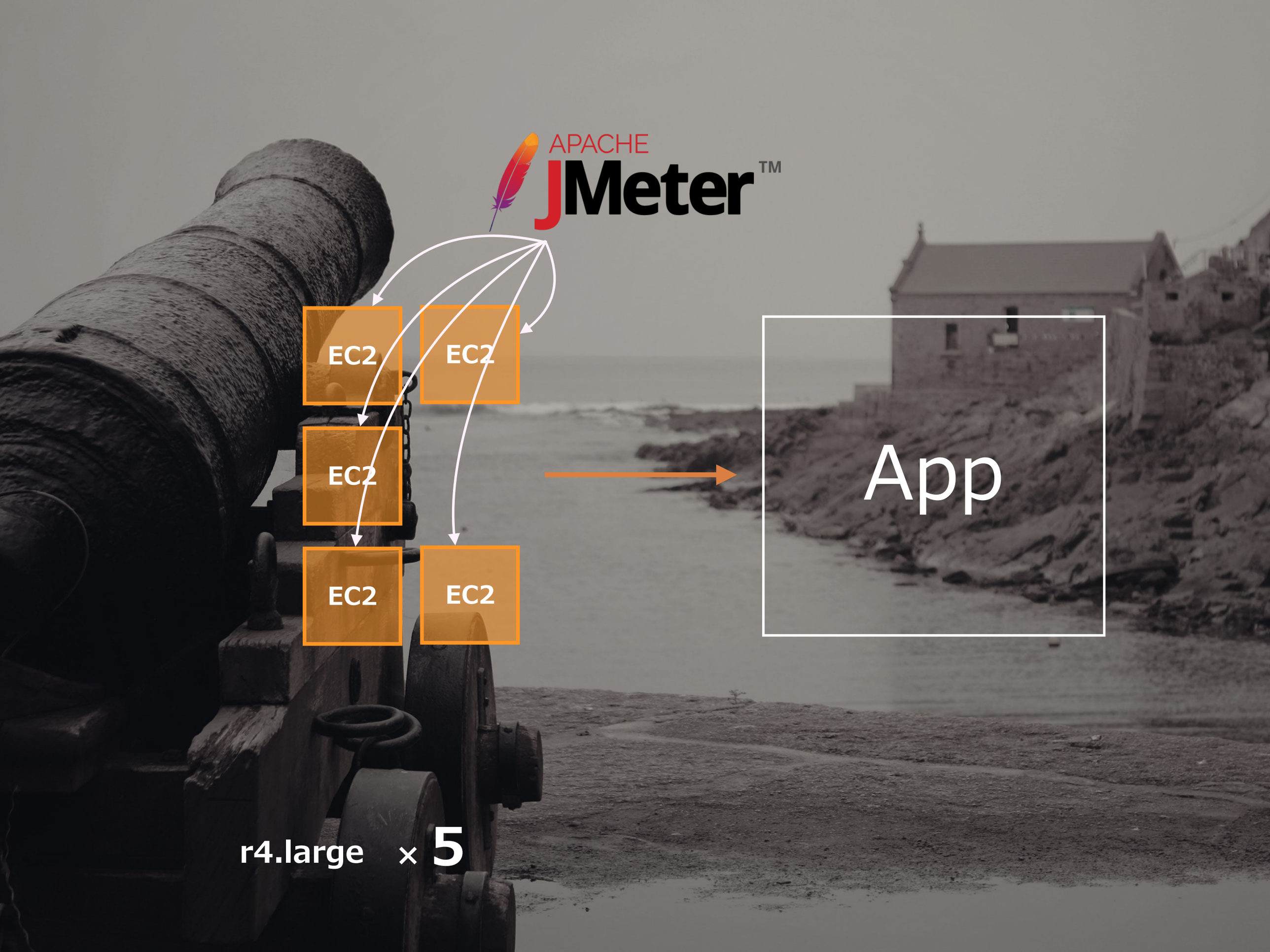
EC2

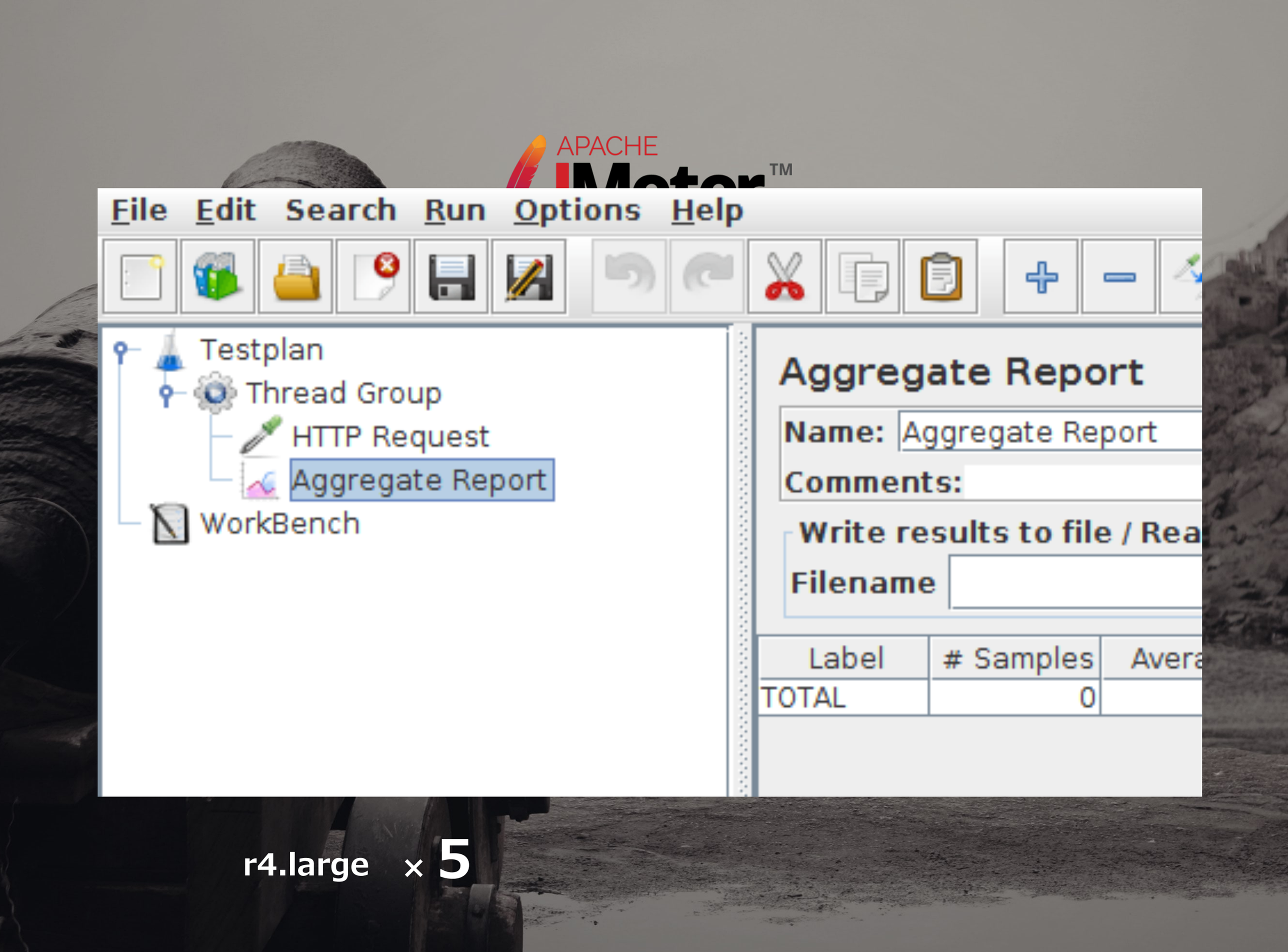
EC2

EC2

App

r4.large × 5


APACHE
JMeter™
EC2
EC2
EC2
EC2
EC2
App
r4.large × 5


APACHE
JMeter™
File Edit Search Run Options Help
Testplan
Thread Group
HTTP Request
Aggregate Report
WorkBench
Aggregate Report
Name: Aggregate Report
Comments:
Write results to file / Rea
Filename
Label
Samples
Avera
TOTAL
0
r4.large × 5

APACHE JMeter™

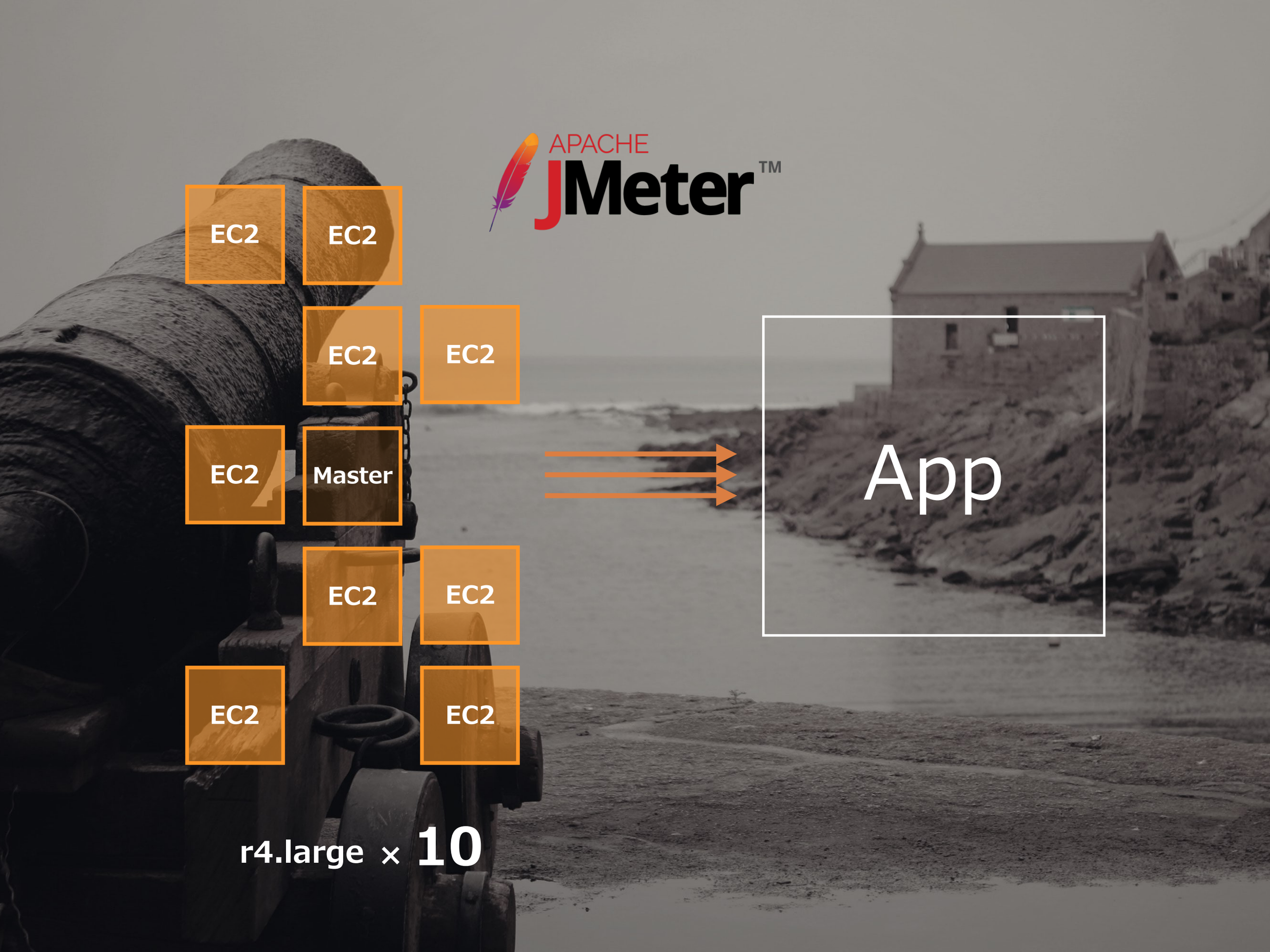
EC2 EC2

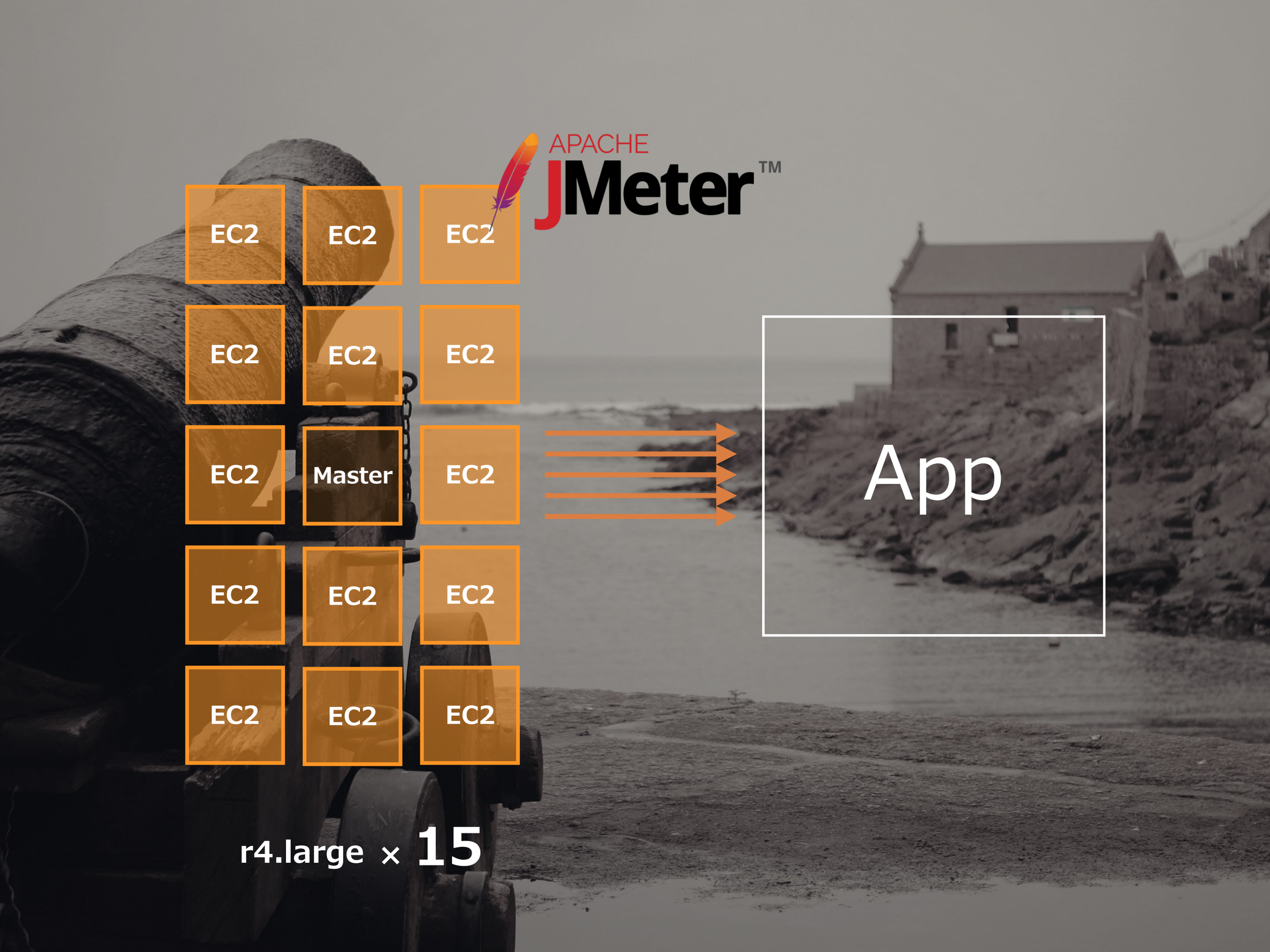
Master

EC2 EC2

App

r4.large × 5


APACHE
JMeter™
EC2
EC2
EC2
EC2
EC2
Master
EC2
EC2
EC2
EC2
App
r4.large × 10


APACHE
JMeter™
EC2
EC2
EC2
EC2
EC2
EC2
EC2
Master
EC2
EC2
EC2
EC2
EC2
EC2
EC2
EC2
EC2
App
r4.large × 15

課題

- 負荷に応じて Slave 台数をスケールさせたい
- サーバの用意に時間がかかる
- GUIでシナリオ作るとメンテしづらい
- EC2 高い・・・💰

Simple

もっと簡単に導入したい

Performance

大量リクエストを生成したい

Cost

お金はかけたくない


⚡Artillery⚡

README.md

Artillery is a modern, powerful & easy-to-use **load testing** and **functional testing** toolkit. Use it to ship **scalable** backends, APIs & services that stay **performant** & **resilient** under high load.

Artillery has a strong focus on developer happiness & ease of use, and a batteries-included philosophy.

Use Cases

- **Peak traffic testing** - ensure your e-commerce backend, IoT service or web API can handle max traffic
- **Pre-launch load testing** - for new websites, mobile app backends, web APIs etc
- **Continuous performance testing** for new microservices as they are being built
- **Preventing performance regressions** - stop performance regressions due to new code or config changes before they are shipped to users.
- **Help profile & debug** common issues such as extensive GC pauses, memory leaks, improperly configured resource pools etc

Features

- **Multiple protocols**: Load test HTTP, WebSocket, Socket.io, Kinesis, HLS and more.
- **Scenarios**: Support for complex *scenarios* to test multi-step interactions in your API or web app (great for ecommerce, transactional APIs, game servers etc).
- **Load testing & Functional testing**: reuse the same scenario definitions to run performance tests or functional tests on your API or backend.
- **Performance metrics**: get detailed performance metrics (latency, requests per second, concurrency, throughput).
- **Scriptable**: write custom logic in JS, using any of the thousands of useful `npm` modules.
- **Integrations**: `statsd` support out of the box for real-time reporting (integrate with Datadog, Librato, InfluxDB etc).
- **Extensible**: write custom reporters, custom plugins, custom protocol engines etc.
- **and more!** HTML reports, nice CLI, parameterization with CSV files.

- **Source**: https://github.com/artilleryio/artillery
- **Issues**: https://github.com/artilleryio/artillery/issues
- **Chat**: https://gitter.im/shoreditch-ops/artillery
- **Community**: https://spectrum.chat/artillery-io
- **Docs**: https://artillery.io/docs/

Node.js製のシンプルな負荷テストツール

Artillery scenario

YAMLファイルに宣言的にシナリオを記述する

```yaml
config:
  target: 'https://artillery.io'
  phases:
    - duration: 60
      arrivalRate: 20
scenarios:
  - flow:
    - get:
        url: "/docs"
```

60秒間、20人の仮想ユーザがリクエスト

Artillery

EC2

App

```
$ artillery run senario.yml
```

Community Plugins & Engines

- **Chance Plugin** - https://github.com/beyonk-adventures/artillery-plugin-chance - generate realistic randomized data for your tests with Chance.js
- **Faker Plugin** - https://www.npmjs.com/package/artillery-plugin-faker - generate data for your tests with Faker.js
- **UUID Plugin** - https://www.npmjs.com/package/artillery-plugin-uuid - generate UUID (version 4) variables for use inyour test script
- **Kafka Engine** - https://github.com/flentini/artillery-engine-kafka - load test Kafka with Artillery
- **Kinesis Stream Engine** - https://www.npmjs.com/package/artillery-engine-kinesisstream - load test Kinesis streams with Artillery
- **AWS Lambda Engine** - https://www.npmjs.com/package/artillery-engine-lambda - load test AWS Lambda with Artillery
- **Splunk Reporter** - https://www.npmjs.com/package/artillery-plugin-splunk - send metrics generated by Artillery to a Splunk HTTP Collector
- **AWS SNS Reporter** - https://www.npmjs.com/package/artillery-plugin-sns - publish Artillery test metrics to an SNS topic
- **Prometheus Reporter** - https://www.npmjs.com/package/artillery-plugin-prometheus - record Artillery metrics into Prometheus via a push gateway

CloudWatch, DataDog, aws-sigv4 etc..

Artillery

EC2

App

```
$ artillery run senario.yml
```


Artillery Pro
ECS Cluster
App
$ artillery run senario.yml

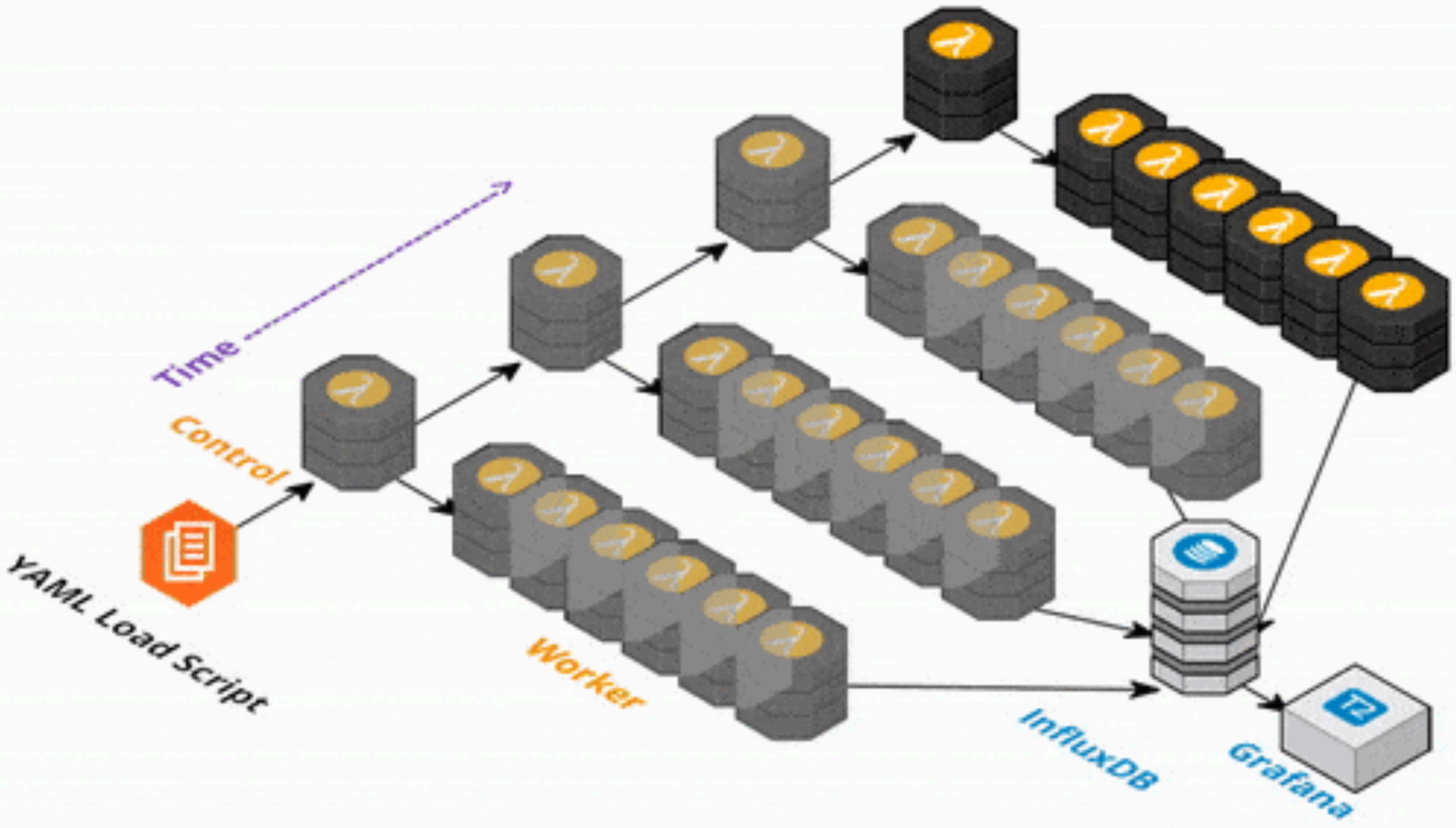
Artillery + Lambda


serverless

serverless-artillery

https://github.com/Nordstrom/serverless-artillery

serverless-artillery



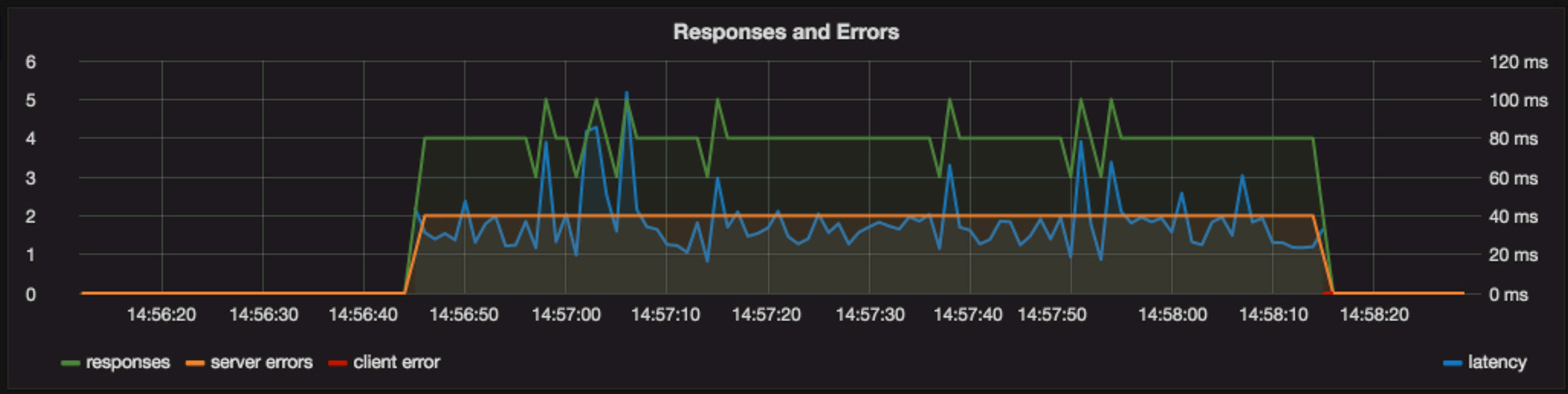
https://github.com/Nordstrom/serverless-artillery

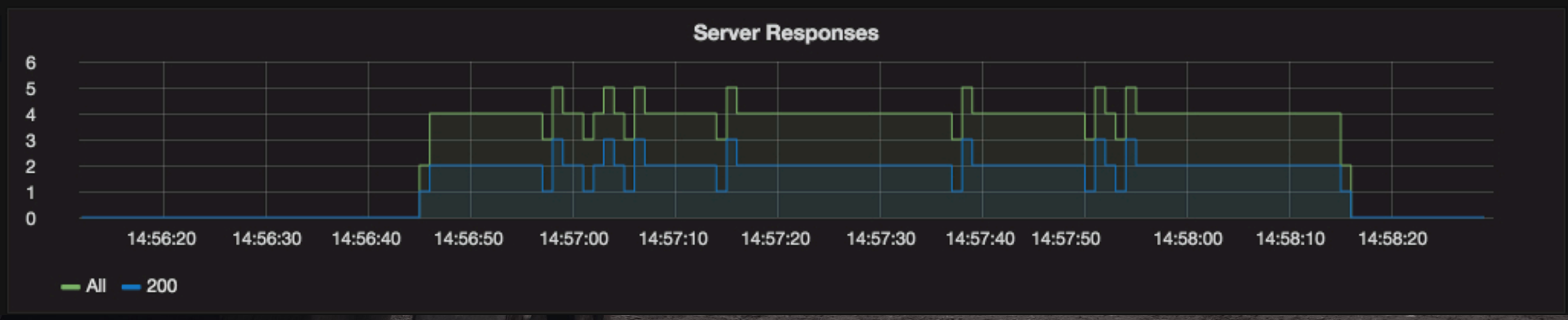
Load Test Results

Zoom Out

Oct 25, 2016 14:56:11 to Oct...

Test Name: a09y-smoke-load-test





InfluxDBに結果を保存、Grafanaで可視化

serverless-artillery

install

```
$ npm install -g serverless-artillery
```

deploy

```
$ slsart deploy –stage dev
```

invoke

```
$ slsart invoke –stage dev
```

remove

```
$ slsart remove
```

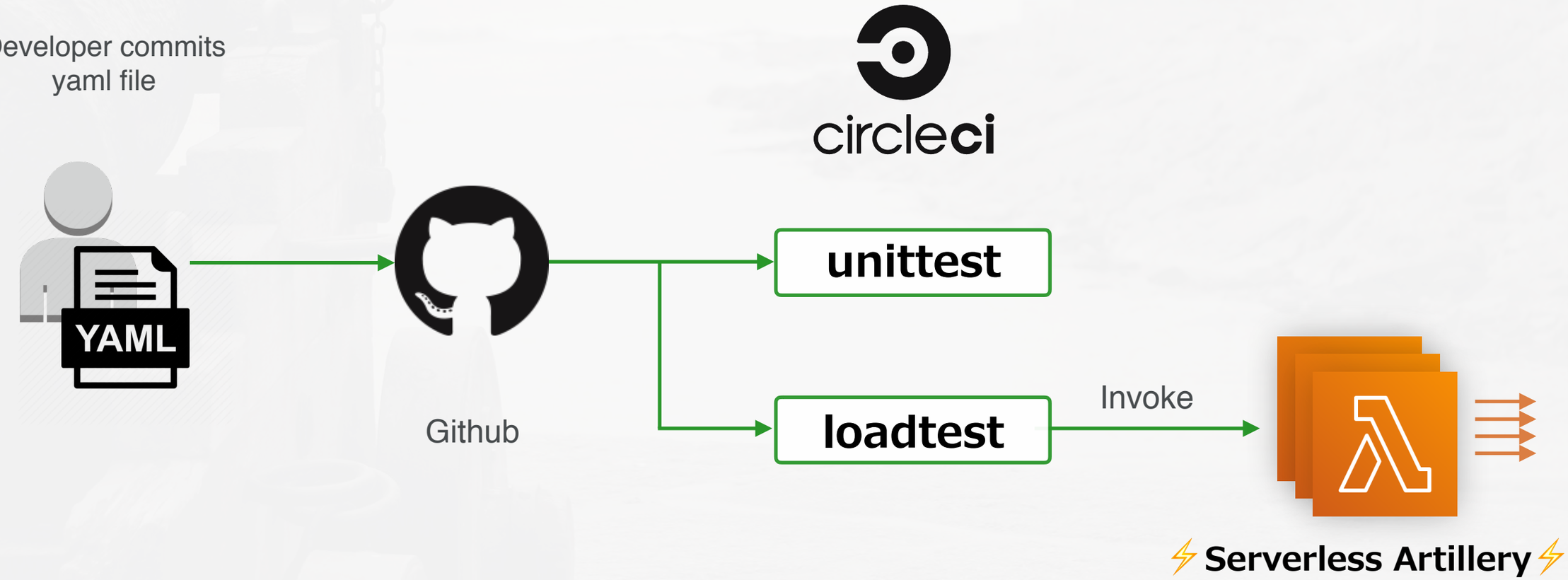

Simple
YAMLによる宣言的シナリオ
Performance
負荷に応じたオートスケール
Cost
Lambdaだから維持費ゼロ

継続的負荷テスト

開発プロセスの早い段階から性能検証を行う

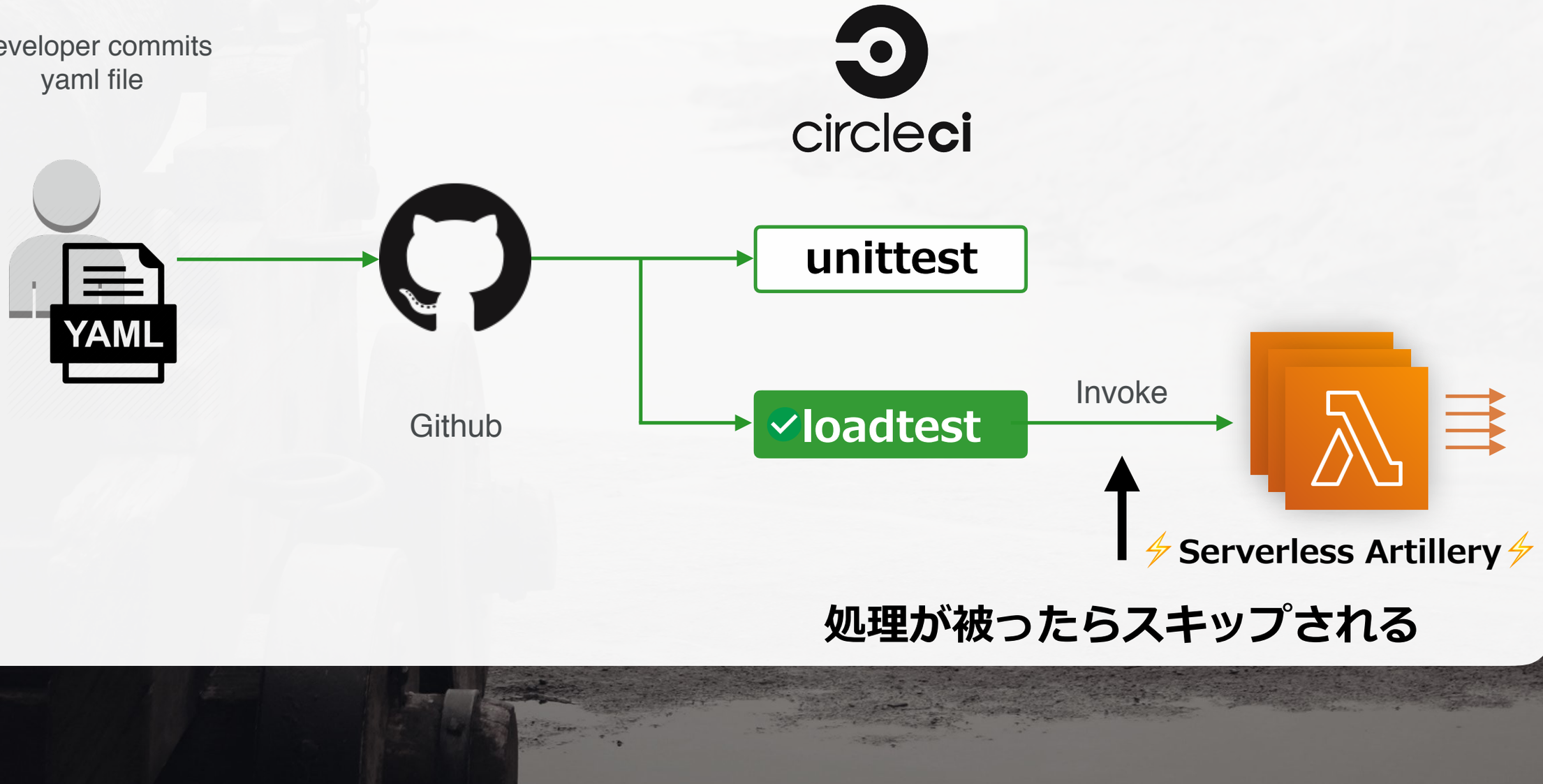
継続的負荷テスト

CircleCIから負荷テストを実行する



継続的負荷テスト

CircleCIから負荷テストを実行する

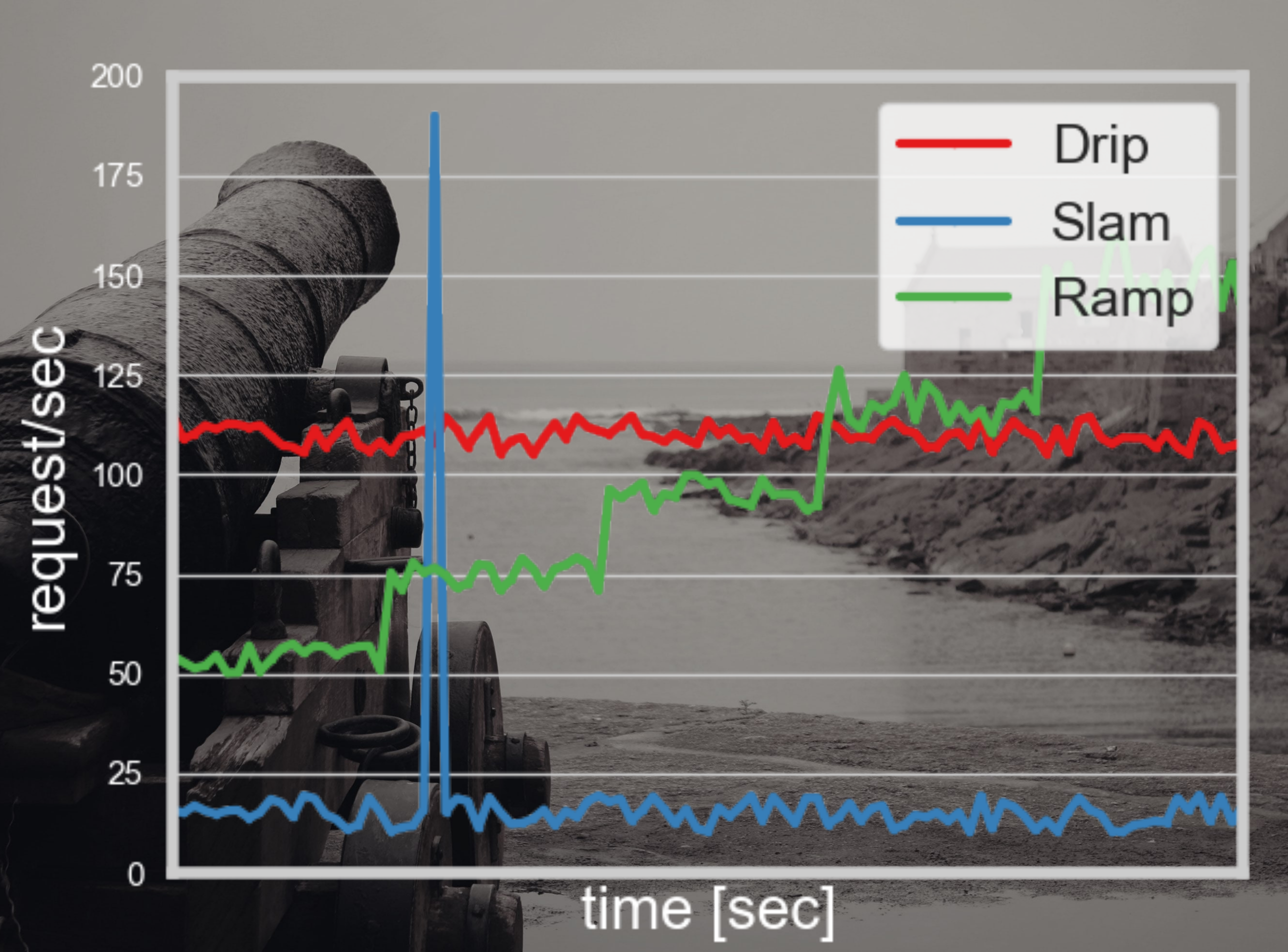


処理が被ったらスキップされる

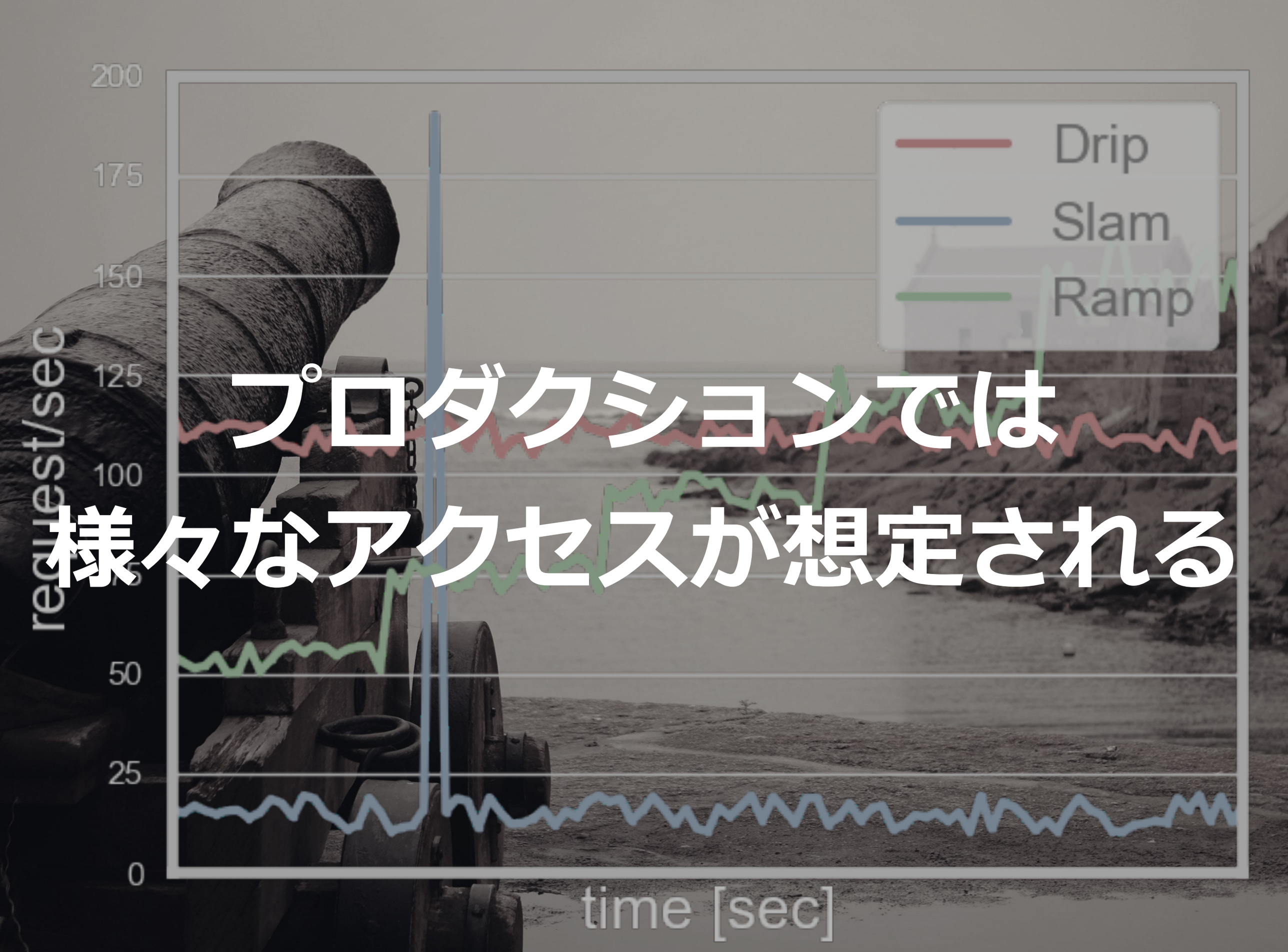
.circleci/config.yml

```
loadtest:
  docker:
    - image: circleci/node:7.10
  working_directory: ~/repo
  steps:
    - checkout
    - run: npm --prefix ./loadtester install
    - run: npm --prefix ./loadtester run loadtest
```

シンプルなインタフェースで
実行できるようにしておく


200
175
150
125
100
75
50
25
0
requests/sec
time [sec]
Drip
Slam
Ramp

プロダクションでは
様々なアクセスが想定される



負荷にランダム性を持たせる

Artillery scenario

YAMLファイルに宣言的にシナリオを記述する

```yaml
config:
  target: 'https://artillery.io'
  phases:
    - duration: 60
      arrivalRate: 20
scenarios:
  - flow:
    - get:
        url: "/docs"
```

この部分を
ランダムに変化させる

60秒間、20人の仮想ユーザがリクエスト

シナリオファイルの負荷量を書き換える

ChaosEngineering!!

generateSenario.js

duration, arrivalRate, phase数の最大・最小値だけ指定して
ランダムな負荷を生成できるようにシナリオファイルを書き換える

.circleci/config.yml

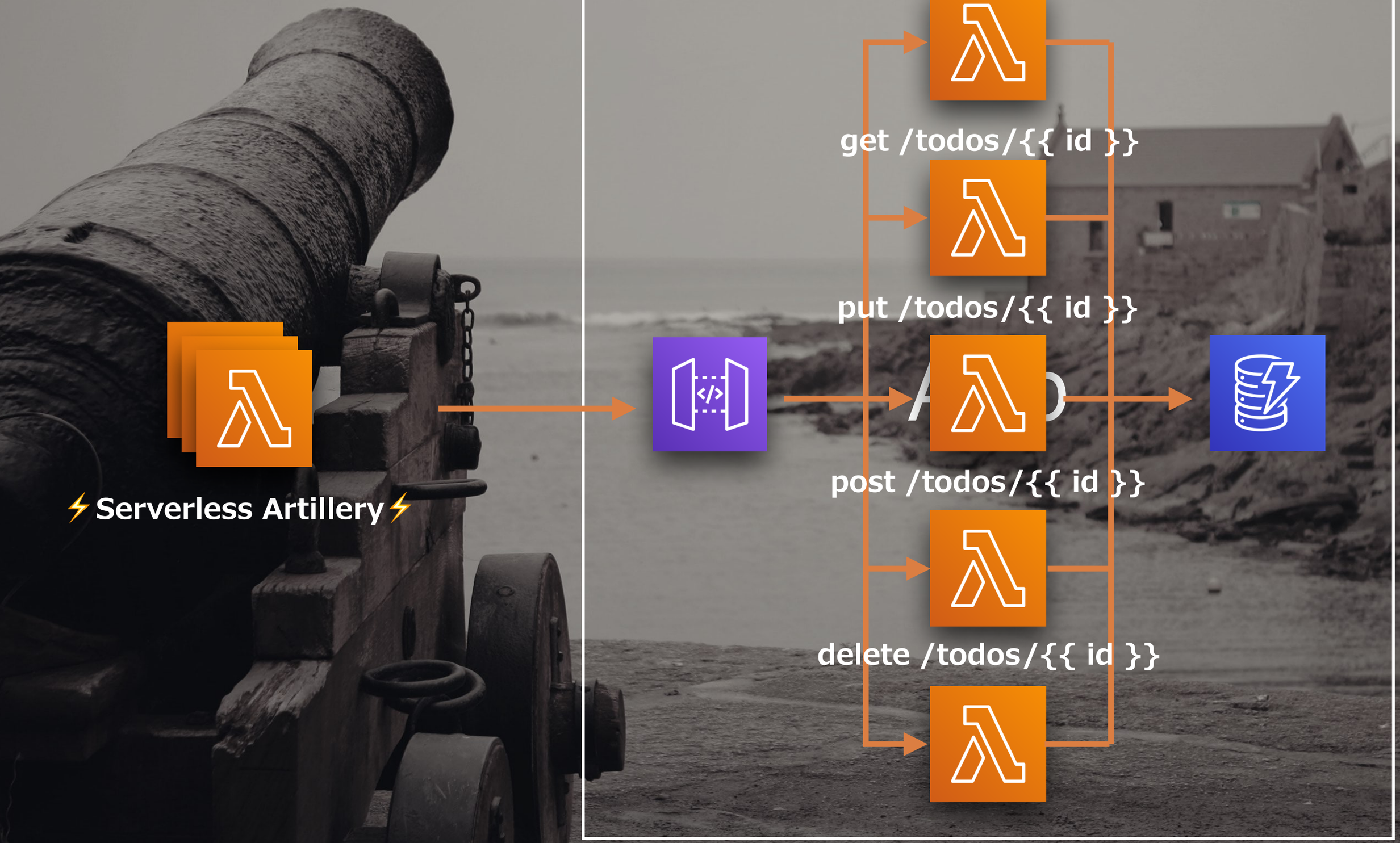
```
loadtest:
  docker:
    - image: circleci/node:7.10
  working_directory: ~/repo
  steps:
    - checkout
    - run: npm --prefix ./loadtester install
    - run: npm --prefix ./loadtester run generate:senario
    - run: npm --prefix ./loadtester run loadtest
```

負荷をランダムにする

負荷テストの環境下で
アプリケーションを開発する


App
⚡Serverless Artillery⚡

TODOリスト アプリケーション



https://github.com/serverless/examples/tree/master/aws-node-rest-api-with-dynamodb-and-offline

TODOリスト アプリケーション
serverless examples の Node.js REST API

Search or jump to... / Pull requests Issues Marketplace Explore

serverless / examples

Watch 249 Star 6.7k Fork 2.4k

Code Issues 79 Pull requests 24 Actions Projects 0 Wiki Security Insights

Branch: master examples / aws-node-rest-api-with-dynamodb-and-offline / Create new file Upload files Find file History

adamatan Use DynamoDB On-Demand ... Latest commit 32a9dd3 2019年9月11日 2:31 JST

..

offline/migrations	renamed example	3 years ago
todos	Fix rest-api-with-dynamodb-and-offline	2 years ago
.gitignore	renamed example	3 years ago
README.md	all example README files updated	last year
package.json	Update package.json	last year
serverless.yml	Use DynamoDB On-Demand	4 months ago

README.md

Serverless REST API with DynamoDB and offline support

This example demonstrates how to run a service locally, using the serverless-offline plugin. It provides a REST API to manage Todos stored in a DynamoDB, similar to the aws-node-rest-api-with-dynamodb example. A local DynamoDB instance is provided by the serverless-dynamodb-local plugin.

Use-case

https://github.com/serverless/examples/tree/master/aws-node-rest-api-with-dynamodb-and-offline

TODOリスト アプリケーション
Artilleryのシナリオ

```
config:
  target: https://35vbyy1dxd.execute-api.ap-northeast-1.amazonaws.com/dev
  phases:
    - duration: 10
      arrivalRate: 10

scenarios:
  - flow:
      - post:                          ← TODOリストに登録
          url: /todos
          json:
            text: "my todo"
          capture:
            json: "$.id"               ← Responseのjsonに含まれるidの値を保持
            as: "id"
    - get:
        url: /todos
    - get:
        url: /todos/{{ id }}           ← idを指定して1件取得
    - put:
        url: /todos/{{ id }}           ← idを指定して1件更新
        json:
          text: "my todo finished"
          checked: true
    - delete:                          ← idを指定して1件削除
        url: /todos/{{ id }}
        json:
          id: {{ id }}
```

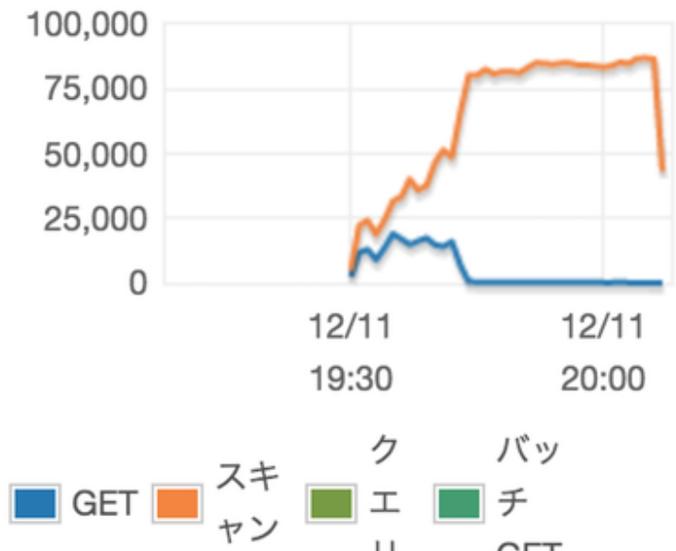
結果

DynamoDB

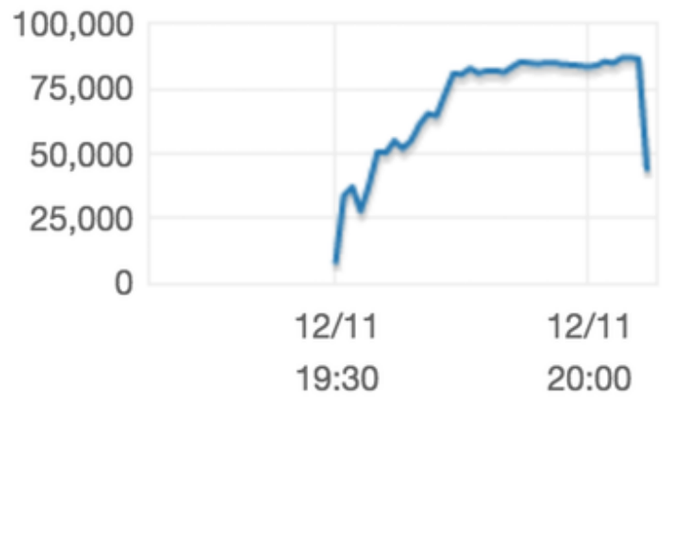
キャパシティー: テーブル

読み込みキャパシティー (ユニット数/秒 - 平均 1 分)

スロットル読み込みリクエスト (カウント)



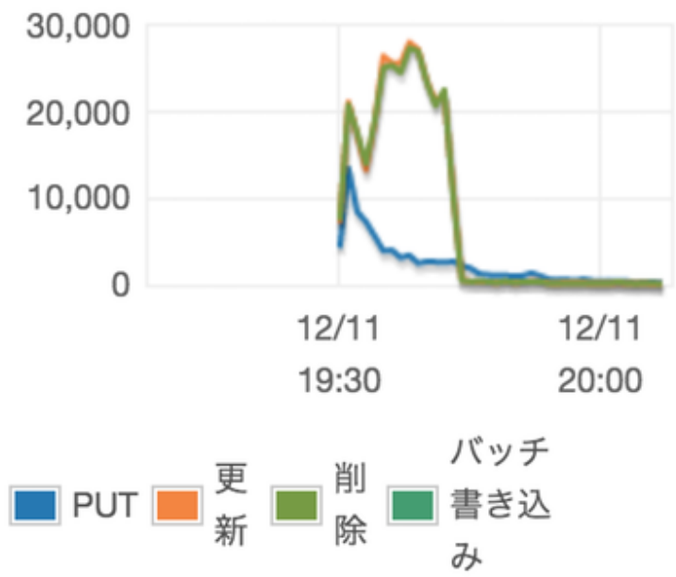
スロットル読み込みイベント (カウント)



キャパシティユニットが全然足りない

書き込みキャパシティー (ユニット数/秒 - 平均 1 分)

スロットル書き込みリクエスト (カウント)



スロットル書き込みイベント (カウント)

TODO一覧 ソース

```
'use strict';

const dynamodb = require('./dynamodb');

module.exports.list = (event, context, callback) => {
  const params = {
    TableName: process.env.DYNAMODB_TABLE
  };

  // fetch all todos from the database
  dynamodb.scan(params, (error, result) => {
    // handle potential errors
    if (error) {
      console.error(error);
      callback(null, {
        statusCode: error.statusCode || 501,
        headers: { 'Content-Type': 'text/plain' },
        body: 'Couldn\'t fetch the todo item.',
      });
      return;
    }

    // create a response
    const response = {
      statusCode: 200,
      body: JSON.stringify(result.Items),
    };
    callback(null, response);
  });
};
```

TODO一覧 ソース

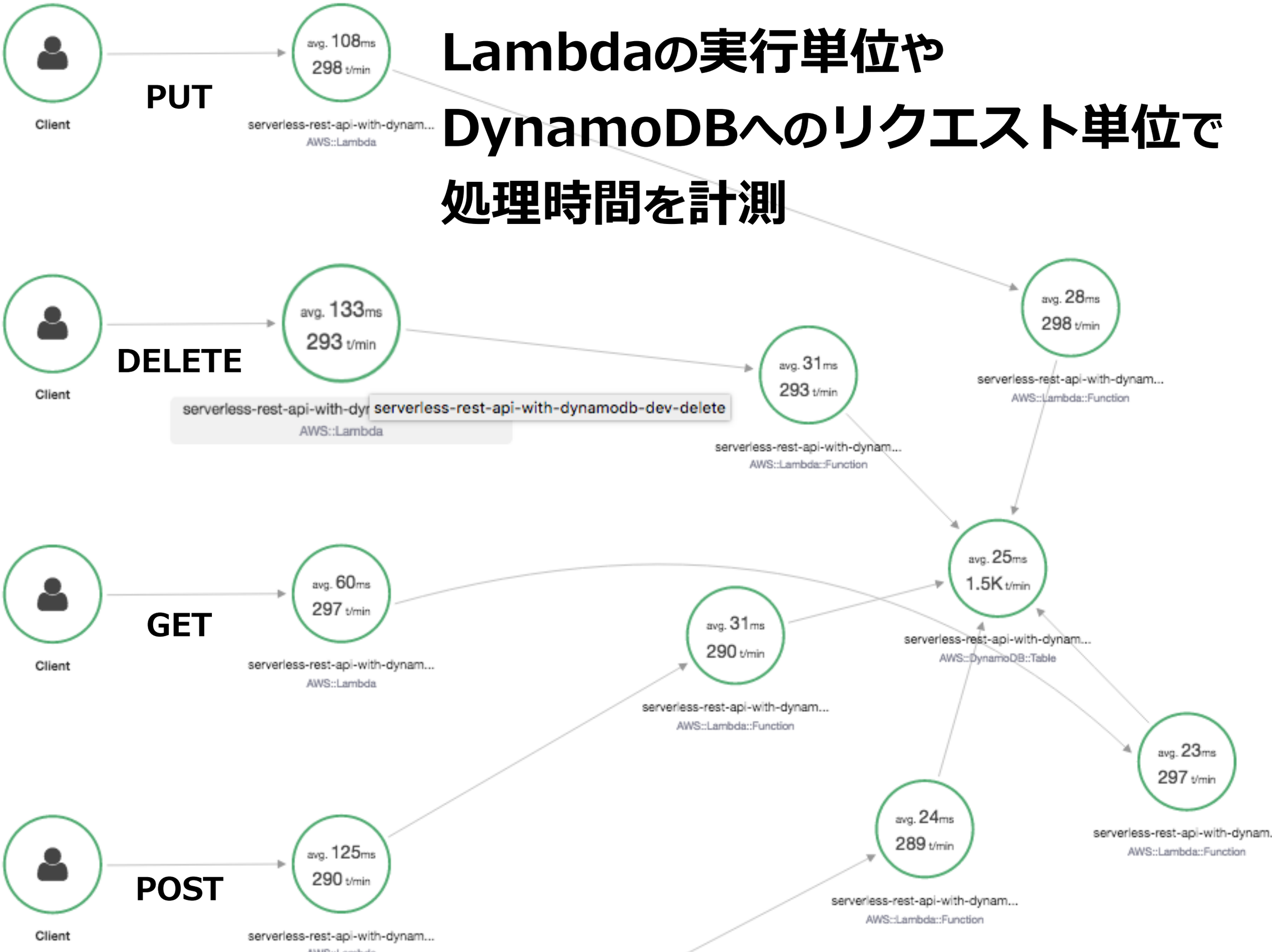
```
'use strict';

const dynamodb = require('./dynamodb');

module.exports.list = (event, context, callback) => {
  const params = {
    TableName: process.env.DYNAMODB_TABLE
  };

  // fetch all todos from the database
  dynamodb.scan(params, (error, result) => {
    // handle potential errors
    if (error) {
      console.error(error);
      callback(null, {
        statusCode: error.statusCode || 501,
        headers: { 'Content-Type': 'text/plain' },
        body: 'Couldn\'t fetch the todo item.',
      });
      return;
    }

    // create a response
    const response = {
      statusCode: 200,
      body: JSON.stringify(result.Items),
    };
    callback(null, response);
  });
};
```

全件スキャンになってる！！

監視

AWS X-Ray

Lambdaの実行単位や DynamoDBへのリクエスト単位で 処理時間を計測



監視

AWS X-Ray

GetServiceGraph API

Response time
4xx (error)
429 (throttling)
5xx (failure)

Lambda
(cron)

ALERT

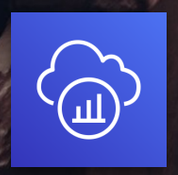
Slack

サーバレス時代の負荷テスト戦略

まとめ

⚡Serverless Artillery⚡

CircleCI



AWS X-Ray

宣言的シナリオ・低コスト・高負荷

継続的負荷テスト・ランダム性

分散トレーシング・監視

ご静聴ありがとうございました