しむそく Radio

Amazon CloudWatch Synthetics now supports Playwright runtime to create canaries with NodeJS の使い所を考える

tubone24
(@meitante1conan)

自己紹介

Yu Otsubo (tubone24)

特徴

@meitante1conan

ゆるふわエンジニア

犬と遊んでいる

自己紹介

Yu Otsubo (tubone24)

AWS re:Invent 2024 参加！！

特徴

nan

ゆるふわエンジニア

犬と遊んでいる

そんな私がお送りするのが

2024/11/21に出たこちらのアップデート

Amazon CloudWatch Synthetics now supports Playwright runtime to create canaries with NodeJS

Posted on: Nov 21, 2024

CloudWatch Synthetics, which continuously monitors web applications and APIs by running scripted canaries to help you detect issues before they impact end-users, now supports the Playwright framework for creating NodeJS canaries enabling comprehensive monitoring and diagnosis of complex user journeys and issues that are challenging to automate with other frameworks.

Playwright is an open-source automation library for testing web applications. You can now create multi-tab workflows in a canary using the Playwright runtime which comes with the advantage of troubleshooting failed runs with logs stored directly to CloudWatch Logs database in your AWS account. This replaces the previous method of storing logs as text files and enables you to leverage CloudWatch Logs Insights for query-based filtering, aggregation, and pattern analysis. You can now query CloudWatch logs for your canaries using the canary run ID or step name, making the troubleshooting process faster and more precise than one relying on timestamp correlation for searching logs. Playwright-based canaries also generate artifacts like reports, metrics, and HAR files, even when canaries times out, ensuring you have the

Amazon CloudWatch Synthetics ってなんだよ...

Amazon CloudWatch Syntheticsとは

WebアプリケーションやAPIエンドポイントに対して、
ユーザーと同じアクションを模倣して実行し、
パフォーマンスや可用性を監視するマネージドサービス

- Canaryと呼ばれるスクリプトを使用して監視を実行
- Node.jsまたはPythonランタイムのLambda関数として実行
- PuppeteerやSeleniumを使用してヘッドレスブラウザで監視
- CloudWatch RUMやApplication Signals(X-Ray)との統合も可能

Amazon CloudWatch Syntheticsとは

WebアプリケーションやAPIエンドポイントに対して、
ユーザーと同じアクションを模倣して実行し、

難しいね....

- Node.jsまたはPythonランタイムのLambda関数として実行
- PuppeteerやSeleniumを使用してヘッドレスブラウザで監視
- CloudWatch RUMやApplication Signals(X-Ray)との統合も可能

Amazon CloudWatch Syntheticsとは

Amazon CloudWatch Synthetics Canary

人工的な

※覚え方、翻訳の仕方は諸説あります..多分

Amazon CloudWatch Syntheticsとは

Amazon CloudWatch Synthetics Canary

人工的なカナリア

※覚え方、翻訳の仕方は諸説あります..多分

Amazon CloudWatch Syntheticsとは

Amazon CloudWatch Synthetics Canary

人工的な

シンセサイザーも「波形を合成」した
人工的な音

カナリア

炭鉱の有毒ガス検知に
カナリアが使われてたらしい

Amazon CloudWatch Syntheticsとは

要はユーザーリクエストを模擬して

Amazon CloudWatch Synthetics Canary

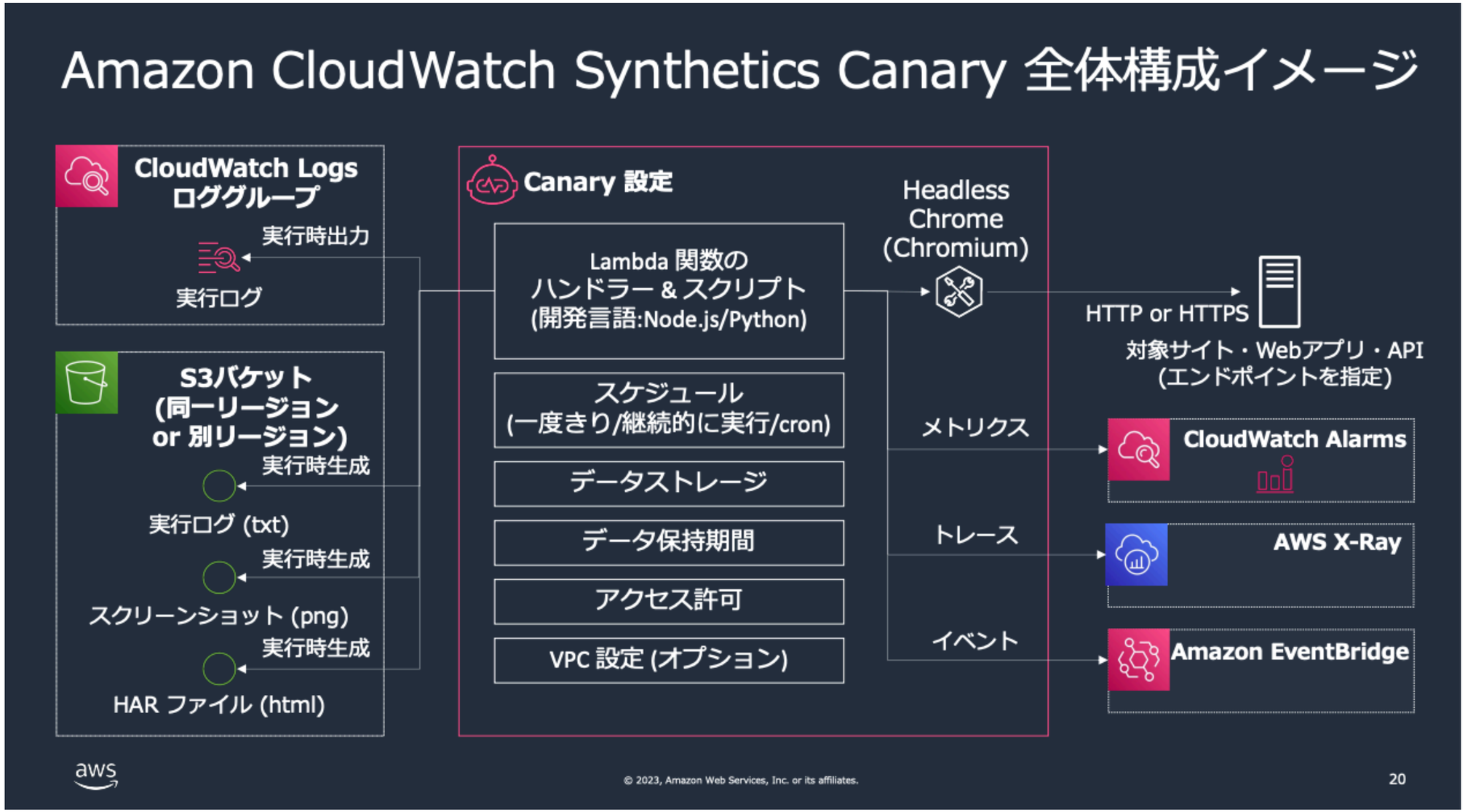
アプリの正常性を

監視する

シンセサイザーも「波形を合成」した
人工的な音

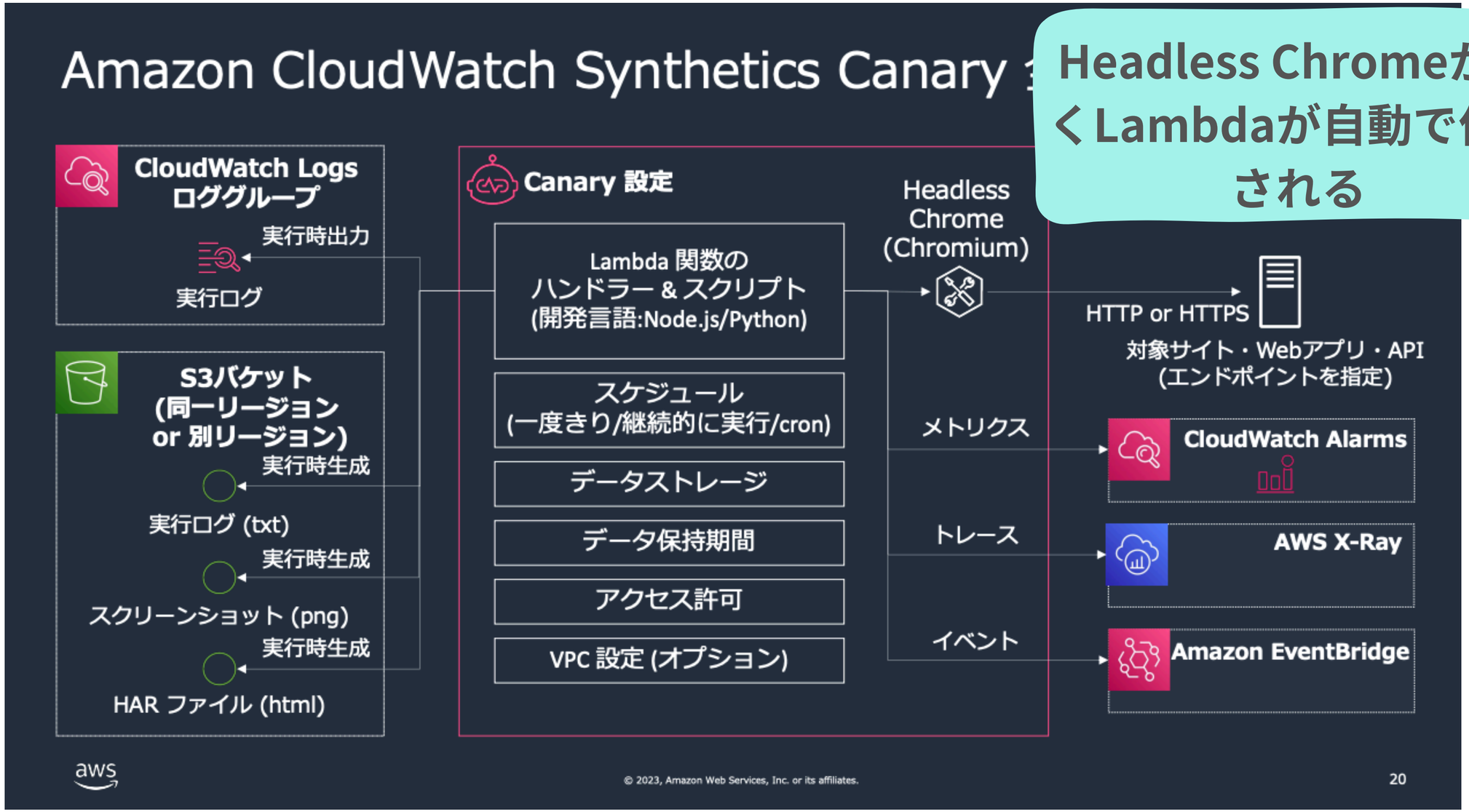
カナリアが使われてたらしい

これがSynthetics Canaryだ！！！



（引用元：【AWS Black Belt Online Seminar】 Amazon CloudWatch Synthetics）

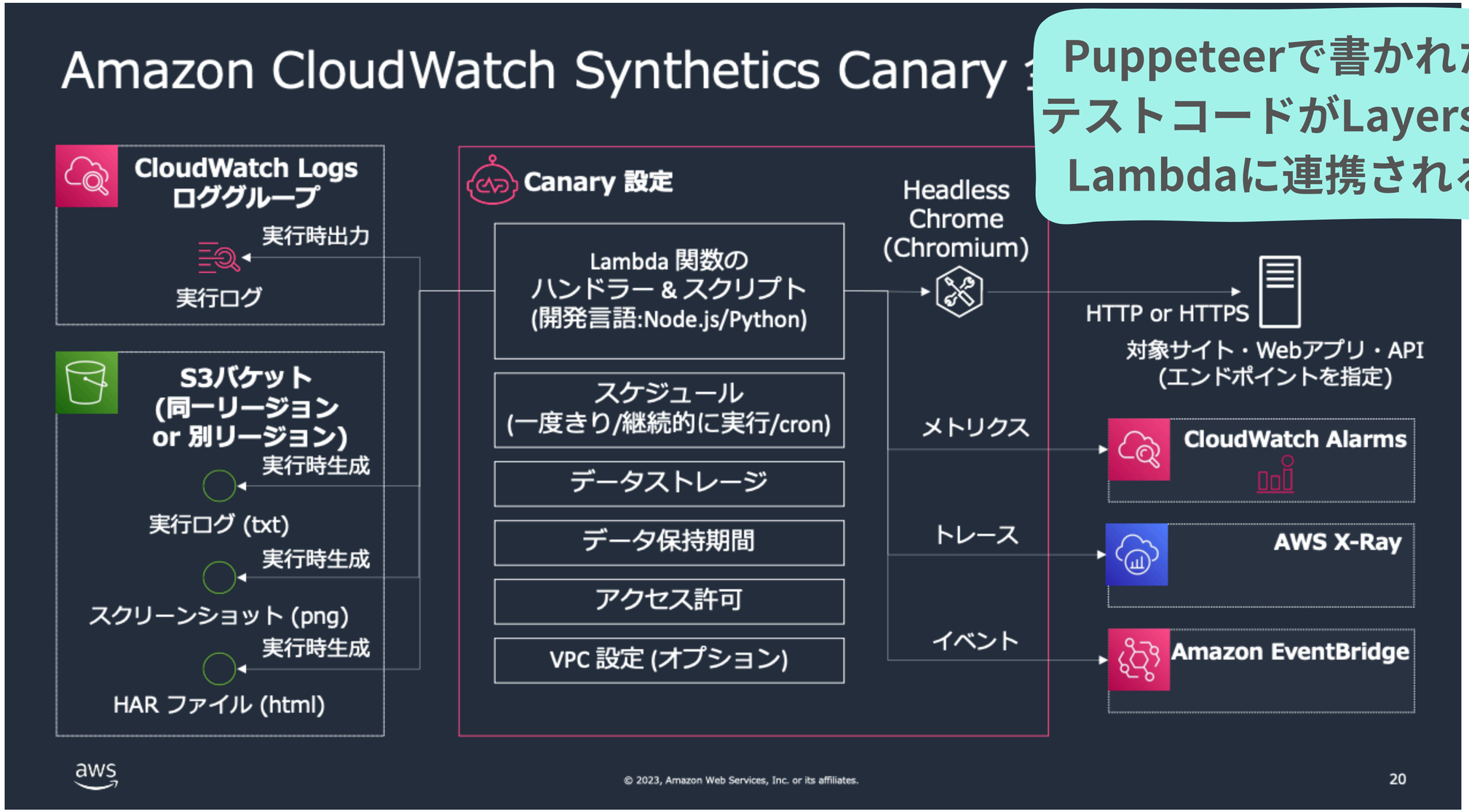
これがSynthetics Canaryだ！！！



Headless Chromeが動くLambdaが自動で作成される

（引用元：【AWS Black Belt Online Seminar】 Amazon CloudWatch Synthetics）

これがSynthetics Canaryだ！！！



Puppeteerで書かれた
テストコードがLayersで
Lambdaに連携される

（引用元：【AWS Black Belt Online Seminar】 Amazon CloudWatch Synthetics）

Amazon CloudWatch Syntheticsとは

WebアプリケーションやAPIエンドポイントに対して、
ユーザーと同じアクションを模倣して実行し、
パフォーマンスや可用性を監視するマネージドサービス

- Canaryと呼ばれるスクリプトを使用して監視を実行
- Node.jsまたはPythonランタイムのLambda関数として実行
- PuppeteerやSeleniumを使用してヘッドレスブラウザで監視
- CloudWatch RUMやApplication Signals(X-Ray)との統合も可能

Amazon CloudWatch Syntheticsとは

WebアプリケーションやAPIエンドポイントに対して、
ユーザーと同じアクションを模倣して実行し、
パフォーマンスや可用性を監視するマネージドサービス

- Canaryと呼ばれるスクリプトを使用して監視
- Node.jsまたはPythonランタイムのLambda関数として実
- PuppeteerやSeleniumを使用してヘッドレスブラウザで監視
- CloudWatch RUMやApplication Signals(X-Ray)との統合も可能

Playwright対応！

で…？

何が嬉しいの？

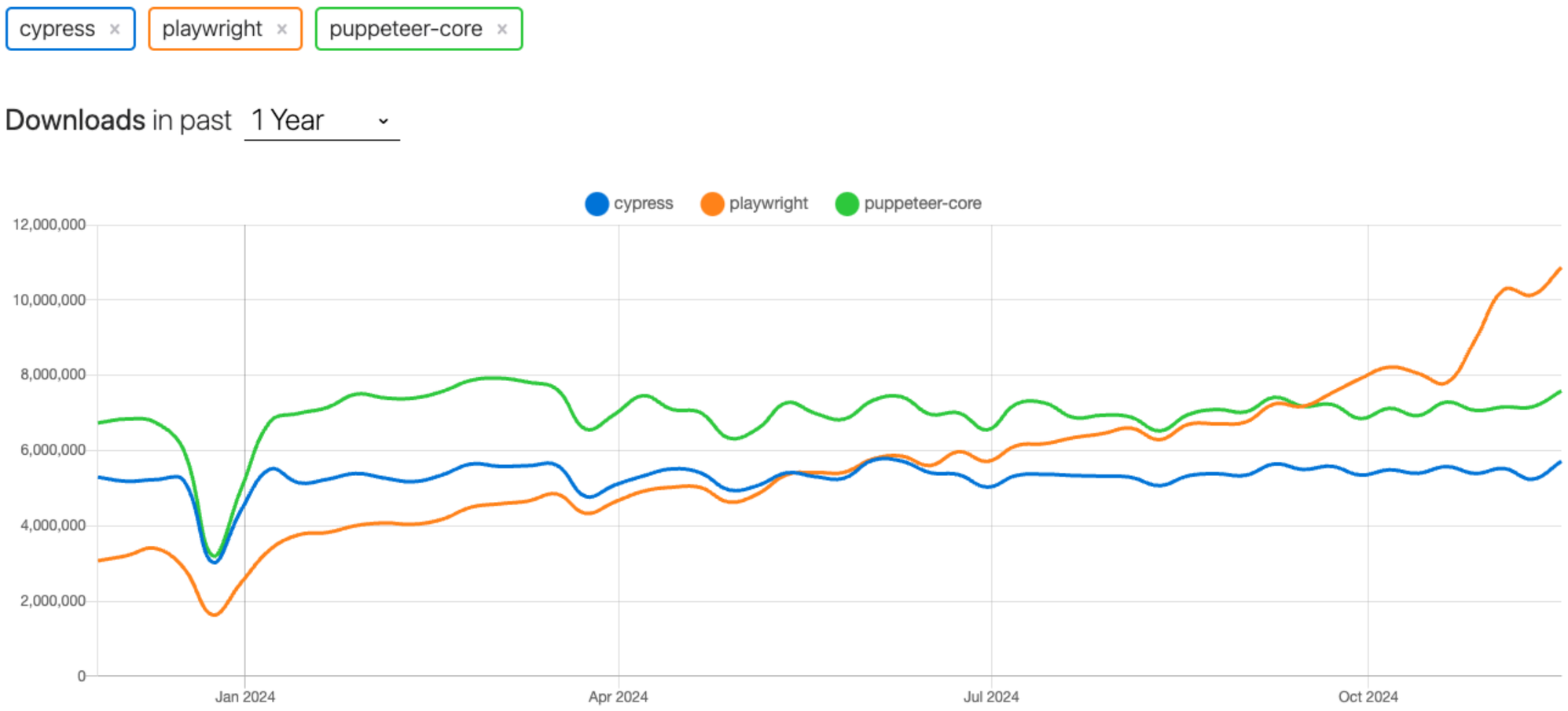
Playwrightだと何が嬉しいの？

テストの書きやすさが向上！
最近では案件の自動テストをPlaywrightで書いている人も多い？

BEFORE (Puppeteer)	AFTER (Playwright)
専用に覚えなきゃ... (とは言ってもシンプルだし、割とデファクトな書き方ではある)	案件の中の自動テストと 同じ書き味

Playwrightだと何が嬉しいの？

だいぶ前(2024/5頃)にはCypressを超えるダウンロード数
直近ではpuppeteerも超えてきた



細かいところですが、ES Moduleでの書き方にも対応

```
const synthetics = require('Synthetics');
const log = require('SyntheticsLogger');

const recordedScript = async function () {…
};
exports.handler = async () => {
    return await recordedScript();
};
```

BEFORE (Puppeteer)

CommonJSで記載されている
スクリプトをIaCなどで別途上げる
際、
node_modules内に閉じ込めないとい
けない。

```
import { synthetics } from '@amzn/synthetics-playwright';

export const handler = async (event, _context) => {…
};
```

AFTER (Playwright)

ES Moduleに対応
ディレクトリも自由度アップ

Playwrightだと何が嬉しいの？

テストの書きやすさが向上！
そもそも案件の自動テストをPlaywrightで書いている人も多い？

BEFORE (Puppeteer)	AFTER (Playwright)
要素が出てこないときの waitを明示的に記載	自動待機機能で ある程度吸収できる

Playwrightだと何が嬉しいの？

```
// ファイルアップロード
const fileInput = await page.$('input[type=file]');
await fileInput.uploadFile(downloadPath);

// ボタンが表示されるまで待機
await page.waitForSelector('[data-testid="test-button"]', {
    visible: true,  // 要素が表示されるまで待機
    timeout: 5000   // タイムアウトを5秒に設定
});

// ボタンクリック
await page.click('[data-testid="test-button"]')
```

BEFORE (Puppeteer)

アップロードが完了してから
ボタンが出現する
↓
ボタンの出現を明示的に待つ

```
    // ファイルアップロード
    const fileInput = await page.locator('input[type=file]');
    await fileInput.setInputFiles(downloadPath);

    // ボタンクリック
    await page.click('[data-testid="analyze-button"]');
```

AFTER (Playwright)

直感的にボタンのクリックを
すればいい

synthetics.jsonでスクリーンショットやログの出力を細かく制御できる

```
{
    "step": {
        "screenshotOnStepStart": false,
        "screenshotOnStepSuccess": false,
        "screenshotOnStepFailure": false,
        "stepSuccessMetric": true,
        "stepDurationMetric": true,
        "continueOnStepFailure": true,
        "stepsReport": true
    },
    "report": {
        "includeRequestHeaders": true,
        "includeResponseHeaders": true,
        "includeUrlPassword": false,
        "includeRequestBody": true,
        "includeResponseBody": true,
        "restrictedHeaders": ['x-amz-security-token', 'Authorization'
        "restrictedUrlP eters": ['Session', 'SigninToken'] // Valu
    },
    "logging": {
        "logRequest": false,
        "logResponse": false,
        "logResponseBody": false,
        "logRequestBody": false,
        "logRequestHeaders": false,
        "logResponseHeaders": false
    },
    "httpMetrics": {
        "metric_2xx": true,
        "metric_4xx": true,
        "metric_5xx": true,
        "failedRequestsMetric": true
```

スクリーンショットのタイミング
やCloudWatch Logsに出力する
ログの制御も細かくできる

synthetics.jsonでスクリーンショットやログの出力を細かく制御できる

```
{
    "step": {
        "screenshotOnStepStart": false,
        "screenshotOnStepSuccess": false,
        "screenshotOnStepFailure": false,
        "stepSuccessMetric": true,
        "stepDurationMetric": true,
        "continueOnStepFailure": true,
        "stepsReport": true
    },
    "report": {
        "includeRequestHeaders": true,
        "includeResponseHeaders": true,
        "includeUrlPassword": false,
        "includeRequestBody": true,
        "includeResponseBody": true,
        "restrictedHeaders": ['x-amz-security-token', 'Authorization'
        "restrictedUrlP  eters": ['Session', 'SigninToken'] // Valu
    },
    "logging": {
        "logRequest": false,
        "logResponse": false,
        "logResponseBody": false,
        "logRequestBody": false,
        "logRequestHeaders": false,
        "logResponseHeaders": false
    },
    "httpMetrics": {
        "metric_2xx": true,
        "metric_4xx": true,
        "metric_5xx": true,
        "failedRequestsMetric": true
```

console.logのほか、ネットワークのロギングもできるため、問題が発生した際の切り分けにも役立つ

でもお高いんでしょう…？

でもお高いんでしょう...?

Lambdaの実行料金もかかってくるのは注意

- 1時間あたり: 12回（60分÷5分）
- 1日あたり: 288回（12回×24時間）
- 1ヶ月あたり: 8,640回（288回×30日）
- 東京リージョンの料金: 0.0019 USD/実行
- 基本月額料金: 約16.23 USD（0.0019 USD × 8,540回）
- 実行時間: 5分（300秒）/回
- Lambda料金: 0.0000166667 USD/GB-秒
- メモリ使用量: 1500MB
- Lambda月額料金: 約63.28 USD

月額料金: 約79.51 USD

（基本料金 + Lambda実行時間料金）

安くはないね...

例えばこうやって使ってみよう

CI/CDに組み込んでデプロイ時の正常性確認に利用する

最初からカナリアを飛ばさずに
ここぞというときにカナリアを飛ばしてみよう

```
- name: Run Canary
  id: run-canary
  run: |
    CANARY_NAME='[redacted]'
    aws synthetics start-canary --name $CANARY_NAME

     sleep 60

    # canaryの実行が完了するまで待機
    while true; do
      STATUS=$(aws synthetics get-canary --name $CANARY_NAME --query 'Canary.Status.State' --output text)
      if [ "$STATUS" = "STOPPED" ]; then
        break
      fi
      sleep 10
    done

    LATEST_RUN_STATUS=$(aws synthetics get-canary-runs --name $CANARY_NAME --query 'CanaryRuns[0].Status.State' --output text)
    if [ "$LATEST_RUN_STATUS" = "PASSED" ]; then
      echo "Canary run passed"
    else
      echo "Canary run failed"
      exit 1
    fi
```

CI/CDに組み込んでデプロイ時の正常性確認に利用する

AWS CLIでSynthetics Canaryを起動できる

```
- name: Run Canary
  id: run-canary
  run: |
    CANARY_NAME="
    aws synth                                    NARY_NAME

                 00

    # canaryの実行が完了するまで待機
    while true; do
      STATUS=$(aws synthetics get-canary --name $CANARY_NAME --qu      'Canary.Status.State' --output text)
      if [ "$STATUS" = "STOPPED" ]; then
        break
      fi
      sleep 10
    done

    LA      RUN STATUS=$(aws synthetic      nary-runs --name $CANARY_NAME --query 'CanaryRuns[0].Status.State' --output text)
    if [ "$LA                    SSED" ]; then
      echo "Canary run passed"
    else
      echo "Canary run failed"
      exit 1
    fi
```

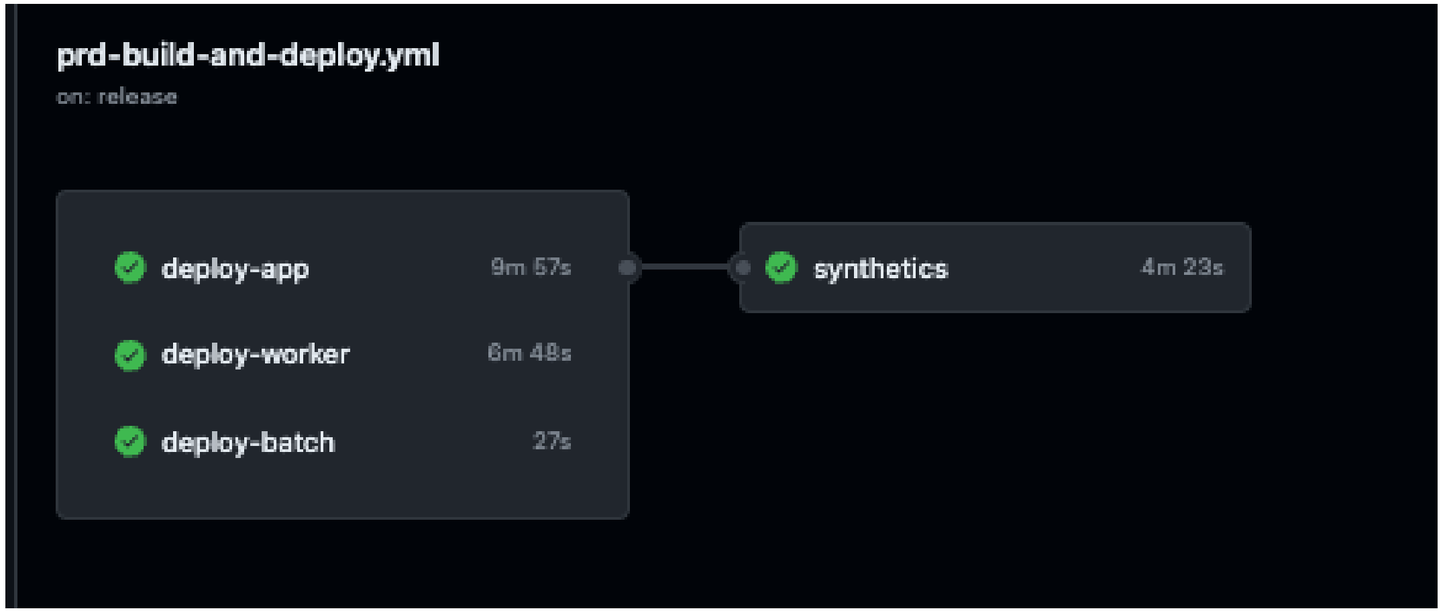
CI/CDに組み込んでデプロイ時の正常性確認に利用する

AWS CLIでステータスを確認できる

```
- name: Run Canary
  id: run-canary
  run: |
    CANARY_NAME='[redacted]'
    aws synthetics start-canary --name $CANARY_NAME

     sleep 60

    # canaryの実行が完了するまで待機
    while true; do
      STATUS=$(aws synthetics get-canary --name $CANARY_NAME --query 'Canary.Status.State' --output text)
      if [ "$STATUS" = "STOPPED" ]; then
        break
      fi
      sleep 10
    done

    LATEST_RUN_STATUS=$(aws synthetics get-canary-runs --name $CANARY_NAME --query 'CanaryRuns[0].Status.State' --output text)
    if [ "$LATEST_RUN_STATUS" = "PASSED" ]; then
      echo "Canary run passed"
    else
      echo "Canary run failed"
      exit 1
    fi
```

CI/CDに組み込んでデプロイ時の正常性確認に利用する

デプロイパイプラインの最後に
カナリアを飛ばすことで、基本動作は
自動で確認完了！！！　浮いた時間で遊ぼう。



おまけ！！

Amazon CloudWatch Synthetics now automatically deletes Lambda resources associated with canaries

Posted on: Nov 21, 2024

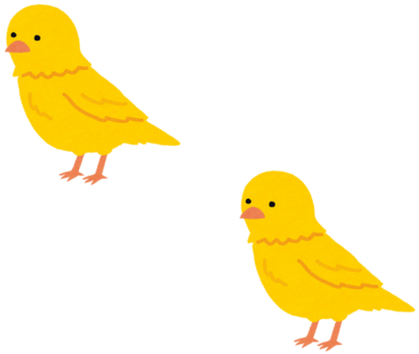
Amazon CloudWatch Synthetics, an outside-in monitoring capability which continually verifies your customers' experience by running snippets of code on AWS Lambda called canaries, will now automatically delete your associated Lambda resources when you try to delete Synthetics canaries minimizing the manual upkeep required to manage AWS resources in your account.

CloudWatch Synthetics creates Lambdas to execute canaries to monitor the health and performance of your web applications or API endpoints. When you delete a canary the Lambda function and its layers are no longer usable. With the release of this feature these Lambdas will be automatically removed when a canary is deleted, reducing the need for additional housekeeping in maintaining your Synthetics canaries. Canaries deleted via AWS console will automatically cleanup related lambda resources. Any new canaries created via CLI/SDK or CFN will automatically opt-in to this feature whereas canaries created before this launch need to be explicitly opted in.

This feature is available in all commercial regions, the AWS GovCloud (US) Regions, and China regions at no additional cost to the customers.

To learn more about the delete behavior of canaries, see the documentation, or refer to the user guide and One Observability Workshop to get started with CloudWatch Synthetics.

おまけ！！



☐	cwsyn-[redacted]-fd8c177a-f125-46e2-bf0b-2dd95fa1e2e3	-	Zip	Node.js 20.x

おまけ！！

このLambda何やねん...!!! 問題の解決に繋がりそう！

☐	cwsyn-[redacted]-fd8c177a-f125-46e2-bf0b-2dd95fa1e2e3	-	Zip	Node.js 20.x

ありがとうございました！