Developers Magazine vol.2

六木本未来ラボ所属研究員　著

1

2019-09-22 版　　六木本未来ラボ 発行

前書き

みなさま初めまして。六木本未来ラボともうします。**六本木未来ラボ**と思った方いらっしゃいませんか？ **六木本未来ラボ**という綴りなのです。左から徐々に漢字の画数が増えていっているような感じがしませんか？ どうでもいい話ですみません。さて、今回六本木付近で活動するエンジニアメンバーを集めて技術書を出版することにいたしました。初回は「Step Up AWS」をテーマに脱初心者向けの内容を寄せ集めて一つの本にしております。将来的には雑誌のように定期連載を目指していき、表紙のオムライスにちなんで「オムライス本」という愛称で親しんでもらえるよう頑張っていきたいと思います。

あ、そうそう。今回の表紙に使用しているオムライスは料理研究家の山田英季さんに作っていただきました。また表紙のデザインは山口靖雄さんに作成していただいてもらいました。この場を借りて感謝を申し上げます。お二人のご紹介は本の最後の著者紹介で掲載させていただきます。

そして改めまして記事の執筆・校正を手伝ってくれたメンバーのみなさま。この本を手に取りお買い上げいただいた読者のみなさま。ありがとうございます。ご満足いただけるかたも十分にご満足いただけないかたもいらっしゃるかもしれません。我々自身の今後の成長のためにも是非フィードバックをいただけますと嬉しい限りです。著者紹介に我々の紹介できるリンクを載せておりますのでそちらまでご連絡ください。

それでは Developers Magazine を最後までどうぞお楽しみくださいませ。

Developers Magazine 編集長　飯嶋

2

目次

目次

第 1 章

AmazonLinux でも使える！ auditd による Linux セキュリティ強化術

1.1　はじめに

Linux には標準で導入されているセキュリティ機能が数多くあり、その中の一つに auditd が有ります。audit という単語が示す通り、監査機能を提供するプログラムです。

そもそも監査って何…？ という話について詳細は割愛しますが、要はシステムがどのように使われたかを記録し、必要な時にそれを参照できるようにする仕組みです。

普段サーバーの稼働中にはプログラムの起動やユーザーのログインなど、様々なイベントが発生します。これらは普段から全てを把握しておく必要性は薄いものの、有事の際にはその内容を確認し実際に何が起きたのかを正確に把握する必要があります。

auditd による監査機能はこのような時に備えて準備しておく、いわばセキュリティの屋台骨です。事象を正確に把握できなければ、起きていることの把握や説明もできず、当然それへの対策案も考えられません。

今後はコンテナやサーバーレスといった環境が主流となっていくのかもしれませんが、それでも Linux サーバーの管理が無くなることはなく、そのセキュリティ対策も当然必要な状況が続きます。本稿では auditd を利用して Linux のセキュリティを少し向上さ

せる方法をお伝えしたいと思います。

1.2 対象の読者と環境

ターミナルによる Linux の基本的な操作は理解されている前提とさせて頂きます。また、以下の環境で動作確認をしています。

- Amazon Linux
- Amazon Linux 2
- CentOS7

1.3 auditd の概要

前述したとおり、標準で導入されている監査ツールです。しかし、デフォルトでは何も設定がされておらず、監査の役に立つ状態ではありません。せっかく準備されているものを腐らせておくのはとてももったいないので、有効活用しましょう。

auditd で設定できる監査項目は「ファイルシステムの監査」「システムコールの監査」の 2 つです。

1.3.1 ファイルシステムの監査

ファイル自体の変更、パーミッションの変更はもちろんの事、読み取りアクセスも監視・記録することが可能です。/etc/passwd など、変更があった場合にチェックしておきたいファイルについて設定を行います。

1.3.2 システムコールの監査

kernel に送られてきたシステムコールを監視・記録することが可能です。とはいえ、システムコールは常に利用されている物で種類も膨大な為、すべてを記録する事はナンセンスです。普段呼び出されない、または人間の操作によって呼び出されるであろうシステムコールを監査する辺りがちょうどよいレベル感ではないかと個人的には考えています。

1.4 auditd を構成するプログラム・ファイル

auditd による監査機能は複数のプログラム・ツールから成り立っています。まずはどんなものがあるのかザッと見てみましょう。

1.4.1 auditd

システム起動時に起動・常駐し、システムの挙動を監視する auditd 本体とも言えるデーモンです。再起動等は systemd、設定等は後述の auditctl で操作するので、直接触ることはほぼありません。

1.4.2 auditctl

auditd の管理を行うコマンドです。systemd と systemctl のような関係ですね。auditd の操作で一番多く触ることになるプログラムです。

1.4.3 ausearch

auditd で記録された監査ログはテキスト形式ではあるものの、そのまま人間が読むのは少々（かなり？）辛いものが有ります。ausearch は監査ログから指定した条件を満たすログを抽出し、人間が読みやすい形で表示する事ができます。

1.4.4 /etc/audit/audit.conf

auditd のメイン設定ファイルです。auditd 自身の設定や、どんなシステムの挙動を記録するのかという監査ルールをこのファイルで設定します。ただし、直接編集することは有りません。auditctl で設定した内容が書き込まれます。

1.4.5 /var/log/audit

auditd によって記録されたログが格納されます。ただし ausearch の項でも触れましたが、記録されるログはテキストではあるものの人間には非常に読みづらいため、ausearch を使ってログを閲覧します。

1.5 auditd の基本操作

1.5.1 利用準備

インストール

基本的に不要です。AWS から提供されている Amazon Linux の AMI で起動したインスタンスには導入済み。CentOS7 公式 AMI も同様です。仮に CentOS7 を VM などに自分で最小構成インストールしたとしてもインストールされます。つまり、あえて意図的に除外しない限り最初から使える環境にあるということです。

サーバー起動時の自動起動

これも不要です。最初からシステム起動時に auditd も起動するようになっています。Amazon Linux 1 では chkconfig、Amazon Linux 2 や CeontOS7 では systemd で制御されています。

必要な権限について

auditd の操作はほとんどが root 権限が必要になります。sudo 等で root 権を利用できるようにしておいてください。

1.5.2 現在の監査ルールの確認

auditctl -l で現在の設定を表示することができます。

▼リスト 1.1　auditctl -l によるルールの表示

```
# auditctl -l

No Rule.
```

デフォルトではこのように監査ルールが何も設定されていない状態で稼働しているため、ルールがない旨の表示がでます。

1.5.3 auditd 自体の設定表示

auditctl -s で auditd 自体の設定を表示できます。

▼リスト 1.2　auditctl -s による設定の表示

```
# auditctl -l

TODO: No Rule.
```

1.6 ルールの追加

ルールの追加は auditctl コマンドを利用して行います。例として /tmp/test.txt というファイルが作成されたり、内容に変更が有ったら記録するファイルシステム監査のルールを設定してみます。

▼リスト 1.3　ルールの追加

```
auditctl -f
```

エラーが出なければルールの追加ができています。確認してみます。

▼リスト 1.4　追加したルールの確認

```
auditctl -l

TODO: aaaaaaaaaaaaaaaaaaaaa
```

追加時のオプションはそれぞれ以下のような意味を持ちます。詳細は man auditctl や RedHat が提供しているマニュアルページを参照してください。

▼表 1.1　ルール追加時の auditctl オプション

	効果
-l	現在のルールを表示します

1.7 ログの確認

これで /tmp/test.txt が作成されたり変更があった際に記録されるようになりました。実際にファイルを作成してみて、それが記録されていることを確認してみましょう。

▼リスト 1.5　テストファイルの作成と ausearch の実行

```
echo "TEST" >> /tmp/text.txt
ausearch
```

ausearch を実行すると以下のような内容が表示されます。

▼リスト 1.6　ausearch の実行結果

```
echo "TEST" >> /tmp/text.txt
ausearch
```

1.8　ルールの削除

ルールの削除も auditctl で行います。

▼リスト 1.7　ルールの削除

```
auditctl -f
```

1.9　ssh ログイン監査の設定事例

auditd を利用して、システムに ssh でログインされた記録を取り、ログインが検知されたらメールで通知するようにしてみましょう。

1.9.1　ssh のログイン検知の仕組み

ssh によるログインが発生した際には必ず xxxx システムコールが発行されて tty が割り当てられます。このシステムコールの実行を auditd で追跡・記録する事で ssh によるログインを検知する事ができます。

1.9.2　監査ルールの追加

auditctl を使って、システムコールの監査ルールを追加します。

▼リスト 1.8　システムコールの監査ルールを追加

```
auditctl -f
```

エラーが出なければルールの追加に成功していますが、念の為確認します。

▼リスト 1.9　追加したルールの確認

```
auditctl -l

TODO: aaaaaaaaaaaaaaaaaaaaa
```

1.9.3 メール送信の設定

ssh によるログインが行われ、auditd が検知したらメールを送信するように設定します。なお、事前にこのシステムからメールが送信できるよう postfix 等の設定をしておいてください。

1.10 まとめ

第 2 章

AWS Challenge

2.1 AWS Challenge とは

「AWS Challenge」とは、 AWS CloudFormation や Serverless Framework を用いて構築された AWS 環境の不具合を見つけ解決しながら AWS に慣れ親しんでもらうことを目的とした学習課題です。

皆様の挑戦をお待ちしております。

第 3 章

AWS CI/CD 入門

皆さん、CI/CD してますか？

3.1 CI/CD とは

CI/CD という言葉は今や IT 業界では一般的になっており、CI/CD が何を示しているのかを説明する機会が減ってきています。またプロジェクト毎や会社毎または、個人の認識の違いで CI/CD が取りうる範囲が少し違うこともしばしばあります。ここではなるべくそういった言葉の齟齬をなるべくなくすために CI/CD が取り扱う範囲を定義します。

CI/CD は英語では「Continuous Integration/Continuous Delevary」日本語に訳すと「継続的インテグレーション/継続的デリバリー」といいます。　ここで「CD にデプロイは入らないの？」と疑問に思った方のために RedHat #@# 脚注参照が定義する CI/CD を例にここでしっかりと言葉の定義をしていきます。

https://www.redhat.com/ja/topics/devops/what-is-ci-cd#

3.1.1 継続的インテグレーション（Continuous Integration）

継続的インテグレーション（以下 CI と呼ぶ）は主に開発メンバーのための開発プロセスの自動化を意味します。

Unit テストなどを開発プロセスに自動で行うように組み込むことで品質を上げつつ開発効率を上げていくことを目的とします。

CI は最終的にプロジェクトで利用している共通のレポジトリに自動で統合されるところまでを行います。こうすることで新しいコードの変更が自動で、コミット、ビルド、テストされ、バグの含みにくい開発プロセスが実施できます。

3.1.2 継続的デリバリー（Continuous Delibary）

継続的デリバリー（以下 CD と呼ぶ）は主に開発コードにバグがないか自動的にテストを行い、レポジトリにアップロードするまでの一連のプロセスの自動化を意味します。

CI の時と違う点は、CD がコードを最終的にアップロードする先が本番環境へデプロイされるために用意されたレポジトリである点です。このレポジトリは常に本番環境がリリースできる状態のコードを保管しておくことが重要です。また、継続的デリバリーのデリバリーが意味するところは、「本番相当の環境へデリバリーする」つまりシステムテストができる環境（ステージングと呼ばれたりする）にコードをデリバリーするところまでを指します。したがって、その先のシステムテストに関しては CD のとりうる範囲ではありません。システムテストを手動でテストするか自動でテストするかはプロジェクト要件次第になります。

CD は開発と運用のコミュニケーションロスを解決し、変更コードの導入作業の負担を下げることを目的としています。本番環境へデプロイする運用者は CD によってアップロードされた最終的なソースコードが保管されているレポジトリさえ確認すれば開発者の変更コードを本番環境へデプロイすることができます。

3.1.3 継続的デプロイメント

継続的デプロイメント（以下 CD（deployment）と呼ぶ）はズバリ、本番環境にデプロイするまでのプロセスの自動化を意味します。つまり、開発者の変更コードが全て自動で本番環境へデプロイされることを指します。したがって、CD（deployment）の重要な本質は本番環境へ自動デプロイするためのシステムテストや UI テストなどの一連のテスト作業の自動化プロセスにあります。筆者もデプロイまでを完全自動化で運用しているサービスにはまだ出会ったことがありません。CD（deployment）を実現させるには事前のシステムテストの洗い出しや、システムテストの自動化、UI テストの自動化などを入念に設計しておかなければなりません。しかし、これらが実現できるプロダクトではサービスのフィードバックを即座にコードに反映させビルド、テスト、デプロイをスピーディーかつ安全に本番環境へデプロイすることができるようになります。時には、変更コードを少しずつアップデートし、デプロイメントによる障害のリスクを低減させることも可能になります。

本章ではこの CD（deployment）のためのテストの自動化に関しては取り扱いません。システムテストや、UI テストはステージ環境で手動で実施することを前提として CI/CD を解説していきます。また、次の項からお話する「本番環境のデプロイ」に関しては CD

（deployment）の本質を意味する「システムテストの自動化」の意味合いを含めず単に本番環境へデプロイするための自動化プロセスのことを指し示すこととします。

最後の CD（deployment）に関してはプロジェクトや会社ごとで認識はバラバラだと思うのでここでは混乱を防ぐために上記のような定義でお話していきます。

CI/CD を設計する上で大切なのは CI/CD はプロジェクトによってその自動化プロセスを自由に定義し、どこまで自動化するかを決めることができることです。開発効率を高めるためにプロジェクトの規模と期間にあった自動化プロセスの定義をプロジェクト毎に都度設計していく必要があります。本章ではそのような環境に置かれている開発者のお手伝いとなれるように私が過去に関わったことのある CI/CD のプロセスの事例について紹介していきます。

3.2 The Twelve-Factor App に沿った CI/CD

ソフトウェア開発における方法論について定義された The Twelve-Factor App にも CI/CD についての記載がされています。本章ではただ CI/CD を構築するのではなくこの The Twelve-Factor App にも沿った CI/CD の設計を意識して解説しています。

3.3 一般的な CI/CD フロー

CI/CD はプロジェクト要件によって自動化プロセスを自由に定義できることところが魅力です。しかし、初めて CI/CD を導入しようとしている方にとっては何をすればいいかわからないかもしれません。ここでは CI/CD 入門ということでどのようなプロジェクトでも利用できる一般的な CI/CD フローを解説します。CI/CD では各フローで実行される処理を「ジョブ」と呼びます。

3.3.1 プルリク/マージリクエスト

CI/CD の旅はここから始まります。多くのプロジェクトは GitFow や GitHub Flow に従った Git のブランチ戦略をとってるかと思います。その場合新規機能を Feature ブランチで開発した後、共通レポジトリにプルリク/マージリクエストを送るはずです。このプルリク/マージリクエストイベントをトリガーにジョブを順番に実行していきます。

3.3.2 コードの静的解析

先程のプルリク/マージリクエストイベントを検知するとコードの静的解析を行います。例えば、javascript を利用している場合は適切な EcmaScript のバージョンで記述さ

れているかを確認するための ESLint を実行します。他の言語でも同様です。いわゆる Lint を実行し、コード規約に沿っているか、推奨されるプログラムの書き方を行っているか、利用していない変数はないか、初期化されていない変数はないか、コードエラーが出ていないかなど様々なコードやプログラム言語に関するチェックを行います。

チェックを行った後は、その結果によって後の CI を続けるか続けないかを決めます。成功した場合は次のプロセスを実行します。失敗した場合は失敗した旨をチャットツールなどコミュニケーションツールに知らせることもできます。

3.3.3 Unit テスト

コードの静的解析が成功すると次に Unit テストを行います。テストコードを各文化がある会社やプロジェクトではそのテスト実行を CI 上で自動化すると良いです。成功した場合、 Unit テストツールによってはカバレッジをログ形式で CI 上に単に表示させることもできますし、分析しやすいように別のグラフ系の Web サービスを使って表示させても良いでしょう。

3.3.4 コードのマージ

Unit テストが成功すると開発用の共通ブランチにマージすることができます。一般的にはこのマージ前に開発リーダや、上級エンジニアの方のコードレビューを行い、承認が通ったコードだけマージさせることが多いでしょう。CI もそのようなフローに合わせて自動でマージさせずコードレビューのための承認フローを入れます。

また、マージされるソースコードは、CI によってコードの規約チェックや Unit テストを行うジョブをくぐり抜けてきています。したがって、開発リーダのコードレビューは「ソースコード上のビジネスロジックの確認に専念」できます。「インデントが揃ってない」や「使ってない変数がある」など、コードレビュー時の本質外のマージリクエストの突き返しがなくなります。開発リーダがコードレビューを行った後は「マージ」ボタンを押して開発用の共通ブランチにマージして終了です。

ここまでが CI が取り扱うジョブの流れになります。

3.8 本番環境に EC2 インスタンスを利用するデプロイパターン

3.8.1 レポジトリに GitHub を利用する場合

AutoScaling 環境下でのローリングアップデート例
AutoScaling 環境下でのブルー・グリーンデプロイ例

3.9 本番環境にサーバレスを利用するデプロイパターン

3.9.1 S3 バケットにデプロイ後 CloudFront のキャッシュを自動削除する例

3.10 目指せ最速デプロイ ElasticBeansTalk を使ったデプロイパターン

3.11 付録

第 4 章

AWS X-RAY で始めるパフォーマンス監視入門

4.1　ほげほげ

4.2　もげもげ

第 5 章

AWS を使用して静的 WordPress をデプロイする

5.1　はじめに

私自身、WordPress での開発・運用を 10 年近くに渡って携わって来ましたけど、ひたすら某 G 社のレンタルサーバーを愛用して来ました。

その理由はただ一つ！！

「安いから！！」「楽だから！！」

そもそも WordPress で運用しているサイトは、企業のコーポレートサイトなどが多いため耐久性や可用性に関する考慮は正直あまり必要ないという点はあります。正直私自身も AWS で WordPress を運用するのは、余程の PV を集めるサイトではないとその必要がないと思ってきました。

ところがここ数年 AWS を勉強していくにしたがって、AWS でも安く WordPress の運用ができるのではないかという可能性を感じるようになりました。この章では、AWS を使用した WordPress の静的化と、そのデメリットとデメリットを実際に検証した結果を元に書いていきます。

5.2　静的化のメリット

5.2.1　なぜ WordPress を静的にするの？

1. セキュリティ静的サイトは安全です。Worpress で使われるプラグインも全て静的なファイルに書き換えられるので、安全性が高くなります。また、アップデート・ハッカー・メンテナンス・スクリプトの破損・__悪いプラグインなどについて心

配する必要は全くないので静的化にした方が安心にサイトの運用をする事ができます。

1. スピード。

当たり前の話になりますが、静的ファイルは動的ファイルよりも速くロードされます。PHP の実行とデータベースへのアクセスも無いので、スピードが俄然速くなります。

1. コスト

要求を処理するために必要なサーバーの通信時間は少なくなるので、静的にした方が安くなります。特にトラフィックが多い場合は、コスト削減の恩恵を沢山受けられるに間違いありません！！ この章の後半で実際に静的にした場合に、どれぐらいのコスト削減ができるかの検証結果を掲載します！！

ポートフォリオや個人的な Web サイトを運営している場合は、HTML と CSS（そして多少の JavaScript）のみで構成されているとすると、その場合も静的化をする事によって多くのメリットを得る事ができます。

5.3 Docker で WordPress を構築する

2
3

以下の手順で Docker 環境を作っていきます。

1. 空のプロジェクトディレクトリを作成します。
2. そのディレクトリに移動します。
3. 以下の内容で docker-compose.yml というファイルを作成します。
4. プロジェクトディレクトリで、次のコマンドを実行します。docker-compose up -d * この時点で WordPress は localhost でアクセス可能になるでしょう。* localhost/wp-admin に移動して WordPress の設定を開始します。* 注：docker コンテナーをシャットダウンするには docker-compose down --volumes を実行してください。
5. これで、Web サイトを変更する準備が整いました。

▼リスト 5.1　docker-compose.yml

```
{
version: '3.3'

services:
   db:
```

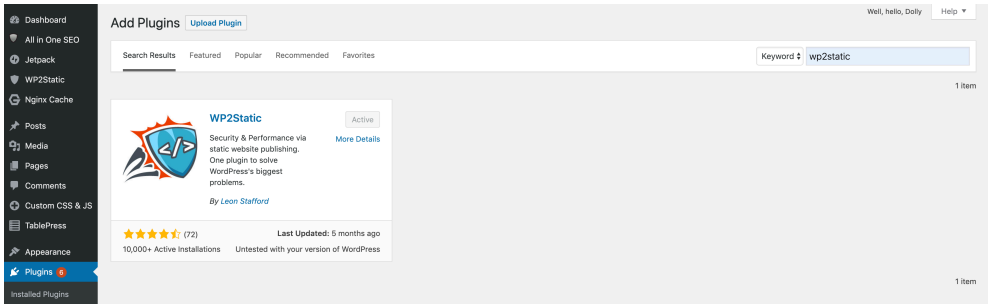
```
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  cli:
    image: wordpress:cli
    volumes:
      - ./site:/var/www/html/

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "80:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress

volumes:
  db_data:
}
```


5.3.1 静的 WordPress をダウンロードして AWS S3 にデプロイする

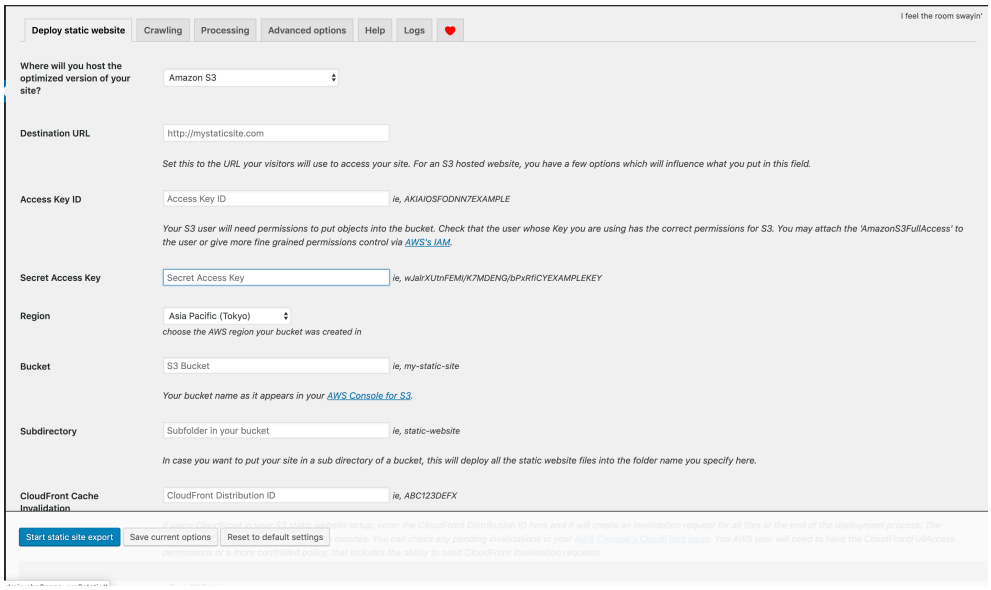
WordPress サイトをホスティングすることは安全ではなく、費用もかかります。静的にすることはこれらの欠点を最適化するための素晴らしい方法です、そしてそれを S3 でホストすることはさらにコストを削減し、よりきめ細かいセキュリティ設定を可能にします。

以下の手順では、WordPress サイトから静的ファイルを生成し、それらをカスタムドメイン用に新しく構成された AWS S3 バケットでホストする方法について説明します。

1. 「プラグイン」 > 「新規追加」 > wp2static を検索 >「インストール」 >「有効化」の順にクリックします。プラグインが有効になったら、サイドパネルの wp2static に移動して Deploy static website タブを選択します。
2. タブの下に Where will you host the optimized version of your site?のところに ZIP アーカイブを選択します。次に保存をクリックします。
3. Destination URL のところに S3 に紐ついてる Domain 名を入れます。
4. Start static site export を押すと zip で静的ファイルのダウンロードできます。



▲図 5.1　wp2static プラグイン



▲図 5.2　static export

5.3.2　S3 バケットの設定

1. AWS にサインインします。S3 に移動して新しいバケットを作成します。ドメインの名前を付けます。例えば showcase-tv.com です。
2. 次に、バケットポリシーを作成する必要があります。AWS にはこの便利なポリシージェネレータがあります。ジェネレータを使用したくない場合は、AWS のド

キュメントにもかなり良い例があります。

3. パケットポリシーを以下のように設定してください
4. 安全を期すために、すべての GET リクエストがドメインにアクセスすることを許可する 1 つのルールを持つ CORS 構成を追加することをお勧めします。この場合は https://showcase-tv.com です。
5. S3 のバケットのページからプロパティのページへ移動してホスティング機能を有効にします。
6. wordpress からダウンロードした Web サイトファイルを s3 bucket にアップロードします。

▼リスト 5.2　AWS Policy Generator

```
Type of policy:：S3 Bucket Policy
Effect: Allow
Principal: *

AWS サービス：Amazon S3（これはポリシーの種類に応じて自動的に設定されます）
Actions：GetObject のみ

AmazonResourceName（ARN）：これはドメイン名に依存しますが、同じフォーマットになります。例えば：
arn：aws：s3 :::showcase-tv.com/*
```

▼リスト 5.3　バケットポリシー

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AllowPublicReadAccess",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::showcase-tv.com/*"
        }
    ]
}
```

▼リスト 5.4　XML の例

```
{
  <?xml version="1.0" encoding="UTF-8"?>
  <CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <CORSRule>
    <AllowedOrigin>https://showcase-tv.com</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <AllowedHeader>*</AllowedHeader>
  </CORSRule>
  </CORSConfiguration>

}
```

5.4 まとめ

第 6 章

AWS SageMaker 入門

6.1 Amazon SageMaker とは

AWS で提供されている機械学習のマネージドサービスです。機械学習やディープラーニングを開発する上で面倒な問題の 1 つにインフラ環境の用意があります。それを開発者がやらなくても良いサービスが近年増えてまいりました。有名なのが Colaboratory という Google のサービスです。Amazon SageMaker もインフラ環境の構築の必要はありません。

6.2 特徴

6.2.1 Amazon SageMaker のメリット

開発環境の構築がすぐにできる
必要な時だけ GPU 環境が作成される

ディープラーニングでは学習時に GPU 環境を用いると高速に学習を行えますが、それ以外で GPU 環境をスタンバイしておく必要はありません。Amazon SageMaker では学習が始まるタイミングに GPU 環境が用意され終了すると自動的にインスタンスが削除されるので、学習している時間だけ費用が発生します。Amazon EC2 だと学習時だけ GPU 環境が立ち上がり、終わるとシャットダウンするという構成にするのが難しいのでそういった点も Amazon SageMaker のメリットです。

AWS の各種サービスと連携しやすい

データセットの保管や呼び出しに Amazon S3 を容易に利用することができます。また学習済みモデルも Amazon S3 に自動的に保管されます。AWS で Web サービスを構築

すると、アクセスログやユーザーの投稿データなどを Amazon S3 に保管しておくことが多いと思いますのでそのデータを簡単につなぎ合わせることが可能です。

推論用 API の自動構築

学習させたモデルはたった 1 行でデプロイすることが可能です。Amazon SageMaker が API エンドポイントを提供してくれるので、Amazon EC2 などに GPU 環境など推論用の環境構築をする必要はありません。またマネージドサービスなので API サーバーのメンテナンスも必要ありません。

学習モデルの再現性

全ての学習ジョブについては、Amazon SageMaker のトレーニングジョブに履歴が残っているため、学習したモデルをいつでも再現することができます。

* 開発・学習・推論好きなところだけを導入できる* S3 などのサービスと連携できる（データセット/モデルの保管など）

* トレーサビリティ* 再現性

6.3 Amazon SageMaker を始める

29

つらつらと 5 ページくらい予定

6.3.1 開発

6.3.2 学習

6.3.3 推論

----------ここまでが第 1 弾---------- == Raspberry Pi に組み込むには Amazon SageMaker Neo を利用して IoT デバイスに最適化された推論モデルにする。

* AWS IoT Greengrass を使用する* 学習モデルを手動で組み込む

6.4 Amazon SageMaker Neo とは

6.4.1 AWS IoT Greengrass を使ってデプロイする

----------ここまでが第 2 弾----------

6.5 Amazon SageMaker の活用事例

そんなない。

6.6 Amazon SageMaker のその他サービス

Amazon SageMaker Ground Truth とか紹介
----------⬆ 書かなくていいけど必要があれば書く----------

6.7 まとめ

6.8 サンプルコードのダウンロード先について

六木本未来ラボ所属研究員紹介

著者

■飯嶋渉 第 8 章担当

2013 年ごろより株式会社 Showcase-TV にエンジニアとして入社。2018 年 11 月に AWS Dev Day Challenge でプレゼンを行い優勝。DeveloperMagazine 編集長を名乗らせてもらっている。

`https://twitter.com/iijima_wataru`

`https://qiita.com/w-iijima`

■佐藤孝昭 第 3 章/第 5 章担当

2017 年 2 月株式会社 Showcase-TV へ転職のため地方から上京。2018 年はアウトプットの一環としてイベントにも参加しなきゃと出場した AWS DevDay Challenge で優勝！AWS 認定ソリューションアーキテクトプロフェッショナル。

`https://qiita.com/sato-takaaki`

`https://github.com/takaaki-s`

■しょいみん 第 6 章/第 7 章担当

2017 年に SIer を脱出 Web 業界でひたすら AWS と格闘している 25 歳。開発もインフラも DevOps もセキュリティも全部やりたい修行中。

`https://twitter.com/AZyuji`

■峯岸友紀 第 2 章/第 4 章担当

株式会社ホワイトパズルで、結婚式向けサービス「ダブルピース」を運用中！

`https://v-peace-v.com`

執筆は、自分との戦いだ！

■吉川功一郎 第 1 章担当

2000 年よりエンジニアとして働き始め、2011 年に独立。IIJ、富士ソフト、LIFULL といった多種多様な現場で経験してきた内容を活かしてクラウド導入及び運用のコンサルタントとして活動中のフリーランスエンジニア。

`https://twitter.com/yskw516`

表紙デザイン

■山口靖雄

2004 年より、株式会社 ほぼ日（旧：株式会社 東京糸井重里事務所）に「ほぼ日刊イトイ新聞」デザイナーとして入社。2013 年に退社し独立現在フリーランスのデザイナー・ディレクター・カメラマン。

オムライス作成

■山田英季

2015 年 11 月に、（株）and recipe を立ち上げ、「ごはんと旅は人をつなぐ。」をテーマにしたサイトの運営をおこなう。料理家としては、韓国、台湾、ロンドン、パリなど世界中に活躍の場を広げている。

校正

■堀徳敏

珍しい名前ですが、正真正銘の日本人。2018 年ぐらいから株式会社 Showcase-TV に在籍して、AWS の知見を深めるために必死に喰らいついている。

素顔は単なる野球バカ・スポーツバカ。家族（妻・娘 5 歳）とディズニーランドに行くのが大好き。

3
3

3
4

Developers Magazine vol.2

2019 年 9 月 22 日　v1.0.0

著　者　六木本未来ラボ所属研究員
編　集　飯嶋渉、佐藤孝昭、しょいみん、峯岸友紀、吉川功一郎、堀徳敏、ギリラメス
発行所　六木本未来ラボ