【AWS Black Belt Online Seminar】 AWS Greengrass で実現する エッジコンピューティング

アマゾン ウェブ サービス ジャパン株式会社
ソリューションアーキテクト　園田修平

2018.05.08

自己紹介

名前：

園田 修平（そのだ しゅうへい）

所属：

アマゾンウェブサービスジャパン株式会社

技術統括本部 ソリューションアーキテクト

担当：

エッジコンピューティングを含む IoT 全般

内容についての注意点

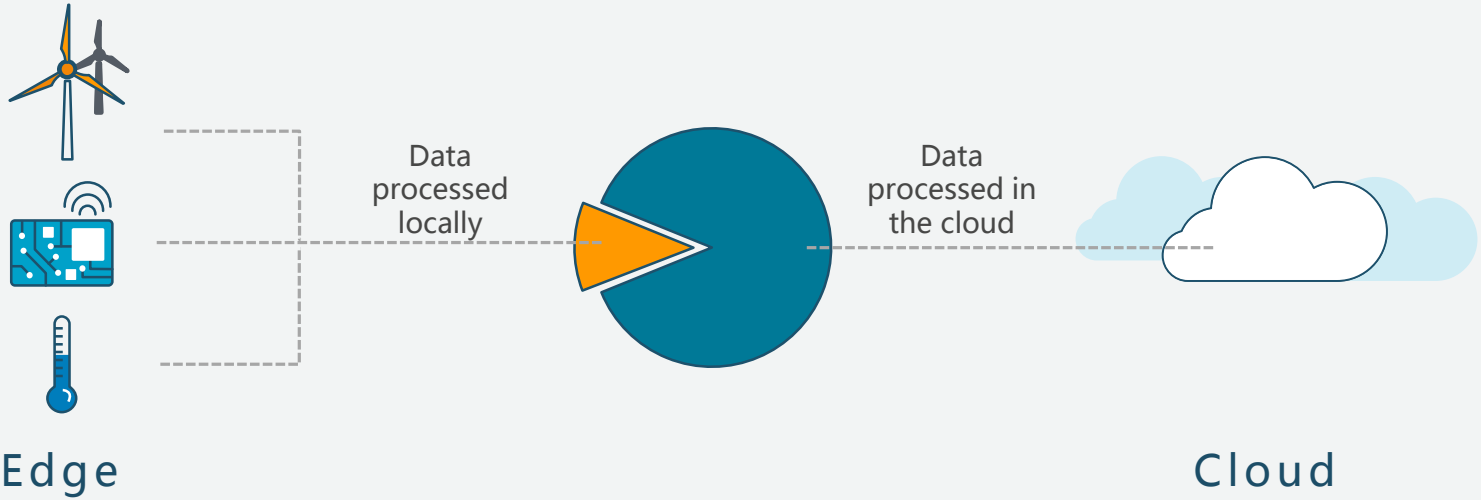
- 本資料では2018年5月8日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。
- 資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。
- 価格は税抜表記となっています。日本居住者のお客様が東京リージョンを使用する場合、別途消費税をご請求させていただきます。
- AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

Agenda

- エッジコンピューティングとは
- AWS Greengrass の詳細
- デザインパターン
- エッジコンピューティングを採用する場合の注意点
- まとめ

エッジコンピューティング

クラウドと異なる場所にコンピューティングリソースと意思決定機能をもたせること



エッジコンピューティングが解決する課題

レイテンシ

帯域幅

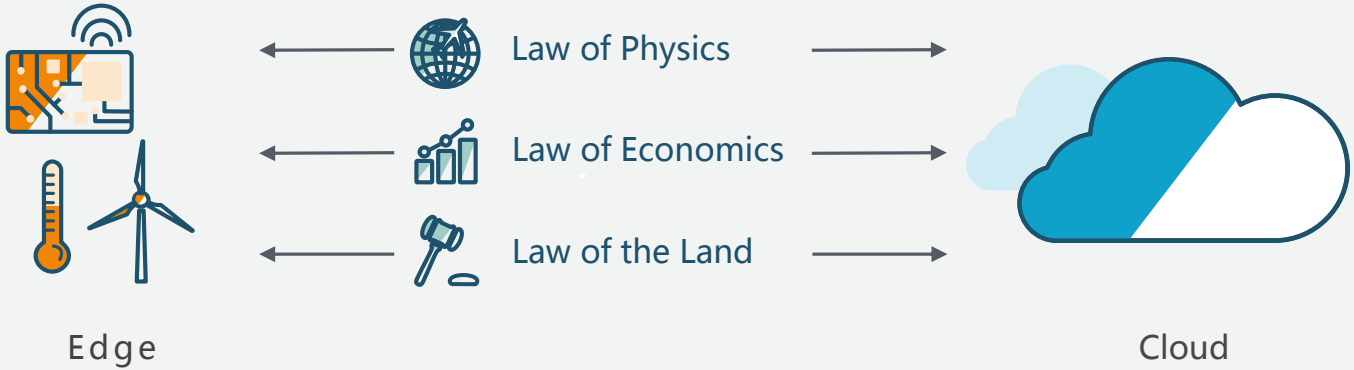
オフライン

セキュリティ

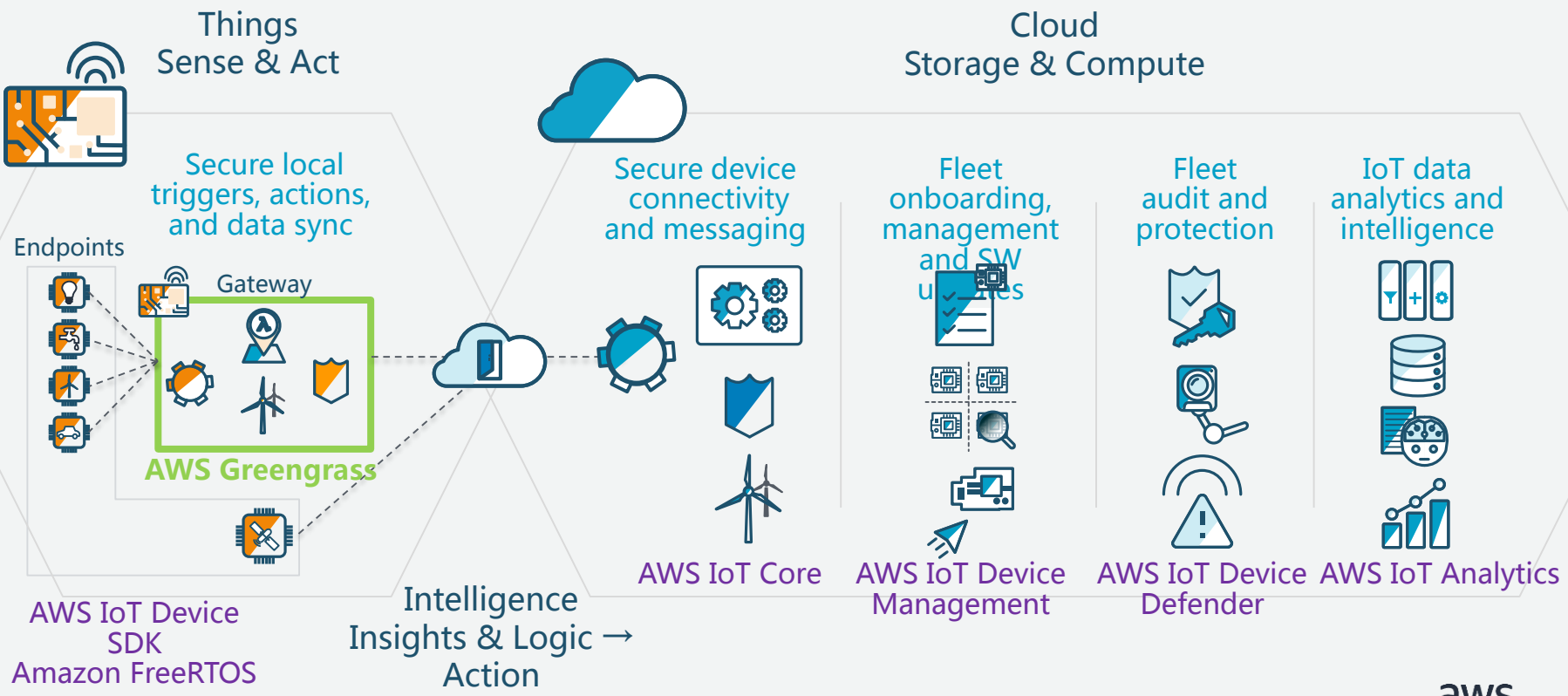
AWS Greengrass

Extend AWS IoT to the Edge

AWS IoT の各種機能をエッジデバイスに拡張
クラウドのメリットを受けつつ
ローカルでのデータ処理を可能にする



AWS の IoT サービスにおける AWS Greengrass の位置づけ



AWS Greengrass

Extend AWS IoT to the edge

クラウドから配布したLambdaをローカルで実行

ローカルでのメッセージングとLambdaのトリガ

データと状態の同期(ローカルシャドウ)

クラウド同様のセキュリティポリシー

Greengrass Core自体のアップデート(OTA)

OPC-UAのプロトコルアダプタ

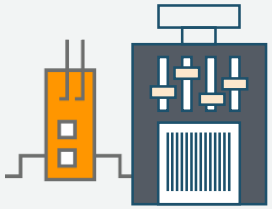
ローカルリソースアクセス

ローカルでML/DLの推論を実行

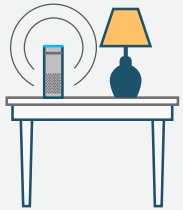
2018.4.4
GA Announce!!

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

ユースケース



Industrial Gateways



Consumer Electronics

Energy

Retail

Medical

Construction

Automotive

Infrastructure

Mining

Agriculture

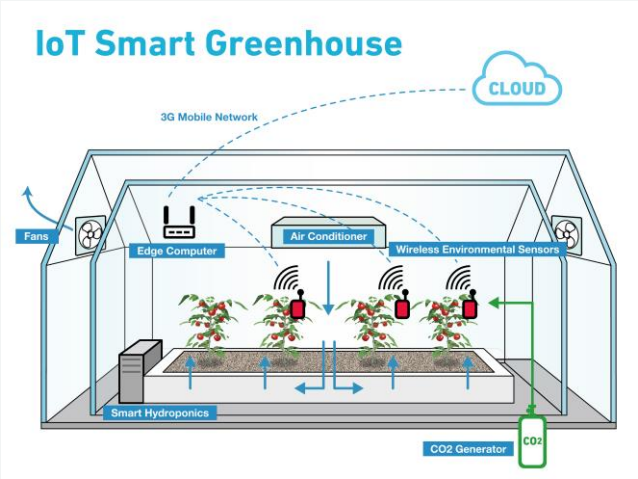
Insurance

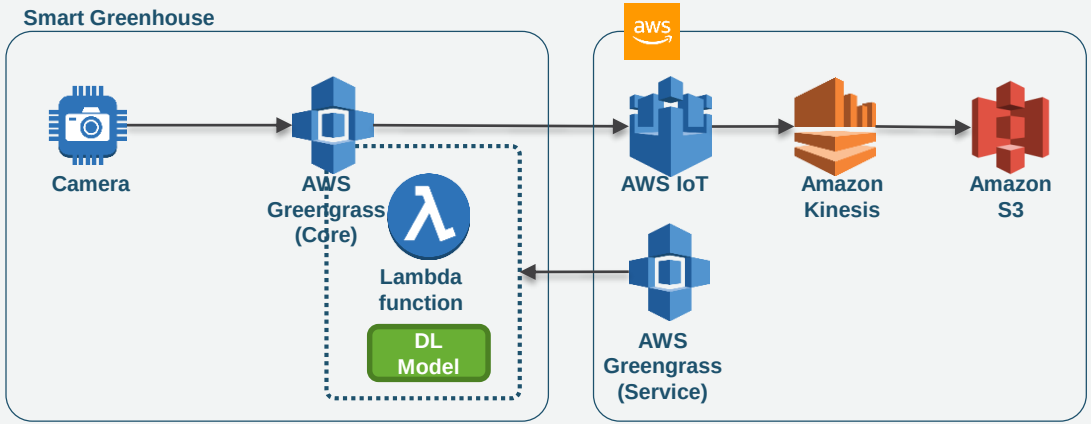
More…

aws

事例: Yanmar IoT Smart Greenhouse

- センシングデータから作物への水やりをエッジで判断することにより自動化、一時的なオフラインにも対応
- 作物の検出や成熟度の判別をエッジで行うことによりネットワーク帯域の問題を解決





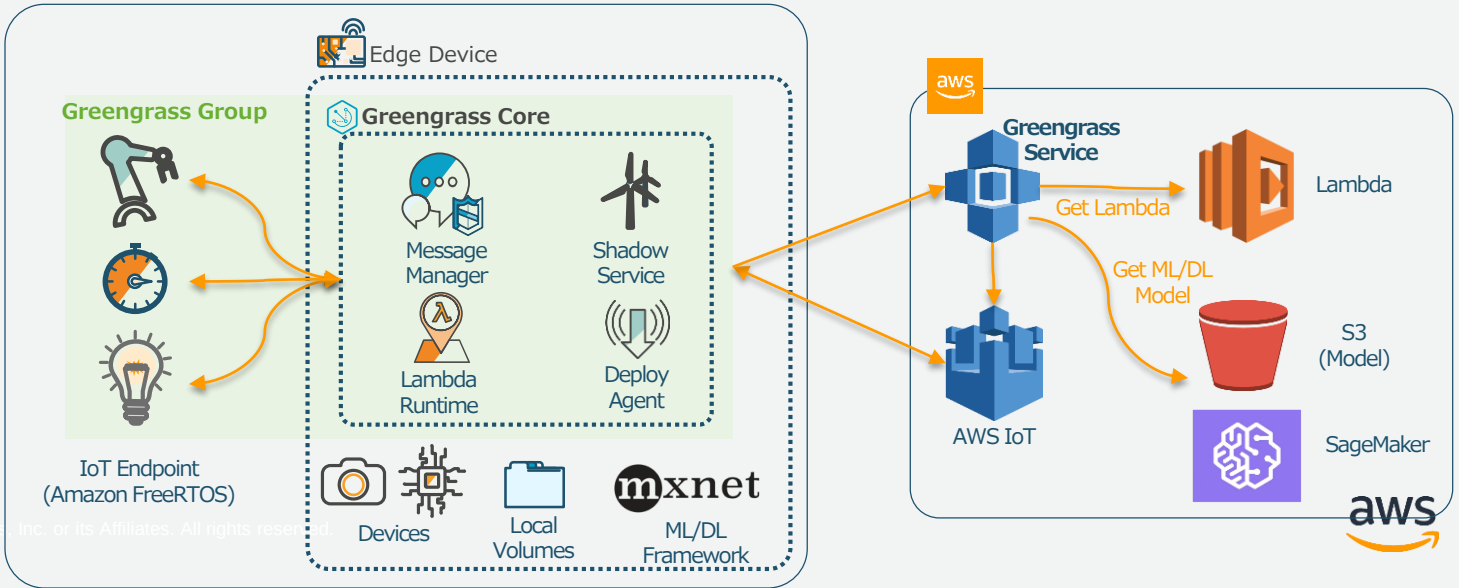
https://aws.amazon.com/jp/blogs/iot/aws-iot-driven-precision-agriculture/

Agenda

- エッジコンピューティングとは
- AWS Greengrass の詳細
- デザインパターン
- エッジコンピューティングを採用する場合の注意点
- まとめ

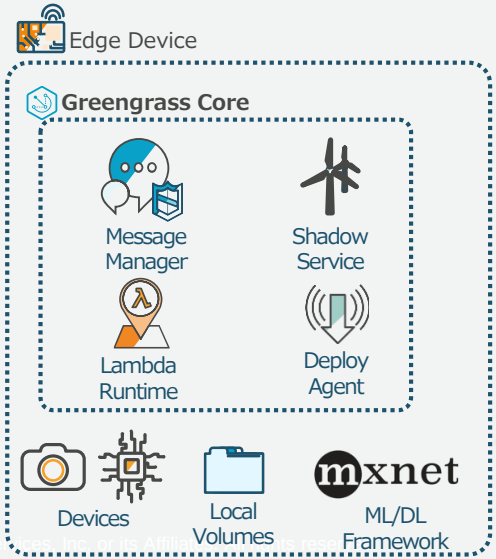
AWS Greengrass の構成要素

- Greengrass は エッジデバイスにインストールするソフトウェア(Greengrass Core)と、そのふるまいの定義(Greengrass Group)やアプリケーションコードをデプロイするクラウドサービス(Greengrass Service)で構成される



Greengrass Core

- エッジデバイスにインストールするソフトウェア
  - 起動すると常駐するプロセスとして動作する
  - 動作させるハードウェアはユーザーが要件にあわせて選択



[動作条件]
- OS: Linux kernel > 4.4
- CPU: >1GHz, Arch = ARMv7l, ARMv8, x86_64
- Memory: >128 MB RAM
- その他、Greengrass 動作に必要な依存関係などの詳細
  - https://docs.aws.amazon.com/greengrass/latest/developerguide/what-is-gg.html#gg-platforms

- Greengrass Core 上で実行したいアプリケーションが必要とするスペックを確認する
  - CPU, メモリサイズ, ストレージ容量
  - ネットワークインターフェース
  - 周辺機器 (カメラ、GPUなど)

Tips) Greengrass Core の動作条件を満たすか確認する

- Greengrassをインストールする準備が整っているかを確認するスクリプトがあります。Githubからダウンロードしてスクリプトを実行します。

```
git clone https://github.com/aws-samples/aws-greengrass-samples.git
cd aws-greengrass-samples/greengrass-dependency-checker-GGCv1.3.0/
sudo ./check_ggc_dependencies
```

- スクリプトの出力の最後に次のようなメッセージが出ていればGreengrassをインストールする準備は整っています。

```
---------------------------------Exit status-------------------------------------
You can now proceed to installing the Greengrass core 1.3 software on the device.
Please reach out to the AWS Greengrass support if issues arise.
```

AWS Greengrass

Extend AWS IoT to the edge

クラウドから配布したLambdaをローカルで実行

ローカルでのメッセージングとLambdaのトリガ

データと状態の同期(ローカルシャドウ)

クラウド同様のセキュリティポリシー

Greengrass Core自体のアップデート(OTA)

OPC-UAのプロトコルアダプタ

ローカルリソースアクセス

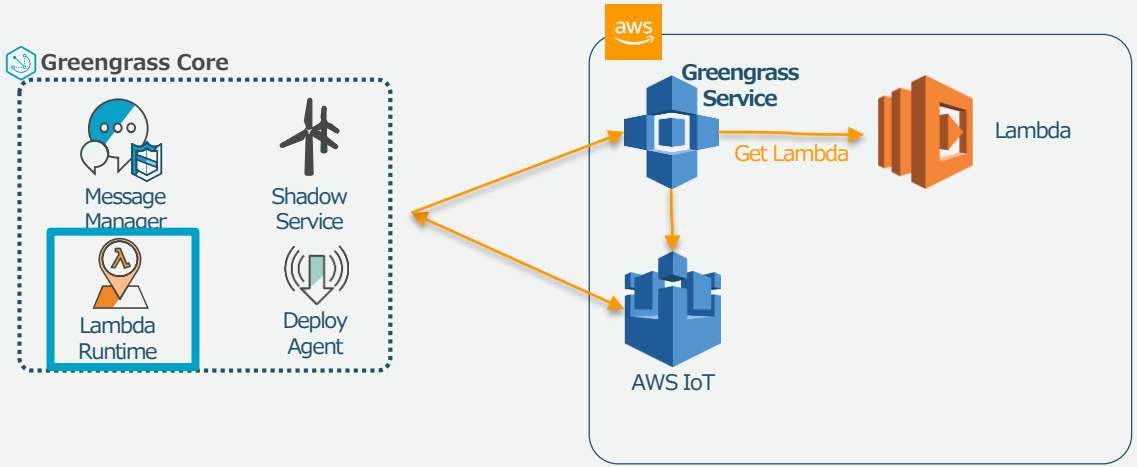
ローカルでML/DLの推論を実行

2018.4.4
GA Announce!!

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Lambda をローカルで実行

- Greengrass Core は Lambda runtime を保持し、Greengrass Service により Deploy された Lambda をエッジデバイス上で実行できる



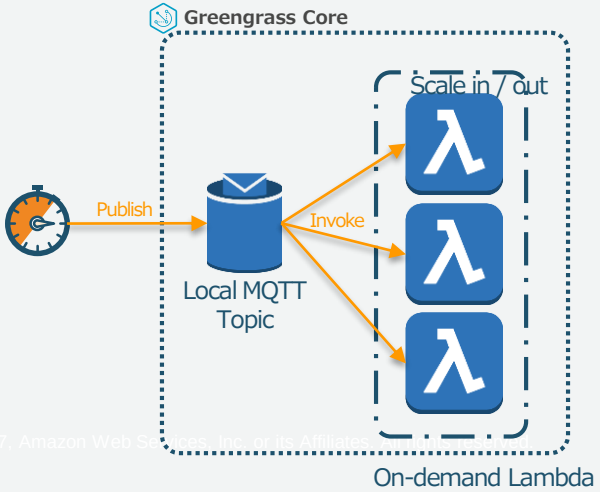
クラウドと Greengrass Lambda の違い

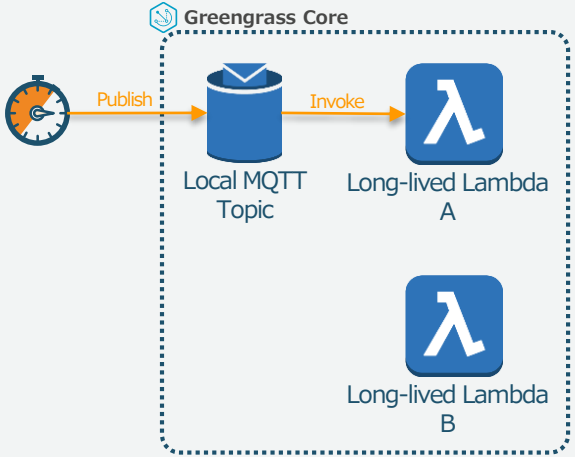
	On Cloud	On Greengrass
実行環境	Amazon Linux (amzn-ami-hvm-2017.03.1.20170812-x86_64-gp2)	GGC がインストールされているデバイスに依存
プログラミング言語	Node.js: v4.3.2, v6.10.3, v8.10 Java: 8 Python: 3.6, 2.7 .NET Core: 1.0.1, 2.0 Go: 1.x	Node.js: v6.10 Python: 2.7 Java: 8
イベントソース	S3, DynamoDB, Kinesis, …(*1)	Local MQTT, GGC 起動時に自動起動
タイムアウト設定	最大 300s	上限なし (常駐を指定可能)
メモリ設定	最小 128MB 最大 3008MB	上限なし (デバイスに依存)
ローカルリソースアクセス (デバイス or ボリューム)	不可	可 (ホワイトリスト方式)
課金	実行回数と実行時間、割り当てたメモリ量に応じて課金	無料(*2)

*1: https://docs.aws.amazon.com/ja_jp/lambda/latest/dg/invoking-lambda-function.html
*2: Greengrass 自体の課金は発生します

Lambda の Lifecycle (1/3)

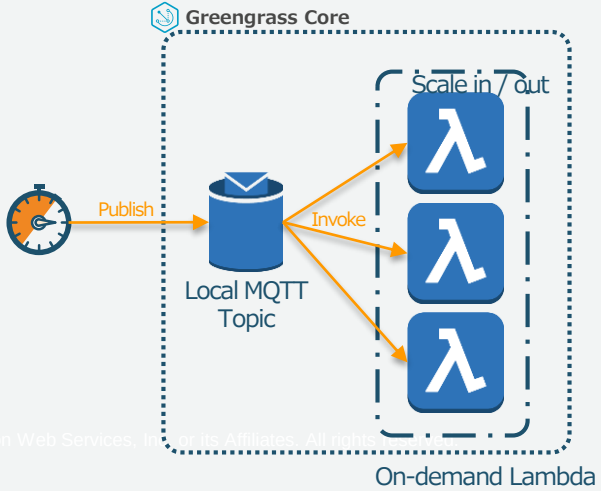
- Greengrass Core で動作させる Lambda は 次の二種類の実行方式を選択することができる
  - On-demand: Local の MQTT メッセージをトリガーに起動する実行方式。クラウドの Lambda の特性と近い
  - Long-lived: Greengrass Core の起動と同時に実行する実行方式。Greengrass Core 上で動かす Lambda 特有の機能





Lambda の Lifecycle (2/3)

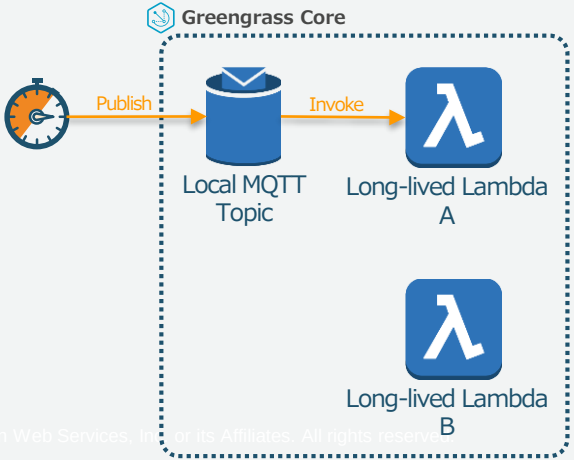
- On-demand Lambda の特徴
  - On-demand Lambda に向けたメッセージが発行されると コンテナが起動し、ハンドラ関数でメッセージを処理する
  - メッセージが溜まっていく/減っていくと自動でスケールする
  - メッセージが連続している場合、同じコンテナを使い回す(メッセージのたびに新規のコンテナを作るわけではない)



[注意点]
- クラウドと異なり、ハードウェア制約があるため、想定されるメッセージ量に対して Lambda が並行起動しても性能上問題ないか負荷試験で確認する必要がある
- クラウドと同じく、どのコンテナに割り振られるかは制御できないので、ステートレスで実装する必要がある

Lambda の Lifecycle (3/3)

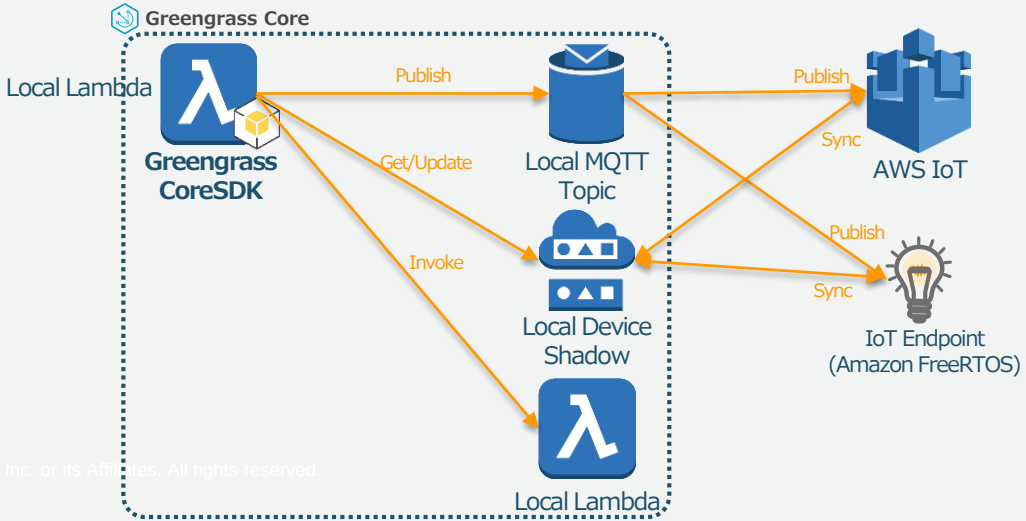
- Long-lived Lambda の特徴
  - Long-lived Lambda は Greengrass Core の起動時に一つのコンテナが起動し、初期化処理を行う。その後メッセージが発行されるとハンドラ関数でメッセージを処理する
  - 初期化処理でループ処理を実装することで、定期的に処理を実行する Lambda を作成することができる。この場合タイムアウトは発生しない



[注意点]
- メッセージの受信間隔は、Lambda の処理時間より長く設定する必要がある (短いとキューにタスクがたまり、いずれエラーが発生する)
- 初期化処理でループさせた場合、メッセージを受信することはできない

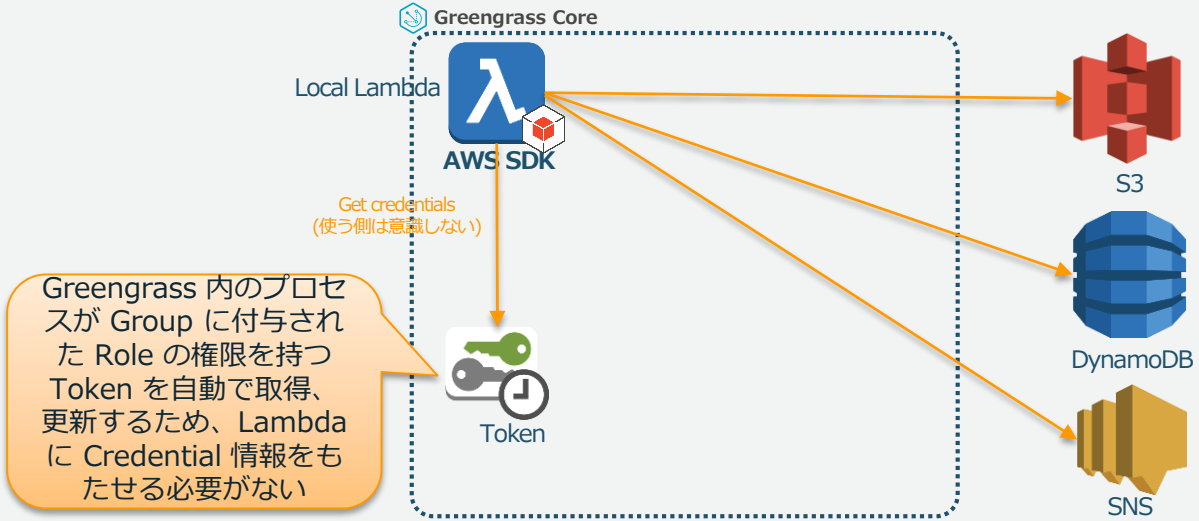
Lambda から Greengrass Core の機能にアクセス

- Greengrass Core SDK を Lambda に組み込むことで、
  Greengrass Core が提供する下記機能を Lambda から利用できる
  - Greengrass Core に対するメッセージの publish
  - Greengrass Core に対するシャドウの get/update
  - Greengrass Core 上の Lambda を直接 invoke



Lambda から AWS の各種サービスにアクセス

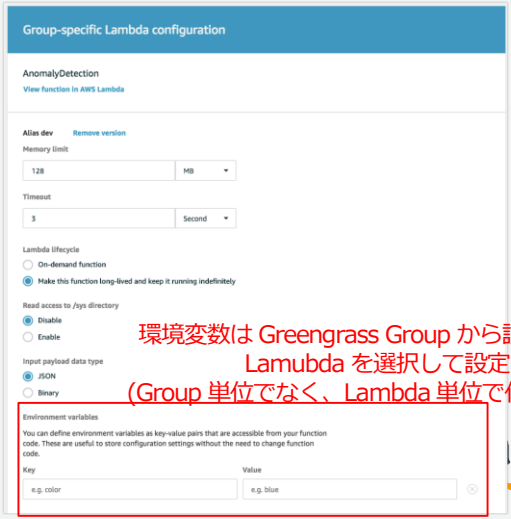
- Greengrass Group に設定した Role が持つ権限を利用して、Lambda から直接 AWS の各種サービスにアクセスすることができる



Tips) ローカル Lambda の 各種設定

- Greengrass Core にデプロイする Lambda は、Lambda のコンソールから作成しますが、そこで設定した Lambda の メモリ上限、タイムアウト、Role 及び環境変数は Greengrass Core 上で動作させた際には反映されません。
- 必要な各種設定は、Greengrass のコンソールから行う必要があります。(CLI も利用可)

Role は Greengrass Group の Settings から設定
(Lambda 単位でなく、Group の単位で付与される)



環境変数は Greengrass Group から設定したい Lamubda を選択して設定
(Group 単位でなく、Lambda 単位で付与される)

AWS Greengrass

Extend AWS IoT to the edge

クラウドから配布したLambdaをローカルで実行

ローカルでのメッセージングとLambdaのトリガ

データと状態の同期(ローカルシャドウ)

クラウド同様のセキュリティポリシー

Greengrass Core自体のアップデート(OTA)

OPC-UAのプロトコルアダプタ

ローカルリソースアクセス

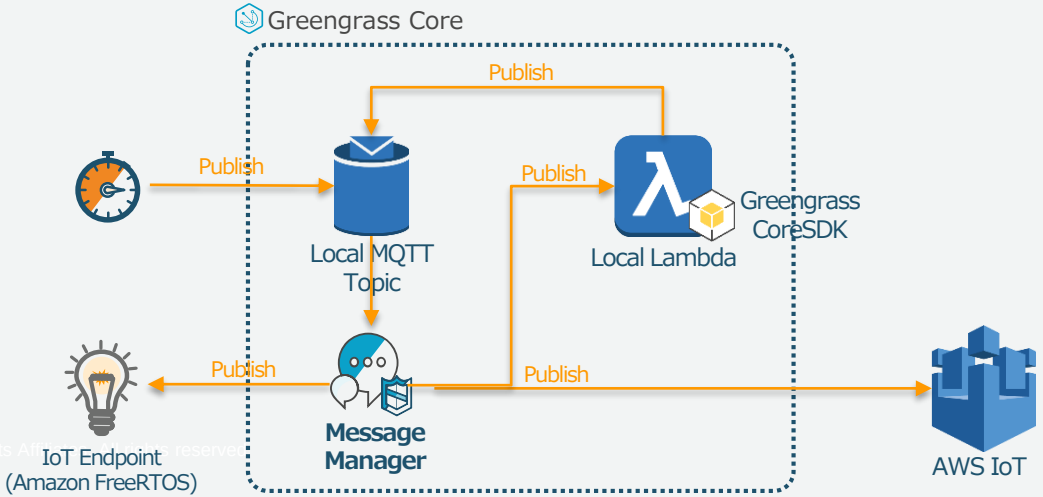
ローカルでML/DLの推論を実行

2018.4.4
GA Announce!!

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

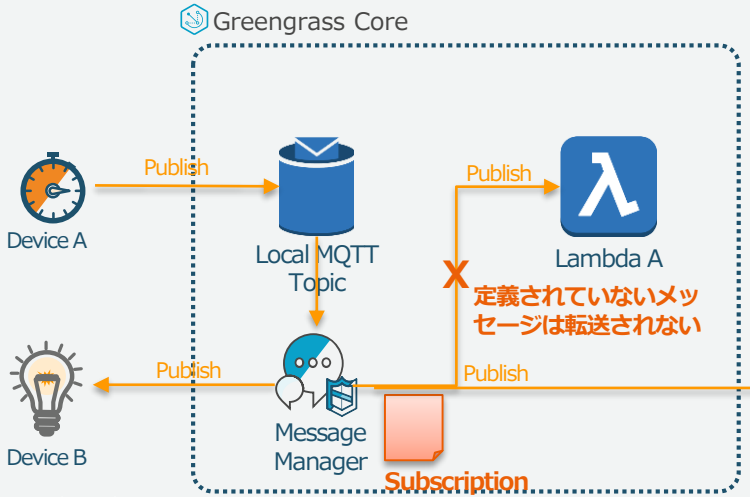
ローカルでのメッセージング

- Greengrass Core は MQTT のメッセージブローカーとして動作し、ローカルのデバイス間のメッセージや、クラウドへのメッセージの中継を行う
- Greengrass Core は 指定された MQTT Topic にメッセージが publish されたことをトリガーに、ローカルの Lambda を実行する



Subscription

- メッセージの Routing table を定義できる
  - デバイス、Lambda、クラウド間でのメッセージのやりとりをホワイトリスト方式で予め定義することで、許可されないメッセージのやりとりがローカルで発生することを防ぐ

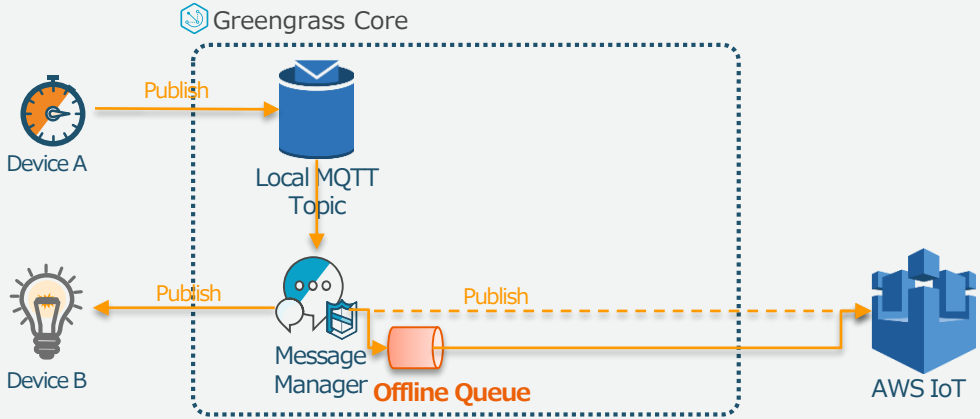


Subscription の定義

Source	Target	Topic
Device A	Device B	/DeviceB/light
Device A	IoT Cloud	/DeviceA/log

クラウド転送におけるオフラインキャッシュ

- 一時的にオフラインになると、クラウドへ転送すべきメッセージはローカルのキューにキャッシュされる (最大 2.5MB*1)
- キャッシュがフルになると古いメッセージから削除されるため、オフラインの期間が長いアプリケーションの場合、本機能に頼るのではなく、ローカルストレージにデータを保存する必要がある



*1: https://docs.aws.amazon.com/ja_jp/general/latest/gr/aws_service_limits.html#limits_greengrass

AWS Greengrass

Extend AWS IoT to the edge

クラウドから配布したLambdaをローカルで実行

ローカルでのメッセージングとLambdaのトリガ

データと状態の同期(ローカルシャドウ)

クラウド同様のセキュリティポリシー

Greengrass Core自体のアップデート(OTA)

OPC-UAのプロトコルアダプタ

ローカルリソースアクセス

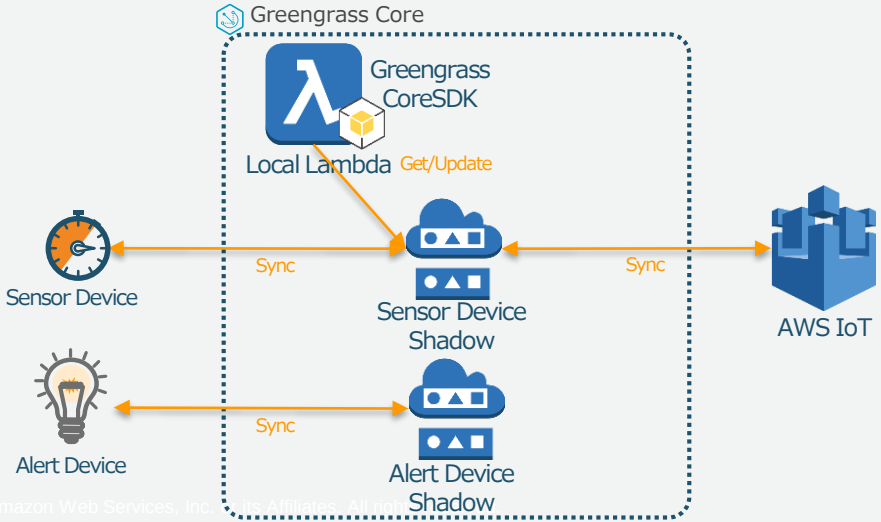
ローカルでML/DLの推論を実行

2018.4.4
GA Announce!!

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

ローカルシャドウ

- Greengrass Core は ローカルのデバイスシャドウを管理する
  - AWS IoT が提供するデバイスシャドウの機能と基本的には同じ*1
  - 設定によりクラウドのデバイスシャドウと同期させることが可能



接続するデバイスごとにシャドウを
クラウドと同期させるか設定できる

*1: https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/iot-device-shadows.html

AWS Greengrass

Extend AWS IoT to the edge

クラウドから配布したLambdaをローカルで実行

ローカルでのメッセージングとLambdaのトリガ

データと状態の同期(ローカルシャドウ)

クラウド同様のセキュリティポリシー

Greengrass Core自体のアップデート(OTA)

OPC-UAのプロトコルアダプタ

ローカルリソースアクセス

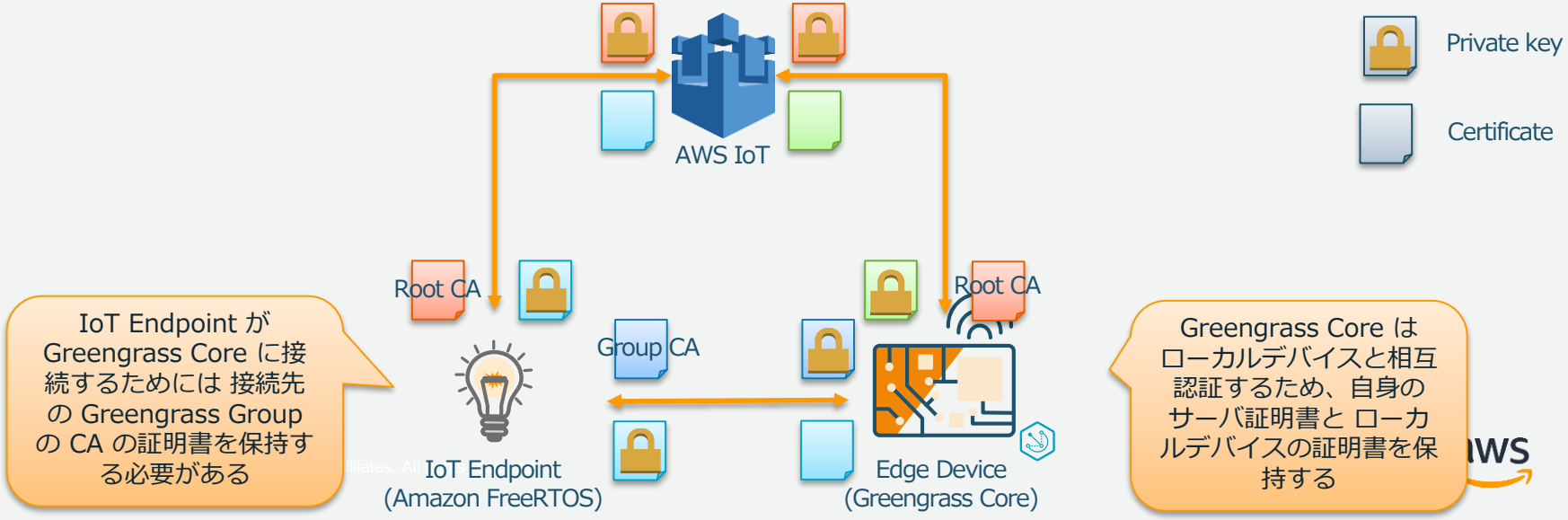
ローカルでML/DLの推論を実行

2018.4.4
GA Announce!!

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

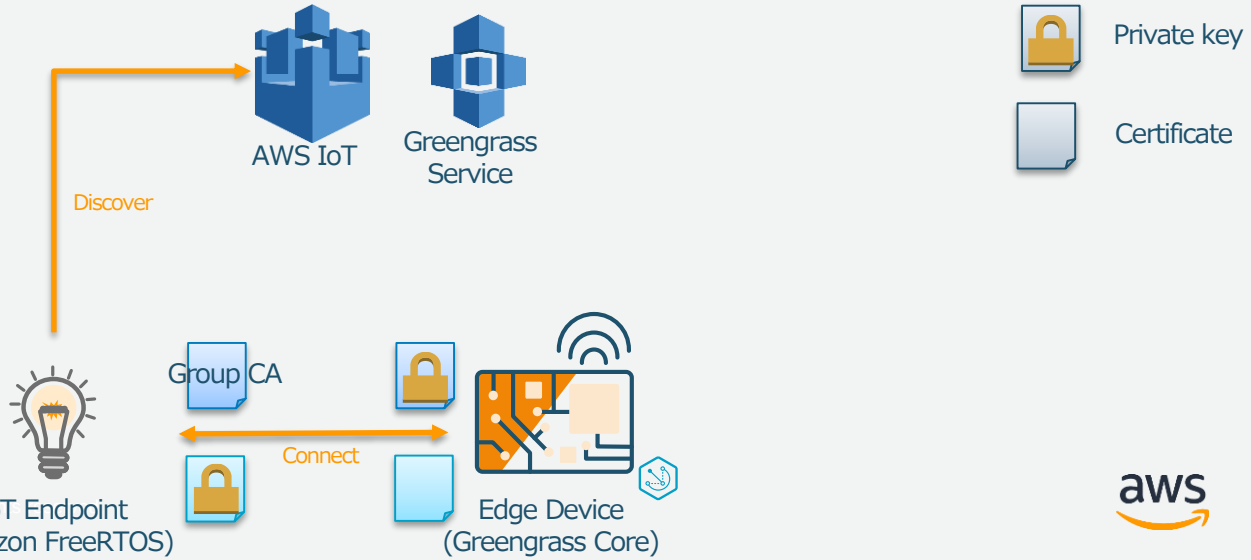
セキュリティポリシー

- Greengrass Core は AWS IoT と同様に、証明書で双方向認証を行う
- Greengrass Core は Greegrass Service が発行する Group ごとに一意の CA で認証された 証明書を保持する



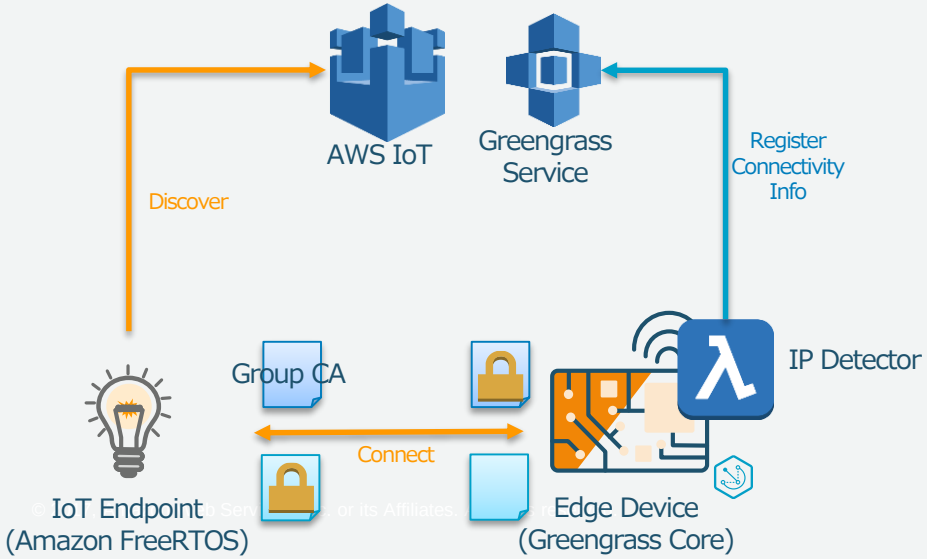
Greengrass Discovery API

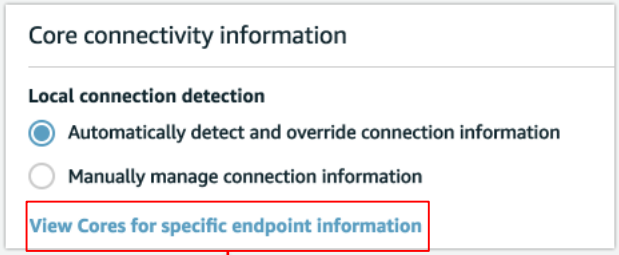
- Greengrass Core に接続するデバイスが、Greengrass の接続先情報を得るための API
- 本 API を実行することで Greengrass Core の IP アドレスもしくはホスト名、Group CA を取得することができる



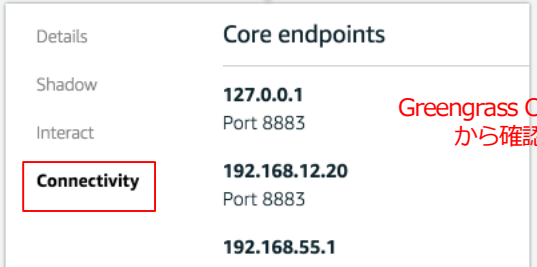
Greengrass Discoverability

- Greengrass Core の接続先情報を設定する方法は二種類
  - 自動検出：Greengrass 上で IP Detector という Lambda が常駐し、自身の IP 情報をアップロードする
  - 手動設定：コンソールや CLI で設定する。Local に DNS がある場合はこちらを利用する





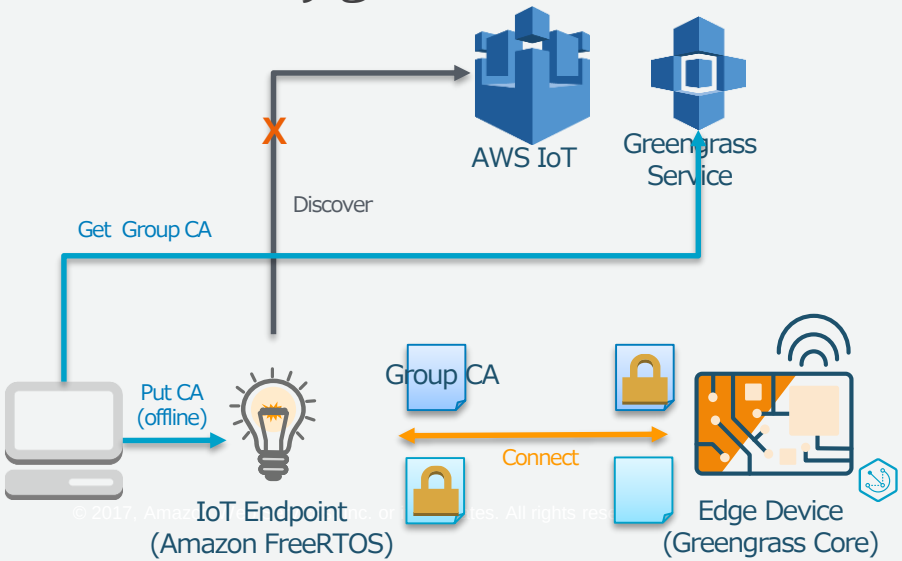
Group の Settings で設定する



Greengrass Core の Connectivity から確認、編集が可能

IoT Endpoint がネット接続不可のケース

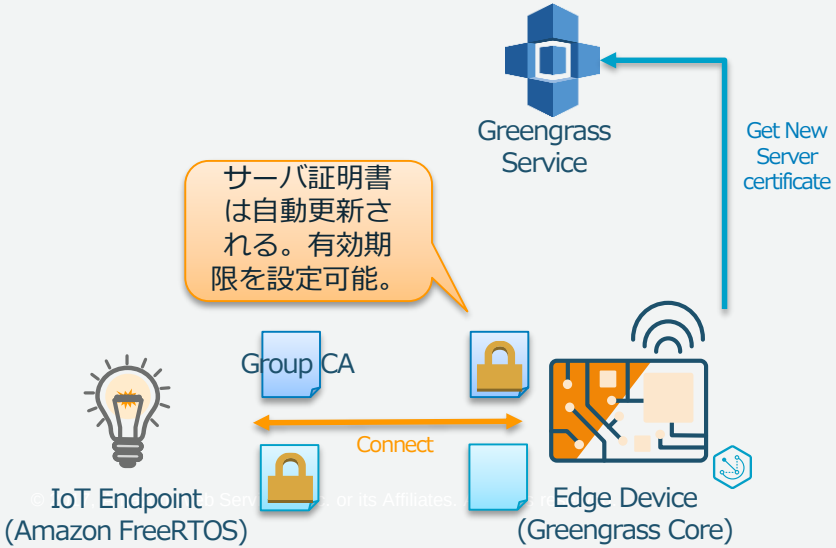
- IoT Endpoint が Discover API を呼び出せない場合、事前に接続先情報とGroup CAの情報をオフラインで渡しておく
  - Group CA は AWS CLI コマンドにより取得が可能
  - 接続先情報が変わらないことは前提になるので固定 IP or Local DNSを導入する



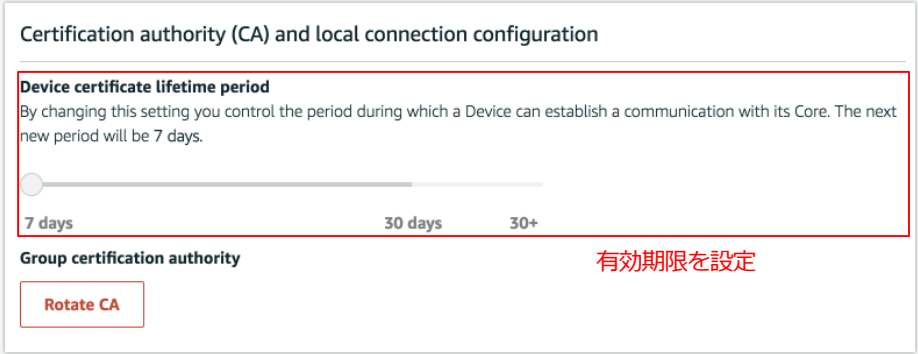

```
$ aws greengrass list-group-certificate-authorities --group-id 92c8ebae-6bf2-4149-8f9a-5709194d4920
{
    "GroupCertificateAuthorities": [
        {
            "GroupCertificateAuthorityArn": "arn:aws:greengrass:us-west-2:202621060708:/greengrass/groups/92c8ebae-6bf2-4149-8f9a-
5709194d4920/certificateauthorities/920f95a33872c3f387cf4293fa28d3c41191e2cbb666ba6710bc2f0144c22612",
            "GroupCertificateAuthorityId": "920f95a33872c3f387cf4293fa28d3c41191e2cbb666ba6710bc2f0144c22612"
        }
    ]
}
$ aws greengrass get-group-certificate-authority --group-id 92c8ebae-6bf2-4149-8f9a-5709194d4920 --certificate-authority-id 920f95a33872c3f387cf4293fa28d3c41191e2cbb666ba6710bc2f0144c22612
{
    "GroupCertificateAuthorityArn": "arn:aws:greengrass:us-west-2:202621060708:/greengrass/groups/92c8ebae-6bf2-4149-8f9a-
5709194d4920/certificateauthorities/920f95a33872c3f387cf4293fa28d3c41191e2cbb666ba6710bc2f0144c22612",
    "GroupCertificateAuthorityId": "920f95a33872c3f387cf4293fa28d3c41191e2cbb666ba6710bc2f0144c22612",
    "PemEncodedCertificate": "-----BEGIN CERTIFICATE-----
¥nMIIEFTCCAv2gAwIBAgIVAOg/EHDcUJfbdcK3gmKCFbdzcqXVMA0GCSqGSIb3DQEB¥nCwUAMIGoMQswCQYDVQQGEwJVUzEYMBYGA1UECgwPQW1hem9uLmNvbSBJbmMuMRww¥nG
gYDVQQLDBNBbWF6b24gV2ViFNlcnZpY2VzMRMwEQYDVQQIDApXYXNoaW5ndG9u¥nMRAwDgYDVQQHDAdTZWF0dGxlMTowOAYDVQQDDDEyMDI2MjEwNjA3MDg6OTJjOGV¥nY
WUtNmJmMi00MTQ5LThmOWEtNTcwOTE5NGQ0OTIwMCAXDTE4MDIyNzA5MDYyNVoY¥nDzIwOTgwMjI3MDkwNj...
```

Greengrass Core の証明書更新

- Greengrass Core はサーバ証明書を自動更新する
  - 有効期限を設定できる。
  - 更新は有効期限を過ぎる前に自動で行われる。
  - 有効期限内に更新できないと失効するので、Greengrass Core のオフライン間隔から有効期限を設定する (できるだけ短い期間を推奨)

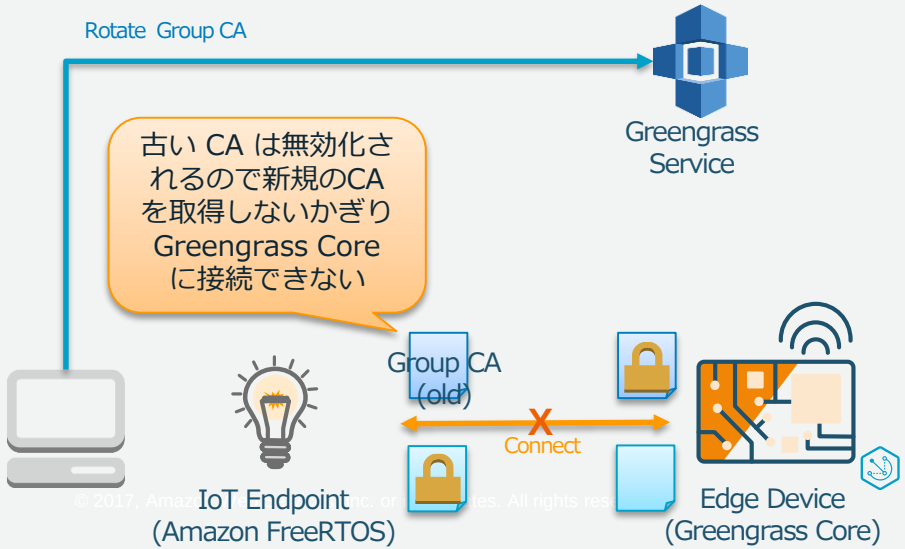


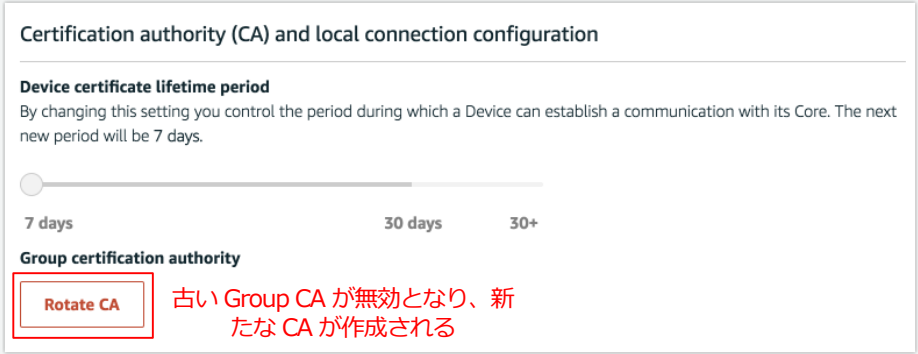
Group の Settings で設定する



Group CA の Rotate

- Greengrass Core のサーバ証明書が Leak した場合、 Group CA を更新することができる
  - 自動的に古い Group CA で署名されたサーバ証明書が無効になる
  - マネジメントコンソールもしくは CLI で更新可能





AWS Greengrass

Extend AWS IoT to the edge

クラウドから配布したLambdaをローカルで実行

ローカルでのメッセージングとLambdaのトリガ

データと状態の同期(ローカルシャドウ)

クラウド同様のセキュリティポリシー

Greengrass Core自体のアップデート(OTA)

OPC-UAのプロトコルアダプタ

ローカルリソースアクセス

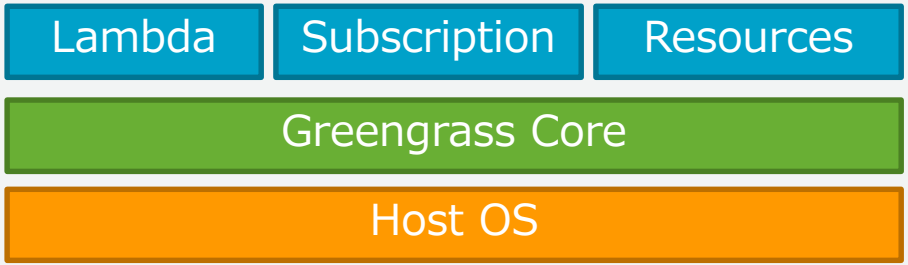
ローカルでML/DLの推論を実行

2018.4.4
GA Announce!!

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

Greengrass Core 自体の OTA

- Greengrass Core 自体をクラウドからアップデートできる
- ユーザの任意のタイミングで実行できるため、Major / Minor / Patch アップデートなど、影響の大きさを考慮のうえ、QA を実施後リリースする



Deployments

Over the Air (OTA) updates

AWS Greengrass

Extend AWS IoT to the edge

クラウドから配布したLambdaをローカルで実行

ローカルでのメッセージングとLambdaのトリガ

データと状態の同期(ローカルシャドウ)

クラウド同様のセキュリティポリシー

Greengrass Core自体のアップデート(OTA)

OPC-UAのプロトコルアダプタ

ローカルリソースアクセス

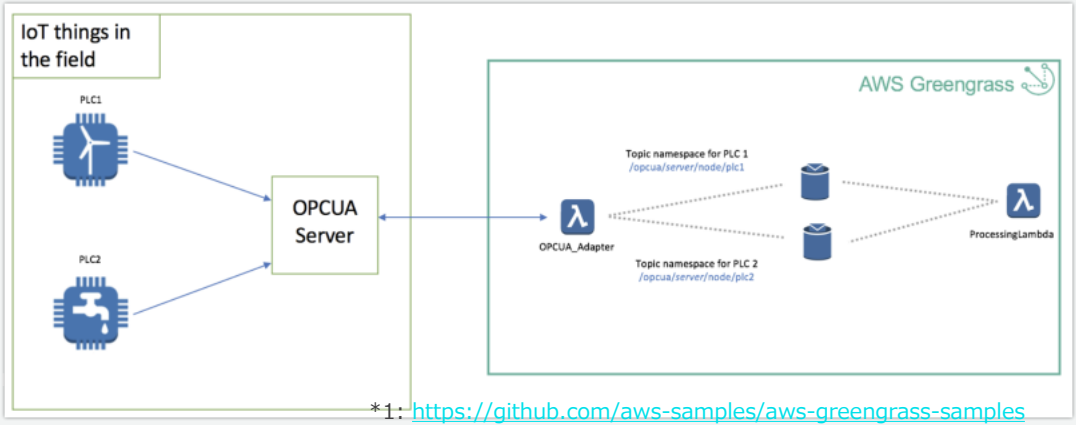
ローカルでML/DLの推論を実行

2018.4.4
GA Announce!!

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

プロトコルアダプタ

- MQTT に対応していないローカルのデバイスの Sense/Act を行うため、Greengrass 上の Lambda が代わりにローカルのデバイスとデータのやり取りを行う
- 最初のサンプルとして産業用通信プロトコルである OPC-UA をサポート
- Github にて公開*1 されておりカスタマイズが可能



*1: https://github.com/aws-samples/aws-greengrass-samples

AWS Greengrass

Extend AWS IoT to the edge

クラウドから配布したLambdaをローカルで実行

ローカルでのメッセージングとLambdaのトリガ

データと状態の同期(ローカルシャドウ)

クラウド同様のセキュリティポリシー

Greengrass Core自体のアップデート(OTA)

OPC-UAのプロトコルアダプタ

ローカルリソースアクセス

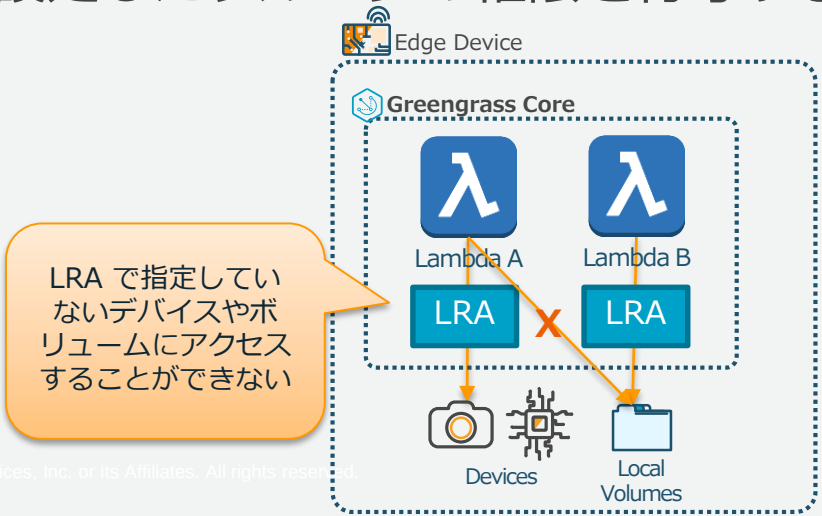
ローカルでML/DLの推論を実行

2018.4.4
GA Announce!!

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

ローカルリソースアクセス (LRA)

- Greengrass 上の Lambda から host OS の デバイスもしくはボリュームにアクセスできる
- 権限はホワイトリスト方式で Lambda ごとに付与される
- Lambda の実行ユーザに、アクセス先のリソースのグループ権限もしくは設定したグループの権限を付与することができる



ローカルリソースアクセスのユースケース

リソース名	タイプ	ユースケース	指定例
カメラ	Device	Lambda からデバイスのカメラの画像を取得する	/dev/video0
GPIO	Device	Lambda から GPIO にアクセスし、接続しているセンサ情報を取得する	/dev/mem
GPU	Device	Lambda からデバイスの GPU にアクセスして Deep Learning の推論処理を実行する	/dev/nvidia0
フォルダ	Volume	Lambda A がローカルデバイスからのメッセージを定期的にローカルストレージに書き出し、 Lambda B が定期的にそのログファイルをクラウドにアップロードする	/home/pi/greengrassAppSample/output/
実行バイナリ	Volume	Lambda から事前に配置した実行バイナリを呼び出す	/home/pi/sample-cpp/build/bin

ローカルリソースアクセス利用時の注意点

- ボリュームリソースとして /sys, /proc, /dev, /var 以下は指定できない
- Lambda の実行ユーザに管理者権限を付与することはできない
- Greengrass は起動時やデプロイ時に指定されたリソースの存在を確認するため、例えば USB デバイスにアクセスする場合など、その USB デバイスが Greengrass Core デバイスに接続されていないと Greengrass 自体の起動ができない

AWS Greengrass

Extend AWS IoT to the edge

クラウドから配布したLambdaをローカルで実行

ローカルでのメッセージングとLambdaのトリガ

データと状態の同期(ローカルシャドウ)

クラウド同様のセキュリティポリシー

Greengrass Core自体のアップデート(OTA)

OPC-UAのプロトコルアダプタ

ローカルリソースアクセス

ローカルでML/DLの推論を実行

2018.4.4
GA Announce!!

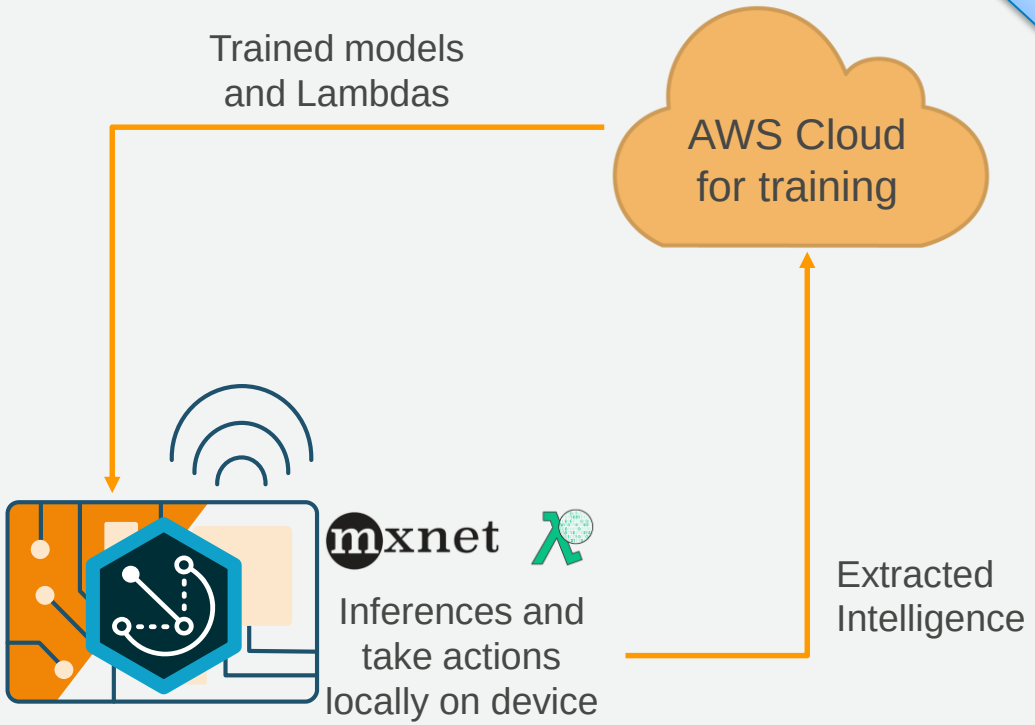
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

AWS Greengrass ML Inference

2018.4.4 GA

エッジ上での機械学習の実行を簡単にする各種機能を提供

- クラウドで学習したモデルを簡単にデプロイ可能
- MXNet/TensorFlowをデバイスに簡単に組み込める
- ローカルで機械学習の推論を行うための実装例を提供
- GPU/FPGA活用が可能

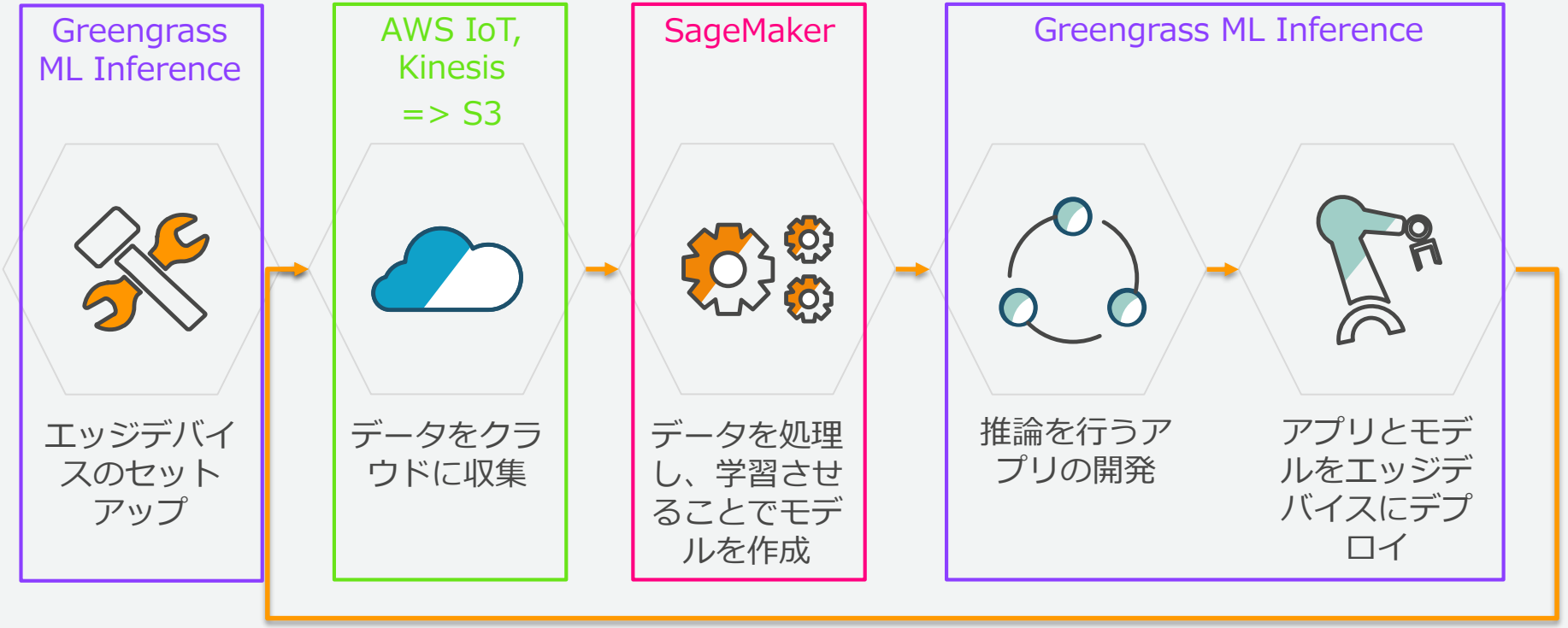


© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

エッジで機械学習を実行するために必要な開発フロー

リリース後も継続して精度改善を行いデプロイしていく

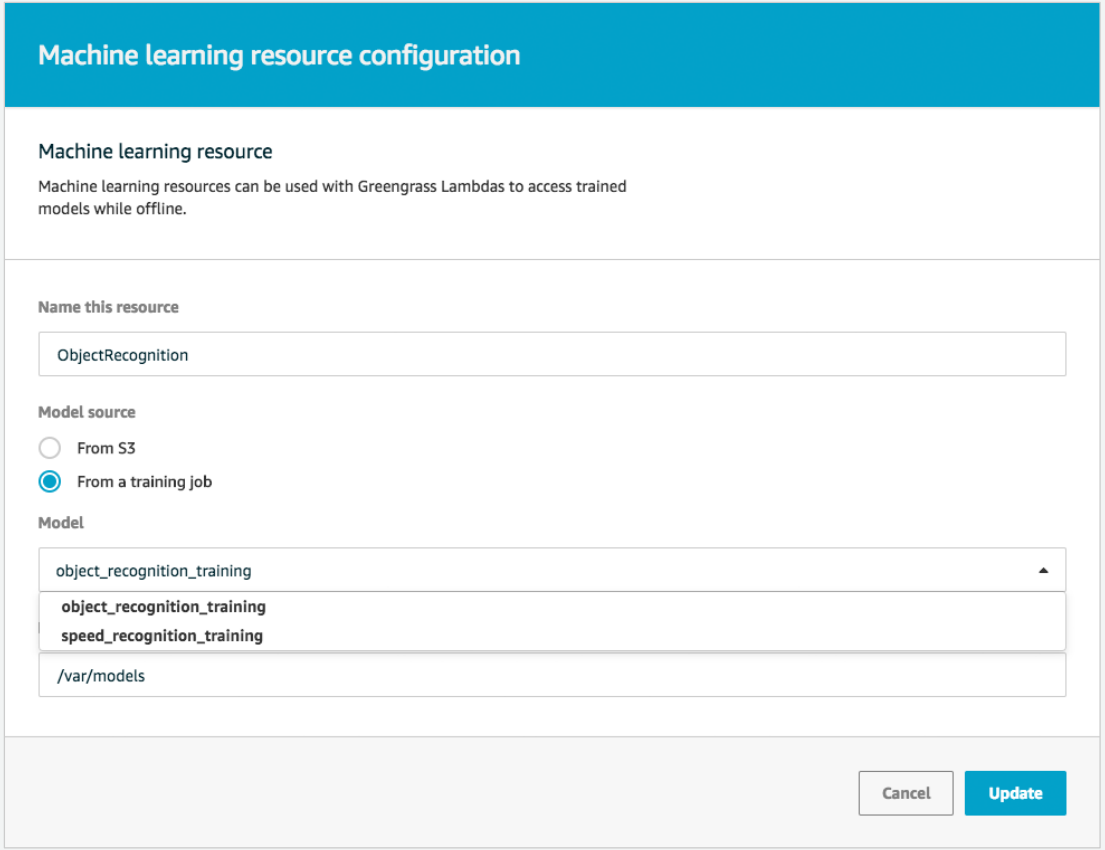
開発フローに対する AWS サービス



学習済みモデルのデプロイ

クラウドで学習したモデルをデバイスにデプロイ

- 「機械学習リソース」として学習済みモデルを Greengrass グループに追加できる
- 設定したモデルが Greengrass デバイスにデプロイされる
- Greengrass コンソールから Amazon SageMaker の学習済みモデルを指定できる
- 独自のモデルを追加できる (MXNet などの ML フレームワークに依存しない)

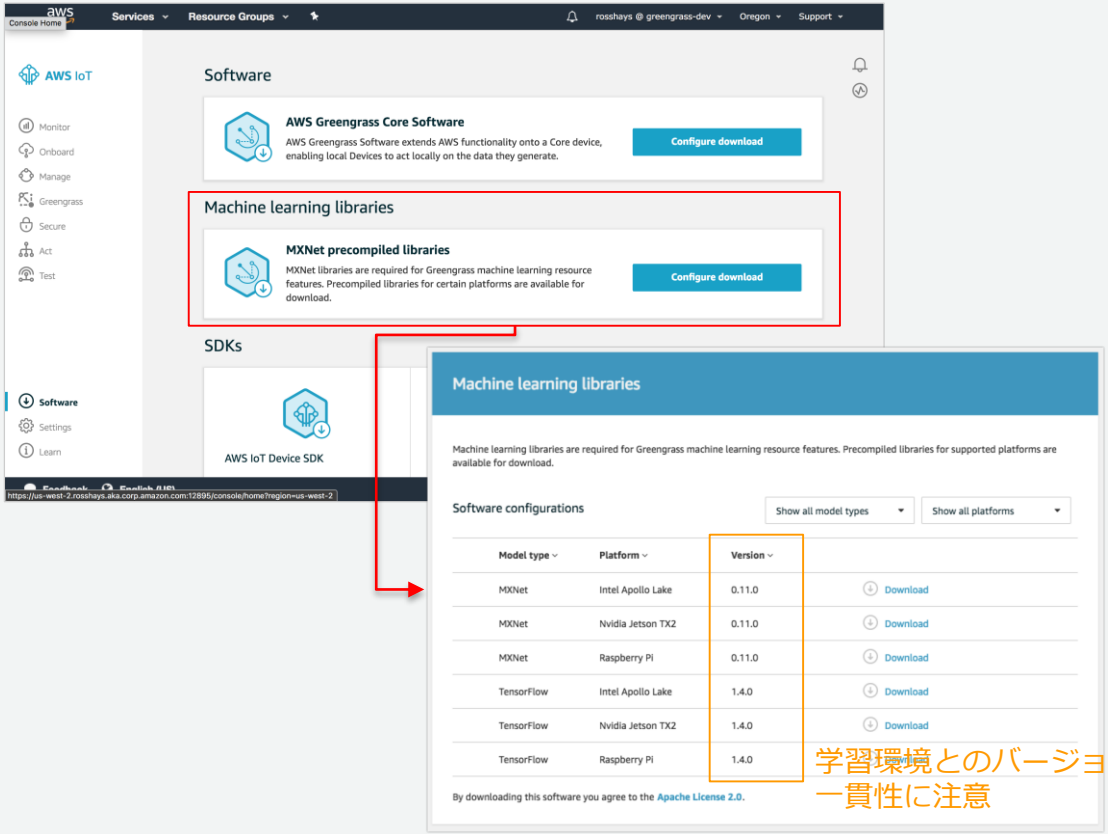


事前ビルドした MXNet/TensorFlowの提供

事前にビルドされたMXNet/TensorFlowパッケージを提供。対象デバイスでゼロからのビルドが不要に。

- Intel Atom E3900 (Apollo lake)
- NVIDIA Jetson TX2
- Raspberry Pi

その他デバイスや MXNet/TensorFlow 以外の ML フレームワークの利用を制限するものではありません。(通常のビルドを行うことで利用可能)



推論を行う Lambda のサンプルコード

推論を行う Lambda の実装例を提供

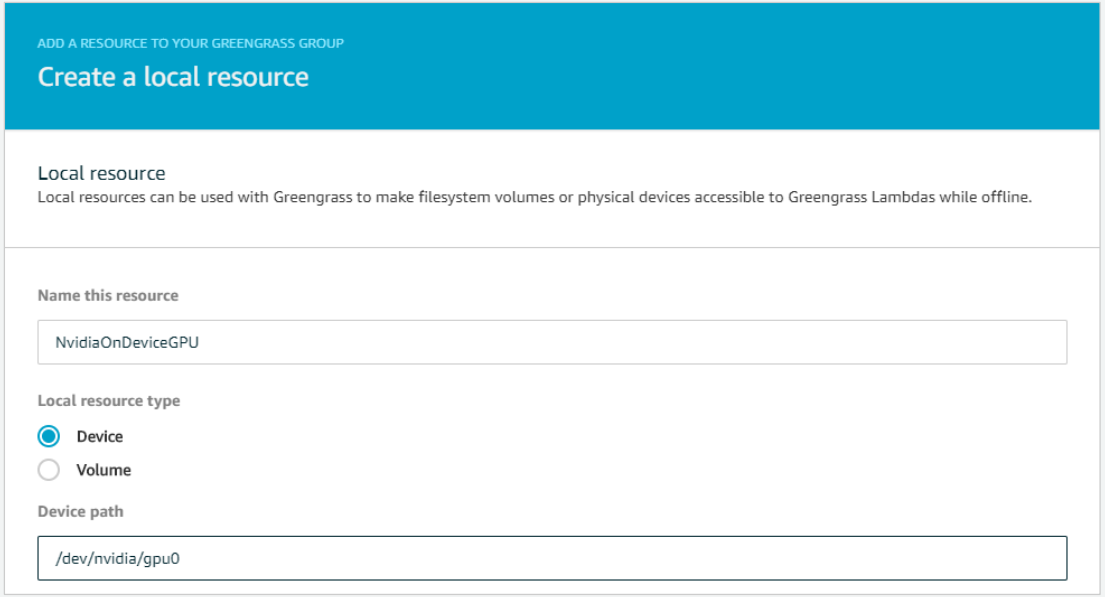
- 学習済みモデルの読み込み
- ローカルで生成されたデータをモデルに適応して推論
- 推論結果に応じたアクション

```python
# load_model_and_predict.py
import mxnet as mx
import picamera

# ...

# model files pulled down by Greengrass
_prefix = "/trained_models/" # "Local Path defined in Greengrass ML resource"
params_file = _prefix+"-0000.params"
symbol_file = _prefix+"-symbol.json"

# Load the network parameters from default epoch 0
sym, arg_params, aux_params = mx.model.load_checkpoint(_prefix, 0)

# Load the network into an MXNet module and bind the corresponding parameters
mod = mx.mod.Module(symbol=sym, label_names=label_names, context=context)
mod.bind(for_training=False, data_shapes= input_shapes)
mod.set_params(arg_params, aux_params)

# ...

# Run forward on the image
mod.forward(Batch([mx.nd.array(img)]))
```

HW アクセラレータへのアクセス

Lambda からデバイス上の GPU や FPGAといった ML アクセラレータにアクセスすることで推論を高速化

- コードの実装は不要
- 「ローカルリソース」としてアクセラレータを宣言することで Lambda からの呼び出しが可能
- カメラやストレージといった HW も同様にアクセス可能



Tips) アクセスするローカルリソースを調べる

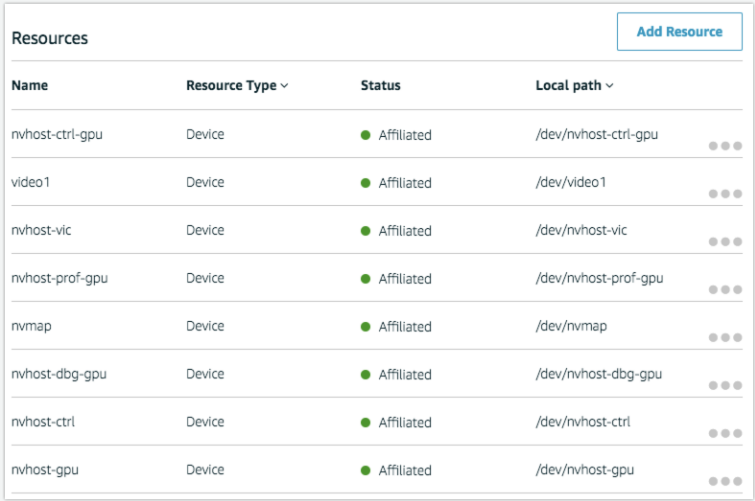
- Deep Learning フレームワークで GPU を利用する場合など、/dev 以下のどのモジュールにアクセスしているかわからないケースがある
- この場合、strace コマンドを利用し、作成した Lambda を Greengrass Core をインストールするデバイス上で直接実行することでアクセスするローカルリソースを調べることができる

```
$ strace python your_lambda.py 2>strace_output.log
$ cat strace_output.log | grep ¥"/dev
```

Tips) strace コマンドの実行と LRA の設定例

- Jetson TX2 で GPU アクセスの例 (MXNet SSD による物体検出)

```
$ strace python demo.py 2>strace_output.log
$ cat strace_output.log | grep ¥"/dev
openat(AT_FDCWD, "/dev/shm/cuda_injection_path_shm", O_RDWR|O_NOFOLLOW|O_CLOEXEC) = -1 ENOENT (No such file or directory)
newfstatat(AT_FDCWD, "/dev/nvidiactl", 0x7fd05da9f0, 0) = -1 ENOENT (No such file or directory)
mknodat(AT_FDCWD, "/dev/nvidiactl", S_IFCHR|0666, makedev(195, 255)) = -1 EACCES (Permission denied)
openat(AT_FDCWD, "/dev/nvidiactl", O_RDWR) = -1 ENOENT (No such file or directory)
faccessat(AT_FDCWD, "/dev/nvhost-gpu", R_OK|W_OK) = 0
openat(AT_FDCWD, "/dev/nvgpu-pci", O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
newfstatat(AT_FDCWD, "/dev/dri/renderD128", 0x7fd05dab30, 0) = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/dev/nvhost-ctrl-gpu", O_RDWR|O_CLOEXEC) = 3
faccessat(AT_FDCWD, "/dev/nvhost-dbg-gpu", R_OK|W_OK) = 0
faccessat(AT_FDCWD, "/dev/nvhost-prof-gpu", R_OK|W_OK) = 0
openat(AT_FDCWD, "/dev/nvhost-prof-gpu", O_RDWR|O_CLOEXEC) = 5
openat(AT_FDCWD, "/dev/nvmap", O_RDWR|O_SYNC|O_CLOEXEC) = 6
openat(AT_FDCWD, "/dev", O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = 33
newfstatat(AT_FDCWD, "/dev/video1", {st_mode=S_IFCHR|0660, st_rdev=makedev(81, 3), ...}, 0) = 0
newfstatat(AT_FDCWD, "/dev/video0", {st_mode=S_IFCHR|0660, st_rdev=makedev(81, 2), ...}, 0) = 0
openat(AT_FDCWD, "/dev/v4l2", O_RDONLY|O_NONBLOCK|O_DIRECTORY|O_CLOEXEC) = -1 ENOENT (No such file or directory)
newfstatat(AT_FDCWD, "/dev/video1", {st_mode=S_IFCHR|0660, st_rdev=makedev(81, 3), ...}, 0) = 0
openat(AT_FDCWD, "/dev/video1", O_RDWR) = 34
...
```



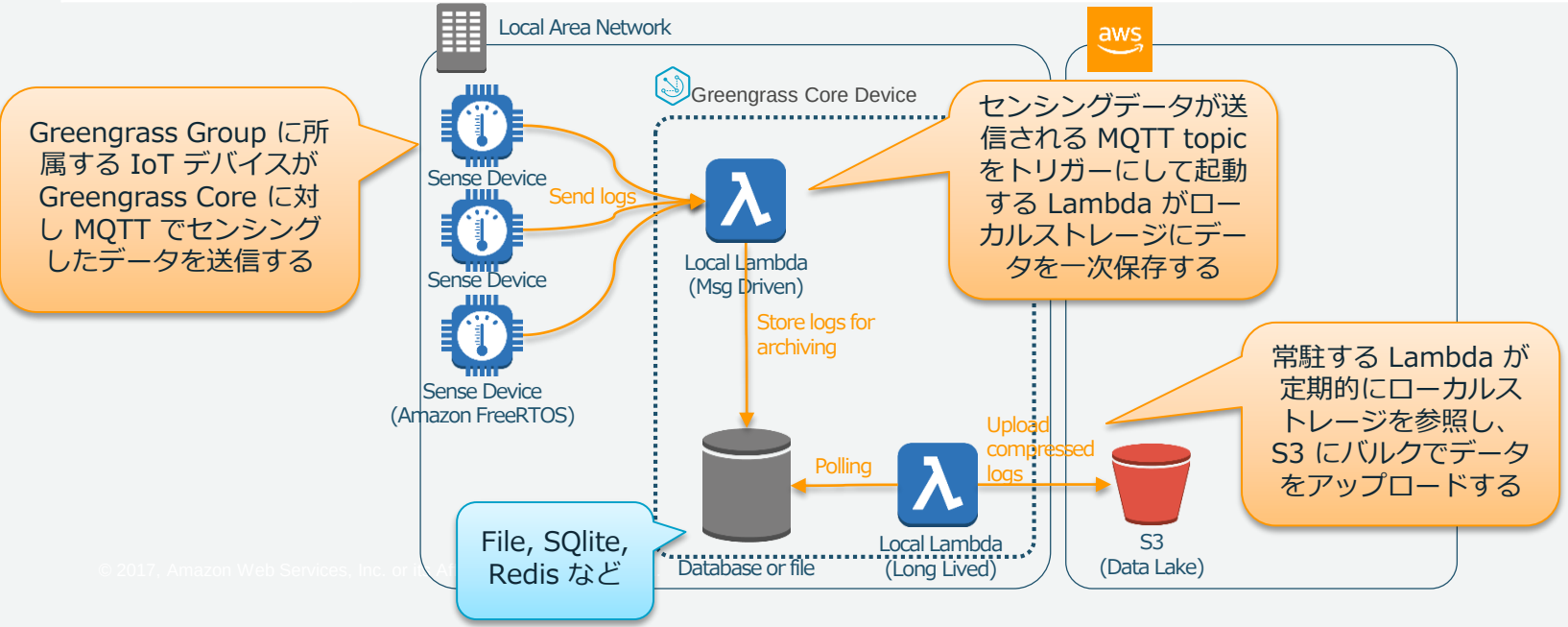
aws

Agenda

- エッジコンピューティングとは
- AWS Greengrass の詳細
- デザインパターン
- エッジコンピューティングを採用する場合の注意点
- まとめ

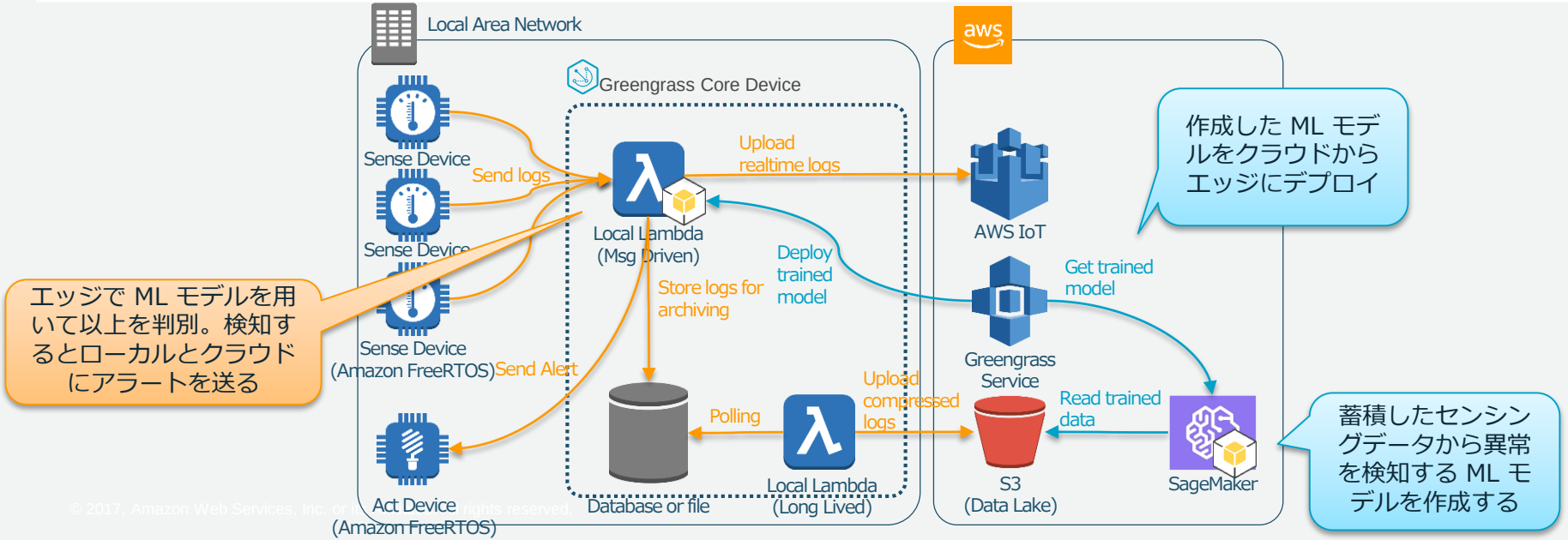
バルクアップロードパターン

対象シーン	- 帯域幅を節約したい - オンラインになる時間帯が限られている
条件	- クラウドでリアルタイムにセンサーデータを処理する必要がない
注意点	- センサーデータを保存するデバイスのストレージ容量を確保する - OnDemand Lambda で保存する場合、並列性に注意する



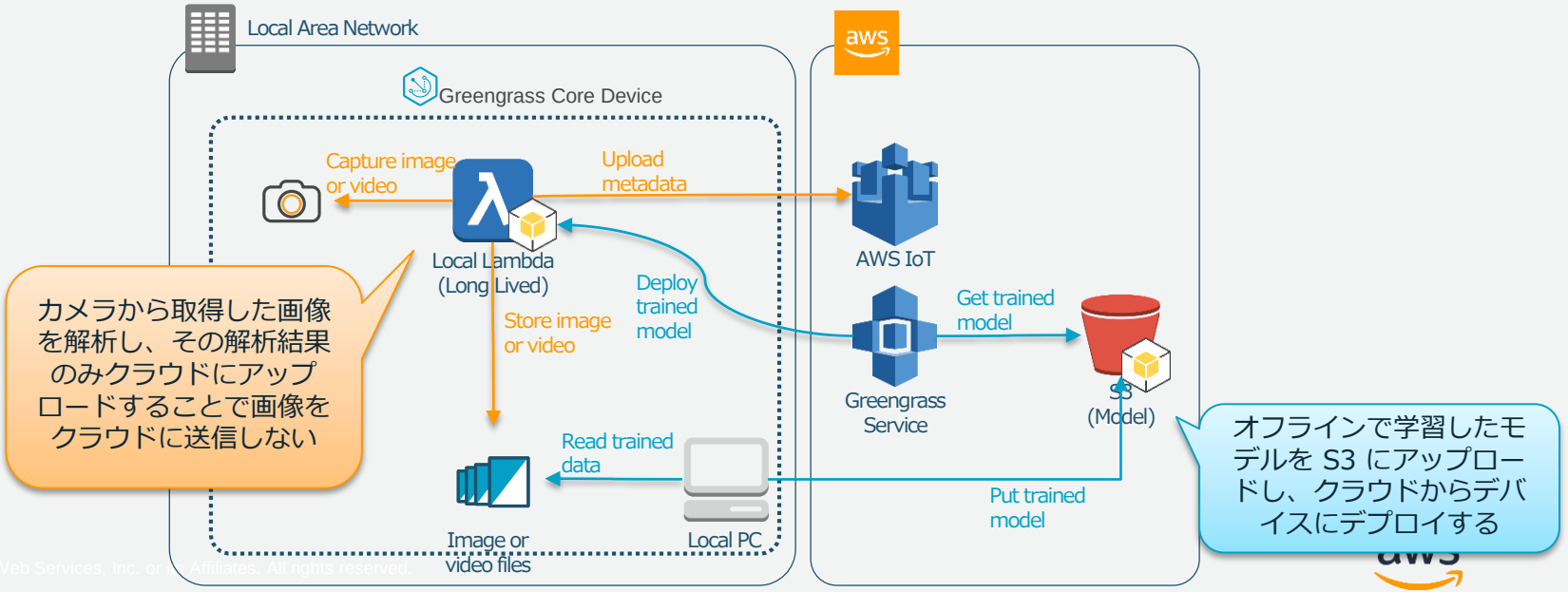
バルクアップロードパターン + 異常検知

対象シーン	- バルクアップロードパターンと同じ - 加えて、異常はリアルタイムに検知したい
条件	- バルクアップロードパターンと同じ - 異常を検知する機械学習モデルを開発している
注意点	- バルクアップロードパターンと同じ



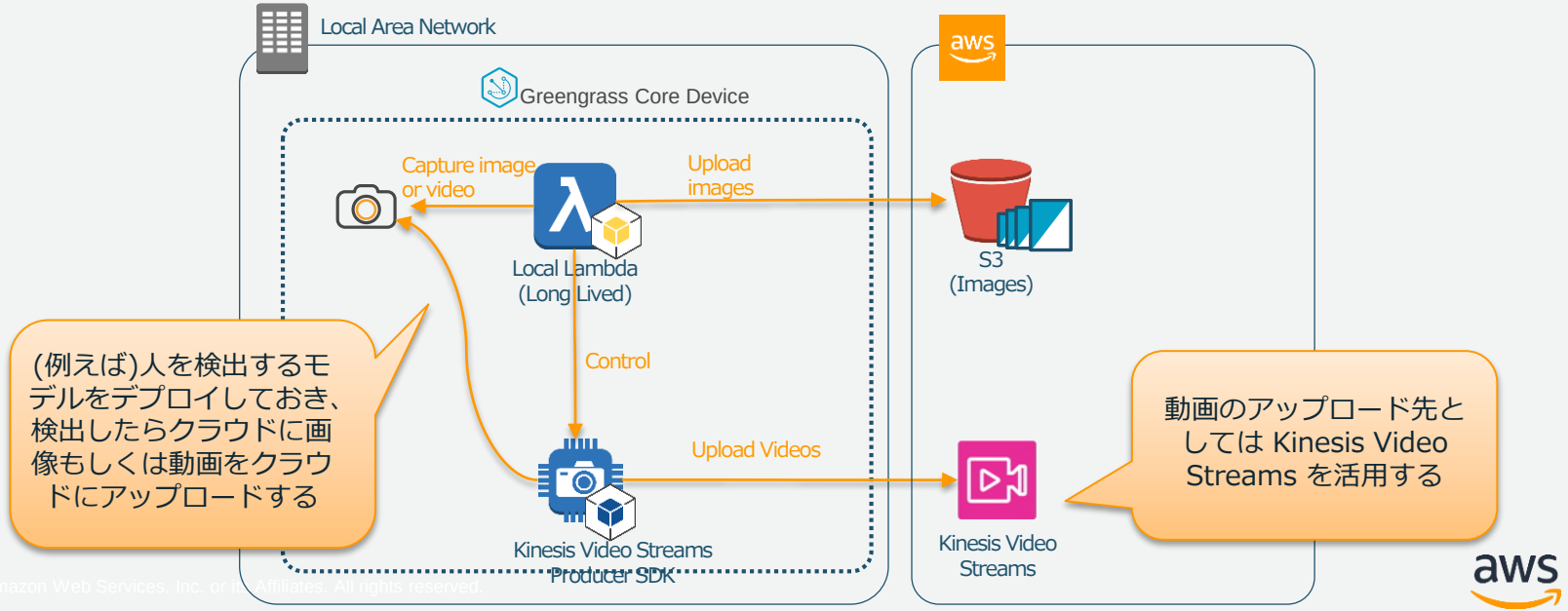
マスキングパターン

対象シーン	- コンプライアンス事情で画像や動画をネットワーク外に持ち出せない
条件	- 事前に取得したいメタ情報を取得する機械学習のモデルを開発済み
注意点	- モデルを改善する仕組みをオフラインで別途用意する



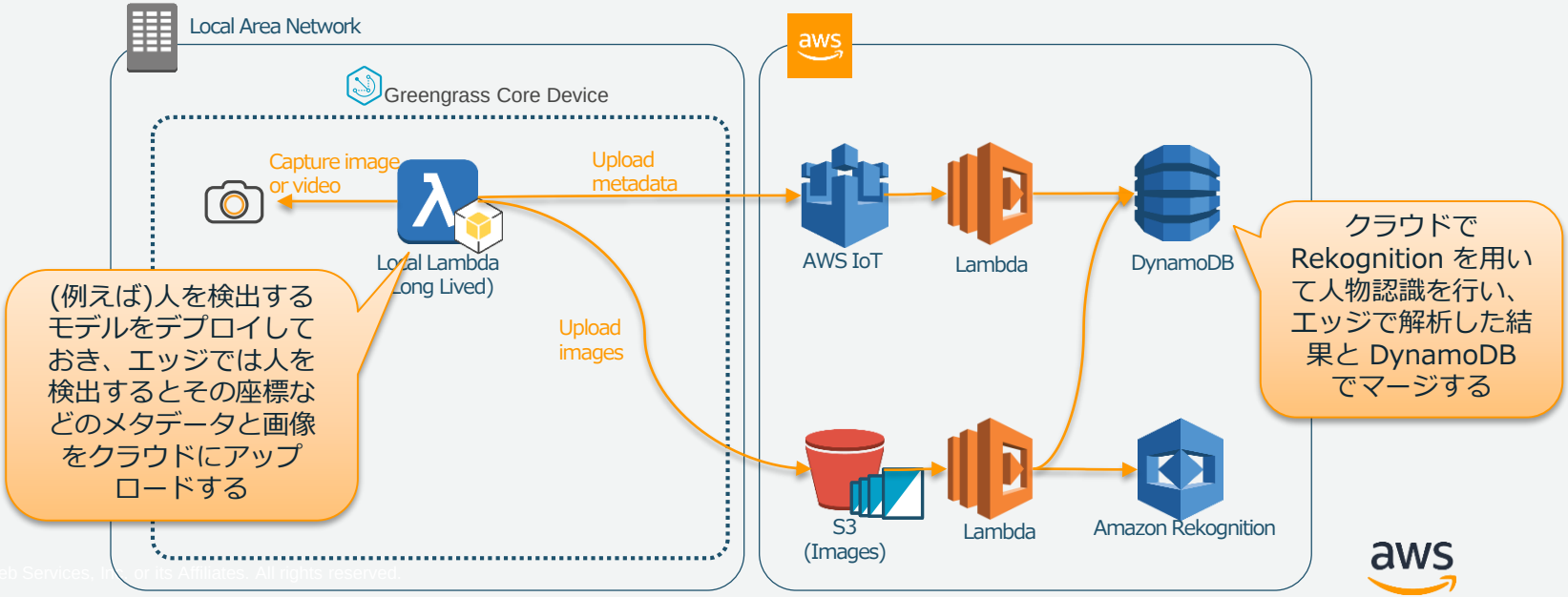
画像/動画の効率的なアップロード

対象シーン	- 条件を満たした時のみクラウドに画像や動画をアップロードしたい
条件	- 画像や動画をアップロードする条件を検出するコード/モデルを開発済み
注意点	- 条件検出モデルを改善する仕組みを別途用意する



エッジとクラウドによる協調処理パターン

対象シーン	- エッジデバイスでリアルタイムに処理(例えば人物検出)しつつ、クラウドでより詳細な解析を行いたい(例えば人物認識)
条件	- クラウドで解析したい条件をエッジで検出するコード/モデルを開発済み
注意点	- エッジのモデルを改善する仕組みを別途用意する

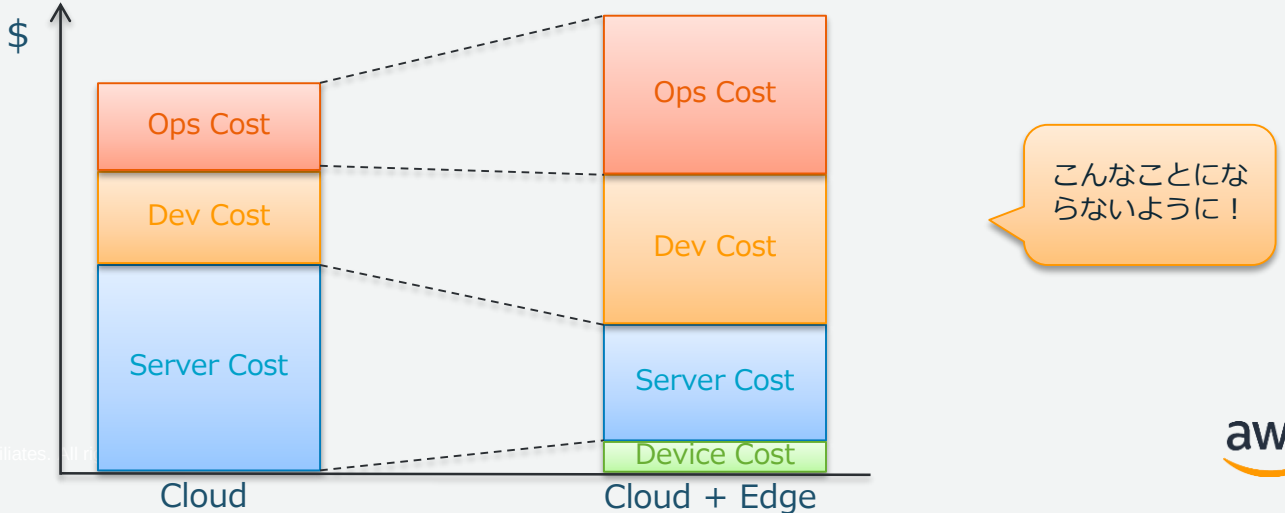


Agenda

- エッジコンピューティングとは
- AWS Greengrass の詳細
- デザインパターン
- エッジコンピューティングを採用する場合の注意点
- まとめ

エッジ活用でシステムのコストが下がった…?

- エッジデバイスは基本的に SPOF となるため、ダウンタイムの発生を考慮したうえで、障害発生時の運用を十分に準備しておく
- サービスのコストはサーバやデバイスのコストだけでなく、開発や運用にかかる工数も含めたトータルで評価し、エッジコンピューティングを活用すべきか検討する



Greengrass における瞬断のタイミング

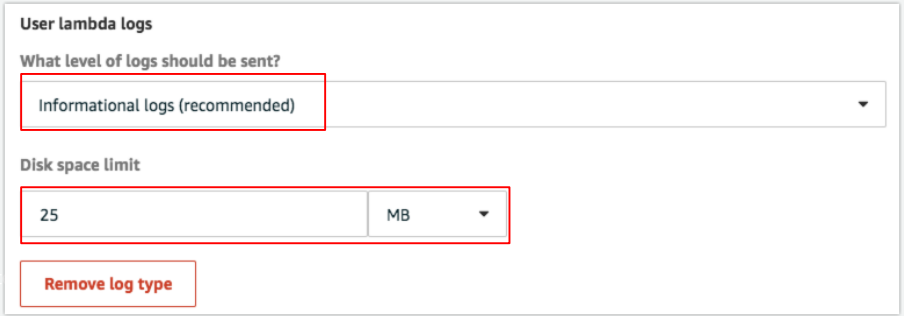
- Greengrass に対するデプロイを行う場合、変更を行っていない Lambda を含め、全ての Lambda が再起動される。そのためデプロイ中は処理が瞬断される
- 上記が許容されない場合、予めデプロイ可能なメンテナンスタイムを合意するなどのプロセスが必要

Greengrass Core デバイスを監視する

- Logging 機能を活用する
  - Greengrass のシステムログと、Lambda のログをローカルストレージもしくは CloudWatch Logs に出力できる
  - ローカルストレージへの出力時、設定済みの上限に達すると自動でローテートとクリーンアップを行う
  - CloudWatch Logs に出力しておけば、予期せぬ不具合の発生時にアラートを出すなど、監視を行うことができる
  - CloudWatch Logs に出力する場合、Greengrass Group Role にアクセス権限を忘れずに付与すること

ログレベルの設定

ログサイズ
1 lambda あたりの
サイズなので注意



Agenda

- エッジコンピューティングとは
- AWS Greengrass の詳細
- デザインパターン
- エッジコンピューティングを採用する場合の注意点
- まとめ

まとめ

- エッジコンピューティングは、クラウドでは対応できない品質特性要件(レイテンシ、ネットワーク帯域、オフライン対応、セキュリティ)がある IoT システムを構築する際に導入を検討する
- AWS Greengrass を活用することで、クラウドにおける意思決定能力やセキュリティモデルをエッジに拡張しつつ、エッジ特有の処理(リソースアクセスやローカル通信)に対応できる
- エッジコンピューティングはクラウドのその他のサービスと組み合わせることで価値をもたせることができる(機械学習など)
- エッジコンピューティングを導入することは、クラウド only に比べて複雑な運用要件をもたらす点に注意する。特に障害時対応など事前の考慮が重要

参考資料 (1/2)

- AWS Greengrass
  - https://aws.amazon.com/jp/greengrass/
- AWS Greengrass ML Inference
  - https://aws.amazon.com/jp/greengrass/ml/
- Amazon FreeRTOS
  - https://aws.amazon.com/jp/freertos/
- [The Internet of Things on AWS – Official Blog] AWS IoT-Driven Precision Agriculture
  - https://aws.amazon.com/jp/blogs/iot/aws-iot-driven-precision-agriculture/

参考資料 (2/2)

- [AWS Black Belt Online Seminar] AWS IoTでのデバイス管理、運用について
  - https://aws.amazon.com/jp/blogs/news/aws-black-belt-online-seminar-aws-iot-device-management/
- [AWS Black Belt Online Seminar] AWS で構築するデータレイク基盤のアーキテクチャ
  - https://aws.amazon.com/jp/blogs/news/webinar-bb-datalake-2018/
- [AWS Black Belt Online Seminar] Amazon SageMaker
  - https://aws.amazon.com/jp/blogs/news/aws-black-belt-online-seminar-amazon-sagemaker/
- [AWS Black Belt Online Seminar] Amazon Kinesis Video Streams
  - https://aws.amazon.com/jp/blogs/news/aws-black-belt-online-seminar-amazon-kinesis-video-streams/

オンラインセミナー資料の配置場所

AWS クラウドサービス活用資料集

- https://aws.amazon.com/jp/aws-jp-introduction/

Amazon Web Services ブログ

- 最新の情報、セミナー中のQ&A等が掲載されています。
- https://aws.amazon.com/jp/blogs/news/

公式Twitter/Facebook
AWSの最新情報をお届けします

@awscloud_jp

もしくは
http://on.fb.me/1vR8yWm

最新技術情報、イベント情報、お役立ち情報、
お得なキャンペーン情報などを日々更新しています！

AWSの導入、お問い合わせのご相談

AWSクラウド導入に関するご質問、お見積、資料請求をご希望のお客様は以下のリンクよりお気軽にご相談下さい。

https://aws.amazon.com/jp/contact-us/aws-sales/

※「AWS 問い合わせ」で検索して下さい。

AWS Well Architected 個別技術相談会お知らせ

- Well Architectedフレームワークに基づく数十個の質問項目を元に、お客様がAWS上で構築するシステムに潜むリスクやその回避方法をお伝えする個別相談会です。
  https://pages.awscloud.com/well-architected-consulting-jp.html
- 参加無料
- 毎週火曜・木曜開催

【毎週火、木曜開催】AWS Well-Architected 個別技術相談会

AWS 上で構築するシステムのリスクの把握・回避方法をご希望のお客様

この度 AWS をご活用頂いているお客様を対象に「AWS Well-Architected 個別技術相談会」を開催致します。

Well-Architected 個別技術相談会では、リスクの把握・回避を目的として、セキュリティ・信頼性・パフォーマンス・コスト・運用の５つの観点で、お客様の AWS 活用状況や構成についてお伺いします。AWS のベストプラクティスに基づき作成された Well-Architected フレームワークを元に、今までお客様がお気づきでなかったリスクやAWS活用の改善点を見つけることができます。例えば、自動車においては納車前点検、車検を定期的に行うのと同様に、本相談会はお客様の AWS 上のシステムをよりよく活用頂くことを目的にしております。

» 説明資料(PDF) [AWS Well-Architected Framework -クラウド設計・運用ベストプラクティスの活用-]

Well-Architected 個別技術相談会にご参加頂くには、本ページにてお申込み後、弊社担当者からお送りするヒアリングシートにご記入・担当者にご送付頂く必要があります。その内容を元に、当日の相談会では AWS のソリューションアーキテクトと共に技術的なディスカッションをさせて頂きます。また、遠方のお客様、アマゾン東京オフィスへのご来社が時間等の関係で難しいお客様は、Web のプレゼンテーションツールや、お電話を活用したリモートでのご相談も承ります。

下記のフォームよりお申込みください。

* 姓:

* 名: