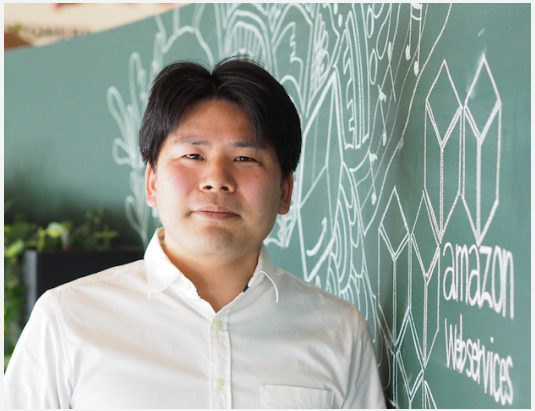
AWS
Black Belt
Online Seminar

【AWS Black Belt Online Seminar】

AWS IoTにおけるデバイス管理

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 小梁川 貴史
2018/03/27

自己紹介



名前

小梁川 貴史（こやながわ　たかし）

所属

技術統括本部 IoT/AI ソリューションビルダー
ソリューション アーキテクト

前職

電機メーカー自社サービスの開発・運用
元AWSユーザ

好きなAWSサービス

AWS IoT , AWS Lambda(python), Amazon Kinesis

AWS Black Belt Online Seminar とは

AWSJのTechメンバがAWSに関する様々な事を紹介するオンラインセミナーです

【火曜 12:00~13:00】
主にAWSのソリューションや業界カットでの使いどころなどを紹介
(例：IoT,金融業界向け etc.)

【水曜 18:00~19:00】
主にAWSサービスの紹介やアップデートの解説
(例：EC2, RDS, Lambda etc.)

※開催曜日と時間帯は変更となる場合がございます。最新の情報は下記をご確認下さい。
オンラインセミナーのスケジュール&申し込みサイト
https://aws.amazon.com/jp/about-aws/events/webinars/

内容についての注意点

- 本資料では2018年03月27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.

はじめに

IoTの特性として、小さなPoC=数点のデバイスからstartし、大量のデバイスを管理することがワークロード上に発生することがある。
また、接続情報やセキュリティ情報を埋め込んだ”機器”を配置/管理することが発生します
また機器が配置される場所は、ITの現場とは遠い生産現場やお客様の手元にあるものということもあります

本セッションにおいて、AWSにおけるIoTデバイスのセキュア/効率的な管理方法などについて説明します

アジェンダ

- デバイスのセキュリティの考え方
- 大量デバイスの管理
- まとめ

デバイスのセキュリティの考え方

AWS IoTとその他AWSサービスでの管理/セキュリティの違い

- IAM/Roleを使ってデバイスを管理？

  セキュリティクレデンシャルベースでの認証
  上限数の問題。PoCは良くても大量デバイスの管理で厳しい

- Cognitoを使う？

  Gateway/Edgeサーバ/sensorなど、画面の問題や定期的な認証など事実上不可能。(B2Cの製品については対応可能な事もある)

- AWS IoT Thingとして扱う(=証明書、Policy)

AWS IoT の制限

モノの制限

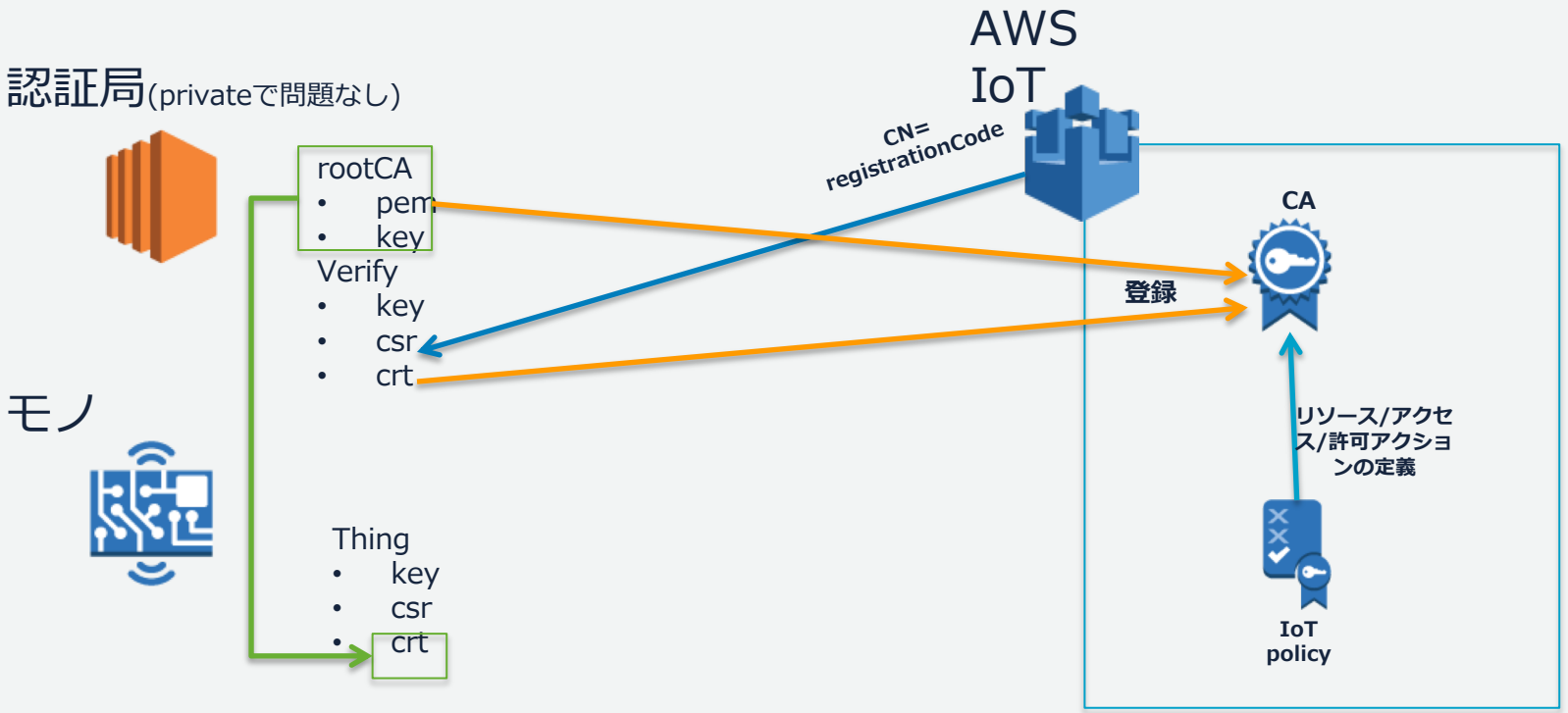
リソース	制限
モノ名のサイズ	UTF-8 エンコード文字で 128 バイト。この制限は Thing Registry および Thing Shadows サービスの両方に適用されます。
モノのタイプがあるモノのモノ属性の最大数	50
モノのタイプがないモノのモノ属性の最大数	3
モノを関連付けることのできるモノのタイプの数	1
AWS アカウントのモノのタイプの最大数	無制限

AWS IoTの証明書

- AWS IoT機能での証明書作成
- 自前認証局(持ち込みCA)からの証明書作成

https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/device-certs-your-own.html#create-device-cert

持ち込みCAセキュリティチェーン概念図



Just in time registration

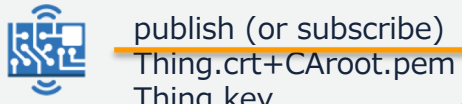
認証局で作成したデバイス証明書を元に、初回通信時に、AWS IoTが証明書を有効化/Policyの関連付けが出来る機能。

※証明書の有効化、Policyの関連付けが完了するまでは通信がrejectされる点に注意

Just in timeの証明発行処理：概要

CERT-ID単位でのtopicになるので、種別などでpolicyを分けたい場合はCAを複数使うことで対応可能

未認可な証明証のアクセスは暗黙的に
$aws/events/certificates/registered/<CERT-ID>
のtopicにイベント通知される



publish (or subscribe)
Thing.crt+CAroot.pem
Thing.key
Root.pem(Verisign)

IoT
rule

```
{
"certificateId": "",
"caCertificateId": "",
 "timestamp": "",
"certificateStatus": "PENDING_ACTIVATION",
 "awsAccountId": "",
"certificateRegistrationTimestamp": ""
}
```

証明書/AWS IoT policyのactivate処理

証明書のactivate/
policy処理が完了するまでは
errorとなることをに注意

https://aws.amazon.com/jp/blogs/news/new-just-in-time-certificate-registration-for-aws-iot/

Just in time registration のサンプルコード

```
var AWS = require('aws-sdk');

exports.handler = function(event, context, callback) {

    //Replace it with the AWS region the lambda will be running in
    var region = "us-east-1";

    var accountId = event.awsAccountId.toString().trim();

    var iot = new AWS.Iot({'region': region, apiVersion: '2015-05-28'});
    var certificateId = event.certificateId.toString().trim();

     //Replace it with your desired topic prefix
    var topicName = `foo/bar/${certificateId}`;

    var certificateARN = `arn:aws:iot:${region}:${accountId}:cert/${certificateId}`;
    var policyName = `Policy_${certificateId}`;

    //Policy that allows connect, publish, subscribe and receive
    var policy = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "iot:Connect"
                ],
                "Resource": `arn:aws:iot:${region}:${accountId}:client/${certificateId}`
            },
            {
                "Effect": "Allow",
                "Action": [
                    "iot:Publish",
                    "iot:Receive"
                ],
                "Resource": `arn:aws:iot:${region}:${accountId}:topic/${topicName}/*`
            },
            {
                "Effect": "Allow",
                "Action": [
                    "iot:Subscribe",
                ],
                "Resource": `arn:aws:iot:${region}:${accountId}:topicfilter/${topicName}/#`
            }
        ]
    };
```

Policy template

```
    /*
    Step 1) Create a policy
    */
    iot.createPolicy({
        policyDocument: JSON.stringify(policy),
        policyName: policyName
    }, (err, data) => {
        //Ignore if the policy already exists
        if (err && (!err.code || err.code !== 'ResourceAlreadyExistsException')) {
            console.log(err);
            callback(err, data);
            return;
        }
        console.log(data);

        /*
        Step 2) Attach the policy to the certificate
        */
        iot.attachPrincipalPolicy({
            policyName: policyName,
            principal: certificateARN
        }, (err, data) => {
            //Ignore if the policy is already attached
            if (err && (!err.code || err.code !== 'ResourceAlreadyExistsException')) {
                console.log(err);
                callback(err, data);
                return;
            }
            console.log(data);
            /*
            Step 3) Activate the certificate. Optionally, you can have your custom Certificate Revocation List (CRL) check
            logic here and ACTIVATE the certificate only if it is not in the CRL. Revoke the certificate if it is in the CRL
            */
            iot.updateCertificate({
                certificateId: certificateId,
                newStatus: 'ACTIVE'
            }, (err, data) => {
                if (err) {
                    console.log(err, err.stack);
                    callback(err, data);
                }
                else {
                    console.log(data);
                    callback(null, 'Success, created, attached policy and activated the certificate ' + certificateId);
                }
            });
        });
    });

}
```

Create Policy

attach Policy

証明書activate

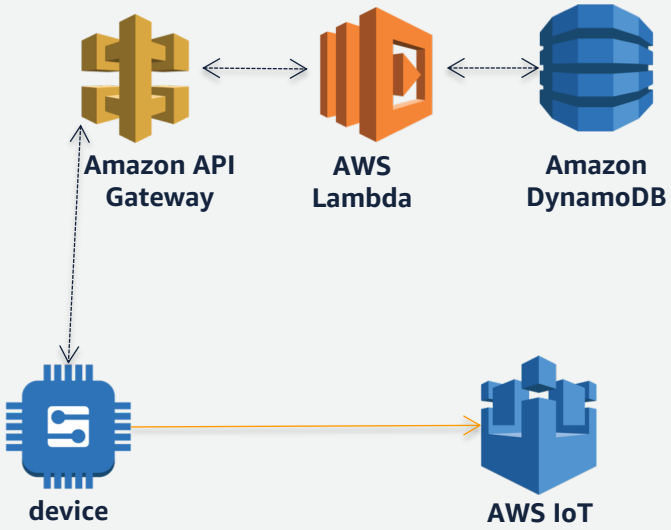
https://github.com/aws-samples/aws-iot-examples/blob/master/justInTimeRegistration/deviceActivation.js

aws

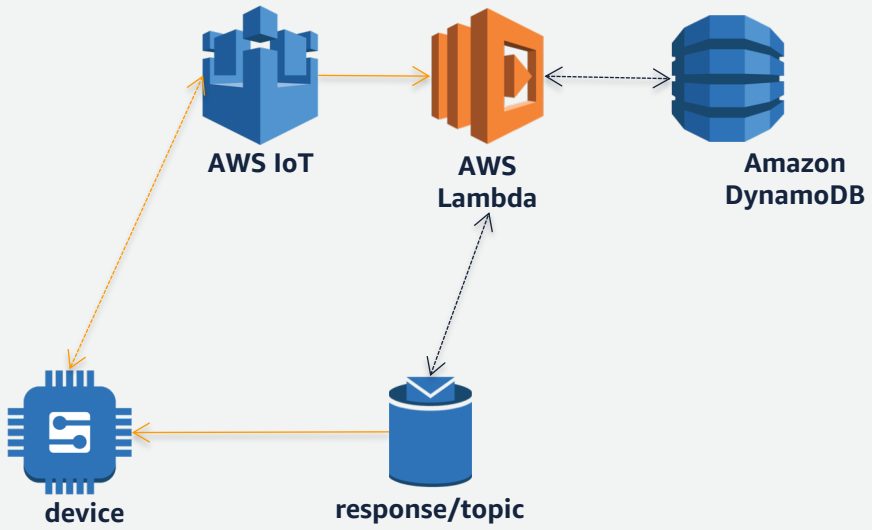
Bootstrapping:

個別証明書の発行が難しい場合のアーキテクチャ

HTTP / MQTTで初回起動時に個別証明書を発行



初回ユーザ登録など合わせてHTTP request で証明書を生成して返却する



プリインする証明書のpolicyは初期化用の mqtt topicのみが実行できるようにする

証明発行のパターンのまとめ

1. AWS IoTによる証明書発行
    - AWS IoTの権限、APIが発行できる環境が必要、
      事前に証明書、関連policy紐付けを完了させる
2. 持ち込みCAによる証明書発行
    - 発行の流れは1)と同じ、自分たちで証明書を生成出来るので証明書の期限がコントロール出来る点が1と比較したメリット
3. Just in time registration
    - デバイス証明書をoff line/AWSの操作権限なしで作成できる。
      事前にpolicy、証明書のアクティベートなどをしなくてもよく、
      自前認証局でdevice証明書を発行だけすれば良い
4. Bootstrapping
    - 生産現場とCloudでどうしても連携が難しい場合の手段

使い分けの例

1) AWS IoTの発行証明書、2) 自前認証局で証明書発行するケース

- 生産ラインですでにserial-IDを入れるなど機器毎の個別処理を持っている、secure ラインがあるケースで検討されることが多い
- 証明書の発行はクラウドの操作権限がある人

3) Just in time registrationを利用するケース

- 生産に関わる機器をinternetに繋げない、外部システムとの連携が難しい場合で、イントラ内のマシンに認証局を建てられるケースで検討される
- AWS IoTへCAの登録が完了していれば、工場内で証明書を生産可能

4) Bootstrappingを利用するケース

- クラウド側とデバイス生産で連携が難しく、クラウド側のみでクラウドのセキュリティの検討が必要なケースで検討されることが多い

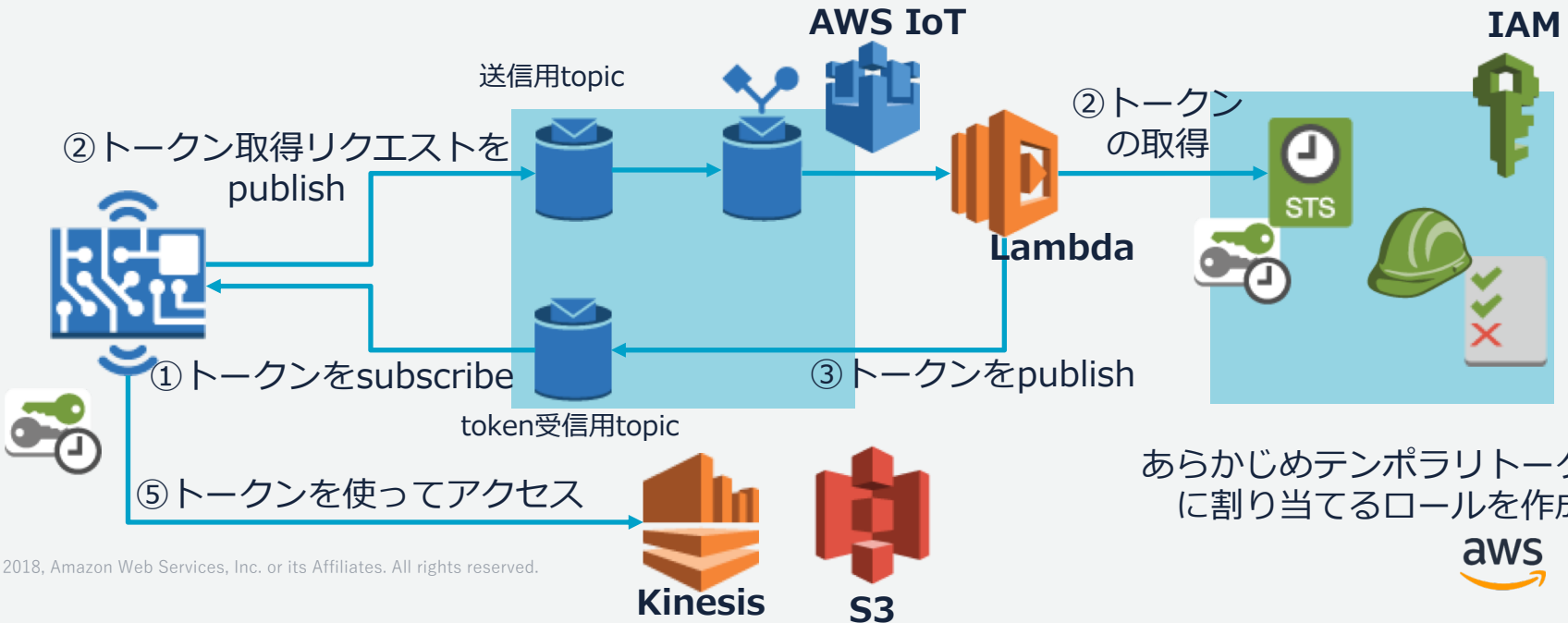
aws

証明書がより便利になり、
証明書を元に他のAWSサービスへの一時権限の取得が可能に

テンポラリトークン取得パターン

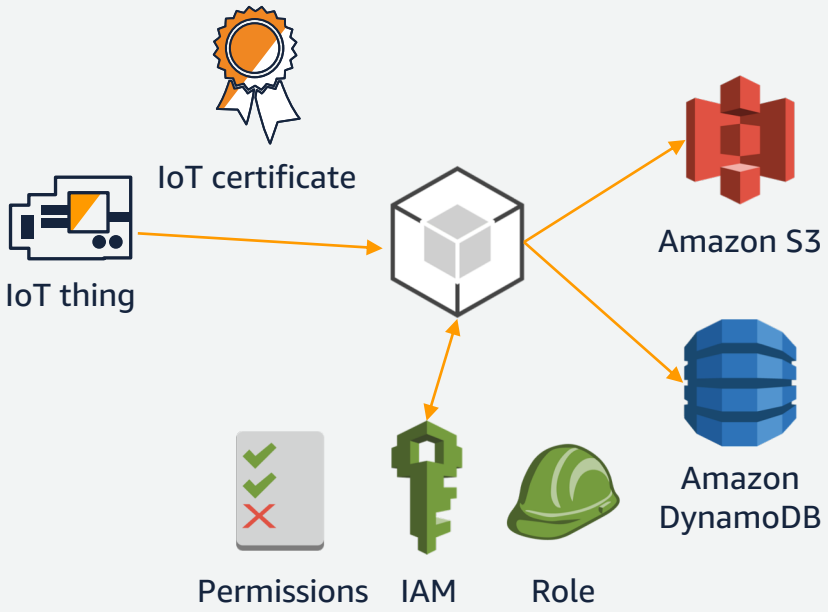
old

対象シーン	- AWS IoTの証明書を利用したセキュアな接続を利用してデバイスへAWSリソースに対してアクセス権を付与
条件	- 事前に利用するIAMが必要
注意点	- 特になし



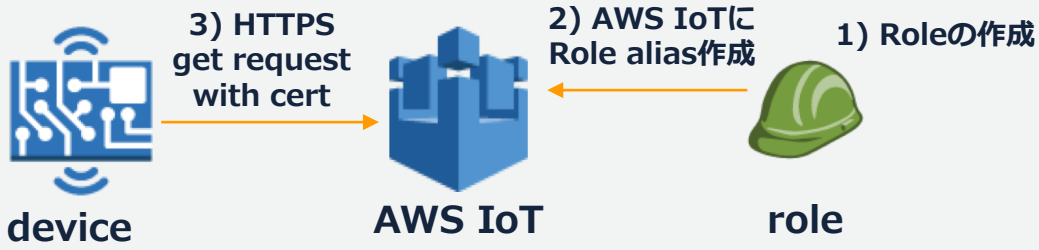
AWS SDK X.509 support

Control via IoT/MQTT
Data via HTTPS
e.g.,
Streaming video (CCTV)
Telemetry upload (Sensors)



x.509を利用したtoken取得

テンポラリトークン取得パターンと同等

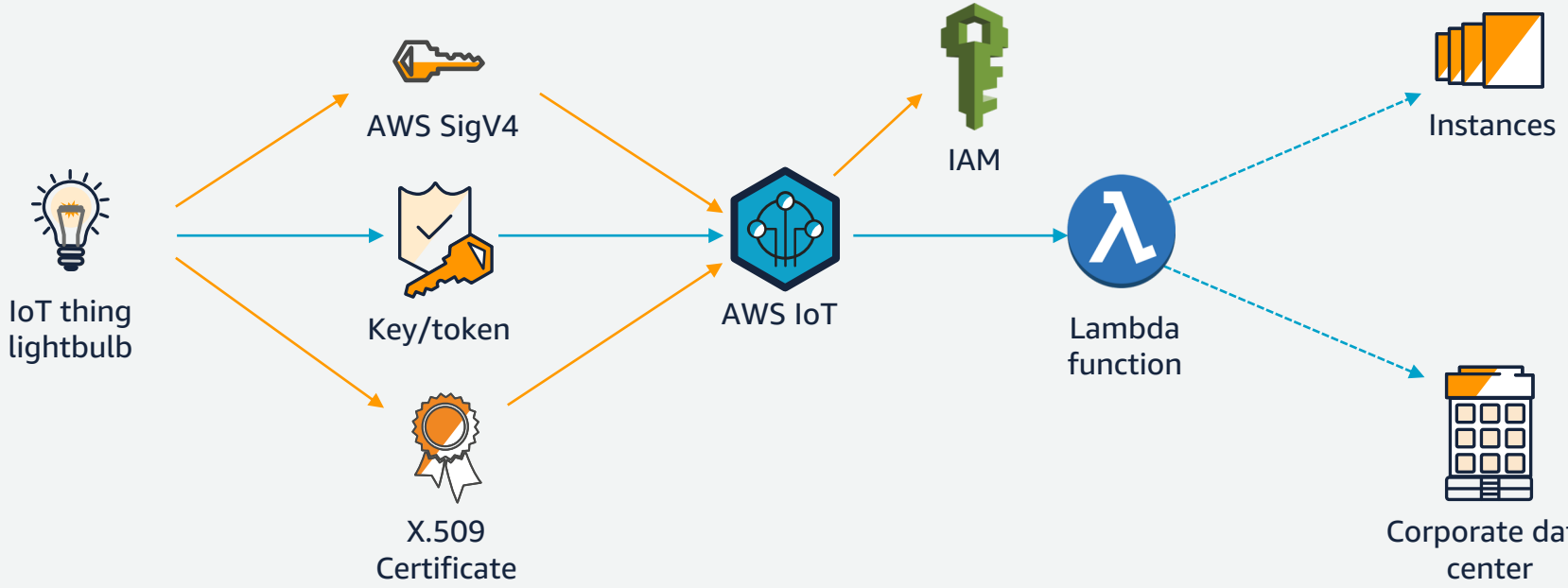


e.g) wgetでtempolayな認証情報の取得
wget
--private-key= private.pem.key
--certificate= certificate.pem.crt
--ca-directory= AmazonRootCA1.pem
https://<your_info>.credentials.iot.us-west-2.amazonaws.com:443/role-aliases/<2で登録したalias>/credentials

※)Credentialの有効期間は 2)のCreateRoleAlias APIで指定可能(900-3600秒)

https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/authorizing-direct-aws.html

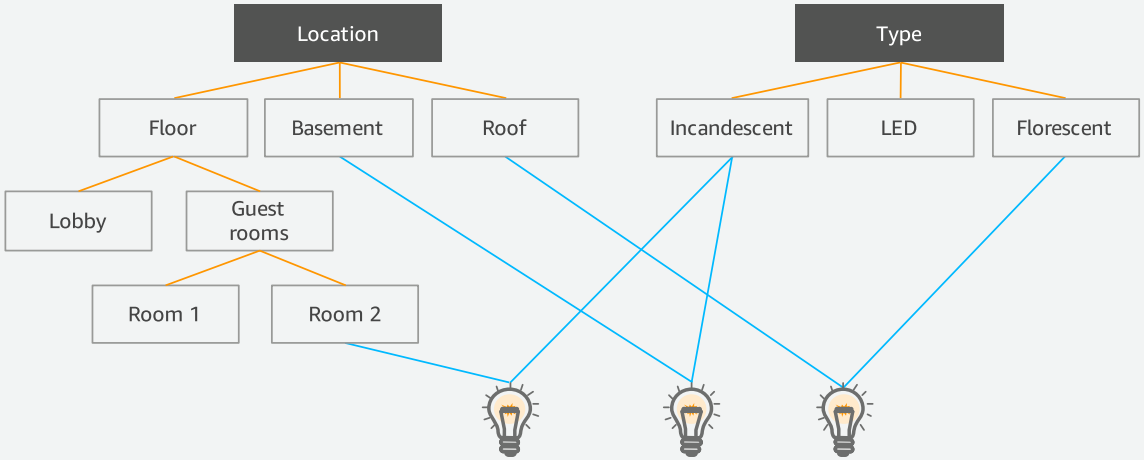
Custom authorizers



https://aws.amazon.com/jp/blogs/security/how-to-use-your-own-identity-and-access-management-systems-to-control-access-to-aws-iot-resources/

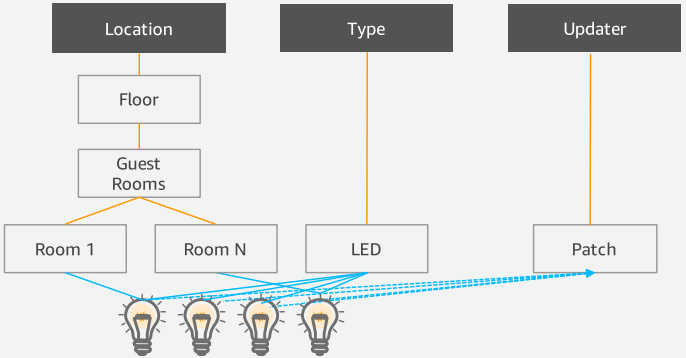
大量デバイスの管理

Thing Group



ThingをGroupとして、論理表現できる。また、Group単位でのPolicy設定が可能
- 個々のThing単位でPolicyを設定しなくても良い
例)
ビル => フロア => 部屋 などの階層設計など自由に設定可能

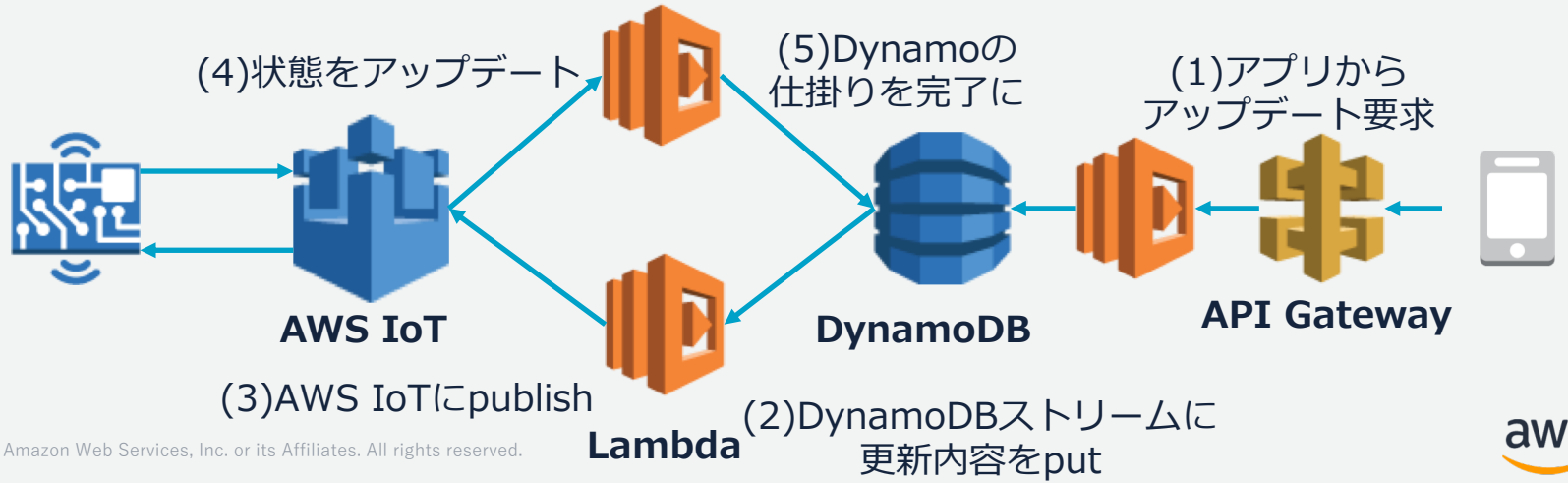
Groupの制約



- Group名の変更は出来ない
- Groupの階層設計(Hierarchy)の変更はできない
- 1Thing - 10Groupが上限
- １つの階層の中で1つのグループにのみ所属出来る
  - 左記の例だと、Location：roo1, roo2に所属できない
- 1Gruop の下に１００個の子グループ
- 階層は7階層まで
- グループにPolicyを設定できる。

今までご紹介していたデバイス管理デザインパターン

インテリジェントシャドウパターン(マルチshadowを利用している場合)	
対象シーン	- Web/スマホアプリなどからデバイスをコントロールしたい
条件	- マルチshadowを利用している
注意点	- シャドウを利用することが必須(シャドウは8Kbの制約)



本デザインパターンでの残課題

1. shadow updateによる通知

- 大量通知が必要でも１つ１つのデバイスのshadowをupdateする必要がある。
  - ３０万台：３０万回のDynamoDB write, Lambda実行、shadow update API
  - 例えば、スロットル回避でsleep(1)とすると、 30万秒 = 約3.5日くらいかかる
    - 効率的な配信のためのAPI発行プログラムの検討など

2. AWS IoTではなく、DynamoDBの状態が絶対

- Shadowの状態を正として動くほうが論理不整合起きにくい

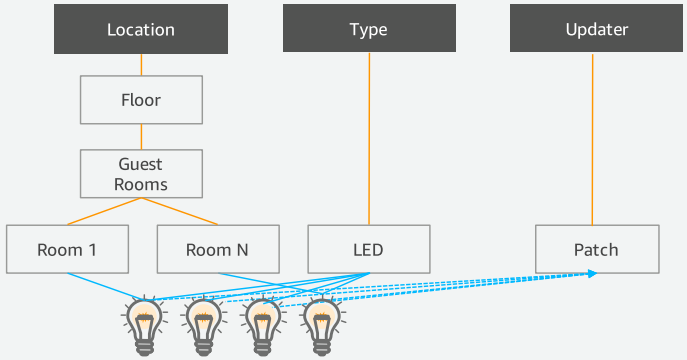
AWS IoT Device Management Jobs: アップデート機能

異なるスケジュールモデル

Continuous

Snapshot

柔軟な配信



ステータス確認

Status

AWS IoTで完結可能

(1)indexを使って
対象を検索
(shadow.reported.firmware <= 1.0.0)

(2)検索結果から
update用のGroup作成

(3) (2)で作成した
Group向けにJob生成

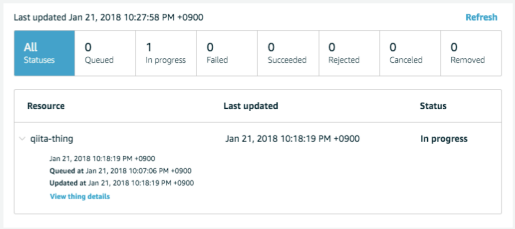
(4)通知

(5)status通知

(5)’ 統計用API

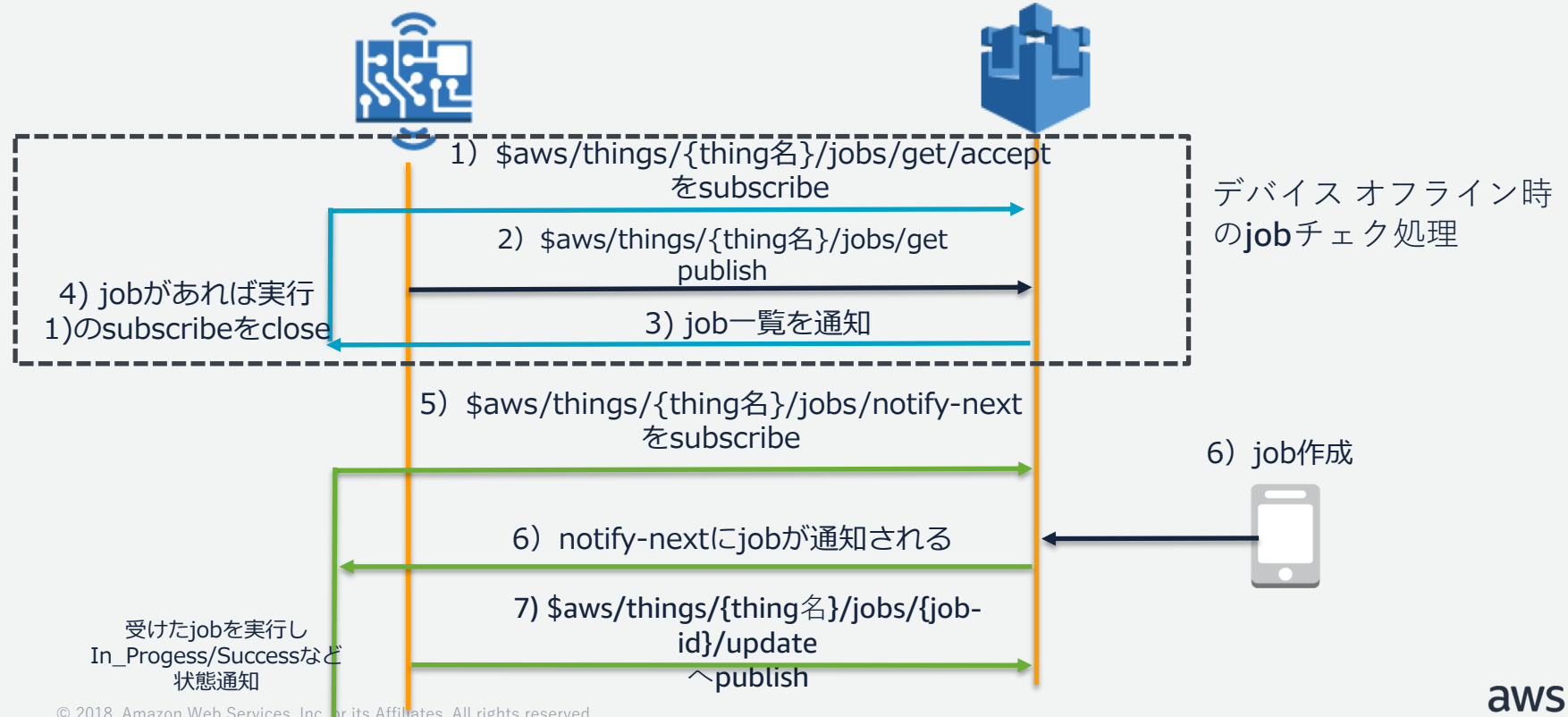
(6)Shadow update

(7)完了通知



Jobを受けるDeviceのtask

(SDKがリリースされれば抽象化される：現在device-sdk-js/embedded Cのみ対応)



起動している間 5)のsubscriberを張りっぱなし

Jobs: アップデート機能

Jobモデル

Continuous　　Snapshot

- **Snapshot**:
  Jobを作成したタイミングに存在するthingが対象で一度作るとJob通知の対象は変更されない

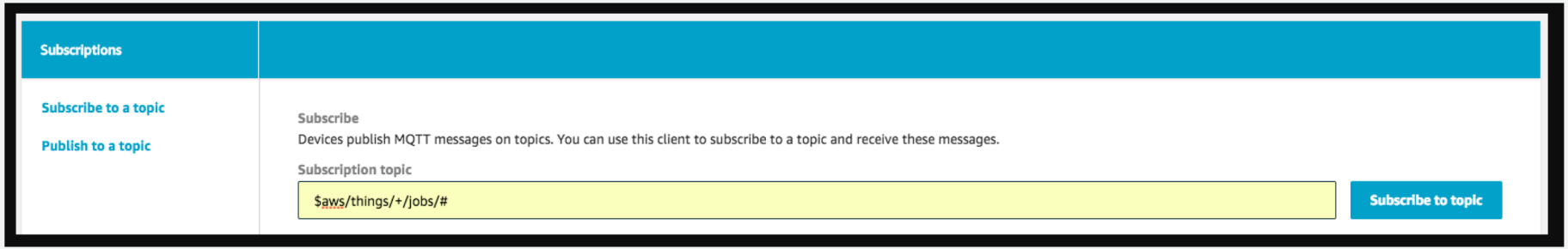
- **Continuous**:
  Jobを作成した後も**配信対象となるGroupにthingが追加される**と通知が行われる。

作成されたJobは90日間有効。90日経過するとlistからも除外されるため、恒久的にJob履歴を残す場合は別途APIで結果を取得し、保管を検討

job開始時刻指定の機能はないので、Job作成をscriptなどで準備しjob作成を時刻指定する必要がある

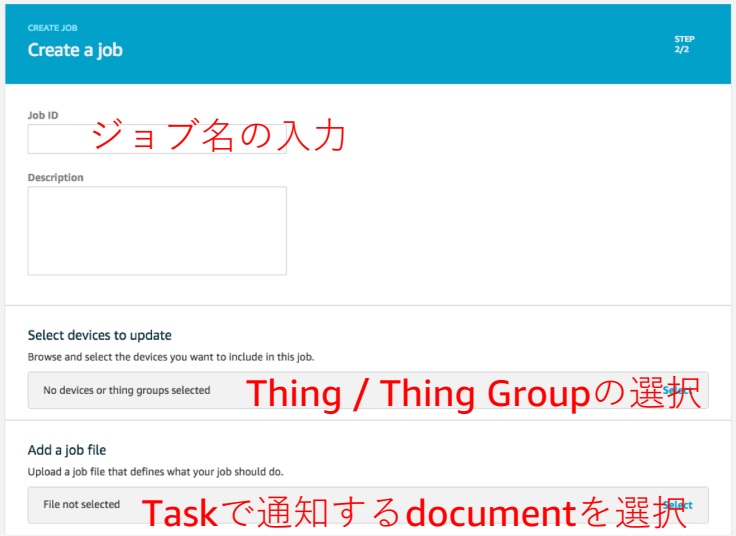
Jobの簡単なテスト方法(1/3)

Jobの通知の確認方法
AWSのマネージメントコンソールから、AWS IoTを選択し左側メニューの"Test" を選択
Jobで利用されるtopicが $aws/things/{thingname}/jobs/{job-id}であるので、ワイルドカードを利用し、画面のSubscription topicへ“$aws/things/+/jobs/#”を入力し、subscribe to topicを押下して、受信状態(この画面を開いた状態)のままにしておく。

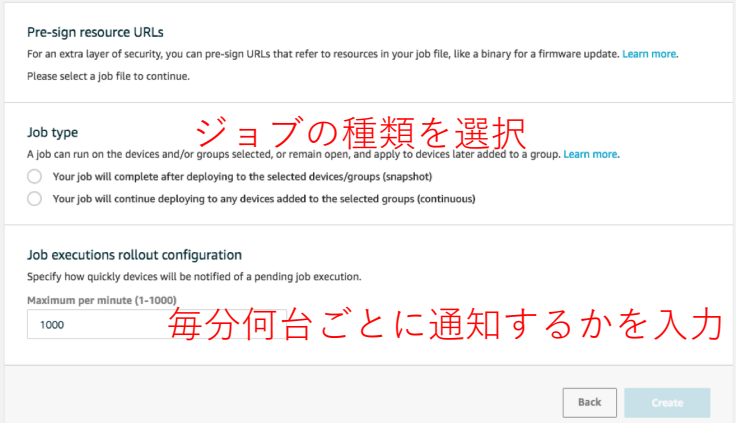


Jobの簡単なテスト方法(2/3)

Jobの作成
1)と別のブラウザタブからAWS IoTコンソール=＞管理=>ジョブ=>作成
=>カスタムジョブの作成

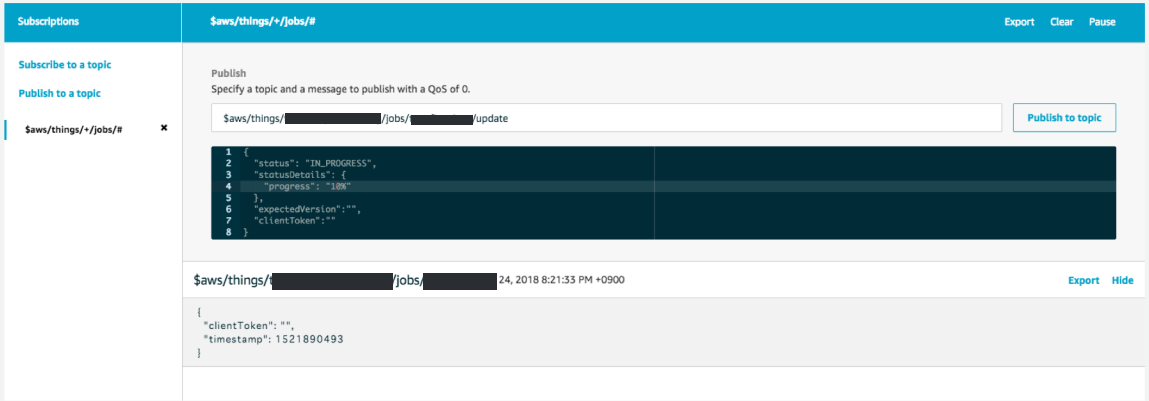


事前に、Jobで通知する内容をJSONファイルを作成しS3へ配置する必要がある



Jobの簡単なテスト方法(3/3)

1)のsubscribe画面にあるpublishから先程、ご自身で作ったjobにあわせてpubhlish topicへ以下を設定する。
topicに $aws/things/YOUR_THING/jobs/JOB_NAME/update
Message入力のJsonフィールドへ以下をいれて、”publish to topic”ボタンを押下、
{ "status": "IN_PROGRESS", "statusDetails": { "progress": "0%" }, "expectedVersion":"", "clientToken":"" }
Jobを完了状態にしたい場合は、IN_PROGRESSをSUCCEEDEDにしてpublishする



statusDetailsは
device側で自由に
記入できる領域

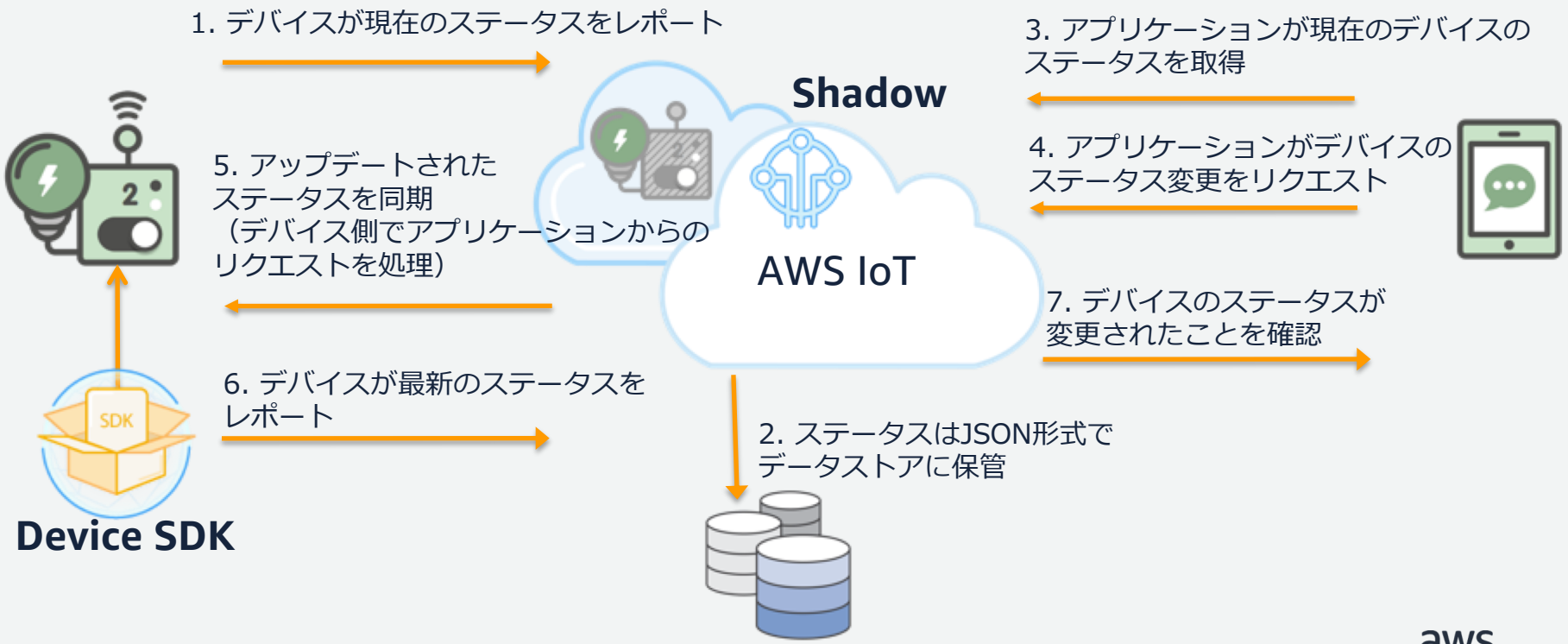
ShadowとJobの使い分け？

AWS IoT Shadow

結果整合性として、強い管理機能がある。
Shadowを一度使うと、機器の制御のコントロールプレインとしてShadowを中心に考える必要がある

以降でshadowのフローを説明

デバイス シャドウの動作フロー



デバイス シャドウ

Thing

1つまたは複数の現状ステータスをシャドウに通知
シャドウから要求されるステータスを取得

Shadow

シャドウは、delta, desired 及びreported
ステータスをメタデータとバージョンをつけて管理

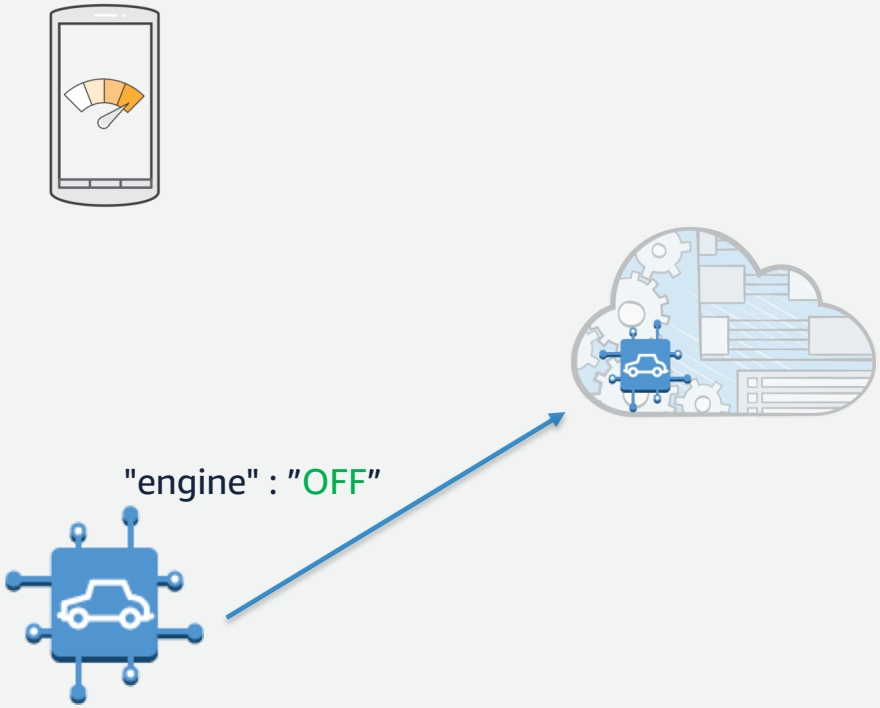
Mobile App

デバイスに対して変更したいステータスをセット
最新の通知されたステータスを取得
シャドウの削除

```
{
    "state" : {
       “desired" : {
           "engine" : "ON"
       },
       "reported" : {
           "engine" : ”OFF"
       },
       "delta" : {
           "engine" : ”ON"
        }
    },
    "version" : 10
}
```


aws

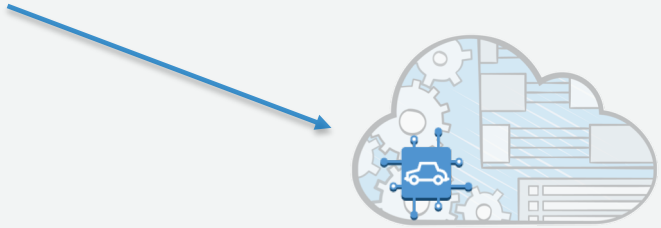
デバイスは現在のステータスをアップデート




```
{
    "state" : {
       “desired" : {
           "engine" : ”ON"
       },
       "reported" : {
           "engine" : ”OFF"
       }
    },
    "version" : 10
}
```

アプリケーションからエンジンON

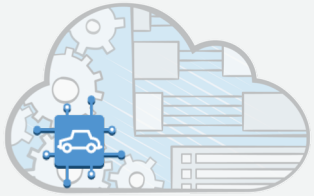
"engine" : ”ON”



```
{
    "state" : {
       “desired" : {
           "engine" : ”ON"
       },
       "reported" : {
           "engine" : ”OFF"
       }
    },
    "version" : 10
}
```


aws

deltaが通知される

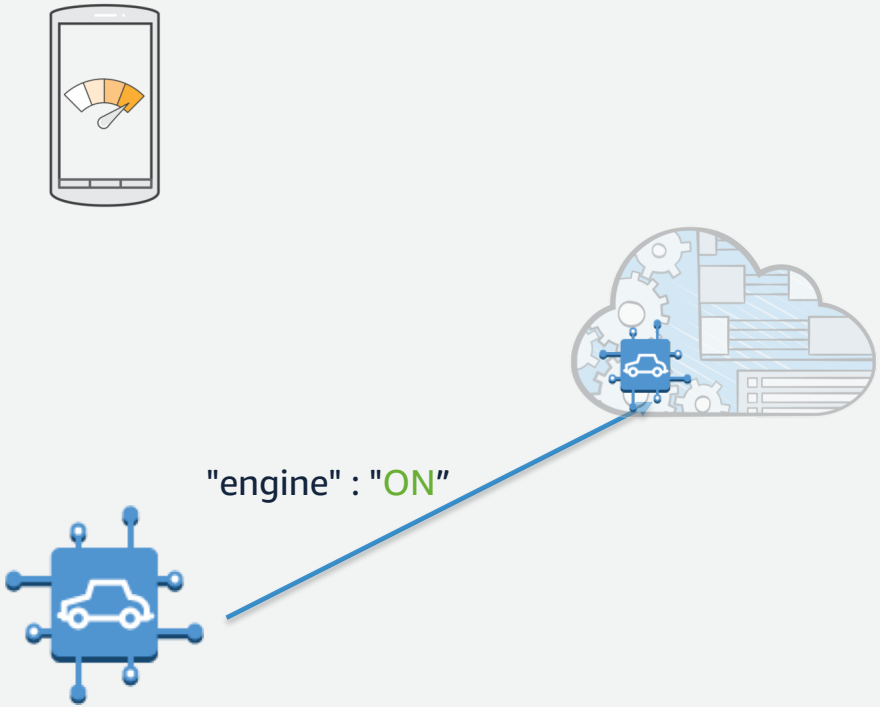


"engine" : "ON”

```
{
    "state" : {
       “desired" : {
           "engine" : "ON"
       },
       "reported" : {
           "engine" : ”OFF"
       },
       "delta" : {
           "engine" : ”ON"
        }
    },
    "version" : 10
}
```

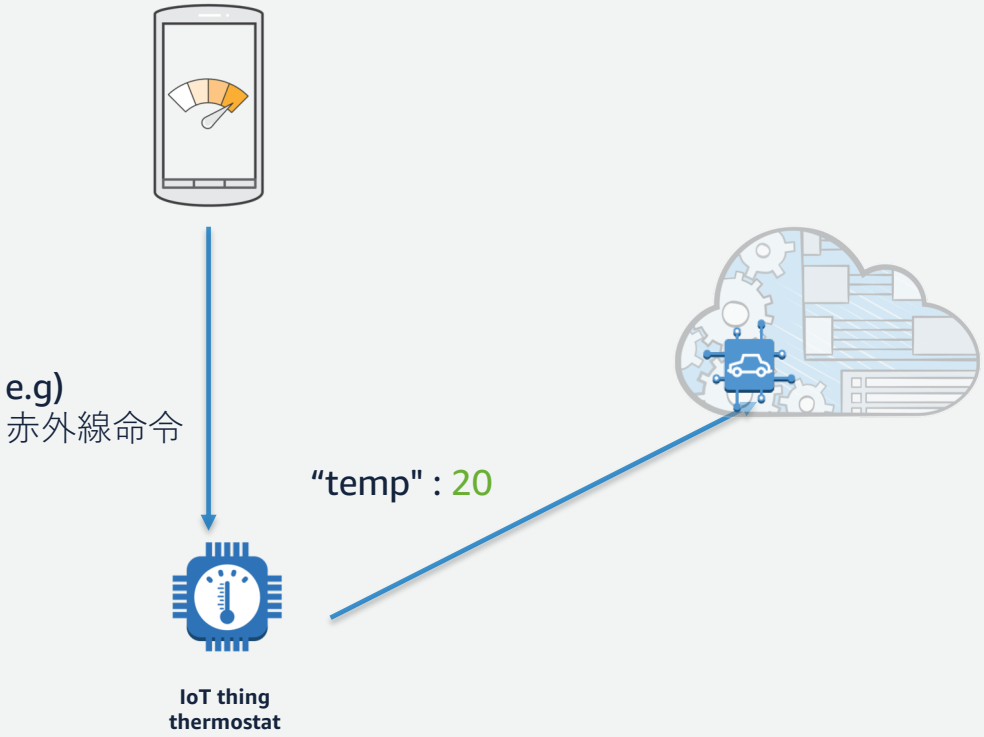

aws

最新のステータスにアップデート(deltaは消える)



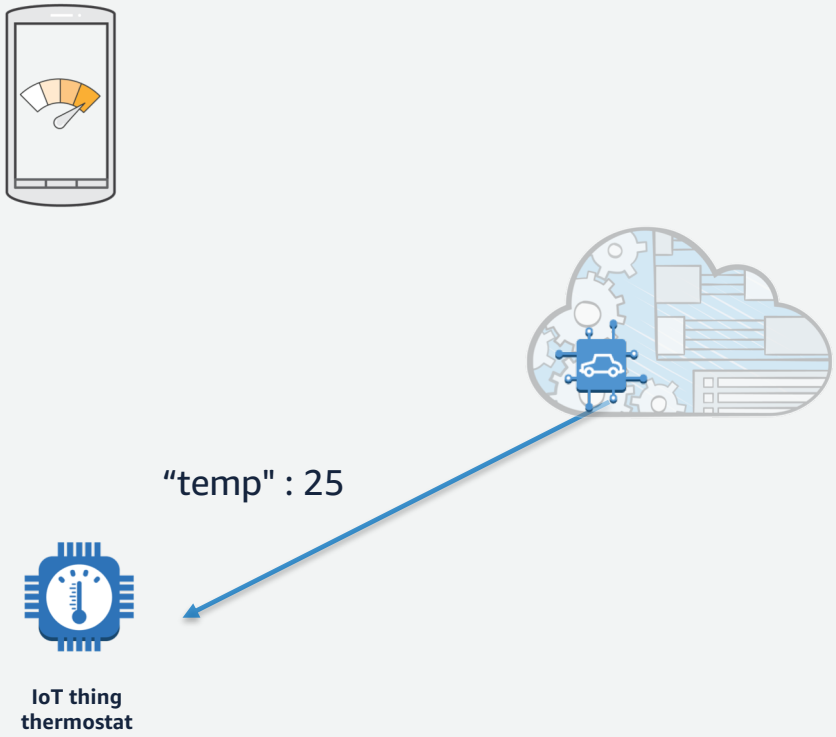

```
{
    "state" : {
       “desired" : {
           "engine" : "ON"
       },
       "reported" : {
           "engine" : ”ON"
       },
       "delta" : {
           "engine" : ”ON"
       }
    },
    "version" : 10
}
```

User操作によりreportだけ変更すると？




```
{
    "state" : {
       “desired" : {
          “temp" : 25
       },
       "reported" : {
          “temp" : 20
       },
    },
    "version" : 10
}
```

User操作によりreportだけ変更すると？




```
{
    "state" : {
       “desired" : {
          “temp" : 25
       },
       "reported" : {
          “temp" : 20
       },
      “delta“: {
            “temp“ : 25
    },
    "version" : 10
}
```

Shadowの使い所

管理者のみがコントロール出来る値/configの一部を管理するようなユースケースや特定機器に対しての設定変更が行いやすい。

例) debug level 、 データ送信頻度など

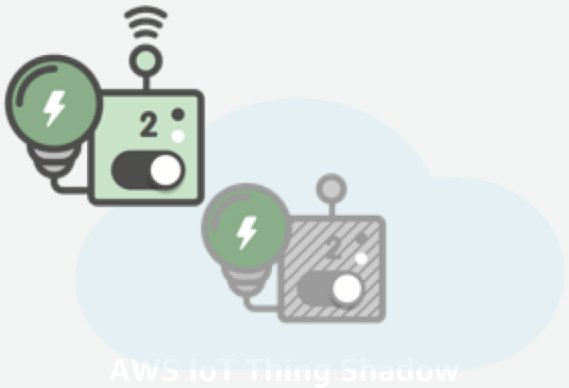
Shadow のdesiredを同値でupdateすると、deltaは発生しないが、$aws/things/*thingName*/shadow/update/documentsにpublishは発生する。ユースケースに応じてshadow関連のtopicは理解しておくこと

シャドウのMQTT Topic

デバイスSDK (C-SDK, JS-SDK)で
シャドウのMQTTトピックを利用

Sensor	Reported	Desired	Delta
LED1	RED	YELLOW	LED1 = Yellow TEMP = 60F
ACCEL	X=1,Y=5,Z=4	X=1,Y=5,Z=4	
TEMP	83F	60F	

UPDATE: $aws/things/{thingName}/shadow/update
DELTA: $aws/things/{thingName}/shadow/update/delta
GET: $aws/things/{thingName}/shadow/get
DELETE: $aws/things/{thingName}/shadow/delete



https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/device-shadow-mqtt.html

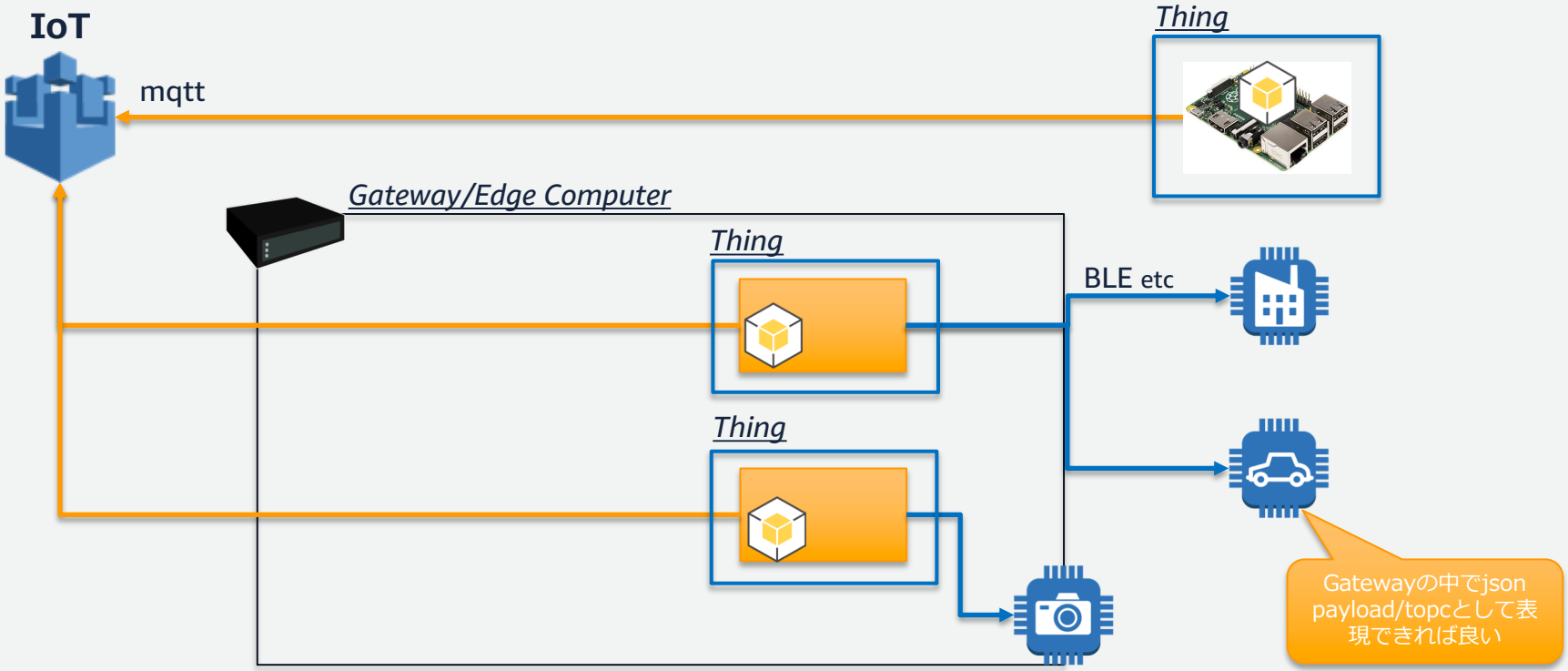
Jobの使い所

Stateをもたない通知や実行命令などを大量のデバイスに行いたい場合に利用をしやすい。(統計機能もある)

firmware version,コンテンツ配信などは前述の通り、shadowと組み合わせる事で, Index機能でshadow検索 =>update Group作成が可能となり、組み合わせで便利に使える機能となる

まとめ

Thing? AWS IoTの場合
AWS IoTと通信するものが証明書を持てば良い



大量デバイスに関わる制約事項の理解も重要

AWS IoT limitより引用:

- アカウントを 1 秒あたり最大 300 MQTT CONNECT リクエストに制限します。
- アカウント別に 1 秒あたり 9,000 publish
  インバウンド発行リクエスト - 最大 1 秒あたり 3,000、
  アウトバウンド発行リクエスト数 - 最大 1 秒あたり 6000
- 1 秒あたり最大 500 のサブスクリプションにアカウントが制限されます。

上限融和可能だが、デバイス単位でタイミングがばらつくような設計を推奨

https://docs.aws.amazon.com/ja_jp/general/latest/gr/aws_service_limits.html#limits_iot

まとめ

証明書など個別機器に対してのセキュリティなどを後回しにしがちだが、**プロジェクト最初からデバイスの管理**(セキュリティやfirm管理など)について決めておくことを推奨

出荷済みの機器や対象のthingに対してクラウドからコントロールできる仕組みを予め検討し、手段を用意しておくことも重要となる

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

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

AWS Solutions Architect 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
- http://aws.typepad.com/sajp/

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、お見積り、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ください
https://aws.amazon.com/jp/contact-us/aws-sales/

※「AWS　問い合わせ」で検索してください

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-2017Q4-jp.html
- 参加無料
- 毎週火曜・木曜開催

aws