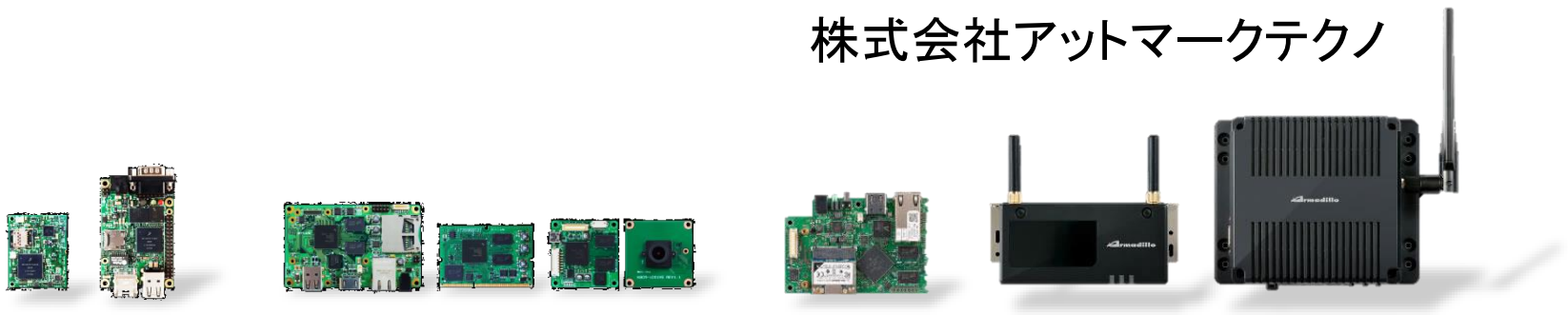
Armadillo-IoT G3/G3L
Armadillo-X1
開発体験セミナー

第6部 クラウドとの連携

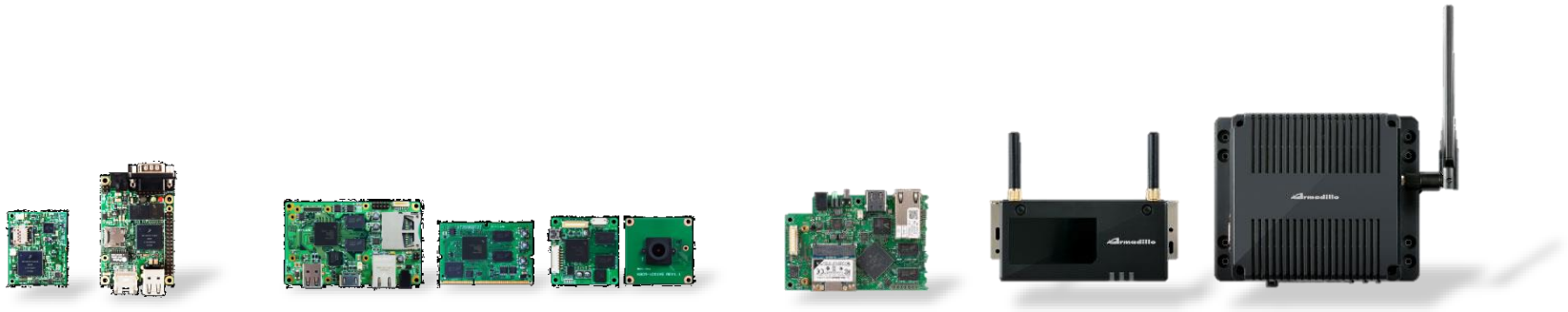
株式会社アットマークテクノ



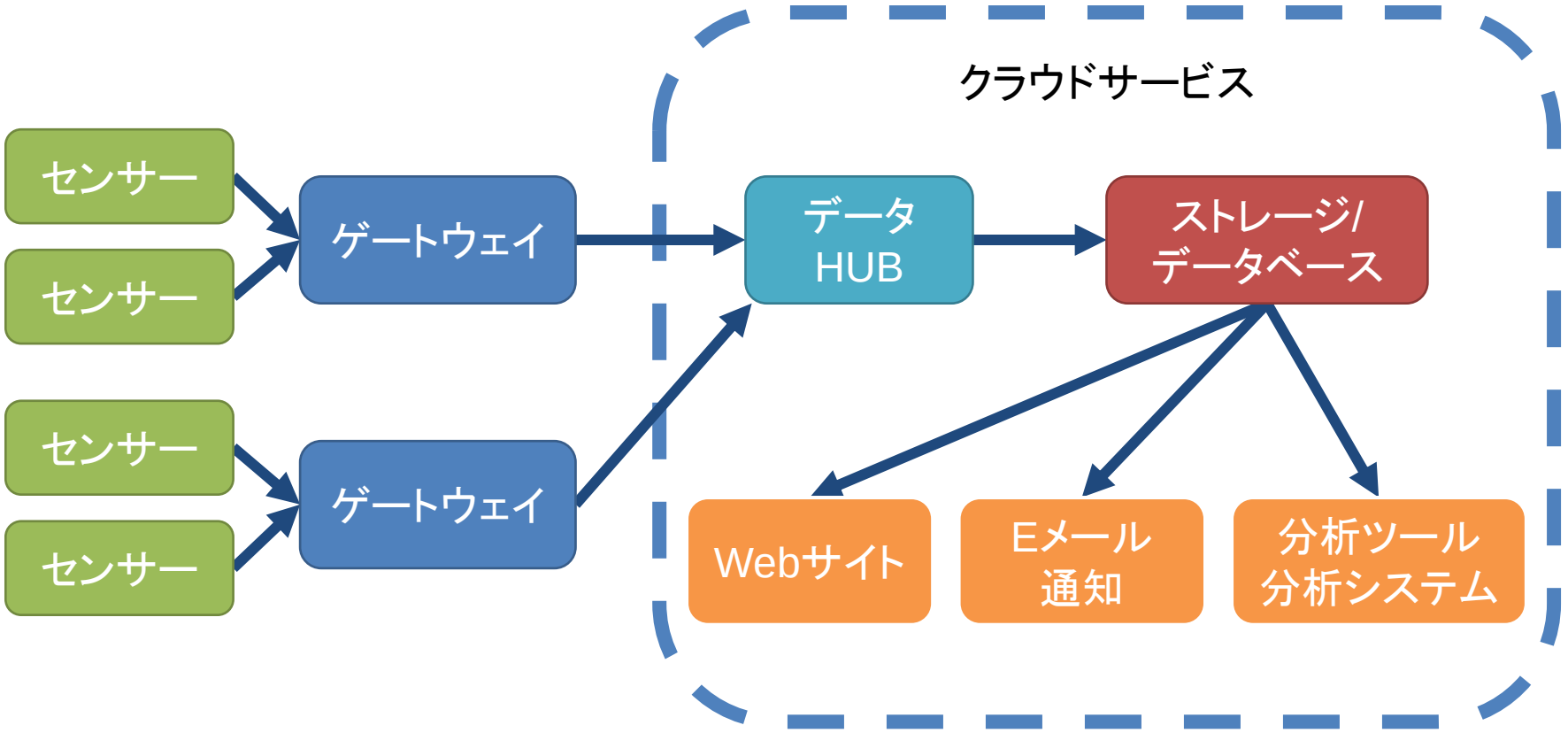
目次

Atmark Techno

クラウドサービス紹介



IoTシステム構成例

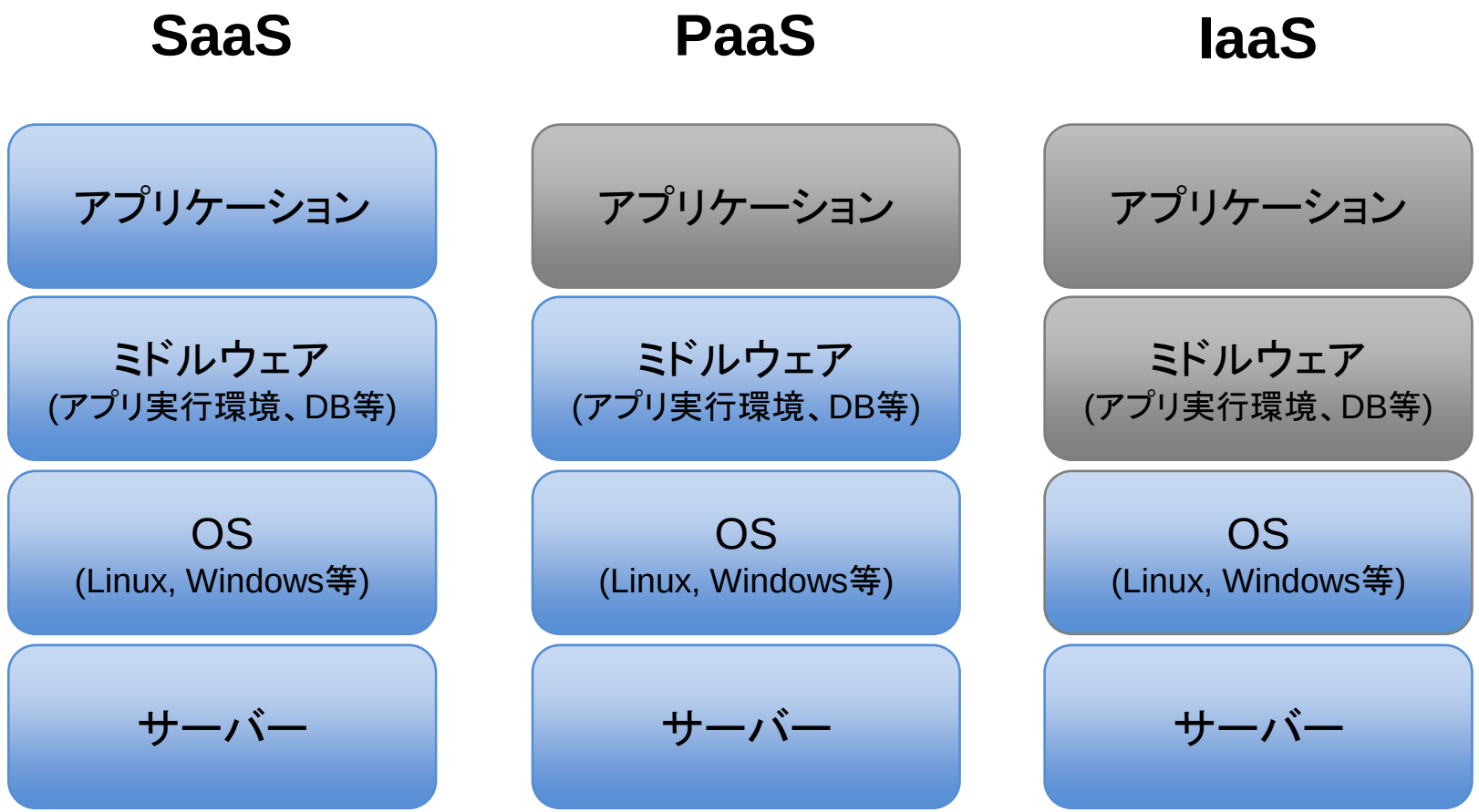

クラウドサービス
センサー
センサー
センサー
センサー
ゲートウェイ
ゲートウェイ
データ
HUB
ストレージ/
データベース
Webサイト
Eメール
通知
分析ツール
分析システム

IoT向けクラウドサービスの提供形態

Atmark Techno

- SaaS(Software as a Service)
  - ソフトウェアの使用をサービスとして提供
  - 一般的にはGmail、Googleマップ、Evernote、Instagram等
- PaaS(Platform as a Service)
  - ソフトウェアの実行環境や、DB実行環境等のプラットフォームをサービスとして提供
- IaaS(Infrastructure as a Service)
  - コンピュータシステムを構築および稼動させるための基盤をサービスとして提供

IoT向けクラウドサービスの提供形態

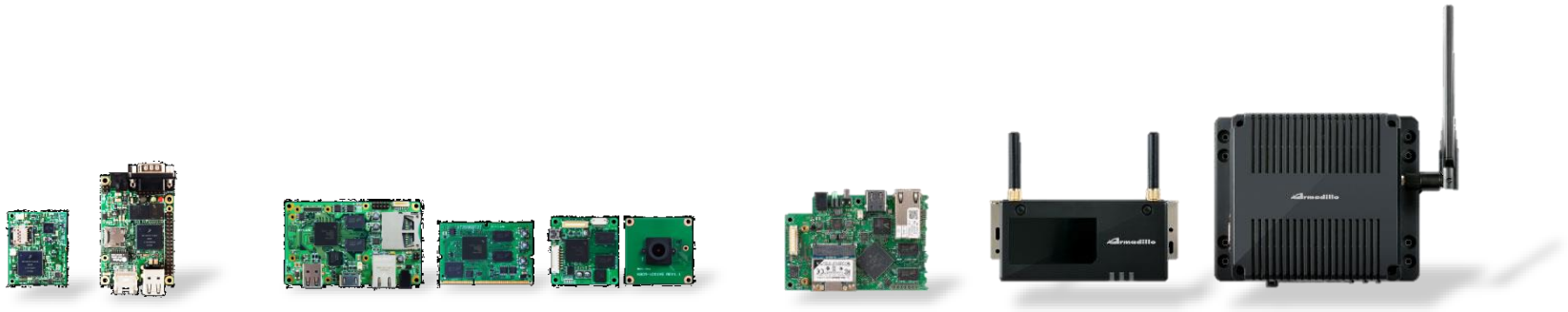


IoT向けクラウドサービス紹介

Atmark Techno

- SaaS
  - Axeda
  - SensorCurpus
  - CUMoNoSu(Cumulocityベース)
- PaaS
  - Microsoft Azure
  - AWS各種サービス
  - IBM Bluemix
- IaaS
  - AWS EC2
  - Google Compute Engine

クラウドとの接続



クラウドとの通信プロトコル

Atmark Techno

- HTTP(REST API)
  - 最も有名で簡易的な方法
  - Webの標準的なプロトコルのためツール類も充実していて使いやすい
- WebSocket
  - インターネット上でソケットを実現するためのプロトコル
  - クライアントからの要求無しに、サーバからのデータ送信が可能
- MQTT
  - 軽量でIoTに適したプロトコル
  - ブローカーというサーバーを使用して1対多の通信が可能
  - QoS(Quality of Service)等の各種機能を持っている

クラウドとの通信データフォーマット

Atmark Techno

- XML形式

```
<xml>
  <No>1</No><labelA>value1A</labelA><labelB>value1B</labelB>
  <No>2</No><labelA>value2A</labelA><labelB>value2B</labelB>
</xml>
```

- JSON形式

```
[
  {No:1, labelA:value1A, labelB:value1B},
  {No:2, labelA:value2A, labelB:value2B},
]
```

ゲートウェイで必要になるソフトウェア

Atmark Techno

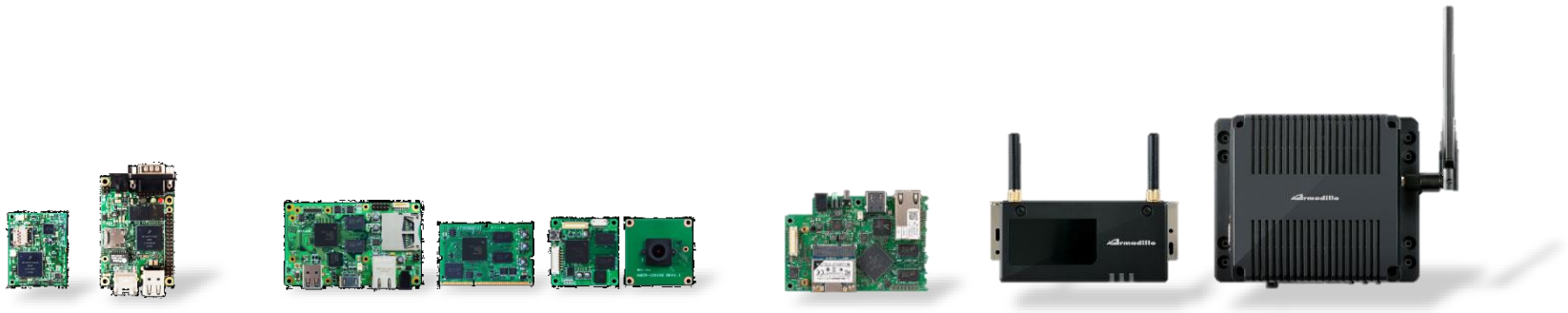
- HTTP/WebSocket/MQTTを使用して通信するソフトウェアを作成
  - クラウド側のAPIに合わせてデータ通信
  - 自前で作成できるため、柔軟性がある
  - 半面、開発コストが増加する傾向
- クラウドサービス側からエージェントが提供されることが多い
  - クラウドと通信するためのソフトウェアのテンプレートのようなもので、環境に合わせてカスタマイズして使用
  - node.js、Python、C、JAVA等の言語で提供
  - 例. AWS IoT, Azure IoT等のPaaSや、SaaSサービス各種

参考: クラウドが使用されることが多い理由

Atmark Techno

- 自前でサーバーを立てるよりも初期費が安価
  - 自前でサーバーを用意するとハードウェアから用意する必要があり初期費がかかる
  - サーバーの保守体制等で費用がかかる
- 柔軟にパフォーマンスを変更できる
  - 登録後すぐに使用できる場合が多い
  - 使用量が多い場合だけ、リソース増強が可能
- すでに構築されたサービスを利用可能
  - ストレージ、DB、分析基盤などが用意されている

サンプルアプリケーションによるテスト



サンプルアプリケーション概要

- クラウドサービス(今回はAWS)との接続確認を行うために、サンプルアプリケーションを使用してテストを行います。

- ArmadilloをAWS IoT coreのモノに登録し、モノの状態を格納するshadowステータスを、整数の値を変化させて1秒毎に更新します。

構築済みのArmadilloの環境

以下環境が整っていない場合は、各設定を行ってください。

- ネットワークへの接続

  →製品マニュアルをご参照ください。

- パッケージの更新

  →「apt-get update && apt-get upgrade」コマンド実行

- python3、pip、bluepy、libglib2.0-devのインストール

  →セミナー資料の5章をご確認ください。

Armadilloの環境の構築

Atmark Techno

- 以下のコマンドを実行し、AWS IoTで動作確認を行うために必要なパッケージのインストール、サンプルプログラムのダウンロードを行います。

```
[armadillo ~]# pip3 install AWSIoTPythonSDK
[armadillo ~]# apt-get install git
[armadillo ~]# git clone https://github.com/aws/aws-iot-device-sdk-python.git
```

- また、カレントディレクトリにAWSの証明書を置くディレクトリも作成しておきましょう。

```
[armadillo ~]# mkdir cert
```

Thing(モノ)の登録

ArmadilloをAWS IoTのモノへ登録してください。

AWS IoTのThing(モノ)の登録方法は、以下の公式開発者ガイドを参考にさせていただきました。

- 開発者ガイド:
  https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/what-is-aws-iot.htmlAmazon

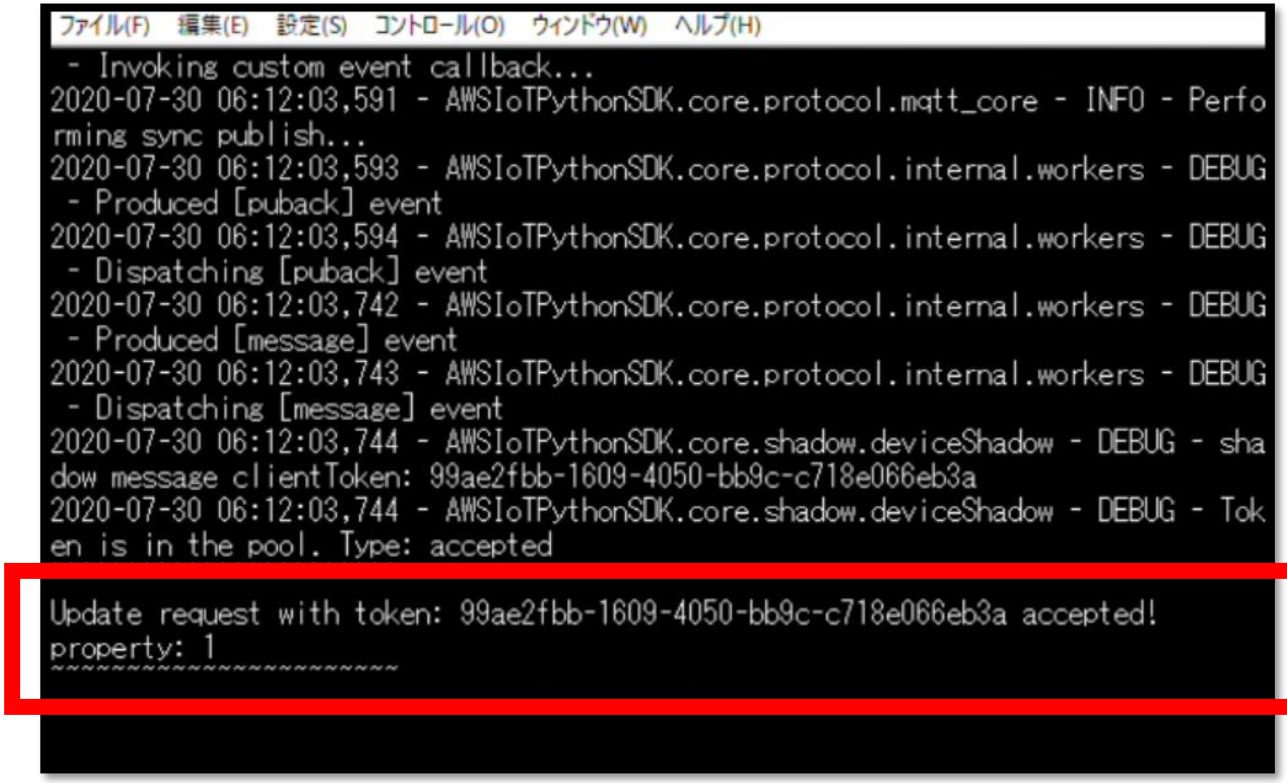
AWS IoT Core Shadowの更新によるテスト

- 以下のコマンドを実行し、AWS IoT coreのモノの状態を格納する「shadow」を更新するサンプルプログラムを実行します。
- []内は、各環境によって変更してください。

```
[armadillo ~]# cd aws-iot-device-sdk-python/samples/basicShadow/
[armadillo ~/aws-iot-device-sdk-python/samples/basicShadow/]#
python3 basicShadowUpdater.py ¥
> ----endpoint [endpoint] ¥
> --rootCA ./cert/AmazonRootCA1.pem ¥
> --cert ./cert/*-certificate.pem.crt ¥
> --key ./cert/*-private.pem.key ¥
> --thingName [thingName]
```

コンソール画面の結果

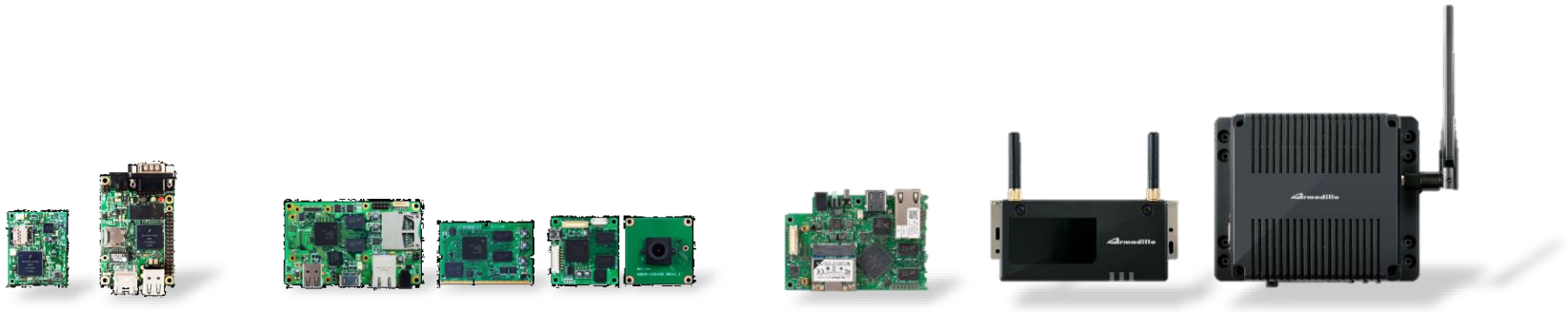
- 実行すると、以下のように整数をカウントアップしながら1秒毎にAWSに送信しているのがわかります。



shadowの確認

- AWS IoT coreのモノのシャドウを確認すると、シャドウステータスが更新され、Armadilloから送信しているデータを受信できていることが確認できます。

サンプルアプリケーション作成



サンプルアプリケーション概要

Atmark Techno

- クラウドサービスとの接続時の開発をイメージしやすくするためにサンプルアプリケーションを作成します

- オムロン環境センサーのデータをArmadilloで取得し、クラウド(AWS)に送信したデータを、AWSのkibanaのダッシュボードで表示するアプリケーションを作成します

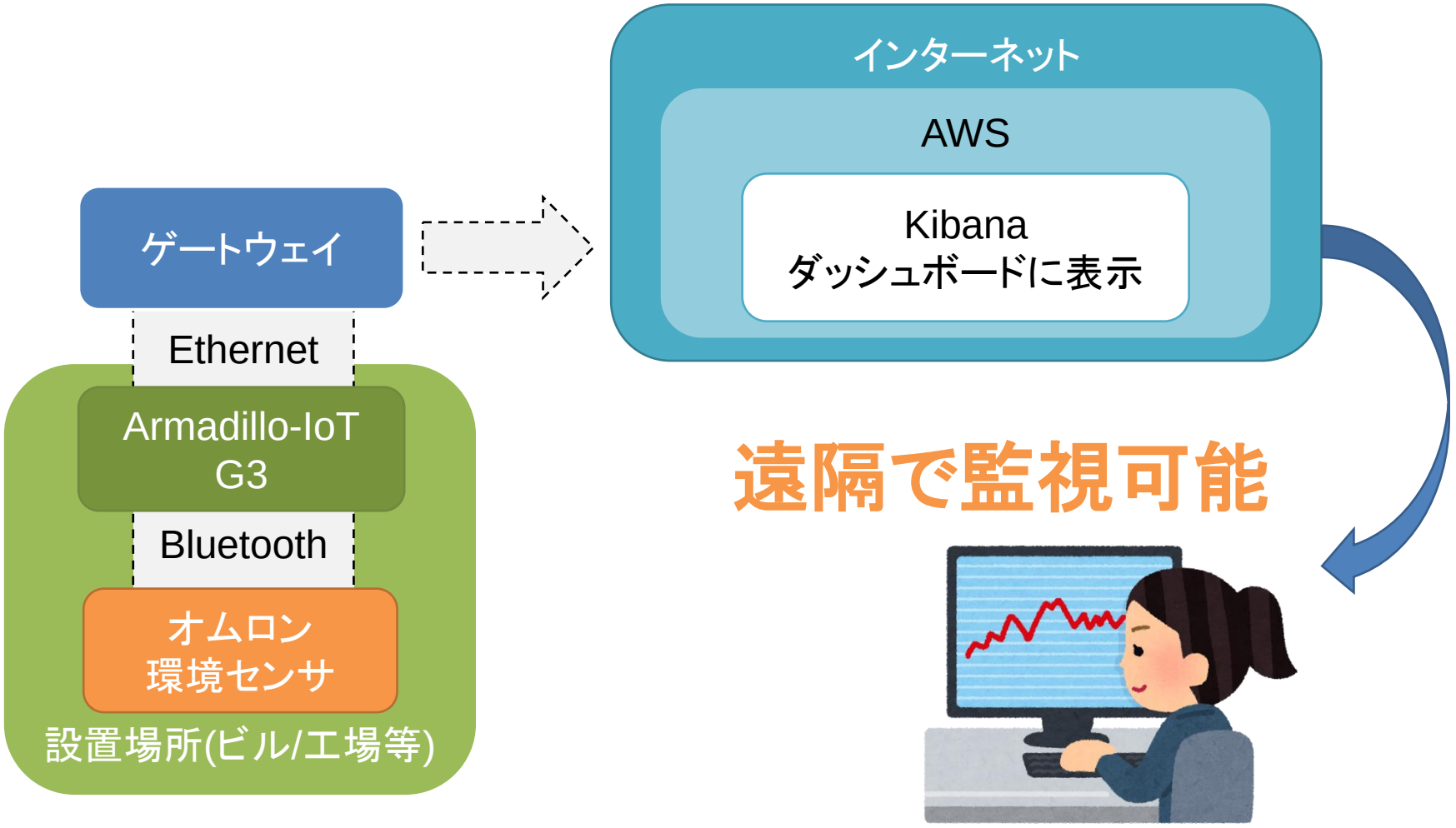
サンプルアプリケーション概要

Atmark Techno

- 以下の機能を持つアプリケーションを作成します
  - オムロン環境センサのデータ値を取得
  - Armadilloの内部で温度と照度データをJSON形式に変換
  - Armadilloから、AWSに温度と照度のデータを送信
- サンプルアプリケーションは、簡単にサンプルが作成できるpythonを使用します

- データを受信するクラウドアプリケーションについては、AWSを使用します
- AWSの環境の構築に関しましては、各参考ページの情報をもとに設定していただけますと幸いです

システム構成


インターネット
AWS
Kibana
ダッシュボードに表示
ゲートウェイ
Ethernet
Armadillo-IoT
G3
Bluetooth
オムロン
環境センサ
設置場所(ビル/工場等)
遠隔で監視可能

サンプルのダウンロード

- 以下のURLから、以下の2つのサンプルプログラムをダウンロードしてください。
- XXXXXXXXXXXXXXXXXXXXXXXXXXXXX(未UP)
- mqtt_aws.py
- omron_usb_gatt_aws.py

サンプルの修正

- 「mqtt_aws.py」の以下の赤文字の箇所をご自身の環境にあわせて編集し、ファイルを保存・終了してください

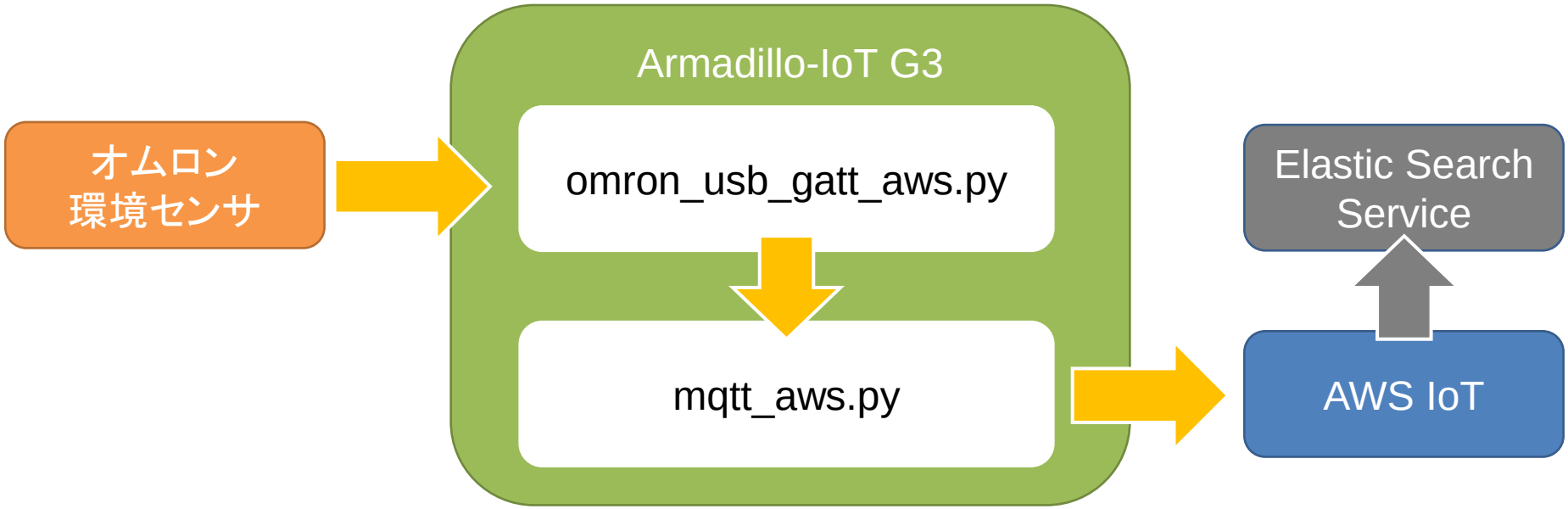
```
…(省略)

def get_data(data):
    myMQTTClient = AWSIoTMQTTClient("myClientID")
    myMQTTClient.configureEndpoint( “AWSIoTエンドポイント", 8883)
    myMQTTClient.configureCredentials("[AmazonRootCA1.pemのPATH]", "[pprivate.
pem.keyのPATH]", "[certificate.pem.crtのPATH]")

(省略) …
```

サンプルプログラムの構成

- 先ほどダウンロードしたサンプルプログラムでは、
  オムロン環境センサからデータを受け取り、
  受け取ったデータをJSON形式にして
  mqtt_aws.pyに渡し、AWS IoTに送信しています。



AWSの環境の構築

Atmark Techno

- Elastic Search ServiceとKibanaの環境は、以下のAmazon Web Servicesブログを参考に構築させていただきました。

- Web Servicesブログ：
  https://aws.amazon.com/jp/blogs/news/get-started-with-amazon-elasticsearch-service-use-amazon-cognito-for-kibana-access-control/

サンプルの実行

- 以下のコマンドのように引数にオムロン環境センサのBDアドレスを追加して、omron_usb_gatt_aws.pyファイルを実行してください

```
[armadillo ~]# python3 omron_usb_gatt_aws.py [BDアドレス]
b’ ¥x13¥a7¥n¥xd8¥x1b%¥x00¥xe1q¥x0f¥x00W¥x10¥x00¥x00¥x90¥x01’
2727
{ “sensor” :{ “temp” :27.27,  “light” :37},” timestamp” :2020-07-31T01:54:29.989515” }
{ “sensor” :{ “temp” :27.27,  “light” :37},” timestamp” :2020-07-31T01:54:29.989515” }
OK
None
```

- サンプルを実行すると、上記のように送信した温度と照度が表示されます

実行結果例

- 作成したアプリケーションを実行すると、AWSのKibanaのダッシュボードに反映されます。
- 実際の画面で確認してみましょう