[AWS Black Belt Online Seminar] FreeRTOS

サービスカットシリーズ

Solutions Architect 飯田 起弘
2020/8/19

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive

自己紹介

飯田 起弘（いいだ たつひろ）

AWS プロトタイピングソリューションアーキテクト

電機メーカーでソフトウェアエンジニアとしてIoT関連の新規事業の立ち上げを経験の後、AWSにてプロトタイピングソリューションアーキテクトとして、IoT関連案件のPoC, 本番導入などの支援に携わる。

@studioeth

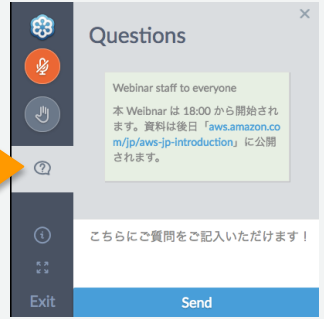
AWS Black Belt Online Seminar とは

「サービス別」「ソリューション別」「業種別」のそれぞれのテーマに分かれて、アマゾン ウェブ サービス ジャパン株式会社が主催するオンラインセミナーシリーズです。

質問を投げることができます！

- 書き込んだ質問は、主催者にしか見えません
- 今後のロードマップに関するご質問は
  お答えできませんのでご了承下さい

① 吹き出しをクリック
② 質問を入力
③ Sendをクリック



Twitter ハッシュタグは以下をご利用ください
#awsblackbelt

内容についての注意点

- 本資料では2020年8月19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には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.

アジェンダ

- 背景
- FreeRTOSとは
- FreeRTOS kernel
- FreeRTOS ライブラリ
- 開発方法
- まとめ

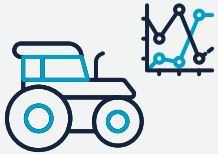
IoTのユースケース

生産性と
プロセスの最適化

リモートで患者の健康と
状態をモニター

在庫の可視化と
倉庫業務の最適化

家庭、建物、都市のスマート化
及びより良い体験の構築



効率的に良品の
作物を育てる

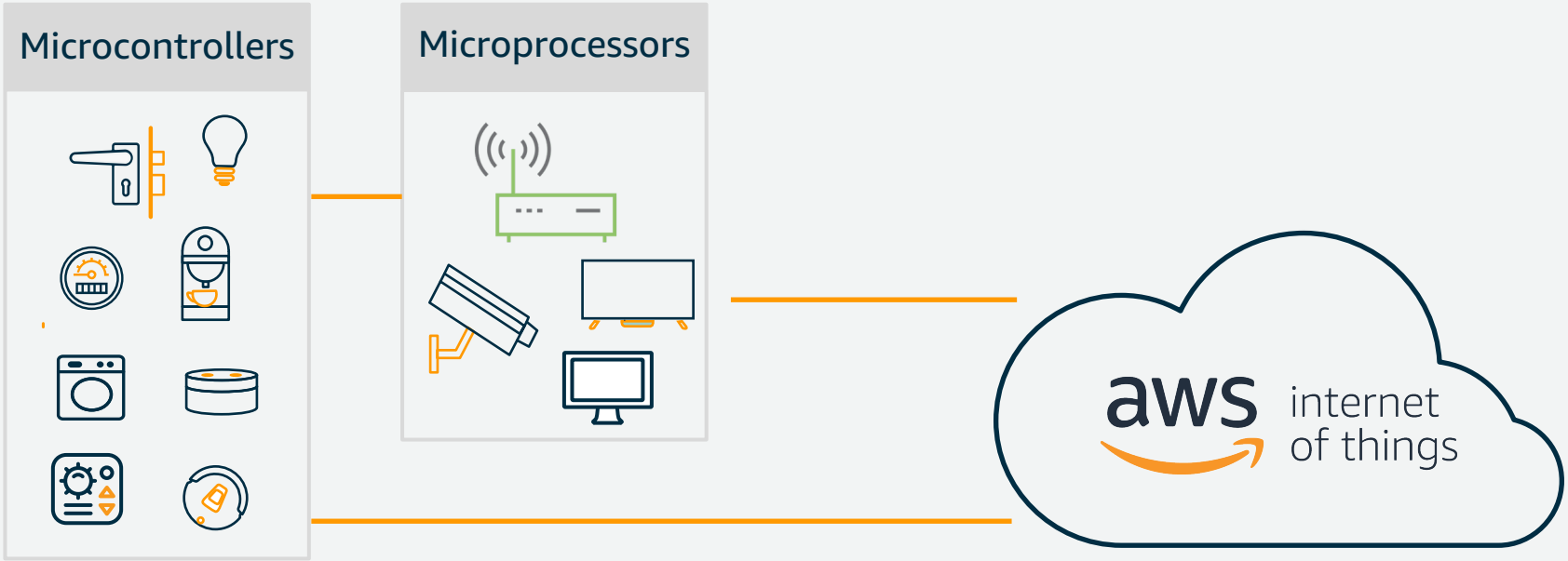
エネルギー資源を
効率的に管理

コネクテッドカー、
自動運転で輸送の変革

家庭、オフィス、工場
の安全性向上

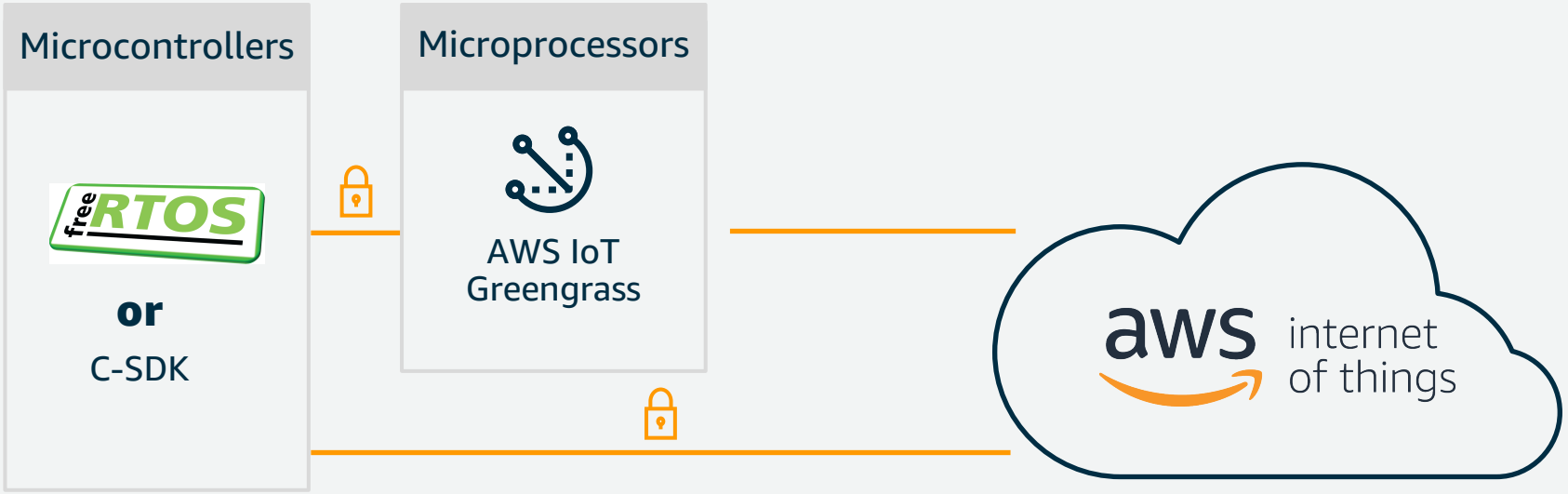
AWS IoTと繋がるデバイス を作るには?

何億ものIoTデバイスは
マイコンやマイクロプロセッサで 動作している



IoT化の課題

セキュアにAWS IoTと接続



アジェンダ

- 背景
- **FreeRTOSとは**
- FreeRTOS kernel
- FreeRTOS ライブラリ
- 開発方法
- まとめ

FreeRTOS

15年間にわたり広く配布された信頼性

RISC-V やArm v8-Mなど
40以上のアーキテクチャでサポート

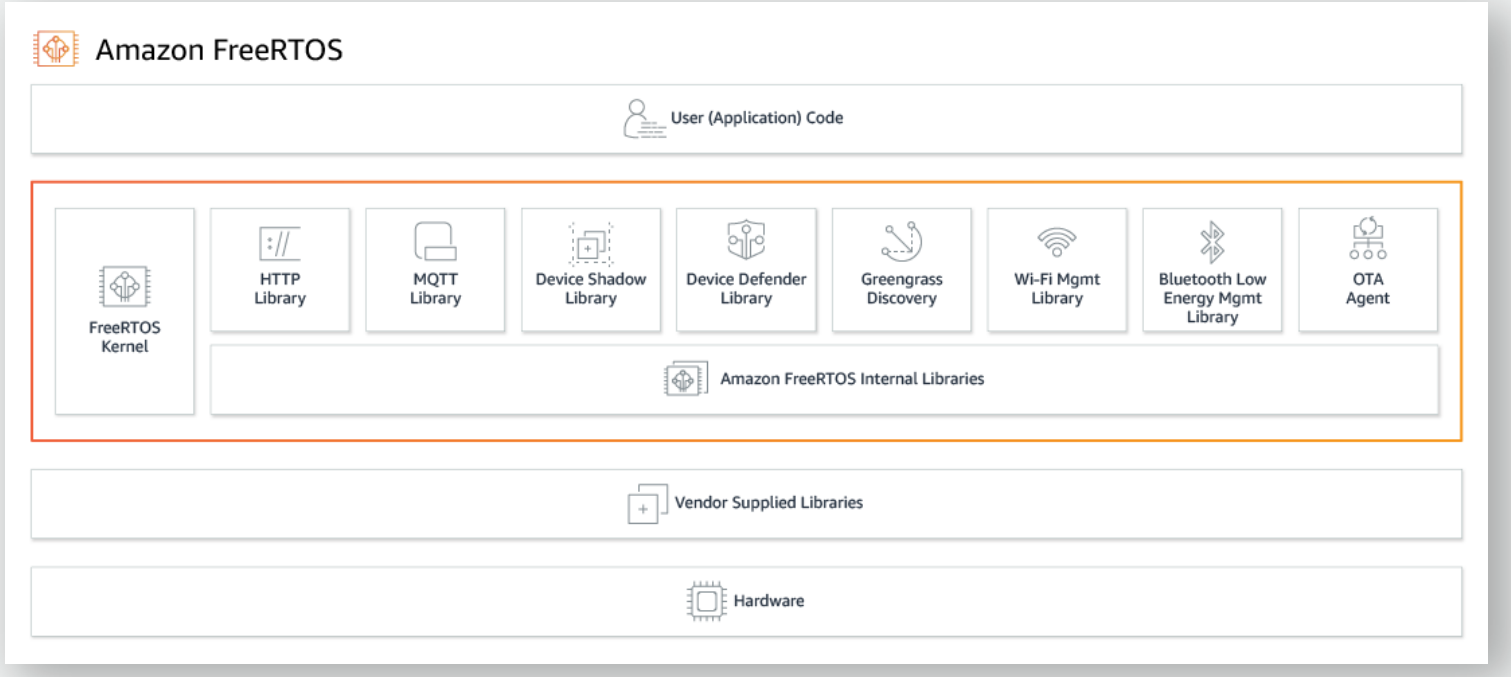
広範なエコシステム

MITライセンスで無償で商用利用可能

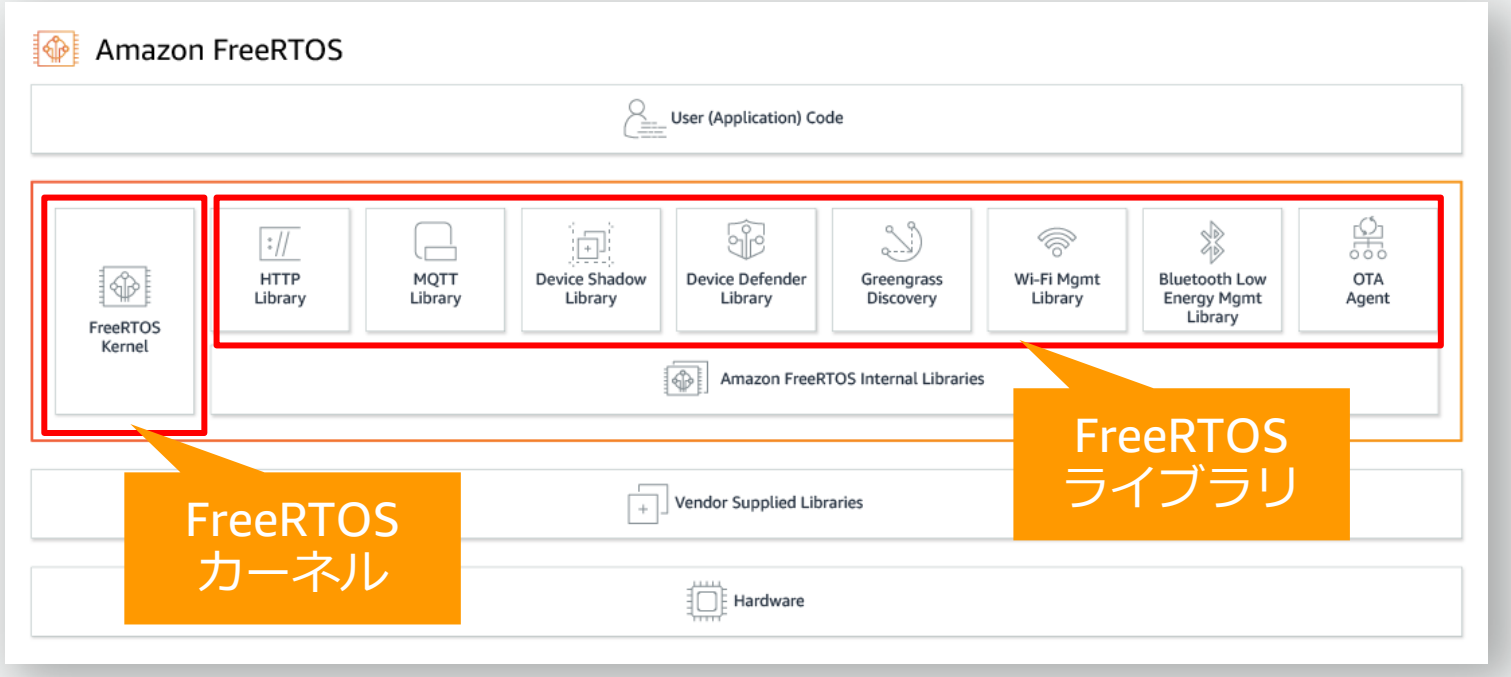
ストリームやメッセージバッファによる
プロセス間通信（IPC）機能

aws

FreeRTOS Architecture



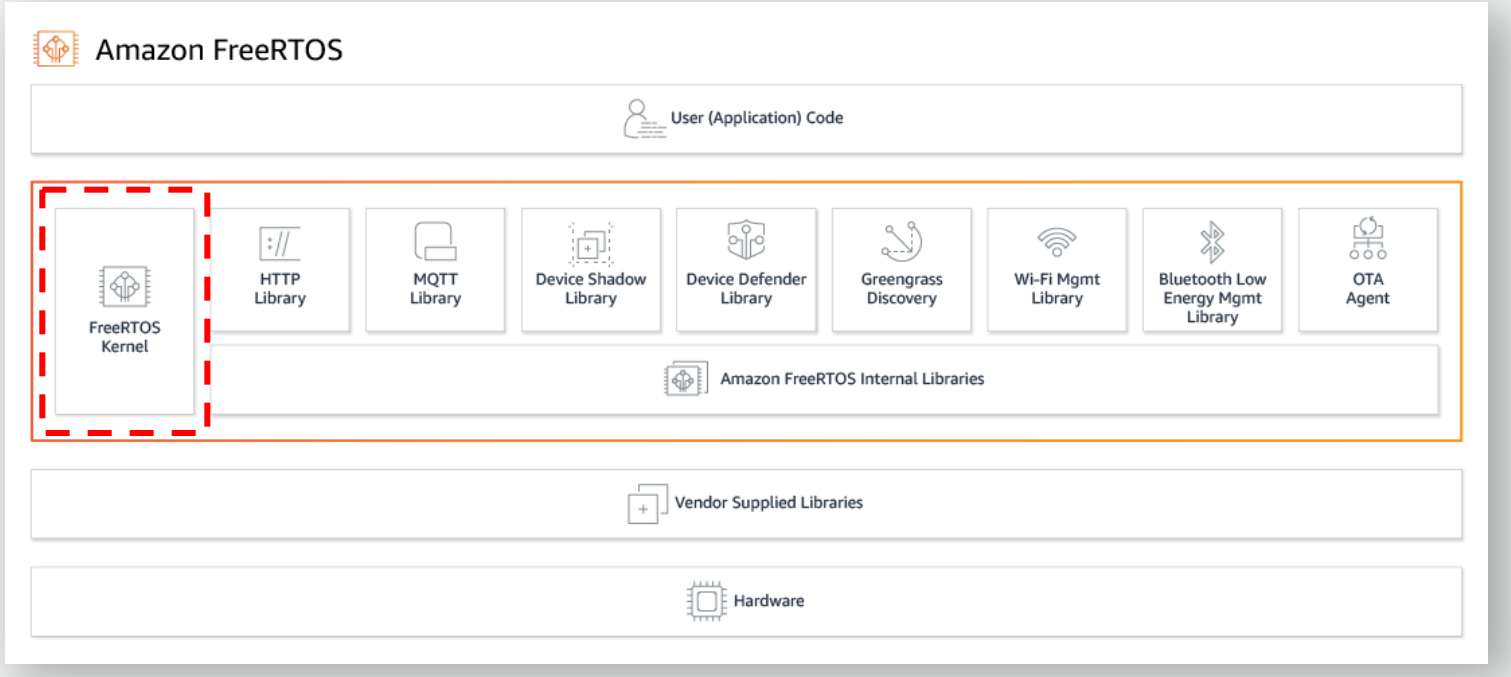
FreeRTOS Architecture



アジェンダ

- 背景
- FreeRTOSとは
- **FreeRTOS kernel**
- FreeRTOS ライブラリ
- 開発方法
- まとめ

FreeRTOS Architecture



FreeRTOS Kernel

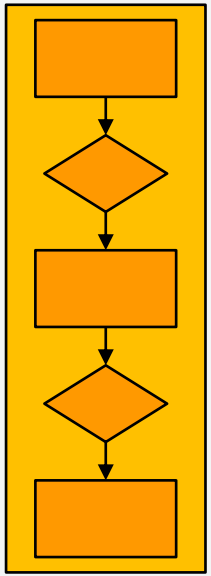
- マルチタスクスケジューラ
- タスク通知、メッセージキュー、セマフォ、ストリーム、メッセージバッファなど、タスク間調整機能
- 複数のメモリ割り当てオプション (静的メモリ割り当ても含む)
- カーネルバイナリイメージ： 4K～9K バイト

https://docs.aws.amazon.com/ja_jp/freertos/latest/userguide/dev-guide-freertos-kernel.html

aws

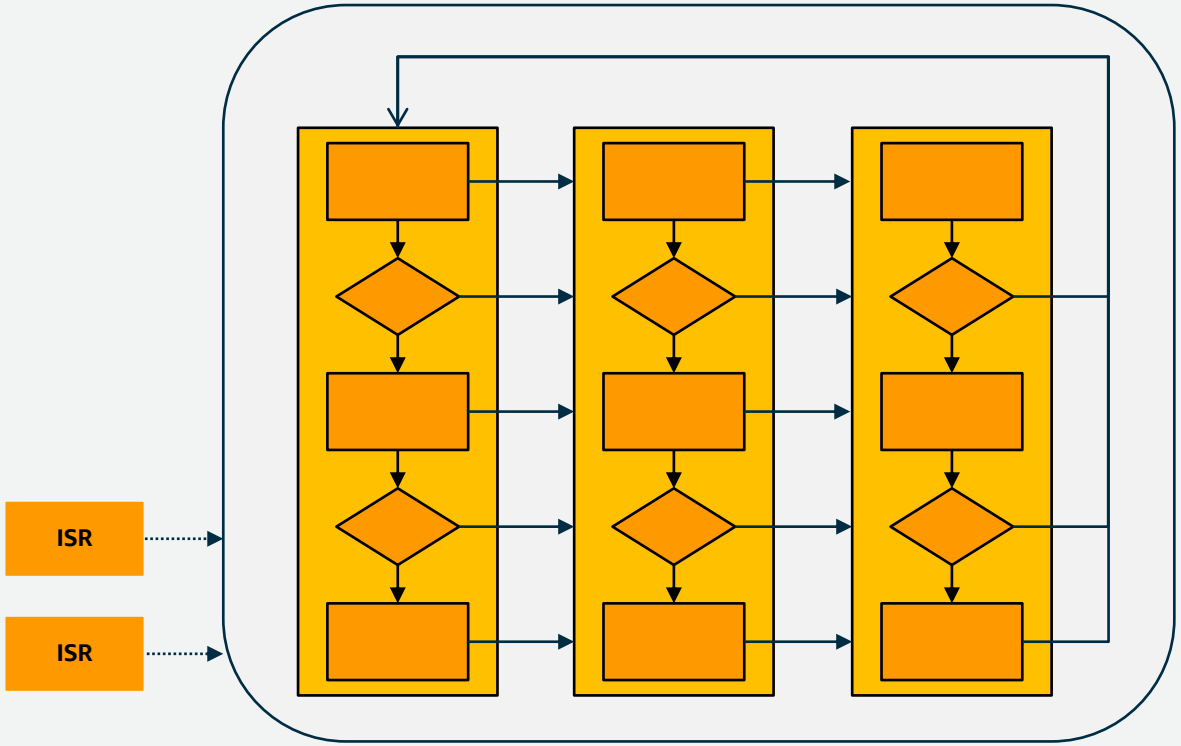
組み込み開発

- 手続き的
- 単純な制御



組み込み開発

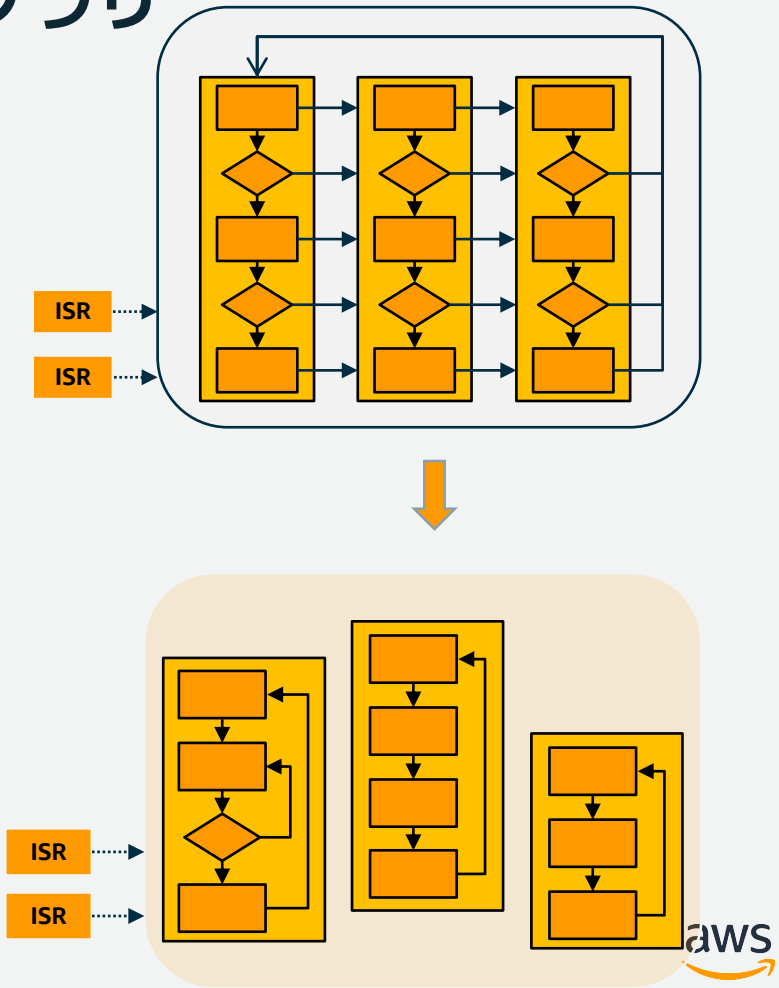
- 要求が高度に
- 設計は複雑化
- スケールが難しい



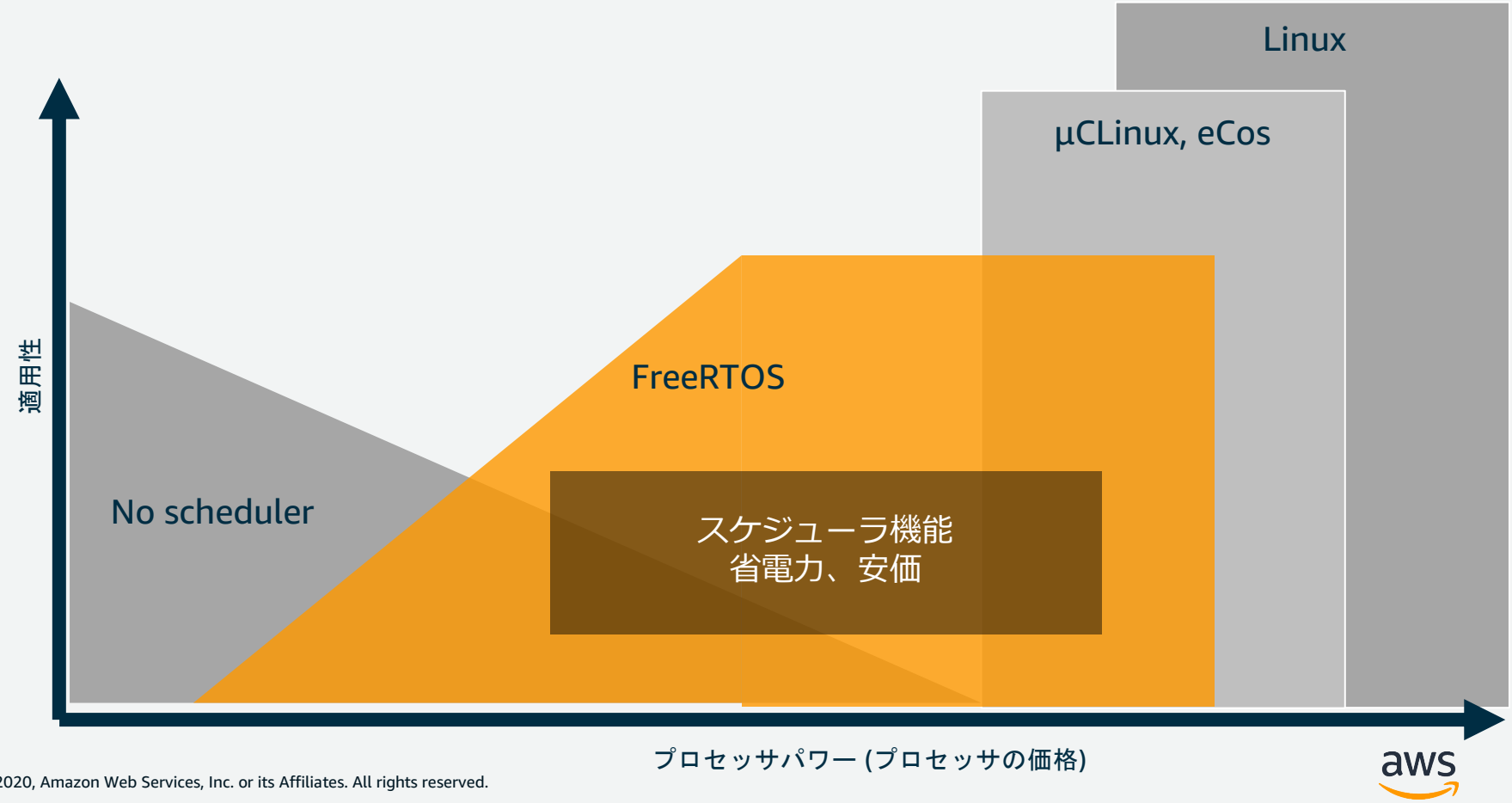
マルチスレッドを実装するライブラリ

FreeRTOS

- タスク、スケジューラ等の機能
- メンテナンス性の維持
- リアルタイム要件を満たす
- ポータビリティ(ハードウェア変更)
- 低コストで市場に素早く投入

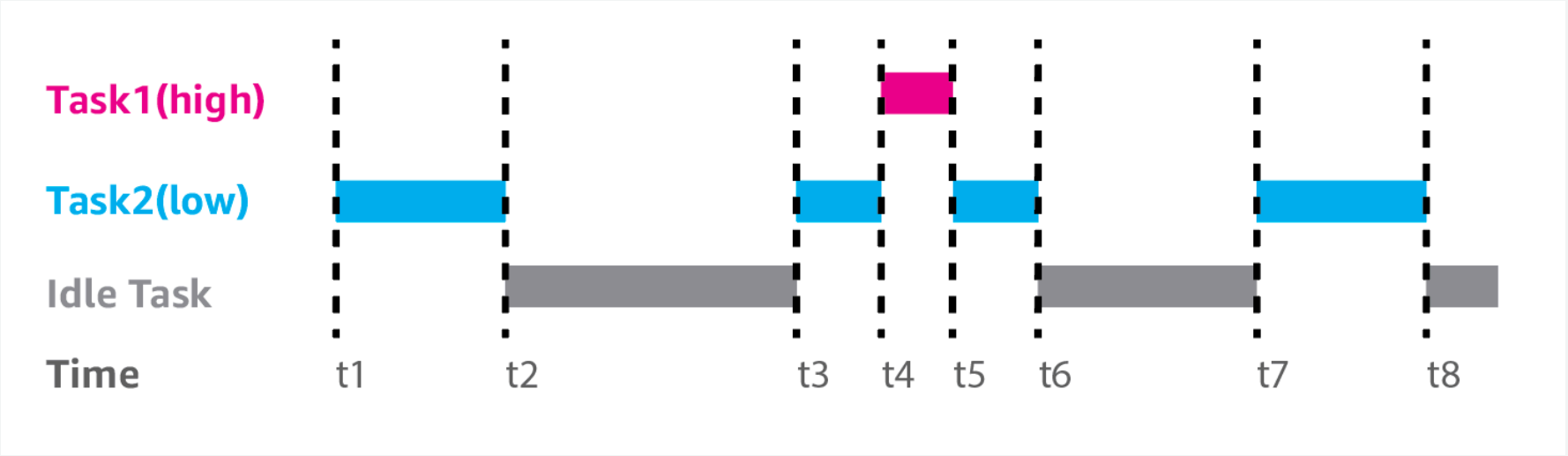


マイコン向けのOS



タスクスケジューラ

- アプリケーションは複数のタスクで構成される
- タスクには厳密な優先順位を割り当てられる
- FreeRTOSのスケジューラがタスクを管理
- 優先度の高いタスクを確実に実行

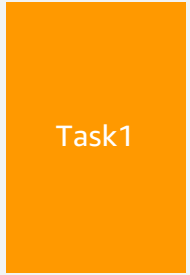


FreeRTOSのAPI

- タスク
- キュー
- ミューテックス・セマフォ
- 通知
- ストリームバッファ
- など

タスク

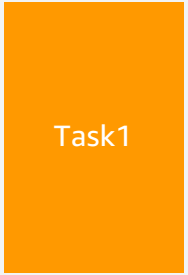
- FreeRTOSではスレッドをTaskと呼ぶ
- タスクのループの中で処理を行う




```
void vTask1( void *pvParameters )
{
    // タスクで使用する変数
    int iVariableExample = 0;
    for(;;)
    {
        // ここに処理を記載
    }
    vTaskDelete(NULL);
}
```

タスクの作成

- xTaskCreate() API 関数を用いる




```
portBASE_TYPE xTaskCreate( pdTASK_CODE pvTaskCode,
    const signed portCHAR * const pcName,
    unsigned portSHORT usStackDepth,
    void *pvParameters,
    unsigned portBASE_TYPE uxPriority,
    xTaskHandle *pxCreatedTask
);
```

- pvTaskCode: タスクを実装した関数
- pcName: タスクの名前（デバッグ用途）
- usStackDepth: スタックサイズ
  - 32bit幅の場合に100を入れると、400バイト確保される
- pvParameters: タスクにわたすパラメータ
- uxPriority: タスクの優先度
  - configMAX_PRIORITIES-1が最大
- pxCreatedTask: ハンドラー
  - タスクの優先度変更、削除、通知などに使用
- Return Value
  - pdTRUE: 成功
  - errCOULD_NOT_ALLOCATE_REQUIRED_MEMORY

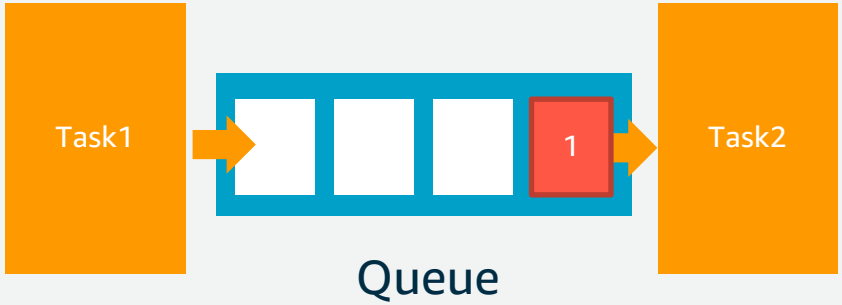
aws

タスクの実行例

```
static void vTask1(void *pvParameters)
{
    for( ;; )
    {
        //ここに処理を記載
        vTaskDelay( pdMS_TO_TICKS( 10000 ) );
    }

    vTaskDelete( NULL );
}


xTaskCreate( vTask1, "Task 1", 2048, NULL, 0, NULL );
```

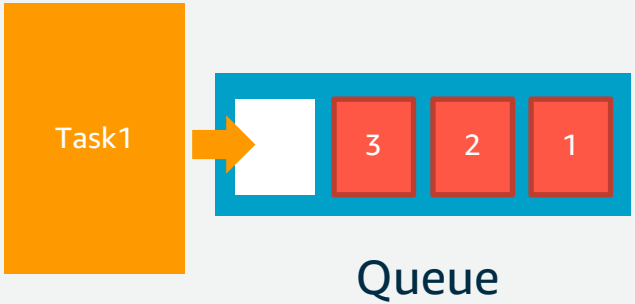
キュー（Queue）

- メッセージ（データ）の受け渡し
- FIFO(先入れ先出し)
- 任意の型を指定可能



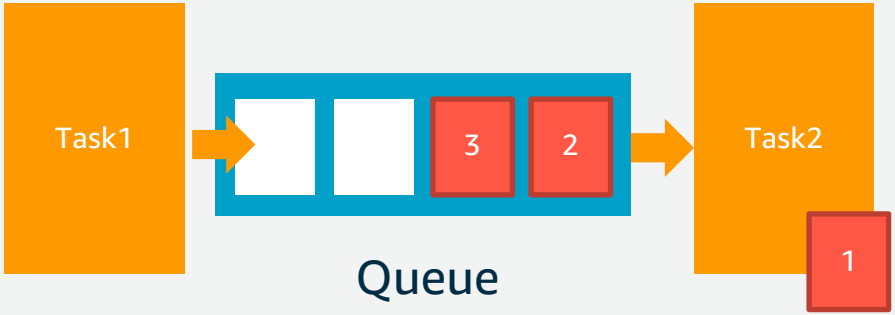
キューの作成, 追加

```
unsigned long ulVar = 10UL;
QueueHandle_t xQueue1;

void vTask1( void *pvParameters )
{
  xQueue1 = xQueueCreate( 4, sizeof( unsigned long ));
  xQueueSendToBack( xQueue1, (void *)&ulVar, (TickType_t) 10)
  …
}
```



キューの取得

```
if( xQueueReceive( xQueue1, &( xRxedStructure ),
     ( TickType_t ) 10 ) == pdPASS ){

     /* xRxedStructure now contains a copy of xMessage. */

}
```

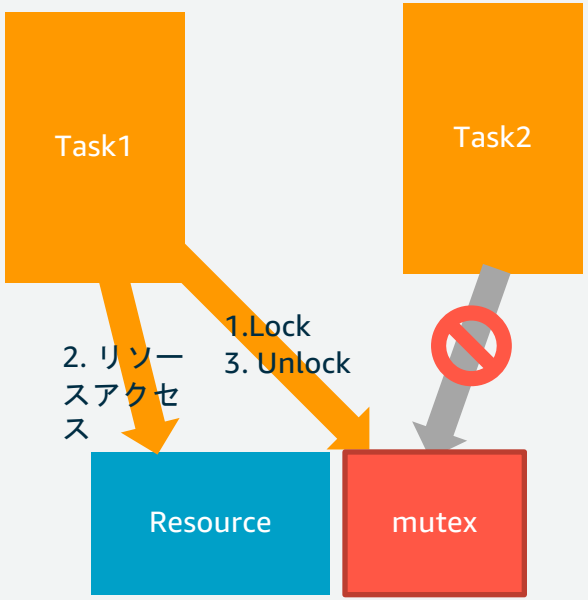


ミューテックスとセマフォ

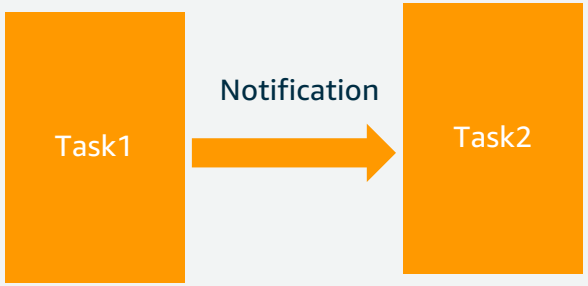
- ミューテックス（Mutex）
  - バイナリセマフォ
  - 排他制御
    - 特定のリソースへの操作を複数のタスクで行う場合で、リソースの競合を避けるために使用
- セマフォ（Semaphore）
  - タスク間のシグナリング
  - リソースへのアクセス数の管理
- 例：駐車場のスペース
  - Mutexの場合：スペースは1台分のみ
  - Semaphoreの場合：スペースは複数台

ミューテックス

```
SemaphoreHandle_t xMutex;

void myTask1(void *p){
     if(xSemaphoreTake(xMutex, portMAX_DELAY) == 1){
          // ここでリソースにアクセス
          xSemaphoreGive(xMutex);
}}

void myTask2(void *p){
     if(xSemaphoreTake(xMutex, portMAX_DELAY) == 1){
          // ここでリソースにアクセス
          xSemaphoreGive(xMutex);
}}
```



タスク通知（Task Notification）

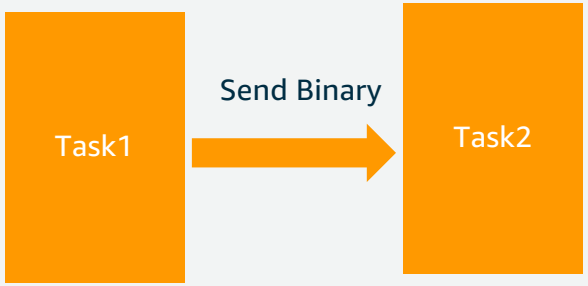


- タスク間の通知
- セマフォより高速、省メモリ

```
void myTask1(void *p){
     xTaskNotifyGive(myTask2Handle);
     …
}

void myTask2(void *p){
     if(ulTaskNotifyTake(pdTRUE, (TickType_t)portMAX_DELAY) > 0){
          // notify received
           …
     }
}
```

ストリームバッファ（Stream Buffers）

- タスク間でデータストリーム送信
- 指定したサイズ以上のデータがたまると受信側のタスクを実行
- Message Buffersという個別のメッセージを扱うAPIもある

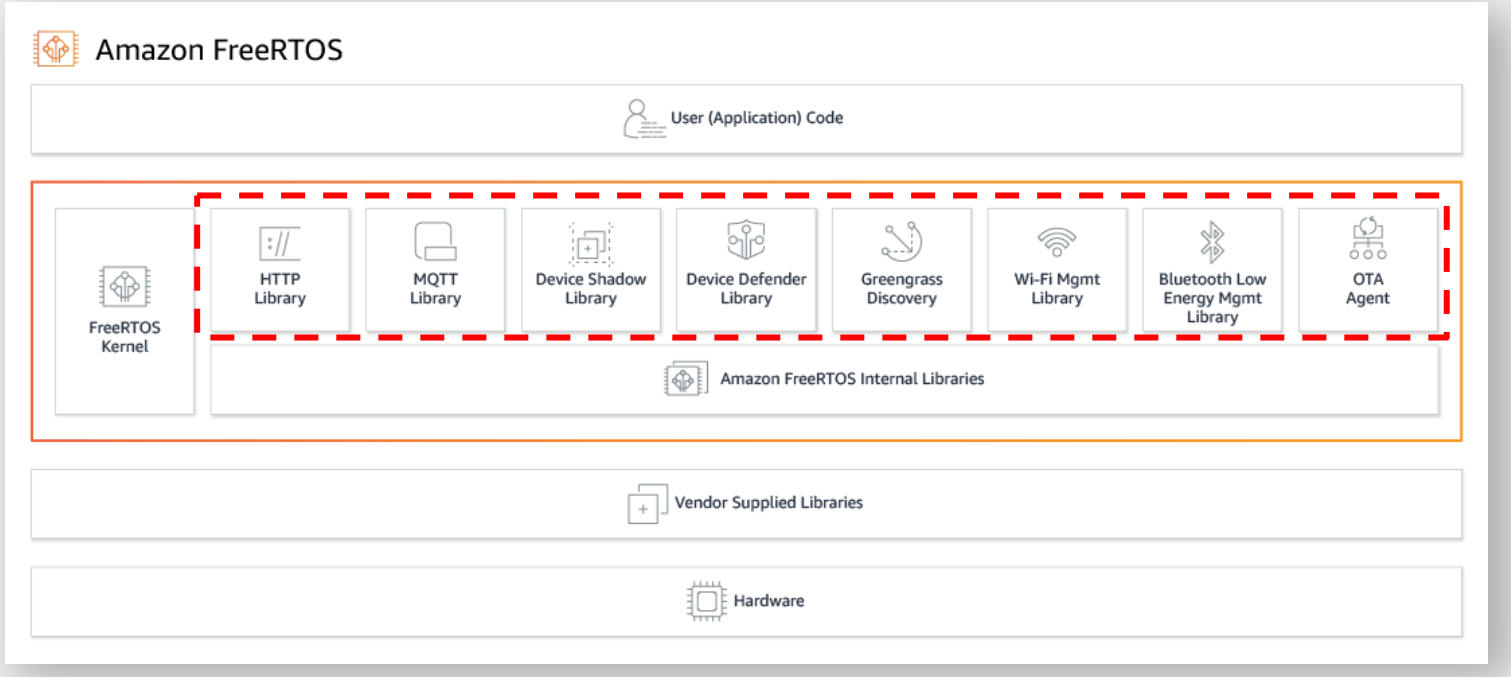



```
xStreamBuffer = xStreamBufferCreate( xStreamBufferSizeBytes, xTriggerLevel );

void myTask1(void *p){
     uint8_t ucArrayToSend[] = { 0, 1, 2, 3 };
     xBytesSent = xStreamBufferSend( xStreamBuffer,
      ( void * ) ucArrayToSend, sizeof( ucArrayToSend ), x100ms );
}

void myTask2(void *p){
     xReceivedBytes = xStreamBufferReceive( xStreamBuffer,
      ( void * ) ucRxData, sizeof( ucRxData ), xBlockTime );
     if( xReceivedBytes > 0 ) {
     }
}
```

アジェンダ

- 背景
- FreeRTOSとは
- FreeRTOS kernel
- **FreeRTOS ライブラリ**
- 開発方法
- まとめ

Amazon FreeRTOS Architecture



aws

FreeRTOS Libraries

ローカルの接続性

クラウド接続無しで AWS IoT Greengrass デバイスと通信

クラウドの接続性

マイコンデバイスから簡単にデータを収集、アクションを実行

セキュリティ

デバイスのデータと接続をセキュアに

OTA コード署名

セキュリティ更新、バグ修正などのファームウェアをデプロイ

クラウドの接続性

デバイスからクラウドにデータを送信
様々なAWSサービスを用いた分析

AWS IoT Coreへの接続

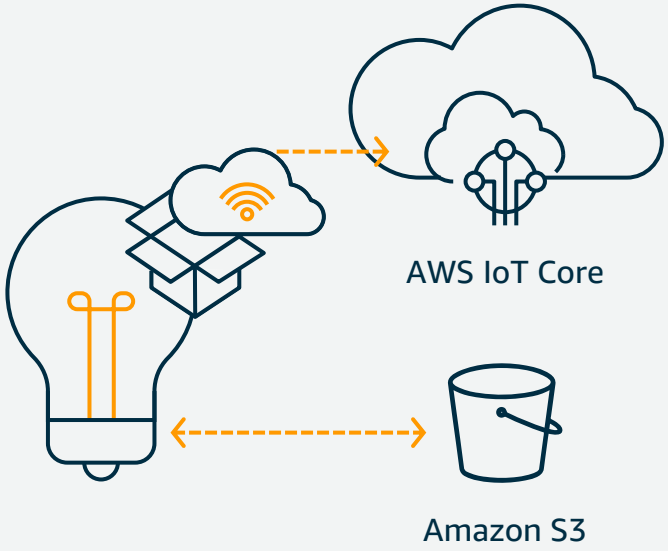
MQTT Pub/Subメッセージング

HTTPS経由でのファイルのアップロード・ダウンロード
e.g., Amazon S3

デバイスシャドウのサポート

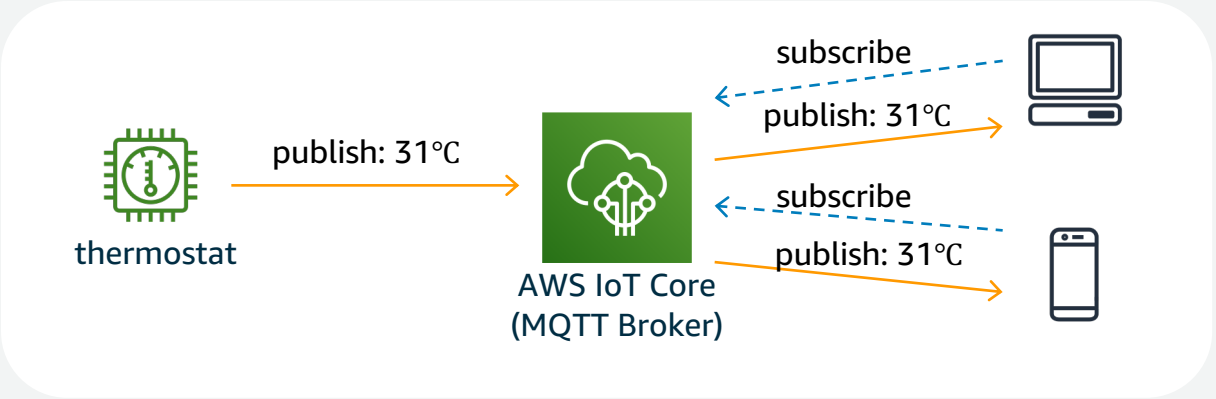
AWS IoT Device Managementなど、AWS IoT Core の機能が利用可能

ベンダーに依存しないライブラリインターフェースで、素早いオンボーディング



MQTTプロトコル

- シンプルで軽量なメッセージプロトコル
  - 制約のあるデバイスや不安定なネットワークを想定した設計
  - 消費する通信帯域やデバイスに要求するリソースを最小化
- Publish / Subscribe モデル
- Topicを介して、双方向、1:n の通信が可能



MQTT ライブラリ

- LWT (遺言) を利用可能
- QoS0 (At most once)、 QoS1 (At least once) をサポート

接続・Publishの例

```
IotMqttError_t result = IotMqtt_Connect( &networkInfo,
                                         &connectInfo,
                                         timeoutMs,
                                         &mqttConnection );

publishStatus = IotMqtt_Publish( mqttConnection,
                                 &publishInfo,
                                 0,
                                 &publishComplete,
                                 NULL );
```


aws

Device Shadow ライブラリ

デバイスが AWS IoT に接続されているかどうかにかかわらず、
アプリ等でデバイスの状態を利用可能に

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

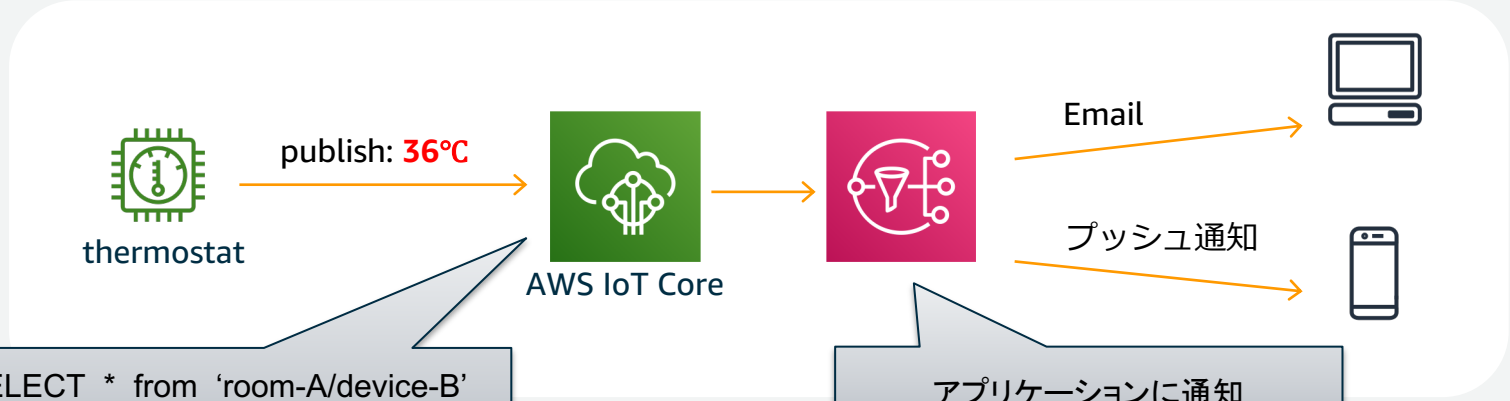
デバイスに対して変更したいステータスをセット
最新の通知されたステータスを取得

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


aws

AWS IoT ルールの利用

- デバイスのメッセージをトリガーに様々なAWSのサービスを利用
  - バイナリデータのパース
  - データベースへの保存
  - ユーザへの通知
  - データレイク
  - など



ローカル接続

クラウド接続なしで、ローカルデバイスと接続

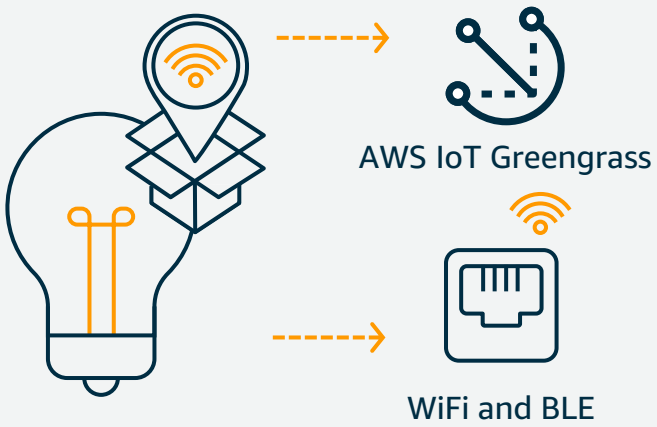
AWS IoT Greengrassのディスカバリー機能のサポートを含む、エッジゲートウェイとのローカル通信

Wi-Fi ライブラリ：Wi-Fiの設定、プロビジョニング、セキュリティ、省電力管理などのWi-Fi機能の抽象化レイヤーを実装

BLE管理ライブラリ：GATTやGAPなどのBLE機能の抽象化レイヤーを実装
クラウド機能と統合するためのiOSおよびAndroid用のコンパニオンSDK

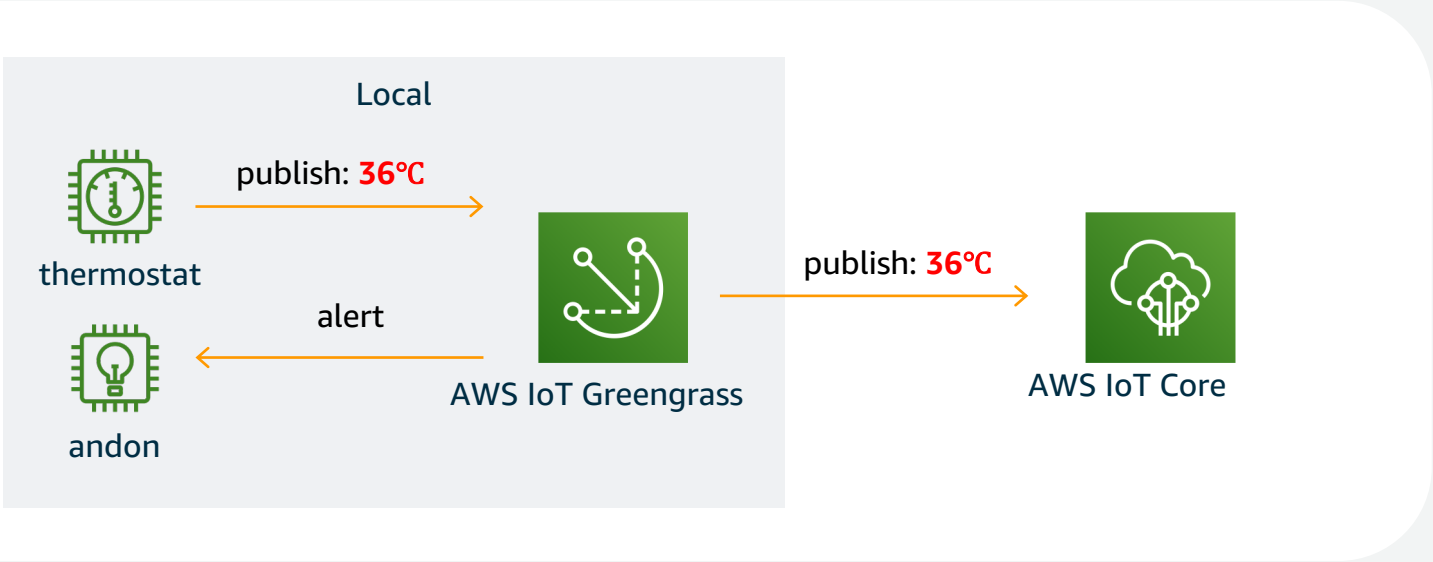
クラウドへの接続が無くても、
コミュニケーションとデータ収集を継続

多くのネットワークトポロジと
ユースケースのサポート



AWS IoT Greengrass 検出ライブラリ

- ネットワーク上の Greengrass コアを検出
- ローカルでMQTT Pub/Sub機能を実現
- オフラインでも継続した制御操作が可能



Wi-Fiライブラリ

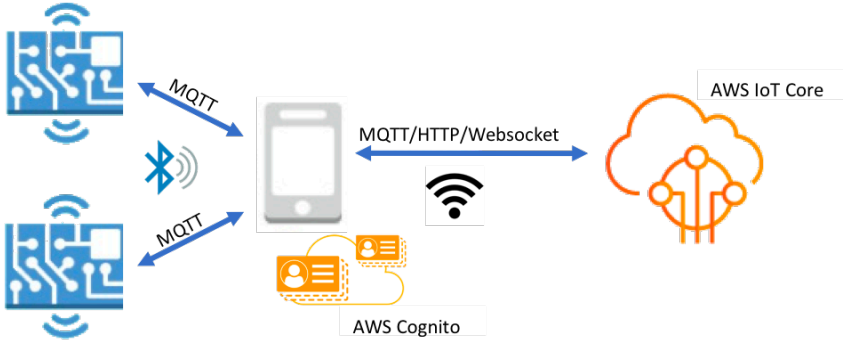
アプリケーションは、SSID, Passwordなどを設定

```
INetworkParams_t xNetworkParams;
WIFIReturnCode_t xWifiStatus;

xWifiStatus = WIFI_On();

xNetworkParams.pcSSID = clientcredentialWIFI_SSID;
xNetworkParams.pcPassword = clientcredentialWIFI_PASSWORD;
xNetworkParams.xSecurity = eWiFiSecurityWPA2;

xWifiStatus = WIFI_ConnectAP( &( xNetworkParams ) );
```


aws

MQTT over BLE

- モバイルを経由し BLE デバイスをAWS IoT に接続
- MQTT Agent が BLE をサポート
- iOS と Android SDK がプロキシライブラリを提供する
- Amazon Cognito がプロキシと AWS IoT 間の認証を行う
- Shadow, Device Defender, OTA の機能も BLE 経由で利用可能



https://docs.aws.amazon.com/ja_jp/freertos/latest/userguide/ble-demo.html#ble-demo-mqtt

Wi-Fi Provisioning over BLE

- Wi-Fi の Credential を モバイルデバイスを使って BLE で設定できる
- 複数のネットワークの設定、優先順位付け、追加、削除が可能
- BLE Secure Connections をサポート (BLE 4.2 以上)

https://docs.aws.amazon.com/ja_jp/freertos/latest/userguide/ble-demo.html#ble-demo-wifi

セキュリティ

デバイスセキュリティを向上

TLSによるSecure sockets

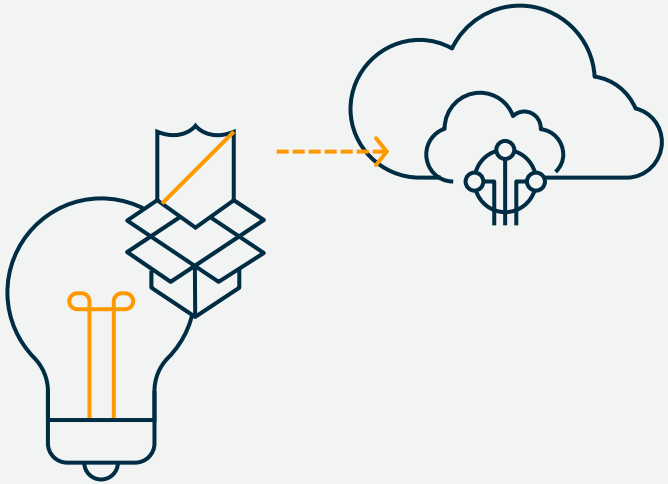
証明書ベースの認証

キー管理用のPKCS #11 インターフェース

セキュアエレメントのサポート

オープンネットワークポート無し

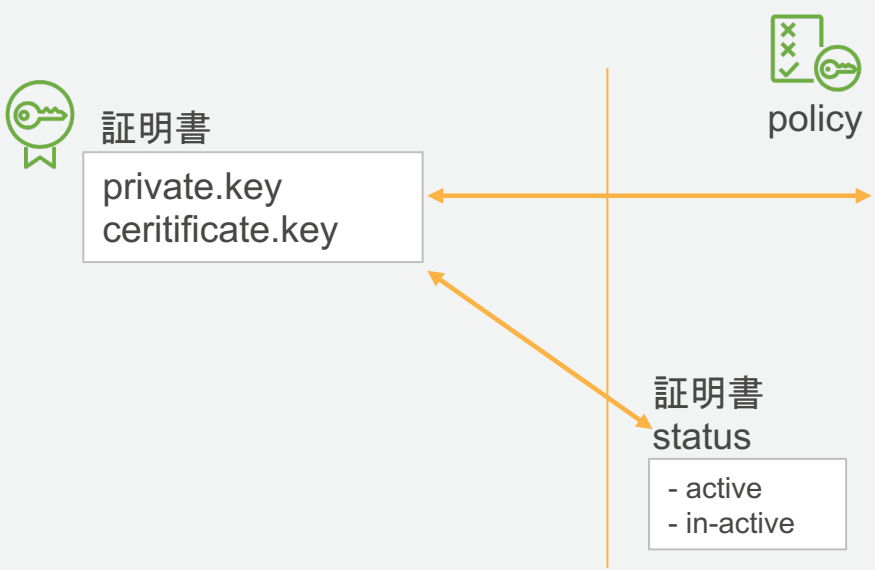
信頼できるコードのみを実行



TLS を使った相互認証

IoTポリシーを用いた認可

デバイス：AWS IoTに証明書を用いて接続
AWS IoT：挙動の正当性をpolicyに従い判定



IoT ポリシーの例

```
{
    effect: allow
    action: ”iot:connect”
     resource : “arn:thing name”
},
{
    effect: allow
    action: “iot:publish”
     resource : “topic
arn/data/send”
},
{
    effect: deny
    action: “iot:subscribe”
    resource: “*”
}
```

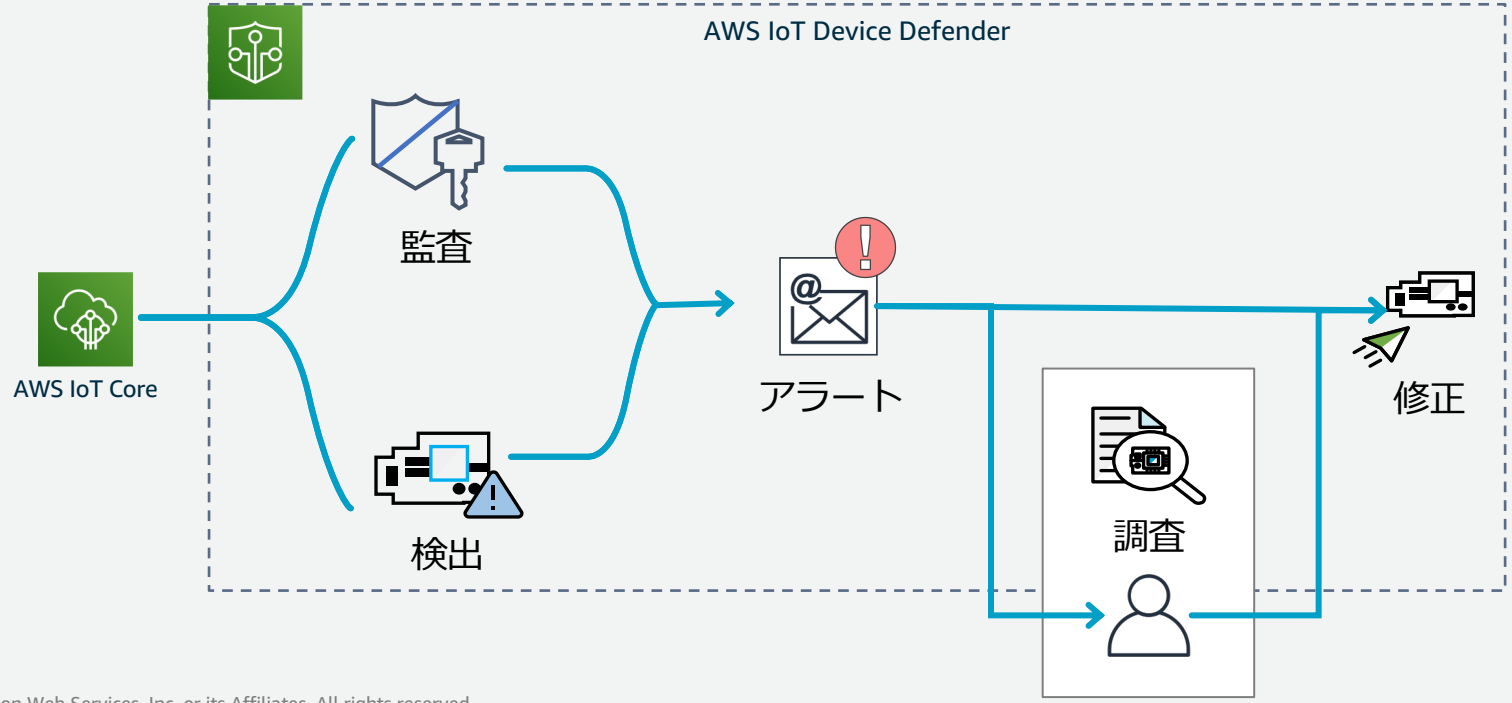
AWS IoT Device Defender ライブラリ

- AWS IoT Device Defenderによる接続デバイスをモニタリング
- 異常動作の検出およびセキュリティリスクの軽減
- デバイスが侵害された場合にはすばやく応答することが可能

```
startInfo.pClientIdentifier = pIdentifier;
startInfo.clientIdentifierLength = ( uint16_t ) strlen( pIdentifier );
startInfo.callback = callback;
startInfo.mqttConnection = mqttConnection;

defenderResult = AwsIotDefender_Start( &startInfo );
```

AWS IoT Device Defender とは

AWS IoT Device Defenderは、フル・マネージドのIoTセキュリティ
サービスであり、接続されたデバイス群を継続的に保護することができます



監査内容 (FreeRTOS側の設定は不要)

- **REVOKED_CA_CERT_CHECK**
  - CA 証明書が取り消されましたが、AWS IoTで有効になっている
- **DEVICE_CERTIFICATE_SHARED_CHECK**
  - 複数の同時接続が同じ X.509 証明書を使用して AWS IoT サービスに対して認証されている
- **DEVICE_CERTIFICATE_KEY_QUALITY_CHECK**
  - 登録されている証明書が求めているセキュリティを満たしているかを確認
- **CA_CERTIFICATE_KEY_QUALITY_CHECK**
  - 登録されているCAの証明書が求めているセキュリティーを満たしているかを確認
- **UNAUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK**
  - Cognito未認証ユーザーの権限が推奨されている以上の権限が付与されているかを確認
- **AUTHENTICATED_COGNITO_ROLE_OVERLY_PERMISSIVE_CHECK**
  - Cognito認証ユーザーの権限が推奨されている以上の権限が付与されているかを確認
- **IOT_POLICY_OVERLY_PERMISSIVE_CHECK**
  - IoT Policyにワイルドカードのような広域な権限が付与されているかを確認

* 詳細なチェック内容はこちらを参照ください
https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/device-defender-audit-checks.html

監査内容 (FreeRTOS側の設定は不要)

- **IOT_ROLE_ALIAS_OVERLY_PERMISSIVE_CHECK**
  - Role Aliasで許可されいてるサービスで"*"が指定されているものを確認
- **IOT_ROLE_ALIAS_ALLOWS_ACCESS_TO_UNUSED_SERVICES_CHECK**
  - Role Aliasで使われていないサービス権限が着いているかをチェック
- **CA_CERT_APPROACHING_EXPIRATION_CHECK**
  - CA 証明書が30日以内に有効期限が切れるか、既に切れているかをチェック
- **CONFLICTING_CLIENT_IDS_CHECK**
  - 複数のデバイスが同じクライアント ID を使用して接続しているかをチェック
- **DEVICE_CERT_APPROACHING_EXPIRATION_CHECK**
  - デバイス証明書が30日以内に有効期限が切れるか、既に切れているかをチェック
- **REVOKED_DEVICE_CERT_CHECK**
  - 取り消されたデバイス証明書が使われているかをチェック
- **LOGGING_DISABLED_CHECK**
  - AWS IoTの口グ設定が有効になっているかをチェック

* 詳細なチェック内容はこちらを参照ください
https://docs.aws.amazon.com/ja_jp/iot/latest/developerguide/device-defender-audit-checks.html

異常の検出

デバイス上で取得するメトリクス

- TCP 接続の確立回数
- 送信先 IP

FreeRTOSの AWS IoT Device Defender ライブラリは、メトリクスのサブセットのみをサポート

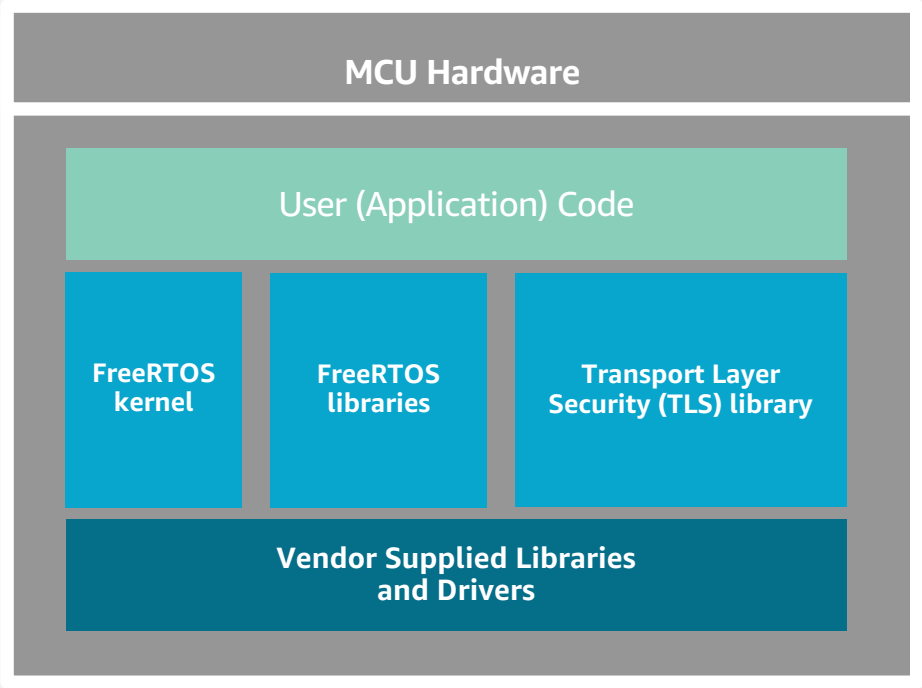
クラウド側で取得するメトリクス

- 認証エラー
- 接続試行
- 切断
- メッセージサイズ
- 受信したメッセージ
- 送信されたメッセージ
- 送信元 IP

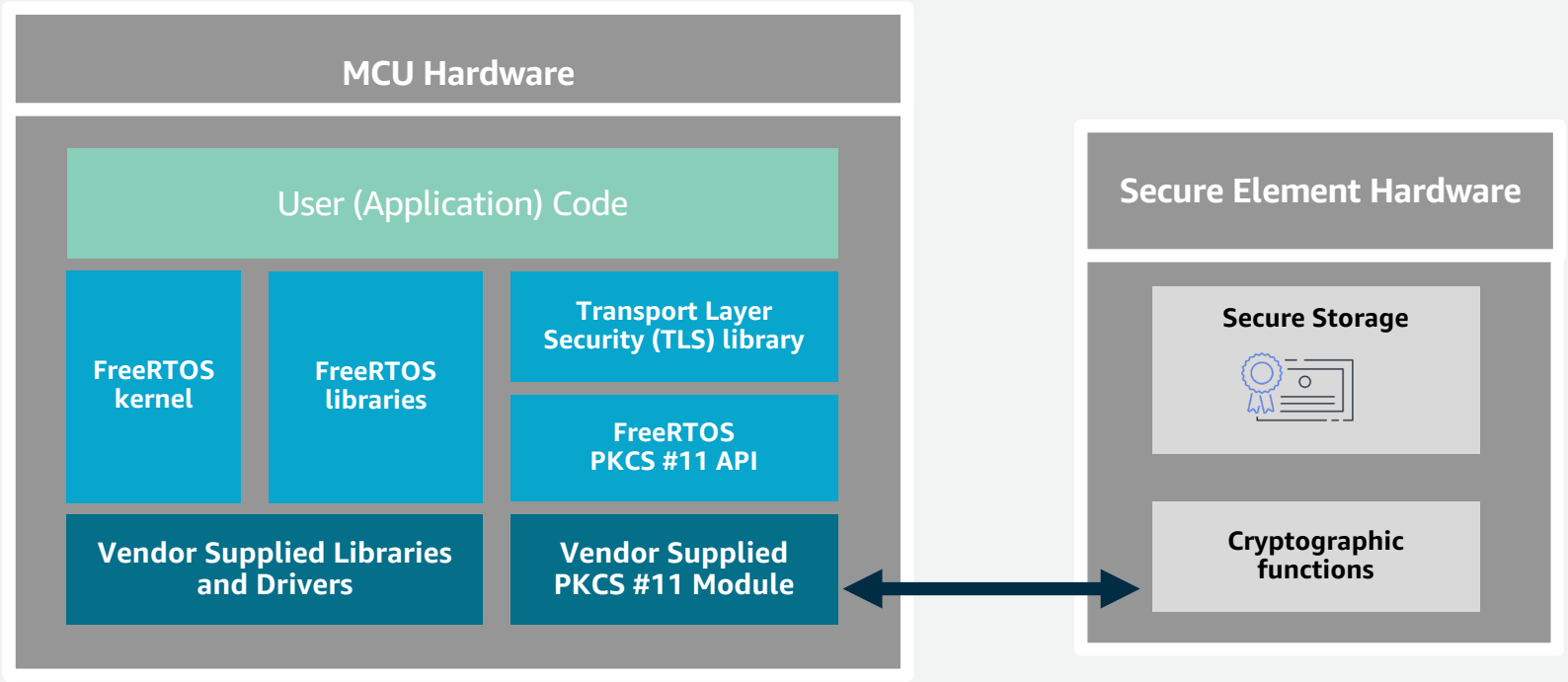
セキュアエレメントの使用

- 暗号鍵の改ざん防止のためのストレージが利用可能
- 物理的な管理外の場所にデバイスを配置する場合などに有用
- 暗号化機能のオフロードにより、メモリ空間を解放し、電力消費を削減
- オープンスタンダードのPKCS#11インターフェースを使用して最大の互換性を確保

セキュアエレメント無しの場合



セキュアエレメントを用いた場合



OTA コード署名

デバイスの機能向上やセキュリティ対策のためのファームウェア更新

AWS IoT Device Managementを使用して
グループにアップデートを割り当て

新しいファームウェアイメージへのコード署名

MQTT や HTTPSを用いてデバイスに更新を配布

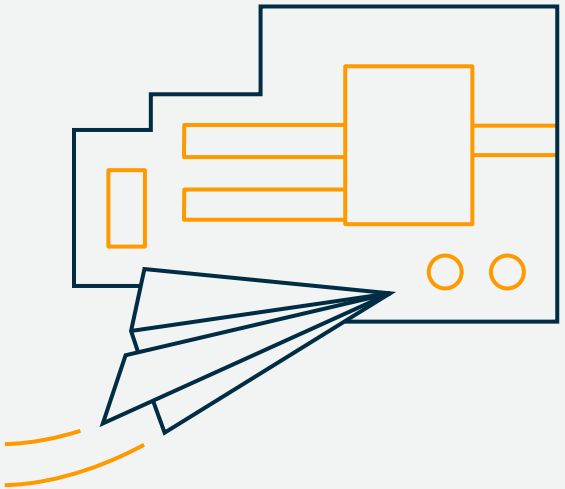
デバイス上での署名検証

インストールやリブート制御のためのAPI

シンプルなグループの管理

デバイスが信頼できるコードのみを実行

メモリ効率のよいアップデートクライアント



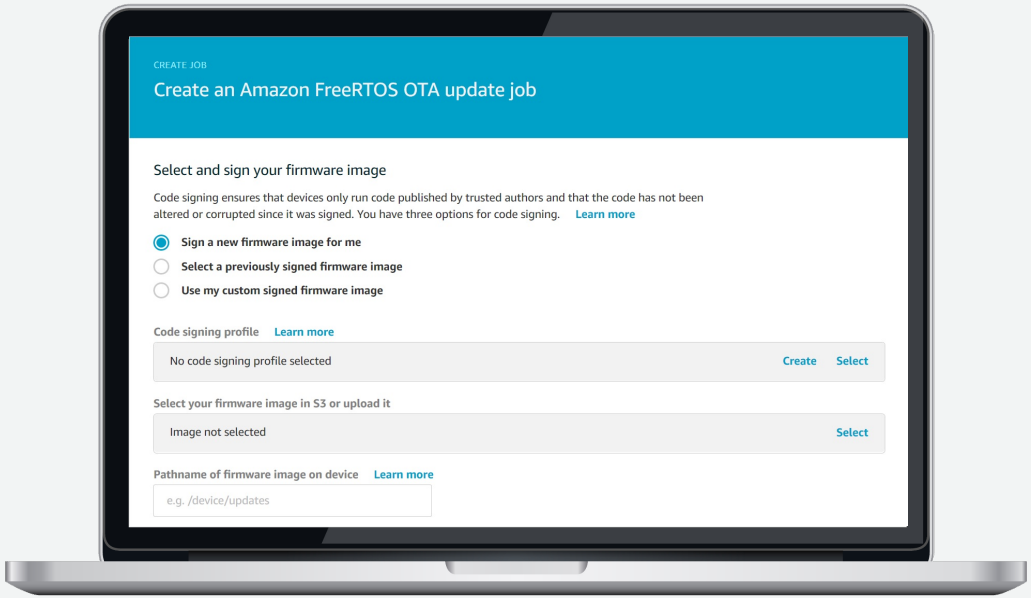
OTA エージェントライブラリ

- ファームウェアの更新を実現
- AWS IoT Device Managementのジョブと連携

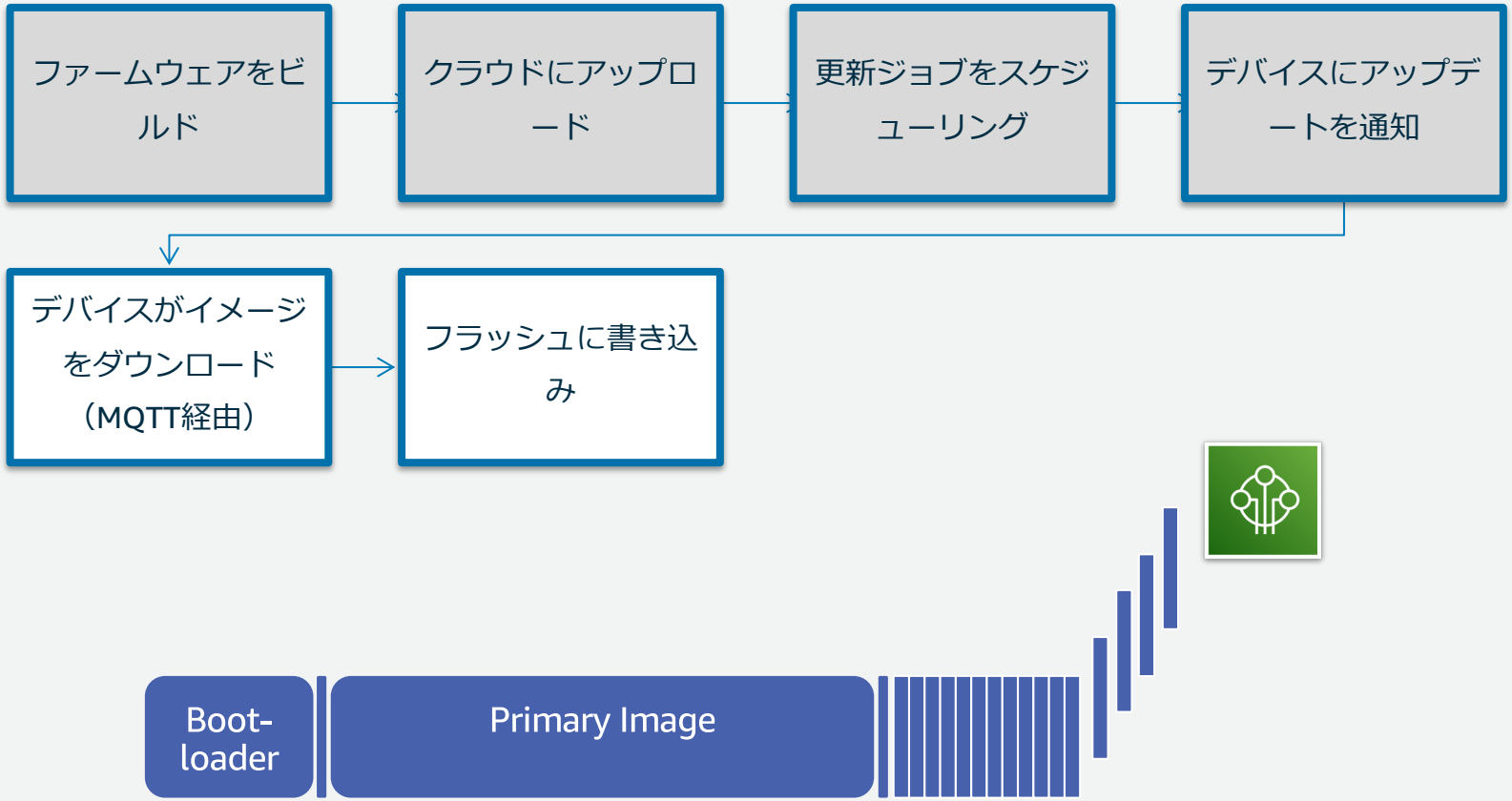
```
OTA_AgentInit( ( void * ) ( &xOTAConnectionCtx ),
                          ( const uint8_t * ) ( clientcredentialIOT_THING_NAME ),
                          App_OTACompleteCallback,
                          ( TickType_t ) ~0 );

while(  ( eState = OTA_GetAgentState() ) != eOTA_AgentState_Stopped  ) {
                IotClock_SleepMs( 1000 );
                IotLogInfo( "State:%s  Received:%u Queued:%u Processed:%u Dropped:%u\r\n",
                        _pStateStr[ eState ],
                        OTA_GetPacketsReceived(), OTA_GetPacketsQueued(),
                        OTA_GetPacketsProcessed(), OTA_GetPacketsDropped() );
 }
```

OTA – 作業者の操作

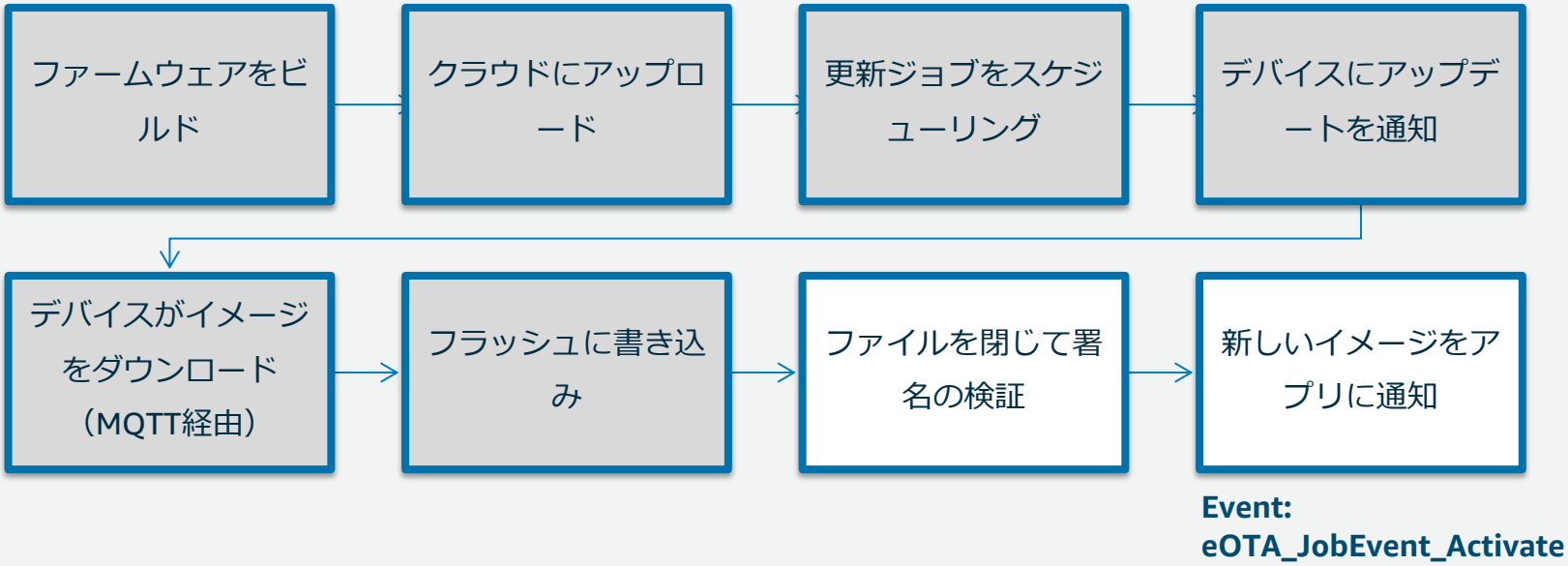
ファームウェアをビルド → クラウドにアップロード → 更新ジョブをスケジューリング → デバイスにアップデートを通知



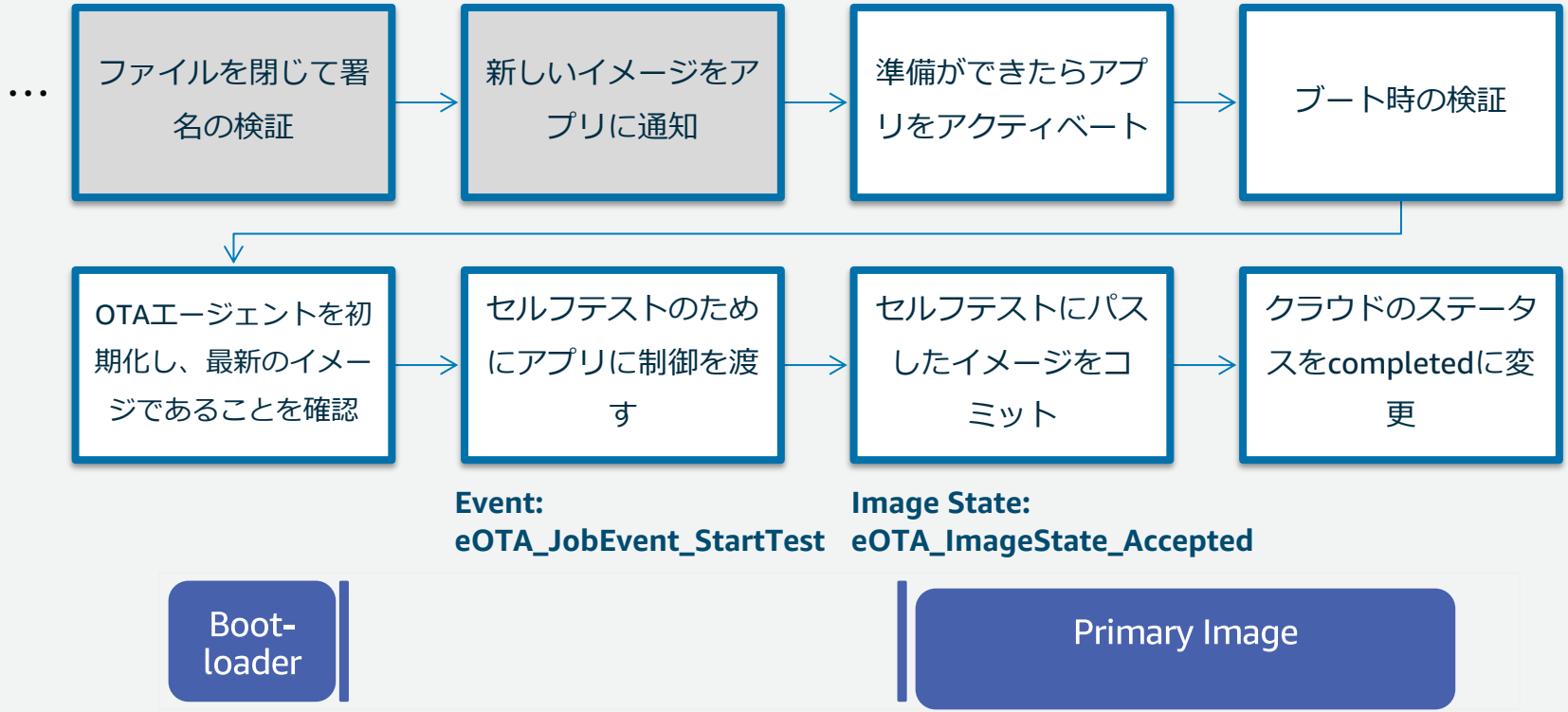
OTA – OTAエージェントのアクション



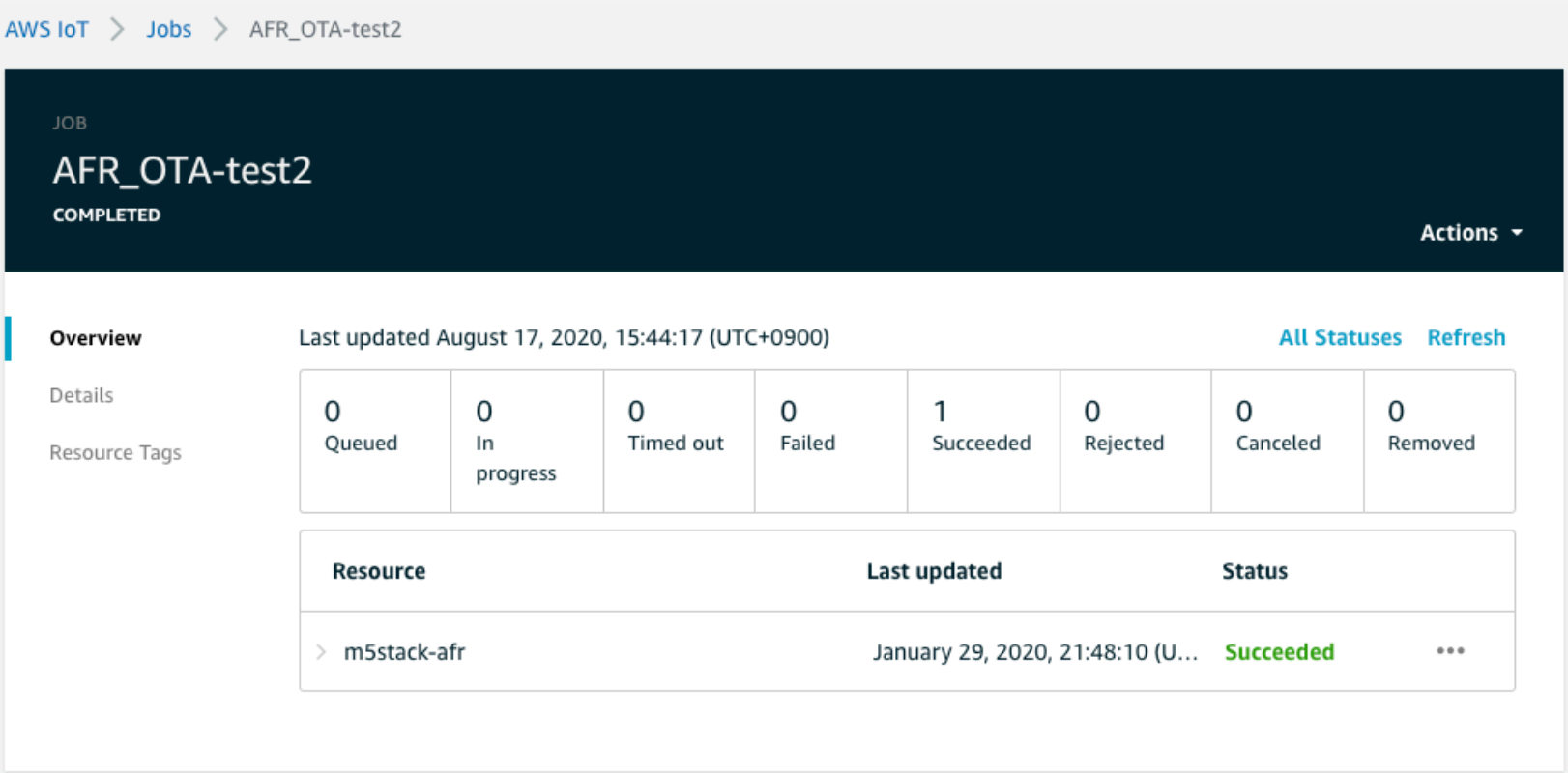
OTA – OTAエージェントのアクション



OTA – ブートローダーおよびアプリ のアクション



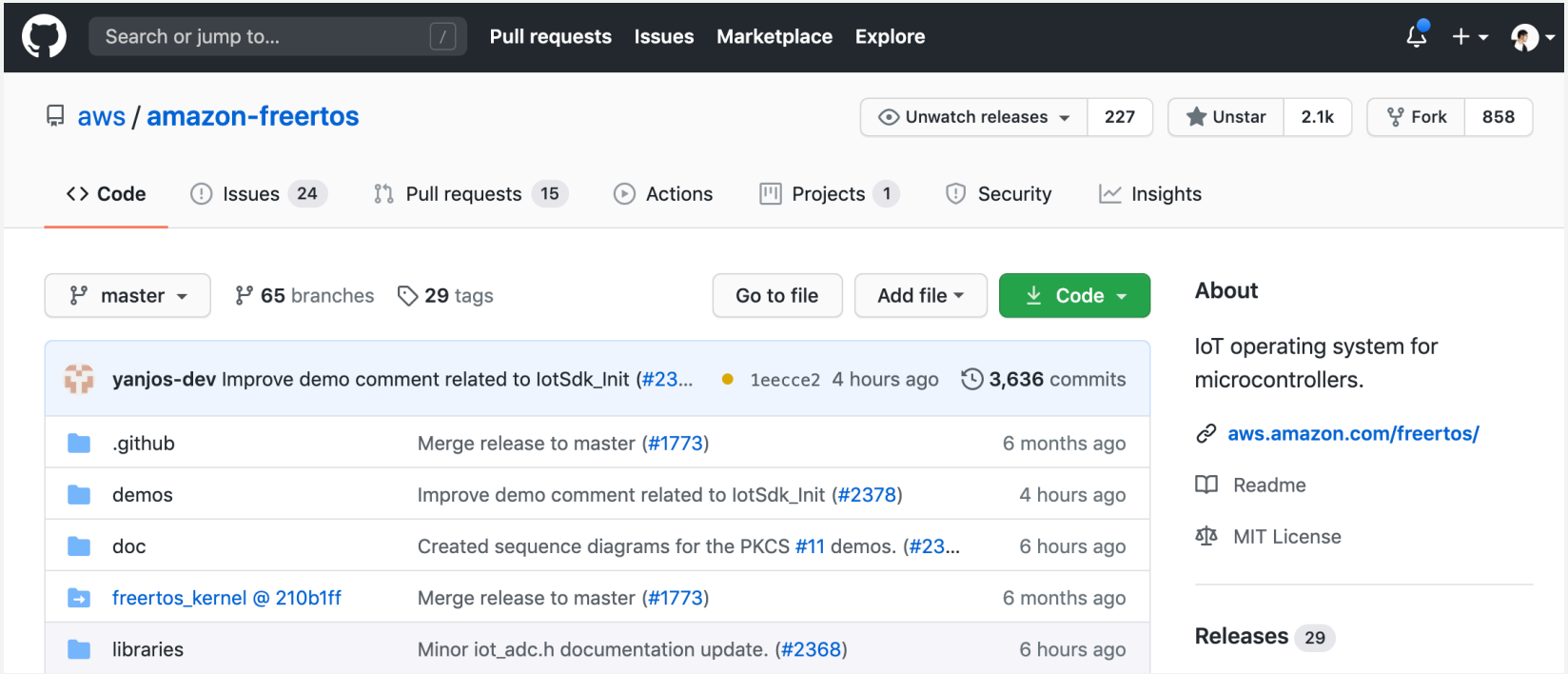
OTA – 更新ジョブのプログレス



アジェンダ

- 背景
- FreeRTOSとは
- FreeRTOS kernel
- FreeRTOS ライブラリ
- **開発方法**
- まとめ

FreeRTOSのソースコード



https://github.com/aws/amazon-freertos

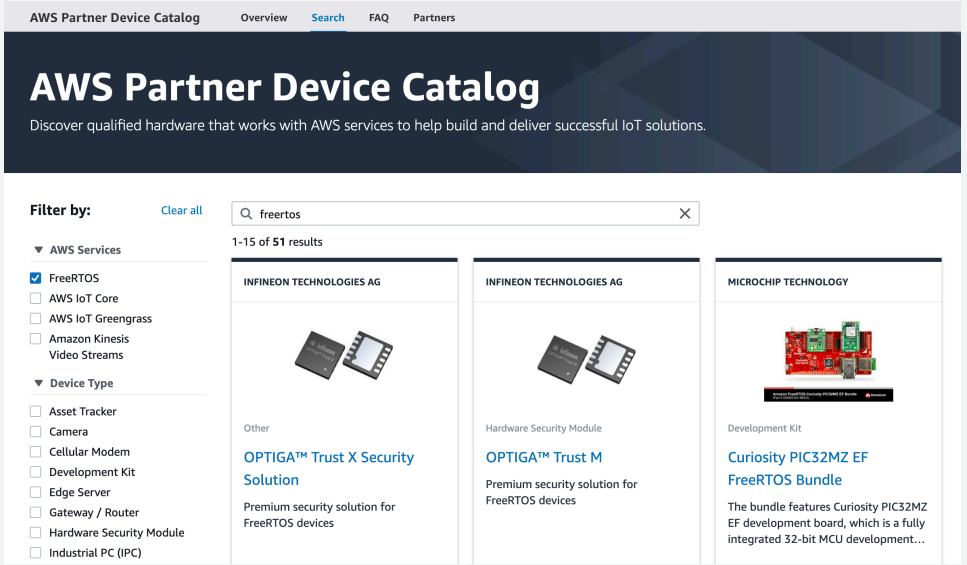
サブモジュールとして、freertos_kernelが登録されている

AWS Partner Device Catalog

- FreeRTOSに対応する製品の一覧

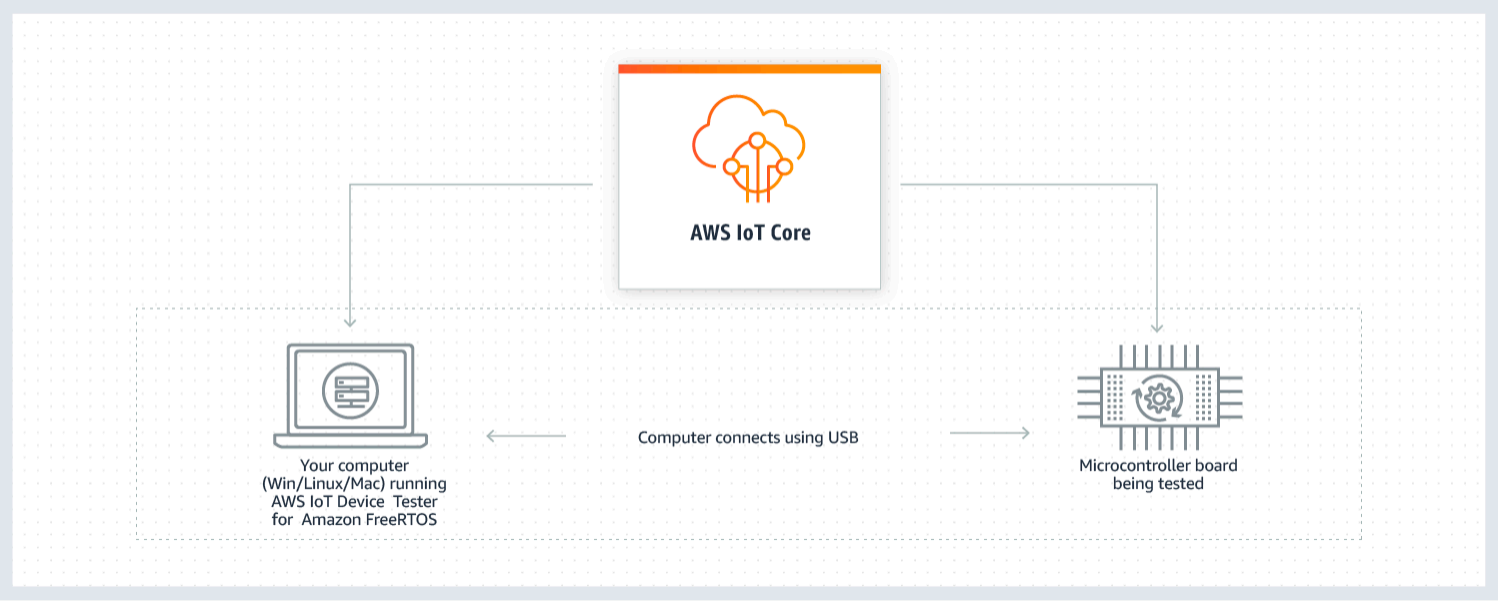
- 開発に必要なドキュメント
    - 使用開始手順
    - ソースコード
    - データシート など

https://devices.amazonaws.com/



Device Tester for FreeRTOS

FreeRTOSのクラウド接続、OTA、およびセキュリティライブラリが
正しく機能するかどうかをテスト



AWS デバイス認定プログラム

aws device qualification

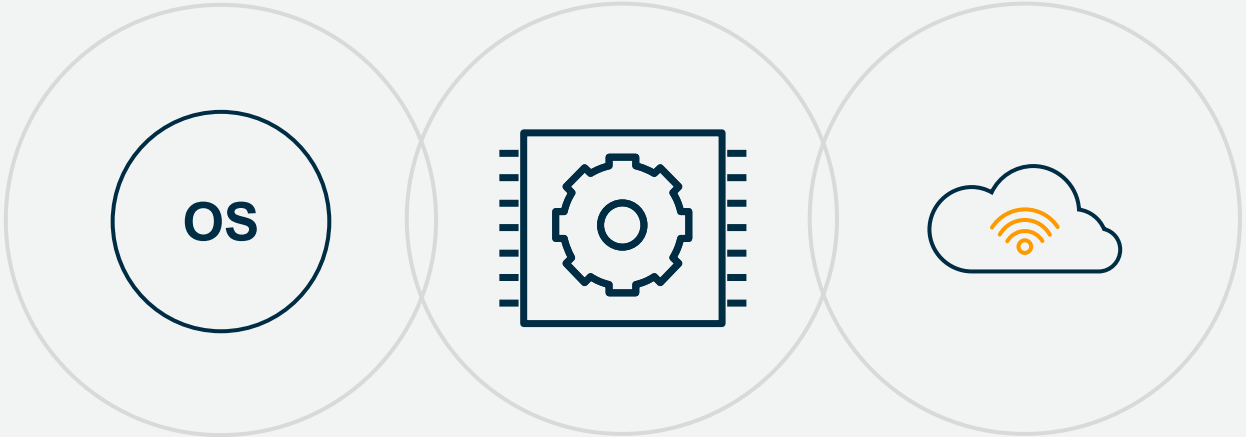
- AWS Device Qualification Program (DQP)
- APNパートナー向けのハードウェア認定およびインセンティブプログラム
- デバイスハードウェア上でFreeRTOSが動作することを検証することで認定が受けられる
- 認定デバイスは、AWS Partner Device Catalog に掲載される

https://aws.amazon.com/jp/partners/dqp/

aws

FreeRTOS以外でのAWS IoTとの接続方法

AWS IoT Embedded C SDKを利用することで、様々なOSを利用するマイクロコントローラやマイクロプロセッサを搭載するデバイスでもAWS IoTの機能を使えます



ご自身のOSを選択

使い慣れたOSを選択できます

FreeRTOSと同様なライブラリ

FreeRTOSで利用できるAWS IoTの機能が利用可能

個別に統合

個々のライブラリをプロジェクトに簡単に統合

アジェンダ

- 背景
- FreeRTOSとは
- FreeRTOS kernel
- FreeRTOS ライブラリ
- 開発方法
- **まとめ**

まとめ

- FreeRTOSを使うことで、デバイスのIoT化をクイックに進められる
- 製品化の際に必要な機能（セキュリティ、OTA、オフライン処理など）をクラウド側も含めて実現
- AWS Partner Device Catalogからデバイスを選び、すぐに開発を開始可能

参考資料

- FreeRTOS のより詳細を知りたい
    - **FreeRTOS**
        - https://aws.amazon.com/jp/freertos/
- 利用可能なデバイスを選びたい
    - **AWS Partner Device Catalog**
        - https://devices.amazonaws.com/
- AWS IoT のデバイス管理について試したい
    - **AWS IoT Device Managementハンズオン**
        - https://iot-device-management.workshop.aws/
- FreeRTOS を試したい
    - **amazon-freertos-m5stickc-workshop**
        - https://github.com/teuteuguy/amazon-freertos-m5stickc-workshop
- IoTの様々な事例やノウハウを知りたい
    - **IoT@Loft**
        - https://aws.amazon.com/jp/start-ups/loft/tokyo/iot-loft/

Q&A

お答えできなかったご質問については
AWS Japan Blog 「https://aws.amazon.com/jp/blogs/news/」にて
後日掲載します。

AWS の日本語資料の場所「AWS 資料」で検索

https://amzn.to/JPArchive

AWS Well-Architected 個別技術相談会

毎週”W-A個別技術相談会”を実施中

- AWSのソリューションアーキテクト(SA)に対策などを相談することも可能
- 申込みはイベント告知サイトから

(https://aws.amazon.com/jp/about-aws/events/)

AWS イベント で[検索]

ご視聴ありがとうございました

AWS 公式 Webinar
https://amzn.to/JPWebinar

過去資料
https://amzn.to/JPArchive