Smart Product Solution

Implementation Guide

Smart Product Solution: Implementation Guide

Table of Contents

Welcome 1
Overview 2
- Cost 2
- Architecture 2

Components 4
- CI/CD Pipeline 4
- AWS Cloud Development Kit 4
  - CDK Manifest File 4
- AWS IoT Core 5
  - Device Gateway 5
  - Rules Engine 5
  - AWS IoT Fleet Indexing 5
- Smart Product Solution Microservices 6
  - JITR Microservice 6
  - Event Proxy Microservice 6
  - Alert Microservice 6
  - Registration Microservice 6
  - Device Microservice 6
  - Admin Microservice 6
- Authentication 6
  - Just-in-Time Registration 7
- Device Management and Monitoring 7
- Analytics 7
- Amazon API Gateway 7
- Amazon DynamoDB 8
- Web Console 8

Considerations 9
- Amazon Cognito Limits 9
- Stopping Components and Stack Deletion 9
- Solution Updates 9
- Regional Deployments 9

Template 11
Deployment 12
- What We'll Cover 12
- Step 1. Launch the Stack 12
- Step 2. Demo the Solution 13
  - Create a User Account and Log in to the Web Console 13
  - Create and Configure a root CA Certificate 14
  - Create and Configure a Device Certificate 15
  - Create a Simulated Smart Product 16
  - Register the Simulated Smart Product 19
  - Configure Amazon QuickSight 19
  - Visualize Data in Amazon QuickSight 19

Security 21
- Authentication 21
- IAM Roles 21
- Amazon CloudFront 21

Resources 22
Appendix A: Solution API 23
- GET /settings/config/{settingId} 24
  - Description 24
  - Request Parameter 24
  - Response 24
- PUT /settings/config/{settingId} 24

Description 24
Request Parameter 24
GET /registration 25
Description 25
Response 25
POST /registration 25
Description 25
Request Body 25
Response 26
GET /devices 26
Description 26
Response 26
GET /devices/{deviceId} 27
Description 27
Request Parameter 27
Response 27
DELETE /devices/{deviceId} 27
Description 27
Request Parameter 27
Response 28
GET /devices/{deviceId}/commands 28
Description 28
Request Parameter 28
Response 28
POST /devices/{deviceId}/commands 29
Description 29
Request Parameter 29
Response 29
GET /devices/{deviceId}/commands/{commandId} 30
Description 30
Request Parameter 30
Response 30
GET /devices/events 31
Description 31
Response 31
GET /devices/alerts 32
Description 32
Response 32
GET /devices/alerts/count 33
Description 33
Response 33
GET /devices/{deviceId}/events 33
Description 33
Request Parameter 33
Response 33
GET /devices/{deviceId}/events/{eventId} 34
Description 34
Request Parameter 34
Response 35
PUT /devices/{deviceId}/events/{eventId} 35
Description 35
Request Parameter 35
GET /devices/{deviceId}/status 36
Description 36
Request Parameter 36
Response 36
Common Errors 36

Appendix B: Operational Metrics 37
Source Code 38
Revisions 39
..... 39

Leverage a framework for connected product services that provides secure connectivity to the AWS Cloud

Publication date: *September 2019 (last update (p. 39): February 2020)*

This implementation guide discusses architectural considerations and configuration steps for deploying the smart product solution in the Amazon Web Services (AWS) Cloud. It includes links to an AWS CloudFormation template that launches and configures the AWS services required to deploy this solution using AWS best practices for security and availability.

The guide is intended for IT infrastructure architects, administrators, and DevOps professionals who have practical experience with AWS IoT services and architecting in the AWS Cloud.

Overview

Amazon Web Services (AWS) enables manufacturers to develop smart products that collect, process, store, analyze, and act on product data, without having to manage any infrastructure.

With AWS, manufacturers can collect data in a variety of formats and frequencies from their connected devices around the world, enrich messages with external sources, store data for analysis, use machine learning for inference and to make predictions, and integrate with service management applications to improve customer satisfaction.

The Smart Product Solution provides secure product connectivity to the AWS Cloud, and includes capabilities for local computing within products, sophisticated event rules, and data processing and storage. The solution features fast and robust data ingestion; highly reliable and durable storage of product telemetry data; simple, scalable big data services for analyzing the data; and global messaging and application services.

This solution is designed to provide a framework for connected product services, allowing you to focus on extending the solution's functionality rather than managing the underlying infrastructure operations. You can build upon this framework to address a variety of use cases.

Cost

You are responsible for the cost of the AWS services used while running this reference deployment. The total cost for running this solution depends on the number of connected devices and the amount of data being loaded, requested, stored, processed, and presented. Prices are subject to change. For full details, see the pricing webpage for each AWS service you will be using in this solution.

Architecture Overview

Deploying this solution builds the following environment in the AWS Cloud.

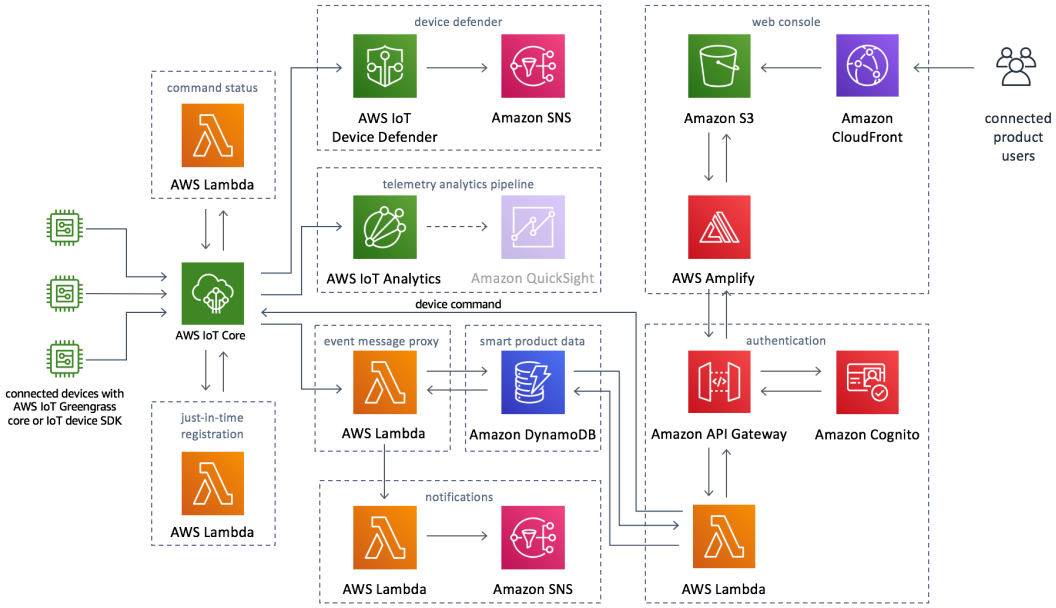

Figure 1: Smart product solution architecture

The smart product solution leverages AWS IoT services to ingest and process messages from connected products according to business rules. When you launch the solution, a continuous integration/continuous delivery (CI/CD) pipeline is deployed that uses the AWS Cloud Development Kit (AWS CDK) and AWS CloudFormation to deploy the solution's architecture. For more information on the pipeline, see CI/CD Pipeline (p. 4).

The solution's CI/CD pipeline deploys AWS IoT Core, which authenticates messages from smart products and routes those messages to the solution's microservices (AWS Lambda functions); AWS IoT Device Defender to continuously audit your devices to ensure they don't deviate from security best practices; and AWS IoT Analytics to analyze data from your smart products.

The template also deploys Amazon DynamoDB tables that store various details about the smart products; Lambda functions that provide the business logic to perform operations and collect data on smart products; and Amazon Simple Notification Service (Amazon SNS) to publish messages from your smart products and deliver those messages to subscribers and other applications.

The solution creates a web console powered by AWS Amplify, and deploys it into an Amazon Simple Storage Service (Amazon S3) bucket configured for web hosting. Amazon CloudFront is used to provide public access to the bucket.

The solution also configures Amazon API Gateway to host the solution's RESTful APIs, and deploys an Amazon Cognito user pool, which you can use to add user registration and sign-in for included web console.

Solution Components

CI/CD Pipeline

This solution leverages a continuous integration/continuous delivery (CI/CD) pipeline to deploy the solution's resources. The CI/CD pipeline contains the following resources in addition to the main solution resources.

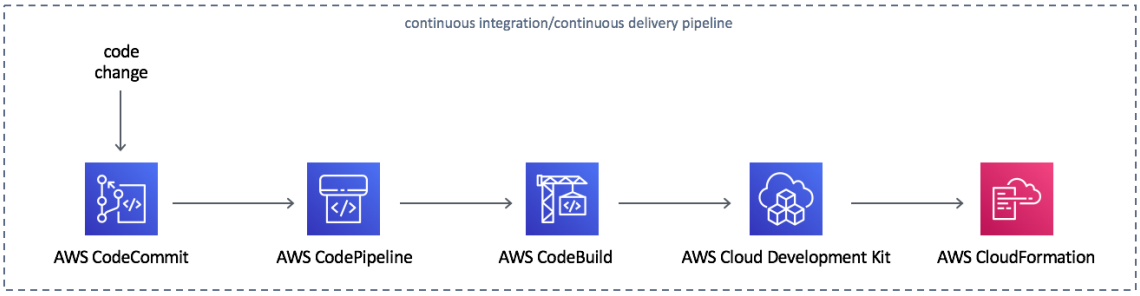

Figure 2: Smart product CI/CD pipeline architecture

The CI/CD pipeline consists of an AWS CodeCommit repository that contains the solution source code; AWS CodeBuild to test the source code, and build and stage solution resources; and the AWS Cloud Development Kit (AWS CDK) and AWS CloudFormation to deploy the resources.

AWS Cloud Development Kit

The AWS Cloud Development Kit (AWS CDK) is a software development framework that enables you to define cloud infrastructure in your preferred programming language and deploy it with AWS CloudFormation. This solution uses the TypeScript CDK.

CDK Manifest File

The solution includes a CDK manifest file (`cdk-manifest.json`) you can use to specify which resources and features are launched.

```
{
  "default": {
    "version": "v0.0.1",
    "telemetry": {
      "deploy": true,
      "env": {
        "telemetryTopic": "smartproduct/telemetry"
      }
    },
    "events": {
      "deploy": true,
      "env": {
        "eventTopic": "smartproduct/event"
      }
    },
```

```
    "jitr": {
      "deploy": true
    },
    "api": {
      "deploy": true
    },
    "ownerapp": {
      "deploy": true
      }
    }
  }
```

Figure 3: Example CDK manifest file

You can choose what solution features are launched by updating the manifest file and committing it to the CodeCommit repository.

When you commit an updated manifest file, the solution's CI/CD pipeline builds the CDK stages, generates a template, runs cfn_nag (an open source linting tool for AWS CloudFormation) on the template, and updates the resources based on the changes in the file.

AWS IoT Core

The smart product solution leverages AWS IoT Core to provide smart product connectivity to the AWS Cloud. AWS IoT Core provides authentication and end-to-end encryption throughout all points of the connection between your products and the AWS Cloud, ensuring that data is never exchanged without proven identity and validation of granular permissions.

AWS IoT Core also stores the latest state of your smart products so that those states can be read or set at any time.

Device Gateway

The AWS IoT Device Gateway enables devices to securely and efficiently communicate with AWS IoT Core. For this solution, smart products communicate with the Device Gateway using a publication/subscription model. In the pub/sub model, devices publish messages to specific logical communication channels called *topics*. Devices subscribe to the topics to receive messages.

Rules Engine

When a smart product publishes a message to the smart product solution, the AWS IoT Rules Engine evaluates, transforms, and delivers the message to the appropriate backend services based on defined rules.

The Rules Engine can be configured to route inbound telemetry data from AWS IoT to several other AWS services such as Amazon Kinesis Data Streams or Amazon DynamoDB, or from one AWS IoT Core topic to another. Data can also be sent to custom applications running on AWS Lambda, giving manufacturers maximum flexibility and power to process smart product data.

AWS IoT Fleet Indexing

The smart product solution uses Fleet Indexing to index and search your registry data, shadow data, and device connectivity data (device lifecycle events) in the cloud. After you set up your fleet index, the service manages the indexing of updates for your thing groups, thing registries, and device shadows. For more information, see Managing Thing Indexing in the *AWS IoT Developer Guide*.

By default, when you delete the solution stack or disable the solution's API, the solution disables fleet indexing. If you would like to keep indexing enabled after you delete the stack or disable the API, you must modify the solution's custom resource before you delete the stack or disable the API.

Smart Product Solution Microservices

The smart product solution microservices are a series of AWS Lambda functions that provide the business logic and data access layer for all device operations. Each Lambda function assumes an AWS Identity and Access Management (IAM) role with least privilege access (minimum permissions necessary) to perform its designated functions.

JITR Microservice

The just-in-time-registration (JITR) microservice handles activating device certificates for devices with unregistered certificates. When a device with an unregistered device certificate connects with AWS IoT Core, this microservice creates a policy for the certificate, attaches the policy to the certificate, activates the certificate, and attaches the certificate to the device-specific IoT thing.

Event Proxy Microservice

The event proxy microservice processes events. When the Rules Engine sends a message to the `/smartproduct/events` AWS IoT Core topic, the event proxy microservice processes the message (event), indexes the event in an Amazon DynamoDB table, and invokes the alert microservice.

Alert Microservice

The alert microservice handles notifying end users of alerts. When the event proxy microservice invokes the alert microservice, the alert microservice displays the alert on the solution's web console and sends the alert to an Amazon Simple Notification Service (Amazon SNS) topic.

Registration Microservice

The registration microservice handles device registration. When you register a device, this microservice indexes the registration data in a DynamoDB table.

Device Microservice

The device microservice handles retrieving smart products and smart product information. This microservice also handles deleting devices from AWS IoT Core and updating the registration DynamoDB table with the deletion.

Admin Microservice

The admin microservice handles all administrative services, including users and general settings.

Authentication

This solution takes advantage of mutual authentication and encryption at all points of connection to AWS IoT to ensure that data is never exchanged between the smart product and AWS IoT without proven identity.

This solution uses an X.509 certificate. You can use your own X.509 certificate, or AWS IoT can generate a certificate for you using the AWS Management Console or the AWS CLI. To use your own certificate, see Use Your Own Certificate. To use AWS IoT to generate a certificate, see Create and Register an AWS IoT Device Certificate in the *AWS IoT Developer Guide*.

Store your certificate, the private key, and the public key in a secure location on your local machine.

Just-in-Time Registration

When a smart product connects to AWS IoT Core for the first time, AWS IoT detects the unknown certificate. If the certificate is signed by a registered CA, the smart product solution attempts to register the product certificate automatically during the Transport Layer Security (TLS) handshake.

An MQTT registration event is published on a registration topic associated with the registered CA certificate. The registration event invokes an AWS Lambda function that activates the certificate and attaches a policy to it. When the certificate is activated and the policy is attached, the certificate can be used for authentication and authorization with AWS IoT Core.

Device Management and Monitoring

This solution uses AWS IoT Device Management to track, monitor, and remotely manage your fleet of smart products. With AWS IoT Device Management, the solution organizes your smart products, monitors and enables you to troubleshoot functionality issues, and query the state of any smart product in your fleet.

If enabled, the solution also deploys AWS IoT Device Defender to continuously audit the configurations of your fleet of smart products to ensure that they don't deviate from security best practices. If a smart product deviates from security best practices, AWS IoT Device Defender alerts you about the issue and recommends mitigation actions.

> **Important**
> If you are running AWS IoT Device Defender in your account before you launch this solution, we do not recommend launching the AWS IoT Device Defender component of this solution. If you were running AWS IoT Device Defender before you launched the solution, and you delete the solution stack or update the stack to change the **Defender** AWS CloudFormation template parameter from `true` to `false`, the solution will stop the AWS IoT Device Defender service, even if it was not deployed with the solution.

Analytics

AWS IoT Analytics automates the process required to analyze data from smart products. It filters, transforms, and enriches the data this solution collects before storing the data in a time-series data store for analysis. You can apply mathematical transforms to process the data, and enrich the data with device-specific metadata such as device type and location before storing it. Then, you can analyze your data by running queries, or perform more complex analytics and machine learning inference.

You can also leverage Amazon QuickSight to visualize the data that this solution collects.

Amazon API Gateway

Users and applications can interact with smart product data securely through the solution's web console and RESTful APIs hosted in Amazon API Gateway. The APIs act as a "front door" for access to

smart product data stored in Amazon DynamoDB. You can also use the APIs to access any extended functionality you build into the solution.

This solution takes advantage of the user authentication features of Amazon Cognito User Pools. After successfully authenticating a user, Amazon Cognito issues a JSON web token that is used to allow the console to submit requests to the solution's APIs (Amazon API Gateway endpoints). HTTPS requests are sent by the console to the APIs with the authorization header that includes the token.

Based on the request, Amazon API Gateway invokes the appropriate Lambda function to perform the necessary tasks on the data stored in the DynamoDB tables. You can use this data, as well as near-real-time MQTT data, to build detailed graphs, charts, and reports.

Amazon DynamoDB

The smart product solution uses Amazon DynamoDB to persist metadata for smart products, settings, and metrics. Each DynamoDB table is provisioned using DynamoDB on-demand.

The solution creates the following tables.

- **Settings:** Contains solution settings including Amazon Cognito user pools IDs and users' alert levels
- **Registrations:** Contains registration information about smart products
- **Events:** Contains information about events
- **Commands:** Contains information about commands
- **Reference:** Contains information about smart products including serial number and model number. This table is used to verify that the smart product is valid during registration.

Web Console

The smart product solution includes a simple web console you can use to interact with your smart products. You can use the console to register users and devices, connect to your smart products, receive events and alerts, and send remote commands to your devices.

The console is designed to demonstrate how you can interact with your smart products. In a production environment, we recommend customizing this web console to meet your needs or building your own console for your use case.

Considerations

Amazon Cognito Limits

This solution uses Amazon Cognito User Pools to manage users. Amazon Cognito sends an email every time you create a user, change a password, or reset a password. Amazon Cognito limits the number of emails sent daily per user pool to 50. For customers who plan to use this solution for a large number of users, we recommend using Amazon Simple Email Service (Amazon SES) for these emails. For more information, see Authorizing Amazon Cognito to Send Amazon SES Email on Your Behalf in the *Amazon Cognito Developer Guide*.

Stopping Components and Stack Deletion

If you want to stop specific solution components, you can update the solution's AWS CloudFormation template parameters and then update the stack. For example, if you want to stop the AWS IoT Device Defender component, set the **Defender** AWS CloudFormation template parameter to `false`. Then, update the stack.

> **Important**
> If you update the stack to stop AWS IoT Device Defender, the solution will stop the AWS IoT Device Defender service, even if it was not deployed with the solution. For more information, see Device Management and Monitoring (p. 7).

When you delete the entire smart product solution stack, the AWS CodeCommit repository and Amazon S3 bucket will not be deleted to ensure your customizations persist. You must delete the repository and the bucket manually.

To ensure all components are deleted cleanly, we recommend stopping all components before you delete the stack by setting all the template parameters to `false` and updating the stack. Then, delete the stack.

Solution Updates

The smart product solution version 1.0.1 uses the most up-to-date Node.js runtime. Version 1.0 uses the Node.js 8.10 runtime, which reaches end-of-life on December 31, 2019. In January, AWS Lambda will block the create operation and, in February, Lambda will block the update operation. For more information, see Runtime Support Policy in the *AWS Lambda Developer Guide*.

To continue using this solution with the latest features and improvements, you must deploy version 1.0.1 as a new stack.

Regional Deployments

This solution uses AWS IoT Analytics and SMS messaging for Amazon Simple Notification Service (Amazon SNS), which are currently available in specific AWS Regions only. Therefore, you must launch this solution in a region where these services are available. For the most current service availability

by region, see AWS service offerings by region. For a list of supported regions for SMS messaging for Amazon SNS, see Supported Regions and Countries in the *Amazon SNS Developer Guide*.

AWS CloudFormation Template

This solution uses AWS CloudFormation to automate the deployment of the smart product solution in the AWS Cloud. It includes the following AWS CloudFormation template, which you can download before deployment:

smart-product-solution.template: Use this template to launch the smart product solution and all associated components. The default configuration deploys AWS CodePipeline, AWS CodeCommit, AWS CodeBuild, the AWS Cloud Development Kit (AWS CDK), AWS IoT Core, AWS IoT Device Defender, AWS IoT Analytics, AWS Lambda functions, Amazon DynamoDB tables, Amazon Simple Notification Service (Amazon SNS) topics, an Amazon Simple Storage Service (Amazon S3) bucket, Amazon CloudFront, AWS Amplify, Amazon API Gateway, and an Amazon Cognito user pool. You can also customize the template based on your specific network needs.

Automated Deployment

Before you launch the automated deployment, please review the architecture, configuration, and other considerations discussed in this guide. Follow the step-by-step instructions in this section to configure and deploy the smart product solution into your account.

Time to deploy: Approximately three minutes

What We'll Cover

The procedure for deploying this architecture on AWS consists of the following steps. For detailed instructions, follow the links for each step.

Step 1. Launch the Stack (p. 12)

- Launch the AWS CloudFormation template into your AWS account.
- Enter values for required parameter: **Stack Name**, **CodeCommit Repository Name**, **Solution Version**
- Review the other template parameters, and adjust if necessary.

Step 2. Demo the Solution (p. 13)

- Create a user account and log in to the web console.
- Configure and register a device.
- Configure Amazon QuickSight.

Step 1. Launch the Stack

This automated AWS CloudFormation template deploys the smart product solution in the AWS Cloud.

Note
You are responsible for the cost of the AWS services used while running this solution. See the Cost (p. 2) section for more details. For full details, see the pricing webpage for each AWS service you will be using in this solution.

1. Sign in to the AWS Management Console and click the button below to launch the `smart-product-solution` AWS CloudFormation template.

   Launch Solution

   You can also download the template as a starting point for your own implementation.
2. The template is launched in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the region selector in the console navigation bar.

   Note
   This solution uses AWS IoT Analytics which is available in specific AWS Regions only. Therefore, you must launch this solution an AWS Region where this service is available. For the most current service availability by region, see AWS service offerings by region.

3. On the **Create stack page**, verify that the correct template URL shows in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, assign a name to your solution stack.
5. Under **Parameters**, review the parameters for the template and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
CodeCommit Repository Name	`smart-product-reference-architecture`	The name of the solution-created repository for the solution's source code
API	`true`	Choose whether to deploy the solution's API component. For more information on the API, see Appendix A (p. 23).
Defender	`true`	Choose whether to deploy the solution's AWS IoT Device Defender component
Event	`true`	Choose whether to deploy the solution's event component
JITR	`true`	Choose whether to deploy the solution's just-in-time-registration component
Web Console	`true`	Choose whether to deploy the solution's web console
Telemetry	`true`	Choose whether to deploy the solution's telemetry analytics pipeline component
Version	`v0.0.1`	The version number of the solution deployment

6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review and confirm the settings. Be sure to check the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.
9. Choose **Create stack** to deploy the stack.

   You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should see a status of **CREATE_COMPLETE** in approximately three minutes.

Step 2. Demo the Solution

Use this procedure to demonstrate the solution's capabilities using a sample smart product.

Create a User Account and Log in to the Web Console

1. In the AWS CloudFormation console stack **Outputs** tab, select the URL value of the **Smart Product Owner Web App** key.

2. In the web console, select **Create Account**.
3. In the **Create Account** window, enter the applicable information.
4. Select **Create Account**.

   A verification email will be sent to the specified email address.
5. Select the link in the verification email to verify your account.
6. Enter your username and password to log in to the web console.

Create and Configure a root CA Certificate

Note
To configure a device certificate, you must have the latest version of the AWS Command Line Interface (AWS CLI) and OpenSSL installed. If you do not have the AWS CLI installed, see Installing the AWS CLI in the *AWS Command Line Interface User Guide*.

1. In a terminal window, run the following command to create the sample root certificate authority (CA) certificate.

```
openssl genrsa -out sampleCACertificate.key 2048
openssl req -x509 -new -nodes -key sampleCACertificate.key -sha256 -days 365 -out
 sampleCACertificate.pem
```

   Note
   For this demo, you will create and register the root CA certificate. In a production environment, the root CA would sign the device certificate and you would register that certificate with AWS IoT.
2. To register the root CA certificate, run the following command in the AWS CLI.

```
aws iot get-registration-code
```

   This command returns a randomly generated, unique registration code for your AWS account. This code does not expire until you delete it.
3. To create a certificate signing request (CSR), run the following command.

```
openssl genrsa -out privateKeyVerification.key 2048
openssl req -new -key privateKeyVerification.key -out privateKeyVerification.csr
```

4. When prompted, enter the registration code in the `Common Name` field of the verification certificate and press **Enter**.

```
...
Organization Name (eg, company) []:
Organizational Unit Name (eg, section)
Common Name (e.g. server FQDN or YOUR name) []:
<your-registration-code>
EMAIL ADDRESS []:
```

5. Create a new CA certificate using the following command.

```
openssl x509 -req -in privateKeyVerification.csr -CA sampleCACertificate.pem -CAkey
 sampleCACertificate.key -CAcreateserial -out privateKeyVerification.crt -days 365 -
sha256
```

6. To use the verification certificate to register the CA certificate, run the following command.

```
aws iot register-ca-certificate --ca-certificate file://sampleCACertificate.pem --
verification-certificate file://privateKeyVerification.crt
```

This command returns a certificate ID you will use in the next step.

7. Run the following command. Replace *<your-certificate-ID>* with the applicable value.

```
aws iot describe-ca-certificate --certificate-id <your-certificate-ID>
```

By default, the CA certificate will be registered in an inactive state. You must activate the CA certificate to register the device certificate.

8. To activate the CA certificate, run the following command.

```
aws iot update-ca-certificate --certificate-id <your-certificate-ID> --new-status
 ACTIVE
```

By default, the `auto-registration-status` of the registered CA certificate is disabled, which means device certificates issued by the CA will not be automatically registered. You must enable automatic registration.

9. To enable automatic registration, run the following command.

```
aws iot update-ca-certificate --certificate-id <your-certificate-ID> --new-auto-
registration-status ENABLE
```

Now, when the solution detects an unknown certificate signed by a registered CA, the solution will automatically register the device certificate.

When the certificate is successfully registered, AWS IoT publishes a message to an MQTT topic and disconnects the client. When the message is a published to the topic, an AWS Lambda function is triggered that completes the provisioning of the certificate.

Create and Configure a Device Certificate

1. To create a device certificate, run the following command.

```
openssl genrsa -out deviceCert.key 2048
openssl req -new -key deviceCert.key -out deviceCert.csr
...
Common Name (e.g. server FQDN or Your name) []: <iot-thing-name>
openssl x509 -req -in deviceCert.csr -CA sampleCACertificate.pem -CAkey
 sampleCACertificate.key -CAcreateserial -out deviceCert.crt -days 365 -sha256
```

2. Create a certificate file that contains the device certificate and the registered CA certificate using the following command.

```
cat deviceCert.crt sampleCACertificate.pem > deviceCertAndCACert.crt
```

3. To use the MQTT Mosquitto client to connect to AWS IoT using the device certificate, use the following command.

```
IOT_ENDPOINT='aws iot describe-endpoint --endpoint-type iot:Data-ATS --region <your-
region>'
curl -o root.cert https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

```
mosquitto_pub --cafile root.cert --cert deviceCertAndCACert.crt --key deviceCert.key -h
 $IOT_ENDPOINT -p 8883 -q 1 -t foo/bar -i anyclientID --tls-version tlsv1.2 -m "Hello"
 -d
```

For more information on just-in-time registration of device certificates, see Just-in-Time Registration of Device Certificates on AWS IoT.

Create a Simulated Smart Product

To demonstrate the just-in-time-registration functionality, you can create an Amazon Elastic Compute Cloud (Amazon EC2) instance to act a simulated smart product, or you can demo the solution using a Raspberry Pi device.

Note
Before you create a simulated smart product, you must wait for the solution stack launch to complete.

To use an Amazon EC2 instance, see Simulate a Device with Amazon EC2 (p. 16). To use a Raspberry Pi, see Set up a Raspberry Pi Device (p. 17).

Simulate a Device with Amazon EC2

Use this procedure to create an Amazon EC2 instance and AWS Identity and Access Management (IAM) role to test the solution. If you already have an EC2 administration IAM role, skip to step 15 (p. 17).

1. Navigate to the IAM console.
2. In the navigation pane, select **Roles**.
3. Select **Create Role**.
4. Select **EC2**. Then, select **Next: Permissions**.
5. Select **Next: Tags**. Then, select **Next: Review**.
6. For **Role name**, enter `SmartProductDemoRole`.
7. For **Role description**, enter `Role for Smart Product demo and test`.
8. Select **Create Role**.
9. On the **Roles** page, search for `SmartProductDemoRole`.
10. Select **+ Add inline policy**.
11. Select the **JSON** tab and replace the existing policy with the following policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "iot:GetRegistrationCode",
                "iot:RegisterCaCertificate",
                "iot:DescribeCaCertificate",
                "iot:UpdateCaCertificate",
                "iot:DescribeEndpoint",
                "dynamodb:BatchWriteItem",
                "dynamodb:ListTables"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

12. Select **Review policy**.
13. For **Name**, enter `SmartProductDemoPolicy`.
14. Select **Create policy**.
15. Navigate to the Amazon EC2 console.
16. Select **Launch instance**.
17. Choose **Select** next to an Amazon Linux or Ubuntu AMI.
18. Select the **radio button** for a t2.micro instance. You can choose any instance type but t2.micro is sufficient for the demo.
19. Select **Next: Configure Instance Details**.
20. In the **IAM role** dropdown menu, select **SmartProductDemoRole**.
21. Select the arrow next to **Advanced Details**.
22. Paste the code into the **User data** text box.

    - If you chose an Amazon Linux AMI, paste the following code in the box.

    ```
    #!/bin/bash
    curl -O https://dzsw62jm0sdrr.cloudfront.net/userdata-amazon-linux.sh
    bash userdata-amazon-linux.sh
    ```

    - If you chose an Ubuntu AMI, paste the following code in the box.

    ```
    #!/bin/bash
    curl -O https://dzsw62jm0sdrr.cloudfront.net/userdata-ubuntu.sh
    bash userdata-ubuntu.sh
    ```

23. Optional: Add storage, tags, security groups, and your key pair.
24. Select **Review and Launch**.
25. Select **Launch**.

Set up a Raspberry Pi Device

To demo the solution using a Raspberry Pi, you must be able to SSH into the device. After you SSH into your device, use this procedure to configure your device.

1. Navigate to the IAM console.
2. In the navigation pane, select **Policies**.
3. Select **Create Policy**.
4. Select the **JSON** tab and replace the existing policy with the following policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "iot:GetRegistrationCode",
                "iot:RegisterCaCertificate",
                "iot:DescribeCaCertificate",
                "iot:UpdateCaCertificate",
                "iot:DescribeEndpoint",
                "dynamodb:BatchWriteItem",
                "dynamodb:ListTables"
            ],
            "Resource": "*",
            "Effect": "Allow"
```

```
        }
    ]
}
```

5. Select **Review policy**.
6. For **Name**, enter `SmartProductDemoPolicy`.
7. For **description**, enter `Smart Product demo policy`.
8. Select **Create policy**.
9. In the navigation pane, select **Users**.
10. Select **Add User**.
11. For **User name**, enter `SmartProductDemoUser` and select the check box next to **Programmatic Access**.
12. Select **Next: Permissions**.
13. Select **Attach existing policies directly** and search for `SmartProductDemoPolicy`.
14. Select the check box next to **SmartProductDemoPolicy** and choose **Next: Tags**.
15. Select **Next: Review**. Then, select **Create user**.
16. Select **Download .csv** or **Show** to see your access key and your secret access key. You will use the access key and secret access key later.
17. If you don't have the AWS CLI installed, install it using the instructions in the *AWS Command Line Interface User Guide*.
18. To provide the Raspberry Pi with your AWS credentials, use the following command. Replace *<your-access-key>*, *<your-secret-key>*, and *<your-region>* with your information. For *<your-region>*, enter the AWS Region where you deployed the solution.

```
aws configure
AWS Access Key ID [None]: <your-access-key>
AWS Secret Access Key [None]: <your-secret-key>
Default region name [None]: <your-region>
Default output format [None]: JSON
```

19. If you don't have Node.js installed on your Raspberry Pi, install it. For instructions, see Install Node.js and Npm on Raspberry Pi.
20. Install Mosquitto Broker on your Raspberry Pi. For instructions, see How to Install Mosquitto Broker on Raspberry Pi.
21. To download and install the file necessary to simulate a smart product, run the following commands. Replace *<your-region>* with the AWS Region where you deployed the solution.

```
mkdir reference
cd reference
curl -O https://dzsw62jm0sdrr.cloudfront.net/reference.tar.gz
tar xvfz reference.tar.gz
rm reference.tar.gz
npm install
export REGION='<your-region>'
export TABLE=`aws dynamodb list-tables --region $REGION --query TableNames[*] | grep
 Reference | tr -d ' ' | sed -e 's/"//g' -e 's/,//'`
export HOST=`aws iot describe-endpoint --endpoint-type iot:Data-ATS --region $REGION --
output text`
npm start
cd ..
mkdir simulator
cd simulator
curl -O https://dzsw62jm0sdrr.cloudfront.net/simulator.tar.gz
tar xvfz simulator.tar.gz
rm simulator.tar.gz
```

```
npm install
```

Register the Simulated Smart Product

After you launch the Amazon EC2 instance or configure your Raspberry Pi, you must complete the registration using the web console.

1. In the web console, select **Register a device**.
2. Enter a serial number, device name, and model number.

   Note
   You can find the serial number and model number in the `Reference` Amazon DynamoDB table.
3. Select **Register**.

After you complete the registration, run the following commands on your instance or device. Replace *<your-device-serial-number>* with the serial number your registered using the web console.

```
cp deviceCert.key deviceCertAndCACert.crt root.cert ~/simulator/certs
cd ~/simulator
sed -i -e 's/REPLACE_HERE/<your-device-serial-number>/g' config.js
npm start
```

When your instance or device successfully connects to AWS IoT, you will start to see simulated logs. You can modify the `config.js` file to change your demo simulator configuration. For example, you can modify the interval at which simulated data is sent.

Configure Amazon QuickSight

Use this procedure to AWS IoT Analytics with Amazon QuickSight to visualize the data this solution collects.

Note
Before you start, your account must be signed up for Amazon QuickSight. For more information, see Setting Up Amazon QuickSight.

1. Navigate to the Amazon QuickSight console.
2. Select your user name on the application and then choose **Manage QuickSight**.
3. Choose **Account settings**.
4. Under **Account permissions**, choose **Edit AWS Permissions**.
5. Under **Connected products & services**, select **Add or remove**.
6. On the **Edit QuickSight read-only access to AWS resources** page, select **AWS IoT Analytics**.
7. Select **Apply**.

Visualize Data in Amazon QuickSight

1. In the Amazon QuickSight console, select **New analysis**.
2. Select **New data set**.
3. Select **AWS IoT Analytics**.
4. Choose **smartproduct_dataset** and **Create data source**.
5. Select **Visualize**.

Now, you can create a dashboard that visualizes the AWS IoT Analytics data. For more information, see Working with Data in Amazon QuickSight and Working with Analyses in the *Amazon QuickSight User Guide*.

If you experience issues while using Amazon QuickSight, see Troubleshooting Amazon QuickSight.

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared model can reduce your operational burden as AWS operates, manages, and controls the components from the host operating system and virtualization layer down to the physical security of the facilities in which the services operate. For more information about security on AWS, visit the AWS Security Center.

Authentication

AWS IoT provides mutual authentication and encryption at all points between the smart product and the AWS IoT Core so that data is never exchanged without proven identity. AWS IoT supports Signature Version 4 and X.509 certificate based authentication. With AWS IoT, you can use AWS IoT generated certificates as well as those signed by your preferred Certificate Authority (CA).

IAM Roles

AWS Identity and Access Management (IAM) roles enable customers to assign granular access policies and permissions to services and users on the AWS Cloud. This solution creates several IAM roles, including roles that grant the solution's AWS Lambda functions access the other AWS services used in this solution. These roles are necessary to allow the services to collect, process, and store smart product data in your account.

Amazon CloudFront

This solution deploys a web console hosted in an Amazon Simple Storage Service (Amazon S3) bucket. To help reduce latency and improve security, this solution includes an Amazon CloudFront distribution with an origin access identity, which is a special CloudFront user that helps provide public access to the solution's website bucket contents. For more information, see Restricting Access to Amazon S3 Content by Using an Origin Access Identity.

Additional Resources

AWS services

- AWS IoT Core
- AWS IoT Device Management
- AWS IoT Device Defender
- AWS IoT Analytics
- AWS Lambda
- Amazon API Gateway
- Amazon DynamoDB
- Amazon Simple Notification Service
- Amazon Simple Storage Service
- AWS Amplify
- Amazon Cognito
- AWS CloudFormation
- Amazon CloudFront
- Amazon Simple Email Service
- Amazon QuickSight

Appendix A: Smart Product Solution API

The smart product solution API enables you to securely expose collected data. The API acts as a “front door” for applications to access product data, business logic, and extended functionality from the smart product backend microservices.

This solution uses an Amazon Cognito user pool integrated with Amazon API Gateway for identification and authorization. When a user pool is used with the API, clients are only allowed to call user pool enabled methods after they provide a valid identity token.

The following operations are available in the smart product solution API.

Admin

- GET /settings/config/{settingId} (p. 24)
- PUT /settings/config/{settingId} (p. 24)

Registration

- GET /registration (p. 25)
- POST /registration (p. 25)

Devices

- GET /devices (p. 26)
- GET /devices/{deviceId} (p. 27)
- DELETE /devices/{deviceId} (p. 27)

Commands

- GET /devices/{deviceId}/commands (p. 28)
- POST /devices/{deviceId}/commands (p. 29)
- GET /devices/{deviceId}/commands/{commandId} (p. 30)

Events

- GET /devices/events (p. 31)
- GET /devices/alerts (p. 32)
- GET /devices/alerts/count (p. 33)
- GET /devices/{deviceId}/events (p. 33)
- GET /devices/{deviceId}/events/{eventId} (p. 34)
- PUT /devices/{deviceId}/events/{eventId} (p. 35)

Status

- GET /devices/{deviceId}/status (p. 36)

GET /settings/config/{settingId}

Description

The `GET /settings/config/{settingId}` operation enables you to retrieve settings values.

Request Parameter

`settingId`

The unique setting ID

Type: String

Required: Yes

Response

Name	Description
`settingId`	The unique setting ID
`createdAt`	The date and time the setting was created
`updatedAt`	The date and time the setting was updated

For information about the errors that are common to all actions, see Common Errors (p. 36).

PUT /settings/config/{settingId}

Description

The `PUT /settings/config/{settingId}` operation enables you to update settings.

Request Parameter

`settingId`

The unique setting ID

Type: String

Required: Yes

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /registration

Description

The `GET /registration` operation enables you to retrieve a list of registered devices for a user, including deleted devices.

Response

Name	Description
`deviceId`	The unique device ID
`deviceName`	The device name
`modelNumber`	The model number of the device
`status`	The status of the device (pending, complete, deleted)
`userId`	The user ID of the device
`details`	Details about the device
`createdAt`	The date and time the device was registered
`updateAt`	The date and time the device was last updated

For information about the errors that are common to all actions, see Common Errors (p. 36).

POST /registration

Description

The `POST /registration` operation enables you to create a new device registration for a user. The response includes details about the newly registered device.

Request Body

Name	Description
`deviceId`	The unique device ID
`deviceName`	The device name
`modelNumber`	The model number of the device

Response

Name	Description
`deviceId`	The unique device ID
`deviceName`	The device name
`modelNumber`	The model number of the device
`status`	The status of the device (pending, complete, deleted)
`userId`	The user ID of the device
`details`	Details about the device
`createdAt`	The date and time the device was registered
`updateAt`	The date and time the device was last updated

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /devices

Description

The `GET /devices` operation enables you to retrieve a list of devices for a user.

Response

Name	Description
`deviceId`	The unique device ID
`deviceName`	The device name
`modelNumber`	The model number of the device
`status`	The status of the device (pending, complete, deleted)
`userId`	The user ID of the device
`details`	Details about the device
`createdAt`	The date and time the device was registered
`updateAt`	The date and time the device was last updated

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /devices/{deviceId}

Description

The `GET /devices/{deviceId}` operation enables you to retrieve details information about a device.

Request Parameter

`deviceId`

The unique device ID

Type: String

Required: Yes

Response

Name	Description
`deviceId`	The unique device ID
`deviceName`	The device name
`modelNumber`	The model number of the device
`status`	The status of the device (pending, complete, deleted)
`userId`	The user ID of the device
`details`	Details about the device
`createdAt`	The date and time the device was registered
`updateAt`	The date and time the device was last updated

For information about the errors that are common to all actions, see Common Errors (p. 36).

DELETE /devices/{deviceId}

Description

The `DELETE /devices/{deviceId}` operation enables you to delete a device.

Request Parameter

`deviceId`

The unique device ID

Type: String

Required: Yes

Response

`Delete successful`

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /devices/{deviceId}/commands

Description

The `GET /devices/{deviceId}/commands` operation enables you to retrieve a list of commands for a device.

Request Parameter

`deviceId`

The unique device ID

Type: String

Required: Yes

Response

Name	Description
`count`	The number of items
`ScannedCount`	The number of scanned items
`items`	Contains item details
`commandId`	The unique command ID
`deviceId`	The unique device ID
`reason`	The reason for the command result
`status`	The status of the command (success, failed, pending)
`details`	Details about the command
`command`	The specific command
`value`	The command value
`createdAt`	The date and time the command was issued

Name	Description
updateAt	The date and time the command was last updated
LastEvaluatedKey	The Amazon DynamoDB last evaluated key
commandStatus	The filtered command status

For information about the errors that are common to all actions, see Common Errors (p. 36).

POST /devices/{deviceId}/commands

Description

The `POST /devices/{deviceId}/commands` operation enables you to execute a command for a device.

Request Parameter

`deviceId`

The unique device ID

Type: String

Required: Yes

Response

Name	Description
commandId	The unique command ID
deviceId	The unique device ID
deviceName	The device name
reason	The reason for the command result
status	The status of the command (success, failed, pending)
details	Details about the command
command	The specific command
value	The command value
createdAt	The date and time the command was issued
updateAt	The date and time the command was last updated

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /devices/{deviceId}/commands/{commandId}

Description

The `GET /devices/{deviceId}/commands/{commandId}` operation enables you to retrieve detailed information about commands.

Request Parameter

`deviceId`

The unique device ID

Type: String

Required: Yes

`commandId`

The unique command ID

Type: String

Required: Yes

Response

Name	Description
`commandId`	The unique command ID
`deviceId`	The unique device ID
`deviceName`	The device name
`reason`	The reason for the command result
`status`	The status of the command (success, failed, pending)
`userId`	The user ID of the device
`details`	Details about the command
`command`	The specific command
`value`	The command value
`createdAt`	The date and time the command was issued
`updateAt`	The date and time the command was last updated

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /devices/events

Description

The `GET /devices/events` operation enables you to retrieve a list of events for a user.

Response

Name	Description
`count`	The number of items
`ScannedCount`	The number of scanned items
`items`	Contains item details
`id`	The unique event ID
`deviceId`	The unique device ID
`deviceName`	The device name
`userId`	The user ID of the device
`messageId`	The message ID
`message`	The event message
`details`	Details about the event
`ack`	Acknowledgement of the event (true, false)
`suppress`	Suppress the event (true, false)
`type`	The event type (error, warning, info, diagnostic)
`Timestamp`	The timestamp of the event
`value`	The event value
`sentAt`	The date and time the event was sent
`createdAt`	The date and time the event was created
`updateAt`	The date and time the event was last updated
`LastEvaluatedKey`	The Amazon DynamoDB last evaluated key
`deviceId`	The filtered device ID
`eventType`	The filtered event type

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /devices/alerts

Description

The `GET /devices/alerts` operation enables you to retrieve a list of alerts for a user.

Response

Name	Description
`count`	The number of items
`ScannedCount`	The number of scanned items
`items`	Contains item details
`id`	The unique alert ID
`deviceId`	The unique device ID
`deviceName`	The device name
`userId`	The user ID of the device
`messageId`	The message ID
`message`	The alert message
`details`	Details about the alert
`ack`	Acknowledgement of the alert (true, false)
`suppress`	Suppress the alert (true, false)
`type`	The alert type (error, warning, info, diagnostic)
`Timestamp`	The timestamp of the alert
`value`	The alert value
`sentAt`	The date and time the alert was sent
`createdAt`	The date and time the alert was created
`updateAt`	The date and time the alert was last updated
`LastEvaluatedKey`	The Amazon DynamoDB last evaluated key
`deviceId`	The filtered device ID

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /devices/alerts/count

Description

The `GET /devices/alerts/count` operation enables you to retrieve a count of unread alerts for a user based on the alert level.

Response

Name	Description
`alertsCount`	The number of unread alerts

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /devices/{deviceId}/events

Description

The `GET /devices/{deviceId}/events` operation enables you to retrieve a list of events for a device.

Request Parameter

`deviceId`

The unique device ID

Type: String

Required: Yes

Response

Name	Description
`count`	The number of items
`ScannedCount`	The number of scanned items
`items`	Contains item details
`id`	The unique event ID
`deviceId`	The unique device ID
`deviceName`	The device name

Name	Description
`userId`	The user ID of the device
`messageId`	The message ID
`message`	The event message
`details`	Details about the event
`ack`	Acknowledgement of the event (true, false)
`suppress`	Suppress the event (true, false)
`type`	The event type (error, warning, info, diagnostic)
`timestamp`	The timestamp of the event
`value`	The event value
`sentAt`	The date and time the event was sent
`createdAt`	The date and time the event was created
`updateAt`	The date and time the event was last updated
`LastEvaluatedKey`	The Amazon DynamoDB last evaluated key
`eventType`	The filtered event type

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /devices/{deviceId}/events/{eventId}

Description

The `GET /devices/{deviceId}/events/{eventId}` operation enables you to retrieve detailed information for an event.

Request Parameter

`deviceId`

The unique device ID

Type: String

Required: Yes

`eventId`

The unique event ID

Type: String

Required: Yes

Response

Name	Description
`id`	The unique event ID
`deviceId`	The unique device ID
`deviceName`	The device name
`userId`	The user ID of the device
`messageId`	The message ID
`message`	The event message
`details`	Details about the event
`ack`	Acknowledgement of the event (true, false)
`suppress`	Suppress the event (true, false)
`type`	The event type (error, warning, info, diagnostic)
`timestamp`	The timestamp of the event
`value`	The event value
`sentAt`	The date and time the event was sent
`createdAt`	The date and time the event was registered
`updateAt`	The date and time the event was last updated
`LastEvaluatedKey`	The Amazon DynamoDB last evaluated key
`eventType`	The filtered event type

For information about the errors that are common to all actions, see Common Errors (p. 36).

PUT /devices/{deviceId}/events/{eventId}

Description

The `PUT /devices/{deviceId}/events/{eventId}` operation enables you to update an event.

Request Parameter

`deviceId`

The unique device ID

Type: String

Required: Yes

`eventId`

The unique event ID

Type: String

Required: Yes

For information about the errors that are common to all actions, see Common Errors (p. 36).

GET /devices/{deviceId}/status

Description

The `GET /devices/{deviceId}/status` operation enables you to retrieve the status of a device.

Request Parameter

`deviceId`

The unique device ID

Type: String

Required: Yes

Response

Name	Description
`state`	The state of the device shadow
`metadata`	The device shadow metadata
`connected`	The connection status of the device (true, false)

For information about the errors that are common to all actions, see Common Errors (p. 36).

Common Errors

Error Code	Description
`400`	Invalid unique identifier
`401`	Access denied exception
`500`	Internal server error

Appendix B: Collection of Operational Metrics

This solution includes an option to send anonymous operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. When enabled, the following information is collected and sent to AWS:

- **Solution ID:** The AWS solution identifier
- **Unique ID (UUID):** Randomly generated, unique identifier for each live streaming solution deployment
- **Timestamp:** Data-collection timestamp

Note that AWS will own the data gathered via this survey. Data collection will be subject to the AWS Privacy Policy. To opt out of this feature, modify the AWS CloudFormation template mapping section as follows:

```
Solution:
   Data:
      SendAnonymousUsageData: "Yes"
```

to

```
Solution:
   Data:
      SendAnonymousUsageData: "No"
```

Source Code

You can visit our GitHub repository to download the templates and scripts for this solution, and to share your customizations with others.

Document Revisions

Date	Change
September 2019	Initial release
December 2019	Added information on support for Node.js update
February 2020	Bug fixes and Node updates

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents current AWS product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. The responsibilities and liabilities of AWS to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

The smart product solution is licensed under Apache License Version 2.0 available at https://www.apache.org/licenses/LICENSE-2.0