Machine to Cloud Connectivity Framework

Implementation Guide

Machine to Cloud Connectivity Framework: Implementation Guide

Table of Contents

Machine to Cloud Connectivity Framework

AWS Solutions Implementation Guide

Publication date: **August 2019 (last update (p. 53): March 2021)**

This implementation guide discusses architectural considerations and configuration steps for deploying Machine to Cloud Connectivity Framework 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 Core and AWS IoT Greengrass, and architecting in the AWS Cloud.

Overview

The Machine to Cloud Connectivity Framework solution helps factory production managers to connect their operational technology assets to the cloud, allowing fast and robust data ingestion into AWS. This solution allows for seamless connection to factory machines using either the OPC Data Access (OPC DA) protocol or the CC-Link Partner Association (CLPA) Seamless Messaging Protocol (SLMP).

This solution deploys AWS IoT Greengrass and related AWS resources on premises, so you can ingest OPC DA and SLMP telemetry data and store it in AWS (with Amazon Simple Storage Service (Amazon S3) or other storage resources), thereby allowing for further analysis of factory machine data for insights and improvements.

This solution is a framework for connecting factory equipment, allowing you to focus on extending the solution's functionality rather than managing the underlying infrastructure operations. For example, you can push the equipment data to Amazon S3 using Amazon Kinesis Data Streams and Amazon Kinesis Data Firehose and run machine learning models on the data for predictive maintenance, or create notifications and alerts.

This guide provides infrastructure and configuration information for planning and deploying the Machine to Cloud Connectivity Framework in the AWS Cloud.

Cost

You are responsible for the cost of the AWS services used while running this solution. As of March 2021, the cost for running this solution with 100 jobs, two tags sending data to AWS every second (200 records per second and approximately 0.6KB per each record), and with the default settings in the US East (N. Virginia) Region is approximately **$61.04 per month**. This includes estimated charges for Amazon Kinesis Data Stream, Amazon Kinesis Data Firehose, and AWS IoT Core connectivity. The cost estimate does not account for Amazon Simple Storage Service PUT requests.

AWS Service	Dimensions	Cost per month
Amazon Kinesis Data Streams	730 shard hours / month 525,600,000 PUT Payload Units / month	$18.31
Amazon Kinesis Data Firehose	494.38GB data ingested / month	$14.34
Amazon Simple Storage Service	296.63GB / month	$6.83
AWS IoT Greengrass	one device / month	$0.16
AWS IoT Core connectivity	43,200 minutes of connection / month	$0.0035
	Total:	**$61.04 / month**

Optionally, if you send your machine data to AWS IoT topics, the pricing for AWS IoT Core messaging would be charged.

Optional AWS Service	Dimensions	Cost per month
AWS IoT Core messaging	518,400,000 messages / month	$518.40

Prices are subject to change. For full details, refer to the pricing webpage for each AWS service you will be using in this solution.

Architecture overview

Deploying this solution with the default parameters builds the following environment in the AWS Cloud.

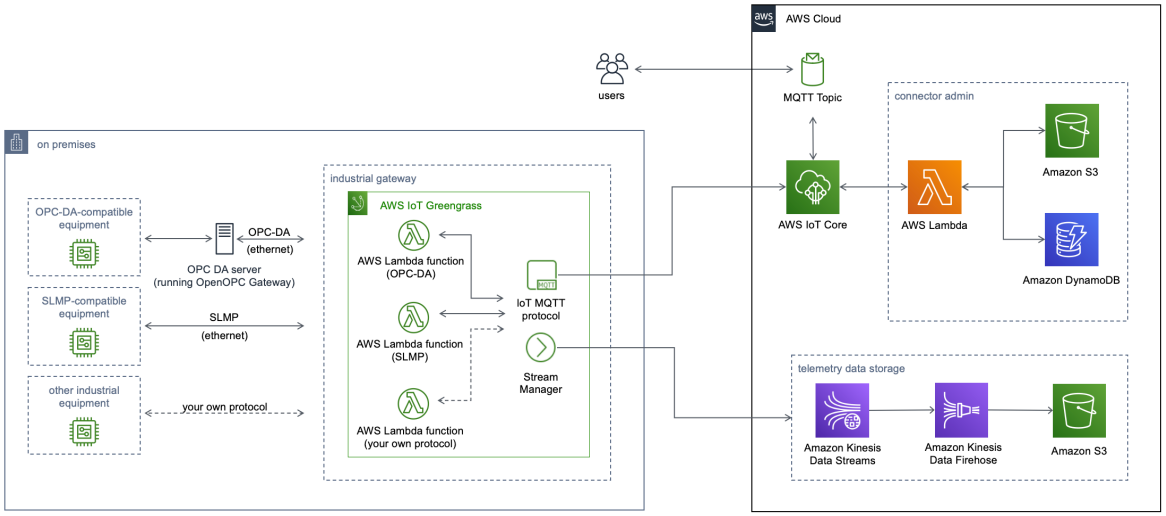

Figure 1: Machine to Cloud Connectivity Framework default architecture on AWS

When deploying the AWS CloudFormation template using the default parameters, AWS IoT Greengrass v1.0, AWS Lambda functions, Amazon Kinesis Data Streams, Amazon Kinesis Data Firehose, and AWS IoT Core are configured and activated in your AWS environment. These AWS services allow your factory equipment to securely connect and interact with cloud applications and other equipment. An AWS Lambda function and IoT AWS IoT Greengrass manage the connection and data ingestion from on-premises equipment.

AWS IoT Core MQTT topics, AWS IoT Greengrass, and AWS Lambda functions manage communication between the factory equipment and the AWS Cloud. AWS IoT Greengrass runs on the gateway device on premises. Kinesis Data Streams, Kinesis Data Firehose, and Amazon S3 store the data.

If you choose to send telemetry data to AWS IoT Core, the solution publishes the data to AWS IoT core using an MQTT topic. The topic on which the data is published is based on the protocol type and job definition under the `m2c2/job/` topic hierarchy.

An AWS IoT thing is created to represent your industrial gateway, which includes:

- An X.509 certificate and 2048-bit RSA keypair
- An associated AWS IoT policy

These security resources are needed to configure your industrial gateway edge device running AWS IoT Greengrass core to securely communicate with resources in your AWS account.

Note
AWS CloudFormation resources are created from AWS Cloud Development Kit (CDK) (AWS CDK) constructs.

Solution components

AWS Lambda functions

Job builder

When you submit a job request, the submission initiates the *<stack-name>*-M2C2JobBuilder AWS Lambda function. This *job builder* Lambda function uses an AWS IoT MQTT message broker and the AWS IoT Rules Engine to receive the job request, invoke the Lambda function, and send commands to the gateway.

This function parses the job request and extracts the run command. Then, the function translates the command and data parameters into the protocol-specific language that the *machine connector* Lambda function will use to communicate with your factory equipment.

The job builder Lambda function also checks the job metadata stored in Amazon DynamoDB to determine whether the job name already exists. If the job name does not exist, the function creates a new JSON job definition and stores it in the DynamoDB table. If the job name already exists, it creates a new version of the job. This function creates a machine connector Lambda function to handle the job. The job builder function also creates a new subscription in the AWS IoT Greengrass group with the machine connector Lambda function as the source and a target topic. When using the OPC DA connector, the target topic is m2c2/job/*<job-name>*/*<site-name>*/*<area>*/*<process>*/*<machine-name>*/*<tag-name>*. When using the SLMP connector, the target topic is m2c2/job/*<job-name>*/*<tag-name>*.

When a job is created or updated, this solution sends a message to the m2c2/job/*<job-name>*/submit topic which starts the machine connector Lambda function.

Machine connector

The m2c2-*<protocol>*-connector Lambda function is provisioned by the job builder Lambda function, and is deployed to the edge gateway when the AWS IoT Greengrass group is deployed. This machine connector Lambda function writes the job to the local gateway, establishes connectivity with the OPC DA server configuration specified in the job, and reads the applicable data from the server. This Lambda function also manages the connectivity with the AWS resources to send data to the cloud, and reads the latest running job and sends it back to the cloud.

When using the default connector functionality, the OPC DA machine connector sends data to your Kinesis data stream and, when using the default configuration provided with the CloudFormation template, Kinesis Data Firehose retrieves the data from the data stream, bundles batches of the data into GZIP files, and stores them in an S3 bucket. If you do not want to use the default CloudFormation configuration, then you need to provide your existing data stream on which the connector will publish the OPC DA telemetry data.

The SLMP machine connector will only send data to an AWS IoT topic.

When an OPC DA job is defined to send data to an IoT topic, the OPC DA machine connector Lambda function sends the data to the m2c2/job/*<job-name>*/*<site-name>*/*<area>*/*<process>*/*<machine-name>*/*<tag-name>* topic in AWS IoT Core. Error messages

are sent to the `m2c2/job/<job-name>/<site-name>/<area>/<process>/<machine-name>/error` topic and information messages are sent to the `m2c2/job/<job-name>/<site-name>/<area>/<process>/<machine-name>/info` topic.

The SLMP machine connector Lambda function sends the data to the `m2c2/job/<job-name>/<tag-name>` topic in AWS IoT Core. Error messages are sent to the `m2c2/job/<job-name>/error` topic and information message are sent to the `m2c2/job/<job-name>/info` topic.

You can use the AWS IoT Rules Engine to subscribe to these topics to either store their data in your own data lake or Amazon S3 bucket, or to initiate notifications using Amazon Simple Notification Service (Amazon SNS).

JSON-formatted job definition

To get data from on-premises equipment, you define a JSON-formatted job definition request containing information about the equipment, the tags associated with the equipment, and the read frequency for the equipment data.

You can publish the job definition using the job builder Lambda function to an IoT topic (`m2c2/job/request`). The job builder Lambda function creates a new machine connector Lambda function with the job configuration, updates the AWS IoT Greengrass group, and automatically publishes the new Lambda function and job request to the industrial gateway running AWS IoT Greengrass. The machine connector Lambda function that is now running in AWS IoT Greengrass industrial gateway edge device receives the request, establishes a connection between the device gateway and the equipment, retrieves the data defined in the request, and sends it to the appropriate resource in your AWS account.

When you use the OPC DA connector with the default CloudFormation configuration, the default destination is a Kinesis data stream, which sends data through Kinesis Data Firehose to an S3 bucket. The OPC DA connector can also send data to an AWS IoT topic or to both an S3 bucket and an AWS IoT topic, depending on the values set within the job definition. If you want to send the data to a location other than the S3 bucket or the AWS IoT topic, you need to provide the data stream that is in your AWS account when launching the CloudFormation stack (using the **Name of the Existing Data Stream in Kinesis Data Streams** parameter), and create a consumer to read from the stream and send the data to your desired destination.

When you use the SLMP connector, the data will only be published to an AWS IoT topic.

The equipment data is translated to JSON and, by default, stored in an S3 bucket or, when configured within the job definition, posted to a topic in AWS IoT Core. The OPC DA connector publishes the data on the `m2c2/job/<job-name>/<site-name>/<area>/<process>/<machine-name>/<tag-name>` topic in AWS IoT Core. The SLMP connector publishes data on the `m2c2/job/<job-name>/<tag-name>` topic. You can also specify what action you want to perform on the data either in Amazon S3 or by configuring your own Rules Engine in AWS IoT Core.

Solution workflow

This solution uses AWS IoT Greengrass to communicate with on-premises equipment using either the OPC Data Access (OPC DA) protocol or the CC-Link Partner Association (CLPA) Seamless Messaging Protocol (SLMP). After launching the CloudFormation template, the following workflow is set up to collect your telemetry data.

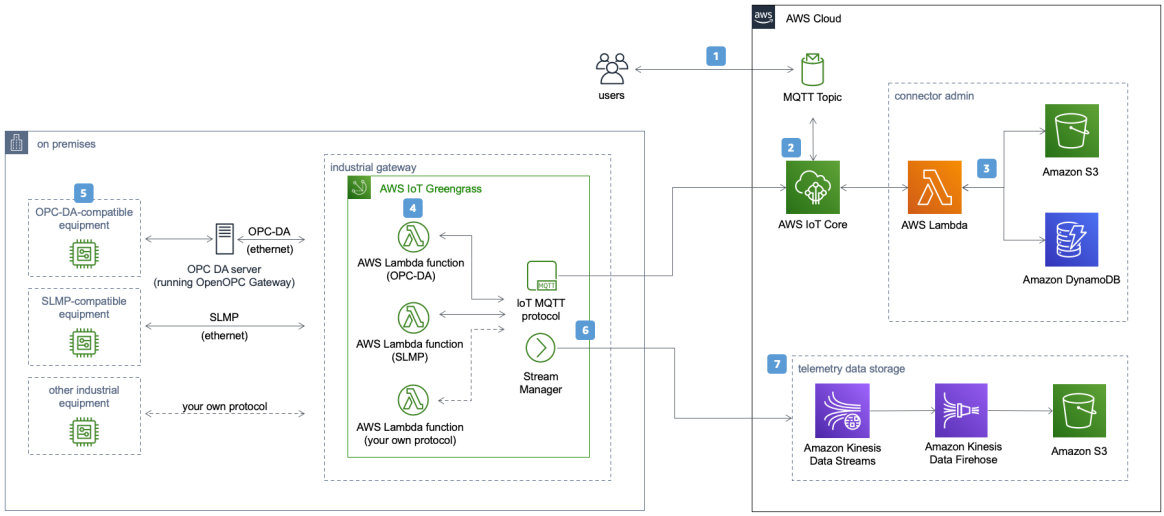

Figure 2: Workflow for the Machine to Cloud Connectivity Framework solution

1. You publish a job definition with the deploy control on an IoT topic.
   - A job defines the details around the telemetry data being collected, including the protocol the source machine is using, the data collection-specific tags, and the poll frequency for the data collection.
   - The deploy control communicates with the system when you want to create a new job.
2. The message is routed to AWS IoT Core where an IoT rule forwards it to the job builder Lambda function.
3. The job builder Lambda function receives the request, creates a new machine connector Lambda function, writes the job data to DynamoDB and, depending on the protocol used, Amazon S3. The Greengrass group is also updated with the necessary settings.
4. The Greengrass group is deployed to your industrial gateway, including the new machine connector Lambda function.
5. The machine connector Lambda function establishes a connection to your industrial device, as configured in the job definition.
6. Once the connection is established, the machine connector Lambda function receives the telemetry data from your device and sends that data to a Kinesis data stream and/or an IoT topic.
7. In the default configuration, when the data is sent to the data stream, a Kinesis Data Firehose delivery stream takes that data, batches the data, and stores it in Amazon S3.

Communicating with your AWS IoT Greengrass industrial gateway

You can issue command and control signals to your on-premises AWS IoT Greengrass industrial gateway to control telemetry data using JSON-formatted job definitions published to an IoT topic. The job builder Lambda function running in your AWS account listens for the IoT topic, transforms the message, and depending on the control specified in the job definition, takes the appropriate actions to publish a message to an IoT topic on which the machine connector Lambda function, running on the AWS IoT Greengrass industrial gateway, is subscribed. The machine connector Lambda function then reads the message from the topic and takes the appropriate action.

Getting data from on-premises equipment to AWS

The telemetry data is transmitted to resources in your AWS account by the industrial gateway running AWS IoT Greengrass. For your industrial gateway to be able to write data to your AWS account on your behalf, the IoT policy attached to your X.509 certificate needs to have the appropriate permissions to write to the resources.

Data destination locations vary depending on the protocol used by your on-premises hardware. If you are collecting data from a machine that uses SLMP, your data will be published on an IoT Topic.

If you are collecting data from an OPC DA machine, you can define where the industrial gateway publishes the data and the machine connector sends the data to the defined location. You can choose to send your data to an IoT topic or to a Kinesis data stream. The destination is defined in the job definition. The default behavior is to send your data to the data stream. You can deactivate this job functionality by setting the `send-data-to-kinesis-stream` value in the job definition to `false`. You can also send your data to an IoT topic. To activate this job functionality, set the `send-data-to-iot-topic` to `true`. When using the default configuration for your CloudFormation stack, a Kinesis Data Firehose delivery stream consumes the data from the data stream, bundles data into larger GZIP files, and stores the GZIP files in an S3 bucket.

For details on the job definitions available for this solution, refer to Job Definitions (p. 37).

AWS CloudFormation parameters

This solution provides two AWS CloudFormation parameters – **ID of the Existing AWS IoT Greengrass Group** and **Name of the Existing Data Stream in Kinesis Data Streams** – to determine whether the necessary AWS resources are created on your behalf or you are providing the necessary AWS resources. By default, these parameters are left empty, which means the solution automatically creates new AWS resources on your behalf.

ID of the Existing AWS IoT Greengrass Group

When you provide a AWS IoT Greengrass group ID value for this parameter, the solution uses this AWS IoT Greengrass group to make configuration changes and as destination to deploy the machine connector Lambda function. The solution does **not** create AWS IoT Greengrass or AWS IoT resources used by the solution, including a AWS IoT Greengrass group, an AWS IoT thing to represent your industrial gateway, or an X.509 certificate or RSA keypair.

Name of the Existing Data Stream in Kinesis Data Streams

When you provide a data stream name value for this parameter, the solution uses the specified stream to push telemetry data. The solution does **not** create a new Kinesis data stream, a Kinesis Data Firehose delivery stream, or an S3 bucket. You will be responsible for creating a consumer to pull messages from the data stream and manage the storage of that data.

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared model reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, the virtualization layer, and the physical security of the facilities in which the services operate. For more information about AWS security, visit AWS Cloud Security.

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 help customers to assign granular access policies and permissions to services and users on the AWS Cloud. The Machine to Cloud Connectivity Framework creates several IAM roles, including roles that grant the job builder AWS Lambda function access to the other AWS services used in this solution.

AWS IoT Core policies

AWS IoT Core policies allow you to control access to the AWS IoT Core data plane. The AWS IoT Core data plane consists of operations that allow you to connect to the AWS IoT Core message broker, send and receive MQTT messages, and get or update a device's shadow. The Machine to Cloud Connectivity Framework solution creates an AWS IoT Core policy which allows the AWS IoT Greengrass core to connect to AWS IoT Core, send and receive MQTT messages, and control AWS IoT Greengrass.

Design considerations

Supported protocols

Machine to Cloud Connectivity Framework currently supports the OPC Data Access (OPC DA) specification. OPC DA defines how real-time data can be transferred between a data source and a data sink without either having to know the other's native protocol.

The solution also supports equipment that uses the CC-Link Partner Association (CLPA) Seamless Messaging Protocol (SLMP). SLMP is a unified protocol for achieving seamless communication between applications without awareness of network hierarchy or boundaries and general-purpose ethernet devices.

AWS IoT Greengrass

Before you can deploy this solution, you must have the following requirements set up in your environment:

- AWS IoT Greengrass version 1.0 is installed

  Important
  As of March 2021, this solution only supports AWS IoT Greengrass v1.0.
- Your on-premises gateway supports AWS IoT Greengrass and is connected to the OPC DA server

AWS IoT Greengrass support

This solution uses AWS IoT Greengrass version 1.0. This version of AWS IoT Greengrass must be installed in your industrial gateway located on-premises or with network connectivity to the PLCs, CNCs, or other hardware that is generating the OPC DA or SLMP telemetry data that you want to collect. Additionally, this AWS IoT Greengrass gateway must have network connectivity to an OPC DA server in your environment. If you do not have AWS IoT Greengrass installed in your AWS account, you have two options to set this up:

- Allow this solution to automatically configure the AWS IoT Greengrass resources you will need.
- Manually prepare your environment.

For set up instructions, refer to Prerequisites (p. 14).

On-premises gateway

Your on-premises gateway must support AWS IoT Greengrass and is connected to the OPC DA server. If you choose to have this solution create the AWS IoT Greengrass group for you (the default set up), you only need to have your gateway running the AWS IoT Greengrass core software. This solution creates the configuration and files needed for your device to securely interact with resources in your AWS account. If you choose to connect your existing AWS IoT Greengrass group, ensure your gateway device is configured to communicate with your account. For more information, refer to Supported Platforms and Requirements in the *AWS IoT Greengrass Developer Guide*. To set up AWS IoT Greengrass on your gateway, refer to Environment Setup for AWS IoT Greengrass.

AWS IoT Greengrass group

A AWS IoT Greengrass group is a collection of settings and components, such as an AWS IoT Greengrass core, devices, and subscriptions. Groups are used to define a scope of interaction. For example, a group might represent one floor of a building, one truck, or an entire mining site.

Using your existing AWS IoT Greengrass and Amazon Kinesis Data Streams resources

You have the option to deploy this solution using your existing AWS IoT Greengrass resources and an Amazon Kinesis data stream. By providing values for the CloudFormation parameters (**ID of the Existing AWS IoT Greengrass Group** and **Name of the Existing Data Stream in Kinesis Data Streams**) during deployment, this solution does **not** create certain resources within your account and instead uses the parameters to use the resources you have provided. For example, if you provide a AWS IoT Greengrass group ID, the solution will not create any of the AWS IoT Greengrass or AWS IoT resources. Instead, it uses the AWS IoT Greengrass group that you manage to make configuration changes such as adding the machine connector Lambda function and creating the required subscriptions. If you provide a Kinesis data stream name, the solution does **not** create a data stream, Kinesis Data Firehose, or an S3 bucket. Instead, it pushes the telemetry data to the data stream you provided.

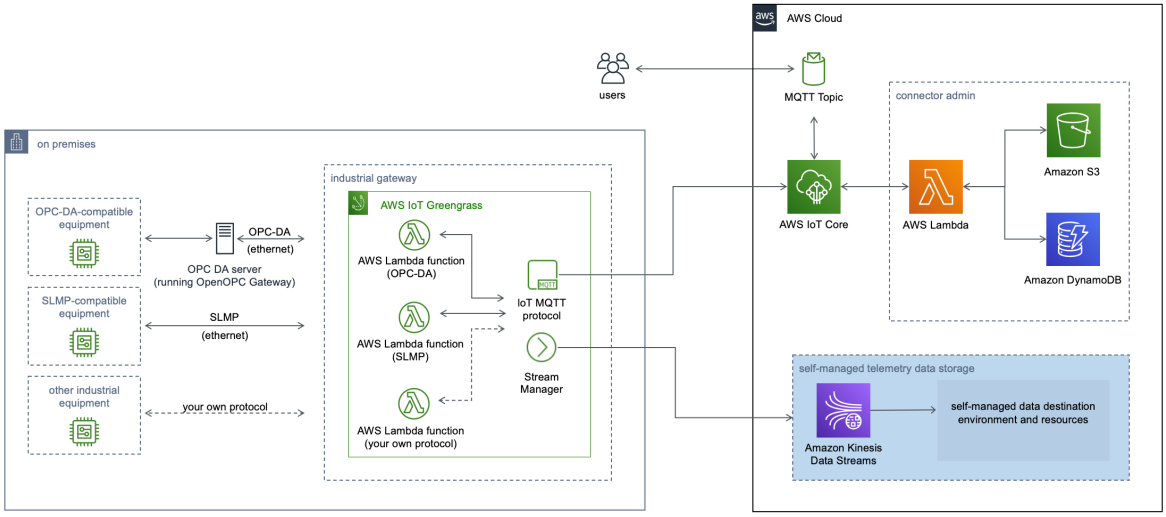

Figure 3: Machine to Cloud Connectivity Framework architecture with a self-managed data stream

The architecture in Figure 3 shows the set up when you connect to an existing Kinesis data stream. This solution does **not** create the data stream, Amazon Kinesis Data Firehose, or Amazon S3 bucket. Instead, it uses your existing data stream as a parameter when launching the CloudFormation template.

When you connect to your existing AWS IoT Greengrass group, the default architecture (as shown in Figure 1) does not change. This solution applies the configuration changes to your self-managed AWS IoT Greengrass group.

X.509 certificates

Communication between your factory equipment and AWS IoT is protected through the use of X.509 certificates. This solution can generate a certificate for you or you can use your own X.509 certificate when you create your own AWS IoT Greengrass group. You can register your preferred Certificate

Authority (CA), which is used to sign and issue the device certificate, with AWS IoT. For more information, refer to Register a CA certificate in the *AWS IoT Core Developer Guide*.

AWS IoT Rules Engine

If you configure your job to send data through IoT topics (p. 37), you can leverage the AWS IoT Rules Engine to take action based on the data that this solution collects. The Rules Engine evaluates inbound messages published to AWS IoT Core and transforms and delivers them to another device or a cloud service, based on business rules you define. For more information, refer to Rules for AWS IoT in the *AWS IoT Core User Guide*.

Error logging

This solution logs errors and information messages to different AWS IoT topics, based upon the deploy connector.

For the OPC DA connector, the errors and information messages are in the `m2c2/job/<site-name>/<area>/<process>/<machine-name>` topic in AWS IoT Core. Error messages are logged in the `m2c2/job/<job-name>/<site-name>/<area>/<process>/<machine-name>/`error topic and information messages are logged `m2c2/job/<job-name>/<site-name>/<area>/<process>/<machine-name>`/info topic.

For the SLMP connector, the solution logs errors and information messages in the `m2c2/job/<job-name>` topic in AWS IoT Core. Error messages are logged in the `m2c2/job/<job-name>/error` topic and information messages are logged in `m2c2/job/<job-name>/info` topic.

We recommend that you monitor the error topic when submitting jobs by subscribing to either the protocol error topic or the `m2c2/#` wildcard topic, which subscribes to all messages published to the `m2c2` hierarchy.

Regional deployments

This solution uses AWS IoT Greengrass, which is not currently available in all AWS Regions. You must launch this solution in an AWS Region where AWS IoT Greengrass is available. For the most current availability by Region, refer to the AWS Regional Services List.

Updating the solution

If you have previously deployed the solution, you must update the solution's CloudFormation stack to get the latest version of the solution's framework. For details, refer to Update the stack (p. 47).

AWS CloudFormation template

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

machine-to-cloud-connectivity-framework: Use this template to launch the solution and all associated components. The default configuration creates an AWS IoT Core thing and associated configurations, an AWS IoT Greengrass group, AWS IoT Core, an X.509 certificate and RSA keypair, AWS Lambda functions, Amazon Simple Storage Service (Amazon S3) buckets, an Amazon Kinesis Data Streams data stream, Amazon Kinesis Data Firehose, and an Amazon DynamoDB table, but you can customize the template to meet your specific needs.

> **Note**
> AWS CloudFormation resources are created from AWS Cloud Development Kit (CDK) (AWS CDK) constructs.

Automated deployment

Before you launch the solution, review the cost, architecture, components, and design considerations discussed in this guide. Follow the step-by-step instructions in this section to configure and deploy the solution into your account.

> **Note**
> If you have previously deployed the solution, you must update the solution's CloudFormation stack to get the latest version of the solution's framework. For details, refer to Update the stack (p. 47).

Time to deploy: Approximately five minutes

Prerequisites

Before you can deploy the AWS CloudFormation template, your on-premises industrial gateway must be configured to support AWS IoT Greengrass and be able to connect to an OPC Data Access (DA) or Seamless Messaging Protocol (SMLP) server. This solution can create the AWS IoT Greengrass group in your account, including the configuration, certificate, and keys that you need to install on your edge device. You can either automate the creation of AWS AWS IoT Greengrass resources or manually prepare your industrial gateway environment.

- Automate the creation of AWS IoT Greengrass resources if you want to get started quickly with a new environment or you do not have an existing AWS IoT Greengrass group you want to use for this solution.
- Manually prepare your industrial gateway environment if you or your company already have a AWS IoT Greengrass group created that you want to use in this solution or your company has security or compliance requirements that require you to create the AWS IoT Greengrass group to satisfy the requirements.

To manually prepare your industrial gateway environment, complete the following sections before deploying this solution.

> **Important**
> As of March 2021, this solution only supports AWS IoT Greengrass v1.0. You must configure AWS IoT Greengrass before you can deploy this solution.

Verify that the Greengrass service role is attached to your AWS Region

To allow your industrial gateway to interact with your AWS account properly, AWS IoT Greengrass requires the necessary permission to communicate with your other services in order to read and write data.

Use the following steps to verify that you have provided AWS IoT Greengrass with the necessary permissions.

1. Sign in to the AWS IoT console.
2. On the Settings page, scroll down to the **AWS IoT Greengrass service role** section.

   If you already have AWS IoT Greengrass set up, a role displays with the associated policy or policies attached. Otherwise, the following message is displayed:

```
You do not have a service role attached to your AWS account in the current
AWS Region.
```

3. To attach a role, choose **Attach role**.
   - If you have roles available, select the `Greengrass_Service` role, and choose **Save**.
   - If no roles are listed, choose **Create role for me**.

The `Greengrass_Service` role is created with the appropriate permissions attached. For more information about this role, refer to Greengrass service role in the *AWS IoT Greengrass Developer Guide, Version 1*.

Automate the configuration of AWS IoT Greengrass on your industrial gateway

If you choose to have the solution create the Greengrass group for you, ensure that your industrial gateway meets the requirements and prerequisites as described in Install the AWS IoT Greengrass Core software in the *AWS IoT Greengrass Developer Guide, Version 1*.

Install AWS IoT Greengrass on a device running the Linux operating system. This machine will be your industrial gateway. Once your industrial gateway has AWS IoT Greengrass installed, you can proceed to Launch the stack (p. 18).

Manually configure AWS IoT Greengrass for your industrial gateway

To manually set up AWS IoT Greengrass on your industrial gateway, refer to Environment Setup for Greengrass and Installing the Greengrass Core Software in the *AWS IoT Greengrass Developer Guide, Version 1*.

After your industrial gateway is set up to run AWS IoT Greengrass, use the following steps to configure the gateway to work with this solution.

1. At a command prompt on your device gateway, navigate to the root directory.
2. Use the following commands to create an `m2c2` folder structure.

```
sudo mkdir -p /m2c2/job
```

3. Use the following command to give the AWS IoT Greengrass user access to the solution folders.

```
sudo chown -R ggc_user /m2c2/job/
```

Note
When launching the CloudFormation template, enter your newly created Greengrass group ID in the parameter field labeled **ID of the Existing Greengrass Group**. You can find your Greengrass group ID in the AWS IoT Greengrass console.

Configure Your X.509 Certificate

This solution must have an activated X.509 certificate to communicate with the resources in your AWS account.

- If you use the default CloudFormation deployment, this certificate is automatically generated. You can proceed directly to Launch the stack (p. 18).
- If you use your existing AWS IoT Greengrass group, you have the option to either apply an existing X.509 certificate or have AWS IoT generate one.

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

If you choose to manually generate a certificate using AWS IoT, you must download the root certificate authority (CA) certificate. To download the CA certificate, refer to Server Authentication in the *AWS IoT Developer Guide*.

Configure Your OPC DA Server

Prior to using this solution with the OPC DA protocol, you must configure an OPC DA server to allow the machine connector AWS Lambda function to connect and collect data. The following sections outline how to install and configure Python 3.x and OpenOPC for Python 3.x, an open source OPC (OLE for Process Control) toolkit.

Requirements:

Verify that your OPC DA server has the following resources installed:

- Python 3.x (32bit)
- PyWin32 (32bit) PyWin32
- Pyro4
- OPC DA Auto Wrapper (32bit)
- OpenOPC for Python 3.x

Security Settings:

If you are running Microsoft Windows server with Windows Firewall or Windows Firewall with Advanced Security activated, then add an inbound rule to allow traffic on port 7766. Details on configuring inbound rules are available in Create and Inbound Port Rule in the Microsoft 365 documentation.

Configuration steps:

The following steps shows a configuration example for an OPC DA server running on a Windows 2016 server. This configuration allows the server to communicate with the machine connectors Lambda functions running AWS IoT Greengrass on the industrial gateway. Your steps may vary depending on the version of Windows you are running.

1. From the Windows server running OPC DA, download and install Python 3.x (32 bit) and the corresponding version of PyWin32 (32 bit).

   Important
   Ensure that the version of PyWin32 matches the Python 3.x version you are downloading.
2. From the OPC DA server's terminal window, add the path to Python and the Python scrips folder as System environment variables:
   - Navigate to **Control Panel**, **System and Security**, **System** and, in the left side panel, select **Advanced System Settings**.
   - From **System Properties**, select the **Advanced** tab and select **Environment Variables**.
   - Under **System Variables**, highlight the **Path** option and select **Edit**.

Note
Verify that you are in the **Systems Variables** section, and not in User Variables.

- Add the Python path and Python scripts path. For example:

```
C:\Users\Administrator\AppData\Local\Programs\Python\Python39-32\

C:\Users\Administrator\AppData\Local\Programs\Python\Python39-32\Scripts
```

Note
Check the exact path to your installed version.

3. Install Pryo4 using pip.

```
pip install Pyro4
```

4. Download and install the OPC DA auto wrapper:
   - Download and unzip the OPC DA Auto Wrapper (32 bit) zip file.
   - In a command prompt, navigate to the package x86 folder location and register the module:

```
cd YOUR_OPC_AUTOMATION_WRAPPER_FOLDER\x86
regsvr32 gbda_aut.dll
```

5. At the command prompt, add the following registry values, replacing *<your_private_IP_address>* with the IP of the windows server:

```
reg add "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /f /v
 OPC_GATE_HOST /t REG_EXPAND_SZ /d "<your_private_IP_address>"

reg add "HKLM\SYSTEM\CurrentControlSet\Control\Session
 Manager\Environment" /f /v OPC_CLASS /t REG_EXPAND_SZ /d
 "Matrikon.OPC.Automation;Graybox.OPC.DAWrapper;HSCOPC.Automation;RSI.OPCAutomation;OPC.Automation"

reg add "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /f /v
 OPC_CLIENT /t REG_EXPAND_SZ /d "OpenOPC"

reg add "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /f /v
 OPC_GATE_PORT /t REG_EXPAND_SZ /d "7766"

reg add "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /f /v
 OPC_HOST /t REG_EXPAND_SZ /d "localhost"

reg add "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /f /v
 OPC_INACTIVE_TIMEOUT /t REG_EXPAND_SZ /d "60"

reg add "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /f /v
 OPC_MAX_CLIENTS /t REG_EXPAND_SZ /d "25"

reg add "HKLM\SYSTEM\CurrentControlSet\Control\Session Manager\Environment" /f /v
 OPC_MODE /t REG_EXPAND_SZ /d "dcom"

reg add "HKLM\SYSTEM\CurrentControlSet\Control\Session
 Manager\Environment" /f /v OPC_SERVER /t REG_EXPAND_SZ /d
 "Hci.TPNServer;HwHsc.OPCServer;opc.deltav.1;AIM.OPC.1;Yokogawa.ExaopcDAEXQ.1;OSI.DA.1;OPC.PHDServerI
 Instruments.OPCLabVIEW;RSLinx OPC
 Server;KEPware.KEPServerEx.V4;Matrikon.OPC.Simulation;Prosys.OPC.Simulation"
```

6. Download and run OpenOPC:
   - Download and unzip the OpenOPC zip file.

- In a text editor, open the `OpenOPCService.py` file in the `src` folder; find and replace all occurrences of "`_winreg`" with "`winreg`".
- At the command prompt, navigate to the `src` folder, and run the following commands:

```
python OpenOPCService.py install
net start zzzOpenOPCService
netstat -an | findstr 7766
```

If the net stat command fails to run, then the connection fails. Verify the OpenOPC gateway is running and inbound traffic on port 7766 is activated on the server.

Deployment overview

Use the following steps to deploy this solution on AWS. For detailed instructions, follow the links for each step.

Step 1. Launch the stack (p. 18)

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

Step 2. Configure and start the AWS IoT Greengrass Core (p. 20)

> **Note**
> Skip Step 2 if you are creating the AWS IoT Greengrass group manually outside of this solution since you would have already completed this step.

Step 3. Deploy a job (p. 21)

- Create a JSON job definition and submit the job request to AWS IoT Core.
- Create the job and supporting configuration, deploy the AWS IoT Greengrass group changes to your industrial gateway, and start collecting data.

Step 4. (Optional) Test connectivity (p. 22)

- If using the OPC DA or SLMP protocols, test the connectivity between your gateway and the factory equipment.

Step 1. Launch the stack

This automated AWS CloudFormation template deploys Machine to Cloud Connectivity Framework in the AWS Cloud. You must complete the prerequisites (p. 14) before launching the stack.

> **Note**
> You are responsible for the cost of the AWS services used while running this solution. For more details, visit the Cost (p. 2) section in this guide, and refer to the pricing webpage for each AWS service used in this solution.

1. Sign in to the AWS Management Console and select the button to launch the machine-to-cloud-connectivity-framework AWS CloudFormation template.

Alternatively, you can download the template as a starting point for your own implementation.

2. The template launches 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 the AWS IoT Greengrass service, which is not currently available in all AWS Regions. You must launch this solution in an AWS Region where AWS IoT Greengrass is available. For the most current availability by Region, refer to the AWS Regional Services List.

3. On the **Create stack** page, verify that the correct template URL is in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, assign a name to your solution stack. For information about naming character limitations, refer to AWS CloudFormation quotas in the *AWS CloudFormation User Guide*.
5. Under **Parameters**, review the parameters for this solution template and modify them as necessary. By leaving a parameter blank, the solution will create the specified resource on your behalf.

Parameter	Default	Description
ID of the Existing AWS IoT Greengrass Group	<Optional input>	Determines whether this solution generates new AWS IoT Greengrass resources or use your existing AWS IoT Greengrass group. If you are using AWS IoT Greengrass v1.0, you can use this group by specifying the ID. You can find the ID using either the AWS Management Console or the AWS CLI. For example, `1234a5b6-78cd-901e-2fgh-3i45j6k178l`
Name of the Existing Data Stream in Kinesis Data Streams	<Optional input>	Determines whether this solution creates a new Kinesis data stream or use your existing one. To use your existing data stream, enter the appropriate name. You can find the name using either the AWS Management Console or the AWS CLI. For example, `mystream`.

6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review and confirm the settings. 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 receive a **CREATE_COMPLETE** status in approximately five minutes.

Note
In addition to the primary AWS Lambda functions, this solution includes the `CustomResourceHelper` and `CertCreator` Lambda functions, which runs only during initial configuration or when resources are updated or deleted.
When you run this solution, you will notice multiple Lambda functions in the AWS console. Only the primary Lambda functions are regularly active. However, you must not delete the `CustomResourceHelper` and `CertCreator` functions, as they are necessary to manage associated resources.

Step 2. Configure and start the AWS IoT Greengrass Core

If you deployed the CloudFormation stack using the default parameters, then this solution created a new AWS IoT Greengrass group. Use the following procedure to configure your industrial gateway to communicate with the AWS IoT Greengrass Core. For information about how to configure the core, refer to Configure the AWS IoT Greengrass Core in the *AWS IoT Greengrass Developer Guide*.

If you entered your existing AWS IoT Greengrass group ID in the parameters, then proceed to Step 3 (p. 21).

1. From the AWS CloudFormation console, select the **Outputs** tab.
2. Under the **Key** column, locate **CertKeyPairS3URL**, and under the **Value** column, copy the URL address.

   Note
   This URL is valid for two hours. If your URL times out, you can download the `tar.gz` file directly from the public S3 bucket. To identify the name of the S3 bucket, review the **Key** column for the **M2C2Bucket** value and use the corresponding **Value** to locate the S3 bucket from the Amazon S3 console.
3. Paste the URL in a new browser window and press Enter. A `tar.gz` file downloads to your local machine.
4. Upload the tar file to your industrial gateway.
5. Connect to your industrial gateway.
6. Unzip the `tar.gz` file. For example, on a Unix/Linux device, use the command:

```
tar xfvz m2c2-greengrass-STACK_NAME.tar.gz
```

7. Run the install script to configure your device to communicate with your AWS account:

```
sudo ./setup.sh
```

   Note
   The setup script places the certificate, key pair, and configuration files in the appropriate directories and prepares your edge device to work with this solution.

Step 3. Create and deploy a JSON job definition file

Use the following procedure to define a data collection job and deploy it to the industrial gateway. This procedure creates and deploys a job for the OPC DA connector (an example is provided in OPC DA job deployment (p. 25)). For more information on the JSON job definition file, refer to Job Definitions (p. 37). You submit the JSON job definition to an AWS IoT topic.

1. In a text editor, create a JSON job definition.

   This job definition will differ based on the protocol you want to use. These steps follow the example to create a job definition for the OPC DA server (p. 25). A Sample SLMP JSON job definition (p. 31) is also available.
2. In the JSON job definition, set the **control** (p. 25) property to deploy.

   ```
   "control": "deploy",
   ```

   Note
   When you create or update a job using the deploy or update controls, but do not specify a Greengrass group ID, the solution uses the ID you specified during initial deployment. The solution may return the message: `Parameter gg-group-id not found in the job request`, but it creates the job with the default Greengrass group ID.
3. By default, this solution sends data to a defined Kinesis data stream. If deploying the default stack, the stream sends data to Kinesis Data Firehose, which bundles the data into `tar.gz` files and stores them in Amazon S3. To turn off this functionality for the job, set the **send-data-to-kinesis-stream** property to false:

   ```
   "send-data-to-kinesis-stream": false,
   ```
4. By default, this solution does **not** send data to an IoT topic. To activate this functionality to send data to the topic: `m2c2/job/<job-name>/<site-name>/<area>/lt;process>/<machine-name>/<tag-name>`, set the **send-data-to-iot-topic** property to true:

   ```
   "send-data-to-iot-topic": true,
   ```
5. Publish the job definition to the `m2c2/job/request` IoT topic on AWS IoT. As an example, you can use the test functionality in the AWS IoT console:

   1. Navigate to the AWS IoT console.

      Note
      This console provides two user interface options: the original console experience or the new console experience. Refer to the banner at the top of the page to identify the version you are using.

      

      Figure 4: AWS IoT banner for the original console
   2. In the navigation pane, choose **Test**.
   3. If you are using the original console:
      - In the **Publish** panel, **Specify a topic to publish to** text box, enter `m2c2/job/request`.
      - Replace the sample JSON code with the JSON job definition code.

- Choose **Publish to topic**.

4. If you are using the new console:
   - Choose **Publish** to a topic tab.
   - In the **Topic name** text box, enter `m2c2/job/request`.
   - In **Message payload**, replace the sample JSON code with the JSON job definition code.
   - Choose **Publish**.

It can take up to two minutes for the command to run and an acknowledgment message to be sent to the topic. The message will be similar to the following sample message.

```
Job successfully created. Deploying the Greengrass group to the edge device.
```

Note
If you deploy a new AWS IoT Greengrass group to your gateway, all running jobs on the gateway will temporarily stop so there will be a pause in sending data during the AWS IoT Greengrass group update deployment.

This solution deploys AWS IoT Greengrass group updates to your edge device and starts sending the data to your AWS account. Previously running jobs are restarted. If you are publishing your data to an IoT topic, the data will be published on a different topic depending on the protocol you are using. For OPC Data Asset (OPC DA), the topic is `m2c2/job/<job-name>/<site-name>/<area>/<process>/<machine-name>`. For SLMP, a message is posted to the `m2c2/job/<job-name>` topic. After the job initiates, use the update, stop, and pull controls to modify the job.

Step 4. (Optional) Test connectivity

Use this procedure to test the connectivity between the gateway and the factory machine.

Note
This will be only available after the AWS IoT Greengrass deployment completes.

1. Navigate to the AWS IoT console.
2. In the navigation pane, select **Test**.
3. In the **Specify a topic to publish to** text box, enter `m2c2/job/request`.
4. Replace the sample JSON code with the following code:

```
{
    "job": {
      "properties": [
        {
          "name": "<job-name>",
          "version": "<job-version>"
        }
      ],
      "control": "push"
    }
}
```

If the connection is successful, the Lambda function on the gateway returns a message to the `m2c2/job/<job-name>/info` topic. The message varies depending on the protocol you use. For OPC DA, refer to OPC Data Access (p. 25). For SLMP, refer to Seamless Messaging Protocol (p. 31).

Job control overview

The following flowchart shows the process for creating new jobs and working with existing jobs.

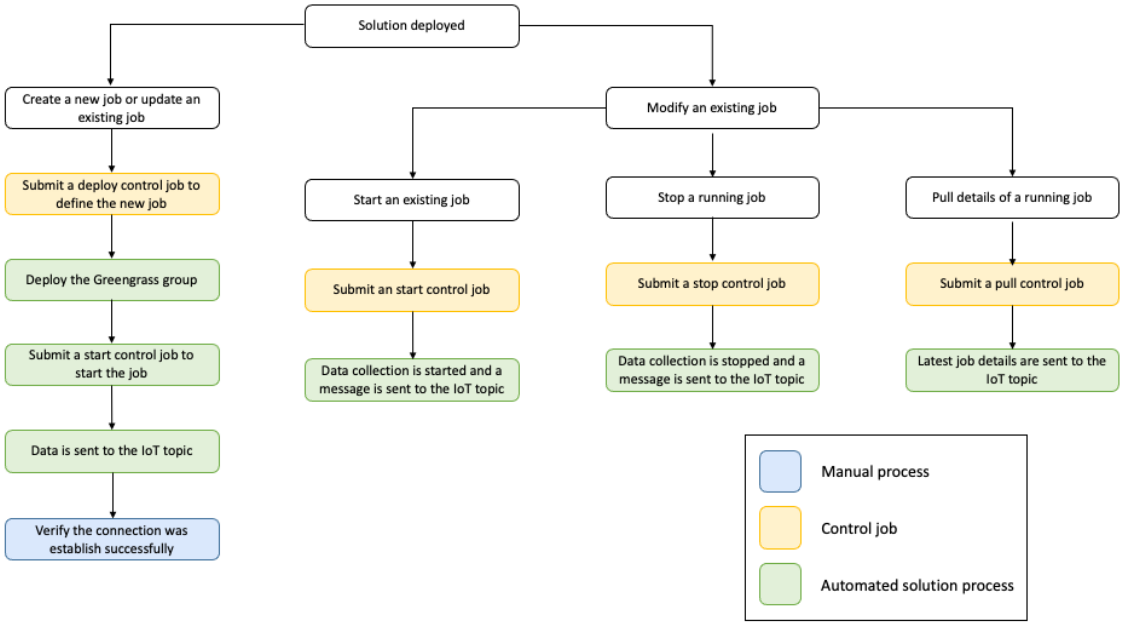

Figure 5: Process flowchart for creating and updating jobs

The following tasks must be completed to create a new job:

- Define the job and submit it. For OPC DA, refer to OPC Data Access (p. 25). For SLMP, refer to Seamless Messaging Protocol (p. 31).
- Manually deploy the AWS IoT Greengrass group (p. 15) to your gateway.
- Check the connectivity (p. 22) between your machine and your gateway.
- Verify that the connection was successful.

Additional resources

AWS services

- Amazon DynamoDB
- Amazon Kinesis Data Firehose
- Amazon Kinesis Data Streams
- Amazon Simple Storage Service
- AWS CloudFormation
- AWS Identity and Access Management
- AWS IoT Core
- AWS IoT Greengrass
- AWS Lambda

AWS documentation

- AWS IoT Greengrass stream manager

OPC Data Access

The Machine to Cloud Connectivity Framework supports the OPC Data Access (OPC DA) specification.

Note
OPC DA typically supports a subscription mechanism that sends the data only when there is a value change. However, this solution does not support subscriptions.

OPC DA job deployment

Once the solution is ready to deploy a job for a OPC DA server, publish a `deploy` control job to the `m2c2/job/request` AWS IoT topic.

Note
To activate the connectivity between the Machine to Cloud Connectivity Framework OPC DA connector you must install OpenOPC on your OPC DA Server by following the steps outlined in the Configure Your OPC DA Server (p. 16) prerequisite section.

The following example shows an OPC DA job data to connect to the Matrikon OPC simulator.

```
{
  "job": {
    "control": "deploy",
    "properties": [
      {
        "version": "1",
        "name": "job-machine1"
      }
    ],
    "send-data-to-iot-topic": false,
    "send-data-to-stream-manager": true,
    "machine-details": {
      "process": "packaging",
      "site-name": "London",
      "area": "floor 1",
      "data-parameters": {
        "machine-query-iterations": 3,
        "machine-query-time-interval": 1,
        "attributes": [
          {
            "function": "read_list",
            "address-list": [
              "Simulation Items.Random.*Int*"
            ]
          },
          {
            "function": "read_tags",
            "address-list": [
              "Random.Int2",
              "Random.Int4"
            ]
          }
        ]
      },
      "connectivity-parameters": {
        "opcda-server-name": "Matrikon.OPC.Simulation.1",
        "machine-ip": "169.254.89.33",
        "protocol": "opcda",
```

```
        "machine-name": "machine 1"
      }
    }
  }
}
```

When the job deployment is successful, this solution deploys the AWS IoT Greengrass group to an AWS IoT Greengrass core automatically and send a `start` control job to the `m2c2/job/request` AWS IoT topic. All previously started jobs also automatically start. For potential issues, refer to Troubleshooting (p. 45).

Connectivity check

To check the connectivity of the OPC DA server, publish a `push` control job to the `m2c2/job/request` AWS IoT topic. The following code shows an example payload.

```
{
  "job": {
    "control": "push",
    "properties": [
      {
        "name": "<job-name>"
      }
    ]
  }
}
```

If the connection is successful, the Lambda function on the gateway returns a message containing a server list to the `m2c2/job/`*`<job-name>`*`/`*`<site-name>`*`/`*`<area>`*`/`*`<process>`*`/`*`<machine-name>`*`/info` topic. The message is similar to the following sample message.

```
{
    "_id_": "a10d3db5-367b-4dff-aac2-fec28134ebd6",
    "_timestamp_": "2019-09-26 18:42:18.665823",
    "version": "1",
    "site-name": "London",
    "area": "floor 1",
    "process": "packaging",
    "machine-name": "machine 1",
    "message": "Available server: ['Matrikon.OPC.Simulation.1']"
}
```

Note
Receiving the message only indicates connectivity between the gateway and your factory equipment. It does not mean your server is correctly configured to send and receive data from your equipment. You must verify that your server configuration, including your access control list, is correct as provided in the Prerequisites (p. 14).

For potential issues with connectivity, refer to the Troubleshooting (p. 45) section of this document.

Start OPC DA job

You can manually start OPC DA jobs. When you start jobs, you can define whether to receive OPC DA data to AWS IoT topic and AWS IoT Greengrass stream manager. By default, the Lambda function on the gateway sends data to AWS IoT Greengrass stream manager only. Currently, AWS IoT Greengrass stream

manager only sends data to Amazon Kinesis Data Streams. To start a job on the OPC DA server, publish a `start` control job to `m2c2/job/request` AWS IoT topic. The following code shows an example payload.

```
{
  "job": {
    "control": "start",
    "properties": [
      {
        "name": "<job-name>",
        "version": 1
      }
    ],
    "send-data-to-iot-topic": false,
    "send-data-to-stream-manager": true
  }
}
```

You can also start multiple jobs at once by putting multiple jobs in `properties`. If the job is successfully started, the Lambda function on the gateway returns a message to the `m2c2/job/<job-name>/<site-name>/<area>/<process>/<machine-name>/info` topic. The message is similar to the following sample message.

```
{
    "_id_": "a10d3db5-367b-4dff-aac2-fec28134ebd6",
    "_timestamp_": "2019-09-26 18:42:18.665823",
    "version": "1",
    "site-name": "London",
    "area": "floor 1",
    "process": "packaging",
    "machine-name": "machine 1",
    "message": "Job started."
}
```

Note
The `start` control job only stops when the job is not running. If the job is already running, The following message displays: `A version of the requested '<version>' is already running. Please stop it before starting it again.`

Once the job is running, the gateway Lambda function sends data to AWS IoT topic and AWS IoT Greengrass stream manager based on the configuration. The following is sample OPC DA data.

```
{
    "_id_": "a10d3db5-367b-4dff-aac2-fec28134ebd6",
    "_timestamp_": "2021-02-19 01:25:26.891876",
    "version": "1",
    "site-name": "London",
    "area": "floor 1",
    "process": "packaging",
    "machine-name": "machine 1",
    "tag": "Random-Int2",
    "alias": "London/floor 1/packaging/machine 1/Random-Int2",
    "messages": [
      {
        "name": "London/floor 1/packaging/machine 1/Random-Int2",
        "value": 4,
        "quality": "Good",
        "timestamp": "2021-02-19 01:25:26.827000+00:00"
      }
    ]
}
```

Stop an OPC DA job

To stop a job on the OPC DA server, publish a `stop` control job to `m2c2/job/request` AWS IoT topic. The following code shows an example payload.

```
{
  "job": {
    "control": "stop",
    "properties": [
      {
        "name": "<job-name>",
        "version": 1
      }
    ]
  }
}
```

You can also stop multiple jobs at once by putting multiple jobs in properties. If the job is successfully stopped, the Lambda function on the gateway returns a message to the `m2c2/job/<job-name>/<site-name>/<area>/<process>/<machine-name>/info` topic. The message is similar to the following sample message.

```
{
    "_id_": "a10d3db5-367b-4dff-aac2-fec28134ebd6",
    "_timestamp_": "2019-09-26 18:42:18.665823",
    "version": "1",
    "site-name": "London",
    "area": "floor 1",
    "process": "packaging",
    "machine-name": "machine 1",
    "message": " Job stopped."
}
```

Note
The stop control job only stops when the job is running. If the job is already stopped, the following message displays: `Job '<job-name>' has already been stopped.`

Update OPC DA job

To update the existing job, publish an `update` control job to `m2c2/job/request` AWS IoT topic. The following code shows an example payload.

```
{
  "job": {
    "control": "update",
    "properties": [
      {
        "version": "2",
        "name": "job-machine1"
      }
    ],
    "send-data-to-iot-topic": false,
    "send-data-to-stream-manager": true,
    "machine-details": {
      "process": "packaging",
      "site-name": "London",
      "area": "floor 1",
      "data-parameters": {
```

```
        "machine-query-iterations": 2,
        "machine-query-time-interval": 1,
        "attributes": [
          {
            "function": "read_tags",
            "address-list": [
              "Random.Int2",
              "Random.Int4"
            ]
          }
        ]
      },
      "connectivity-parameters": {
        "opcda-server-name": "Matrikon.OPC.Simulation.1",
        "machine-ip": "169.254.89.33",
        "protocol": "opcda",
        "machine-name": "machine 1"
      }
    }
  }
}
```

If the job is successfully updated, the Lambda function on the gateway returns a message to the `m2c2/job/<job-name>/<site-name>/<area>/<process>/<machine-name>/info` topic. The message is similar to the following sample message.

```
{
  "_id_": "3d1f3f5a-d3f8-4d5b-8102-e6629ba1e2f4",
  "_timestamp_": "2021-02-19 01:49:20.329063",
  "version": "2",
  "site-name": "London",
  "area": "floor 1",
  "process": "packaging",
  "machine-name": "machine 1",
  "message": "Job updated."
}
```

In addition, the gateway Lambda function automatically starts with the updated job configuration.

> **Note**
> If the job does not exist, the following error message displays: `A unique database entry must exist for Job <job-name> version <version>. in the m2c2/job/<job-name>/error` AWS IoT topic.

Get OPC DA local JSON job definition

To get the OPC DA local JSON job definition from the gateway, publish a `pull` control job to `m2c2/job/request` AWS IoT topic. The following code shows an example payload.

```
{
"job": {
    "control": "pull",
    "properties": [
      {
        "name": "<job-name>"
      }
    ]
  }
}
```

If there is a local JSON job definition in the gateway, the Lambda function on the gateway returns a message containing local JSON job definition details to the `m2c2/job/<job-name>/<site-name>/<area>/<process>/<machine-name>/info` topic. The message is similar to the following sample message.

```
{
  "_id_": "b4569cab-d827-4b08-b22e-11ad96775a71",
  "_timestamp_": "2021-02-19 01:40:10.549725",
  "version": "1",
  "site-name": "London",
  "area": "floor 1",
  "process": "packaging",
  "machine-name": "machine 1",
  "message": {
    "job": {
      "control": "deploy",
      "properties": [
        {
          "version": "1",
          "name": "job-machine1"
        }
      ],
      "send-data-to-iot-topic": false,
      "send-data-to-stream-manager": true,
      "machine-details": {
        "process": "packaging",
        "site-name": "London",
        "area": "floor 1",
        "data-parameters": {
          "machine-query-iterations": 3,
          "machine-query-time-interval": 1,
          "attributes": [
            {
              "function": "read_list",
              "address-list": [
                "Simulation Items.Random.*Int*"
              ]
            },
            {
              "function": "read_tags",
              "address-list": [
                "Random.Int2",
                "Random.Int4"
              ]
            }
          ]
        },
        "connectivity-parameters": {
          "opcda-server-name": "Matrikon.OPC.Simulation.1",
          "machine-ip": "169.254.89.33",
          "protocol": "opcda",
          "machine-name": "machine 1"
        }
      },
      "gg-group-id": "e47c02f3-4335-43e5-b01f-292bad39f2d3",
      "_last-update-timestamp_": "2021-02-19 01:25:28.426170"
    }
  }
}
```

Note
The gateway Lambda function only returns the local JSON job definition details when the job has been started.

Seamless Messaging Protocol

The Machine to Cloud Connectivity Framework supports equipment that uses the CC-Link Partner Association (CLPA) Seamless Messaging Protocol (SLMP).

Prerequisites

To use SLMP, you must meet the following prerequisites:

- You must add an SLMP node to your programmable logic controller (PLC).
- The communication code, IP address, and port number you submit with your job request must match those specified in your PLC configuration.
- Verify that your industrial gateway IP in your PLC is in the allow list.

Considerations

This solution supports the following SLMP functions:

- **Device read:** reads a value from the next consecutive device number.
- **Device read random:** reads a value from a random device number.
- **Array label read:** reads data from a label of an array type.
- **Label read random:** reads data from the specified label.

All SLMP parameters must be specified in the correct format according to the SLMP specification. For example, head device or number of points must be in ASCII or binary format.

Currently, the SLMP label function supports one abbreviation.

Connectivity

To check the connectivity, submit a push control job. If the connection is successful, the Lambda function on the gateway returns a message to the `m2c2/job/<job-name>/info` topic that confirms whether TCP or UDP communication can occur.

Sample SLMP JSON job definition

Once the solution has established connectivity, submit a job request using a JSON job definition. The following example shows a JSON job definition in ASCII that performs four jobs.

- A device read from M100 to M131 (two words)
- A device read random of D0, T0, M100 to M115, X20 to X2F by word access, and D1500 to D1501, Y160 to Y17F, M1111 to M1142 by double word access.

- An array label read of four words from the label of structured type with a data type `word`, `"Typ1.led[2]"`, and two words from the label of structure type with a data type `word`, `"Typ1.No[1]"`.
- A label read random from the primitive data type label `"LabelB"` with the data type `bit`, the primitive data type label `"LabelW"` with the data type `word`, and structured type label `"Sw.led"` with the data type `word`.

```
{
  "job": {
    "control": "deploy",
    "properties": [
      {
        "version": "1",
        "name": "machine1"
      }
    ],
    "machine-details": {
      "site-name": "Herndon",
      "process": "packaging",
      "machine-name": "machine 1",
      "data-parameters": {
        "attributes": [
          {
            "function": "device_read",
            "address-list": {
              "tag-name": "tst_tag",
              "subcommand": "0000",
              "device-code": "M*",
              "head-device": "000100",
              "number-of-points": "0002"
            }
          },
          {
            "function": "device_read_random",
            "address-list": {
              "subcommand": "0000",
              "words": [
                {
                  "tag-name":"tst_wd_0",
                  "device-code":"D*",
                  "head-device":"000000"
                },{
                  "tag-name":"tst_wd_1",
                  "device-code":"TN",
                  "head-device":"000000"
                },{
                  "tag-name":"tst_wd_2",
                  "device-code":"M*",
                  "head-device":"000100"
                },{
                  "tag-name":"tst_wd_3",
                  "device-code":"X*",
                  "head-device":"000020"
                }
              ],
              "dwords": [
                {
                  "tag-name":"tst_dwd_0",
                  "device-code":"D*",
                  "head-device":"001500"
                },{
                  "tag-name":"tst_dwd_1",
                  "device-code":"Y*",
```

```
              "head-device":"000160"
            },{
              "tag-name":"tst_dwd_2",
              "device-code":"M*",
              "head-device":"001111"
            }
          ]
        }
      },
      {
        "function": "array_label_read",
        "address-list": {
          "subcommand": "0000",
          "abbreviation": [
            "Typ1"
          ],
          "label-list":[
            {
              "tag-name":"tag1",
              "label": "%1.led[2]",
              "data-length":8,
              "read-unit":1
            },{
              "tag-name":"tag2",
              "label": "%1.No[1]",
              "data-length":4,
              "read-unit":1
            }
          ]
        }
      },
      {
        "function": "label_read_random",
        "address-list": {
          "subcommand": "0000",
          "abbreviation": [],
          "label-list":[
            {
              "tag-name":"tag3",
              "label": "LabelB"
            },{
              "tag-name":"tag4",
              "label": "LabelW"
            },{
              "tag-name":"tag5",
              "label": "Sw.led"
            }
          ]
        }
      }
    ],
    "machine-query-iterations": 5,
    "machine-query-time-interval": 1
  },
  "connectivity-parameters": {
    "port-number": 5548,
    "machine-ip": "192.168.3.250",
    "protocol": "slmp",
    "network": 0,
    "station": 255,
    "module": "03FF",
    "multidrop": 0,
    "timer": 0,
    "subheader": "with serial",
    "communication-code": "ascii",
    "ethernet": "tcp"
```

```
      },
      "area": "floor 1"
    }
  }
}
```

The following example shows a JSON job definition in binary.

```
{
  "job": {
    "control": "deploy",
    "properties": [
      {
        "version": "1",
        "name": "temp"
      }
    ],
    "machine-details": {
      "site-name": "Herndon",
      "process": "packaging",
      "machine-name": "machine 1",
      "data-parameters": {
        "attributes": [
          {
            "function": "device_read",
            "address-list": {
              "tag-name": "tst_tag",
              "subcommand": "0000",
              "device-code": "90",
              "head-device": "640000",
              "number-of-points": "0200"
            }
          },
          {
            "function": "device_read_random",
            "address-list": {
              "subcommand": "0000",
              "words": [
                {
                  "tag-name":"tst_wd_0",
                  "device-code":"A8",
                  "head-device":"000000"
                },{
                  "tag-name":"tst_wd_1",
                  "device-code":"C2",
                  "head-device":"000000"
                },{
                  "tag-name":"tst_wd_2",
                  "device-code":"90",
                  "head-device":"640000"
                },{
                  "tag-name":"tst_wd_3",
                  "device-code":"9C",
                  "head-device":"200000"
                }
              ],
              "dwords": [
                {
                  "tag-name":"tst_dwd_0",
                  "device-code":"A8",
                  "head-device":"DC0500"
                },{
                  "tag-name":"tst_dwd_1",
                  "device-code":"9D",
                  "head-device":"600100"
```

```
            },{
              "tag-name":"tst_dwd_2",
              "device-code":"90",
              "head-device":"570400"
            }
          ]
        }
      },
      {
        "function": "array_label_read",
        "address-list": {
          "subcommand": "0000",
          "abbreviation": [
            "Typ1"
          ],
          "label-list":[
            {
              "tag-name":"tag1",
              "label": "%1.led[2]",
              "data-length":8,
              "read-unit":1
            },{
              "tag-name":"tag2",
              "label": "%1.No[1]",
              "data-length":4,
              "read-unit":1
            }
          ]
        }
      },
      {
        "function": "label_read_random",
        "address-list": {
          "subcommand": "0000",
          "abbreviation": [],
          "label-list":[
            {
              "tag-name":"tag3",
              "label": "LabelB"
            },{
              "tag-name":"tag4",
              "label": "LabelW"
            },{
              "tag-name":"tag5",
              "label": "Sw.led"
            }
          ]
        }
      }
    ],
    "machine-query-iterations": 5,
    "machine-query-time-interval": 1
  },
  "connectivity-parameters": {
    "port-number": 5548,
    "machine-ip": "192.168.3.250",
    "protocol": "slmp",
    "network": 0,
    "station": 255,
    "module": "03FF",
    "multidrop": 0,
    "timer": 0,
    "subheader": "with serial",
    "communication-code": "binary",
    "ethernet": "tcp"
  },
```

```
      "area": "floor 1"
    }
  }
}
```

Job definitions

To extract data from your connected factory equipment, create a JSON job definition that contains the details about the job including connectivity and data parameters. A JSON job definition contains both generic and protocol-specific properties. The JSON job definition can contain the following properties.

Property	Type	Required	Protocol(s)	Description
job	Object	Yes	All	Contains the structure of the configuration of the machine you want to connect to.
gg-group-id	String	No	All	The AWS IoT Greengrass group ID of the group where you want to deploy the machine connector AWS Lambda function.
properties	Object	Yes	All	Contains an array that stores job metadata including the job name and version.
name	String	Yes	All	A logical name for the job that you want to submit. This property is used to determine whether the job is new or an update to an existing job.
version	String	No	All	The version of the job you want to submit. This should be incremented for every update to an existing job. Enter the value as an integer in a string. For example, enter 2 for version 2 (do **not** enter 2.0. Only the latest version of the job will be executed at the gateway.

Property	Type	Required	Protocol(s)	Description
control	Enum	Yes	All	Lets you control the running of the job. Allowed values include: deploy, push, start, stop, update, and pull.
send-data-to-kinesis-stream	Boolean	No	OPC-DA	Lets you toggle sending telemetry data to AWS IoT Greengrass stream manager. Allowed values are true and false. The default value is set to true. Today, the stream manager sends messages to a Kinesis data stream.
send-data-to-iot-topic	Boolean	No	OPC-DA	Lets you toggle sending telemetry data to an IoT topic. Allowed values are true and false. The default value is set to false.
machine-details	Object	Yes	All	Contains the details of the machine you want to connect to.
site-name	String	No	All	The name of the factory where the machine is located. For example, site1.
area	String	No	All	The area within the factory where the machine is located. For example, floor1.
process	String	No	All	The name of the process where the machine is located. For example, packaging.

Property	Type	Required	Protocol(s)	Description
connectivity-parameters	Object	Yes	All	Contains parameters that are required to connect to the machine.
machine-name	String	Yes	All	The name of the machine you want to connect to. For example, machine1.
protocol	String	Yes	All	The protocol supported by the machine. For example, opcda.
machine-ip	String	Yes	OPC-DA, SLMP	The IP address of the machine you want to connect to.
opcda-server-name	String	Yes	OPC-DA	The name of the OPC DA server you want to connect to. For example, opcdaserver1.
port-number	Integer	Yes	SLMP	The number of the TCP or UPD port. Specify the same port you specified in your PLC configuration.
network	Integer	Yes	SLMP	The request destination network number corresponding to the access destination.
station	Object	Yes	SLMP	The request destination station number corresponding to the access destination.
module	Object	Yes	SLMP	The module of the access destination.
multidrop	Object	Yes	SLMP	The request destination multidrop station number.

Property	Type	Required	Protocol(s)	Description
timer	Object	Yes	SLMP	The monitoring timer (unit: 250 milliseconds).
subheader	Object	Yes	SLMP	Choose whether to include a serial number in the subheader. Choose with serial or without serial.
communication-code	Object	Yes	SLMP	Specify the code for data communication. Choose binary or ascii. This must match the code in your PLC configuration.
ethernet	Object	Yes	SLMP	Choose the Ethernet protocol. Choose either tcp or udp.
data-parameters	Object	Yes	All	Contains the data parameters for the data you want to read from the machine.
machine-query-time-interval	Float	Yes	All	The interval, in seconds, at which you want to read data from the machine. If you do not specify an interval, data is read every second. Specify a value between 0.5 and 30 seconds. For example, enter 1 to read data every second. If you use the deploy or update control commands, this property is required. For more information, refer to Machine Queries (p. 43).

Property	Type	Required	Protocol(s)	Description
machine-query-iterations	Integer	No	All	The number of iterations that will occur before data is sent to the cloud. Specify a value between 1 and 30. For example, enter 20 to send data every 20 iterations. If you use the deploy or update control commands, this property is required. For more information, refer to Machine Queries (p. 43).
attributes	Array	Yes	All	An array of functions and tags you want to read from the machine.
function	String	Yes	All	The function you want to read data. For OPC DA, the solution currently supports read_list and read_tags enabling you to read a full list of a tags or specific tags from the machine.
subcommand	Object	Yes	SLMP	The SLMP subcommand.
words	Object	Yes	SLMP	A list of parameters that are read as words.
dwords	Object	Yes	SLMP	A list of parameters that are read as double words.
tag-name	Object	Yes	SLMP	A user-defined tag name.
device-code	Object	Yes	SLMP	A code that identifies the access destination device for request data.

Property	Type	Required	Protocol(s)	Description
head-device	Object	Yes	SLMP	The number of the device that the file is read from.
abbreviation	Object	Yes	SLMP	Indicates that the label name is in abbreviated form. Specify a value such as %1, %2, etc.
label	Object	Yes	SLMP	The label name.
data-length	Object	Yes	SLMP	The length of the read data.
read-unit	Object	Yes	SLMP	The unit of the read data.
number-of-points	Object	Yes	SLMP	The number of points to be read.
address-list	String	Yes	All	A comma-delimited list of tags and list names from the machine that you want to read. For example, tag1, tag2, tag3.

The following code example shows the properties for an OPC DA job definition. For a code example that shows the properties for an SLMP job definition, refer to Sample SLMP JSON job definition (p. 31).

```
{
  "job": {
    "gg-group-id": String
    "properties": [
        {
            "name": String,
            "version": String
        },
        ...
        {
            "name": String,
            "version": String
        }
    ]
    "control": String,
    "send-data-to-iot-topic": Boolean,
    "send-data-to-kinesis-stream": Boolean,
    "machine-details": {
      "site-name": String,
      "area": String,
      "process": String,
      "connectivity-parameters": {
        "machine-name": String,
        "protocol": String,
```

```
        "machine-ip": String,
        "opcda-server-name": String
      },
      "data-parameters": {
          "machine-query-time-interval": Float,
          "machine-query-iterations": Integer,
          "attributes": [
            {
              "function": String,
              "address-list": [
                String,
                String,
                …
                String
              ]
            },
            {
              "function": String,
              "address-list": [
                String,
                String,
                …
                String
              ]
            }
          ]
        }
      }
    }
  }
}
```

Machine queries

The frequency at which data is read from the machine and is sent to the cloud is controlled by two properties: **machine-query-time-interval** and **machine-query-iterations**. The time interval is the interval, in seconds, at which you want to read data from the machine. The iteration is the number of iterations that occurs before data is sent to the cloud.

For example, to read data every 0.5 seconds and send it to the cloud every 10 seconds, set the time interval property to `0.5` and set the iteration property to `20`. The iteration property value is the number of data reads that will be completed before sending the data to the cloud. The result will be a message sent to resources in your account every 10 seconds with 20 samples of data that were taken every 0.5 seconds. To read data every 2 seconds and send it to the cloud every minute, set the time interval parameter to `2` and set the iteration to `30`.

Job controls

Job control commands allows you to have full control over the job that is running on each machine. This solution supports the following commands.

- **Deploy:** This command deploys the JSON job definition to the device gateway, deploys the updated AWS IoT Greengrass group to the device gateway, and starts the job on the gateway. To deploy a job to a different device gateway than the one you specified during initial deployment, use the **gg-group-id** property. Make sure you populate all protocol-specific properties correctly.
- **Start:** This command starts the job on the gateway. To start multiple jobs, specify multiple job properties.

- **Stop:** This command stops a job that is running on the gateway. To stop multiple jobs, specify multiple job properties.
- **Update:** This command updates any job that has already been submitted to the gateway with a new version number and parameters. Populate all protocol-specific properties correctly.

  Note
  Update will stop the running job and restart it with the updated parameters.
- **Push:** This command tests the connectivity between the gateway and the factory machine. To push multiple jobs, specify multiple job properties.
- **Pull:** This command retrieves details on the last version of a job submitted to the gateway. Note that the version property is not required for this control.

Troubleshooting

Problem: I don't receive any messages on the IoT topic after the following message:

```
Job successfully created. Deploying the Greengrass group to the edge device.
```

Resolution: This solution only supports AWS IoT Greengrass v1.0. Verify that you have this version of AWS IoT Greengrass installed in your AWS account.

Navigate to your AWS IoT Greengrass group in your console by going to the AWS IoT console, select **Greengrass**, select **Classic(V1)**, and choose **Groups**. Select your group name and review the deployments.

- If deployments have been In Progress for an extended time, check:
  - that the Greengrass process is running.
  - you have the correct certificate, key pair, and configuration on your Greengrass industrial gateway so that the group changes can be pushed to your gateway.
  - if you are using your own Greengrass group, verify that the IoT policy attached to your certificate has the correct permissions to communicate with Greengrass.
- If the deployment failed, select the deployment to get more information on the reason for failure.
- Explore the logs for any possible failures. You can view the logs in CloudWatch logs. Navigate to your CloudFormation stack, select **Resources**, and search for `M2C2JobBuilderLambdaFunction`. Select the link to go to your Lambda function. Select the **Monitor** tab and choose **View logs in CloudWatch**.

Problem: Running the command `net start zzzOpenOPCService` on the Windows OPC DA server returns the following error message:

```
net : The service is not responding to the control function.
```

Resolution: Verify that you entered the python paths in Systems variables. For more information, refer to the Prerequisites (p. 14) section.

Problem: The AWS IoT Greengrass group deployment fails or is stuck.

Resolution: Verify that the industrial gateway is configured correctly. Refer to Prerequisites (p. 14).

Problem: If the connection to the OPC DA server fails and you are not receiving any messages to either the Kinesis data stream or the AWS IoT topic.

Resolution: Verify the following:

- The OpenOPC gateway is running on the OPC DA server
- Port 7766 (used for OPC DA) is open on the OPC DA server
- The firewall settings on the OPC DA server are configured to allow the machine to communicate with the gateway

Problem: I created a deploy job and received the following error message:

```
TES service role is not associated with this account.
```

Resolution: Verify that you have a service role associated with your account. Refer to the Verify the Greengrass service role is attached to your Region (p. 14).

Problem: There is no response from the Greengrass Lambda function on any `m2c2/job/<job-name>` topics.

Resolution: Verify that your industrial gateway Lambda function is not returning errors and that it can communicate with your OPC DA server.

- Connect to the industrial gateway
- View the connector log for errors. For example: `/greengrass/ggc/var/log/user/<Region>/<account_id>/m2c2-opcda-connector-<job-name>-<hash-ID>.log`
- Check the connectivity to the OPC DA server:
  - In the AWS IoT Greengrass console, navigate to the Lambda deployment directory. For example:

    `/greengrass/ggc/deployment/lambda/arn.aws.lambda.<Region>.<account_id>.function.m2c2-opcda-connector-<job-name>-<hash-ID>.1.`
  - In this directory, run python in an interactive session by running the command `python3`. At the prompt run:

```
import OpenOPC
opc = OpenOPC.open_client(host="PRIVATE IP ADDRESS OF OPC DA SERVER")
```

  - When there is no connection, the attempt stalls, or you receive an error. There is an issue with connectivity between your OPC DA server and your AWS IoT Greengrass industrial gateway.

Update the stack

Complete the following steps to update your AWS CloudFormation stack to the current version.

1. Sign in to the AWS CloudFormation console.
2. Select your existing Machine to Cloud Connectivity Framework CloudFormation stack, and choose **Update**.
3. On the **Update stack** page, verify that **Replace current template** is selected.
   - In the **Specify template** section, select **Amazon S3 URL**.
   - Copy the link of the latest template.
   - Paste the link in the Amazon S3 URL box.
   - Verify that the correct template URL shows in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, under **Parameters**, review the parameters for the template and modify them as necessary. Refer to Step 1. Launch the stack (p. 18) for details about the parameters.
5. Choose **Next**.
6. On the **Configure stack options** page, choose **Next**.
7. On the **Review** page, review and confirm the settings. Check the box acknowledging that the template might create AWS Identity and Access Management (IAM) resources.
8. Choose **View change set** and verify the changes. This may take a few minutes, but you can view the updated/modified/deleted changes.
9. Choose **Update stack** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive an **UPDATE_COMPLETE** status in approximately five minutes.

Uninstall the solution

You can uninstall the Machine to Cloud Connectivity Framework solution from the AWS Management Console or by using the AWS Command Line Interface. You must manually delete the Amazon Simple Storage Service (Amazon S3) bucket created by this solution. AWS Solutions Implementations do not automatically delete this resource in case you have stored data to retain.

Using the AWS Management Console

1. Sign in to the AWS CloudFormation console.
2. Select this solution's installation stack.
3. Choose **Delete**.

Deleting the Amazon S3 buckets

This solution is configured to retain the solution-created Amazon S3 buckets if you decide to delete the AWS CloudFormation stack to prevent accidental data loss. After uninstalling the solution, you can manually delete the S3 buckets if you do not need to retain the data. Follow these steps to delete the S3 buckets.

1. Navigate to the Amazon S3 console.
2. Choose **Buckets** from the left navigation pane.
3. In the **Find buckets by name** field, enter the name of this solution's stack.
4. Select one of the solution's S3 buckets and choose **Empty**.
5. Enter **permanently delete** in the verification field and choose **Empty**.
6. Choose the S3 bucket name you just emptied and choose **Delete**.
7. Enter the S3 bucket name in the verification field and choose **Delete bucket**.

Repeat steps 4 through 7 until you delete all the S3 buckets.

Using AWS Command Line Interface

Determine whether the AWS Command Line Interface (AWS CLI) is available in your environment. For installation instructions, refer to What Is the AWS Command Line Interface in the *AWS CLI User Guide*. After confirming that the AWS CLI is available, run the following commands.

```
$ aws cloudformation delete-stack --stack-name <installation-stack-name>

$ aws s3 ls | grep <installation-stack-name>

$ bucket=<your_bucket> # You may need to do this one multiple times

$ aws s3api delete-objects --bucket ${bucket} \
--delete "$(aws s3api list-object-versions --bucket ${bucket} \
--query='{Objects: Versions[].{Key:Key,VersionId:VersionId}}')"

$ aws s3api delete-objects --bucket ${bucket} \
```

```
--delete "$(aws s3api list-object-versions --bucket ${bucket} \
--query='{Objects: DeleteMarkers[].{Key:Key,VersionId:VersionId}}')"

$ aws s3 rb s3://${bucket} -force
```

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 activated, the following information is collected and sent to AWS when the job control is published to `m2c2/job/request` AWS IoT topic:

- **Solution ID:** The AWS solution identifier
- **Unique ID (UUID):** Randomly generated, unique identifier for each Machine to Cloud Connectivity Framework deployment
- **Timestamp:** Data-collection timestamp
- **Stack Version:** The version of the stack currently running the solution
- **Event Type:** The job control event (`deploy`, `start`, `stop`, `update`, `push`, or `pull`)
- **Job Details:** The detail information of the job. Example data:
  - OPC Data Access Protocol:

```
{
"protocol": "opcda",
"machine-query-time-interval": 1,
"machine-query-iterations": 3,
"number-of-lists": 1,
"number-of-tags": 0
}
```

  - Seamless Messaging Protocol:

```
{
"protocol": "slmp",
"machine-query-time-interval": 1,
"machine-query-iterations": 3,
"details": {
  "device_read": 1,
  "device_read_random": 0,
  "array_label_read": 0,
  "label_read_random": 1
}
}
```

The following information is collected and sent to AWS at stack creation, update or deletion:

- **Solution ID:** The AWS solution identifier
- **Unique ID (UUID):** Randomly generated, unique identifier for each Machine to Cloud Connectivity Framework deployment
- **Timestamp:** Data-collection timestamp
- **Stack Version:** The version of the stack that is created, updated, or deleted
- **Region:** The AWS Region where the stack is created, updated, or deleted
- **Event Type:** The stack event (`DeployStack`, `Updatestack`, or `DeleteStack`)
- **Using Existing Greengrass Group:** The flag if an existing Greengrass group is used
- **Using Existing Kinesis Data Stream:** The flag if an existing Amazon Kinesis Data Stream is used

AWS owns the data gathered though this survey. Data collection is subject to the AWS Privacy Policy. To opt out of this feature, modify the AWS CloudFormation template mapping section as follows:

```
Metrics:
  General:
    SendAnonymousUsageData: "Yes"
```

to

```
Metrics:
  General:
    SendAnonymousUsageData: "No"
```

Source code

Visit our GitHub repository to download the source files for this solution and to share your customizations with others. The Machine to Cloud Connectivity Framework template is generated using the AWS Cloud Development Kit (CDK) (AWS CDK). Refer to the README.md file for additional information.

Revisions

Date	Change
August 2019	Initial release
October 2019	Added information about support for Mitsubishi Seamless Messaging Protocol
October 2020	Release version 2.1.0: For the OPC DA connector only: added configuration for Amazon Kinesis Data Firehose and Amazon S3 to store the data from the AWS IoT Core topics; for more information, refer to the CHANGELOG.md file in the GitHub repository
March 2021	Release version 2.2.0: For the OPC DA connector only: added default optional configuration to send data to Amazon Kinesis Data Stream, Amazon Kinesis Firehose and Amazon S3, automating the creation of AWS IoT Greengrass resources and edge device configuration to help customers more easily set up and configure their edge device, and ability to activate and deactivate sending data to the AWS IoT Core topics. For information about updates and changes to this solution, refer to the CHANGELOG.md file in the GitHub repository.

Contributors

- Nimay Mehta
- Thibaut Grandmougin
- Erin McGill
- Beomseok Lee

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 AWS current 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. AWS responsibilities and liabilities 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.

Machine to Cloud Connectivity Framework is licensed under the terms of the of the Apache License Version 2.0 available at The Apache Software Foundation.