Machine to Cloud Connectivity Framework

AWS Implementation Guide

Nimay Mehta

Thibaut Grandmougin

August 2019

Last updated: October 2019 (see revisions)

Contents

About This Guide

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

Overview

Amazon Web Services (AWS) enables manufacturers to build serverless industrial IoT applications that gather, process, analyze, and act on factory equipment data, without having to manage any infrastructure.

With AWS IoT, you can connect factory equipment such as PLCs and CNC machines to the AWS Cloud securely, with low latency and with low overhead. You can also combine AWS IoT with other AWS services to build event-driven applications that help provide insights that increase your operational efficiency, and accelerate your pace of innovation.

Machine to Cloud Connectivity Framework is a solution that provides secure equipment connectivity to the AWS Cloud. The solution features fast and robust data ingestion; highly reliable and durable storage of equipment data; and serverless event-driven applications that help manage the factory configuration. The solution is easy to deploy, and can help drive an increase in plant efficiency and uptime, improve production flexibility, and identify new business opportunities.

By default, this solution connects with equipment that uses the OPC Data Access (OPC DA) protocol or the Seamless Messaging Protocol (SLMP) promoted by CC-Link Partner Association (CLPA). But you can use this solution as a reference to build connectors for the protocols your factory equipment uses.

This solution is designed to provide a framework for connected 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 in AWS IoT Core to your own data lake or visualization tools, run machine learning models on the data for predictive maintenance, or create notifications and alerts.

Cost

You are responsible for the cost of the AWS services used while running this reference deployment. The total cost to run this solution depends on the amount of data being simulated, collected, stored, processed, and presented. 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.

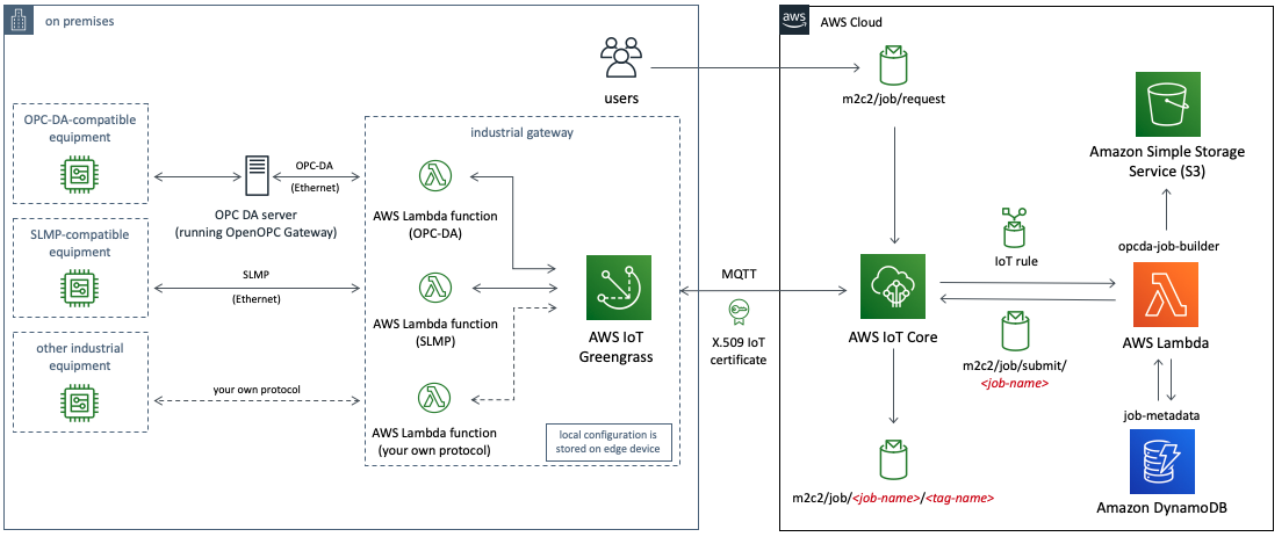


Figure 1: Machine to Cloud Connectivity Framework architecture on AWS

Machine to Cloud Connectivity Framework leverages AWS IoT Core which lets connected factory equipment easily and securely interact with cloud applications and other equipment. The solution's AWS CloudFormation template deploys AWS IoT Core topics, AWS IoT Greengrass, and AWS Lambda functions to manage communication between factory equipment and the AWS Cloud. The template also deploys Amazon DynamoDB to store job metadata, and Amazon Simple Storage Service (Amazon S3) to store job files.

The user creates a job request that contains information about the equipment, the data they want to read from the equipment, and the frequency at which they want to read the data. A Lambda function (`job builder`) translates the request into machine-readable format and pushes the translated request to the connected device and stores the request on the device gateway running AWS IoT Greengrass. A Lambda function (`protocol convertor`) running in AWS IoT Greengrass reads the translated request, establishes a connection between the device gateway and the equipment, gets the data defined in the request, and sends it to a topic in AWS IoT Core.

The data is translated to human-readable format and posted to the `job/<job-name>/<tag-name>` topic in AWS IoT Core. You can then specify what action you want to perform on the data using the Rules Engine in AWS IoT Core.

Solution Components

Job Definitions

When you want to extract data from your connected factory equipment, you create a job file that contains the details about the job including connectivity and data parameters. A job file contains both generic and protocol-specific properties. The job file 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 you want to submit. This property is used to determine whether the job is new or an update to an existing job.
version	String	Yes	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, not `"2.0"`. Only the latest version of the job will be executed at the gateway.
control	Enum	Yes	All	Enables you to control the running of the job. Allowed values are `"deploy"`, `"push"`, `"start"`, `"stop"`, `"update"`, and `"pull"`.
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, `"inbound"`.
process	String	No	All	The name of the process where the machine is located. For example, `"packaging"`.
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"`.

Property	Type	Required	Protocol(s)	Description
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.
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 `"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, see Machine Queries.
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, see Machine Queries.

Property	Type	Required	Protocol(s)	Description
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
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, see Sample SLMP Job File.

```
{
  "job": {
    "gg-group-id": String
    "properties": [
        {
            "name": String,
            "version": String
        },
        ...
        {
            "name": String,
            "version": String
```

```
        }
      ]
      "control": String,
      "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 will occur 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`. 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 enable you to have full control over the job that is running on each machine. The solution supports the following commands.

- **Deploy:** This command deploys the job file to the device 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. Note that this command will stop the running job and restart it with the updated parameters. Make sure you populate all protocol-specific properties correctly.
- **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.

Job Builder

When you submit a job request, the submission triggers the `job builder` AWS Lambda function. The `job builder` 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.

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

The `job builder` function also checks the job metadata stored in Amazon DynamoDB to see if the job name already exists. If the job name does not exist, the function creates a new job file and stores it in Amazon Simple Storage Service (Amazon S3). The function adds the new job name to the metadata stored in DynamoDB, creates a `machine connector`

Lambda function to handle the job, and updates the job resource file variable. 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 `/job/<machinename>/<tag-name>` as the target topic.

If the job name does exist, the function updates the job file in Amazon S3 and updates the version number in the job metadata in DynamoDB.

When a job file is created or updated, the solution sends a message to the `job/submit/<job-name>` topic which triggers the `protocol convertor` Lambda function.

Machine Connector

The `machine connector` Lambda function is provisioned by the `job builder` function, and is deployed to the edge gateway when the Greengrass group is deployed. The `machine connector` 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. The function also manages the connectivity with AWS IoT Core to send data to the cloud, and reads the latest running job and sends it back to the cloud.

The `machine connector` function sends the data to the `job/<job-name>/<tag-name>` topic in AWS IoT Core. Error messages are sent to the `job/<job-name>/error` topic and information message are sent to the `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 trigger notifications using Amazon Simple Notification Service (Amazon SNS).

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 Seamless Messaging Protocol (SLMP) promoted by CC-Link Partner Association (CLPA). 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 deploy this solution, you must have an AWS IoT Greengrass-supported gateway in your environment that is connected to the OPC DA server. For more information, see Supported Platforms and Requirements in the *AWS IoT Greengrass Developer Guide*. To set up Greengrass on your gateway, see Environment Setup for Greengrass.

AWS IoT Greengrass Group

This solution enables you to use an existing AWS IoT Greengrass group. Or, the solution can create one for you. A 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.

X.509 Certificates

Communication between your factory equipment and AWS IoT is protected through the use of X.509 certificates. AWS IoT can generate a certificate for you or you can use your own X.509 certificate. You can register your preferred Certificate Authority (CA), which is used to sign and issue the device certificate, with AWS IoT.

AWS IoT Rules Engine

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, see Rules for AWS IoT in the *AWS IoT Core User Guide*.

Error Logging

This 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.

Stack Updates

To update the solution stack after initial deployment, in the AWS CloudFormation console, select the Machine to Cloud Connectivity Framework stack and select **Update**. Choose whether to use or replace the current template. Modify the parameters, if necessary, and deploy the stack.

Regional Deployment

This solution uses AWS IoT Greengrass which is available in specific AWS Regions only. Therefore, you must deploy this solution in a region that supports AWS IoT Greengrass. For the most current service availability by region, see AWS service offerings by region.

AWS CloudFormation Template

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

View template

machine-to-cloud-connectivity-framework: Use this template to launch the solution and all associated components. The default configuration deploys AWS IoT Core, AWS Lambda functions, an Amazon Simple Storage Service (Amazon S3) bucket, and an Amazon DynamoDB table, but you can also customize the template based on your specific needs.

Automated Deployment

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

Time to deploy: Approximately 10 minutes

Prerequisites

Before you start, you must have an AWS IoT Greengrass-supported gateway that is connected to an OPC DA server in your environment. To set up Greengrass on your gateway, see Environment Setup for Greengrass and Installing the Greengrass Core Software.

> **Note:** You must configure Greengrass before you start it.

Configure Your Industrial Gateway

After your gateway is set up to run Greengrass, use the following procedure to configure the gateway to work with this solution.

1. In the root directory of your device gateway, use the following commands to create an `m2c2` folder structure.

```
cd /
mkdir m2c2
cd m2c2
mkdir job
```

2. In the root directory, use the following command to give the Greengrass user access to the solution folders.

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

Configure Your X.509 Certificate

You must have an activated X.509 certificate before you launch this solution. If you want the solution to create a new Greengrass group, enter the certificate Amazon Resource Name (ARN) in the **Greengrass Core Certificate ARN** AWS CloudFormation template parameter during deployment.

You can use your own X.509 certificate. 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.

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

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

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

Step 2. Configure and Start the AWS IoT Greengrass Core

- Configure the AWS IoT Greengrass Core and start the core.

Step 3. Deploy a Job

- Create a job file and submit the job request to AWS IoT Core.

Step 4. Deploy the Greengrass Group to your Gateway

- Deploy the AWS IoT Greengrass group to your device gateway.

Step 5. Test Connectivity (Optional)

- If you use the OPC or SLMP protocols, you can test your connectivity between your gateway and the factory equipment.

Step 6. Start a Job

- Start a job.

Step 1. Launch the Stack

This automated AWS CloudFormation template deploys Machine to Cloud Connectivity Framework in the AWS Cloud. Verify that you have completed the prerequisites before you launch the stack.

> **Note**: You are responsible for the cost of the AWS services used while running this solution. See the Cost 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 to the right to launch the `machine-to-cloud-connectivity-framework` AWS CloudFormation template.

   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 Greengrass, which is currently available in specific AWS Regions only. Therefore, you must launch this solution in an AWS Region where AWS IoT Greengrass is available. For the most current 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
Existing Greengrass Group	`false`	Choose whether to use an existing AWS IoT Greengrass group **Note:** If you do not have an existing Greengrass group, leave this parameter set to `false`. The solution will create a new group, thing, and resources, and link the certificate to the group.
Existing Greengrass Group ID	<Optional input>	If you use an existing Greengrass group, specify the ID. You can find the ID using the console or the AWS CLI. **Note:** If **Existing Greengrass Group** is set to `false`, leave this parameter blank.
Greengrass Group Name	<Optional input>	If you want the solution to create a Greengrass group, specify a name for the group. **Note:** If **Existing Greengrass Group** is set to `true`, leave this parameter blank.
Device Gateway Name	<Optional input>	If you want the solution to create a Greengrass group, specify a name for the device gateway that will be linked to the group. **Note:** If **Existing Greengrass Group** is set to `true`, leave this parameter blank.
Greengrass Core Certificate ARN	*<Requires input>*	Specify the X.509 certificate ARN. For information, see Configure Your X.509 Certificate.

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 10 minutes.

Step 2. Configure and Start the AWS IoT Greengrass Core

If you use this solution to create a new Greengrass group during initial deployment, you must update the `config.json` file on your industrial gateway to configure the AWS IoT Greengrass Core. For information about how to configure the core, see Configure the AWS IoT Greengrass Core in the *AWS IoT Greengrass Developer Guide*.

You can use the following command to view the contents of the `config.json` file.

```
cat /<greengrass-root>/config/config.json
```

Note that the solution will generate an Amazon Resource Name (ARN) for your thing. You must update the `config.json` file with the thing ARN.

After you configure the Greengrass Core, use the following commands to start the core.

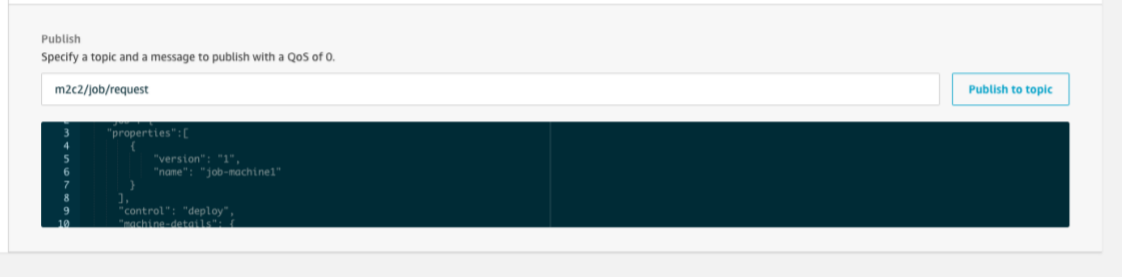
```
cd /greengrass/ggc/core
sudo ./greengrassd start
```

Step 3. Deploy a Job

1. Create a job file. For more information on the job file, see Job Definitions.
2. In the job file, set the **control** property to `deploy`.

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

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



7. Select **Publish to topic**.

After the deploy command is executed, a message will be posted to the `m2c2/job/<job-name>` topic. It can take up to 10 seconds for the command to be executed and an acknowledgment message to be sent to the topic.

The message will be similar to the following sample message.

```
Job successfully created. Deploy the Greengrass Group from the AWS IoT
console and use the push control to check connectivity.
```

Step 4. Deploy the Greengrass Group to your Gateway

Use the following procedure to deploy the AWS IoT Greengrass group to your gateway.

> **Note:** You must complete this procedure for every new job you submit. You do not need to complete this procedure for updates to an existing job.

1. Verify that all the X.509 certificates you created are deployed to your gateway and that Greengrass is running on your gateway.
2. Navigate to the AWS IoT Greengrass console.
3. In the navigation pane under **Greengrass**, select **Groups**.
4. Select the applicable group.
5. For **Actions**, choose **Deploy**.

> **Note:** If you deploy a new AWS IoT Greengrass group to your gateway, all running jobs on the gateway will be terminated. You must manually start all the jobs running on the gateway. Use the start control command to start the jobs.

Step 5. Test Connectivity (Optional)

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

1. Navigate to the AWS IoT Core 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 will return a message to the `m2c2/job/<job-name>/info` topic. The message will vary depending on the protocol you use. For OPC DA, see Appendix B. For SLMP, see Appendix C.

Step 6. Start a Job

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

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

After the job is started, the `machine connector` Lambda function on the gateway will start communicating with the OPC DA server. The function will publish data back to the `m2c2/job/<job-name>/<tag-name>` topic in AWS IoT Core.

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.

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. 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.

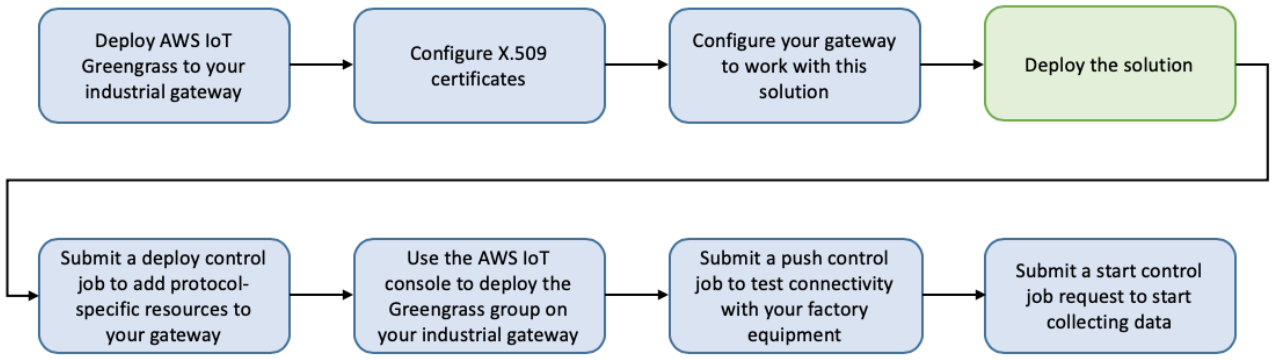
Additional Resources

AWS services

- AWS CloudFormation
- Amazon Simple Storage Service
- AWS Lambda
- AWS IoT Core
- AWS IoT Greengrass
- Amazon DynamoDB

Appendix A: Solution Processes

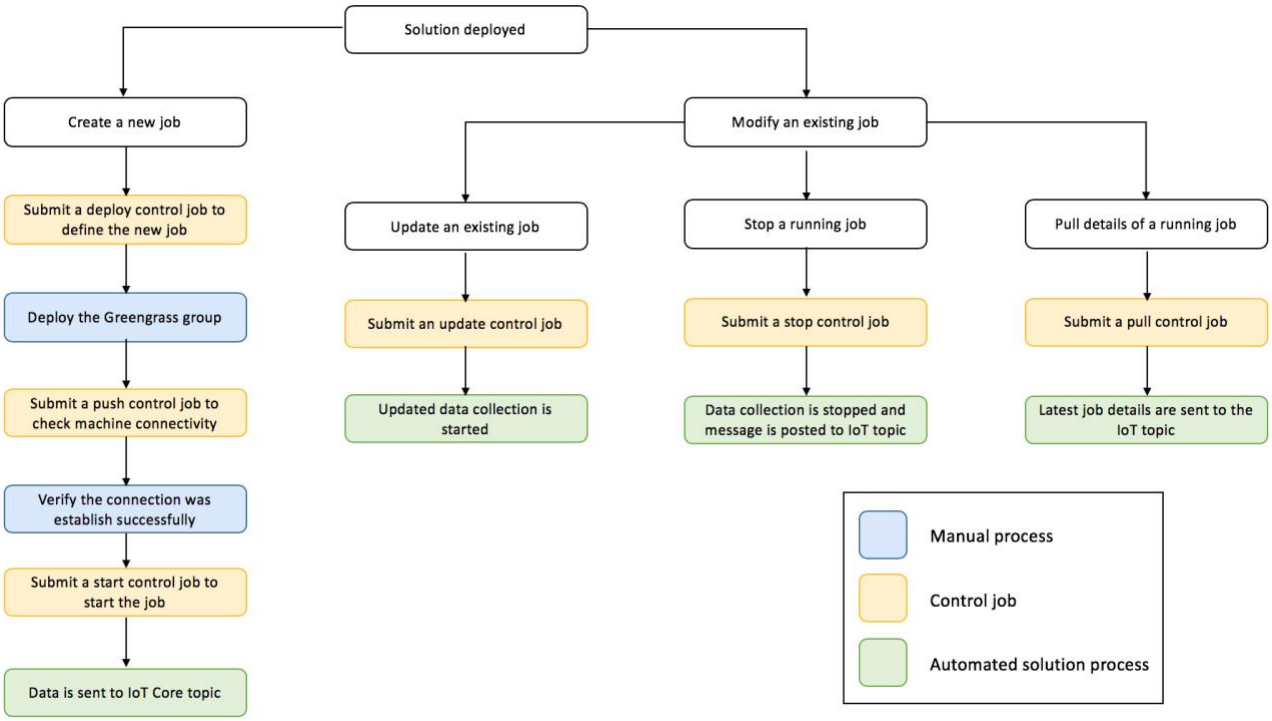
The following flowchart shows the process for preparing your environment and deploying the Machine to Cloud Connectivity Framework.



To prepare your environment, you must do the following:

- Deploy AWS IoT Greengrass to a Linux machine. This machine will be your gateway.
- Configure your X.509 certificate.
- Configure your gateway to work with the solution.
- Deploy the solution's AWS CloudFormation template.
- Deploy a job.

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



To create a new job, you must do the following:

- Define the job and submit it. For OPC DA, see Appendix B. For SLMP, see Appendix C.
- Manually deploy the AWS IoT Greengrass group to your gateway.
- Check the connectivity between your machine and your gateway.
- Verify that the connection was successful.
- Start the job.

After the job is started, you can use the update, stop, and pull controls to modify the job.

> **Note:** When you create or update a job using the deploy or update controls, but do not specify a Greengrass group ID, the solution will use the ID you specified during initial deployment. The solution might return the message `Parameter gg-group-id not found in the job request`. But, the solution will create the job with the default Greengrass group ID.

Appendix B: OPC Data Access

The Machine to Cloud Connectivity Framework currently 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.

Configure Your OPC DA Server

To use the OPC DA protocol, you must configure an OPC DA server to allow the solution's `machine connector` AWS Lambda function to connect to it. Use the following procedure to configure the server.

1. Download OpenOPC for Python.
2. Use the following command to install OpenOPC.

```
C:\OpenOPC\bin> OpenOPCService.exe -install
```

3. Use the following command to start OpenOPC.

```
C:\OpenOPC\bin> net start zzzOpenOpcService
```

To stop the service, use the following command.

```
C:\OpenOPC\bin> net stop zzzOpenOpcService
```

Connectivity

To check the connectivity of the OPC DA server, submit a push control job. If the connection is successful, the Lambda function on the gateway will return a message containing a server list to the `m2c2/job/<job-name>/info` topic. The message will be similar to the following sample message.

```
{
    "message": "Available server:  [u 'Matrikon.OPC.Simulation.1']",
    "version": "1",
    "_id_": "a10d3db5-367b-4dff-aac2-fec28134ebd6",
    "_timestamp_": "2019-09-26 18:42:18.665823"
}
```

> **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.

Troubleshooting

If the connection fails, verify the following:

- The OpenOPC gateway is running on the OPC DA server
- Port 7776 (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

Sample OPC-DA Job File

Once the solution has established connectivity with the OPC DA server, submit a job request using a job file. The following example shows an OPC DA job file to connect to the Matrikon OPC simulator.

```
{
  "job": {
    "control": "deploy",
    "properties": [
      {
        "version": "2",
        "name": "job-machine1"
      }
    ],
    "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*"
            ]
          }
        ]
      },
      "connectivity-parameters": {
        "opcda-server-name": "Matrikon.OPC.Simulation.1",
        "machine-ip": "169.254.89.33",
        "protocol": "opcda",
        "machine-name": "machine 1"
      }
```

```
        }
      }
    }
```

Appendix C: Seamless Messaging Protocol

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

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.
- You may need to whitelist your industrial gateway IP in your PLC.

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:** read data from a label of an array type.
- **Label read random:** read 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 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 will return a message to the `m2c2/job/<job-name>/info` topic that confirms whether TCP or UDP communication can occur.

Sample SLMP Job File

Once the solution has established connectivity, submit a job request using a job file. The following example shows a job file 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 job file 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"
      }
    }
}
```

Appendix D: 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 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:

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

to

```
Metrics:
   General:
      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
August 2019	Initial release
October 2019	Added information about support for Mitsubishi Seamless Messaging Protocol