# Table of Contents

What Is AWS Greengrass? .................................................................................................................. 1
Greengrass Core Software .................................................................................................................. 2
AWS Greengrass Groups ...................................................................................................................... 3
Devices in AWS Greengrass .................................................................................................................. 4
SDKs .................................................................................................................................................. 5
Supported Platforms and Requirements ............................................................................................. 9

Getting Started with AWS Greengrass ............................................................................................... 12
Requirements .................................................................................................................................... 12
Create an AWS Account ..................................................................................................................... 12

Module 1: Environment Setup for Greengrass ................................................................................ 12
  Setting Up a Raspberry Pi ................................................................................................................... 13
  Setting Up an Amazon EC2 Instance ................................................................................................. 16
  Setting Up Other Devices ............................................................................................................... 19

Module 2: Installing the Greengrass Core ......................................................................................... 20
  Configuring AWS Greengrass on AWS IoT ..................................................................................... 21
  Starting AWS Greengrass on a Device ............................................................................................. 25

Module 3 (Part 1): AWS Lambda on AWS Greengrass .................................................................... 31
  Create and Package a Lambda Function .......................................................................................... 31
  Configure Lambda for AWS Greengrass .......................................................................................... 38
  Deploying Cloud Configurations to an AWS Greengrass Core Device ........................................... 44
  Verify the Lambda Function Is Running on the Device .................................................................. 45

Module 3 (Part 2): AWS Lambda on AWS Greengrass .................................................................... 48
  Create and Package the Lambda Function ....................................................................................... 48
  Configure Long-Lived Lambda Functions for AWS Greengrass ...................................................... 49
  Testing Long-Lived Lambda Functions ............................................................................................ 50
  Testing On-Demand Lambda Functions .......................................................................................... 54

Module 4: Interacting with Devices in an AWS Greengrass Group .................................................... 58
  Create AWS IoT Devices in an AWS Greengrass Group ................................................................. 59
  Configure Subscriptions .................................................................................................................. 62
  Install the AWS IoT Device Python SDK ....................................................................................... 63
  Testing Communications .................................................................................................................. 64

Module 5: Interacting with Device Shadows ..................................................................................... 65
  Configure Devices and Subscriptions ............................................................................................... 66
  Download Required Files ................................................................................................................ 68
  Test Communications (Device Syncs Disabled) ............................................................................. 69
  Test Communications (Device Syncs Enabled) ................................................................................. 70

Module 6: Accessing AWS Cloud Services ....................................................................................... 72
  Configure IAM Roles ........................................................................................................................ 73
  Create and Configure Lambda ......................................................................................................... 77
  Configure Subscriptions .................................................................................................................. 80
  Test Communications ...................................................................................................................... 82

OTA Updates of AWS Greengrass Core Software ............................................................................. 84
  Greengrass OTA Agent ..................................................................................................................... 85
  Integration with Init systems ............................................................................................................ 87
  OTA Self-Update with Managed Respawn ....................................................................................... 88
  AWS Greengrass core Update with Managed Respawn .................................................................. 88

OTA Agent Self-Update ....................................................................................................................... 88
Greengrass Core Software Update .................................................................................................... 89
Limitations ......................................................................................................................................... 89

Reset Deployments .......................................................................................................................... 90
Access Local Resources with Lambda Functions ................................................................................ 93
Prerequisites ....................................................................................................................................... 93
How to Configure Local Resource Access ......................................................................................... 93
Troubleshooting ................................................................................................................................. 97

---

AWS Greengrass Developer Guide

---
What Is AWS Greengrass?

AWS Greengrass is software that extends AWS cloud capabilities to local devices, making it possible for them to collect and analyze data closer to the source of information, while also securely communicating with each other on local networks. More specifically, developers who use AWS Greengrass can author serverless code (AWS Lambda functions) in the cloud and conveniently deploy it to devices for local execution of applications.

The following diagram shows the basic architecture of AWS Greengrass.

AWS Greengrass makes it possible for customers to use Lambda functions to build IoT devices and application logic. Specifically, AWS Greengrass provides cloud-based management of applications that can be deployed for local execution. Locally deployed Lambda functions are triggered by local events, messages from the cloud, or other sources.

In AWS Greengrass, devices securely communicate on a local network and exchange messages with each other without having to connect to the cloud. AWS Greengrass provides a local pub/sub message manager that can intelligently buffer messages if connectivity is lost so that inbound and outbound messages to the cloud are preserved.

AWS Greengrass protects user data:

- Through the secure authentication and authorization of devices.
- Through secure connectivity in the local network.
- Between local devices and the cloud.

Device security credentials function within a group until they are revoked, even if connectivity to the cloud is disrupted, so that the devices can continue to securely communicate locally.

AWS Greengrass provides secure, over-the-air software updates of Lambda functions.

AWS Greengrass consists of:

- Software distributions
AWS Greengrass Developer Guide
Greengrass Core Software

- AWS Greengrass core software
- AWS Greengrass core SDK
- Cloud service
  - AWS Greengrass API
- Features
  - Lambda runtime
  - Thing shadows implementation
  - Message manager
  - Group management
  - Discovery service

Greengrass Core Software

The AWS Greengrass core software provides the following functionality:

- Allows deployment and execution of local applications created using Lambda functions and managed through the deployment API.
- Enables local messaging between devices over a secure network using a managed subscription scheme through the MQTT protocol.
- Ensures secure connections between devices and the cloud using device authentication and authorization.
- Provides secure, over-the-air software updates of user-defined Lambda functions.

The AWS Greengrass core software consists of:

- A message manager that routes messages between devices, Lambda functions, and AWS IoT.
- A Lambda runtime that runs user-defined Lambda functions.
- An implementation of the Thing Shadows service that provides a local copy of thing shadows, which represent your devices. Thing shadows can be configured to sync with the cloud.
- A deployment agent that is notified of new or updated AWS Greengrass group configuration. When new or updated configuration is detected, the deployment agent downloads the configuration data and restarts the AWS Greengrass core.

AWS Greengrass core instances are configured through AWS Greengrass APIs that create and update AWS Greengrass group definitions stored in the cloud.

AWS Greengrass core versions:

GCC v1.3.0

Current version.

v1.1.0 and v1.3.0 have the same directory structure.

There is a new folder containing the OTA (Over-The-Air) Agent under the /greengrass/ota/ota-agent/ggc-ota directory.

GCC v1.1.0

To migrate from the previous version of the AWS Greengrass core:
AWS Greengrass Groups

An AWS Greengrass group definition is a collection of settings for AWS Greengrass core devices and the devices that communicate with them. The following diagram shows the objects that make up an AWS Greengrass group.

In the preceding diagram:
A: AWS Greengrass group definition
   A collection of information about the AWS Greengrass group.
B: AWS Greengrass group settings
   These include:
   • AWS Greengrass group role.
   • Log configuration.
   • Certification authority and local connection configuration.
   • AWS Greengrass core connectivity information.
 Devices in AWS Greengrass

There are two types of devices:

- AWS Greengrass cores.
- AWS IoT devices connected to an AWS Greengrass core.

An AWS Greengrass core is an AWS IoT device that runs specialized AWS Greengrass software that communicates directly with the AWS IoT and AWS Greengrass cloud services. It is an AWS IoT device with its own certificate used for authenticating with AWS IoT. It has a device shadow and exists in the AWS IoT device registry. AWS Greengrass cores run a local Lambda runtime, a deployment agent, and an IP address tracker that sends IP address information to the AWS Greengrass cloud service to allow AWS IoT devices to automatically discover their group and core connection information.

Any AWS IoT device can connect to an AWS Greengrass core. An AWS Greengrass core runs software written with the AWS IoT Device SDK.

The following table shows how these device types are related.
SDKs

The following SDKs are used when working with AWS Greengrass:

GGC v1.3.0

AWS SDKs

Using the AWS SDKs, you can build applications that work with any AWS service, including Amazon S3, Amazon DynamoDB, AWS IoT, AWS Greengrass, and more. In the context of AWS Greengrass, you can use the AWS SDK inside deployed Lambda functions to make direct calls to any AWS service.

AWS IoT Device SDKs

The AWS IoT Device SDKs helps you connect your device to AWS IoT or AWS Greengrass services. Devices must know to which AWS Greengrass group they belong and the IP address of the AWS Greengrass core to which they should connect.

Although you can use any of the AWS IoT Device SDKs to connect to an AWS Greengrass core, only the C++ and Python Device SDKs provide AWS Greengrass-specific functionality, such as

<table>
<thead>
<tr>
<th></th>
<th>Core</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Certificate</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IoT Policy</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>IoT Thing</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Device use sample</td>
<td>Gateway</td>
<td>Sensor and/or Actuator</td>
</tr>
<tr>
<td>Software</td>
<td>Greengrass Core Software</td>
<td>AWS IoT Device SDK</td>
</tr>
<tr>
<td>Group membership</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Functions outside an AWS Greengrass Group</td>
<td>✗</td>
<td>✓</td>
</tr>
</tbody>
</table>
access to the AWS Greengrass Discovery Service and AWS Greengrass core root CA downloads. For more information, see AWS IoT Device SDK.

AWS Greengrass Core SDK

The AWS Greengrass Core SDK enables Lambda functions to interact with the AWS Greengrass core on which they run in order to publish messages, interact with the local Thing Shadows service, or invoke other deployed Lambda functions. This SDK is used exclusively for writing Lambda functions running in the Lambda runtime on an AWS Greengrass core. Lambda functions running on an AWS Greengrass core can interact with AWS cloud services directly using the AWS SDK. The AWS Greengrass Core SDK and the AWS SDK are contained in different packages, so you can use both packages simultaneously. You can download the AWS Greengrass Core SDK from the Software section of the AWS IoT console.

The AWS Greengrass Core SDK follows the AWS SDK programming model. It allows you to easily port Lambda functions developed for the cloud to Lambda functions that run on an AWS Greengrass core. For example, using the AWS SDK, the following Lambda function publishes a message to the topic "/some/topic" in the cloud:

```python
import boto3
client = boto3.client('iot-data')
response = client.publish(
    topic = "/some/topic",
    qos = 0,
    payload = "Some payload".encode()
)
```

To port this Lambda function for execution on an AWS Greengrass core, replace the `import boto3` statement with the `import greengrasssdk`, as shown in the following snippet:

```python
import greengrasssdk
client = greengrasssdk.client('iot-data')
response = client.publish(
    topic='/some/topic',
    qos=0,
    payload='some payload'.encode()
)
```

This allows you to test your Lambda functions in the cloud and migrate them to AWS Greengrass with minimal effort.

**Note**

The AWS Greengrass Core SDK only supports sending MQTT messages with QoS = 0. The AWS SDK is natively part of the environment when executing a Lambda function in the AWS cloud. If you want to use `boto3` in a Lambda function deployed on an AWS Greengrass core, make sure to include the AWS SDK in your package. In addition, if you choose to use both the AWS Greengrass Core SDK and the AWS SDK simultaneously in the same package, your Lambda functions must use the correct namespace. For more information about how to create your deployment package, see:

- AWS Lambda Creating a Deployment Package (Python)
- AWS Lambda Creating a Deployment Package (NodeJS)
- AWS Lambda Creating a Deployment Package (Java)
AWS Greengrass Developer Guide

SDKs

GGC v1.1.0

AWS SDKs

Using the AWS SDKs, you can build applications that work with any AWS service, including Amazon S3, Amazon DynamoDB, AWS IoT, AWS Greengrass, and more. In the context of AWS Greengrass, you can use the AWS SDK inside deployed Lambda functions to make direct calls to any AWS service.

AWS IoT Device SDKs

The AWS IoT Device SDKs helps you connect your device to AWS IoT or AWS Greengrass services. Devices must know to which AWS Greengrass group they belong and the IP address of the AWS Greengrass core to which they should connect.

Although you can use any of the AWS IoT Device SDKs to connect to an AWS Greengrass core, only the C++ and Python Device SDKs provide AWS Greengrass-specific functionality, such as access to the AWS Greengrass Discovery Service and AWS Greengrass core root CA downloads. For more information, see AWS IoT Device SDK.

AWS Greengrass Core SDK

The AWS Greengrass Core SDK enables Lambda functions to interact with the AWS Greengrass core on which they run in order to publish messages, interact with the local Thing Shadows service, or invoke other deployed Lambda functions. This SDK is used exclusively for writing Lambda functions running in the Lambda runtime on an AWS Greengrass core. Lambda functions running on an AWS Greengrass core can interact with AWS cloud services directly using the AWS SDK. The AWS Greengrass Core SDK and the AWS SDK are contained in different packages, so you can use both packages simultaneously. You can download the AWS Greengrass Core SDK from the Software section of the AWS IoT console.

The AWS Greengrass Core SDK follows the AWS SDK programming model. It allows you to easily port Lambda functions developed for the cloud to Lambda functions that run on an AWS Greengrass core. For example, using the AWS SDK, the following Lambda function publishes a message to the topic "/some/topic" in the cloud:

```python
import boto3
client = boto3.client('iot-data')
response = client.publish(
    topic = '/some/topic',
    qos = 0,
    payload = 'Some payload'.encode()
)
```

To port this Lambda function for execution on an AWS Greengrass core, replace the `import boto3` statement with the `import greengrasssdk`, as shown in the following snippet:

```python
import greengrasssdk
client = greengrasssdk.client('iot-data')
response = client.publish(
    topic= '/some/topic',
    qos=0,
    payload='some payload'.encode()
)
```

This allows you to test your Lambda functions in the cloud and migrate them to AWS Greengrass with minimal effort.

**Note**
The AWS Greengrass Core SDK only supports sending MQTT messages with QoS = 0. The AWS SDK is natively part of the environment when executing a Lambda function.
in the AWS cloud. If you want to use `boto3` in a Lambda function deployed on an AWS Greengrass core, make sure to include the AWS SDK in your package. In addition, if you choose to use both the AWS Greengrass Core SDK and the AWS SDK simultaneously in the same package, your Lambda functions must use the correct namespace. For more information about how to create your deployment package, see:

- AWS Lambda Creating a Deployment Package (Python)
- AWS Lambda Creating a Deployment Package (NodeJS)
- AWS Lambda Creating a Deployment Package (Java)

GGC v1.0.0

AWS SDKs

Using the AWS SDKs, you can build applications that work with any AWS service, including Amazon S3, Amazon DynamoDB, AWS IoT, AWS Greengrass, and more. In the context of AWS Greengrass, you can use the AWS SDK inside deployed Lambda functions to make direct calls to any AWS service.

AWS IoT Device SDKs

The AWS IoT Device SDKs helps you connect your device to AWS IoT or AWS Greengrass services. Devices must know to which AWS Greengrass group they belong and the IP address of the AWS Greengrass core to which they should connect.

Although you can use any of the AWS IoT Device SDKs to connect to an AWS Greengrass core, only the C++ and Python Device SDKs provide AWS Greengrass-specific functionality, such as access to the AWS Greengrass Discovery Service and AWS Greengrass core root CA downloads. For more information, see AWS IoT Device SDK.

AWS Greengrass Core SDK

The AWS Greengrass Core SDK enables Lambda functions to interact with the AWS Greengrass core on which they run in order to publish messages, interact with the local Thing Shadows service, or invoke other deployed Lambda functions. This SDK is used exclusively for writing Lambda functions running in the Lambda runtime on an AWS Greengrass core. Lambda functions running on an AWS Greengrass core can interact with AWS cloud services directly using the AWS SDK. The AWS Greengrass Core SDK and the AWS SDK are contained in different packages, so you can use both packages simultaneously. You can download the AWS Greengrass Core SDK from the Software section of the AWS IoT console.

The AWS Greengrass Core SDK follows the AWS SDK programming model. It allows you to easily port Lambda functions developed for the cloud to Lambda functions that run on an AWS Greengrass core. For example, using the AWS SDK, the following Lambda function publishes a message to the topic "/some/topic" in the cloud:

```python
import boto3
client = boto3.client('iot-data')
response = client.publish(
    topic = "/some/topic",
    qos = 0,
    payload = "Some payload".encode()
)
```

To port this Lambda function for execution on an AWS Greengrass core, replace the `import boto3` statement with the `import greengrasssdk`, as shown in the following snippet:

```python
import greengrasssdk
```
client = greengrasssdk.client('iot-data')
response = client.publish(
    topic='/some/topic',
    qos=0,
    payload='some payload'.encode()
)

This allows you to test your Lambda functions in the cloud and migrate them to AWS Greengrass with minimal effort.

Note
The AWS Greengrass Core SDK only supports sending MQTT messages with QoS = 0. The AWS SDK is natively part of the environment when executing a Lambda function in the AWS cloud. If you want to use boto3 in a Lambda function deployed on an AWS Greengrass core, make sure to include the AWS SDK in your package. In addition, if you choose to use both the AWS Greengrass Core SDK and the AWS SDK simultaneously in the same package, your Lambda functions must use the correct namespace. For more information about how to create your deployment package, see:
* AWS Lambda Creating a Deployment Package (Python)

---

**Supported Platforms and Requirements**

The AWS Greengrass core software is supported on the platforms listed below, and requires a few dependencies.

**GGC v1.3.0**

- **Supported platforms:**
  - Architecture: ARMv7l; OS: Linux; Distribution: Raspbian Jessie, 2017-03-02
  - Architecture: x86_64; OS: Linux; Distribution: Amazon Linux (amzn-amd64-2016.09.1.20170119-x86_64-ebs)
  - Architecture: x86_64; OS: Linux; Distribution: Ubuntu 14.04 – 16.04
  - Architecture: ARMv8 (AArch64); OS: Linux; Distribution: Ubuntu 14.04 – 16.04 (Annapurna Alpine V2)

- The following items are required:
  - Minimum 128 MB RAM allocated to the AWS Greengrass core device.
  - Linux kernel version 4.4 or greater: while several versions may work with AWS Greengrass, for optimal security and performance, we recommend version 4.4 or greater.
  - Glibc library version 2.14 or greater.
  - The `/var/run` directory must be present on the device.
  - AWS Greengrass requires hardlink and softlink protection to be enabled on the device. Without this, AWS Greengrass can only be run in insecure mode, using the `-i` flag.
  - The `ggc_user` and `ggc_group` user and group must be present on the device.
  - The following Linux kernel configurations must be enabled on the device:
    - Namespace: CONFIG_IPC_NS, CONFIG_UTS_NS, CONFIG_USER_NS, CONFIG_PID_NS
    - CGroups: CONFIG_CGROUP_DEVICE, CONFIG_CGROUPS, CONFIG_MEMCG
    - Others: CONFIG_POSIX_MQUEUE, CONFIG_OVERLAY_FS, CONFIG_HAVE_ARCH_SECCOMP_FILTER, CONFIG_SECCOMP_FILTER, CONFIG_KEYS, CONFIG_SECCOMP
  - The `sqlite3` package is required for AWS IoT device shadows. Ensure it's added to your `PATH` environment variable.
  - `/dev/stdin`, `/dev/stdout`, and `/dev/stderr` must be enabled.
• The Linux kernel must support **cgroups**.
• The *memory* cgroup must be enabled and mounted to allow AWS Greengrass to set the memory limit for Lambdas.
• The root certificate for Amazon S3 and AWS IoT must be present in the system trust store.

• The following items may be optional:
  • The *devices* cgroup must be enabled and mounted if Lambdas with **Local Resource Access (LRA)** (p. 93) are used to open files on the AWS Greengrass core device.
  • **Python** version 2.7 is required if Python Lambdas are used. If so, ensure that it's added to your **PATH** environment variable.
  • **NodeJS** version 6.10 or greater is required if Node.JS Lambdas are used. If so, ensure that it's added to your **PATH** environment variable.
  • **Java** version 8 or greater is required if Java Lambdas are used. If so, ensure that it's added to your **PATH** environment variable.
  • **OpenSSL** 1.01 or greater is required for **Greengrass OTA Agent** (p. 85) as well as the following commands: `wget`, `realpath`, `tar`, `readlink`, `basename`, `dirname`, `pidof`, `df`, `grep`, and `umount`.

GGC v1.1.0

• Supported platforms:
  • Architecture: ARMv7l; OS: Linux; Distribution: Raspbian Jessie, 2017-03-02
  • Architecture: x86_64; OS: Linux; Distribution: Amazon Linux (amzn-ami-hvm-2016.03.1.20170119-x86_64-ebs)
  • Architecture: x86_64; OS: Linux; Distribution: Ubuntu 14.04 – 16.04
  • Architecture: ARMv8 (AArch64); OS: Linux; Distribution: Ubuntu 14.04 – 16.04 (Annapurna Alpine V2)

• The following items are required:
  • Minimum 128 MB RAM allocated to the AWS Greengrass core device.
  • Linux kernel version 4.4 or greater: while several versions may work with AWS Greengrass, for optimal security and performance, we recommend version 4.4 or greater.
  • **Glibc library** version 2.14 or greater.
  • The `/var/run` directory must be present on the device.
  • AWS Greengrass requires hardlink and softlink protection to be enabled on the device. Without this, AWS Greengrass can only be run in insecure mode, using the `-i` flag.
  • The `ggc_user` and `ggc_group` user and group must be present on the device.
  • The following Linux kernel configurations must be enabled on the device:
    • Namespace: `CONFIG_IPC_NS`, `CONFIG_UTS_NS`, `CONFIG_USER_NS`, `CONFIG_PID_NS`
    • CGroups: `CONFIG_CGROUP_DEVICE`, `CONFIG_CGROUPS`, `CONFIG_MEMCG`
    • Others: `CONFIG_POSIX_MQUEUE`, `CONFIG_OVERLAY_FS`, `CONFIG_HAVE_ARCH_SECCOMP_FILTER`, `CONFIG_SECCOMP_FILTER`, `CONFIG_KEYS`, `CONFIG_SECCOMP`
  • The `sqlite3` package is required for AWS IoT device shadows. Ensure it's added to your **PATH** environment variable.
  • `/dev/stdin`, `/dev/stdout`, and `/dev/stderrmus` must be enabled.
  • The Linux kernel must support **cgroups**.
  • The *memory* cgroup must be enabled and mounted to allow AWS Greengrass to set the memory limit for Lambdas.
  • The root certificate for Amazon S3 and AWS IoT must be present in the system trust store.

• The following items may be optional:
• **Python** version 2.7 is required if Python Lambdas are used. If so, ensure that it's added to your **PATH** environment variable.

• **NodeJS** version 6.10 or greater is required if Node.JS Lambdas are used. If so, ensure that it's added to your **PATH** environment variable.

• **Java** version 8 or greater is required if Java Lambdas are used. If so, ensure that it's added to your **PATH** environment variable.

GGC v1.0.0

- **Supported platforms:**
  - Architecture: ARMv7l; OS: Linux; Distribution: Raspbian Jessie, 2017-03-02
  - Architecture: x86_64; OS: Linux; Distribution: Amazon Linux (amzn-ami-hvm-2016.09.1.20170119-x86_64-ebs)
  - Architecture: x86_64; OS: Linux; Distribution: Ubuntu 14.04 – 16.04
  - Architecture: ARMv8 (AArch64); OS: Linux; Distribution: Ubuntu 14.04 – 16.04 (Annapurna Alpine V2)

- The following items are required:
  - Minimum 128 MB RAM allocated to the AWS Greengrass core device.
  - Linux kernel version 4.4 or greater: while several versions may work with AWS Greengrass, for optimal security and performance, we recommend version 4.4 or greater.
  - **Glibc library** version 2.14 or greater.
  - The `/var/run` directory must be present on the device.
  - AWS Greengrass requires hardlink and softlink protection to be enabled on the device. Without this, AWS Greengrass can only be run in insecure mode, using the `-i` flag.
  - The `ggc_user` and `ggc_group` user and group must be present on the device.
  - The following Linux kernel configurations must be enabled on the device:
    - Namespace: CONFIG_IPC_NS, CONFIG_UTS_NS, CONFIG_USER_NS, CONFIG_PID_NS
    - CGroups: CONFIG_CGROUP_DEVICE, CONFIG_CGROUPS, CONFIG_MEMCG
    - Others: CONFIG_POSIX_MQUEUE, CONFIG_OVERLAY_FS, CONFIG_HAVE_ARCH_SECCOMP_FILTER, CONFIG_SECCOMP_FILTER, CONFIG_KEYS, CONFIG_SECCOMP
  - The `sqlite3` package is required for AWS IoT device shadows. Ensure it's added to your **PATH** environment variable.
  - `/dev/stdin`, `/dev/stdout`, and `/dev/stderrmus` must be enabled.
  - The Linux kernel must support cgroups.
  - The memory cgroup must be enabled and mounted to allow AWS Greengrass to set the memory limit for Lambdas.
  - The root certificate for Amazon S3 and AWS IoT must be present in the system trust store.

- The following items may be optional:
  - **Python** version 2.7 is required if Python Lambdas are used. If so, ensure that it's added to your **PATH** environment variable.
Getting Started with AWS Greengrass

This tutorial includes six modules, each designed to show you AWS Greengrass basics and help you get started in as few steps as possible. This tutorial covers:

- The AWS Greengrass programming model.
- Fundamental concepts, such as AWS Greengrass cores, groups, and subscriptions.
- The deployment process for running AWS Lambda functions at the edge.

Requirements

To complete this tutorial, you will need the following:

- A Mac, Windows PC, or UNIX-like system.
- An Amazon Web Services (AWS) account. If you don't have an AWS account, see the section called “Create an AWS Account” (p. 12).
- A Raspberry Pi Model 3 with a 8 GB microSD card, or an Amazon EC2 instance. Because AWS Greengrass is intended to be used with physical hardware, we recommend that you use a Raspberry Pi.
- Basic familiarity with Python 2.7.

Although this tutorial focuses on running AWS Greengrass on a Raspberry Pi or an Amazon EC2 instance, other platforms are supported. For more information, see Supported Platforms and Requirements (p. 9).

Create an AWS Account

If you don't have an AWS account, follow these steps:

1. Open the AWS home page, and choose Create an AWS Account.

   **Note**

   If you've signed in to AWS recently, you might see Sign In to the Console instead.

2. Follow the online instructions. Part of the sign-up procedure involves receiving a phone call and entering a PIN using your phone keypad.

Module 1: Environment Setup for Greengrass

This module shows you how to get an out-of-the-box Raspberry Pi, Amazon EC2 instance, or other device ready to be used by AWS Greengrass. Use the Filter View drop-down list in the upper-right corner of this webpage to choose your platform.

This module should take less than 30 minutes to complete.
Setting Up a Raspberry Pi

If you are setting up a Raspberry Pi for the first time, you must follow all of these steps. If you are using an existing Raspberry Pi, you can skip to step 9. However, we recommend that you reimage your Raspberry Pi with the operating system as recommended in step 2.

1. Download and install the SD Memory Card Formatter. Insert the SD card into your computer. Start the program and choose the drive where you have inserted your SD card. You can quick format the SD card.

2. Download the Raspbian Jessie operating system as a .zip file. Using Etcher or another SD card-writing tool, flash the downloaded .zip file onto the SD card. Eject your SD card from your computer, and install the microSD card onto your Raspberry Pi. Because the operating system image is large, this step might take some time.

3. For the first boot, we recommend that you connect the Raspberry Pi to a monitor (through HDMI), a keyboard, a mouse, and a micro USB power source. Connect your Pi to a USB power source and Raspbian should start up. Next, connect your Raspberry Pi to a Wi-Fi network.

   **Note**
   We recommend that you connect your Raspberry Pi to the same network that your computer is connected to. You can use an Ethernet cable to connect to your Raspberry Pi, but be sure that both your computer and Raspberry Pi have internet access.

4. You have to set up SSH on your Pi to remotely connect to it. On your Raspberry Pi, open a Terminal window and type the following:

   ```bash
   sudo raspi-config
   ```

   You should see the following:

   ![Raspberry Pi Software Configuration Tool](image)

   Scroll down and choose **Interfacing Options**, and then choose **P2 SSH**. When prompted, choose **Yes**. SSH should now be enabled.

5. On your Raspberry Pi, in a Terminal window, type the following:

   ```bash
   ifconfig
   ```

   You should see some networking information. Note the IP address of your Raspberry Pi, which is next to `inet`. In this example, the IP address is 169.254.138.24, but yours will most likely be different.
If you are using macOS, open a Terminal window and type the following:

```
ssh pi@IP-address
```

Here, `IP-address` corresponds to the IP address of your Raspberry Pi. The default Raspberry Pi password is `raspberry`.

If you are using Windows, you need to install and configure Putty. Type the IP address of the Raspberry Pi and choose Open with default settings.

6. You are now ready to set up the Raspberry Pi for AWS Greengrass. First, create the user `ggc_user` and group `ggc_group`. On the Raspberry Pi, in your SSH window or a Terminal window, type the following commands:

```
sudo adduser --system ggc_user
sudo addgroup --system ggc_group
```

Type the following commands to install `sqlite3`:

```
sudo apt-get install sqlite3
```

7. Run the following commands to update the Linux kernel version of your Raspberry Pi.

```
sudo apt-get install rpi-update
sudo rpi-update b81a11236fc911170b40a0b09bbd63c84bc5ad59
```

Although several kernel versions might work with AWS Greengrass, for the best security and performance, we recommend that you use the kernel version in step 2. Next, run the following command to reboot your Raspberry Pi.
Run the following command to check the current kernel version of the Raspberry Pi:

```bash
uname -a
```

8. To improve security on the device, we recommend that you type the following commands to enable hardlink and softlink protection at operating system startup.

```bash
cd /etc/sysctl.d
ls
```

If you see the `98-rpi.conf` file, use a text editor (Leafpad, Nano, vi) to add the following two lines to the end of the file. You might need to change permissions (using the `chmod` command) so you can write to the file, or you can use `sudo` to edit as the root user. If you do not see the `98-rpi.conf` file, follow the instructions in the `README.sysctl` file.

```bash
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

Type the following to reboot the Raspberry Pi:

```bash
sudo reboot
```

Run the following to confirm the change:

```bash
sudo sysctl -a | grep fs.protected
```

You should see the hardlinks and softlinks set to 1.

9. Your Raspberry Pi should be ready for AWS Greengrass. To be sure that you have all of the dependencies required for AWS Greengrass, download the AWS Greengrass dependency checker from the GitHub repository and run it on the device.

On macOS, download the dependency checker as a `.zip` file, and then type the following command in a Terminal window:

```bash
scp /path/to/greengrass-dependency-checker-GGCv.x.x.x.zip pi@RPi-IP-address:/home/pi/Downloads
```

On Windows, if you do not have the `scp` command line utility installed, you can use WinSCP to transfer the package.

In the Terminal window or SSH window, type the following commands to run the dependency checker:

```bash
cd /home/pi/Downloads
unzip greengrass-dependency-checker-GGCv.x.x.x.zip
sudo modprobe configs
sudo ./check_ggc_dependencies
```
Setting Up an Amazon EC2 Instance

1. Sign in to the AWS Management Console and launch an Amazon EC2 instance using the Amazon Linux AMI. For information about Amazon EC2 instances, see the Amazon EC2 Getting Started Guide.

2. You need to enable port 8883 to allow incoming MQTT communications so that other devices can connect with the AWS Greengrass core. In the left pane of the Amazon EC2 console, choose Security Groups.

Choose the instance that you just launched, and then choose the Inbound tab.
By default, only one port for SSH is enabled. To enable port 8883, choose **Edit**. Create a custom TCP rule as shown here, and then choose **Save**.

3. In the left pane, choose ** Instances**, choose your instance, and then choose ** Connect**. Connect to your Amazon EC2 instance by using SSH. You can use **Putty** for Windows or **Terminal** for macOS.
4. Run the following commands to create user ggc_user and group ggc_group.

```
sudo adduser --system ggc_user
sudo groupadd --system ggc_group
```

5. To improve security on the device, we recommend that you enable hardlink/softlink protection on the operating system at startup. Type the following commands:

```
ls /etc/sysctl.d
```

Using your favorite text editor (Nano, Emacs, vi), add the following two lines to the end of the `00-defaults.conf` file. You might need to change permissions (using the `chmod` command) to write to the file, or use the `sudo` command to edit as root.

```
fs.protected_hardlinks = 1
fs.protected_symlinks = 1
```

Type the following command to reboot the Amazon EC2 instance.

```
sudo reboot
```
After a few minutes, connect to your instance by using SSH. Run the following command to confirm the change.

```
sudo sysctl -a | grep fs.protected
```

You should see hardlinks and softlinks set to 1.

6. Extract and run the following script to mount Linux control groups (`cgroups`). (This is an AWS Greengrass dependency.):

```
curl https://raw.githubusercontent.com/tianon/cgroupfs-mount/951c38ee8d802330454bdede20d85e1c0f8d312/cgroupfs-mount > cgroupfs-mount.sh
chmod +x cgroupfs-mount.sh
sudo bash ./cgroupfs-mount.sh
```

Your Amazon EC2 instance should be ready for AWS Greengrass. To be sure that you have all of the dependencies, extract and run the following AWS Greengrass dependency script from the GitHub repository.

```
git clone https://github.com/aws-samples/aws-greengrass-samples.git
cd greengrass-dependency-checker-GGCvx.x.x
sudo ./check_ggc_dependencies
```

## Setting Up Other Devices

If you are new to AWS Greengrass, we recommend that you use a Raspberry Pi or an Amazon EC2 instance and follow the steps provided to set up the device. Follow these steps to make your own AWS Greengrass-supported device ready for AWS Greengrass.

1. To make sure you have other devices ready to run AWS Greengrass, download and extract the Greengrass dependency checker from the GitHub repository, and then run the following commands:

```
git clone https://github.com/aws-samples/aws-greengrass-samples.git
cd greengrass-dependency-checker-GGCvx.x.x
sudo ./check_ggc_dependencies
```

This script runs on AWS Greengrass supported platforms and requires the following Linux system commands to execute:

```
printf, uname, cat, la, head, find, zcat, awk, sed, sysctl, wc, cut, sort, expr, grep, test, dirname, readlink, xargs, strings, uniq
```

2. Install required dependencies on your device, as indicated by the dependency script. For missing kernel-level dependencies, you might have to recompile your kernel. For mounting Linux control groups (`cgroups`), you can run the `cgroupfs-mount` script.

   **Note**

   If no errors appear in the output, there is a good chance that AWS Greengrass can run successfully on your device.

These are the AWS Greengrass dependencies:
Required items:

- For best security and performance, Linux kernel version 4.4 or later.
- The `/var/run` directory must be present on the device.
- Hardlink/softlink protection must be enabled on the device. Otherwise, AWS Greengrass can only be run in insecure mode, using the `-i` flag.
- `ggc_user` and `ggc_group` must be present on the device.
- The following Linux kernel configurations must be enabled on the device:
  - Namespace: `CONFIG_IPC_NS, CONFIG_UTS_NS, CONFIG_USER_NS, CONFIG_PID_NS`
  - Cgroups: `CONFIG_CGROUP_DEVICE, CONFIG_CGROUPS, CONFIG_MEMCG`
  - Others: `CONFIG_POSIX_MQUEUE, CONFIG_OVERLAY_FS, CONFIG_HAVE_ARCH_SECCOMP_FILTER, CONFIG_SECCOMP_FILTER, CONFIG_KEYS, CONFIG_SECCOMP`
- The `sqlite3` package is required for AWS IoT device shadows. Make sure it is added to your `PATH` environment variable.
- `/dev/stdin`, `/dev/stdout`, and `/dev(stderr` must be enabled.
- The Linux kernel must support cgroups.
- The `memory` cgroup must be enabled and mounted to allow AWS Greengrass to set the memory limit for Lambda functions.
- The root certificate for Amazon S3 and AWS IoT must be present in the system trust store.

Optional or feature-dependent items:

- The `devices` cgroup must be enabled and mounted if Lambda functions with local resource access are used to open device files.
- If you are using Python Lambda functions, Python 2.7 is required. Make sure it is added to your `PATH` environment variable.
- If you are using Node.JS Lambda functions, Node.JS 6.10 or later is required. Make sure it is added to your `PATH` environment variable.
- If you are using Java Lambda functions, Java 8 or later is required. Make sure it is added to your `PATH` environment variable.
- OpenSSL 1.0.1 or later and the following commands are required for the over-the-air (OTA) agent: `wget`, `realpath`, `tar`, `readlink`, `basename`, `dirname`, `pidof`, `df`, `grep`, and `umount`.

Module 2: Installing the Greengrass Core

This module shows you how to install the AWS Greengrass core software on your device. Before you begin, make sure that you have completed Module 1 (p. 12).

The AWS Greengrass core software provides the following functionality:

- Allows deployment and execution of local applications that are created by using AWS Lambda functions and managed through the deployment API.
- Enables local messaging between devices over a secure network by using a managed subscription scheme through the MQTT protocol.
- Ensures secure connections between devices and the cloud using device authentication and authorization.
- Provides secure, over-the-air, software updates of user-defined Lambda functions.
This module should take less than 30 minutes to complete.

**Configuring AWS Greengrass on AWS IoT**

1. Sign in to the [AWS Management Console](https://aws.amazon.com) on your computer and open the AWS IoT console. In the left pane, choose **Greengrass**.

2. On the **Welcome to AWS Greengrass** page, choose **Get Started**:
3. Create an AWS Greengrass group. An AWS Greengrass group contains information about the devices and how messages are processed in the group. Each AWS Greengrass group requires an AWS Greengrass core device that processes messages sent within the group. An AWS Greengrass core needs a certificate and an AWS IoT policy to access AWS Greengrass and AWS Cloud Services. On the Set up your Greengrass group page, choose Use easy creation.

4. Type a name for your group (for example, MyFirstGroup), then choose Next:
5. Use the default name for the AWS Greengrass core, and choose **Next**:

6. On the **Run a scripted easy Group creation** page, choose **Create Group and Core**.
AWS IoT creates an AWS Greengrass group for you with default security policies and configuration files for you to load onto your device.

7. On the confirmation page, choose Download these resources as a tar.gz (to download security resources for your AWS Greengrass core) and Download Greengrass version x.x.x with the CPU architecture that corresponds to your device (to download the Greengrass binary). If you are using a Raspberry Pi, the architecture is ARMv7L. If you are using an Amazon EC2 instance, the architecture is x86_64.

Choose Finish only after you have downloaded the security resources and the AWS Greengrass binary.
Starting AWS Greengrass on a Device

1. From the earlier step (p. 24), the AWS Greengrass core software is in the `greengrass-platform-version.tar.gz` file that you downloaded from the AWS Greengrass console. If you are using Windows, WinSCP provides an interface for you to copy the files to your device. If you are using macOS, you can use the `scp` utility to copy the software binary and the downloaded resources onto your device.

   ```
   cd path-to-downloaded-files
   scp greengrass-platform-version.tar.gz pi@IP-address
   scp GUID-setup.tar.gz pi@IP-address
   ```

   Use SSH to connect to your device (using the Terminal for macOS or Putty for Windows) and then extract the binary and certificates.

   ```
   sudo tar -xzf greengrass-platform-version.tar.gz -C /
   ```
sudo tar -xzG GUID-setup.tar.gz -C /greengrass

The /greengrass directory is created in your root folder of your device. The certificates are installed in the /greengrass/certs folder. The config.json file, which contains configurations specific to your AWS Greengrass core and AWS IoT endpoint, is installed in the /greengrass/config folder. For more information, see config.json Parameter Summary (p. 26).

2. Install the Symantec VeriSign root CA onto your device. This certificate enables your device to communicate with AWS IoT using the MQTT messaging protocol over TLS. Make sure your device is connected to the internet and then run the following commands.

```bash
cd ~
sudo mv root.ca.pem /greengrass/certs/
```

You cannot use curl to move the certificate into the /greengrass/certs/ folder. You have to use curl outside of /greengrass/ and then move it.

3. Use the following commands to start AWS Greengrass. (x.x.x represents the version number.)

```bash
cd /greengrass/ggc/packages/x.x.x/
sudo ./greengrassd start
```

You should see output similar to the following:

```
pi@raspberrypi:/greengrass/ggc/packages/1.1.0 $ sudo ./greengrassd start
Setting up greengrass daemon
Validating execution environment
Found cgroup subsystem: cpu
Found cgroup subsystem: cpuset
Found cgroup subsystem: blkio
Found cgroup subsystem: memory
Found cgroup subsystem: devices
Found cgroup subsystem: freezer
Found cgroup subsystem: net_cls
Starting greengrass daemon
Greengrass successfully started with PID: 1658
```

Run the following command to confirm that the AWS Greengrass core is working. Replace the PID shown here with your own number.

```
ps aux | grep 1658
```

If you run into issues starting AWS Greengrass, see Troubleshooting AWS Greengrass Applications (p. 115).

**config.json Parameter Summary**

GGC v1.3.0

```json
{
  "coreThing": {
    "caPath": "ROOT_CA_PEM_HERE",
    "certPath": "CLOUD_PEM_CRT_HERE",
  }
}
```
The `config.json` file appears in `/greengrass/config/` and contains the following parameters:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>caPath</td>
<td>The path to the AWS IoT root CA relative to the <code>/greengrass/certs</code> folder.</td>
<td>Save the file under the <code>/greengrass/certs</code> folder.</td>
</tr>
<tr>
<td>certPath</td>
<td>The path to the AWS Greengrass core certificate relative to the <code>/greengrass/certs</code> folder.</td>
<td>Save the file under the <code>/greengrass/certs</code> folder.</td>
</tr>
<tr>
<td>keyPath</td>
<td>The path to the AWS Greengrass core private key relative to <code>/greengrass/certs</code> folder.</td>
<td>Save the file under the <code>/greengrass/certs</code> folder.</td>
</tr>
<tr>
<td>thingArn</td>
<td>The Amazon Resource Name (ARN) of the AWS IoT thing that represents the AWS Greengrass core.</td>
<td>You can find it in the AWS Greengrass console under the definition for your AWS IoT thing.</td>
</tr>
<tr>
<td>iotHost</td>
<td>Your AWS IoT endpoint.</td>
<td>You can find it in the AWS IoT console under <code>Settings</code>.</td>
</tr>
<tr>
<td>ggHost</td>
<td>Your AWS endpoint.</td>
<td>You can find it in the AWS IoT console under <code>Settings</code> with <code>greengrass</code> prepended.</td>
</tr>
<tr>
<td>keepAlive</td>
<td>The MQTT KeepAlive period, in seconds.</td>
<td>This is an optional value. The default value is 600 seconds.</td>
</tr>
<tr>
<td>useSystemd</td>
<td>A binary flag, if your device uses <code>systemd</code>.</td>
<td>Values are <code>yes</code> or <code>no</code>. Use the dependency script in Module 1 (p. 12) to see if your device uses <code>systemd</code>.</td>
</tr>
<tr>
<td>managedRespawn</td>
<td>An optional over-the-air (OTA) updates feature, this indicates that the OTA agent needs to run custom code before an update.</td>
<td>For more information, see...</td>
</tr>
</tbody>
</table>
The AWS Greengrass core software comes packaged with an OTA Update Agent that is capable of updating the core's software or the OTA Update Agent itself to the latest respective versions. You can start an update by invoking the CreateSoftwareUpdateJob API or from the Greengrass console. Updating the Greengrass core software provides the following benefits:

- Fix security vulnerabilities.
- Address software stability issues.
- Deploy new or improved features.

An OTA update makes all these benefits available without having to perform the update manually or having the device which is running the core software physically present. The OTA Update Agent also performs a rollback in case of a failed OTA update. Performing an OTA update is optional but can help you manage your AWS Greengrass core devices. Look for announcements of new versions of the core's software on the Greengrass developer forum.

In order to support an OTA update of Greengrass core software by using the OTA Update Agent, your Greengrass core device must:

- Have available local storage three times the amount of the core's runtime usage requirement.
- Not have trusted boot enabled in the partition containing the Greengrass core platform software. (The AWS Greengrass core can be installed and run on a partition without trusted boot.

---

**Note**

This feature is only available for AWS Greengrass core v1.3.0.
The config.json file exists in `/greengrass/config/` and contains the following parameters:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>caPath</td>
<td>The path to the AWS IoT root CA relative to the <code>/greengrass/certs</code> folder.</td>
<td>Save the file under the <code>/greengrass/certs</code> folder.</td>
</tr>
<tr>
<td>certPath</td>
<td>The path to the AWS Greengrass core certificate relative to the <code>/greengrass/certs</code> folder.</td>
<td>Save the file under the <code>/greengrass/certs</code> folder.</td>
</tr>
<tr>
<td>keyPath</td>
<td>The path to the AWS Greengrass core private key relative to the <code>/greengrass/certs</code> folder.</td>
<td>Save the file under the <code>/greengrass/certs</code> folder.</td>
</tr>
<tr>
<td>thingArn</td>
<td>The Amazon Resource Name (ARN) of the AWS IoT thing that represents the AWS Greengrass core.</td>
<td>You can find it in the AWS Greengrass console under the definition for your AWS IoT thing.</td>
</tr>
<tr>
<td>iotHost</td>
<td>Your AWS IoT endpoint.</td>
<td>You can find it in the AWS IoT console under Settings.</td>
</tr>
<tr>
<td>ggHost</td>
<td>Your AWS endpoint.</td>
<td>You can find it in the AWS IoT console under Settings with greengrass. prepended.</td>
</tr>
<tr>
<td>keepAlive</td>
<td>The MQTT KeepAlive period, in seconds.</td>
<td>This is an optional value. The default value is 600 seconds.</td>
</tr>
<tr>
<td>useSystemd</td>
<td>A binary flag, if your device uses systemd.</td>
<td>Values are yes or no. Use the dependency script in Module</td>
</tr>
</tbody>
</table>
GGC v1.0.0

```
{"coreThing": {
  "caPath": "ROOT_CA_PEM_HERE",
  "certPath": "CLOUD_PEM_CRT_HERE",
  "keyPath": "CLOUD_PEM_KEY_HERE",
  "thingArn": "THING_ARN_HERE",
  "iotHost": "HOST_PREFIX_HERE.iot.AWS_REGION_HERE.amazonaws.com",
  "ggHost": "greengrass.iot.AWS_REGION_HERE.amazonaws.com",
  "keepAlive": 600
},
"runtime": {
  "cgroup": {
    "useSystemd": "yes\|no"
  }
}
}
```

The `config.json` file exists in `/greengrass/configuration/` and contains the following parameters:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>caPath</td>
<td>The path to the AWS IoT root CA relative to the <code>/greengrass/configuration/certs</code> folder.</td>
<td>Save the file under the <code>/greengrass/configuration/certs</code> folder.</td>
</tr>
<tr>
<td>certPath</td>
<td>The path to the AWS Greengrass core certificate relative to the <code>/greengrass/configuration/certs</code> folder.</td>
<td>Save the file under the <code>/greengrass/configuration/certs</code> folder.</td>
</tr>
<tr>
<td>keyPath</td>
<td>The path to the AWS Greengrass core private key relative to the <code>/greengrass/configuration/certs</code> folder.</td>
<td>Save the file under the <code>/greengrass/configuration/certs</code> folder.</td>
</tr>
<tr>
<td>thingArn</td>
<td>The Amazon Resource Name (ARN) of the AWS IoT thing that represents the AWS Greengrass core.</td>
<td>You can find it in the AWS Greengrass console under the definition for your AWS IoT thing.</td>
</tr>
<tr>
<td>iotHost</td>
<td>Your AWS IoT endpoint.</td>
<td>You can find it in the AWS IoT console under <strong>Settings</strong>.</td>
</tr>
<tr>
<td>ggHost</td>
<td>Your AWS endpoint.</td>
<td>You can find it in the AWS IoT console under <strong>Settings</strong> with <code>greengrass. prepended</code>.</td>
</tr>
</tbody>
</table>
Module 3 (Part 1): AWS Lambda on AWS Greengrass

This module shows you how to configure and deploy an AWS Lambda function on your AWS Greengrass core device. It contains information about MQTT messaging, subscriptions, deployments on AWS Greengrass, and Lambda function configurations.

Part 1 of this module shows how to deploy a Lambda function on the AWS Greengrass core that sends "Hello World" messages to the AWS Greengrass cloud. Part 2 (p. 48) covers the differences between on-demand and long-lived Lambda functions running on the AWS Greengrass core. Before you begin, make sure that you have completed Module 1 (p. 12) and Module 2 (p. 20) and have a running AWS Greengrass core device. Module 3 (Part 1) (p. 31) and Module 3 (Part 2) (p. 48) should take approximately 30 minutes each.

Create and Package a Lambda Function

1. From the GitHub repository, download the `greengrassHelloWorld.py` Python script onto your computer (not your device). You can do this by selecting `greengrassHelloWorld.py` and right-clicking the Raw button near the top of the file.
Save the file as a `.py` file:

2. The AWS Greengrass core SDK must be packaged with the `greengrassHelloWorld.py` Lambda function in order to be executed on the AWS Greengrass device. In the left pane of the AWS IoT console, choose **Software**.
To get the AWS Greengrass Core SDK, on the home page, scroll down to **SDKs**, and choose **Configure download**.
From the drop-down list, choose **Python 2.7 version 1.0.0**, and then choose **Download Greengrass Core SDK**.

The AWS Greengrass Core SDK enables Lambda functions to interact with the Greengrass Core on which they run. This allows them to publish messages and interact with shadow data or invoke Lambda functions within the Greengrass Core.

3. Choose the `sdk` folder to extract the downloaded `greengrass-core-python-sdk-1.0.0.tar.gz` file.

After you have extracted the file, you should see the following three subfolders:

```
- examples
- manual
- sdk
```

Copy these folders and the `greengrassHelloWorld.py` file you downloaded earlier into a compressed `.zip` file named `hello_world_python_lambda.zip`. 
4. You are now ready to upload your Lambda function .zip into AWS Lambda. From the AWS Management Console, open the Lambda console.

Choose **Create function.**

Choose **Author from scratch.**

Name your function **Greengrass_HelloWorld.** You can create a security role for your Lambda function or use an existing one. If you create one, use the **AWS IoT Button permissions** role.
Choose **Create Function**.

On the **Configuration** tab, for **Code entry type**, choose **Upload a .zip file**. For **Runtime**, choose **Python 2.7**. For **Handler**, choose `greengrassHelloWorld.function_handler`, and then upload the `hello_world_python_lambda.zip` file you created earlier.

Choose **Save**.
To publish this Lambda function, under Actions, choose **Publish new version**:  

Write a description, such as **First version**, then select **Publish**:

Publish a new version will save a "snapshot" of the code and configuration of the $LATEST version. You will be unable to edit the new version's code. Please click to confirm.

Now, create an alias for the Lambda function. Aliases create a single entity for your Lambda function that AWS Greengrass devices can subscribe to without having to update subscriptions with Lambda version numbers every time a function is modified. From Actions, choose **Create alias**.

**Note**  
If future versions of this Lambda are published, you will have to delete the alias created and create a new one with the same name which points to the new version.

Name the alias **GG_HelloWorld**, set the version to 1 (1 corresponds to the latest published version), and then choose **Create**. AWS Greengrass does not support Lambda aliases for $LATEST.
Configure Lambda for AWS Greengrass

You are now ready to configure your Lambda function to AWS Greengrass.

1. In the AWS IoT console, go to Greengrass, and from Groups, echoose the group you created in Module 2 (p. 20). On the Group Overview page, choose Lambdas.
Choose **Add your first Lambda**.
Choose **Use existing Lambda**.

Search for the name of the Lambda you created in the previous step (**Greengrass_HelloWorld**, not the alias name), select it, and then choose **Next**.
For the version, choose **Alias: GG_HelloWorld**, and then choose **Next**. You should see the **Greengrass_HelloWorld** Lambda function in your group, using the **GG_HelloWorld** alias. Right-click the ellipsis for the Lambda function, and choose **Edit Configuration**:

On the configuration page, change **Timeout** to 25 seconds in that the Lambda function sleeps for 20 seconds before each invocation. Under **Lambda lifecycle**, select **Make this function long-lived and keep it running indefinitely**, and then choose **Update**.
Alias GG_HelloWorld

Memory limit

16 MB

Timeout

25 Sec...

Lambda lifecycle

On-demand function

Make this function long-lived and keep it running indefinitely

Note
A long-lived Lambda function starts automatically after AWS Greengrass starts and keeps running in its own container (or sandbox). This is in contrast to an on-demand Lambda function, which only starts when invoked and stops when there are no tasks left to execute. Information about long-lived and on-demand Lambda functions is provided in Module 3 (Part 2) (p. 48).

2. An AWS Greengrass Lambda function can subscribe or publish messages (using the MQTT protocol):

- To and from other devices (or device shadows) within the AWS Greengrass core. Information about device shadows is provided in Module 5.
- To other Lambda functions,
- Or to the AWS IoT cloud.

The AWS Greengrass group controls the way in which these components interact by using subscriptions that enable more security and predictable interactions.

A subscription consists of a source, target, and topic. The source is the originator of the message. The target is the destination of the message. The topic allows you to filter the data that is sent from the source to the target. The source or target can be an AWS Greengrass device, a Lambda function, a device shadow, or the AWS IoT cloud. A subscription is directed. For an AWS Greengrass device to send messages to and receive messages from a Lambda function, you must set up two subscriptions: one from the device to the Lambda and another from the Lambda function to the device. The Greengrass_HelloWorld Lambda function sends messages only to the hello/world topic in the AWS IoT cloud, so you only need to create one subscription from the Lambda function to the AWS IoT cloud.
On the **Group configuration** page, choose **Subscriptions**, and then choose **Add your first Subscription**.

On the **Lambdas** tab, choose **Greengrass_HelloWorld** as the source.

On the **Service** tab, choose **IoT Cloud**, and then choose **Next**.
Deploying Cloud Configurations to an AWS Greengrass Core Device

Deploy the Lambda function and subscription configurations onto your AWS Greengrass core device. Make sure that your AWS Greengrass core device is connected to the internet. To check that the AWS Greengrass daemon has been started, run the following command on your device:

```
ps aux | grep -E 'greengrass.*daemon'
```

On the **Group** page, choose **Deployments**. Under **Actions**, choose **Deploy**.
Choose **Automatic detection**.

In order for your Group's Devices to discover and communicate with your Core they must be able to acquire connectivity information (e.g. IP address, DNS, port, etc.) before connecting.

**Automatically detect Core endpoints (recommended)**
Greengrass will detect and override connection information as it changes.

**Manually configure Core endpoints**
Manually manage connection information. This can be accessed via your Core device's settings.

The first deployment might take a few minutes. When the deployment is complete, you should see **Successfully completed** under **Status**.

**Verify the Lambda Function Is Running on the Device**

From the left pane of the AWS IoT console, choose **Test**.
In **Subscription topic**, type *hello/world*. For **Quality of Service**, select 0. For **MQTT payload display**, select **Display payloads as strings**.
Verify the Lambda Function Is Running on the Device

1. Choose **Subscribe to topic**.

2. If the Lambda function is running on your device, it can publish messages to the **hello/world** topic.

Next, choose **Subscribe to topic**.

If the Lambda function is running on your device, it can publish messages to the **hello/world** topic.

1. **Publish**
   - Specify a topic and a message to publish with a QoS of 0.
   - **hello/world**

   ```
   1 {
   2     "message": "Hello from AWS IoT console"
   3   }
   ```

   **Hello world!** Sent from Greengrass Core running on platform: Linux-4.9.41-v7+-armv7l-with-debian-9.1

   **Hello world!** Sent from Greengrass Core running on platform: Linux-4.9.41-v7+-armv7l-with-debian-9.1
Module 3 (Part 2): AWS Lambda on AWSGreengrass

This module shows you how to configure and deploy an AWS Lambda function on your AWS Greengrass core device. It contains information about MQTT messaging, subscriptions, deployments on AWS Greengrass, and Lambda function configurations.

Part 1 (p. 31) of this module described how to deploy a Lambda function on a AWS Greengrass core that sends a "Hello World" messages to the AWS Greengrass cloud. This part explores the differences between on-demand and long-lived Lambda functions running on the AWS Greengrass core. Before you begin, make sure you have completed Module 1 (p. 12), Module 2 (p. 20), and Module 3 (Part 1) (p. 31). This module should take approximately 30 minutes to complete.

Create and Package the Lambda Function

1. Download the greengrassHelloWorldCounter.py script from the GitHub repository onto your computer (not the AWS Greengrass device). Package this file with the greengrasssdk, greengrass_common, and greengrass_ipc_python_sdk folders (from the AWS Greengrass SDK), as shown in Module 3 (Part 1) (p. 31). Name this .zip file hello_world_counter_python_lambda.zip.

2. Upload your Lambda function into the AWS Lambda console. Name your function Greengrass_HelloWorld_Counter with the default role you created in Module 3 (Part 1) (p. 31). For Handler, type greengrassHelloWorldCounter.function_handler.

Publish a new version, and create an alias for version 1 called GG_HW_Counter.
Configure Long-Lived Lambda Functions for AWS Greengrass

You are now ready to configure your Lambda function to AWS Greengrass. In the AWS IoT console, go to Greengrass, and then choose Groups. Choose the group you created in Module 2 (p. 20). On the Group Overview page, choose Lambdas. Search for Greengrass_HelloWorld.Counter and then choose Next:

For the version, choose Alias: GG_HW.Counter and then choose Next. On the Group Configuration page, under Lambdas, for the Greengrass_HelloWorld.Counter Lambda function, choose Edit Configuration.
On the Configuration page, change Timeout to 25 seconds in that the Lambda function sleeps for 20 seconds before each invocation. Under Lambda lifecycle, select Make this function long-lived and keep it running indefinitely, and then choose Update.

**Greengrass_HelloWorld.Counter**

*View function in AWS Lambda*

**Alias GG_HW.Counter**

**Memory limit**

| 16 | MB |

**Timeout**

| 25 | Sec... |

**Lambda lifecycle**

- [ ] On-demand function
- [x] Make this function long-lived and keep it running indefinitely

**Testing Long-Lived Lambda Functions**

A long-lived Lambda function starts automatically when the AWS Greengrass core starts (and runs in a single container or sandbox). Any variables or preprocessing that are defined outside of the function handler are retained for every invocation of the function handler. Multiple invocations of the function handler are queued until earlier invocations have been executed. The greengrassHelloWorldCounter.py Lambda function is similar to the greengrassHelloWorld.py function except there is a variable, my_counter, that is outside of the function_handler(event, context) method.

1. On the Group configuration page, choose Subscription and then choose Add Subscription. Under Select a source, choose Greengrass_HelloWorld.Counter. Under Select a target, choose IoT Cloud, and then choose Next.
The topic filter should be **hello/world/counter**. Choose **Next**, and then choose **Finish**.

This single subscription goes in one direction only: from the **Greengrass_HelloWorld_Counter** Lambda function to the AWS IoT cloud. To trigger this Lambda function from the cloud, you need to create a subscription in the opposite direction. Add another subscription with **IoT Cloud** as the source and **Greengrass_HelloWorld_Counter** as the target. Use the same **hello/world/counter/trigger** topic. Note the **/trigger** extension. Because you have created two subscriptions, we do not want them to interfere with each other.
2. Make sure that your AWS Greengrass core device is connected to the internet and that the AWS Greengrass daemon is running. Next, under Actions, choose Deploy to deploy your new configurations.

3. After your deployment is complete, in the AWS IoT console, choose Test. In Subscription topic, type hello/world/counter. For Quality of Service, select 0. For MQTT payload display, select Display payloads as strings, and then choose Subscribe to topic.
Unlike Part 1 (p. 31) of this module, you should not be able to see any messages after you subscribe. This is because in the greengrassHelloWorldCounter.py function the code to publish to this topic is inside the `function_handler(event, context)` function. `function_handler(event, context)` is triggered only when it receives an MQTT message on the `hello/world/counter/trigger` topic. To trigger the handler, publish the default message to the `hello/world/counter/trigger` topic.

Every time a message is published to the `hello/world/counter/trigger` topic, the `Invocation Count` variable is incremented. Because the function handler in the Lambda function includes a 20-second sleep cycle, repeatedly triggering the handler queues up responses from the AWS Greengrass core.
Testing On-Demand Lambda Functions

An on-demand Lambda function is similar in functionality to an AWS cloud Lambda function. Multiple invocations of an on-demand Lambda function can run in parallel. Each invocation of the Lambda function creates a new, separate container to process invocations. The container can be reused for future invocations if resources permit. For information about container reuse, see Understanding Container Reuse in AWS Lambda. Any variables or preprocessing that are defined outside of the function handler are not retained when new containers are created. As a best practice, we recommend that you use on-demand Lambda functions instead of long-lived functions whenever possible because they are less resource-intensive.

1. On the configuration page for your group, choose Lambdas. For the Greengrass_HelloWorld_Counter Lambda function, choose Edit Configuration.

2. Under Lambda lifecycle, select On-demand function.
3. On the Group Configuration page, under Actions, choose Deploy to deploy your updated configurations to the AWS Greengrass core device.

4. After your deployment is complete, in the AWS IoT console, choose Test. For Subscription topic, type hello/world/counter. For Quality of Service, select 0. For MQTT payload display, select Display payloads as strings and then choose Subscribe to topic.
Again, you should not be able to see any messages after you subscribe. Trigger the function to the `hello/world/counter/trigger` topic, then choose **Publish to topic** three times, within five seconds of each press of the button.

```
1
2  
3  
```

Each publish is triggering the function handler and creating a new container for each invocation. The invocation count is not incremented for each of the three times you triggered the function.
Wait approximately thirty seconds or more, and then choose **Publish to topic**. This time you should see an incremented invocation count.
This means that a container, first created from a prior invocation, is being reused, and preprocessing variables outside of the function handler have been stored.

You should now understand the two types of Lambda functions that can run on the AWS Greengrass core. Module 4 (p. 58), shows you how devices can interact within an AWS Greengrass group.

Module 4: Interacting with Devices in an AWS Greengrass Group

This module shows you how AWS IoT devices can connect to and communicate with an AWS Greengrass core device. AWS IoT devices that connect to an AWS Greengrass core are part of an AWS Greengrass group and can participate in the AWS Greengrass programming paradigm. In this module, one AWS Greengrass device sends a HelloWorld message to another AWS Greengrass device within the AWS Greengrass core.
Before you begin, make sure you have a running AWS Greengrass core device and have completed Module 1 (p. 12), Module 2 (p. 20), Module 3 (Part 1) (p. 31). You do not need other components or devices. This module should take less than 30 minutes to complete.

**Create AWS IoT Devices in an AWS Greengrass Group**

1. In the AWS IoT console, under **Greengrass**, under **Groups**, choose your group to open its configuration page. Choose **Devices**, and then choose **Add your first Device**.
Choose **Create New Device**:

**Add a Device**

Greengrass Devices can be created by re-purposing an existing IoT Thing from your Registry or by creating new Registry items, and then adding them to a Greengrass Group.

**Create a new Device**

You will create a new Device and generate a certificate, a private key and a public key.

**Use an existing IoT Thing as an Device**

You can add an existing IoT Thing to your Group.

Register this device as **HelloWorld_Publisher**, then choose **Next.**
Every Greengrass Group requires a device running Greengrass software. It enables communication between Devices, local Lambda functions, and AWS cloud computing services. Adding information to the Registry is the first step in provisioning a device as your Greengrass Core.

**Name**

HelloWorld_Publisher

Show optional configuration (this can be done later)

---

For **1-Click**, choose **Use Defaults**:

**Set up security**

The Device needs a certificate and a policy before it can be added to Greengrass. If you're unfamiliar with any of these steps we recommend the Automated Setup.

**1-Click**

This will generate a certificate, public key and private key using AWS IoTs root CA, generate a default policy, and create a new IAM role with default permissions.

**Advanced setup**

This will allow you to handle certificate signing request (CSR) based on a private key you own, customize your own policy, and use an existing IAM role or create a new one.

Download certificates for your device onto your computer and then decompress them.
2. Repeat step 1 to add another device to the group and name it **HelloWorld_Subscriber**. You should now have two devices in your AWS Greengrass core.

3. Download the AWS IoT root certificate from Symantec and save it as **root-ca-cert.pem**. Save all of the certificates in one folder on your computer so they are easily accessible. (You need them later in this module.)

**Configure Subscriptions**

1. The **HelloWorld_Publisher** device sends a **HelloWorld** message to the **HelloWorld_Subscriber** device. Create a subscription with **HelloWorld_Publisher** as the source and **HelloWorld_Subscriber** as the target.
Install the AWS IoT Device Python SDK

From the GitHub repository, install the AWS IoT Device Python SDK on your computer. Follow the instructions in the README file. This SDK is used by all AWS IoT devices to communicate with the AWS IoT cloud and AWS Greengrass cores. If you use the `pip` command to install the SDK, then download the SDK as a `.zip` and decompress it. In the `samples` subfolder, under `greengrass`, you should see a `basicDiscovery.py` file. Copy this file to the folder with the certificates you downloaded earlier.
Testing Communications

1. Install Python 2.7 on your computer.

2. If you are using macOS, open two Terminal windows. If you’re using Windows, open two Command Prompt windows. One window is for the HelloWorld_Publisher device and the other is for the HelloWorld_Subscriber device. Every time the following script is executed for the first time, the AWS Greengrass discovery service connects to an AWS Greengrass core. Internet access for the device is required. After a device has discovered the AWS Greengrass core and successfully connected to it, future messaging and operations can be executed locally. Before you execute the following commands, make sure that your machine is connected to the Internet and your AWS Greengrass core device (that is, make sure your computer can successfully ping your AWS Greengrass core device). It works best when your computer and the AWS Greengrass device are on the same network.

3. Type the following into the window for the HelloWorld_Publisher device:
   ```
cd /path/with/certs
python basicDiscovery.py -e AWS_IOT_ENDPOINT -r root-ca-cert.pem -c publisher-certificate.pem.crt -k publisher-private.pem.key -n HelloWorld_Publisher -m publish -t 'hello/world/pubsub' -M 'Hello, World! Sent from HelloWorld_Publisher'
   ```
   You should see the following output:
   ```
   Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 0}
   2017-11-15 21:12:24,296 - AWSIoTPythonSDK.core.protocol.internal.workers - DEBUG - Produced [puback] event
   2017-11-15 21:12:27,381 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
   Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 1}
   2017-11-15 21:12:28,395 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
   Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 2}
   2017-11-15 21:12:29,019 - AWSIoTPythonSDK.core.protocol.mqtt_core - INFO - Performing sync publish...
   Published topic hello/world/pubsub: {"message": "Hello, World! Sent from HelloWorld_Publisher", "sequence": 3}
   ```

4. Type the following into the window for the HelloWorld_Subscriber device:
   ```
cd /path/with/certs
python basicDiscovery.py -e AWS_IOT_ENDPOINT -r root-ca-cert.pem -c subscriber-certificate.pem.crt -k subscriber-private.pem.key -n HelloWorld_Subscriber -m subscribe -t 'hello/world/pubsub'
   ```
   You should see the following output:
Module 5: Interacting with Device Shadows

This advanced module shows you how AWS Greengrass devices can interact with AWS IoT device shadows in an AWS Greengrass group. A thing shadow is a JSON document that is used to store and retrieve current state information for a thing. In this module, you see how one AWS Greengrass device (GG_Switch) can modify the state of another AWS Greengrass device (GG_TrafficLight) and how these states can be synced to the AWS Greengrass cloud.

Before you begin, make sure that you have completed Module 1 (p. 12), Module 2 (p. 20), Module 3 (Part 1) (p. 31), and Module 3 (Part 2) (p. 48). You should also understand how to connect devices to an AWS Greengrass core (Module 4 (p. 58)). You do not need other components or devices. This module should take about 30 minutes to complete.
Configure Devices and Subscriptions

1. Create two devices in your AWS Greengrass group, **GG_Switch** and **GG_TrafficLight**. Use the default security settings.

   **Note**
   You can detach devices used in earlier modules.

   ![My1stGroup](image)

   Save the certificates for the devices on your local machine. You use them later.

   Now, each AWS IoT device shadow can be synced to AWS IoT when the AWS Greengrass core is connected to the internet. First, you use device shadows without syncing the shadows to the cloud. Later in the module, you enable syncing. By default, cloud syncing should be disabled. If it’s not disabled, under **Devices**, choose the ellipsis (…), and then choose **Local Shadow Only**.

2. Create the following subscriptions for your group:

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Topic</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GG_Switch</td>
<td>Local Shadow Service</td>
<td>$aws/things/GG_TrafficLight/shadow/update</td>
<td>The GG_Switch sends an update request to update topic.</td>
</tr>
<tr>
<td>Local Shadow Service</td>
<td>GG_Switch</td>
<td>$aws/things/GG_TrafficLight/shadow/update/accepted</td>
<td>The GG_Switch needs to know whether the update request was accepted or rejected.</td>
</tr>
</tbody>
</table>
### Configure Devices and Subscriptions

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Topic</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Shadow Service</td>
<td>GG_Switch</td>
<td>$aws/things/GG_TrafficLight/shadow/update/rejected</td>
<td>The GG_Switch needs to know whether the update request was accepted or rejected.</td>
</tr>
<tr>
<td>Local Shadow Service</td>
<td>GG_TrafficLight</td>
<td>$aws/things/GG_TrafficLight/shadow/update/delta</td>
<td>The Local Shadow Service sends a received update to GG_TrafficLight through the delta topic.</td>
</tr>
<tr>
<td>Local Shadow Service</td>
<td>GG_TrafficLight</td>
<td>$aws/things/GG_TrafficLight/shadow/update/accepted</td>
<td>The GG_TrafficLight needs to know whether the update request from GG_Switch was accepted or rejected.</td>
</tr>
<tr>
<td>Local Shadow Service</td>
<td>GG_TrafficLight</td>
<td>$aws/things/GG_TrafficLight/shadow/update/rejected</td>
<td>The GG_TrafficLight needs to know whether the update request from GG_Switch was accepted or rejected.</td>
</tr>
</tbody>
</table>

**Note**
Although you can use wildcards (for example, `$aws/things/GG_TrafficLight/shadow/#`) to consolidate some of the subscriptions, we do not recommend this practice.

The topic must be written exactly as shown in the table. Do not include an extra `/` at the end of a topic. Here is the end result:
Download Required Files

1. If you haven’t already done so, install the AWS IoT Device Python SDK. Follow the instructions in the README file. This SDK is used by all AWS IoT devices to communicate with the AWS IoT cloud and AWS Greengrass cores.

2. From the GitHub repository, download the lightController.py and trafficLight.py files to your computer and copy them to the same folder where your device certificates are stored. The lightController.py file corresponds to the GG_Switch device. The trafficLight.py file corresponds to the GG_TrafficLight device.
Test Communications (Device Syncs Disabled)

1. If you are using macOS, open two Terminal windows. If you are using Windows, open two Command Prompt windows. One window is for the GG_Switch device and the other is for the GG_TrafficLight device. Both scripts, when executed for the first time, run the AWS Greengrass discovery service to connect to an AWS Greengrass core. (For this reason, internet access is required.) After a device has discovered and successfully connected to an AWS Greengrass core, future operations can be executed locally. Before you execute the following commands, make sure that your machine is connected to the internet and your AWS Greengrass core device (that is, make sure that your computer can successfully ping your AWS Greengrass core device). It works best when both your computer and AWS Greengrass device are on the same network.

Run the following commands for GG_Switch:

```bash
cd /path/with/certs
python lightController.py -e AWS_IOT_ENDPOINT -r root-ca-cert.pem -c switch-certificate.pem.crt -k switch-private.pem.key -n GG_TrafficLight -id GG_Switch
```

Run the following command for GG_TrafficLight:

```bash
cd /path/with/certs
python trafficLight.py -e AWS_IOT_ENDPOINT -r root-ca-cert.pem -c trafficlight-certificate.pem.crt -k trafficlight-private.pem.key -n GG_TrafficLight -id GG_TrafficLight
```

Every 20 seconds, the switch updates the shadow state to R, Y, and G, and the light displays its state.

GG_Switch output:

```
[95] INFO - reatter Push
Update request with token: 8c39829e-3b0c-45b0-b38c-8e176de50f4 accepted!
property: Y
~~~~~~~~~~~~~~~~~~~~
{"state":{"desired":"R"}}
```

GG_TrafficLight output:
2. In the AWS IoT console, under your group’s configuration, under Devices, choose **GG_TrafficLight**.

From the left pane, choose **Shadow**. Under **Shadow State**, there should not be any updates to this shadow topic after the **GG_Switch** changes states.

Test Communications (Device Syncs Enabled)

1. In the AWS IoT console, under your group’s configuration, under Devices, choose the ellipsis for the **GG_TrafficLight** device, then choose **Sync to the Cloud**.
You should receive a notification that the device shadow has been updated.

2. Redeploy your modified configurations to your AWS Greengrass device.
3. Repeat the step in which you create two windows (p. 69).
4. In the AWS IoT console, under your group’s configuration, under Devices, choose **GG_TrafficLight**.

Because you enabled syncs of the **GG_TrafficLight** shadow to AWS IoT, the shadow state in the cloud should be updated automatically whenever **GG_Switch** sends an update. This functionality can be used to expose the state of an AWS Greengrass device to AWS IoT.

**Shadow Document**

**Last update:** Nov 13, 2017 3:34:22 PM -0800

**Shadow state:**

```
1  {
2    "desired": {
3      "property": "G"
4    },
5    "reported": {
6      "property": "R"
7    },
8    "delta": {
9      "property": "G"
10   }
11  }
```
5. You can troubleshoot by looking at the AWS Greengrass core logs, specifically the `router.log` file in `/greengrass/ggc/var/log/system/router.log`. For more information, see Troubleshooting AWS Greengrass Applications (p. 115).

Module 6: Accessing AWS Cloud Services

This advanced module shows you how AWS Greengrass cores can interact with other Amazon Web Services in the cloud. It builds on the traffic light example in Module 5 (p. 65) and uses an additional Lambda function that processes shadow states and uploads a summary to an Amazon DynamoDB table.
Before you begin, make sure that you have completed Module 1 (p. 12) through Module 5 (p. 65). You do not need other components or devices. This module should take about 30 minutes to complete.

**Note**
This module creates and updates a table in DynamoDB. Although most of the operations are small and fall within the AWS Free Tier, see the DynamoDB pricing documentation.

## Configure IAM Roles

1. Because you are creating a Lambda function that accesses other AWS services, you need to create an IAM role that has access to DynamoDB and AWS Greengrass. For more information about IAM, see AWS Identity and Access Management documentation.

   In the AWS Management Console, search for and go to the IAM console. Under **Roles**, choose **Create Role**.
Under **AWS service**, choose **Greengrass**.
On the permissions page, search for and check the following policies: `AWSGreengrassResourceAccessRolePolicy`, `AWSGreengrassFullAccess`, and `AWSDynamoDBFullAccess`. Choose Next: Review. For Role name, type `Greengrass_DynamoDB_Role`, and then choose Create role.
2. Repeat step 1 to create the role for AWS Lambda (instead of AWS Greengrass). Give the role the same policies (AWSGreengrassResourceAccessRolePolicy, AWSGreengrassFullAccess, and AWSDynamoDBFullAccess). For Role name, type Lambda_DynamoDB_Role.

3. In the AWS IoT console, go to the configuration page for your AWS Greengrass group, and choose Settings. Choose Add Role.

The IAM role you just created should appear in the drop-down list. If it does not appear, search for it, select it, and then choose Save:
The following Lambda function simulates, on every GG_TrafficLight shadow state change to G, the passing of a randomized number of cars (from 1 to 20). Basic statistics, such as min and max, are then passed to a DynamoDB table on every third G light change.

1. From the GitHub account download the carAggregator.py Lambda function. Because this Lambda function also accesses other Amazon Web Services, you need to install the boto3 package and its dependencies to the same folder where carAggregator.py is stored. If you're using macOS, open a Terminal window. If you are using Windows, open a Command Prompt window. Type the following:

```
pip install boto3 -t /path/to/carAggregator.py
```
Compress these files into one .zip file and name it `car_aggregator.zip`.

2. In the Lambda console, create a function named `GG_Car_Aggregator` and upload the .zip file you just created. Be sure to update the function handler to `carAggregator.function_handler`, as shown here.
Publish the version of this Lambda function and create an alias for it named **GG_CarAgg**. For step-by-step instructions, see Module 3 (Part 1) (p. 31).

3. In the AWS IoT console, under your group’s configuration page, add the Lambda function you just created to your group. The GG_CarAgg alias should point to this Lambda function:

   ![Lambda function image]

   **Note**
   You can remove other Lambda functions from earlier modules.

   Edit the configuration of this Lambda function. Choose the ellipsis (...). Under **Lambda lifecycle**, select **Make this function long-lived and keep it running indefinitely**, and then choose **Update**.
Configure Subscriptions

1. In the AWS IoT console, under your group's configuration page, create a subscription from GG_TrafficLight to the GG_Car_Aggregator Lambda function (pointing to the GG_CarAgg alias). This is in addition to the other subscriptions you created in Module 5 (p. 65). The following table shows the complete list of required subscriptions. The new subscription appears in the last row of the table.
### Configure Subscriptions

<table>
<thead>
<tr>
<th>Source</th>
<th>Target</th>
<th>Topic</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>GG_Switch</td>
<td>Local Shadow Service</td>
<td>$aws/things/GG_TrafficLight/shadow/update</td>
<td>The GG_Switch sends an update request to update topic.</td>
</tr>
<tr>
<td>Local Shadow Service</td>
<td>GG_Switch</td>
<td>$aws/things/GG_TrafficLight/shadow/update/accepted</td>
<td>The GG_Switch needs to know whether the update request was accepted or rejected.</td>
</tr>
<tr>
<td>Local Shadow Service</td>
<td>GG_Switch</td>
<td>$aws/things/GG_TrafficLight/shadow/update/rejected</td>
<td>The GG_Switch needs to know whether the update request was accepted or rejected.</td>
</tr>
<tr>
<td>Local Shadow Service</td>
<td>GG_TrafficLight</td>
<td>$aws/things/GG_TrafficLight/shadow/update/delta</td>
<td>The shadow service sends a received update to GG_TrafficLight through the delta topic.</td>
</tr>
<tr>
<td>Local Shadow Service</td>
<td>GG_TrafficLight</td>
<td>$aws/things/GG_TrafficLight/shadow/update/accepted</td>
<td>The GG_TrafficLight needs to know whether the update request from GG_Switch was accepted or rejected.</td>
</tr>
<tr>
<td>Local Shadow Service</td>
<td>GG_TrafficLight</td>
<td>$aws/things/GG_TrafficLight/shadow/update/rejected</td>
<td>The GG_TrafficLight needs to know whether the update request from GG_Switch was accepted or rejected.</td>
</tr>
</tbody>
</table>

**Note**

Except for Module 5 (p. 65), you can delete the subscriptions from earlier modules () that are not included in this table.

2. Deploy your modified group configurations to your AWS Greengrass core.
Test Communications

1. If you are using macOS, open two Terminal windows. If you are using Windows, open two Command Prompt windows. Just as in Module 5 (p. 65), one window is for the GG_Switch device and the other is for the GG_TrafficLight device.

Run the following commands for the GG_Switch device:

```bash
cd /path/with/certs
python lightController.py -e AWS_IOT_ENDPOINT -r root-ca-cert.pem -c switch-certificate.pem.crt -k switch-private.pem.key -n GG_TrafficLight -id GG_Switch
```

Run the following command for for GG_TrafficLight device:

```bash
cd /path/with/certs
python trafficLight.py -e AWS_IOT_ENDPOINT -r root-ca-cert.pem -c trafficLight-certificate.pem.crt -k trafficLight-private.pem.key -n GG_TrafficLight -id GG_TrafficLight
```

Every 20 seconds, the switch updates the shadow state to R, Y, and G, and the light displays its state.

2. On every third green light (every 3 minutes), the function handler of the Lambda function is triggered, and a new DynamoDB record is created. After `lightController.py` and `trafficLight.py` have been executed for three minutes, go to the AWS Management Console, search for and open the DynamoDB console. Make sure the us-east-1 region is selected. Under Tables, you should see a CarStats table. Choose the Items tab.

You should see entries with basic statistics on cars passed (one entry every three minutes).
3. You can troubleshoot by looking at the AWS Greengrass core logs, specifically /greengrass/ggc/var/log/system/router.log. For more information, see Troubleshooting AWS Greengrass Applications (p. 115).

You have reached the end of this tutorial and should now understand the AWS Greengrass programming model and its fundamental concepts, including AWS Greengrass cores, groups, subscriptions, and the deployment process for AWS Lambda functions running at the edge.
OTA Updates of AWS Greengrass Core Software

Note
This feature is only available for AWS Greengrass core v1.3.0.

The AWS Greengrass core software comes packaged with an OTA Update Agent that is capable of updating the core's software or the OTA Update Agent itself to the latest respective versions. You can start an update by invoking the CreateSoftwareUpdateJob API or from the Greengrass console. Updating the Greengrass core software provides the following benefits:

- Fix security vulnerabilities.
- Address software stability issues.
- Deploy new or improved features.

An OTA update makes all these benefits available without having to perform the update manually or having the device which is running the core software physically present. The OTA Update Agent also performs a rollback in case of a failed OTA update. Performing an OTA update is optional but can help you manage your AWS Greengrass core devices. Look for announcements of new versions of the core's software on the Greengrass developer forum.

In order to support an OTA update of Greengrass core software by using the OTA Update Agent, your Greengrass core device must:

- Have available local storage three times the amount of the core's runtime usage requirement.
- Not have trusted boot enabled in the partition containing the Greengrass core platform software. (The AWS Greengrass core can be installed and run on a partition with trusted boot enabled, but cannot perform an OTA update.)
- Have read/write permissions on the partition containing the Greengrass core platform software.
- Have a connection to the AWS cloud.
- Have a correctly configured AWS Greengrass core and appropriate certificates.

Before launching an OTA Update of Greengrass core software, it is important to note the impact that it will have on the devices in your Greengrass group, both on the core device and on client devices connected locally to that core:

- The core will be shut down during the update.
- Any Lambda functions running on the core will be shut down. If those functions write to local resources, they might leave those resources in an incorrect state unless shut down properly.
- During the core's downtime, all its connections with the cloud will be lost and messages routed through the core by client devices will be lost.
- Credential caches will be lost.
- Queues which hold pending work for Lambda functions will be lost.
- Long-lived Lambdas will lose their dynamic state information and all pending work will be dropped.

The following state information will be preserved during an OTA Update:

- Local shadows
Greengrass OTA Agent

The Greengrass OTA Agent is the software component on the device which handles update jobs created and deployed in the cloud. The Greengrass OTA Agent is distributed in the same software package as the Greengrass core software. The agent is located in `/greengrass/ota/ota_agent/ggc-ota` and creates its logs in `/var/log/greengrass/ota/ggc-ota.txt`.

You can start the Greengrass OTA Agent by executing the binary manually or by integrating it as part of an init script such as a systemd service file. The binary should be run as root. Once started, the Greengrass OTA Agent will begin listening for Greengrass update jobs from the cloud and execute them sequentially. The Greengrass OTA Agent will ignore all other IoT job types.

Do not start multiple OTA Agent instances as this may cause conflicts.

If your Greengrass core or Greengrass OTA Agent is managed by an init system, see Integration With Init Systems (p. 87) for related configurations.

CreateSoftwareUpdateJob API

The CreateSoftwareUpdateJob API creates a software update for a core or for several cores. This API can be used to update the OTA Agent as well as the Greengrass core software. It makes use of the AWS IoT Jobs feature which provides additional commands to manage a Greengrass core software update job. See Jobs for more information on how to manage a Greengrass Update.

The following example shows how to create a Greengrass core software update job using the CLI:

```bash
aws greengrass create-software-update-job \
  --update-targets-architecture x86_64 \ 
  --update-targets arn:aws:iot:us-east-1:123456789012:thing/myDevice \ 
  --update-targets-operating-system ubuntu \ 
  --software-to-update core \ 
  --s3-url-signer-role arn:aws:iam::123456789012:role/IotS3UrlPresigningRole \ 
  --update-agent-log-level WARN \ 
  --amzn-client-token myClientToken1
```

The `create-software-update-job` command returns a JSON object containing the job id and job ARN:

```json
{
  "IotJobId": "Greengrass-OTA-c3bd7f36-ee80-4d42-8321-alda0e5b1303",
  "IotJobArn": "arn:aws:iot:us-east-1:123456789012:job/Greengrass-OTA-c3bd7f36-ee80-4d42-8321-alda0e5b1303"
}
```

The `create-software-update-job` command has the following parameters:

--update-targets-architecture

The architecture of the core device. Must be one of `armv7l`, `x86_64` or `aarch64`.

--update-targets

A list of the targets to which the OTA update should be applied. The list can contain the ARNS of things which are cores, and the ARNs of thing groups whose members are cores. See IoT thing groups for more information on how to place cores in an IoT thing group.
The operating system of the core device. Must be one of ubuntu, amazon_linux or raspbian.

--software-to-update

Specifies whether the core's software or the OTA Agent software should be updated. Must be one of core or ota_agent.

--s3-url-signer-role

The IAM role which is used to presign the S3 url which links to the Greengrass software update. You must provide a role that has the appropriate policy attached. Here is an example policy document with the minimum required permissions:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowsIotToAccessGreengrassOTAUpdateArtifacts",
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": [
        "arn:aws:s3:::eu-central-1-greengrass-updates/*",
        "arn:aws:s3:::us-east-1-greengrass-updates/*",
        "arn:aws:s3:::ap-northeast-1-greengrass-updates/*",
        "arn:aws:s3:::us-west-2-greengrass-updates/*",
        "arn:aws:s3:::ap-southeast-2-greengrass-updates/*",
        "arn:aws:s3:::ap-southeast-2-greengrass-updates/*"
      ]
    }
  ]
}
```

Here is an example Assume Role policy document with the minimum required trusted entities:

```json
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": "iot.amazonaws.com"
      },
      "Effect": "Allow",
      "Sid": "AllowIotToAssumeRole"
    }
  ]
}
```

--amzn-client-token

[Optional] A client token used to make idempotent requests. Provide a unique token to prevent duplicate updates from being created due to internal retries.

--update-agent-log-level

[Optional] The logging level for log statements generated by the OTA Agent. Must be one of NONE, TRACE, DEBUG, VERBOSE, INFO, WARN, ERROR, or FATAL. The default is ERROR.

Here is an example IAM policy with the minimum permissions required to call the API:
Note
Because Greengrass is only supported on a subset of the architecture and operating system combinations possible with this command, `CreateSoftwareUpdateJob` will reject requests except for the following supported platforms:

- ubuntu/x86_64
- ubuntu/aarch64
- amazon_linux/x86_64
- raspbian/armv7l

Integration with Init systems

During an OTA update, binaries, some of which may be running, will be updated and restarted. This may cause conflicts if an init system is monitoring the state of either the AWS Greengrass core software or the Greengrass OTA Agent during the update. To help integrate the OTA update mechanism with your monitoring strategies, Greengrass provides the opportunity for user-defined shell scripts to run before and after an update. To tell the OTA agent to run these shell scripts, you must include the `managedRespawn = true` flag in the `.greengrass/config/config.json` file. For example:

```json
{
  "coreThing": {
    ...
  },
  "runtime": {
    ...
  },
  "managedRespawn": true
}
```
When the `managedRespawn` flag is set, the scripts must exist in the directory or the OTA Agent will fail the update. The directory tree should look as follows:

```
<greengrass_root>
|-- certs
| -- config
|   |-- config.json
|-- ggc
|-- usr/scripts
| |-- ggc_pre_update.sh
| |-- ggc_post_update.sh
| |-- ota_pre_update.sh
| |-- ota_post_update.sh
|-- ota
```

**OTA Self-Update with Managed Respawn**

As the OTA Agent prepares to do a self-update, if the `managedRespawn` flag is set to `true` then the OTA Agent will look in the `.greengrass/usr/scripts` directory for the `ota_pre_update.sh` script and run it.

After the OTA Agent completes the update, it will attempt to run the `ota_post_update.sh` script from the `.greengrass/usr/scripts` directory.

**AWS Greengrass core Update with Managed Respawn**

As the OTA Agent prepares to do an AWS Greengrass core update, if the `managedRespawn` flag is set to `true`, then the OTA Agent will look in the `.greengrass/usr/scripts` directory for the `ggc_pre_update_script.sh` script and run it.

After the OTA Agent completes the update, it will attempt to run the `ggc_post_update.sh` script from the `.greengrass/usr/scripts` directory.

**Note:**

- The user-defined scripts in `.greengrass/usr/scripts` should be owned by root and executable by root only.
- If `managedRespawn` is set to `true`, the scripts must exist and return a successful return code.
- If `managedRespawn` is set to `false`, the scripts will not be run even if present on the device.
- It is imperative that a device which is the target of an update not run two OTA agents for the same AWS IoT thing. Doing so will cause the two OTA Agents to process the same jobs which will lead to conflicts.

**OTA Agent Self-Update**

To perform an OTA Agent self-update follow these steps:

1. Ensure that the AWS Greengrass core is correctly provisioned with valid `config.json` file entries and the necessary certificates.
2. If the OTA Agent is being managed by an init system, ensure that `managedRespawn = true` in the `config.json` file and the scripts `ota_pre_update.sh` and `ota_post_update.sh` are present in the `.greengrass/usr/scripts` directory.
3. Start the ggc-ota agent by running `.greengrass/ota/ota_agent/ggc-ota`. 
4. Create an OTA self update job in the cloud with the CreateSoftwareUpdateJob API (aws greengrass create-software-update-job), making sure the --software-to-update parameter is set to ota_agent.
5. The OTA Agent will perform a self update.

**Greengrass Core Software Update**

To perform a AWS Greengrass core software update follow these steps:

1. Ensure that the AWS Greengrass core is correctly provisioned with valid config.json file entries and the necessary certificates.
2. If the AWS Greengrass core software is being managed by an init system, ensure that managedRespawn = true in the config.json file and the scripts ggc_pre_update.sh and ggc_post_update.sh are present in the ./greengrass/usr/scripts directory.
3. Start the ggc-ota agent by running ./greengrass/ota/ota_agent/ggc-ota.
4. Create an OTA self update job in the cloud with the CreateSoftwareUpdateJob API (aws greengrass create-software-update-job), making sure the --software-to-update parameter is set to core.
5. The OTA Agent will perform an update of AWS Greengrass core software.

**Limitations**

When the OTA Agent is started, it expects to be able to establish a connection to the AWS cloud to perform initialization. If the connection is unavailable within 30 seconds, the Greengrass OTA Agent will exit. If you want to start the OTA Agent in a disconnected environment, a workaround is to have an init script respawn the OTA Agent if it exits during startup with a return code of ETIMEDOUT (-110).
Reset Deployments

GGC v1.3.0

You may want to reset a group's deployments in order to:

- Delete the group (for example, when the group's core has been reimaged.)
- Move the group's core to a different group.
- Revert the group to its state prior to any deployments.
- Remove the deployment configuration from the core device.
- Delete sensitive data from the core device or from the cloud.
- Deploy a new group configuration to a core without having to replace the core with another in the current group.

The `ResetDeployments` command will clean up all deployment information which is stored in the cloud for a given group. It will then instruct the group's core device to clean up all of its deployment related information as well (Lambda functions, user logs, shadow database and server certificate, but not the user defined config.json or the Greengrass core certificates.) You cannot initiate a reset of deployments for a group if the group currently has a deployment with status `Pending` or `Building`.

```
aws greengrass reset-deployments --group-id <GroupId> [--force]
```

**Arguments for the `reset-deployments` CLI command:**

- `--group-id`
  
  The group ID.

- `--force`
  
  [Optional] Use this parameter if the group's core device has been lost, stolen or destroyed. This option causes the reset deployment process to report success once all deployment information in the cloud has been cleaned up, without waiting for a core device to respond. However, if the core device is or becomes active, it will perform its clean up operations as well.

The output of the `reset-deployments` CLI command will look like this:

```
{
  "DeploymentId": "4db95ef8-9309-4774-954a-eea580b6ceef",
  "DeploymentArn": "arn:aws:greengrass:us-west-2:106511594199:/greengrass/groups/b744ed45-a7df-4227-860a-8d4492aa412/deployments/4db95ef8-9309-4774-954a-eea580b6ceef"
}
```

You can check the status of the reset deployment with the `get-deployment-status` CLI command:

```
aws greengrass get-deployment-status --deployment-id DeploymentId --group-id GroupId
```
Arguments for the `get-deployment-status` CLI command:

--deployment-id
The deployment ID.
--group-id
The group ID.

The output of the `get-deployment-status` CLI command will look like this:

```
{
    "DeploymentStatus": "Success",
    "UpdatedAt": "2017-04-04T00:00:00.000Z"
}
```

The `DeploymentStatus` is set to Building when the reset deployment is being prepared. When the reset deployment is ready but the AWS Greengrass core has not picked up the reset deployment, the `DeploymentStatus` is InProgress.

GGC v1.1.0

You may want to reset a group's deployments in order to:

- Delete the group (for example, when the group's core has been reimaged.)
- Move the group's core to a different group.
- Revert the group to its state prior to any deployments.
- Remove the deployment configuration from the core device.
- Delete sensitive data from the core device or from the cloud.
- Deploy a new group configuration to a core without having to replace the core with another in the current group.

The `ResetDeployments` command will clean up all deployment information which is stored in the cloud for a given group. It will then instruct the group's core device to clean up all of its deployment related information as well (Lambda functions, user logs, shadow database and server certificate, but not the user defined config.json or the Greengrass core certificates.) You cannot initiate a reset of deployments for a group if the group currently has a deployment with status Pending or Building.

```
aws greengrass reset-deployments --group-id <GroupId> [--force]
```

Arguments for the `reset-deployments` CLI command:

--group-id
The group ID.
--force
[Optional] Use this parameter if the group's core device has been lost, stolen or destroyed. This option causes the reset deployment process to report success once all deployment information in the cloud has been cleaned up, without waiting for a core device to respond. However, if the core device is or becomes active, it will perform its clean up operations as well.

The output of the `reset-deployments` CLI command will look like this:
You can check the status of the reset deployment with the `get-deployment-status` CLI command:

```
aws greengrass get-deployment-status --deployment-id DeploymentId --group-id GroupId
```

**Arguments for the `get-deployment-status` CLI command:**

--deployment-id

The deployment ID.

--group-id

The group ID.

The output of the `get-deployment-status` CLI command will look like this:

```
{
  "DeploymentStatus": "Success",
  "UpdatedAt": "2017-04-04T00:00:00.000Z"
}
```

The `DeploymentStatus` is set to Building when the reset deployment is being prepared. When the reset deployment is ready but the AWS Greengrass core has not picked up the reset deployment, the `DeploymentStatus` is InProgress.

**GGC v1.0.0**

The ResetDeployments feature is not available in this version. Also, please note that it is not possible to delete a group that has been deployed.
Access Local Resources with Lambda Functions

Note
This feature is only available for AWS Greengrass core v1.3.0.

Developers who use AWS Greengrass can author AWS Lambda functions in the cloud and deploy them to core devices for local execution. On Greengrass cores running Linux, these locally deployed Lambda functions can access local resources that are physically present on the Greengrass core device. Two types of resources can be accessed — volumes (files or directories on the root file system, except those under /sys, /proc and /dev) and devices (files under /dev). For example:

- Volume resources:
  - Folders and files (e.g., /usr/lib/python2.x/site-packages/local) to read/write information across Greengrass Lambda functions.
- Device resources:
  - Serial ports (e.g., /dev/ttyS0, /dev/ttyS1) used to communicate with devices connected via serial ports.
  - USB (e.g., /dev/ttyUSB0 or /dev/bus/usb) used to connect USB peripherals.
  - GPIOs (e.g., /dev/gpiomem) used for sensors and actuators via GPIO.
  - GPUs (e.g., /dev/nvidia0) used to accelerate machine learning using on-board GPUs.
  - Cameras (e.g., /dev/video0) to capture images and videos.

Note: an exception is /dev/shm which can be configured as a volume resource.

Prerequisites

- Access to local resources is only available with AWS Greengrass Core v1.3.0 and above.
- The local resource (including any required drivers and libraries) must be properly installed on the Greengrass core host device and consistently available during use.
- The desired operation of the resource, and access to the resource, must not require root privileges.
- You must provide the full path of the local resource on the host operating system.
- You must verify that the Lambda function process has the required file access to the local resource.

How to Configure Local Resource Access

In order to use a local resource, you must add a resource definition to the group definition which will be deployed to your Greengrass core device. The group definition must also contain a Lambda function definition in which you grant access permissions for local resources to your Lambda functions.

In what follows, we describe the process of creating a local resource and configuring access to it using the AWS CLI. We assume you already have created a Greengrass group as described in Getting Started with AWS Greengrass (p. 12).

First, you create a resource definition that specifies the resources to be accessed by using the command CreateResourceDefinition. In this example, we create two resources TestDirectory and TestCamera:
aws greengrass create-resource-definition --cli-input-json '{
   "Name": "MyLocalVolumeResource",
   "InitialVersion": {
      "Resources": [
         {
            "Id": "data-volume",
            "Name": "TestDirectory",
            "ResourceDataContainer": {
               "LocalVolumeResourceData": {
                  "SourcePath": "/src/LRAtest",
                  "DestinationPath": "/dest/LRAtest",
                  "GroupOwnerSetting": {
                     "AutoAddGroupOwner": true,
                     "GroupOwner": ""
                  }
               }
            }
         },
         {
            "Id": "data-device",
            "Name": "TestCamera",
            "ResourceDataContainer": {
               "LocalDeviceResourceData": {
                  "SourcePath": "/dev/video0",
                  "GroupOwnerSetting": {
                     "AutoAddGroupOwner": true,
                     "GroupOwner": ""
                  }
               }
            }
         }
      ]
   }
}'

**Resources**: A list of Resource objects in the Greengrass group. One Greengrass group can have up to 50 resources.

**ResourceId**: The unique identifier of the resource. The id is used to refer a resource in the Lambda function configuration. Max length 128 characters. Pattern: [a-zA-Z0-9-._]*.

**ResourceName**: The name of the resource. The resource name is displayed on the Greengrass console. Max length 128 characters. Pattern: [a-zA-Z0-9-._]*.

**LocalVolumeResourceData#SourcePath**: The local absolute path of the volume resource on the host. The source path for a volume resource type cannot start with /proc or /sys.

**LocalDeviceResourceData#SourcePath**: The local absolute path of the device resource. The source path for a device resource can only refer to a character device or block device under /dev.

**LocalVolumeResourceData#DestinationPath**: The absolute path of the volume resource inside the Lambda environment.

**GroupOwnerSetting**: Allows you to configure additional group privileges for the Lambda process. This field is optional.

**GroupOwnerSetting#AutoAddGroupOwner**: If true, Greengrass automatically adds the specified Linux OS group owner of the resource to the Lambda process privileges. Thus the lambda process will have the file access permissions of the added Linux group.

**GroupOwnerSetting#GroupOwner**: Specifies the name of the Linux OS group whose privileges will be added to the lambda process. This field is optional.
A resource definition version ARN will be returned by CreateResourceDefinition and should be used when updating a group definition. For example:

```json
{
    "Name": "MyLocalVolumeResource",
    "LastUpdatedTimestamp": "2017-11-15T01:18:42.153Z",
    "LatestVersion": "a4d9b882-d025-4760-9cfe-9d4fada5390d",
    "CreationTimestamp": "2017-11-15T01:18:42.153Z",
    "Id": "ab14d0b5-116e-4951-a322-9cde24a30373",
    "Arn": "arn:aws:greengrass:us-west-2:123456789012:/greengrass/definition/resources/ab14d0b5-116e-4951-a322-9cde24a30373"
}
```

After the resources are created, create the Greengrass function using CreateFunctionDefinition and grant the function access to the resource:

```bash
aws greengrass create-function-definition --cli-input-json '{
    "Name": "MyFunctionDefinition",
    "InitialVersion": {
        "Functions": [
            {
                "Id": "greengrassLraTest",
                "FunctionConfiguration": { 
                    "Pinned": false,
                    "MemorySize": 16384,
                    "Timeout": 120,
                    "Environment": { 
                        "ResourceAccessPolicies": [ 
                            { 
                                "ResourceId": "data-volume",
                                "Permission": "rw"
                            },
                            { 
                                "ResourceId": "data-device",
                                "Permission": "ro"
                            }
                        ],
                        "AccessSysfs": true
                    }
                }
            }
        ]
    }
}
```

**ResourceAccessPolicies**: Contains the resourceId and permission which grant the Lambda access to the resource. A Lambda function can have at most 10 resources.

**ResourceAccessPolicy#Permission**: Specifies which permissions the Lambda has on the resource. The available options are `rw` (read/write) or `ro` (readonly).

**AccessSysfs**: If true, the Lambda process can have read access to the host `/sys` folder. This is used in cases where the Greengrass Lambda needs to read device information from `/sys`.

Again, a function definition version ARN is returned by CreateFunctionDefinition and should be used in your group definition version.
How to Configure Local Resource Access

Finally, use CreateGroupVersion to add the function to the group. For example:

```bash
aws greengrass create-group-version --group-id "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5" \
  greengrass/definition/resources/db6bf40b-29d3-4c4e-9574-21ab7d74916c/versions/31d0010f-
  e19a-4c4c-8098-688b799066f887" \
  greengrass/definition/cores/adbf3475-f6f3-48e1-84d6-502f02729067/
  versions/297c419a-9de6-46dd-8ccc-341fc67031e8\b" \
  greengrass/definition/functions/d1123830-da38-4c4c-9eb7-e92ecb76d3e/versions/a2e90400-
  caea-4fdd-b23a-db1892a3c3c8" \
  greengrass/definition/subscriptions/7a8ef3d8-1de3-426c-9554-5b5a12b5bc6/
  versions/470c858c-7eb3-4abd-9d48-230236bfb6a"
```

A new group version is returned:

```json
{
  "Arn": "arn:aws:greengrass:us-west-2:012345678901:/greengrass/groups/
  b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5/versions/291917fb-ec54-4895-823e-27b52da25481",
  "Version": "291917fb-ec54-4895-823e-27b52da25481",
  "CreationTimestamp": "2017-11-22T01:47:22.487Z",
  "Id": "b36a3aeb-3243-47ff-9fa4-7e8d98cd3cf5"
}
```

You now have a Greengrass group containing a Lambda function "lraTest" that has access to two resources: TestDirectory and TestCamera.

Here is an example Lambda function, "lraTest.py", written in python, which writes to the local volume resource:

```python
# lraTest.py
# Demonstrates a simple use case of local resource access.
# This Lambda function writes a file "test" to a volume mounted inside
# the lambda function under "/dest/LRAtest". Then it reads the file and
# publishes the content to AWS IoT 'LRA/test' topic.
import time
import sys
import greengrasssdk
import platform
import os
import logging
```

96
# Creating a greengrass core sdk client
```python
client = greengrasssdk.client('iot-data')
volumePath = '/dest/LRA/test'
```
def function_handler(event, context):
    try:
        volumeInfo = os.stat(volumePath)
        client.publish(topic='LRA/test', payload=str(volumeInfo))
        with open(volumePath + '/test', 'a') as output:
            output.write('Successfully write to a file.\n')
        with open(volumePath + '/test', 'r') as myfile:
            data=myfile.read()
        client.publish(topic='LRA/test', payload=data)
    except Exception as e:
        logging.error("Experiencing error :{}".format(e))
time.sleep(5)
return
```

These commands are provided by the Greengrass API to create and manage resource definitions and resource definition versions:

- CreateResourceDefinition
- CreateResourceDefinitionVersion
- DeleteResourceDefinition
- GetResourceDefinition
- GetResourceDefinitionVersion
- ListResourceDefinitions
- ListResourceDefinitionVersions
- UpdateResourceDefinition

Here are some important things to consider when using locally deployed Lambda functions to access local resources:

- You must guarantee the security of:
  - Your physical hardware.
  - Your Greengrass core host device OS.
- Constraints:
  - You must provide the full path of the local resource.
  - Only those devices under /dev are supported.
  - Only a regular file or directory is allowed for a volume resource.
  - You cannot configure a file under /sys, /proc, /var, /var/run, and /var/lib as a volume resource.
  - Only read/write permissions are available. Lambdas cannot perform privileged operations on the resources.

## Troubleshooting

- **Q:** Why does my Greengrass group deployment fail with an error similar to:

  ```
  group config is invalid:
  ggc_user or [ggc_group root tty] don't have ro permission on the file: /dev/tty0
  ```
A: This error indicates that the Lambda process doesn’t have permission to access the specified resource. The solution is to change the file permission of the resource so that Lambda can access it (see Group Owner File Access Permission (p. 98) for more details).

Q: When I configure /var/run as a volume resource, why does the Lambda function fail to start with an error message in the runtime.log:

```plaintext
[ERROR]-container_process.go:39, Runtime execution error: unable to start lambda container.
container_linux.go:259: starting container process caused "process_linux.go:345: container init caused "rootfs_linux.go:62: mounting \"/var/run\" to rootfs \"/greengrass/ggc/packages/1.3.0/rootfs_sys\" at \"/greengrass/ggc/packages/1.3.0/rootfs_sys/run\"
caused \"invalid argument\""
```

A: GGC currently doesn’t support the configuration of /var, /var/run and /var/lib as volume resources. One workaround is to first mount /var, /var/run or /var/lib in a different folder and then configure the folder as a volume resource.

Q: When I configure /dev/shm as a volume resource with readonly permission, why does the Lambda function fail to start with an error in the runtime.log:

```plaintext
[ERROR]-container_process.go:39, Runtime execution error: unable to start lambda container.
container_linux.go:259: starting container process caused "process_linux.go:345: container init caused "rootfs_linux.go:62: mounting \"/dev/shm\" to rootfs \"/greengrass/ggc/packages/1.3.0/rootfs_sys\" at \"/greengrass/ggc/packages/1.3.0/rootfs_sys/dev/shm\"
caused \"operation not permitted\"
```

A: /dev/shm can only be configured as read/write. Changing the resource permission to rw will resolve the issue.

**Group Owner File Access Permission**

An AWS Greengrass Lambda function process normally runs as ggc_user and ggc_group. It may also receive additional permissions as specified in the local resource definition with the GroupOwnerSetting#AutoAddGroupOwner flag or the GroupOwnerSetting#GroupOwner field. If the GroupOwnerSetting#AutoAddGroupOwner flag is set to true, the group owner of the resource is automatically added to the Lambda function process. The group specified in the GroupOwnerSetting#GroupOwner field will also be added to the Lambda process.

An AWS Greengrass Lambda function process will have all the file system permissions that ggc_user, ggc_group and the added groups have. In order for the Lambda function to access a resource, you need to make sure that the Lambda process has the required file permissions to the resource, using the chmod(1) command to change the permission of the resource, if necessary.
Greengrass Discovery RESTful API

All devices that communicate with an AWS Greengrass core must be a member of a Greengrass group. Each group must have an AWS Greengrass core. The Discovery API enables devices to retrieve information required to connect to an AWS Greengrass core that is in the same Greengrass group as the device. When a device first comes online, it can connect to the AWS Greengrass cloud service and use the Discovery API to find:

- The group to which it belongs.
- The IP address and port for the AWS Greengrass core in the group.
- The group's root CA certificate, which can be used to authenticate the AWS Greengrass core device.

To use this API, send HTTP requests to the following URI:

`https://your-aws-endpoint/greengrass/discover/thing/thing-name`

The endpoint is specific to your AWS account. To retrieve your endpoint, use the `aws iot describe-endpoint` CLI command:

```
$ aws iot describe-endpoint
{
  "endpointAddress": "a1b2c3d4e5f6g7.iot.us-west-2.amazonaws.com"
}
```

Use port 8443 when connecting. For a list of region-specific endpoints, see AWS IoT Regions and Endpoints in the AWS General Reference. This is a data plane only API. The endpoints used for working with rules, certificates, and policies do not support the Discovery API.

Request

The request contains the standard HTTP headers and is sent to the following URI:

```
HTTP GET https://your-aws-endpoint/greengrass/discover/thing/thingName
```

Response

Upon success, the response includes the standard HTTP headers plus the following code and body:

```
HTTP 200
BODY: response document
```

For more information see, Example Discover Response Documents (p. 100).
Authorization

Retrieving the connectivity information requires a policy that allows the caller to perform the `greengrass:Discover` action. TLS mutual authentication with a client certificate is the only accepted form of authentication. The following is an example policy that allows a caller to perform this action:

```json
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "greengrass:Discover",
  }]
}
```

Example Discover Response Documents

The following document shows the response for a device that is a member of a group with one AWS Greengrass core, one endpoint, and one group CA:

```json
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": "core-01-port",
              "metadata": "core-01-description"
            }
          ]
        }
      ],
      "CAs": ["-----BEGIN CERTIFICATE-----
              cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```

The following document shows the response for a device that is a member of two groups with one AWS Greengrass core, multiple endpoints, and multiple group CAs:

```json
{
  "GGGroups": [
    {
      "GGGroupId": "gg-group-01-id",
      "Cores": [
        {
          "thingArn": "core-01-thing-arn",
          "Connectivity": [
            {
              "id": "core-01-connection-id",
              "hostAddress": "core-01-address",
              "portNumber": "core-01-port",
              "metadata": "core-01-description"
            }
          ]
        }
      ],
      "CAs": ["-----BEGIN CERTIFICATE-----
              cert-contents-----END CERTIFICATE-----"
      ]
    },
    {
      "GGGroupId": "gg-group-02-id",
      "Cores": [
        {
          "thingArn": "core-02-thing-arn",
          "Connectivity": [
            {
              "id": "core-02-connection-id",
              "hostAddress": "core-02-address",
              "portNumber": "core-02-port",
              "metadata": "core-02-description"
            }
          ]
        }
      ],
      "CAs": ["-----BEGIN CERTIFICATE-----
              cert-contents-----END CERTIFICATE-----"
      ]
    }
  ]
}
```
"hostAddress": "core-01-address",
"portNumber": "core-01-port",
"metadata": "core-01-connection-1-description"
},
{
  "id": "core-01-connection-id-2",
  "hostAddress": "core-01-address-2",
  "portNumber": "core-01-port-2",
  "metadata": "core-01-connection-2-description"
}
]
]
"CAs": [
  "-----BEGIN CERTIFICATE-----
cert-contents
-----END CERTIFICATE-----",
  "-----BEGIN CERTIFICATE-----
cert-contents
-----END CERTIFICATE-----",
  "-----BEGIN CERTIFICATE-----
cert-contents
-----END CERTIFICATE-----",
]

"GGGroupId": "gg-group-02-id",
"Cores": [
  {
    "thingArn": "core-02-thing-arn",
    "Connectivity": [
      {
        "id": "core-02-connection-id",
        "hostAddress": "core-02-address",
        "portNumber": "core-02-port",
        "metadata": "core-02-connection-1-description"
      }
    ],
    "CAs": [
      "-----BEGIN CERTIFICATE-----
cert-contents
-----END CERTIFICATE-----",
      "-----BEGIN CERTIFICATE-----
cert-contents
-----END CERTIFICATE-----",
      "-----BEGIN CERTIFICATE-----
cert-contents
-----END CERTIFICATE-----"
    ]
  }
]

**Note**

An AWS Greengrass group must define exactly one AWS Greengrass core. Any response from the AWS Greengrass cloud service that contains a list of AWS Greengrass cores only contains one AWS Greengrass core.
Use Greengrass OPC-UA to Communicate with Industrial Equipment

Greengrass supports OPC-UA, an information exchange standard for industrial communication. OPC-UA allows you to ingest and process messages from industrial equipment and deliver them to devices in your Greengrass group or to the cloud based on rules you define.

The Greengrass implementation of OPC-UA supports certificate-based authentication. It is based on an open-source implementation, and is fully customizable. You can also bring your own implementation of OPC-UA, and implement your own support for other custom, legacy, and proprietary messaging protocols.

In this section we will cover the following steps:

- Connect to an existing OPC-UA server.
- Monitor an existing OPC-UA node within that server.
- Get called back when the monitored node's value changes.

Architectural Overview

Greengrass implements OPC-UA as a Lambda function in NodeJS. Since Lambda functions running on Greengrass cores have access to network resources, you can create Lambda functions that proxy information from your existing OPC-UA servers over TCP to other functions or services in your Greengrass group.

You can configure Greengrass to have a long-lived connection to your OPC-UA server(s), and, using OPC-UA Subscriptions, you can have your OPCUA_Adapter Lambda function monitor changes to predefined nodes. Any change to those nodes triggers a Publish event from the OPC-UA server, which will be received by your Lambda function, and republished into predefined topic names.

The topic structure is constructed as follows:
Set Up a Test OPC-UA Server

Use the following commands to set up a test OPC-UA server. Or, if you already have an OPC-UA server you'd like to use instead, you may skip this step.

```
git clone git://github.com/node-opcua/node-opcua.git
cd node-opcua
git checkout v0.0.64
npm install
node bin/simple_server
```

The server produces the following output:

```
[ec2-user@<your_instance_id> node-opcua]$ node bin/simple_server
server PID          : 28585
registering server to :opc.tcp://<your_instance_id>4840/UADiscovery
err Cannot find module 'usage'
skipping installation of cpu_usage and memory_usage nodes
server on port      : 26543
epointUrl           : opc.tcp://<your_instance_id>us-west-2.compute.internal:26543
serverInfo          :
  applicationUri                  : urn:54f7890cca4c49a1:NodeOPCUA-Server
  productUri                      : NodeOPCUA-Server
  applicationName                 : locale=en text=NodeOPCUA
  applicationType                 : SERVER
  gatewayServerUri                : null
  discoveryProfileUri             : null
  discoveryUrls                   :
    productName                     : NODEOPCUA-SERVER
  buildInfo           :
    productUri                     : NodeOPCUA-Server
    manufacturerName               : Node-OPCUA : MIT Licence ( see http://node-opcua.github.io/)
    productName                    : NODEOPCUA-SERVER
    softwareVersion                 : 0.0.65
    buildNumber                     : 1234
    buildDate                       : Thu Aug 03 2017 00:13:50 GMT+0000 (UTC)
server now waiting for connections. CTRL+C to stop
```

Make sure your Greengrass Group is ready

- Create a Greengrass group (find more details in Configuring AWS Greengrass on AWS IoT (p. 21).)
- Set up a Greengrass Core on one of the supported platforms (Raspberry-pi for example (p. 13))
Use Greengrass OPC-UA to Interact with your OPC-UA Server

1. Prepare your Lambda function

Get the code for an OPC-UA adapter Lambda function from GitHub:

```bash
git clone https://github.com/aws-samples/aws-greengrass-samples.git
cd aws-greengrass-samples/greengrass-opcua-adapter-nodejs
npm install
```

**Note:** This Lambda function uses the node-opcua library (v0.0.64), which attempts to re-generate some model files at runtime. That doesn't work when running as a Lambda function on Greengrass, because Lambda functions start with a Read-Only file system, so any code trying to generate other code would not work. The next step fixes this.

2. Change the file at `node_modules/node-opcua/lib/misc/factories.js` line 109 to this:

```javascript
var generated_source_is_outdated = (!generated_source_exists);
```

Run this command to make that change:

```bash
sed -i '109s/.*/    var generated_source_is_outdated = (!generated_source_exists);/'
node_modules/node-opcua/lib/misc/factories.js
```

3. Configure the server and monitored nodes

Change the `configSet` variable inside the `index.js` file of the OPC-UA Lambda function to contain the server IP and Port that you want to connect to, as well as the node IDs you would like to monitor. By default it comes with the following example configuration:

```javascript
const configSet = {
    server: {
        name: 'server',
        url: 'opc.tcp://localhost:26543',
    },
    subscriptions: [
        {
            name: 'MyPumpSeed',
            nodeId: 'ns=1;s=PumpSeed',
        },
    ],
};
```

In this case, we are connecting to an OPC-UA server running on the same host as our Greengrass Core, on port 26543, and monitoring one node that has an OPC-UA ID 'ns=1;s=PumpSeed'.

4. Configure the authentication mode

The **OPC-UA library** used in this example supports three modes of Authentication to your OPC-UA server. The most secure method is Certificate Based Authentication, but the library also allows you to specify username/password or no authentication.
Here is how to set Certificate Based Authentication:

- Package your certificate and private key with your Lambda function, for example under a directory named `certs/`.
- Change the `clientOptions` variable to contain `certificateFile`, `privateKeyFile` and `securityModes`, `securityPolicies` options:

```javascript
const clientOptions = {
    keepSessionAlive: true,
    certificateFile: '/lambda/certs/<certificate_name>.pem.crt',
    privateKeyFile: '/lambda/certs/<private_key_name>.pem.key',
    securityModes: MessageSecurityMode.SIGN,
    securityPolicies: SecurityPolicy.BASIC256,
    connectionStrategy: {
        maxRetry: 1000000,
        initialDelay: 2000,
        maxDelay: 10 * 1000,
    },
};
```

5. **Upload your Lambda**

Create a Greengrass Lambda function. You can find more details on how to do that in Configure Lambda for AWS Greengrass (p. 38). In a nutshell, create a Lambda function code archive by doing the following:

```
# Download the nodejs greengrass sdk from

# Install Greengrass SDK in the node_modules directory
tar -zxvf aws-greengrass-core-sdk-js-*.tar.gz -C /tmp/
unzip /tmp/aws_greengrass_core_sdk_js/sdk/aws-greengrass-core-sdk.zip -d node_modules

# Archive the whole directory as a zip file
zip -r opcuaLambda.zip * -x \.git\*

# Create an AWS Lambda with the created zip
aws lambda create-function --function-name <Function_Name> --runtime 'nodejs6.10' --role <Your_Role> --handler 'index.handler' --zip-file opcuaLambda.zip
```

Add this Lambda to your Greengrass Group. Details are, again, in: Configure Lambda for AWS Greengrass (p. 38).

6. **Configure and Deploy the Lambda function to your Greengrass Group**

After creating your AWS Lambda function, you add it to your Greengrass Group. Follow the instructions in same section as above.

- Make sure to specify the Lambda function as Long-Running.
- Give it at least 64MB of memory size.

You can now create a deployment with your latest configuration. You can find details in Deploying Cloud Configurations to an AWS Greengrass Core Device (p. 44).
Verify that your Lambda function is receiving OPC-UA Publishes and posting them onto Greengrass

As described in the Architecture section (p. 102), your Lambda function should start receiving messages from your OPC-UA server. If you are using your own custom OPC-UA server, make sure you trigger a change in the OPC-UA node Id you specified, so that you see the change received by your Lambda function. If you are using the example server above, the PumpSpeed node is configured to simulate a series of consecutive updates, so you should expect your Lambda function to receive multiple messages a second.

You can see messages received by your Lambda function in one of two ways:

- Watch the Lambda function's logs

  You can view the logs from your Lambda function by running the following command:

  ```
  sudo cat ggc/var/log/user/us-west-2/your_account_id/your_function_name.log
  ```

  The logs should look similar to:

  ```
  [2017-11-14T16:33:09.05Z][INFO]-started subscription : 305964
  [2017-11-14T16:33:09.05Z][INFO]-monitoring node id = ns=1;s=PumpSeed
  [2017-11-14T16:33:09.099Z][INFO]-monitoredItem initialized
  [2017-11-15T23:49:34.752Z][INFO]-Publishing message on topic "/opcua/server/node/MyPumpSpeed" with Payload "{"id":"ns=1;s=PumpSpeed","value":
  {"dataType":"Double","arrayType":"Scalar","value":237.5250759433095}}"
  ```

- Configure Greengrass to forward messages from your Lambda function to the IoT Cloud.

  Follow the steps outlined in Verify the Lambda Function Is Running on the Device (p. 45) to receive messages on the IoT console.

  **Note:**

  - Make sure there is a Subscription from your Lambda function going to the IoT Cloud. Details are in Configure Lambda for AWS Greengrass (p. 38).
  - Since messages are forwarded to the cloud, make sure you terminate either the example server you configured above, or stop the Greengrass core, so that you don't end up publishing a lot of messages to IoT cloud and getting charged for them!

Next Steps

With Greengrass, you can use this same architecture to create your own implementation of OPC-UA, and also implement your own support for custom, legacy, and proprietary messaging protocols. Since Lambda functions running on Greengrass cores have access to network resources, you can use them to implement support for any protocol that rides on top of TCP-IP. In addition, you can also take advantage of Greengrass Local Resource Access to implement support for protocols that need access to hardware adapters/drivers.
AWS Greengrass Security

AWS Greengrass uses X.509 certificates, managed subscriptions, AWS IoT policies, and IAM policies and roles to ensure your Greengrass applications are secure. AWS Greengrass core devices require an AWS IoT thing, a device certificate, and an AWS IoT policy to communicate with the Greengrass cloud service.

This allows AWS Greengrass core devices to securely connect to the AWS IoT cloud services. It also allows the Greengrass cloud service to deploy configuration information, Lambda functions, and managed subscriptions to AWS Greengrass core devices.

AWS IoT devices require an AWS IoT thing, a device certificate, and an AWS IoT policy to connect to the Greengrass service. This allows AWS IoT devices to use the Greengrass Discovery Service to find and connect to an AWS Greengrass core device. AWS IoT devices use the same device certificate used to connect to AWS IoT device gateway and AWS Greengrass core devices. The following diagram shows the components of the AWS Greengrass security model:

A - Greengrass service role

A customer-created IAM role that allows AWS Greengrass access to your AWS IoT and Lambda resources.

B - Core device certificate

An X.509 certificate used to authenticate an AWS Greengrass core.

C - Device certificate

An X.509 certificate used to authenticate an AWS IoT device.

D - Group role

A role assumed by AWS Greengrass when calling into the cloud from a Lambda function on an AWS Greengrass core.
E - Group CA

A root CA certificate used by AWS Greengrass devices to validate the certificate presented by an AWS Greengrass core device during TLS mutual authentication.

Configuring Greengrass Security

To configure your Greengrass application's security:

1. Create an AWS IoT thing for your AWS Greengrass core device.
2. Generate a key pair and device certificate for your AWS Greengrass core device.
3. Create and attach an AWS IoT policy to the device certificate. The certificate and policy allow the AWS Greengrass core device access to AWS IoT and Greengrass cloud services.
4. Create a Greengrass service role. This IAM role grants AWS Greengrass access to your Greengrass and AWS IoT resources. You only need to create a service role once per AWS account.
5. (Optional) Create a Greengrass group role. This role grants permission to Lambda functions running on an AWS Greengrass core to call other AWS services (in the cloud). You need to do this for each Greengrass group you create.
6. Create an AWS IoT thing for each device that will connect to your AWS Greengrass core.
7. Create device certificates, key pairs, and AWS IoT policies for each device that will connect to your AWS Greengrass core.

Note
You can also use existing AWS IoT things and certificates.

Device Connection Workflow

This section describes how devices connect to the AWS Greengrass cloud service and AWS Greengrass core devices.

- An AWS Greengrass core device uses its device certificate, private key, and the AWS IoT root CA certificate to connect to the Greengrass cloud service.
- The AWS Greengrass core device downloads group membership information from the Greengrass service.
- When a deployment is made to the AWS Greengrass core device, the Device Certificate Manager (DCM) handles certificate management for the AWS Greengrass core device.
- An AWS IoT device connects to the Greengrass cloud service using its device certificate, private key, and the AWS IoT root CA. After making the connection, the AWS IoT device uses the Greengrass Discovery Service to find the IP address of its AWS Greengrass core device. The device can also download the group's root CA certificate, which can be used to authenticate the Greengrass core device.
- An AWS IoT device attempts to connect to the AWS Greengrass core, passing its device certificate and client ID. If the client ID matches the thing name of the device and the certificate is valid, the connection is made. Otherwise, the connection is terminated.

Greengrass Messaging Workflow

A subscription table is used to define how messages are exchanged within a Greengrass group (between AWS Greengrass core devices, AWS IoT devices, and Lambda functions). Each entry in the subscription...
table specifies a source, a destination, and an MQTT topic over which messages are sent/received. Messages can be exchanged only if an entry exists in the subscription table specifying the source (message sender), the target (message recipient), and the MQTT topic. Subscription table entries specify passing messages in one direction, from the source to the target. If you want two-way message passing, create two subscription table entries, one for each direction.

**MQTT Core Server Certificate Rotation**

The MQTT core server certificate expires, by default, in 7 days. You can set the expiration to any value between 7 and 30 days. When the MQTT core server certificate expires, any attempt to validate the certificate fails. The device must be able to detect the failure and terminate the connection. Existing connections are not affected. When the certificate expires, the AWS Greengrass core device attempts to connect to the Greengrass cloud service to obtain a new certificate. If the connection is successful, the AWS Greengrass core device downloads a new MQTT core server certificate and restarts the local MQTT service. At this point, all AWS IoT devices connected to the core are disconnected.

If there is no internet connection when the AWS Greengrass core attempts to get a new MQTT core server certificate, AWS IoT devices are unable to connect to the AWS Greengrass core until the connection to the Greengrass cloud service is restored and a new MQTT core server certificate can be downloaded.

When AWS IoT devices are disconnected from a core, they have to wait a short period of time and then attempt to reconnect to the AWS Greengrass core device.

**AWS Greengrass Cipher Suites**

As opposed to the AWS IoT cloud, the AWS Greengrass core supports the following local network TLS cipher suites:

<table>
<thead>
<tr>
<th>TLS Version</th>
<th>Cipher</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLSv1.0</td>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA</td>
</tr>
<tr>
<td>TLSv1.1</td>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA</td>
</tr>
<tr>
<td>TLSv1.2</td>
<td>TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_128_GCM_SHA256</td>
</tr>
<tr>
<td>TLS Version</td>
<td>Cipher</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_CBC_SHA</td>
</tr>
<tr>
<td></td>
<td>TLS_RSA_WITH_AES_256_GCM_SHA384</td>
</tr>
</tbody>
</table>
Monitoring with AWS Greengrass Logs

GGC v1.3.0

AWS Greengrass consists of the cloud service and the AWS Greengrass core software. The AWS Greengrass core software can write logs to CloudWatch Logs or to the local file system of your AWS Greengrass core device. If the logging component is configured to log events to the local file system, the log files are stored under /greengrass/ggc/var/log with the following directory structure:

```
/greengrass/ggc/var
  - crash.log
  - system
    - log files for each Greengrass system components
  - user
    - log files generated by each Lambda function
```

Currently, for file system logging, we support size-based rotation and automatic cleanup when the amount of log data is close to the configured limit.

GGC v1.1.0

AWS Greengrass consists of the cloud service and the AWS Greengrass core software. The AWS Greengrass core software can write logs to CloudWatch Logs or to the local file system of your AWS Greengrass core device. If the logging component is configured to log events to the local file system, the log files are stored under /greengrass/ggc/var/log with the following directory structure:

```
/greengrass/ggc/var
  - crash.log
  - system
    - log files for each Greengrass system components
  - user
    - log files generated by each Lambda function
```

Currently, for file system logging, we support size-based rotation and automatic cleanup when the amount of log data is close to the configured limit.

GGC v1.0.0

AWS Greengrass consists of the cloud service and the AWS Greengrass core software. The AWS Greengrass core software can write logs to CloudWatch Logs or to the local file system of your AWS Greengrass core device. If the logging component is configured to log events to the local file system, the log files are stored under /greengrass/var/log with the following directory structure:

```
/greengrass/var
  - log
    - system
      - log files for each Greengrass system components
    - user
      - log files generated by each Lambda function
```

Current content is in v1.0.0.
Currently, for file system logging, we support size-based rotation and automatic cleanup when the amount of log data is close to the configured limit.

## Configuring Greengrass Logging

The logging component can be configured through the Greengrass logging list APIs with a payload similar to the following:

```json
{
    "loggingModelList": [
        {
            "Type": "log-storage-type",
            "Component": component-type,
            "Level": logging-level,
            "Space": max-storage
        },
        {
            "Type": ...
            "Level": ...
            ...
        }
    ]
}
```

**Type**

AWS CloudWatch and FileSystem. This field specifies which storage mechanism is used for log events. When AWS CloudWatch is used, log events are sent to CloudWatch with a limited number of retries in case there's no internet connectivity. After the retries are exhausted, the event is dropped. When FileSystem is used, log events are stored on the local file system.

**Component**

Supported values are GreengrassSystem and Lambda. When GreengrassSystem is used, log events from Greengrass system components are filtered based on the log level threshold and stored to the designated location. Depends on the Type. When Lambda is used, log events from a user's Lambda functions are filtered based on the log level and stored to the designated location. Again, depends on the Type.

**Level**

Log events below this threshold are filtered out and aren't stored.

**Space**

The maximum amount of local storage, in KB, to use for storing logs. This field applies only when Type is set to FileSystem.

If logging is not configured, before the first deployment, the AWS Greengrass core software does not contain any logging configuration information. It uses the following settings:

```json
[
    {
        "Type": "FileSystem",
        "Component": "GreengrassSystem",
        "Level": "INFO",
        "Space": 128
    }
]
```
After the first deployment, if you explicitly configure the Greengrass software to emit no logs, none will be emitted. If you don't configure logging, the following settings are used by default:

```json
[
  {
    "Type": "FileSystem",
    "Component": "GreengrassSystem",
    "Level": "INFO",
    "Space": 128
  },
  {
    "Type": "FileSystem",
    "Component": "Lambda",
    "Level": "INFO",
    "Space": 128
  }
]
```

### Required IAM Permissions

To enable logging to CloudWatch, the following CloudWatch Logs actions must be present in the AWS Greengrass group role:

- `logs:PutLogEvents`
- `logs:CreateLogGroup`
- `logs:CreateLogStream`
- `logs:DescribeLogStreams`

### Logging Output

If the logging component is configured to write logs to CloudWatch, the following log groups are displayed:

- `/aws/greengrass/GreengrassSystem/GreengrassSystemComponentName`
- `/aws/greengrass/Lambda/aws-region/accountId/lambda-function-name`

Under each log group, you see log streams with the following structure:

```plaintext
date/accountId/greengrass-group-id/name-of-core-that-generated-log
```

The AWS Greengrass core logs look the same whether they are written to local storage or to CloudWatch Logs. The format for AWS Greengrass system logs is:

```
[<timestamp>][<Log level>]<error message>
```

The format for Lambda logs is:

```
[<timestamp>][<Log level>]<error message>
```
AWS Greengrass Logging Limitations

Transactions per Second

When logging to CloudWatch is enabled, the logging component batches log events locally before sending them to CloudWatch, so you can log at a rate higher than five requests per second per log stream.

Memory

If AWS Greengrass is configured to send logs to CloudWatch and a Lambda function logs more than 5 MB/second for a prolonged period of time, the internal processing pipeline eventually fills up. The theoretical worst case is 6 MB per Lambda function.

Clock Skew

When logging to CloudWatch is enabled, the logging component signs requests to CloudWatch using the normal Signature Version 4 process. If the system time on the AWS Greengrass core device is out of sync by more than 15 minutes, then the requests are rejected.

Disk Usage

Use the following formula to calculate the total maximum amount of disk usage for logging.

\[
\text{disk-usage} = \text{greengrass-storage} \times 7 \quad / / \quad 6 \text{ if IP Detector is not used} \\
\quad + \quad 128\text{KB} \quad / / \quad \text{localwatch} \\
\quad \text{internal log} \\
\quad + \quad \text{lambda-storage} \times \text{lambda-count} \quad / / \quad \text{different versions of a lambda are treated as one.}
\]

Where:

- **greengrass-storage**
  - The maximum amount of local storage for AWS Greengrass logs.
- **lambda-storage**
  - The maximum amount of local storage for Lambda logs.
- **lambda-count**
  - The number of deployed Lambda functions.

Log Loss

If your AWS Greengrass core device is configured to log only to CloudWatch and there's no internet connectivity, you have no way to retrieve the logs currently in the memory.

When Lambda functions are terminated (for example, during deployment), a few seconds worth of logs are not written to CloudWatch.
# Troubleshooting AWS Greengrass Applications

## Common Issues

<table>
<thead>
<tr>
<th>Symptom</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>You see 403 Forbidden error on deployment in the logs.</td>
<td>Make sure the AWS Greengrass core's policy in the cloud includes &quot;greengrass:*&quot; as an allowed action.</td>
</tr>
<tr>
<td>Device's shadow does not sync with the cloud.</td>
<td>Check that the AWS Greengrass core has permissions for &quot;iot:UpdateThingShadow&quot; and &quot;iot:GetThingShadow&quot; actions.</td>
</tr>
<tr>
<td></td>
<td><strong>Also see</strong> Troubleshooting Shadow Synchronization Timeout Issues (p. 118).</td>
</tr>
<tr>
<td>The AWS Greengrass core software does not run on Raspberry Pi because user namespace is not enabled.</td>
<td>Run <code>rpi-update</code> to update. Raspbian has released a new kernel 4.9 that has user namespace enabled.</td>
</tr>
<tr>
<td>A ConcurrentDeployment error occurs when you run create-deployment for the first time.</td>
<td>A deployment might be in progress. You can run <code>get-deployment-history</code> to see if a deployment was created. If not, try creating the deployment again.</td>
</tr>
<tr>
<td>The AWS Greengrass core software does not start successfully.</td>
<td>• Check that you are using the binaries appropriate for your architecture.</td>
</tr>
<tr>
<td></td>
<td>• Check that your AWS Greengrass core device has local storage available.</td>
</tr>
<tr>
<td></td>
<td><strong>For more information, see</strong> Troubleshooting with Logs (p. 116).</td>
</tr>
<tr>
<td>The greengrassd script displays: unable to accept TCP connection, accept tcp [::]:8000: accept4: too many open files.</td>
<td>The file descriptor limit for the AWS Greengrass core software has reached the threshold and must be increased.</td>
</tr>
<tr>
<td></td>
<td>Use the following command:</td>
</tr>
<tr>
<td></td>
<td><code>ulimit -n 2048</code></td>
</tr>
<tr>
<td></td>
<td>and restart the AWS Greengrass core software.</td>
</tr>
<tr>
<td></td>
<td><strong>Note</strong> <strong>In this example, the limit is increased to 2048. Choose a value appropriate for your use case.</strong></td>
</tr>
<tr>
<td>Deployment fails with the following error message: Greengrass is not authorized to assume the Service Role associated with this account.</td>
<td>Using the AWS CLI, check that an appropriate service role has been tied to your account using <code>AssociateServiceRoleToAccount</code> and that the account has at least the</td>
</tr>
</tbody>
</table>
Troubleshooting with Logs

GGC v1.3.0

If logs are configured to be stored on the local file system, start looking in the following locations:

- **GREENGRASS_ROOT/ggc/var/log/crash.log**
  Shows messages generated when an AWS Greengrass core crashes.

- **GREENGRASS_ROOT/ggc/var/log/system/runtime.log**
  Shows messages about which component failed.

- **GREENGRASS_ROOT/ggc/var/log/system/**
  This folder contains all the logs from the AWS Greengrass system Lambda functions. Using the messages in ggc/var/log/system/ and ggc/var/log/system/runtime.log, you should be able to find out which error occurred in AWS Greengrass system Lambda functions.

  **Note**
  By default, **GREENGRASS_ROOT** is the /greengrass directory.

If the logs are configured to be stored on the cloud, use CloudWatch Logs to view log messages. The crash log is still found on the AWS Greengrass core device.

If connection errors occur when system Lambda function attempt to connect to AWS IoT, look in the AWS IoT logs in CloudWatch for more information.

  **Note**
  AWS IoT must be configured to write logs to CloudWatch.

GGC v1.1.0

If logs are configured to be stored on the local file system, start looking in the following locations:

- **GREENGRASS_ROOT/ggc/var/log/crash.log**
  Shows messages generated when an AWS Greengrass core crashes.

- **GREENGRASS_ROOT/ggc/var/log/system/runtime.log**
  Shows messages about which component failed.

- **GREENGRASS_ROOT/ggc/var/log/system/**
  This folder contains all the logs from the AWS Greengrass system Lambda functions. Using the messages in ggc/var/log/system/ and ggc/var/log/system/runtime.log, you should be able to find out which error occurred in AWS Greengrass system Lambda functions.

  **Note**
  By default, **GREENGRASS_ROOT** is the /greengrass directory.

If the logs are configured to be stored on the cloud, use CloudWatch Logs to view log messages. The crash log is still found on the AWS Greengrass core device.
If connection errors occur when system Lambda function attempt to connect to AWS IoT, look in the AWS IoT logs in CloudWatch for more information.

Note
AWS IoT must be configured to write logs to CloudWatch.

GGC v1.0.0

If logs are configured to be stored on the local file system, start looking in the following locations:

`GREENGRASS_ROOT/crash.log`

Shows messages generated when an AWS Greengrass core crashes.

`GREENGRASS_ROOT/var/log/system/runtime.log`

Shows messages about which component failed.

`GREENGRASS_ROOT/var/log/system/`

This folder contains all the logs from the AWS Greengrass system Lambda functions. Using the messages in `var/log/system/` and `var/log/system/runtime.log`, you should be able to find out which error occurred in AWS Greengrass system Lambda functions.

Note
By default, `GREENGRASS_ROOT` is the `/greengrass` directory.

If the logs are configured to be stored on the cloud, use CloudWatch Logs to view log messages. The crash log is still found on the AWS Greengrass core device.

If connection errors occur when system Lambda function attempt to connect to AWS IoT, look in the AWS IoT logs in CloudWatch for more information.

Note
AWS IoT must be configured to write logs to CloudWatch.

Troubleshooting Storage Issues

When the local file storage is full, some components might start failing:

- Local shadow updates do not happen.
- New AWS Greengrass core MQTT server certificates cannot be downloaded locally.
- Deployments fail.

You should always be aware of the amount of free space available locally. This can be calculated based on the sizes of deployed Lambda functions, the logging configuration (see Troubleshooting with Logs (p. 116)), and the number of shadows stored locally.

Troubleshooting Messages

All messages sent within AWS Greengrass are sent with QoS 0. If an AWS Greengrass core is restarted, messages that have not been processed yet are lost. For this reason, restart the AWS Greengrass core when the service disruption is the lowest. The AWS Greengrass core is restarted by a deployment, too.
Troubleshooting Shadow Synchronization Timeout Issues

GGC v1.3.0

If there is a significant delay in communication between a Greengrass core device and the cloud, then shadow synchronization may fail due to a timeout. You may see something like this in your log files:

```
```

A possible fix is to configure the amount of time your Greengrass core device waits for a host response. Open the `GREENGRASS_ROOT/config/config.json` file and add a `system.shadowSyncTimeout` field with a timeout value in seconds. For example:

```
{
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    "thingArn": "arn:aws:iot:us-west-2:049039099382:thing/GGTestGroup42_Core",
    "iotHost": "your-AWS-IoT-endpoint",
    "ggHost": "greengrass.iot.us-west-2.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes"
    }
  }
}
```

If no `shadowSyncTimeout` value is specified in the `config.json` file, the default is 1 second.

GGC v1.1.0

If there is a significant delay in communication between a Greengrass core device and the cloud, then shadow synchronization may fail due to a timeout. You may see something like this in your log files:

```
```
A possible fix is to configure the amount of time your Greengrass core device waits for a host response. Open the `GREENGRASS_ROOT/config/config.json` file and add a `system.shadowSyncTimeout` field with a timeout value in seconds. For example:

```json
{
  "coreThing": {
    "caPath": "root-ca.pem",
    "certPath": "cloud.pem.crt",
    "keyPath": "cloud.pem.key",
    "thingArn": "arn:aws:iot:us-west-2:049039099382:thing/GGTestGroup42_Core",
    "iotHost": "your-AWS-IoT-endpoint",
    "ggHost": "greengrass.iot.us-west-2.amazonaws.com",
    "keepAlive": 600
  },
  "runtime": {
    "cgroup": {
      "useSystemd": "yes"
    }
  },
  "system": {
    "shadowSyncTimeout": 10
  }
}
```

If no `shadowSyncTimeout` value is specified in the `config.json` file, the default is 1 second.

GGC v1.0.0

Not supported.