AWS Robomaker on Robotics RB3 Development Kit

These instructions are designed to help you get started with Amazon AWS Robomaker and Qualcomm® Robotics RB3 Development Kit. The robotics platform is based on the Qualcomm ® Snapdragon™ 845 mobile platform from Qualcomm Technologies, so you may see the kit referenced as Qualcomm® SDA845 in some sections. The project will walk you through the following steps:

1. Build and simulate a robot application in AWS cloud
2. Deploy the robot application to the Qualcomm Robotics RB3 development kit through Greengrass
3. Run the deployed robot application on the Qualcomm Robotics RB3 development kit

Here are a few things to keep in mind and test before you start. Please make sure that the Wi-Fi on target (Qualcomm Robotics RB3 Development Kit) can access the AWS website. After deployment, the ROS master and robot application are run inside the Docker. The sensor node/movebase packages/Kobuki packages are run outside the Docker (run on the target). Please launch the movebase/Kobuki packages only after the ROS master is running successfully inside the Docker.

Build and simulate a robot application in AWS cloud

1. **Hello World**
   The "HelloWorld" example is designed to help you understand some basic concepts on ROS and AWS cloud such as S3 bucket and deployment process. You don't need to change any code in this stage. You only need to repeat steps 1 and 2 below to "**Restart the Hello World Simulation Application**"
   a. Create an AWS account.

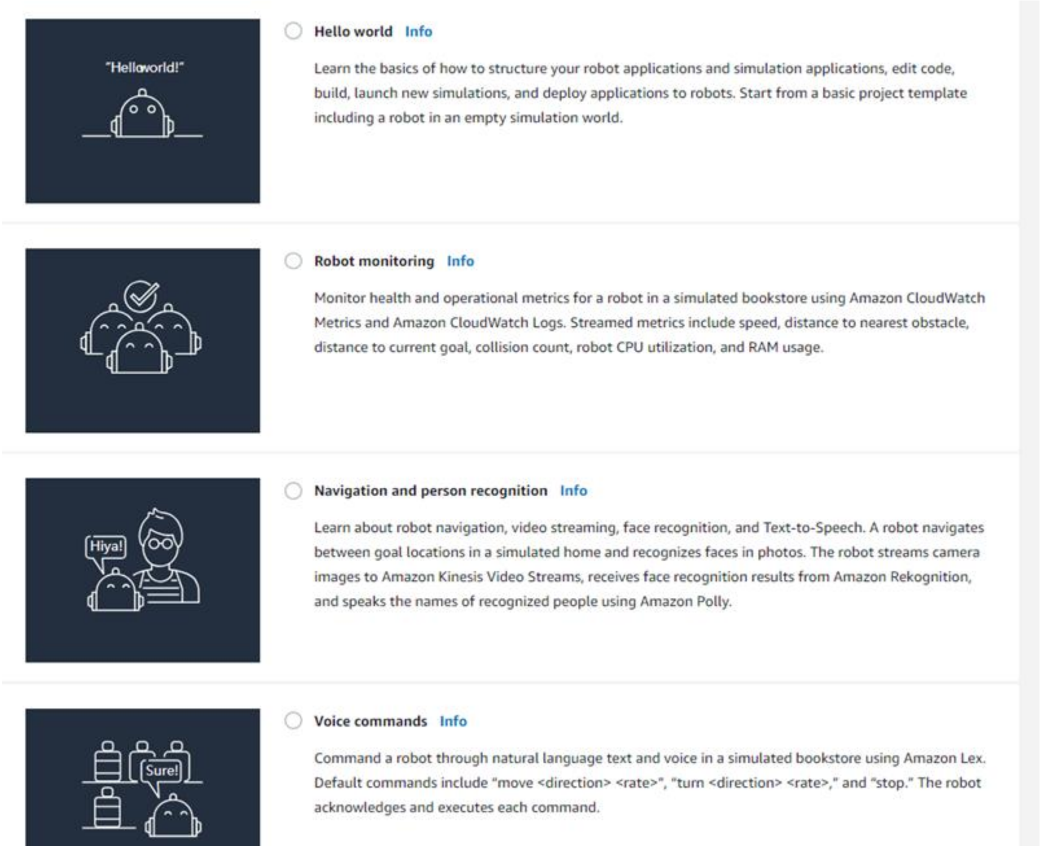
   b. Run example Hello world simulation job.

   c. Create a development environment and a cloud 9 workspace. This will create a virtual PC (VPC) and a workspace on that VPC.

   d. Run HelloWorld app in your workspace.

   e. Deploy the robot APP to target.
   More details for this step can be found in section 2 of the instructions.

2. **Other Examples**
   After the HelloWorld example, we recommend that you try other examples on Robomaker for better understanding and improving your skills. Please keep in mind that each example has a corresponding simulation job.
   Run a new simulation job, and choose from one of the examples as seen below. We recommend "Robot Monitoring" at this stage as it is based on movebase (navigation stack).



○ **Hello world** Info

Learn the basics of how to structure your robot applications and simulation applications, edit code, build, launch new simulations, and deploy applications to robots. Start from a basic project template including a robot in an empty simulation world.

○ **Robot monitoring** Info

Monitor health and operational metrics for a robot in a simulated bookstore using Amazon CloudWatch Metrics and Amazon CloudWatch Logs. Streamed metrics include speed, distance to nearest obstacle, distance to current goal, collision count, robot CPU utilization, and RAM usage.

○ **Navigation and person recognition** Info

Learn about robot navigation, video streaming, face recognition, and Text-to-Speech. A robot navigates between goal locations in a simulated home and recognizes faces in photos. The robot streams camera images to Amazon Kinesis Video Streams, receives face recognition results from Amazon Rekognition, and speaks the names of recognized people using Amazon Polly.

○ **Voice commands** Info

Command a robot through natural language text and voice in a simulated bookstore using Amazon Lex. Default commands include "move <direction> <rate>", "turn <direction> <rate>," and "stop." The robot acknowledges and executes each command.

a. Download the source code of example you choose.

b. Start from the section to "**Modify and Build Applications**" because you have already created an environment while running the HelloWorld example.

c. Download the source code that corresponds to the simulation job you choose.

d. Run this new simulation.

3. **Create your own**

Once you are familiar with the AWS Robomaker examples, it's time to create your own application workspace, simulation job and deploy your own robot application.

a. Create your workspace.

i. You can utilize some code samples from existing examples, for example the movebase demo.

b. Build and bundle your application.

c. Create a simulation job.

Useful Tips:

i. The last section **Create a Simulation Job** can be done from the Robomaker console.

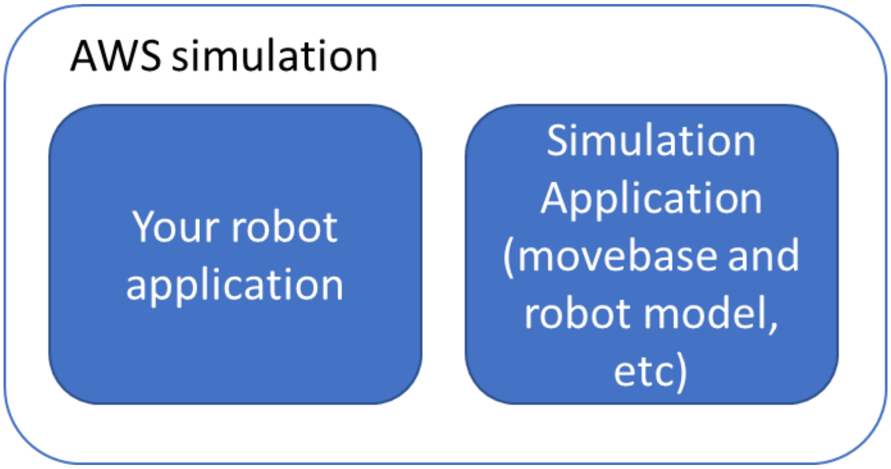
ii. For the rest of the steps, please follow the guild in your cloud 9 command prompt.

iii. After the simulation job is created successfully, you will see it's in running state. In case of failure, you can check the log to troubleshoot.

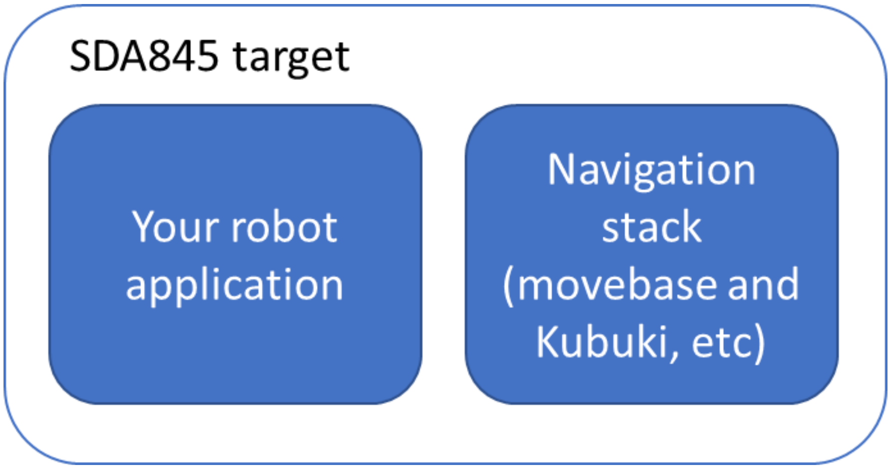
d. You can create a reference workspace based on movebase by the following steps.

i. On the AWS simulation environment, a workspace includes a robot application and a simulation application. For the simulation application, you can utilize the **Robot Monitoring** example. Copy the whole folder of this simulation application to your new simulation app and remove the package **aws_robomaker_simulation_common.**

ii. Saving the below python script as a reference robot application to send the navigation goal to movebase. the folder tree please refer an exist AWS example.



iii. After deployment, the robot application runs on the SDA845 target inside the docker. the movebase runs outside the docker on SDA845. please refer to last section of guide to launch movebase and Kobuki(the real robot).



the below python script is a reference for your robot application.

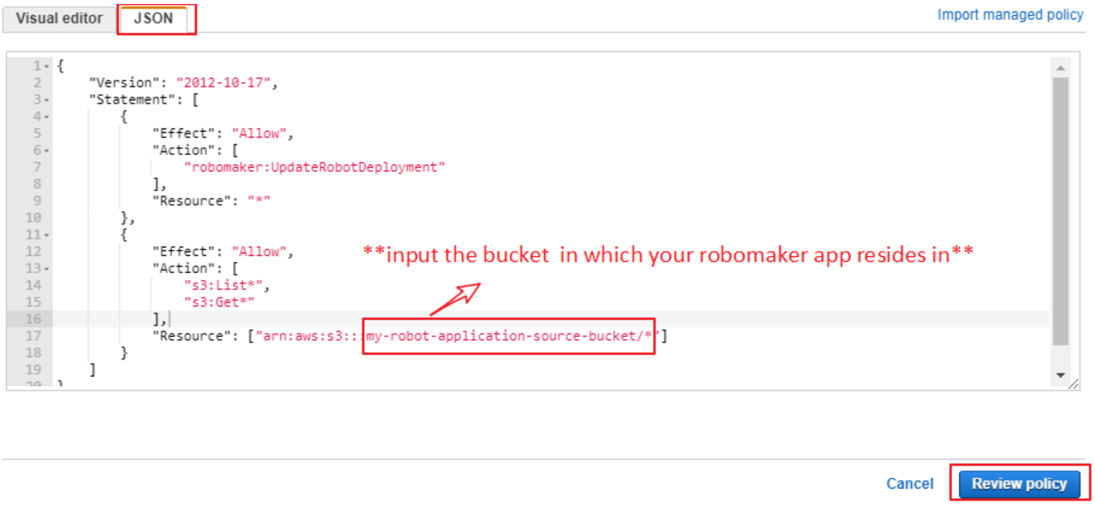
```
#!/usr/bin/env python


import rospy
import actionlib
from actionlib_msgs.msg import *
from geometry_msgs.msg import Pose, Point, Quaternion, Twist
from move_base_msgs.msg import MoveBaseAction, MoveBaseGoal

class MoveBaseTest():
    def __init__(self):
        rospy.init_node('nav_test', anonymous=False)
        rospy.on_shutdown(self.shutdown)

        #p1 = Point(-1.04219532013, 5.23599052429, 0.0)
        p1 = Point(-1.04219532013, 2.23599052429, 0.0)
        q1 = Quaternion(0.0, 0.0, -0.573064998815, 0.819509918874)

        p2 = Point(1.64250051975, 1.58413732052, 0.0)
        q2 = Quaternion(0.0, 0.0, -0.0192202632229, 0.999815273679)

        p3 = Point(5.10259008408, 0.883781552315, 0.0)
        q3 = Quaternion(0.0, 0.0, -0.455630867938, 0.890168811059)

        p4 = Point(6.15312242508, -6.41992664337, 0.0)
        q4 = Quaternion(0.0, 0.0, 0.999290790059, -0.037655237394)

        p5 = Point(1.73421287537, -5.13594055176, 0.0)
        q5 = Quaternion(0.0, 0.0, 0.718415199022, 0.695614549743)

        p6 = Point(-3.83528089523, -5.31936645508, 0.0)
        q6 = Quaternion(0.0, 0.0, 0.701646950739, 0.712524776073)

        quaternions = list()
        quaternions.append(q1)
        quaternions.append(q2)
        quaternions.append(q3)
        #quaternions.append(q4)
        #quaternions.append(q5)
        #quaternions.append(q6)

        points = list()
        points.append(p1)
        points.append(p2)
        points.append(p3)
        #points.append(p4)
        #points.append(p5)
        #points.append(p6)

        goals = list()
        goals.append(Pose(points[0], quaternions[0]))
        goals.append(Pose(points[1], quaternions[1]))
        goals.append(Pose(points[2], quaternions[2]))
        #goals.append(Pose(points[3], quaternions[3]))
        #goals.append(Pose(points[4], quaternions[4]))
        #goals.append(Pose(points[5], quaternions[5]))

        rospy.loginfo("*** started navi test")

        # Publisher to manually control the robot (e.g. to stop it, queue_size=5)
        self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=5)

        # Subscribe to the move_base action server
        self.move_base = actionlib.SimpleActionClient("move_base", MoveBaseAction)
        self.move_base.wait_for_server()
        rospy.loginfo("Connected to move base server")
        rospy.loginfo("Starting navigation test")

        # Initialize a counter to track goals
        i = 0
        while not rospy.is_shutdown():
            # Intialize the waypoint goal
            goal = MoveBaseGoal()
            goal.target_pose.header.frame_id = 'map'
            goal.target_pose.header.stamp = rospy.Time.now()
            goal.target_pose.pose = goals[i%3]

            #move toward the goal
            self.move(goal)
            i += 1


    def move(self, goal):
            # Send the goal pose to the MoveBaseAction server
            self.move_base.send_goal(goal)

            # Allow 1 minute to get there
            finished_within_time = self.move_base.wait_for_result(rospy.Duration(60))

            # If we don't get there in time, abort the goal
            if not finished_within_time:
                self.move_base.cancel_goal()
                rospy.loginfo("Timed out achieving goal")
            else:
                if self.move_base.get_result():
                    rospy.loginfo("Goal done: %s", goal)


    def shutdown(self):
        rospy.loginfo("Stopping the robot...")
        # Cancel any active goals
        self.move_base.cancel_goal()
        rospy.sleep(2)
        # Stop the robot
        self.cmd_vel_pub.publish(Twist())
        rospy.sleep(1)


if __name__ == '__main__':
    try:
        MoveBaseTest()
    except rospy.ROSInterruptException:
        rospy.loginfo("Navigation test finished.")
```

e. Deploy your own application
   Details for deploying your application are described below.

Deploy the robot application to RB3 development kit through AWS IOT Greengrass

Refer to the AWS Greengrass official guide for the latest getting started instructions.

1. **Create IAM policy**
    a. Open IAM page above and select "Policies" ---> "Create policy"
    b. Choose "Greengrass"
    c. Type the policy info in "JSON" tab, copy the JSON code below and modify s3 BUCKET info

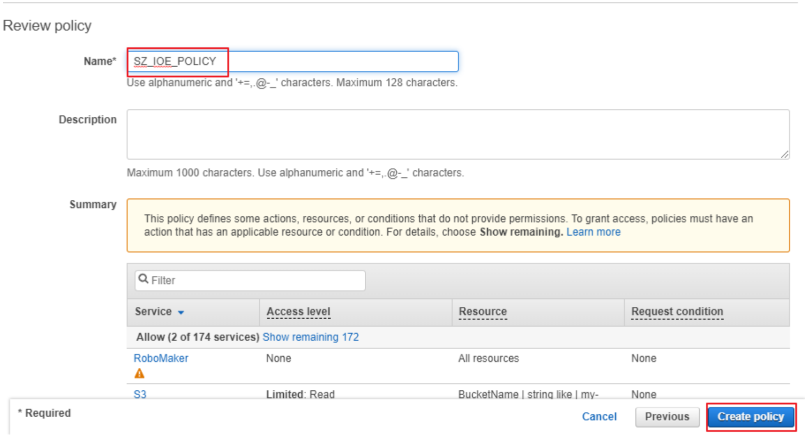


```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "robomaker:UpdateRobotDeployment"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:List*",
                "s3:Get*"
            ],
            "Resource": ["arn:aws:s3:::my-robot-application-source-bucket/*"]
        }
    ]
```

input the bucket in which your robomaker app resides in

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "robomaker:UpdateRobotDeployment"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:List*",
                "s3:Get*"
            ],
            "Resource": ["arn:aws:s3:::my-robot-application-source-bucket/*"]
        }
    ]
}
```

    d. Input your own policy name and then "Create policy"



Review policy

Name* SZ_IOE_POLICY

Use alphanumeric and '+=,.@-_' characters. Maximum 128 characters.

Description

Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.

Summary

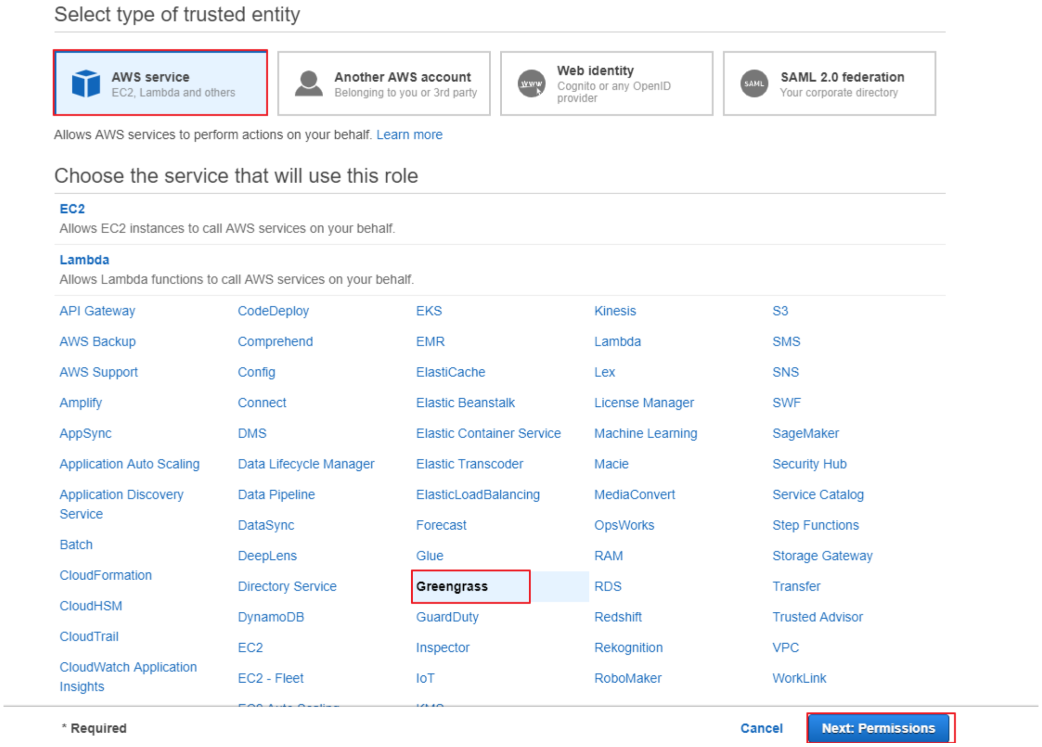
This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition. For details, choose **Show remaining.** Learn more

Service	Access level	Resource	Request condition
Allow (2 of 174 services) Show remaining 172			
RoboMaker ⚠	None	All resources	None
S3	**Limited**: Read	BucketName | string like | my-	None

2. Create IAM role

a. Open IAM page below and select “Role” ---> “Create role”

b. Choose “Greengrass”



Select type of trusted entity

- **AWS service** EC2, Lambda and others
- **Another AWS account** Belonging to you or 3rd party
- **Web identity** Cognito or any OpenID provider
- **SAML 2.0 federation** Your corporate directory

Allows AWS services to perform actions on your behalf. Learn more

Choose the service that will use this role

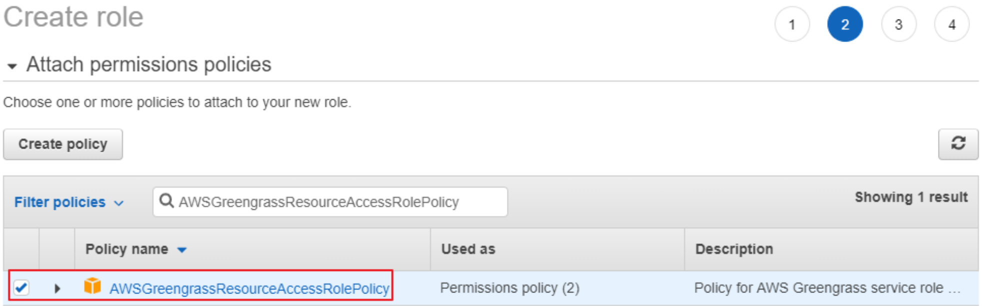
EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

API Gateway
AWS Backup
AWS Support
Amplify
AppSync
Application Auto Scaling
Application Discovery Service
Batch
CloudFormation
CloudHSM
CloudTrail
CloudWatch Application Insights
CodeDeploy
Comprehend
Config
Connect
DMS
Data Lifecycle Manager
Data Pipeline
DataSync
DeepLens
Directory Service
DynamoDB
EC2
EC2 - Fleet
EKS
EMR
ElastiCache
Elastic Beanstalk
Elastic Container Service
Elastic Transcoder
ElasticLoadBalancing
Forecast
Glue
Greengrass
GuardDuty
Inspector
IoT
Kinesis
Lambda
Lex
License Manager
Machine Learning
Macie
MediaConvert
OpsWorks
RAM
RDS
Redshift
Rekognition
RoboMaker
S3
SMS
SNS
SWF
SageMaker
Security Hub
Service Catalog
Step Functions
Storage Gateway
Transfer
Trusted Advisor
VPC
WorkLink

c. Select the policies below and then select “Next”

   i. AWSGreengrassResourceAccessRolePolicy



Create role

Attach permissions policies

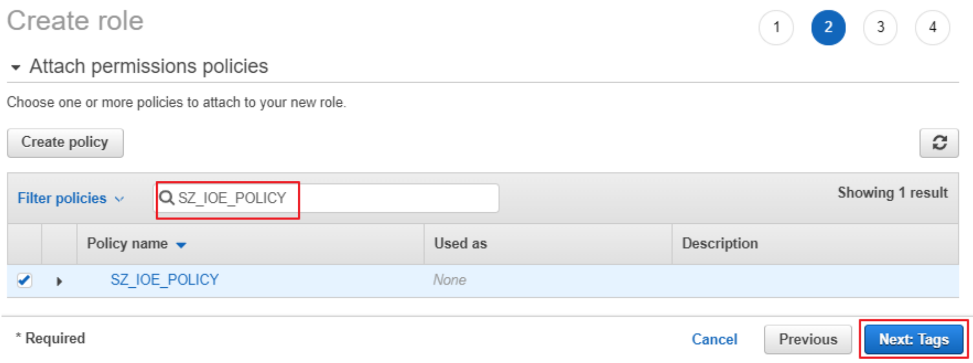
Choose one or more policies to attach to your new role.

AWSGreengrassResourceAccessRolePolicy

Showing 1 result

		Policy name	Used as	Description
☑	▸	AWSGreengrassResourceAccessRolePolicy	Permissions policy (2)	Policy for AWS Greengrass service role ...

   ii. SZ_IOE_POLICY



Create role

Attach permissions policies

Choose one or more policies to attach to your new role.

SZ_IOE_POLICY

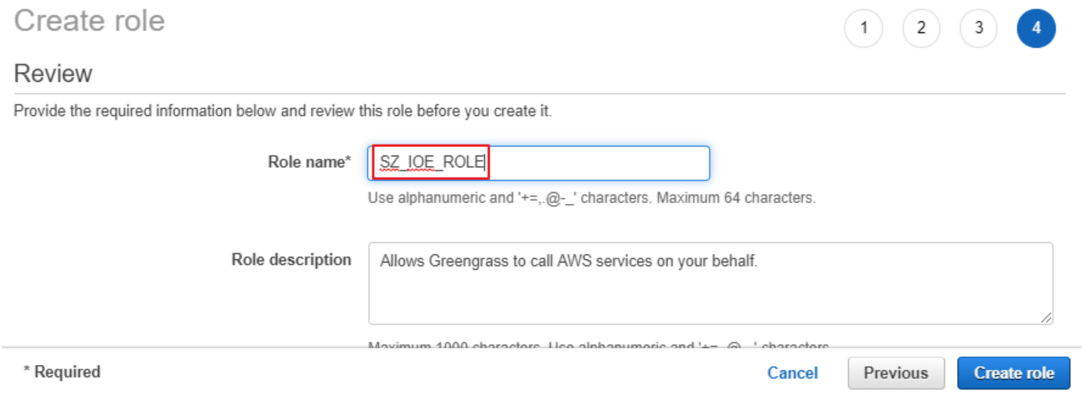
Showing 1 result

		Policy name	Used as	Description
☑	▸	SZ_IOE_POLICY	None	

* Required

d. “Add tags” page is optional, skip it by selecting “Next”

e. Enter your IAM role name and create role.



Create role

Review

Provide the required information below and review this role before you create it.

Role name* SZ_IOE_ROLE

Use alphanumeric and '+=,.@-_' characters. Maximum 64 characters.

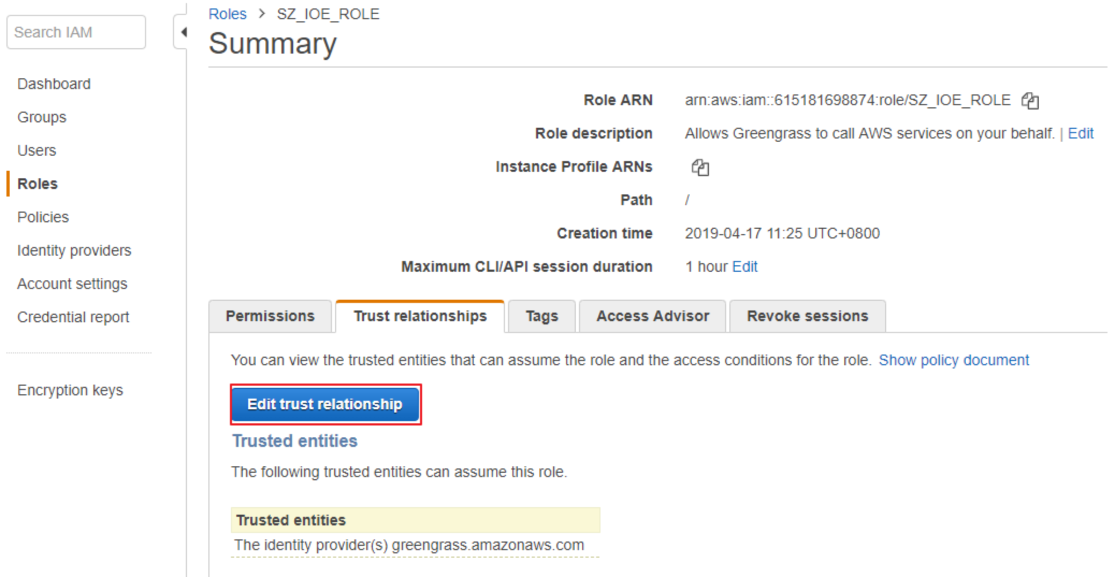
Role description Allows Greengrass to call AWS services on your behalf.

Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.

* Required

f. Edit trust relationship, and copy the JSON settings seen below:

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

      }

    ]

}
```



Summary

Role ARN	arn:aws:iam::615181698874:role/SZ_IOE_ROLE
Role description	Allows Greengrass to call AWS services on your behalf. | Edit
Instance Profile ARNs	
Path	/
Creation time	2019-04-17 11:25 UTC+0800
Maximum CLI/API session duration	1 hour Edit

You can view the trusted entities that can assume the role and the access conditions for the role. Show policy document

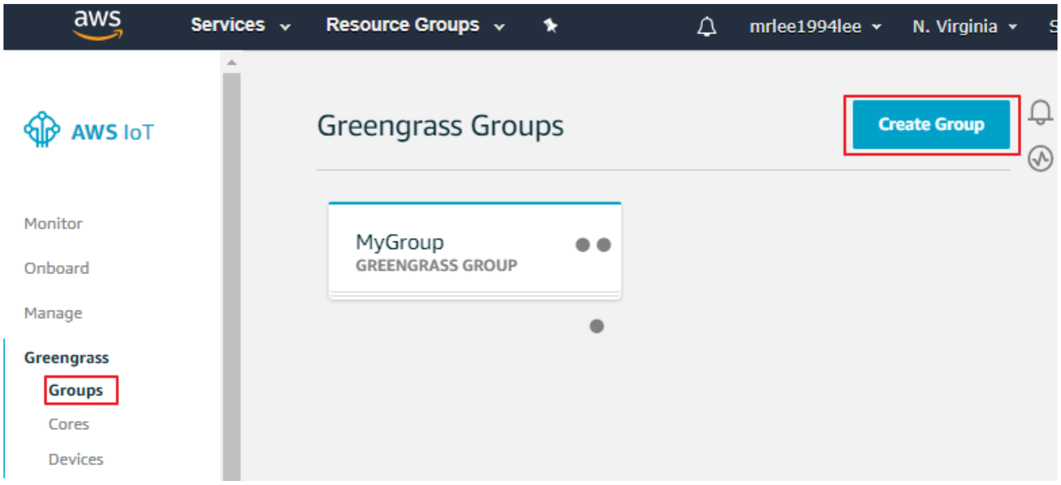
Trusted entities

The following trusted entities can assume this role.

Trusted entities
The identity provider(s) greengrass.amazonaws.com

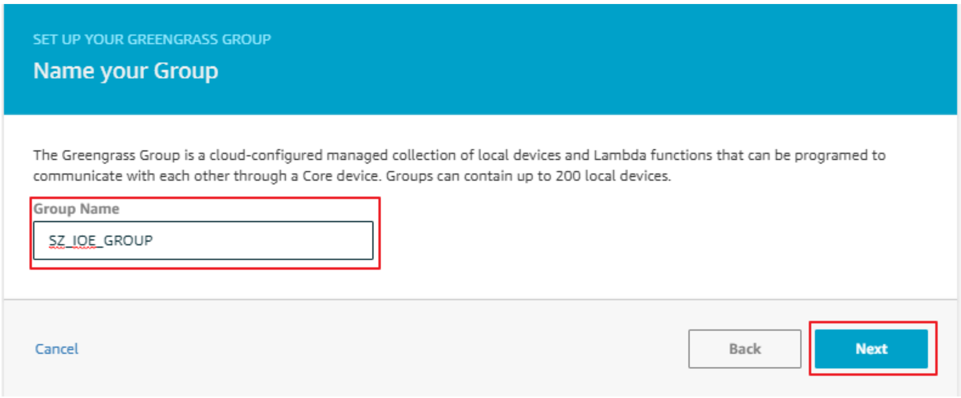
3. Create AWS IoT Greengrass Group

a. Open page below, select "Create Group"



b. Select "Use easy creation"

c. Specify a Group name and then click "Next"

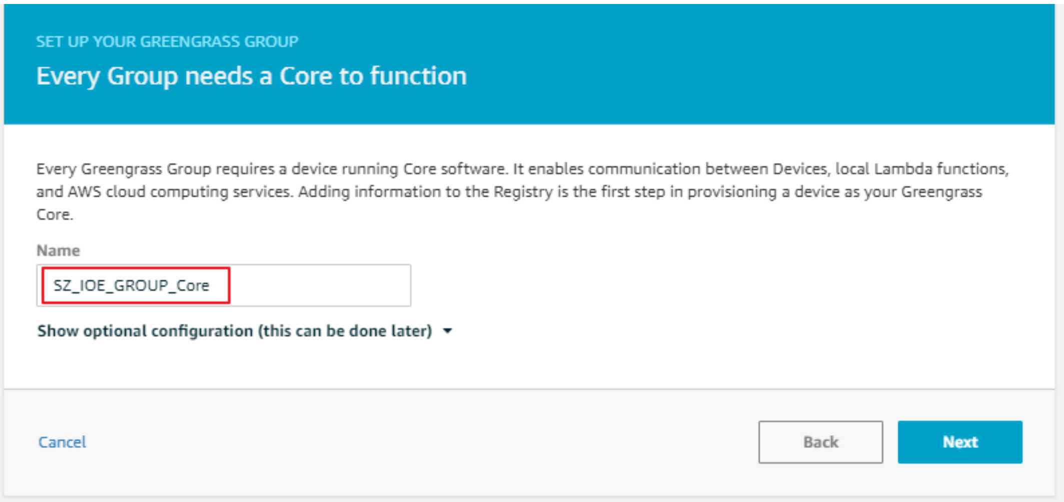

SET UP YOUR GREENGRASS GROUP

Name your Group

The Greengrass Group is a cloud-configured managed collection of local devices and Lambda functions that can be programed to communicate with each other through a Core device. Groups can contain up to 200 local devices.

Group Name: SZ_IOE_GROUP

d. Specify a Greengrass Group name and then click “Next”



SET UP YOUR GREENGRASS GROUP

Every Group needs a Core to function

Every Greengrass Group requires a device running Core 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.

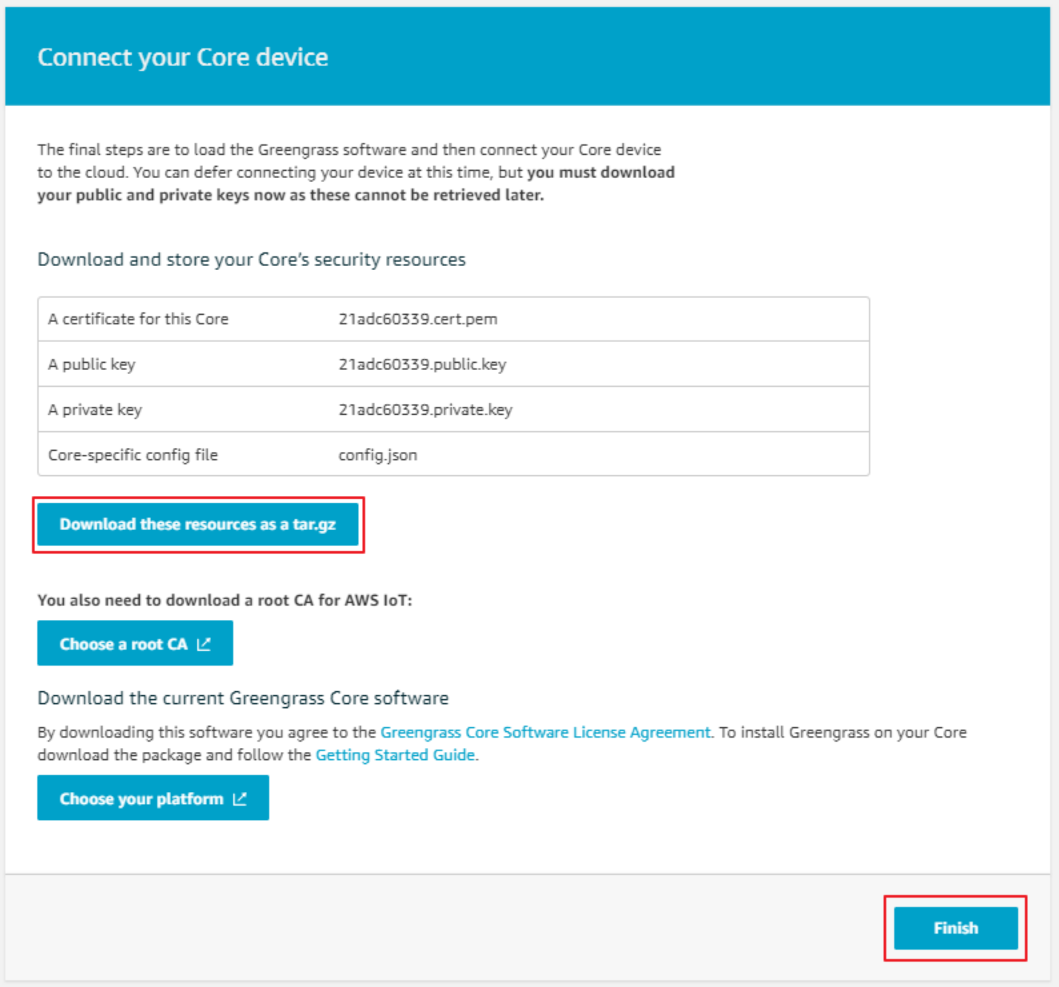
Name: SZ_IOE_GROUP_Core

e. Select “Create Group and Core”

f. Download your security resources as pic shown below, and select “Finish”

*** This is your only chance to download the security resources.

*** Downloaded security keys will be used in the next step.



Connect your Core device

The final steps are to load the Greengrass software and then connect your Core device to the cloud. You can defer connecting your device at this time, but **you must download your public and private keys now as these cannot be retrieved later.**

Download and store your Core’s security resources

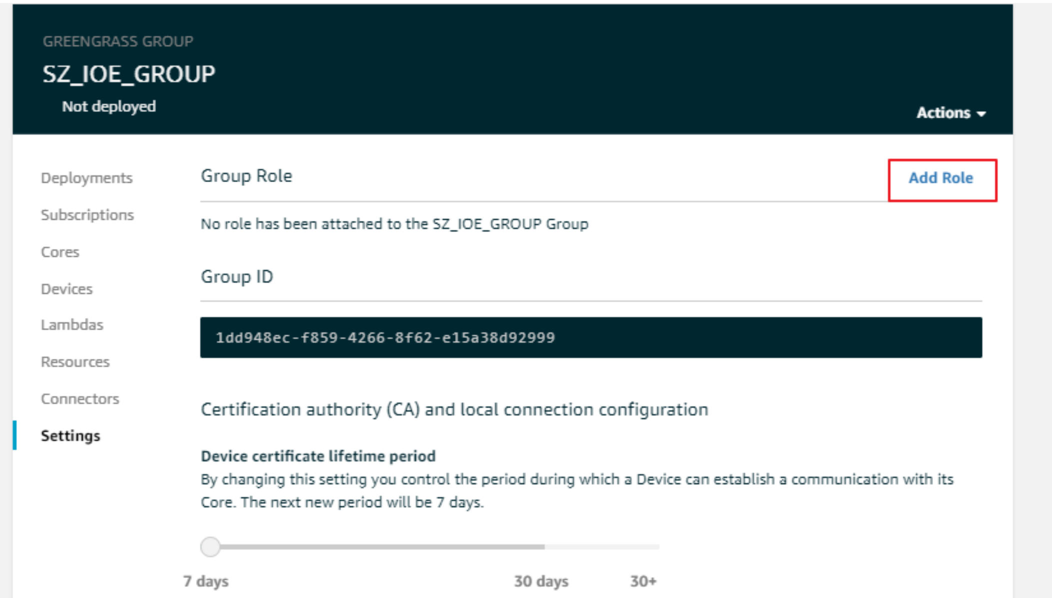
A certificate for this Core	21adc60339.cert.pem
A public key	21adc60339.public.key
A private key	21adc60339.private.key
Core-specific config file	config.json

You also need to download a root CA for AWS IoT:

Download the current Greengrass Core software

By downloading this software you agree to the Greengrass Core Software License Agreement. To install Greengrass on your Core download the package and follow the Getting Started Guide.

g. Attach the IAM role to the Greengrass Group



Group Role

No role has been attached to the SZ_IOE_GROUP Group

Group ID

```
1dd948ec-f859-4266-8f62-e15a38d92999
```

Certification authority (CA) and local connection configuration

Device certificate lifetime period

By changing this setting you control the period during which a Device can establish a communication with its Core. The next new period will be 7 days.

Congratulations! You have successfully created the IAM policy, role and created a Greengrass group for Robomaker. Next, let’s look at how you can run the Greengrass Core on RB3 development kit.

Run GG-Core in RB3 development kit Docker

1. **Prerequisites and launching the docker service**

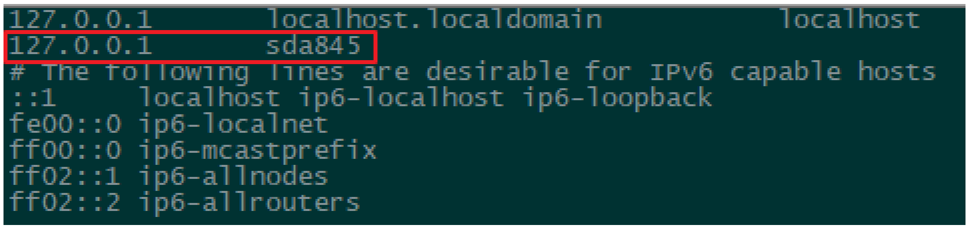
a. Follow the steps below to connect the development kit to the internet.
Use below steps to enable WLAN and dhcp.

```
$ insmod usr/lib/modules/4.9.103/extra/wlan.ko

$ ifconfig wlan0 up

$ wpa_supplicant -iwlan0 -Dnl80211 -c /data/misc/wifi/wpa_supplicant.conf -O
data/misc/wifi/sockets &

$ /usr/sbin/dhcpcd wlan0 -t 0 -o domain_name_servers --noipv4ll -h -b &

$ wpa_cli -iwlan0 -p /data/misc/wifi/sockets

$ add_network

$ set_network 0 ssid "Your SSID"

$ set_network 0 psk "SSID Password"

$ enable_network 0
```

Ping some website to make sure wlan network is up.

b. Run **chronyd**, and make sure system time is correct.

c. Resolve host name “sda845” to “127.0.0.1” by add content below to /etc/hosts



```
127.0.0.1       localhost.localdomain       localhost
127.0.0.1       sda845
# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

d. create a work directory on target

```
$ mkdir -p /greengrass/certs
$ mkdir -p /greengrass/config
```

e. Push files listed below to **/greengrass** directory
   i. arm32v7-ubuntu-18.04-aws-iot-greengrass.tar
   ii. your-security-file.tar.gz

f. Copy the content in page below and save it as **/greengrass/certs/root.ca.pem**
   https://www.amazontrust.com/repository/AmazonRootCA1.pem

g. Decompress the secure file

```
$ tar xzvf your-security-file.tar.gz -C /greengrass
```

h. launch Docker service

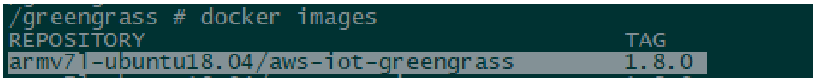
```
$ systemctl start docker
```

Pro Tip: You can check docker with the command “ps -ef | grep docker”

i. load docker image

```
$ docker load -i arm32v7-ubuntu-18.04-aws-iot-greengrass.tar
```

Pro Tip: You can run command “docker images” and you’ll see docker images already installed your system



```
/greengrass # docker images
REPOSITORY                                TAG
armv7l-ubuntu18.04/aws-iot-greengrass     1.8.0
```

j. Environment setup is now done, proceed to run Greengrass Group core on target

```
$ docker run --rm -it --name aws-iot-greengrass --entrypoint /greengrass-entrypoint.sh -v
/greengrass/certs:/greengrass/certs -v  /greengrass/config:/greengrass/config -v
/greengrass/log:/greengrass/ggc/var/log -p 8883:8883 armv7l-ubuntu18.04/test-aws-iot-
greengrass:1.8.0
```

Pro Tip: Press “CTRL+P+Q” keys to detach docker, it’s running in the background now!

k. Check docker status

```
$ docker ps
```

l. Check Greengrass Group core log
   A sample log seen below indicates that your Greengrass Group core successfully connected.

```
$ tail -F /greengrass/log/system/runtime.log



[2019-04-18T04:23:20.122Z][INFO]-Started Deployment Agent and listening for updates

[2019-04-18T04:23:20.122Z][INFO]-Started Deployment Agent and listening for updates

[2019-04-18T04:23:20.122Z][INFO]-MQTT connection connected. Start subscribing: clientId:
SZ_IOE_GROUP_Core

[2019-04-18T04:23:20.122Z][INFO]-Deployment agent connected to cloud

[2019-04-18T04:23:20.123Z][INFO]-Start subscribing 2 topics, clientId: SZ_IOE_GROUP_Core

[2019-04-18T04:23:20.123Z][INFO]-Trying to subscribe to topic $aws/things/SZ_IOE_GROUP_Core-
gda/shadow/update/delta

[2019-04-18T04:23:20.806Z][INFO]-Subscribed to : $aws/things/SZ_IOE_GROUP_Core-
gda/shadow/update/delta

[2019-04-18T04:23:20.806Z][INFO]-Trying to subscribe to topic $aws/things/SZ_IOE_GROUP_Core-
gda/shadow/get/accepted
```

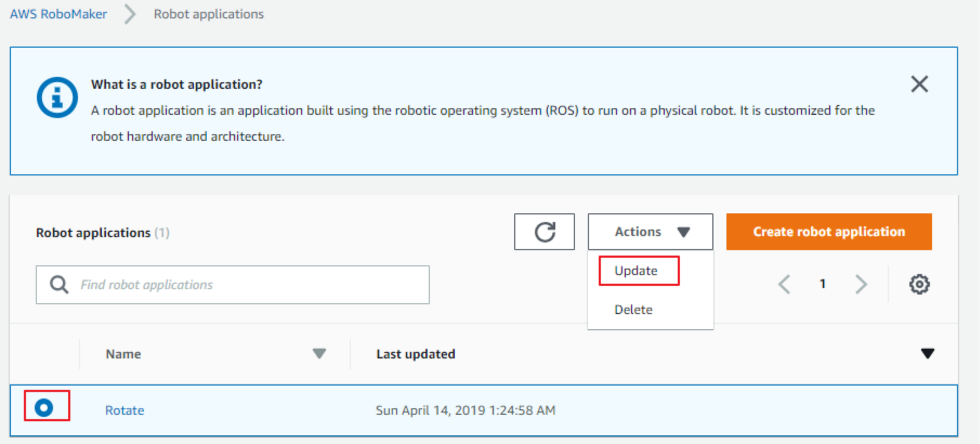
```
[2019-04-18T04:23:21.307Z][INFO]-Subscribed to : $aws/things/SZ_IOE_GROUP_Core-
gda/shadow/get/accepted

[2019-04-18T04:23:21.789Z][INFO]-All topics subscribed, clientId: SZ_IOE_GROUP_Core
```

m. kill container (stop greengrass-core)

```
$ docker kill <ggc container-id>        ## get container id by docker ps
```

Pro Tip: If you do not kill the container now, you will encounter a Greengrass Group core crash issue in the next step.

2. **Create robot application**
Follow the steps above to create your own application. While creating the application, be sure to select the correct AWS region (us-east-1, us-west-2, etc.).
   a. Configure your robot app
      i. Inside “Development” – “Robot applications” page, select your application and click “Actions” button,



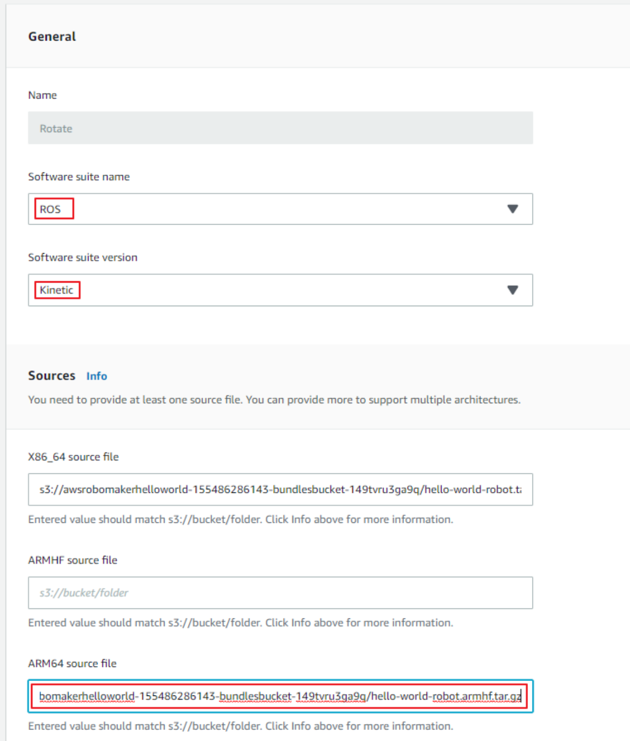
What is a robot application?

A robot application is an application built using the robotic operating system (ROS) to run on a physical robot. It is customized for the robot hardware and architecture.

Robot applications (1)

	Name	Last updated
●	Rotate	Sun April 14, 2019 1:24:58 AM

      ii. Enter your robot-app S3 address to the “ARM64 source file”



General

Name: Rotate

Software suite name: ROS

Software suite version: Kinetic

Sources Info

You need to provide at least one source file. You can provide more to support multiple architectures.

X86_64 source file: s3://awsrobomakerhelloworld-155486286143-bundlesbucket-149tvru3ga9q/hello-world-robot.t

Entered value should match s3://bucket/folder. Click Info above for more information.

ARMHF source file: s3://bucket/folder

Entered value should match s3://bucket/folder. Click Info above for more information.

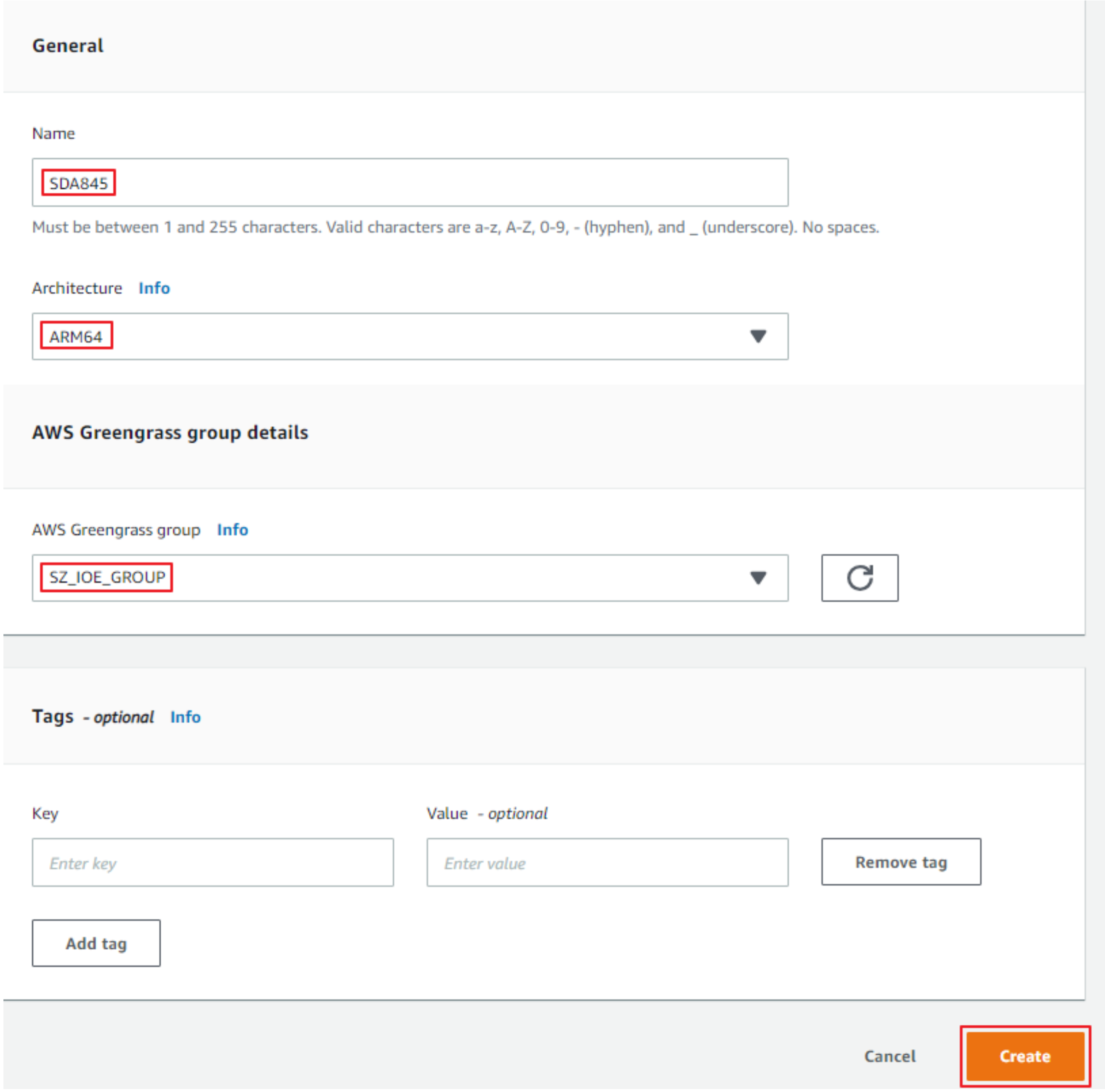
ARM64 source file: bomakerhelloworld-155486286143-bundlesbucket-149tvru3ga9q/hello-world-robot.armhf.tar.gz

Entered value should match s3://bucket/folder. Click Info above for more information.

You can get this info from page https://s3.console.aws.amazon.com/s3/home?region=us-east-1

iii. Inside “Development” – “Robot applicants”, click your app name, and then select “create new version”

b. “Fleet management” – “Robots” – “Create robot”



General

Name

SDA845

Must be between 1 and 255 characters. Valid characters are a-z, A-Z, 0-9, - (hyphen), and _ (underscore). No spaces.

Architecture

ARM64

AWS Greengrass group details

AWS Greengrass group

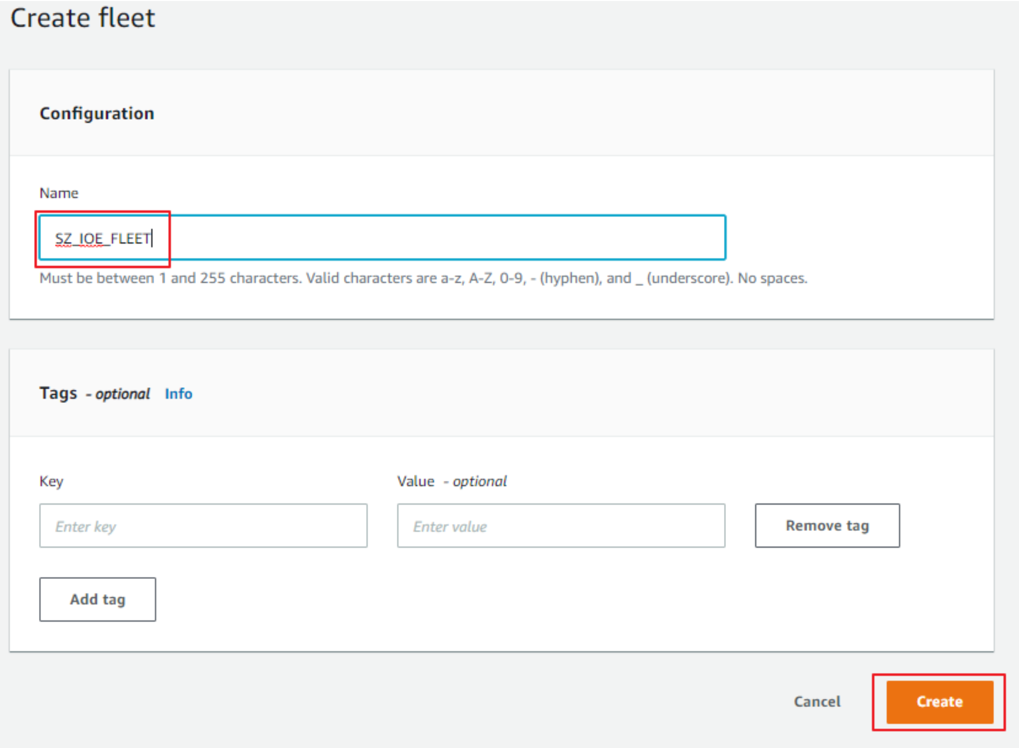
SZ_IOE_GROUP

Tags - *optional*

Key

Value - *optional*

c. “Fleet management” – “Fleets” – “Create fleet”



Create fleet

Configuration

Name

SZ_IOE_FLEET

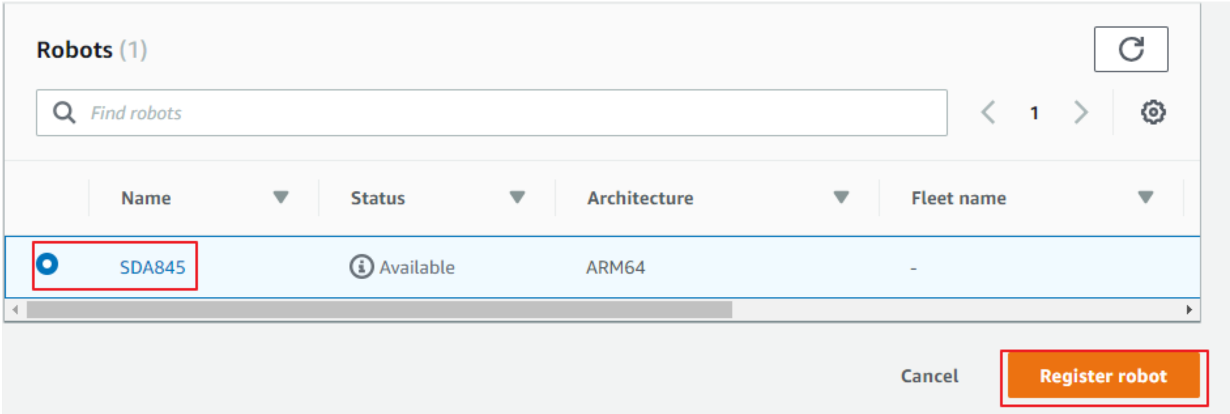
Must be between 1 and 255 characters. Valid characters are a-z, A-Z, 0-9, - (hyphen), and _ (underscore). No spaces.

Tags - *optional* Info

Key: *Enter key*

Value - *optional*: *Enter value*

d. Click your fleet name inside “Fleets” page, then click “Register new” button and register your robot.

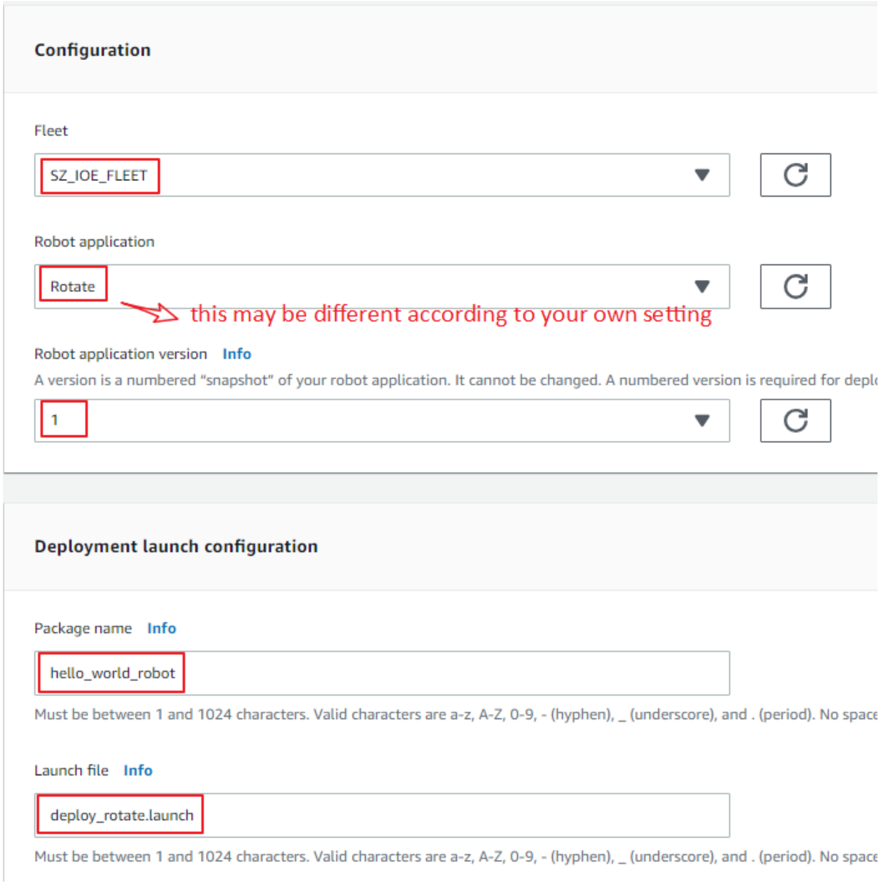


Robots (1)

Find robots

	Name	Status	Architecture	Fleet name
●	SDA845	Available	ARM64	-

e. Inside “Fleet management” – “Deployment” – “Create deployment”. Configure your robot app info, and then click “create”



Configuration

Fleet

SZ_IOE_FLEET

Robot application

Rotate

this may be different according to your own setting

Robot application version Info

A version is a numbered "snapshot" of your robot application. It cannot be changed. A numbered version is required for depl

1

Deployment launch configuration

Package name Info

hello_world_robot

Must be between 1 and 1024 characters. Valid characters are a-z, A-Z, 0-9, - (hyphen), _ (underscore), and . (period). No space

Launch file Info

deploy_rotate.launch

Must be between 1 and 1024 characters. Valid characters are a-z, A-Z, 0-9, - (hyphen), _ (underscore), and . (period). No space

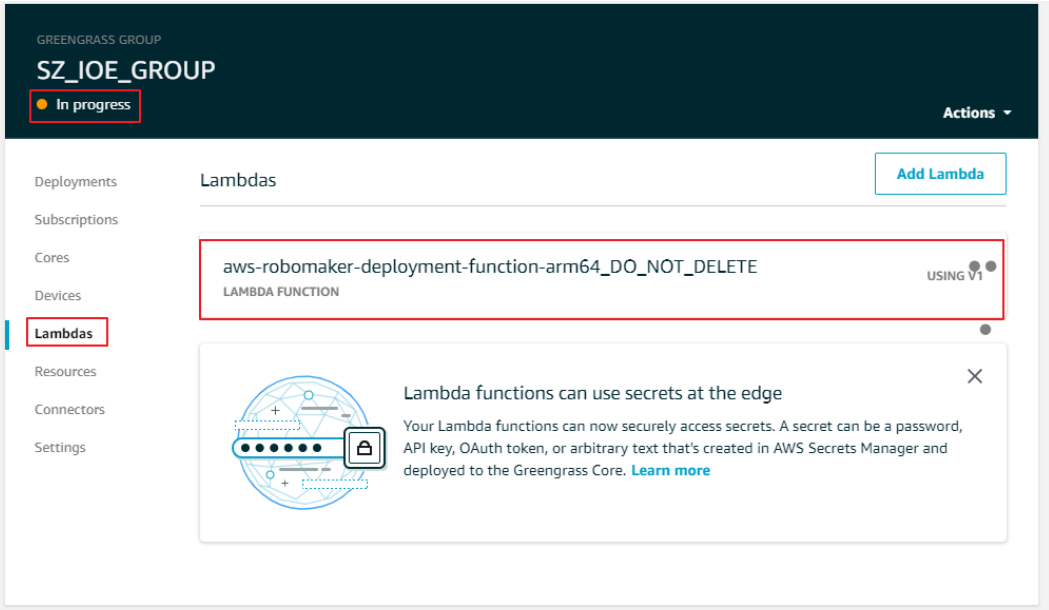
3. **Deploy lambda (robot app) to target**

   Log into the Greengrass console and navigate to the Group hub.

   Here you can see:

   a. A lambda function is added to the robot application that was created.

   b. Group status is "In progress"



GREENGRASS GROUP

SZ_IOE_GROUP

● In progress

aws-robomaker-deployment-function-arm64_DO_NOT_DELETE

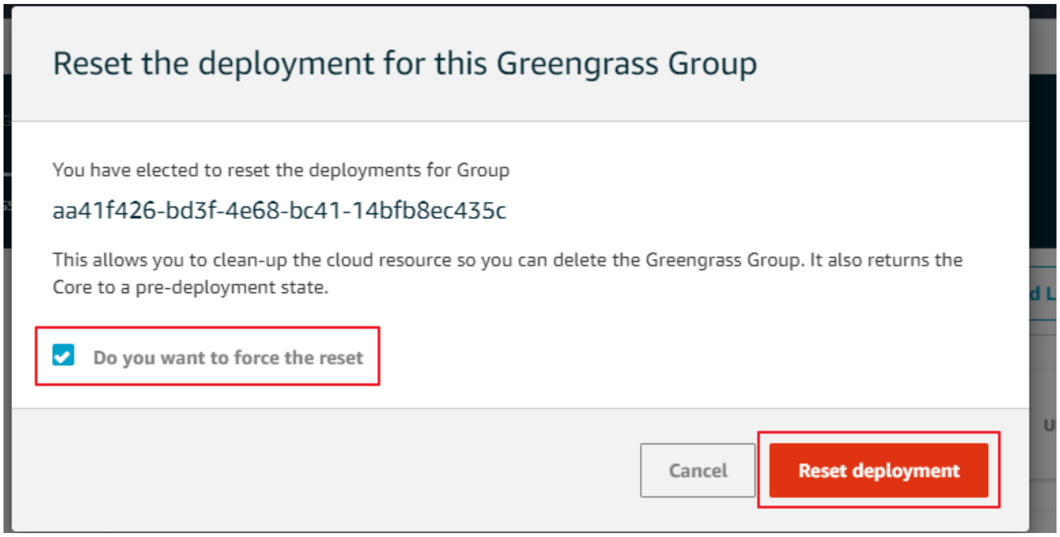
LAMBDA FUNCTION

USING V1

Lambda functions can use secrets at the edge

Your Lambda functions can now securely access secrets. A secret can be a password, API key, OAuth token, or arbitrary text that's created in AWS Secrets Manager and deployed to the Greengrass Core. Learn more

   c. Select "Action" -- "reset deployment" to reset the status because we need some other configuration



Reset the deployment for this Greengrass Group

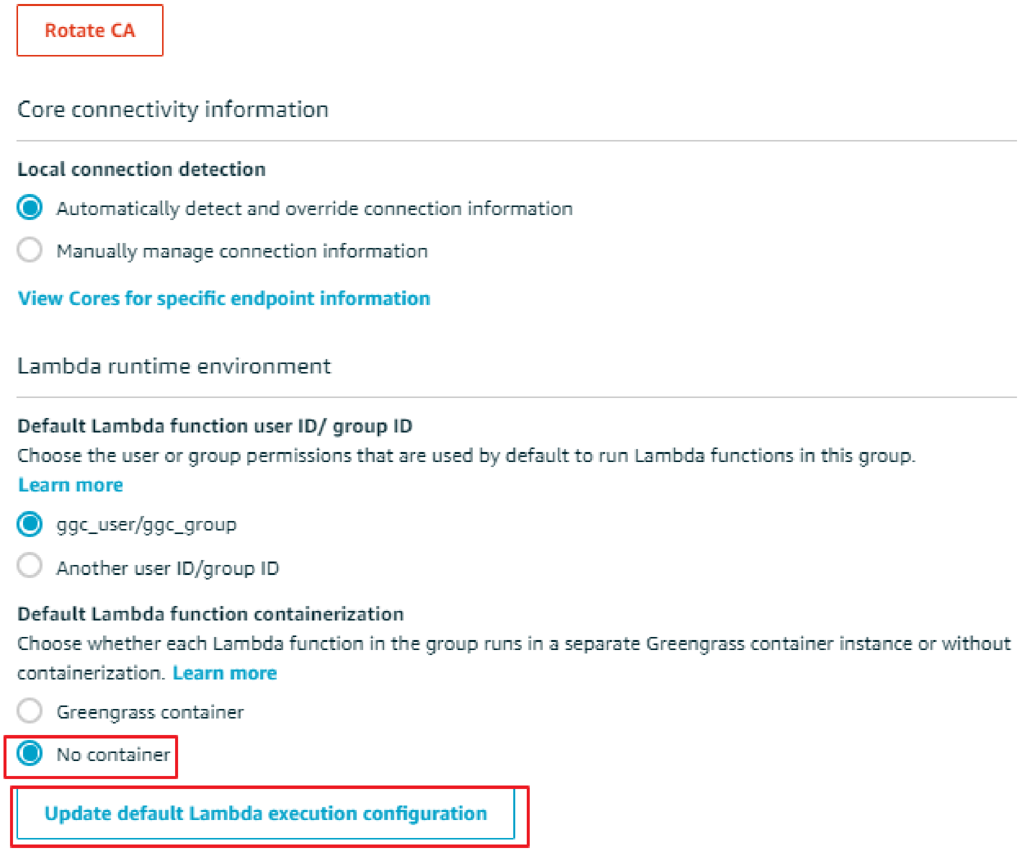
You have elected to reset the deployments for Group

aa41f426-bd3f-4e68-bc41-14bfb8ec435c

This allows you to clean-up the cloud resource so you can delete the Greengrass Group. It also returns the Core to a pre-deployment state.

☑ Do you want to force the reset

d. In “setting” page, set “Lambda function containerization” to **“no container”**
This is an important step before you can deploy the Lambda, or Greengrass Group core will crash



Rotate CA

Core connectivity information

Local connection detection

● Automatically detect and override connection information

○ Manually manage connection information

View Cores for specific endpoint information

Lambda runtime environment

Default Lambda function user ID/ group ID

Choose the user or group permissions that are used by default to run Lambda functions in this group. Learn more

● ggc_user/ggc_group

○ Another user ID/group ID

Default Lambda function containerization

Choose whether each Lambda function in the group runs in a separate Greengrass container instance or without containerization. Learn more

○ Greengrass container

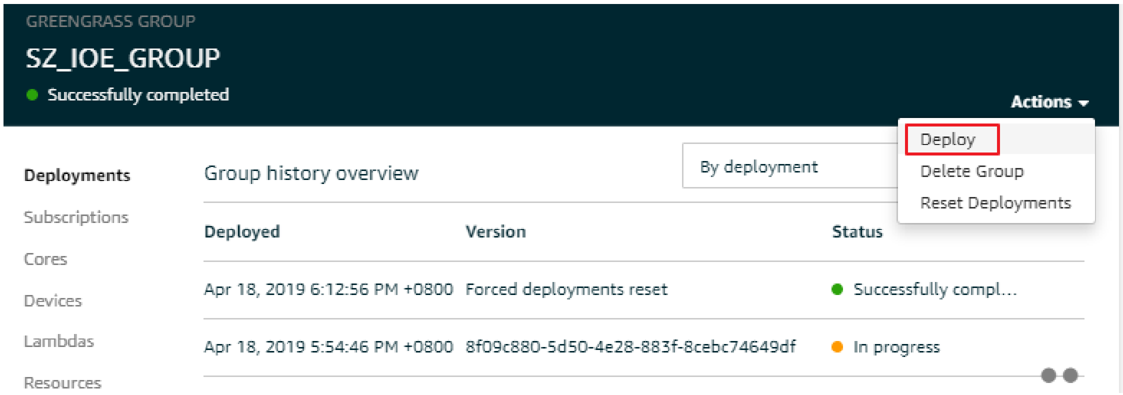
● No container

Update default Lambda execution configuration

e. Run Greengrass Group core on target

```
$ docker run --rm -it --name aws-iot-greengrass --entrypoint /greengrass-entrypoint.sh -v
/greengrass/certs:/greengrass/certs -v  /greengrass/config:/greengrass/config -v
/greengrass/log:/greengrass/ggc/var/log --network host armv7l-ubuntu18.04/test-aws-iot-
greengrass:1.8.0
```

f. Deploy



GREENGRASS GROUP

SZ_IOE_GROUP

● Successfully completed

Group history overview

Deployed	Version	Status
Apr 18, 2019 6:12:56 PM +0800	Forced deployments reset	● Successfully compl...
Apr 18, 2019 5:54:46 PM +0800	8f09c880-5d50-4e28-883f-8cebc74649df	● In progress

Congratulations! You have successfully deployed the robot application to RB3 development kit through AWS IoT Greengrass.

Run the deployed robot application on RB3 development kit

The robot application ROS node would run along with ROS master inside the docker once the deployment is finished. You need to run the Kobuki ROS package or other ROS packages (for example movebase) after ROS master is running. Before you run these packages, you need to setup the devices. Here is a script to help you with easy setup.

```
#! /bin/sh

#hack the kobuki_node minimal.launch first: remap odom to wheel_odom

#hack the /etc/ros_8009.bash: set the ROS_IP, ROS_HOSTNAME and

#ROS_MASTER_URI with IP address directly,  'localhost' doesnot work

source /etc/ros_845.bash

roslaunch /opt/ros/indigo/share/kobuki_node/launch/minimal.launch &

sleep 5

roslaunch /data/pathplan/launch/movebase_845.launch
```

Setup the ROS env:

1. Copy the script to the RB3 kit.
   ```
   adb push launch_movebase.sh /home
   adb shell
   ```
2. Edit the ROS environment to change the IP address
   ```
   vi /opt/ros/indigo/share/ros_env.bash
   ```
3. Set IP address as seen below
   ```
   export ROS_MASTER_URI=http://192.168.1.102:11311
   export ROS_IP=192.168.1.102
   export ROS_HOSTNAME=192.168.1.102
   ```
4. Switch to home directory
   ```
   cd /home
   ```
5. Launch!
   ```
   $ ./launch_movebase.sh
   ```

Congratulations! You are now up and running with Robomaker on the RB3 Development kit.
The “Hello World” example is designed to make the Kobuki base rotate in place. The reference application is designed to make the Kobuki base move. We cannot wait to see how you use these powerful platforms, you can share your projects with us here.