Amazon AWS Greengrass on DragonBoard™ 410c Development Board

This project shows how to get Amazon AWS Greengrass Core up and running on a DragonBoard 410c by Arrow Electronics.

The project is planned to be expanded to include examples that show how to setup a "virtual" sensor which outputs data and is processed using a Lambda function running locally on the DragonBoard 410c. With this first example, only a DragonBoard 410c is required.

The project is planned to be further expanded to include actual sensors so that once you have mastered the concepts around Greengrass, the additional examples will use a 96boards Sensors Mezzanine board with actual sensors attached to the DragonBoard 410c.

This initial example deploys the Hello World example from the AWS Greengrass User Guide.

Before you get started

You will need an AWS account with Greengrass and have downloaded the *AWS Greengrass Core Software* and the *AWS Greengrass Core SDK* from the AWS IoT Console

Initial Greengrass Service Setup

Before you can use the Greengrass service for the first time, you will need to associate a service role so that deployments may work. The steps below require that the AWS CLI is installed and that the Greengrass CLI model has been added.

Dependencies

Install and Setup AWS CLI

Install the AWS CLI command. It is preferable to do this on your host machine where you do your development and have your AWS keys configured. You can also do this on

the DragonBoard 410c itself, but we don't recommend using permanent keys on the Greengrass Core device.

Assuming you have Python pip installed, you should simply be able to do:

```
pip install --upgrade --user awscli
```

It is a good ideal to run the above command on a regular basis to keep the AWS CLI up to date.

AWS GG CLI services

Prior to the public release of AWS Greengrass it was necessary to install the Greengrass CLI commands manually. The above `pip install` command with the `--upgrade` option will make sure to install the latest version of the AWS CLI with the greengrass CLI commands.

If you had installed the preview Greengrass CLI command models manually, they can be removed from the `~/.aws/models` path.

NOTE: The command snippets below are meant for the reader to be able to cut and paste into their console and build upon each other. These commands will save the output of the AWS cli commands and parse them to extract specific values into environment variables that will be required by follow-on commands. The JSON response files are stored in the same path that the commands are executed and are prefixed with `tmp-`.

Other tools

The examples below will use jq (https://stedolan.github.io/jq/) to process the output from the AWS CLI. Follow the instructions on their page to download and install `jq` for your platform.

Verify your installation works with a simple example:

```
echo '{"param":"test successful"}' | jq ".param"
```

It should display "test successful".

Check for existing service role.

You can check if a service role has already been associated with the following command:

```
aws greengrass get-service-role-for-account
```

This should return a structure that looks like this:

```
{
    "AssociatedAt": "<time stamp>",
    "RoleArn": "<role arn>"
}
```

If it does not have a service role attached then create and attach one using the following steps.

Create and attach a Greengrass Service Role

Using the AWS CLI you first create an IAM role for the Greengrass service:

```
aws iam create-role --role-name Greengrass-Service-Role \
--description "Allows AWS Greengrass to call AWS Services on your behalf" \
--assume-role-policy-document \
'{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "Service": "greengrass.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
    ]
}' > tmp-iam-role.json
IAMROLENAME=`jq -r ".Role.RoleName" tmp-iam-role.json`
IAMROLEARN=`jq -r ".Role.Arn" tmp-iam-role.json`
```

Please note that we will be using the *IAMROLENAME* and *IAMROLEARN* environment variables in the following AWS CLI commands.

Next you attach AWS managed policies for the services that you will require. At a minimum you need **AWSGreengrassResourceAccessRolePolicy** for GG deployments to work. We will also be including **AWSLambdaReadOnlyAccess** and **AWSIoTFullAccess** managed policies.

```
aws iam attach-role-policy --policy-arn arn:aws:iam::aws:policy/service-
role/AWSGreengrassResourceAccessRolePolicy --role-name $IAMROLENAME
```

```
aws iam attach-role-policy --policy-arn
arn:aws:iam::aws:policy/AWSLambdaReadOnlyAccess --role-name $IAMROLENAME
aws iam attach-role-policy --policy-arn
arn:aws:iam::aws:policy/AWSIoTFullAccess --role-name $IAMROLENAME
```

Add any additional custom policies. Here we are adding the ability to execute any Greengrass command.

```
aws iam put-role-policy --role-name $IAMROLENAME --policy-name $IAMROLENAME-
Policy --policy-document \
'{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1491600492000",
            "Effect": "Allow",
            "Action": [
                "greengrass:*"
                ],
            "Resource": [
                "*" ]
        } ]
}'
```

Finally attach this service role to the Greengrass service:

```
aws greengrass attach-service-role-to-account --role-arn $IAMROLEARN
```

The next section will walk you through the set up of the Greengrass Core on your DragonBoard 410c.

Greengrass Core

Now that your account is ready to use the Greengrass service, the next step is to install and deploy a Greengrass application. Here you will find details for setting up and running Greengrass core on your DragonBoard 410c device. This information has been distilled from Amazon's Greengrass Developer and User Guides. Please refer to them for more details.

Installation/Setup

This section will walk you through the steps required to install and run Greengrass Core. First you will define your Greengrass Core (GGC) device in the cloud and then you will

install and configure the GGC on the DragonBoard 410c. These steps are to be repeated for every device running GGC.

Defining the Greengrass Core Device in the Cloud

Here you will create:

- AWS IoT Thing: This thing will represent your GGC
- Certificate and Keys: These are used when the GGC connects to the cloud and is how the cloud identifies/authenticates the GGC.
- Policy: This defines the permissions/authorization for an AWS IoT Thing

Create a Thing for the GGC

"Things" are identified by their `thingName` or sometimes by their `thingArn`. These scripts save those values to environment variable for use in later steps.

```
GGC_DEPLOYMENT="HelloWorld-GGC"
aws iot create-thing --thing-name "$GGC_DEPLOYMENT" > tmp-ggc-thing.json
THING_GGC=`jq -r ".thingArn" tmp-ggc-thing.json`
THINGNAME_GGC=`jq -r ".thingName" tmp-ggc-thing.json`
```

You can view the output of the `create-thing` command with:

```
cat tmp-ggc-thing.json
```

Create a Certificate to identify the GGC

When something connects to the AWS IoT Cloud or to a GGC, it is identified by the Certificate used to establish the TLS connection. Here you will create the certificate. A few steps later you will attach the certificate to the "Thing" from the prior step.
The `create-keys-and-certificate` command generates a private key and certificate in pem format. It is important to save these as this is the only time the key is presented and you will not be able to retrieve it. Make sure to keep the files created by these commands in a safe place until you can deploy them on your GGC.

```
aws iot create-keys-and-certificate --set-as-active > tmp-ggc-cert.json
CERTARN_GGC=`jq -r ".certificateArn" tmp-ggc-cert.json`
jq -r ".certificatePem" tmp-ggc-cert.json > cloud.pem.crt
jq -r ".keyPair.PrivateKey" tmp-ggc-cert.json > private.pem.key
jq -r ".keyPair.PublicKey" tmp-ggc-cert.json > public.pem.key
```

The above creates the files `cloud.pem.crt`, `private.pem.key`, and `public.pem.key`, which hold the Certificate, Private Key and Public Key for the GGC device. These will need to be transferred to the DragonBoard 410c running GGC.

Create a Policy with the permissions for the GGC

Policies are used to give permissions to things. The `create-policy` command is used to create the policy with the permissions that will be assigned to a *thing* via its *certificate*. The following policy is very permissive as it allows all `iot` and `greengrass` commands by any resource that is assigned this policy. This policy will be re-used for other *things* in these samples. ***NOTE*** This is not a best practice.

```
aws iot create-policy --policy-name "$GGC_DEPLOYMENT-IOT-Policy" --policy-
document \
'{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "iot:*",
            "greengrass:*"
        ],
        "Resource": "*"
    }
    ]
}' > tmp-iot-policy.json
POLICYNAME_IOT=`jq -r ".policyName" tmp-iot-policy.json`
```

In the next couple of steps you will attach the “Policy” to the “Certificate” and then the “Certificate” to the “Thing”.

Attach Policy to the GGC Certificate

```
aws iot attach-principal-policy --policy-name $POLICYNAME_IOT --principal
$CERTARN_GGC
```

Attach the GGC Certificate to the GGC Thing

```
aws iot attach-thing-principal --thing-name $THINGNAME_GGC --principal
$CERTARN_GGC
```

Now you that the “Thing” for your DragonBoard 410c is setup in the cloud, you will proceed to setup and configure the DragonBoard 410c with the Greengrass Core software.

Setting up the DragonBoard 410c

OS Installation

This example uses the default Debian distribution for the DragonBoard 410c found at the DragonBoard 410c download page. As of this writing that version is 17.04.1. There is a known issue with the prior 17.04 version that caused some DragonBoard410c's to reboot on a regular basis. Alternatively, the 16.09 release also works with Greengrass Core.

The commands in the following steps are to be executed on the DragonBoard 410c itself. Open a terminal session if you are using a keyboard, mouse and monitor on the DragonBoard 410c. Get the IP address of the DragonBoard 410c using the `/sbin/ifconfig` command as it will be useful to copy files using `scp` and also to open `ssh` connections. The default user/password on the Debian image is `linaro/linaro`.

Enable symlink and hardlink protection

Greengrass Core does system checks for hard/soft link protection. You can choose to run GGC in non-secure mode but it is not recommended. The Debian release for the DragonBoard 410c has symlink and hardlink protection disabled by default. The following creates a `sysctl` configuration file with the options to enable the protection.

```
sudo sh -c 'echo "fs.protected_hardlinks = 1" >>/etc/sysctl.d/local-ggc.conf'
sudo sh -c 'echo "fs.protected_symlinks = 1" >>/etc/sysctl.d/local-ggc.conf'
```

After rebooting your DragonBoard 410c, you can verify the protection is working by checking:

```
sudo cat /proc/sys/fs/protected_{hardlinks,symlinks}
```

You should see two 1s.

Create Greengrass User and Group accounts

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

Install Greengrass Core

Copy the Greengrass Core software release tar file that was fetched from the software download page of the AWS IoT console to the DragonBoard 410c using `scp` or a USB thumb drive. Once on the DragonBoard 410c, unpack the tar file.

```
sudo tar -zxvf greengrass-linux-aarch64-1.0.0.tar.gz -C /
```

Install GGC Certificates

Copy the *`cloud.pem.crt`* and *`private.pem.key`* files created in the above "Create a Certificate to identify the GGC" section to the */greengrass/configuration/certs* path on the DragonBoard 410c.

Get the root CA certificate from Verisign and move it to the same path. Use the following command from the DragonBoard 410c:

```
wget
https://www.symantec.com/content/en/us/enterprise/verisign/roots/VeriSign-
Class%203-Public-Primary-Certification-Authority-G5.pem
sudo mv VeriSign-Class\ 3-Public-Primary-Certification-Authority-G5.pem
/greengrass/configuration/certs/root-ca.pem
```

Configure GGC

```
IOTHOST=`aws iot describe-endpoint | jq -r ".endpointAddress"`
echo "{
    \"coreThing\": {
        \"caPath\": \"root-ca.pem\",
        \"certPath\": \"cloud.pem.crt\",
        \"keyPath\": \"private.pem.key\",
        \"thingArn\": \"$THING_GGC\",
        \"iotHost\": \"$IOTHOST\",
        \"ggHost\": \"greengrass.iot.us-west-2.amazonaws.com\",
        \"keepAlive\": 600
    },
    \"runtime\": {
        \"cgroup\": {
            \"useSystemd\": \"yes\"
        }
    }
}" > config.json
sudo mv config.json /greengrass/configuration/.
```

Start GGC

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

You now have the Greengrass Core software up and running on your DragonBoard 410c. It is connected to AWS and waiting for deployment instructions. The following section will walk you through creating a deployment that is automatically pushed to your DragonBoard 410c.

Creating a Greengrass Deployment

This section will walk through the steps to deploy the HelloWorld example included with the Greengrass Distribution. The AWS Greengrass User Guide provides instructions for deploying the example using the AWS IoT Console. The following will show you how to do so entirely from the command line using the AWS CLI with the help of jq to parse the results.

A Greengrass deployment generally requires the following steps:

- Create an AWS Greengrass Group
- Create a Core Definition
- Create a Device Definition
- Create a Lambda Function Definition
- Create a Subscription Definition
- Create a Logger Definition
- Update the Group Version
- Create a Deployment
- Update the Connectivity Information Service

The commands below are AWS CLI command and should be executed in the same environment that was used in the earlier sections to create the GGC Thing in the cloud. They will rely on environment variables that were defined while executing those commands. If this is a new shell session they can be recreated from the saved json response files with the following commands:

```
GGC_DEPLOYMENT="HelloWorld-GGC"
CERTARN_GGC=`jq -r ".certificateArn" tmp-ggc-cert.json`
THING_GGC=`jq -r ".thingArn" tmp-ggc-thing.json`
```

Create an AWS Greengrass Group

A Group is the object that encompasses all the parts and pieces that make up a Greengrass deployment. This step creates the Group in the cloud. In a later step the Group will be updated with the items that are to be a part of this deployment.

```
aws greengrass create-group --name "$GGC_DEPLOYMENT-Group1" > tmp-ggc-
group.json
GGC_GROUPID=`jq -r ".Id" tmp-ggc-group.json`
GGC_GROUPARN=`jq -r ".Arn" tmp-ggc-group.json`
GGC_GROUPNAME=`jq -r ".Name" tmp-ggc-group.json`
```

The following “Definitions” are the pieces and parts that will be assigned to the group.

Create a Core Definition

The “Core Definition” will tell the Group about the Greengrass Cores that will be part of the deployment. First you create the core definition by giving it a name, and then you create a version in which you include the Arn and Certificate that identify the Greengrass cores. In this example, we only have 1 core and the Arn and Certificate are the ones created earlier in the “Greengrass Core” section.

```
aws greengrass create-core-definition --name "$GGC_DEPLOYMENT-Core" > tmp-
ggc-core.json
GGC_COREID=`jq -r ".Id" tmp-ggc-core.json`
cat << EOF > tmp-config-cores.json
[{
    "CertificateArn": "$CERTARN_GGC",
    "Id": "$GGC_DEPLOYMENT-1",
    "SyncShadow": true,
    "ThingArn": "$THING_GGC"
}]
EOF
aws greengrass create-core-definition-version --core-definition-id
"$GGC_COREID" \
    --cores file://tmp-config-cores.json  > tmp-ggc-coreversion.json
GGC_COREVERSION=`jq -r ".Arn" tmp-ggc-coreversion.json`
```

Create a Device Definition

The “Device Definition” describes all the IoT Devices that can connect to this Greengrass deployment. Typically, these devices are referred to as “Greengrass Aware Devices”. They would normally connect to the cloud first to “discover” the connection information for the Greengrass Core that they would ultimately connect to.

The AWS Greengrass HelloWorld example does not have any IoT devices.

Create a Lambda Function Definition

The “Lambda Function Definition” describes the Lambda functions that will be part of this Greengrass deployment.

The AWS Greengrass HelloWorld example has a single Lambda function that runs indefinitely and posts “Hello World” messages to an MQTT topic.

The lambda function must be created first to create the lambda function definition.

Create Lambda Function

First we will check that there is an IAM Role available that is valid for assigning to a Lambda function. The role itself does not matter as the GGC does not use it, but it is required for creating the lambda function.

The following command will query your IAM Roles for one that can be used with the Lambda service. It will print a warning message if one does not exist.

```
aws iam list-roles > tmp-iam-roles.json
GGC_ROLELAMBDA=`jq '.Roles[] | select (..|.Service? | contains("lambda")?) |
.Arn' tmp-iam-roles.json | head -1`
[ -z "$GGC_ROLELAMBDA" ] && echo "STOP: you must create a role that can be
used by lambda functions."
```

Verify the environment variable GGC_ROLELAMBDA has an Arn for a role. If it does not, you will need to create one with the following command:

```
aws iam create-role --role-name Basic-Lambda-Service-Role \
--assume-role-policy-document \
'{
    "Version": "2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "Service": "lambda.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
    }
    ]
}' > tmp-iam-lambdarole.json
GGC_ROLELAMBDA=`jq -r ".Role.Arn" tmp-iam-lambdarole.json`
```

The next command will create the Lambda function by uploading the greengrassHelloWorld.zip file containing the code. The file can be found in the “AWS Greengrass Core SDK” distribution once is it unpacked at the following location:

`aws_greengrass_core_sdk/examples/HelloWorld/greengrassHelloWorld.zip`.

The command below assumes the `greengrassHelloWorld.zip` is in the path that the command is being run from. Adjust the `–zip-file` parameter as needed or copy the file to your path.

```
aws lambda create-function \
--region us-west-2 \
--function-name HelloWorld \
--zip-file fileb://greengrassHelloWorld.zip \
--role $GGC_ROLELAMBDA \
--handler greengrassHelloWorld.lambda_handler \
--runtime python2.7 > tmp-ggc-lambdaHW.json
GGC_LAMBDA_HWARN=`jq -r ".FunctionArn" tmp-ggc-lambdaHW.json`
```

Now you need to publish a version of this lambda function and create an alias for it. The advantage of using function aliases is that you can update the lambda function and point the alias to the new version and not have to update the Greengrass function definition.

```
aws lambda publish-version --function-name $GGC_LAMBDA_HWARN > tmp-ggc-
lambdaHW-version.json
GGC_LAMBDA_HWVERSNAME=`jq -r ".FunctionName" tmp-ggc-lambdaHW-version.json`
GGC_LAMBDA_HWVERSION=`jq -r ".Version" tmp-ggc-lambdaHW-version.json`
aws lambda create-alias --function-name $GGC_LAMBDA_HWVERSNAME --name
storyLineMessage \
--function-version $GGC_LAMBDA_HWVERSION > tmp-ggc-lambdaHW-alias.json
GGC_LAMBDA_HWALIASARN=`jq -r ".AliasArn" tmp-ggc-lambdaHW-alias.json`
```

Now we can create the function definition that tells the Greengrass deployment which Lambda functions to include.

```
aws greengrass create-function-definition \
--name "$GGC_DEPLOYMENT-Functions" > tmp-ggc-functiondef.json
GGC_FUNCDEFID=`jq -r ".Id" tmp-ggc-functiondef.json`
cat << EOF > tmp-config-functions.json
[{
    "Id": "uptime-lambda",
    "FunctionArn": "$GGC_LAMBDA_HWALIASARN",
    "FunctionConfiguration": {
        "Executable": "greengrassHelloWorld.lambda_handler",
        "MemorySize": 128000,
        "Pinned": true,
        "Timeout": 3
```

```
    }
}]
EOF
aws greengrass create-function-definition-version \
--function-definition-id "$GGC_FUNCDEFID" \
--functions file://tmp-config-functions.json > tmp-ggc-functions.json
GGC_FUNCDEFVERS=`jq -r ".Arn" tmp-ggc-functions.json`
```

Create a Subscription Definition

The subscription defines the routing of data through the GGC. The example below contains a single route. It takes any (#) mqtt message posted by the Lambda function created above and routes it to the AWS IoT device gateway (cloud). You can also set this to the specific topic that the HelloWorld example posts to of "hello/world". You will see these messages coming in to the cloud by using the test tool within the AWS IoT Console.

```
aws greengrass create-subscription-definition --name "$GGC_DEPLOYMENT-
Subscription" > tmp-ggc-sub.json
GGC_SUBSID=`jq -r ".Id" tmp-ggc-sub.json`
cat << EOF > tmp-config-subscriptions.json
[{
    "Id": "1",
    "Source": "$GGC_LAMBDA_HWALIASARN",
    "Subject": "#",
    "Target": "cloud"
}]
EOF
aws greengrass create-subscription-definition-version --subscription-
definition-id "$GGC_SUBSID" \
--subscriptions file://tmp-config-subscriptions.json > tmp-ggc-
subsversion.json
GGC_SUBSVERSION=`jq -r ".Arn" tmp-ggc-subsversion.json`
```

Create a Logger Definition

A logger definition is not required. It tells your Greengrass deployment where to store the logs generated by GGC and your Lambda functions for debug purposes. The example below stores them locally but they may also be sent to Cloudwatch.

```
aws greengrass create-logger-definition --name "$GGC_DEPLOYMENT-Logger" >
tmp-ggc-logger.json
GGC_LOGGERID=`jq -r ".Id" tmp-ggc-logger.json`
cat << EOF > tmp-config-loggers.json
[
```

```
    {
        "Id": "system-logs",
        "Component": "GreengrassSystem",
        "Level": "INFO",
        "Space": "5120",
        "Type": "FileSystem"
    },
    {
        "Id": "lambda-logs",
        "Component": "Lambda",
        "Level": "DEBUG",
        "Space": "5120",
        "Type": "FileSystem"
    }
]
EOF
aws greengrass create-logger-definition-version --logger-definition-id
"$GGC_LOGGERID" \
--loggers file://tmp-config-loggers.json > tmp-ggc-loggerversion.json
GGC_LOGGERVERSION=`jq -r ".Arn" tmp-ggc-loggerversion.json`
```

Update Your Group Version

Now that you have created all your “Definitions” you are ready to create a “Group Version” pointing to them. This “Group Version” is what will define your deployment. If you need to change any of your definitions, you will repeat this step to create a new version that can be deployed.

```
aws greengrass create-group-version --group-id "$GGC_GROUPID" \
    --core-definition-version-arn "$GGC_COREVERSION" \
    --function-definition-version-arn "$GGC_FUNCDEFVERS" \
    --logger-definition-version-arn "$GGC_LOGGERVERSION" \
    --subscription-definition-version-arn "$GGC_SUBSVERSION" \
> tmp-ggc-grpversion.json
GGC_GROUPIDVERSION=`jq -r ".Version" tmp-ggc-grpversion.json`
```

Create a Deployment

The final step is to deploy the Greengrass Group to your GGC. Since this is the first deployment, the `deployment-type` is `NewDeployment`. If you want to update an existing deployment, you would set it to Redeployment to replace an existing deployment.

```
aws greengrass create-deployment --deployment-type NewDeployment \
--group-id "$GGC_GROUPID" --group-version-id "$GGC_GROUPIDVERSION" \
> tmp-ggc-deployment.json
```

```
GGC_DEPLOYID=`jq -r ".Id" tmp-ggc-deployment.json`
```

You can check the deployment status with:

```
aws greengrass get-deployment-status --deployment-id "$GGC_DEPLOYID" --group-id "$GGC_GROUPID"
```

Your deployment should be up and running and the Lambda function shall be publishing continuously to the hello/world topic. To see this, go to the AWS IoT Console and click on the "Test" option in the left panel. This will bring up an MQTT Client where you can enter # in the Subscription topic to see all messages or "hello/world" if you only want to see the ones from the example Lambda function. After clicking "Subscribe to topic", the topic you entered will appear in the left "Subscriptions" panel. Click on it to see that subscription.

Updating the Connectivity Information Service

This section is not required for the HelloWorld example as it does not have IoT Devices connecting to the GGC.

For IoT Devices to connect to the GGC they need to know the IP address of the GGC. The `update-connectivity-info` command is used to provide that information. IoT Devices will be able to get this information as part of their discovery process. The cloud will generate an SSL certificate for the GGC and let it know to update. As part of the discovery process, the IoT Devices will get the root CA they can use to validate the GGC certificate.

This example assumes all IoT Devices will be running locally on the GGC. Hence the `HostAddress` of 127.0.0.1.

```
aws greengrass update-connectivity-info --thing-name "$GGC_DEPLOYMENT" --connectivity-info \
'[ {
    "Id": "localhost",
    "HostAddress": "127.0.0.1",
    "PortNumber": 8443,
    "Metadata": "DragonBoard GG Demo on 410c"
}]'
```

GGC Certificate Authority

If you are using an AWS IoT SDK that is not Greengrass aware (i.e., does not have the discovery functionality) such as the Python SDK, you will need to fetch the CA certificate prior to running the IoT Device or it will not be able to verify the authenticity of the GGC. As of this writing only the C++ IoT SDK has the automatic discovery function.

This command first gets the group id (`GGC_GROUPID`) from an earlier JSON response.

```
GGC_GROUPID=`jq -r ".Id" tmp-ggc-group.json`
aws greengrass list-group-certificate-authorities \
--group-id $GGC_GROUPID > tmp-ggc-grpcerts.json
GGC_GRPCERTID=`jq -r ".GroupCertificateAuthorities[].GroupCertificateAuthorityId"
tmp-ggc-grpcerts.json`
aws greengrass get-group-certificate-authority --group-id $GGC_GROUPID \
--certificate-authority-id $GGC_GRPCERTID | jq -r ".PemEncodedCertificate" >
localggc-ca.pem
```

The file `localggc-ca.pem` will contain the CA you will need when configuring the IoT device. Please note that the endpoint the IoT device connects to must match the `HostAddress` that was set in the `update-connectivity-info` above or the connection will fail. If the connectivity info is updated on a running system, the IoT Device will need to be reconfigured with the new endpoint and possibly an updated CA. An IoT Device that uses a Greengrass Aware SDK does not require any reconfiguring as that would happen automatically.

NOTE: Example Code Coming Soon…

The project will also be expanded shortly to include examples that show:

- A "virtual" sensor which outputs data and is processed using a Lambda function running locally on the DragonBoard 410c
  How to use actual sensors and a 96boards Sensors Mezzanine board attached to the DragonBoard 410c