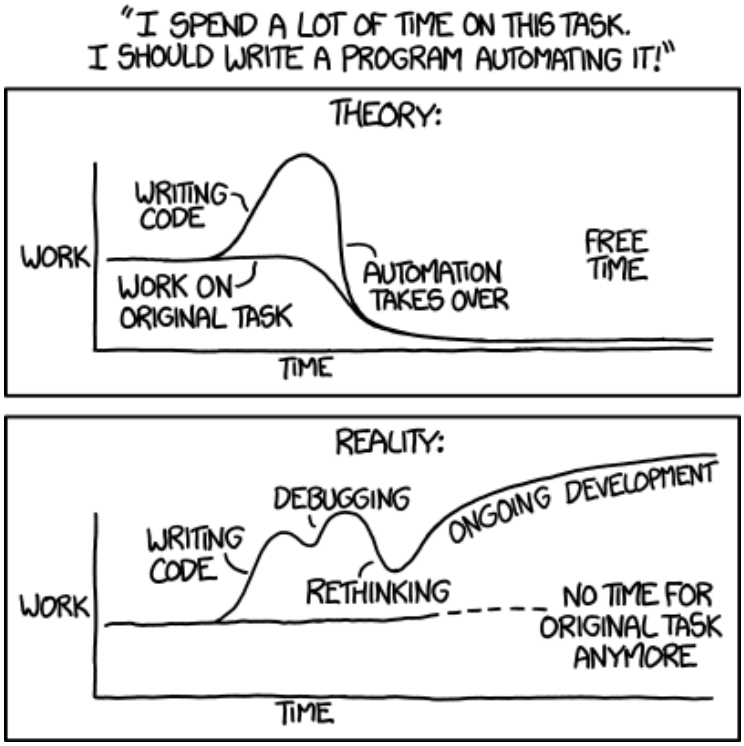
How to onboard thousand of devices on AWS Greengrass and live happily

3 February 2023 - 9 min. read

AWS Greengrass *Internet of Things (IoT)*

"If I had six hours to chop down a tree, I'd spend the first four hours sharpening the axe."
Abraham Lincoln.

I must confess: I'm lazy, so lazy that I spend my time working on automating everything. My fellow colleagues know my attitude, and I hope to avoid ending in this situation:



https://xkcd.com/1319/

When it comes to massive deployments, automation is the only strategy that guarantees survival in the IT world.

A customer asked us to help him use its extra edge computational capacity spread over the country. Every branch had a virtualization environment: running on-demand workloads with the least possible maintenance and configuration effort would be the icing on the cake.

Since everything is on the edge and networking communication could be tricky, we thought about AWS Greengrass. Mattia already used Greengrass to deploy long-running and on-demand functions at the edge; we were confident that the technology could fit our case. Our last problem to solve was avoiding to onboard every virtual machine manually.

Lucky for us, Greengrass Fleet provisioning is the solution we were searching for. It allows massive deployments of edge devices, maintaining security over the enrollment process.

To automatically provision a fleet of devices, we need first to authenticate our request during the onboarding phase; otherwise, everyone could add themselves and do evil things inside our deployment. Once the provisioning phase is completed, a unique certificate will be assigned to that device and will be used as the authentication method to interact with AWS Services.

Two authentication methods (claims and trusted users) are available; their usage depends on the deployment scenario.

Provisioning using a trusted user applies to scenarios when user onboarding needs user interaction (like wifi-connected appliances with companion mobile apps).

It impersonates a user who requests a certificate on behalf of the IoT device, then uploads the final certificate to the device.

Provisioning using claims uses a "base" certificate (stored on the device) to authenticate and request the final certificate directly from the IoT device. This scenario requires no user interaction, so massive deployments are a perfect fit.

Bingo! We have our solution: we will use the AWS IoT Greengrass Core with the AWS IoT fleet provisioning plugin.

First, let's dig into the claims provisioning workflow to understand it better.

1. Create a provisioning certificate and allow it to obtain final certificates by attaching a policy
2. Create a provisioning template that automatically fills device details (like name, group, or serial number) in AWS IoT Core

3. Install Greengrass software on a template edge device,
4. Configure the fleet provisioning plugin and customize the deployment.
5. Start deploying devices!

Since we are lazy, we will use scripting and CloudFormation as much as possible. As a bonus point, we'll create a golden virtual machine image with all the installed components we can use as a template to deploy at the edge.

First, a script will create and save provisioning certificates using AWS CLI. We will save them using AWS SecretsManager, as you will see later, we use SecretsManager to automate their deployment on the golden virtual machine image.

```
#!/bin/bash
ENV=${1}


SECRETSMANAGER_PREFIX="MyAwesomeProjectSecret"
THINGGROUP_NAME="MyThingGroup"

# Create the certificate
CERTIFICATE_ARN=$(aws iot create-keys-and-certificate \
                --certificate-pem-outfile "claim-certs/claim.pem.crt" \
                --public-key-outfile "claim-certs/claim.public.pem.key" \
                --private-key-outfile "claim-certs/claim.private.pem.key"
\
                --set-as-active | jq -r .certificateArn)

SECRET_STRING=$(echo "$(cat claim-certs/claim.pem.crt)|$(cat claim-certs/
claim.private.pem.key)"  | tr '\n' ';' )
aws secretsmanager create-secret --name $SECRETSMANAGER_PREFIX/$ENV/claim
s-certificate --secret-string $SECRET_STRING

# Attach the policy to the certificate
aws iot attach-policy --policy-name GreengrassProvisioningClaimPolicy --t
arget $CERTIFICATE_ARN
aws iot create-thing-group --thing-group-name $THINGGROUP_NAME
```

Run this script before deploying this CloudFormation template that will create the thing group, policies to associate it with claim certificates, and the required service roles.

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: GreenGrass with Fleet Manager provisioning template

Metadata:

  'AWS::CloudFormation::Interface':
        ParameterGroups:
        - Label: {default: 'Required parameters'}
        Parameters:
        - Env
        - ThingGroupName
        - Label: {default: 'Optional parameters'}
        Parameters:
        - ProjectName

Parameters:

  Env:
        Type: String
        Description: "Insert the environment"

  ProjectName:
        Type: String
        Description: "Insert the name of the project"

  ThingGroupName:
        Type: String
        Description: "Insert the Thing Group name"

Resources:

  GreengrassTokenExchangeRole:
        Type: AWS::IAM::Role
```

```
        Properties:
        AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: 'Allow'
                Principal:
                Service:
                - 'credentials.iot.amazonaws.com'
                Action:
                - 'sts:AssumeRole'
        - Effect: 'Allow'
        Policies:
        - PolicyName: 'LogPersmission'
        PolicyDocument:
                Version: '2012-10-17'
                Statement:
                - Effect: 'Allow'
                Action:
                - "logs:CreateLogGroup"
                - "logs:CreateLogStream"
                - "logs:PutLogEvents"
                - "logs:DescribeLogStreams"
                - "s3:GetBucketLocation"
                - "iam:PassRole"
                Resource: '*'
                RoleName: !Sub '${ProjectName}-${Env}-V2TokenExchangeRol
e'

  GreengrassTokenExchangeRoleAlias:
        Type: AWS::IoT::RoleAlias
        Properties:
        RoleAlias: !Sub '${ProjectName}-${Env}-CoreTokenExchangeRoleAlia
s'
        RoleArn: !GetAtt GreengrassTokenExchangeRole.Arn
        Tags:
        - Key: Name
        Value: !Sub "${ProjectName}-${Env}-CoreTokenExchangeRoleAlias"
```

```
  GreengrassV2IoTThingPolicy:
        Type: AWS::IoT::Policy
        Properties:
        PolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: 'Allow'
                Action:
                - "iot:Publish"
                - "iot:Subscribe"
                - "iot:Receive"
                - "iot:Connect"
                - "greengrass:*"
                Resource: "*"
        - Effect: 'Allow'
                Action:
                - "iot:AssumeRoleWithCertificate"
                Resource: !GetAtt GreengrassTokenExchangeRoleAlias.RoleAl
iasArn
        PolicyName: V2IoTThingPolicy

  GreengrassFleetProvisioningRole:
        Type: AWS::IAM::Role
        Properties:
        AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
        - Effect: 'Allow'
                Principal:
                Service:
                - 'iot.amazonaws.com'
                Action:
                - 'sts:AssumeRole'
        Policies:
        - PolicyName: 'ECRAccess'
        PolicyDocument:
```

```
            Version: '2012-10-17'
            Statement:
            - Effect: 'Allow'
            Action:
            - "logs:CreateLogGroup"
            - "logs:CreateLogStream"
            - "logs:PutLogEvents"
            - "logs:DescribeLogStreams"
            - "s3:GetBucketLocation"
            Resource: '*'
      RoleName: !Sub '${ProjectName}-${Env}-FleetProvisioningRole'

GreengrassFleetProvisioningTemplate:
      Type: AWS::IoT::ProvisioningTemplate
      Properties:
      Description: "template to provision iot fleet resources"
      Enabled: true
      ProvisioningRoleArn: !GetAtt GreengrassFleetProvisioningRole.Arn
      TemplateBody: |
      {
      "Parameters": {
            "ThingName": {
            "Type": "String"
            },
            "ThingGroupName": {
            "Type": "String"
            },
            "AWS::IoT::Certificate::Id": {
            "Type": "String"
            }
      },
      "Resources": {
            "MyThing": {
            "OverrideSettings": {
            "AttributePayload": "REPLACE",
            "ThingGroups": "REPLACE",
            "ThingTypeName": "REPLACE"
```

```
                },
                "Properties": {
                "AttributePayload": {},
                "ThingGroups": [
                {
                        "Ref": "ThingGroupName"
                }
                ],
                "ThingName": {
                "Ref": "ThingName"
                }
                },
                "Type": "AWS::IoT::Thing"
                },
                "MyPolicy": {
                "Properties": {
                "PolicyName": "V2IoTThingPolicy"
                },
                "Type": "AWS::IoT::Policy"
                },
                "MyCertificate": {
                "Properties": {
                "CertificateId": {
                "Ref": "AWS::IoT::Certificate::Id"
                },
                "Status": "Active"
                },
                "Type": "AWS::IoT::Certificate"
                }
        }
        }
        TemplateName: !Sub "${ProjectName}-${Env}-FleetTemplate"

  GreengrassProvisioningClaimPolicy:
        Type: AWS::IoT::Policy
        Properties:
        PolicyDocument:
```

```
        Version: '2012-10-17'
        Statement:
        - Effect: 'Allow'
                Action:
                - "iot:Connect"
                Resource: "*"
        - Effect: 'Allow'
                Action:
                - "iot:Subscribe"
                Resource:
                - !Sub "arn:aws:iot:${AWS::Region}:${AWS::AccountId}:topi
cfilter/$aws/certificates/create/*"
                - !Sub "arn:aws:iot:${AWS::Region}:${AWS::AccountId}:topi
cfilter/$aws/provisioning-templates/${GreengrassFleetProvisioningTemplat
e}/provision/*"
        - Effect: 'Allow'
                Action:
                - "iot:Publish"
                - "iot:Receive"
                Resource:
                - !Sub "arn:aws:iot:${AWS::Region}:${AWS::AccountId}:topi
c/$aws/certificates/create/*"
                - !Sub "arn:aws:iot:${AWS::Region}:${AWS::AccountId}:topi
c/$aws/provisioning-templates/${GreengrassFleetProvisioningTemplate}/prov
ision/*"
        PolicyName: GreengrassProvisioningClaimPolicy
```

The

```
GreengrassFleetProvisioningTemplate
```

resource will register our IoT device using a GroupName and a ThingName. You can personalize the template according to your needs. Later, you will see that "ThingName" will be our VM hostname.

If you want to get notified and run custom actions when a device registers, you can add a rule like this and the corresponding SNS topics (I will not include the complete code since it's a simple lambda function)

```
ThingCreationIotRule:
        Type: AWS::IoT::TopicRule
        Properties:
        RuleName: !Sub '${ProjectName}_${Env}_greengrass_rule'
        TopicRulePayload:
        RuleDisabled: 'false'
        Sql: SELECT * FROM '$aws/events/thing/#'
        Actions:
        - Lambda:
                FunctionArn: !GetAtt RegisterThingLambda.Arn
        ErrorAction:
        Sns:
                TargetArn: !Ref SNSFailedNotificationTopic
                RoleArn: !GetAtt NotificationRole.Arn
```

Now it's time to create our golden virtual machine image using Ubuntu 22.04 as a base install; you use VirtualBox or VMware as a testing and development environment.

The following script will install the required packages and retrieve claim certificates. To use it, you should set temporary AWS IAM credentials as environment variables or, for better, integrate everything in a packer template.

```
#!/bin/bash

ENV=${1}

SECRETSMANAGER_PREFIX="MyAwesomeProjectSecret"

mkdir -p claim-certs
SECRET_STRING=$(aws secretsmanager get-secret-value --secret-id $SECRETSM
ANAGER_PREFIX/$ENV/claims-certificate --query SecretString --output text)

echo $SECRET_STRING | cut -d '|' -f 1 | tr ';' '\n' > claim-certs/claim.p
em.crt
echo $SECRET_STRING | cut -d '|' -f 2 | tr ';' '\n' > claim-certs/claim.p
rivate.pem.key
```

```
export DEBIAN_FRONTEND=noninteractive
apt update && apt upgrade -y
apt -yq install python3-pip zip open-vm-tools default-jdk libgl1-mesa-glx


mkdir -p /greengrass/v2/installer
mkdir /greengrass/v2/claim-certs


mv claim-certs/claim.private.pem.key /greengrass/v2/claim-certs/
mv /claim-certs/claim.pem.crt /greengrass/v2/claim-certs/



cd /greengrass/v2
curl -o /greengrass/v2/AmazonRootCA1.pem https://www.amazontrust.com/repository/AmazonRootCA1.pem
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/greengrass-nucleus-latest.zip > greengrass-nucleus-latest.zip
unzip greengrass-nucleus-latest.zip -d installer && rm greengrass-nucleus-latest.zip
curl -s https://d2s8p88vqu9w66.cloudfront.net/releases/aws-greengrass-FleetProvisioningByClaim/fleetprovisioningbyclaim-latest.jar > installer/aws.greengrass.FleetProvisioningByClaim.jar
```

All the required Greengrass components are now installed. Let's now add the ultimate lazy sysadmin touch to our solution: we definitely want to have something that, when started, configures itself without user intervention.

We will define a template for our Greengrass configuration and, taking advantage of

```
systemd
```

unit and scripting, use the virtual machine's hostname as ThingName
Create a file named config.yml in

```
/greengrass/v2/installer/
```

specifying that we will use the IoT fleet provisioning plugin.

```
---
services:
  aws.greengrass.Nucleus:
        version: "2.9.1"
  aws.greengrass.FleetProvisioningByClaim:
        configuration:
        rootPath: "/greengrass/v2"
        awsRegion: "eu-west-1"
        iotDataEndpoint: "endpoint.iot.region.amazonaws.com"
        iotCredentialEndpoint: "endpoint.credentials.iot.region.amazonaw
s.com"
        iotRoleAlias: "myproject-environment-CoreTokenExchangeRoleAlias"
        provisioningTemplate: "myproject-environment-FleetTemplate"
        claimCertificatePath: "/greengrass/v2/claim-certs/claim.pem.crt"
        claimCertificatePrivateKeyPath: "/greengrass/v2/claim-certs/clai
m.private.pem.key"
        rootCaPath: "/greengrass/v2/AmazonRootCA1.pem"
        templateParameters:
        ThingName: "REPLACEME"
        ThingGroupName: "MyThingGroup"
```

Change this file according to your environment. You can obtain

```
iotDataEndpoint
```

and

```
iotCredentialEndpoint
```

values by issuing these commands:

```
aws iot describe-endpoint --endpoint-type iot:Data-ATS
aws iot describe-endpoint --endpoint-type iot:CredentialProvider
```

As you can see, ThingName value is set as "**REPLACEME**". A script

```
(/greengrass/configure-device.sh)
```

will replace this value and a systemd service will run it at the first boot.

```
#!/bin/bash

set -e

sed -i s/REPLACEME/$(hostname)/g /greengrass/v2/installer/config.yaml
echo "********************** RUN SERVICE ************************"
java -Droot="/greengrass/v2" -Dlog.store=FILE \
-jar /greengrass/v2/installer/lib/Greengrass.jar \
--trusted-plugin /greengrass/v2/installer/aws.greengrass.FleetProvisioningByClaim.jar \
--init-config /greengrass/v2/installer/config.yaml \
--component-default-user ggc_user:ggc_group \
--setup-system-service true
```

The first line

```
(sed -i s/REPLACEME/$(hostname)/g /greengrass/v2/installer/config.yaml)
```

replaces the value with the virtual machine's current hostname.

Create now a unit file in the /lib/systemd/system directory with a meaningful name (we used "greengrass-config.service")

```
[Unit]
Before=systemd-user-sessions.service
Wants=network-online.target
After=network-online.target
ConditionPathExists=!/greengrass/greengrass_installed

[Service]
Type=oneshot
ExecStart=/greengrass/configure-device.sh
```

```
ExecStartPost=/usr/bin/touch /greengrass/greengrass_installed
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

This is a pretty standard systemd unit, with an exception:

```
ConditionPathExists=!/greengrass/greengrass_installed
```

tells systemd to run this unit only if this file does not exist.

```
ExecStartPost=/usr/bin/touch /greengrass/greengrass_installed
```

tells systemd to create the file that, so after the first time, the service will not run anymore.

Activate service at boot with

```
systemctl daemon-reload
systemctl enable greengrass-config.service
```

Once you finish this configuration, just export your virtual machine image in a format such as OVA (or create a VMware template) and deploy it!

Remember: this is only a sample configuration; you may need to optimize and update your system to shrink disk size and disable unnecessary system services.

Once you deploy this solution, you can run functions on your distributed environment and take advantage of all the unused computing resources you may have at your disposal on edge.

Other AWS Services can use this solution as a base, like running managed SageMaker jobs at the edge, but let's keep this topic for another time.

Today we saw how automation could relieve us from the execution of boring tasks, even if, at first glance, everything seems to get more complicated.

Did you ever find yourself in an automation frenzy to accommodate laziness?

Let us know in the comments!

See you in 14 days on **#Proud2beCloud**

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Damiano Giorgi

Ex on-prem systems engineer, lazy and prone to automating boring tasks. In constant search of technological innovations and new exciting things to experience. And that's why I love Cloud Computing! At this moment, the only "hardware" I regularly dedicate myself to is that my bass; if you can't find me in the office or in the band room try at the pub or at some airport, then!
