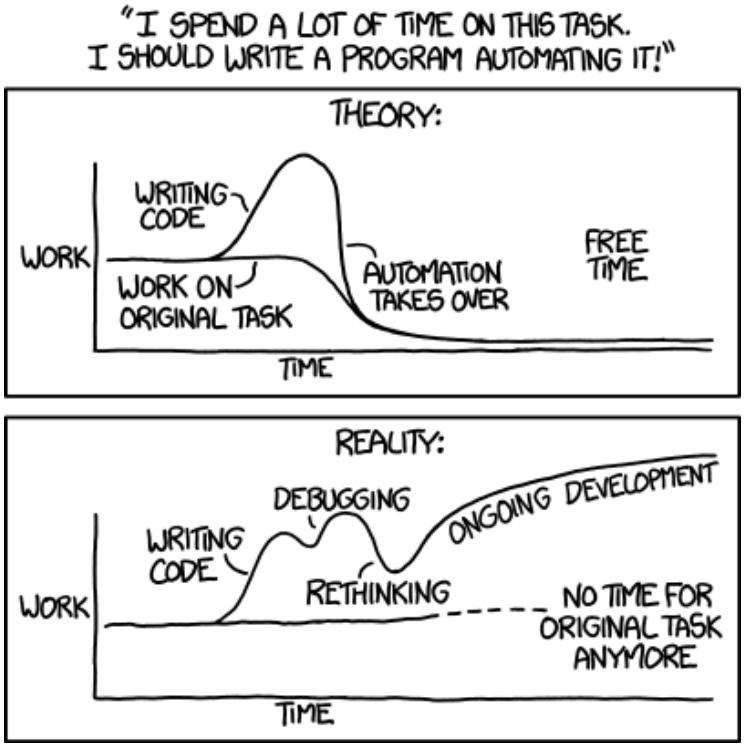
Come fare il setup di migliaia di dispositivi su AWS Greengrass e vivere felici

3 Febbraio 2023 - 10 min. read

AWS Greengrass *Internet of Things (IoT)*

Dammi sei ore per abbattere un albero e spenderò le prime quattro ore per affilare l'ascia.
Abraham Lincoln.

Lo confesso: sono pigro, talmente pigro che spendo la maggior parte del mio tempo ad automatizzare qualsiasi cosa, ne sanno qualcosa i miei colleghi. Spero solo di non finire mai in questo modo:



https://xkcd.com/1319/

Quando si parla di deployment massivi, l'automazione è l'unica strategia di sopravvivenza nel mondo IT.

Un cliente ci ha chiesto aiuto per utilizzare la potenza computazionale inutilizzata di migliaia di server sparsi per tutta Italia. Ogni ufficio ha a disposizione un server con un hypervisor, quindi poter eseguire workload in modalità on-demand senza appesantire le operazioni di manutenzione e configurazione sarebbe la ciliegina sulla torta.

Dato che la parte computazionale sarà "on the edge" e la comunicazione di rete potrebbe non essere facile, abbiamo pensato di utilizzare **AWS Greengrass** per semplificare questi aspetti.

Nell'articolo di Mattia potete trovare già un esempio di come utilizzare Greengrass per fare il deploy di funzioni long-running e on-demand.

Siamo sicuri che la soluzione possa fare al caso nostro, ci manca però trovare un modo per evitare di configurare manualmente ogni singola virtual machine.

Per nostra fortuna, Greengrass Fleet provisioning è la soluzione che stiamo cercando: permette di fare deploy massivi di edge device, rendendo sicuro il processo di onboarding.

Per fare il provisioning automatico di una flotta di dispositivi dobbiamo per prima cosa trovare un meccanismo che permetta di autenticare la richiesta di onboard, altrimenti chiunque potrebbe aggiungere un dispositivo e, potenzialmente, causare danni.

Una volta fatta la prima autenticazione, al device viene assegnato un certificato "definitivo" che gli permetterrà di essere riconosciuto.

Esistono due meccanismi di autenticazione: **claim** e **trusted users**. Come vedremo, il loro utilizzo dipende dal caso d'uso.

Il provisioning mediante "trusted user" è solitamente utilizzato quando è richiesta l'interazione dell'utente, ad esempio per dispositivi wi-fi smart controllati da applicazioni mobili.

La richiesta è fatta con un utente per conto del dispositivo. Il certificato definitivo, una volta ottenuto, dovrà essere caricato sul device.

Il provisioning utilizzando claim fa in modo che sia direttamente il dispositivo ad effettuare la richiesta, utilizzando per l'autenticazione certificati "base" già installati. Questo scenario

non richiede interazione da parte dell'utente, per cui i deploy massivi sono il caso d'uso perfetto.

Eureka! Abbiamo la nostra soluzione: **useremo AWS IoT Greengrass Core con il plugin per il fleet provisioning**.

Per prima cosa approfondiamo i passi necessari. Per usare il provisioning con claim occorre:

1. Creare un certificato e, mediante una policy, autorizzarlo ad ottenere nuovi certificati
2. Creare un template in AWS IoT Core che definisca i dettagli identificativi del dispositivo (come ad esempio nome, gruppo, numero di serie)
3. Installare il software Greengrass su una macchina che sarà usata come template
4. Configurare il plugin per il fleet provisioning e personalizzare il deployment
5. Iniziare a fare il deploy di tutti i dispositivi

Siccome siamo pigri, useremo CloudFormation e script per automatizzare il più possibile. Come bonus, creeremo una virtual machine di riferimento, installando tutti i componenti che andremo ad usare.

Per prima cosa, creiamo uno script per creare e salvare su SecretsManager i certificati per il provisioning. In questo modo, riusciremo ad automatizzare il loro deploy sulla macchina virtuale

```
#!/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
```

Questo script va usato prima di fare il deploy del template CloudFormation che segue, che si occuperà di creare il grouppo, le policy da associare con i certificati ed i ruoli necessari.

```
---
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
```

La risorsa

```
GreengrassFleetProvisioningTemplate
```

farà in modo che il nostro IoT device si registrerà usando le proprietà **GroupName** e **ThingName**. Le proprietà sono personalizzabili a seconda delle necessità. In seguito vedremo che "ThingName" sarà popolato con il valore dell'hostname della macchina.

Per ricevere una notifica ed eseguire azioni in modo automatico quando un dispositivo si registra, è possibile aggiungere una regola come quella che segue, aggiungendo anche un topic SNS (non includeremo il codice completo per brevità):

```
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
```

È finalmente arrivata l'ora di creare una macchina virtuale. Useremo Ubuntu 22.04 per la base, va benissimo usare VirtualBox o Vmware.

Lo script che segue si occupa di installare il software e ottenere i certificati da usare per il claim provisioning. Per usarlo occorre ottenere credenziali temporanee IAM come variabili di ambiente o, ancora meglio, integrare tutto in un template packer.

```
#!/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/repo
sitory/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-Fle
etProvisioningByClaim/fleetprovisioningbyclaim-latest.jar > installer/aw
s.greengrass.FleetProvisioningByClaim.jar
```

Tutti i componenti necessari sono ora installati. Ora manca l'ultimo tocco del sysadmin pigro: vogliamo qualcosa che, una volta avviata, si configuri in autonomia senza dover intervenire manualmente.

Faremo in modo che il campo "ThingName" sia popolato automaticamente con l'hostname grazie ad una unit systemd ed uno script.
Basta creare un file config.yml in

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

specificando che useremo il plugin IoT fleet provisioning.

```
---
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"
```

Questo file va adattato con i valori corretti. Il valore di

```
iotDataEndpoint
```

e

```
iotCredentialEndpoint
```

con questi comandi:

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

Il valore di ThingName è impostato a "**REPLACEME**" Lo script

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

si occuperà di sostituire il valore e un servizio systemd sarà eseguito al primo avvio.

```
#!/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.FleetProvisioning
gByClaim.jar \
--init-config /greengrass/v2/installer/config.yaml \
--component-default-user ggc_user:ggc_group \
--setup-system-service true
```

La prima riga (sed -i s/REPLACEME/$(hostname)/g /greengrass/v2/installer/config.yaml) sostituisce il valore con l'hostname.

Ora basta creare una unit nella directory /lib/systemd/system con un nome parlante (abbiamo usato 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
```

Si tratta di una unit systemd quasi standard, con due piccole particolarità:

La riga

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

fa si che systemd esegua il comando solo se il file non esiste

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

fa si che il file sia creato. In questop modo il serivizo sarà eseguito solo una volta.

Non ci resta che attivare ed abilitare il tutto con:

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

Finita la configurazione, basta esportare l'immagine della macchina virtuale in un formato compatibile con l'hypervisor (OVA o template VMware) ed iniziare a fare il deploy massivo!

Un ultimo appunto: questa è una configurazione di base, è necessario fare in modo che sia aggiornata ed ottimizzata (ad esempio si può ridurre la dimensione del disco disinstallando e disabilitando servizi non essenziali)

Al termine di questo percorso possiamo finalmente eseguire long-running functions usando le risorse inutilizzabili at the edge.

È possibile integrare altri servizi AWS usando questa soluzione come base: i job SageMaker sono un buon esempio (e magari uno spunto che approfondiremo in futuro).

Abbiamo visto come l'automazione ci può aiutare liberandoci dal dover eseguire compiti noiosi, anche se, a prima vista, può sembrare complesso.

Vi è mai capitato di farvi prendere dalla frenesia dell'automazione per troppa pigrizia?

Fatecelo sapere nei commenti!

Ci vediamo tra 14 giorni su **#Proud2beCloud**

About Proud2beCloud

Proud2beCloud è il blog di beSharp, APN Premier Consulting Partner italiano esperto nella progettazione, implementazione e gestione di infrastrutture Cloud complesse e servizi AWS avanzati. Prima di essere scrittori, siamo Solutions Architect che, dal 2007, lavorano quotidianamente con i servizi AWS. Siamo innovatori alla costante ricerca della soluzione più all'avanguardia per noi e per i nostri clienti. Su Proud2beCloud condividiamo regolarmente i nostri migliori spunti con chi come noi, per lavoro o per passione, lavora con il Cloud di AWS. Partecipa alla discussione!

Damiano Giorgi

Ex sistemista on-prem, pigro e incline all'automazione di task noiosi. Alla ricerca costante di novità tecnologiche e quindi passato al cloud per trovare nuovi stimoli.L'unico hardware a cui mi dedico ora è quello del mio basso; se non mi trovate in ufficio o in sala prove provate al pub o in qualche aeroporto!
