**Despliegue de infraestructura IoT con AWS**

El propósito del presente documento es ilustrar algunas de las funcionalidades disponibles en AWS para la integración con dispositivos IoT. En este caso, usaremos los siguientes elementos:

-Una placa Raspberry Pi

-AWS IoT Core

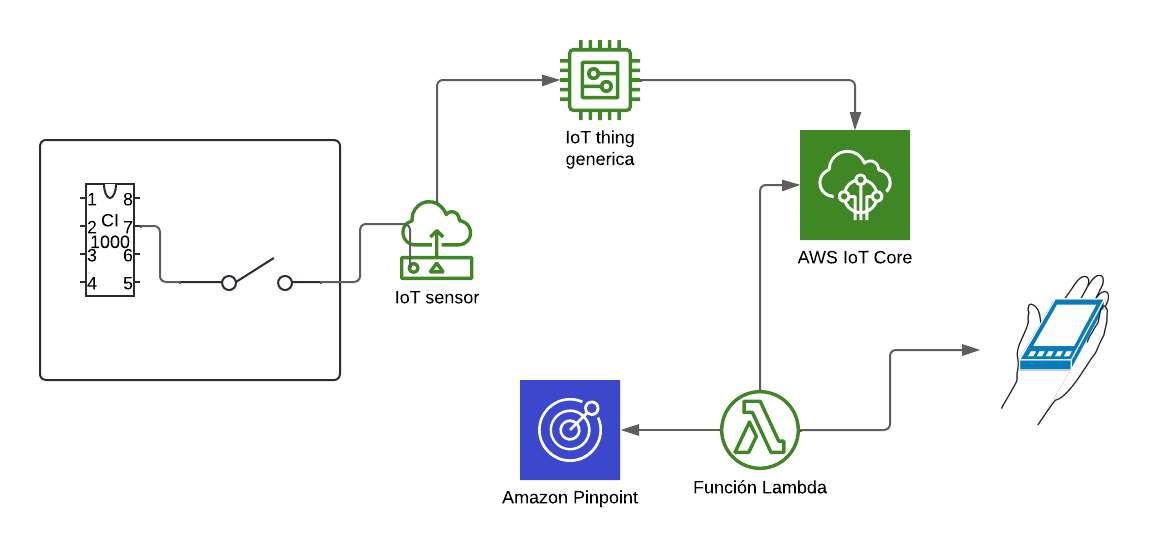
-AWS Lambda

-AWS Pinpoint

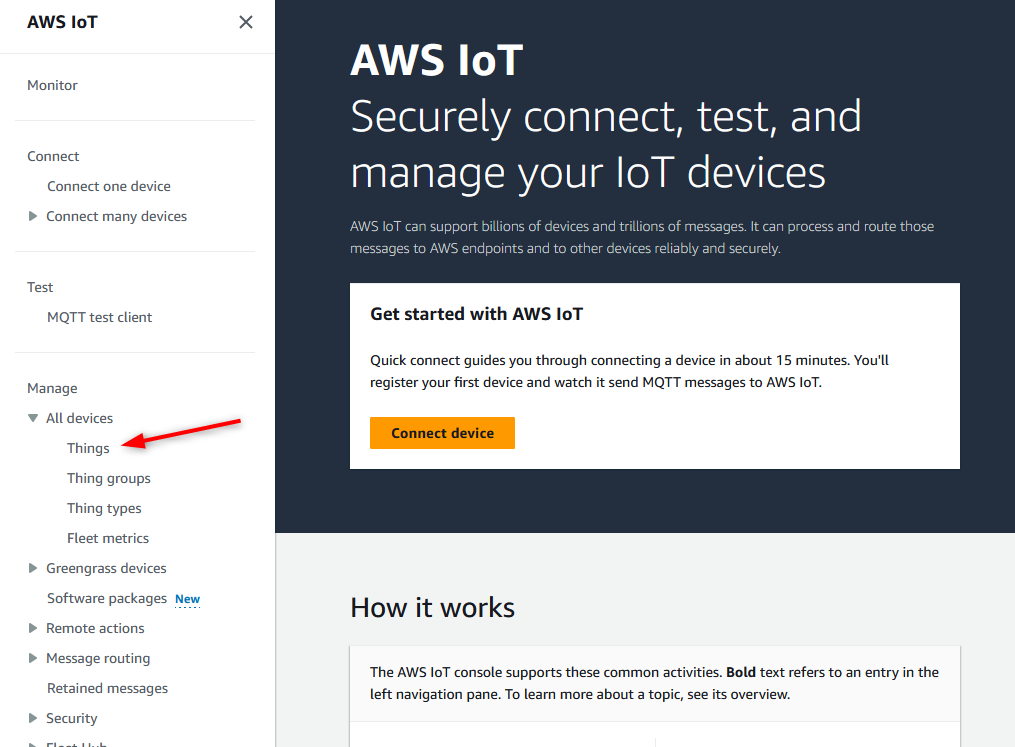
El propósito es interactuar a través del dispositivo Raspberry con IoT Core, enviando un mensaje ante el cumplimiento de una condición. En el caso de la práctica, mostramos el uso de un sensor de proximidad (FC-51), pero puede sustituirse con cualquier elemento. Una vez disparada la condición, la placa envía el mensaje usando MQTT a AWS IoT Core, el que disparará la ejecución de una función Lambda la cual interactúa con AWS Pinpoint para enviar un SMS de notificación.

Claro está que la función Lambda puede modificarse para realizar la acción que sea requerida, ante la recepción del mensaje de la Raspberry.

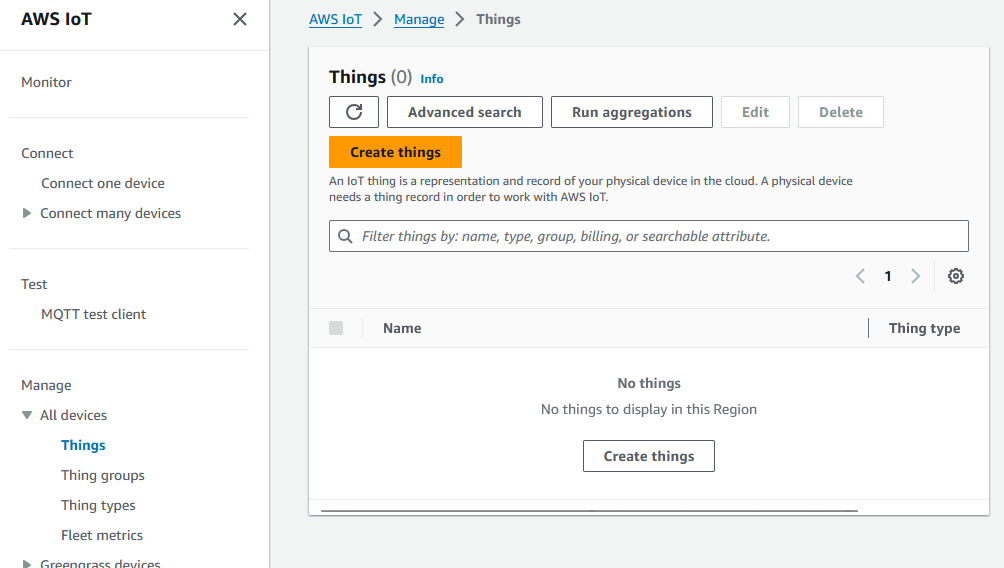
El diagrama de la solución es el siguiente:



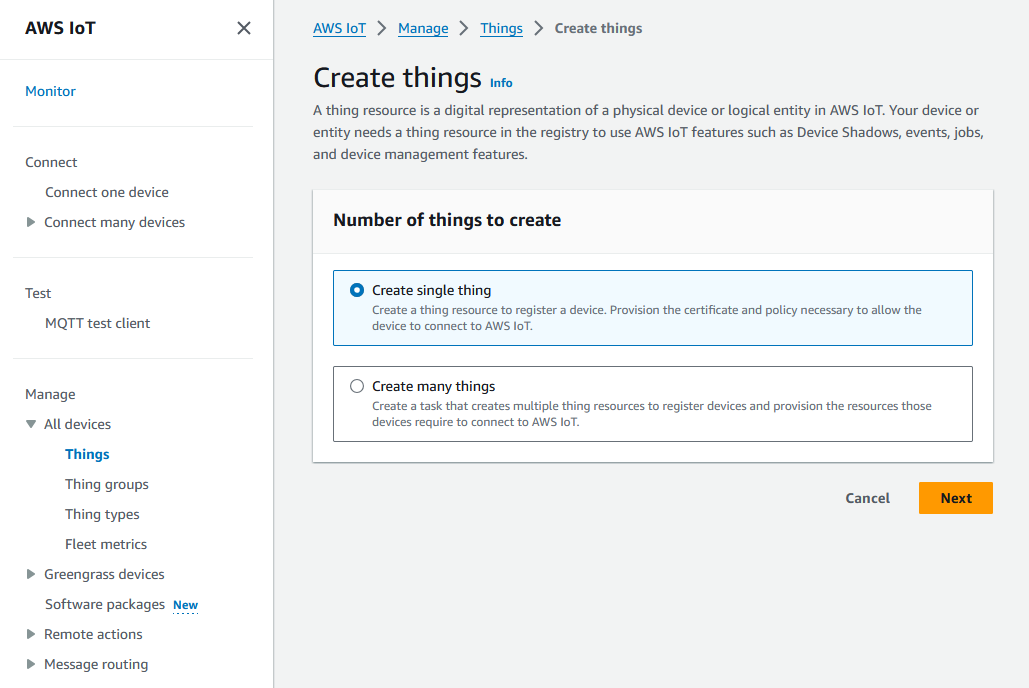
Avancemos en el desarrollo de la solución.

Entramos dentro de la consola web de AWS a IoT Core, y ahí a la sección Things, para crear nuestro dispositivo.  


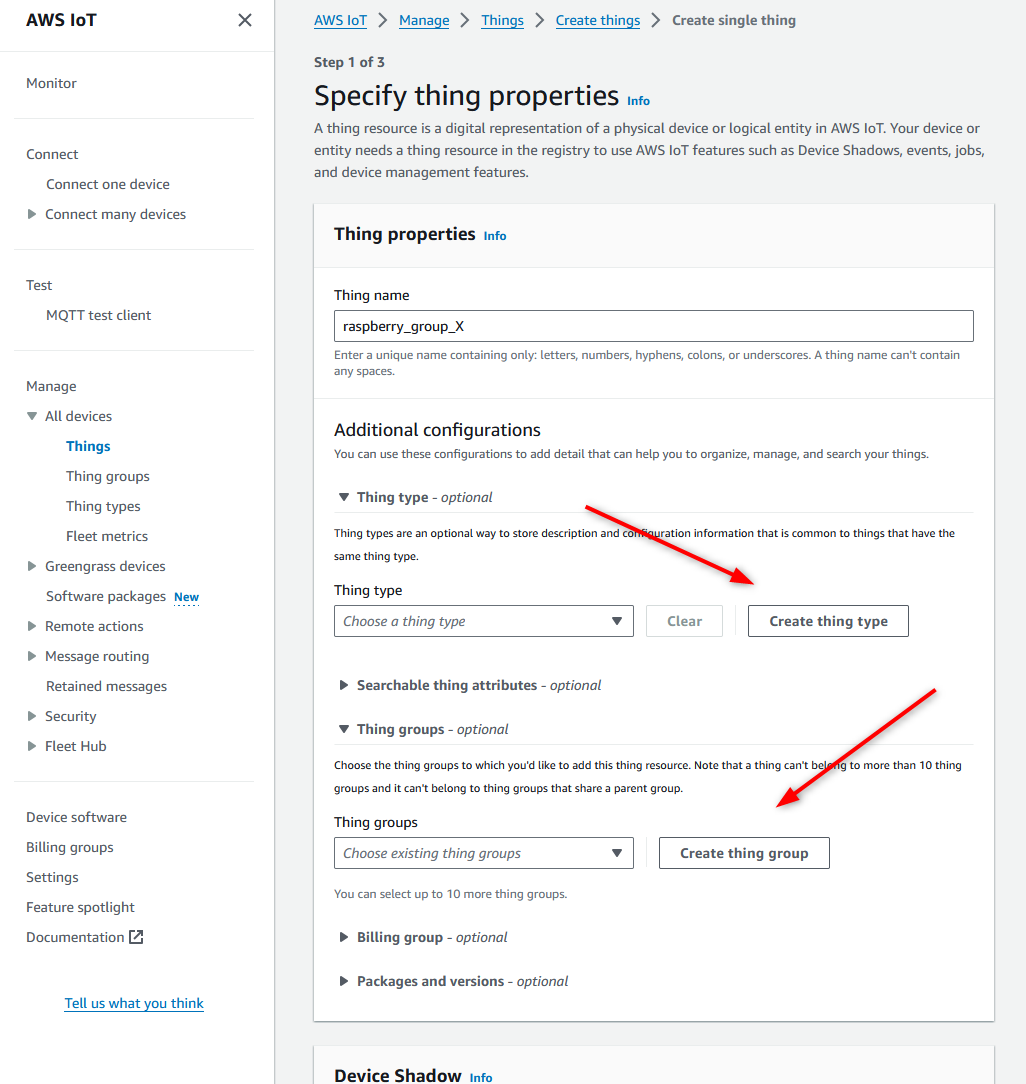
Una vez estemos en esta sección, vamos al botón Create things.



Dentro de esta sección, podemos elegir crear un dispositivo o varios. Elijamos la creación de un dispositivo, marcando la opción Create single thing.

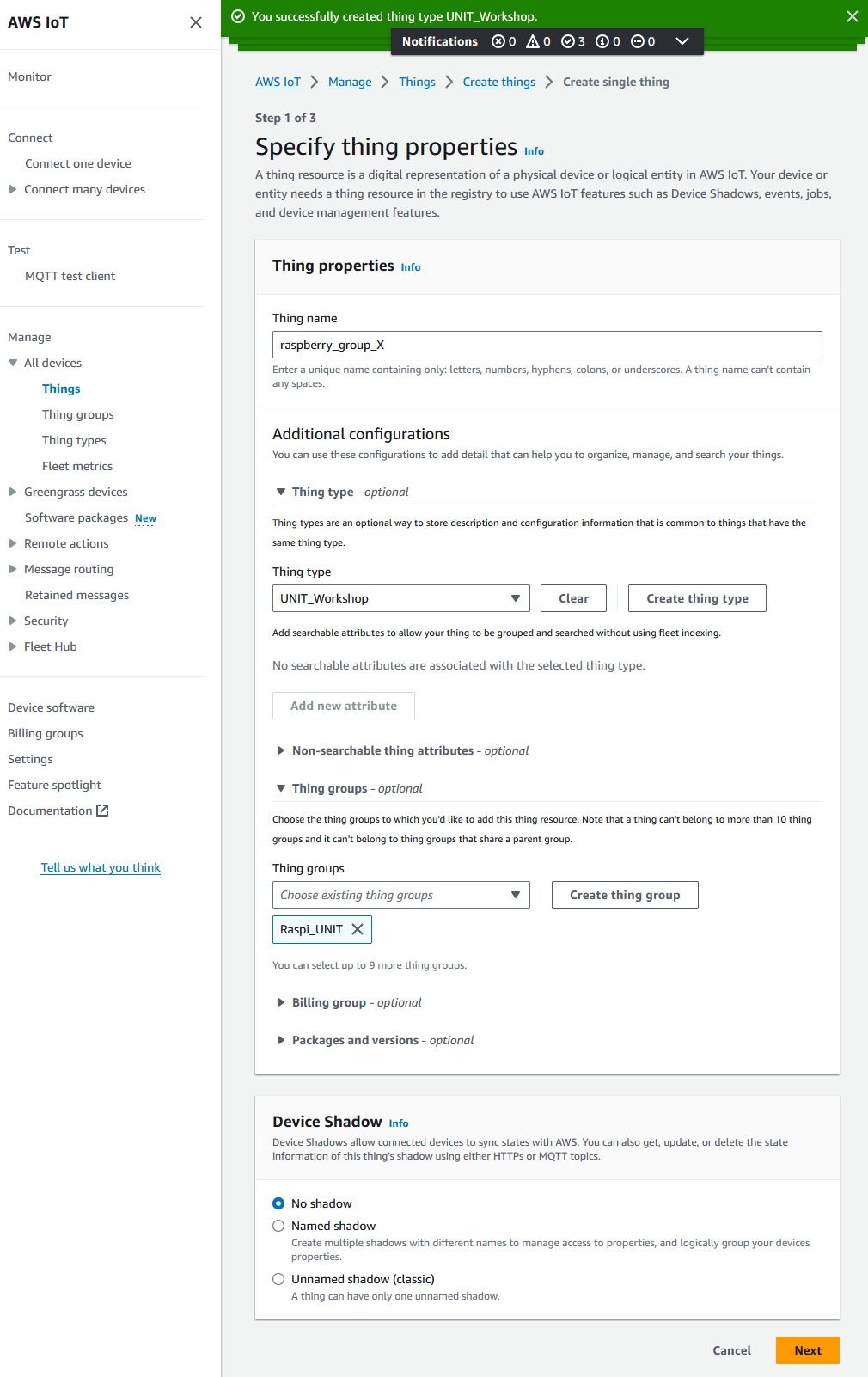
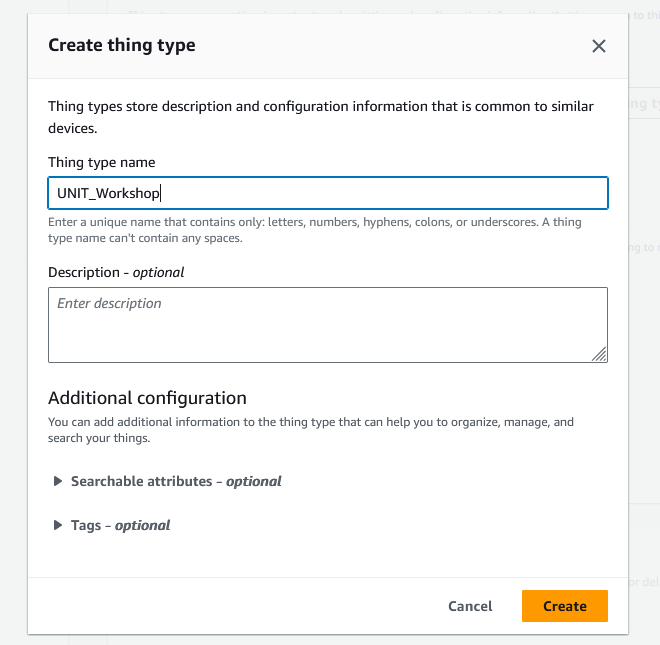


Dentro de la creación del dispositivo, le vamos a dar un nombre en Thing name,y opcionalmente podemos crear un tipo de dispositivo en Thing type, y un opcionalmente un nombre al grupo de dispositivos en Thing group. Esto es útil si vamos a tener más de un dispositivo, para que a futuro puedan ser gestionados de forma más eficiente.



La creación de un tipo de dispositivo, requiere incluir un nombre en Thing type name, y opcionalmente una descripción, la cual recomendamos incluir para colocar información adicional que pudiera ser de interés.

El procedimiento de creación de grupo de dispositivos en Create thing group es análogo. En esta práctica creamos el grupo Raspi\_UNIT.



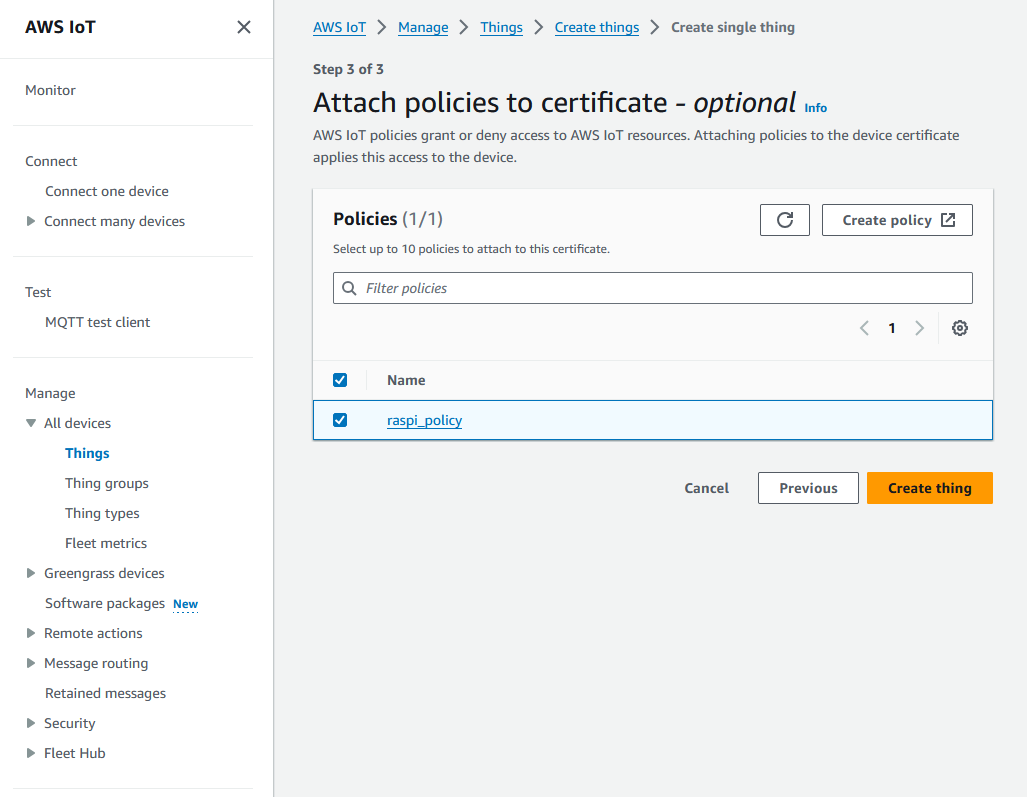
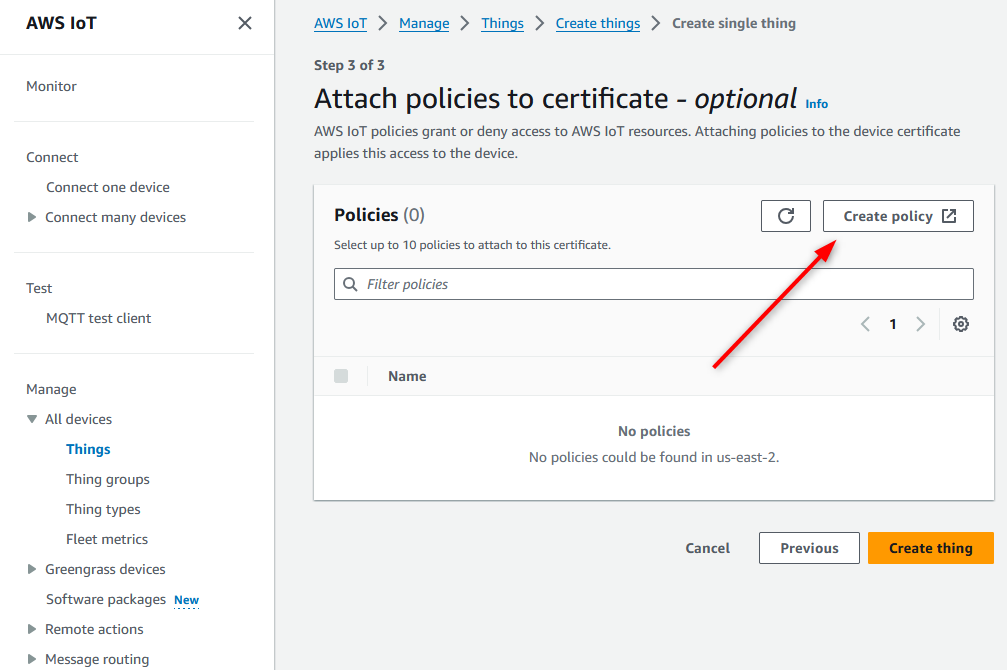
Un aspecto importante es asegurar el tráfico entre los dispositivos y AWS IoT. El tráfico viaja cifrado a través de HTTPS, y para que cada dispositivo pueda presentarse frente a AWS, podemos usar certificados. Esto permite que un tercero no autorizado que no dispone del certificado, no pueda consumir los servicios de AWS IoT.

Recomendamos que los certificados sean generados automáticamente por AWS y firmados por su autoridad certificadora.

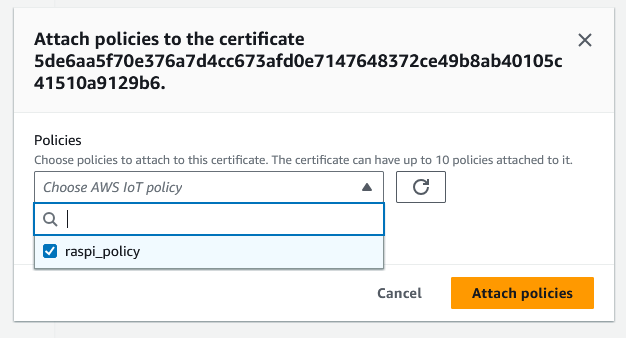
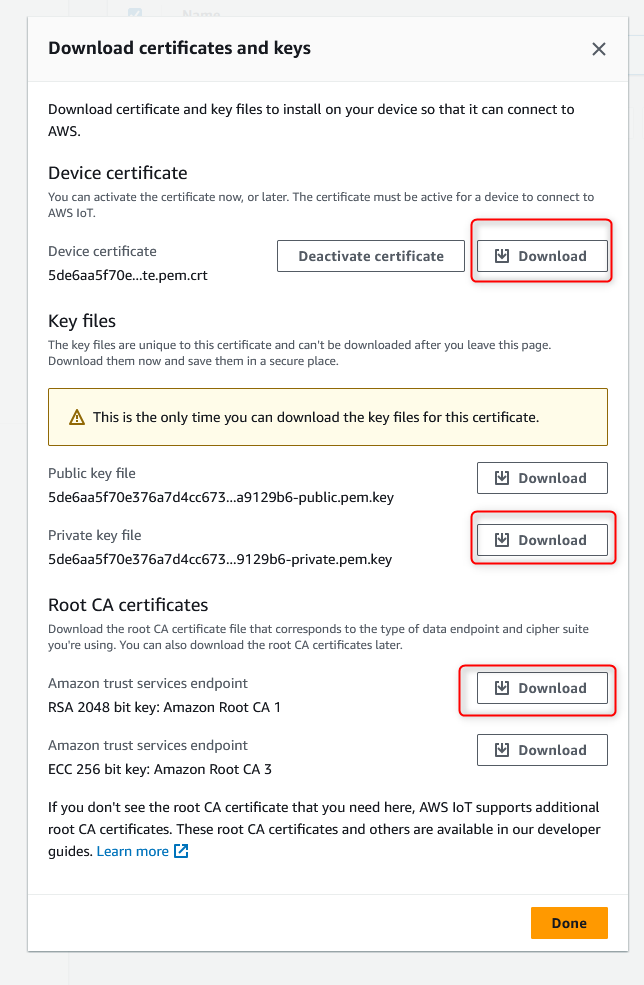


Cada certificado puede tener asociada políticas, las que se utilizan para autorizar o negar el acceso a recursos o funcionalidades de AWS. Procederemos a crear una política de ejemplo que permita cualquier acción, la cual es recomendable que sea refinada a las acciones o recursos requeridos, siguiendo la lógica del mínimo privilegio.

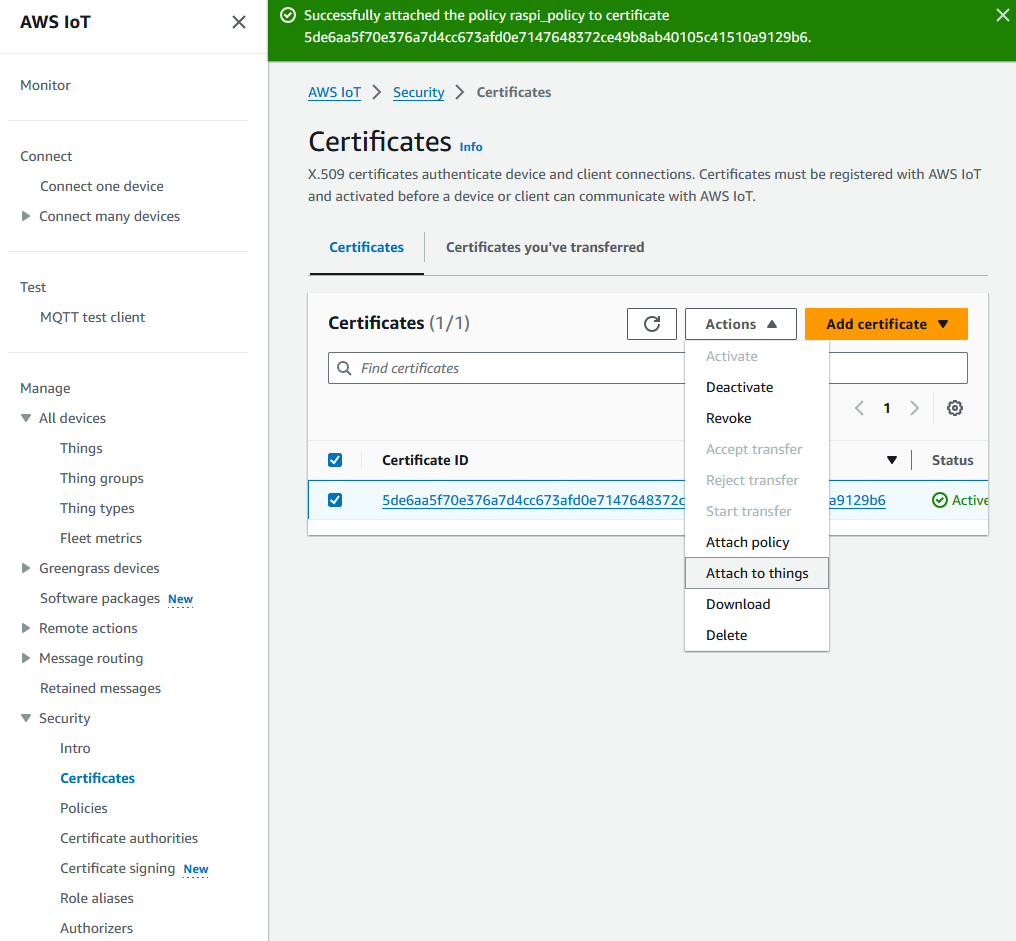
Esta política tiene un nombre, raspi\_policy en este caso, se crea una política que permite o niega (allow/deny) una acción (policy action) sobre un recurso (policy resource).



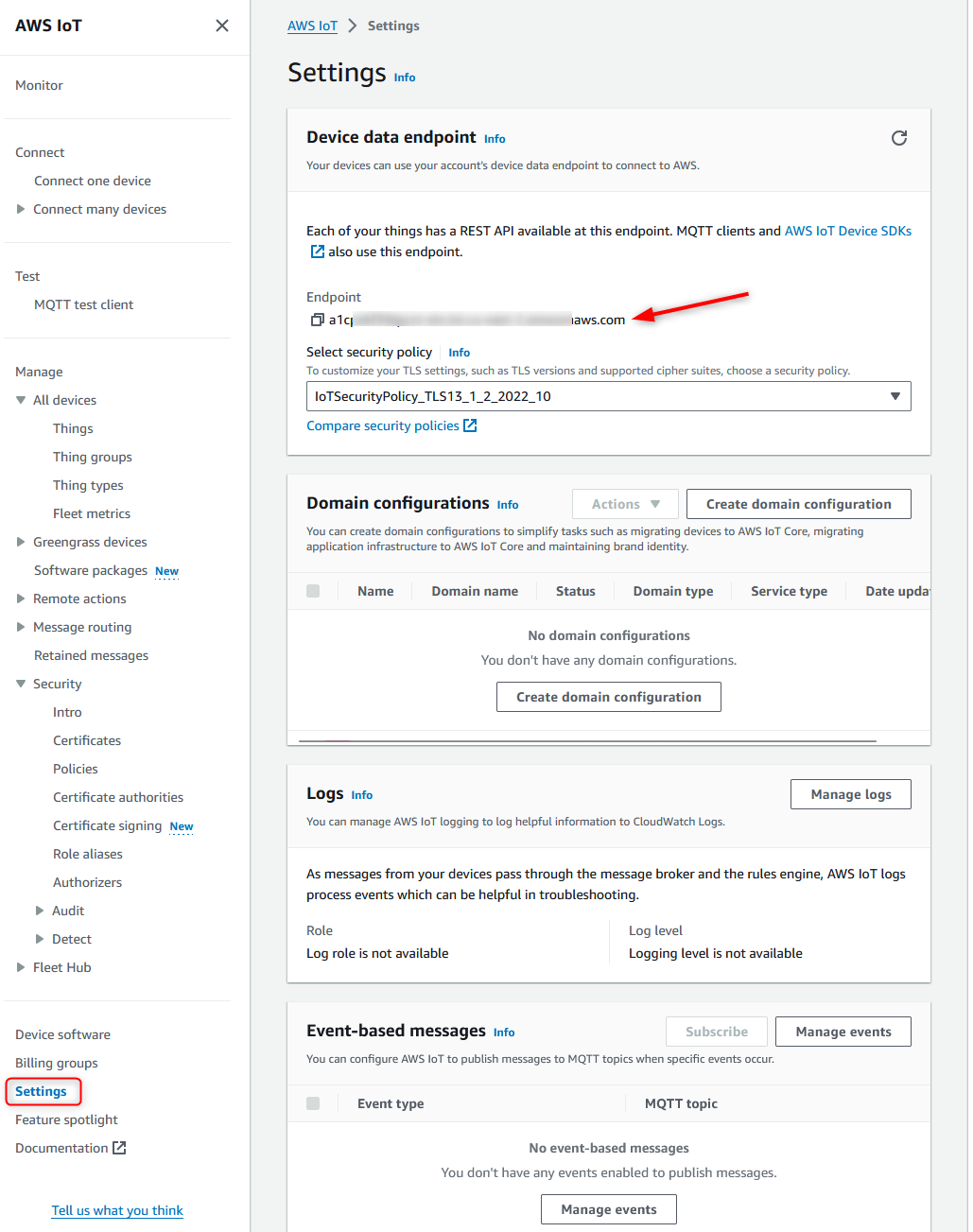
Una vez generada la policy y asignada al certificado, se pueden descargar las claves públicas y privadas del certificado emitido. Adicionalmente, se deberá descargar la clave pública de la certificadora raíz que emitió el certificado. En este caso, se encuentran marcados en rojo los archivos que deberán descargarse, los que se usarán luego en el script Python que correrá en Raspberry Pi.



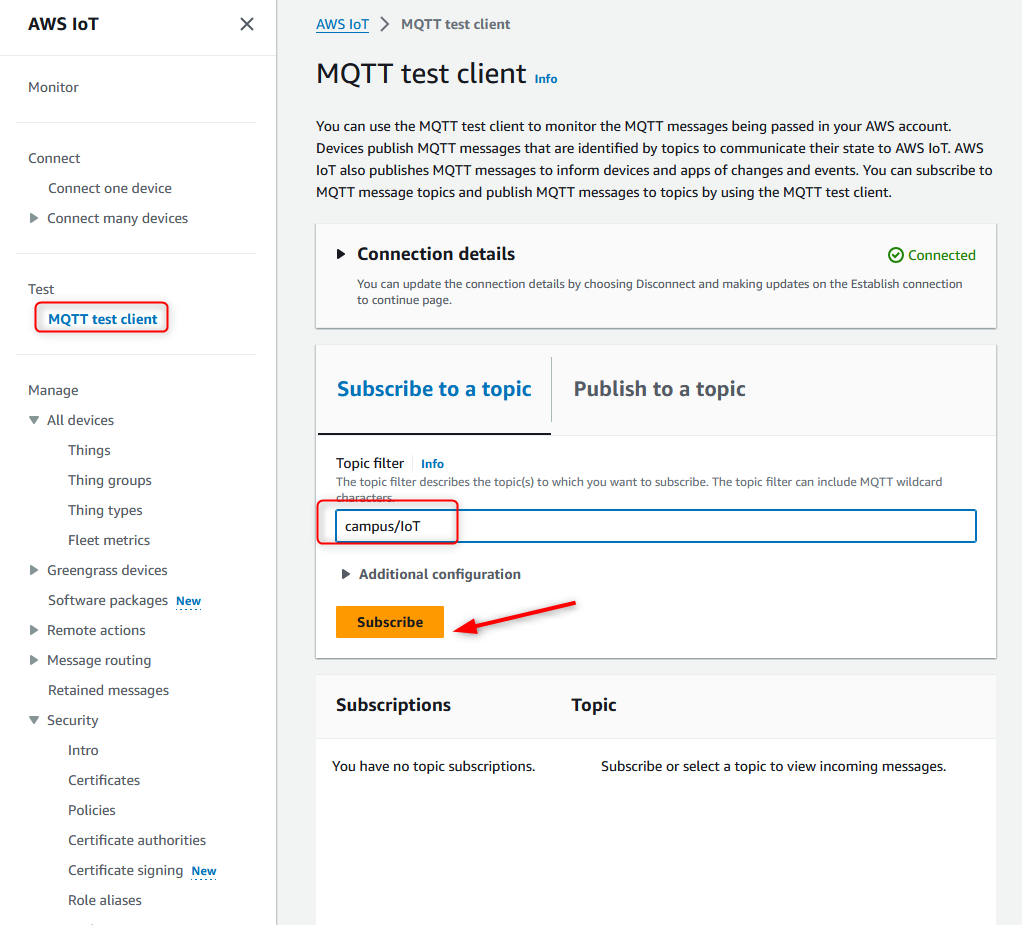
Una vez contamos con el certificado emitido, procedemos a adjuntarlo al dispositivo para que pueda autenticarse con él. Para ello, en Security – Certificates, seleccionamos el certificado a usar, y en Actions vamos a Attach to things, seleccionando el dispositivo que queramos asignar.



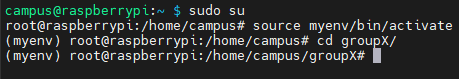
Una vez que el dispositivo ya tenga asignado su certificado, en Settings procederemos a tomar nota de algunos elementos importantes, tales como la dirección del endpoint y la política de seguridad. El endpoint es la dirección del servicio que atiende en AWS y con el que debemos comunicarnos usando los certificados que hemos creado y los dispositivos a los que se los hemos asignado. La política de seguridad especifica los protocolos permitidos en cuanto a cifrado del canal de comunicación, los cuales deberán ser soportados por los dispositivos que interactúen con AWS.



Hagamos una pequeña prueba usando el cliente MQTT integrado, escribiéndonos a un tópico de mensajería al cual enviaremos mensajes desde la Raspberry. En este esquema de prueba, podemos suscribirnos a un tópico y leer los mensajes enviados, o bien publicar a un tópico que leerá el dispositivo y actuará en consecuencia. En este caso, nos suscribimos al tópico “campus/IoT” para determinar que la mensajería enviada desde la placa Raspberry Pi es recibida correctamente en AWS.



Vayamos a la Raspberry a través de SSH. Como primer comando, debemos activar el entorno que vamos a utilizar para ejecutar la SDK de AWS. Para ello, escalamos a privilegios de root con sudo su, y por otro lado ejecutamos el comando source con la ruta hacia el entorno creado, en este caso myenv/bin/activate. Si el comando se ejecutó correctamente, se verá el prompt (myenv) delante del cursor.



Procedemos a copiar los tres certificados necesarios para interactuar con AWS, deberían copiarse al menos estos tres archivos al sistema de archivos de la Raspberry:

* Un archivo que contiene “-private” en su nombre, que es la clave privada
* Un archivo que contiene “-certificate” en su nombre, que es el certificado público
* Un archivo que contiene “root” en su nombre, que es la autoridad certificadora raíz que firmó el certificado.

Estos archivos serán invocados desde el script Python, que deberán modificar en las siguientes secciones:

**endpoint:** debe contener el endpoint que se obtuvo desde la sección Settings de AWS IoT Core

**cert\_filepath:** debe contener la ruta completa al archivo “-certificate”, con su extensión

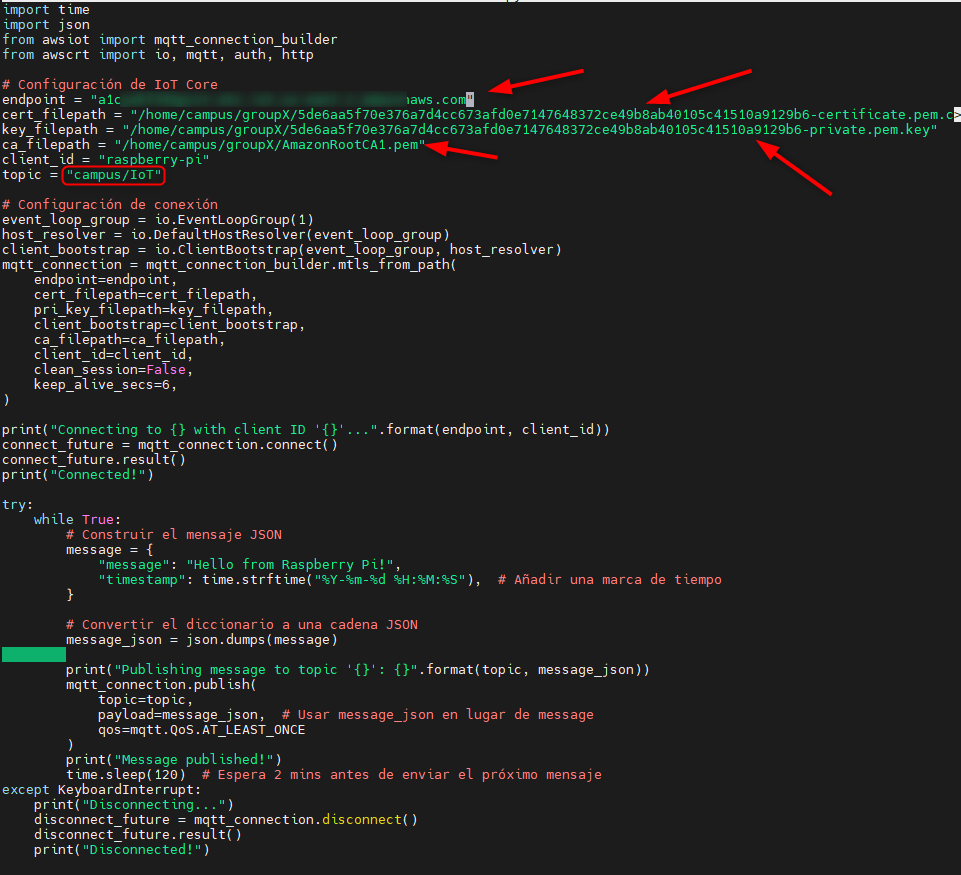
**key\_filepath:** debe contener la ruta completa al archivo “-private”, con su extensión

**ca\_filepath:** debe contener la ruta completa al archivo “root”, con su extensión

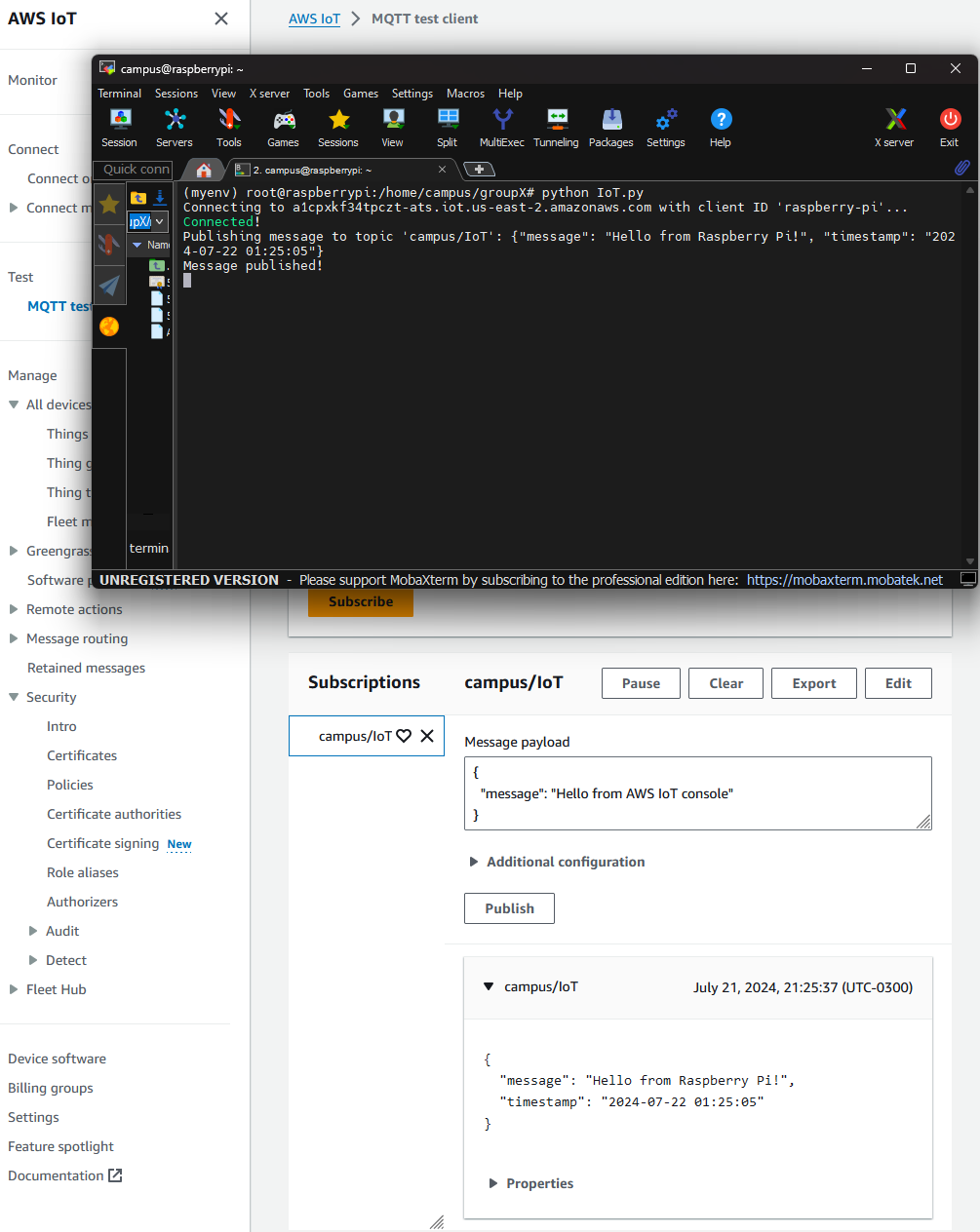
**client\_id:** debe contener el nombre del cliente, es libre y descriptivo

**topic:** debe contener el nombre del tópico que se utilizó en el MQTT test client

Si desearan modificar el mensaje que envía el script, se modifica la sección “message” con el contenido deseado. Por otro lado, el script espera 120 segundos entre mensaje y mensaje, se puede configurar



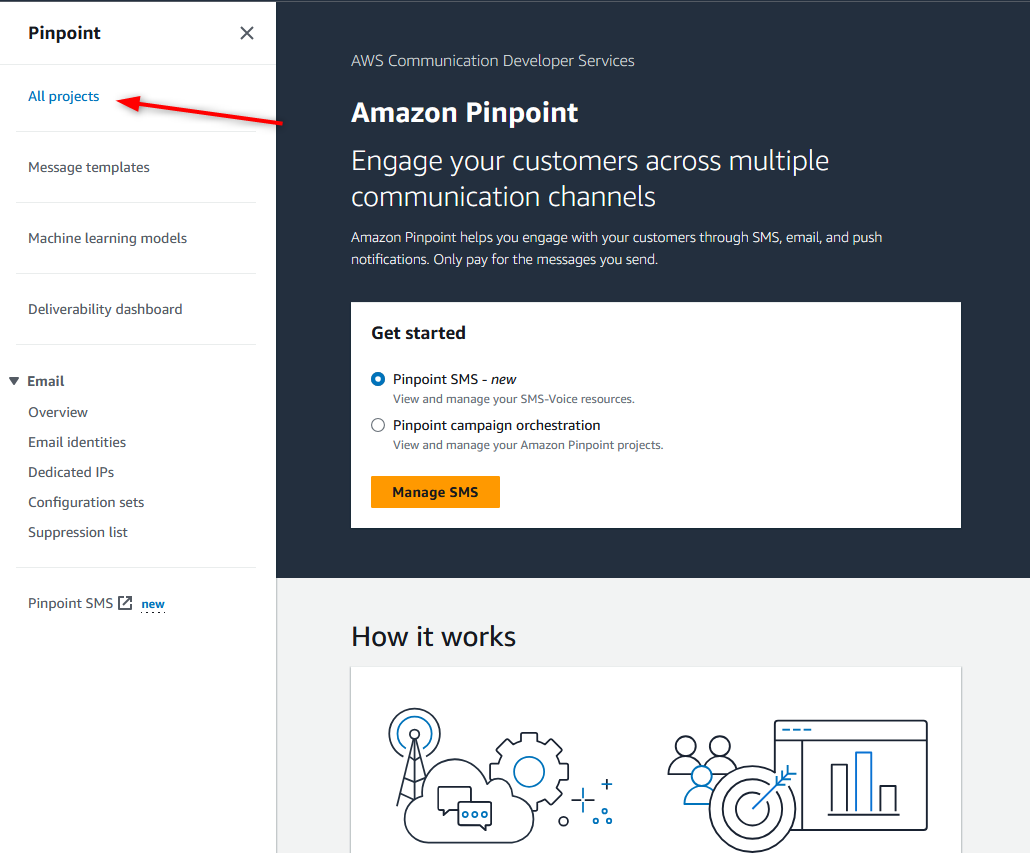
Probemos una ejecución, revisando lo que se vería la consola de AWS. Para ello, dentro del environment creado, ejecutamos el comando **python IoT.py**:



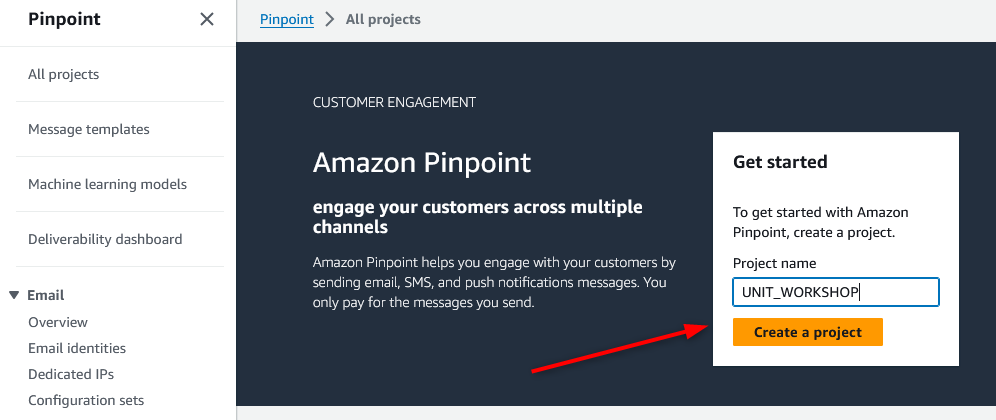
Podemos corroborar en la parte inferior que el mensaje fue recibido correctamente.

Si quisiéramos interactuar con otros servicios de AWS, podemos integrar IoT Core con Lambda y con Pinpoint para el envío de SMS. Veamos un ejemplo de configuración de Pinpoint para el envío de mensajes de texto, el que invocamos luego. Cabe destacar que Pinpoint no está disponible en todas las regiones de AWS.

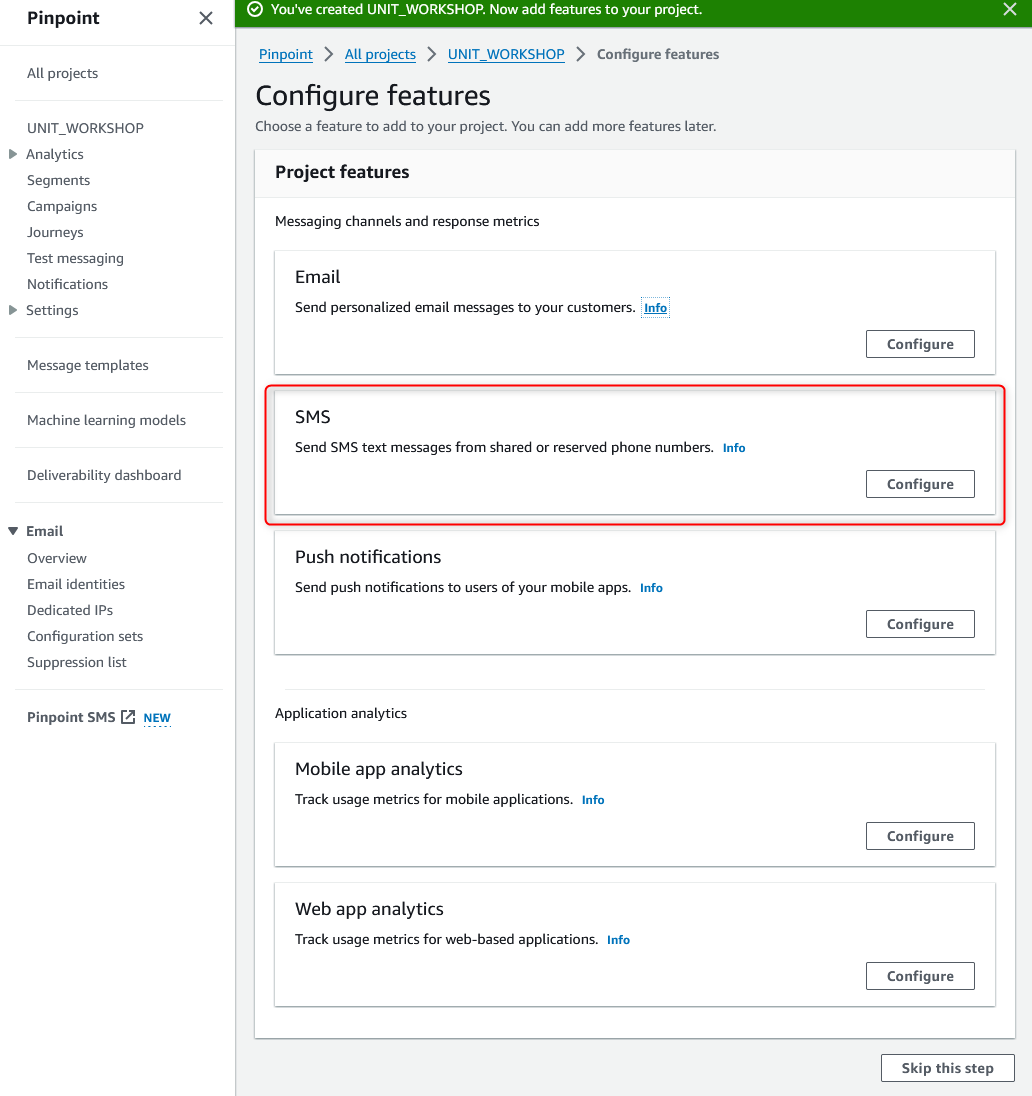
Desde la consola web de AWS, vamos a Pinpoint, y vamos a All projects.



Creamos un proyecto, al que le pondremos el nombre que deseemos, en este caso UNIT\_WORKSHOP.

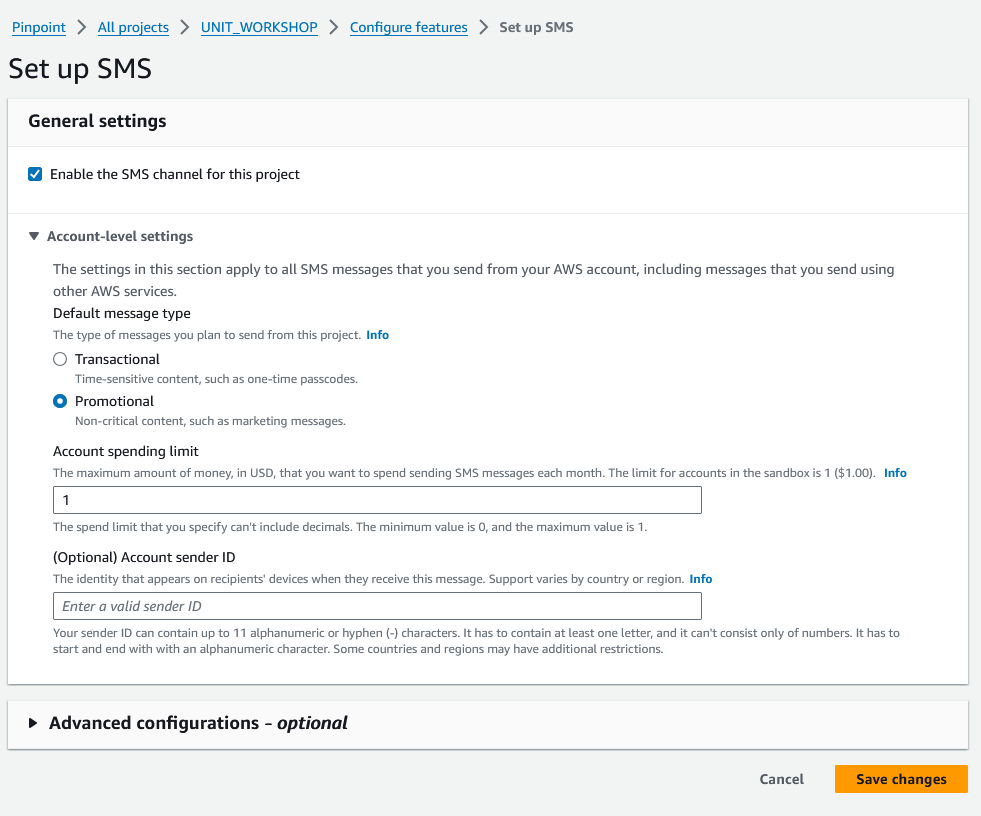


Cada proyecto tiene capacidades u opciones. En nuestro caso, seleccionaremos SMS, si bien con Pinpoint también podemos manejar el envío de emails o notificaciones push.

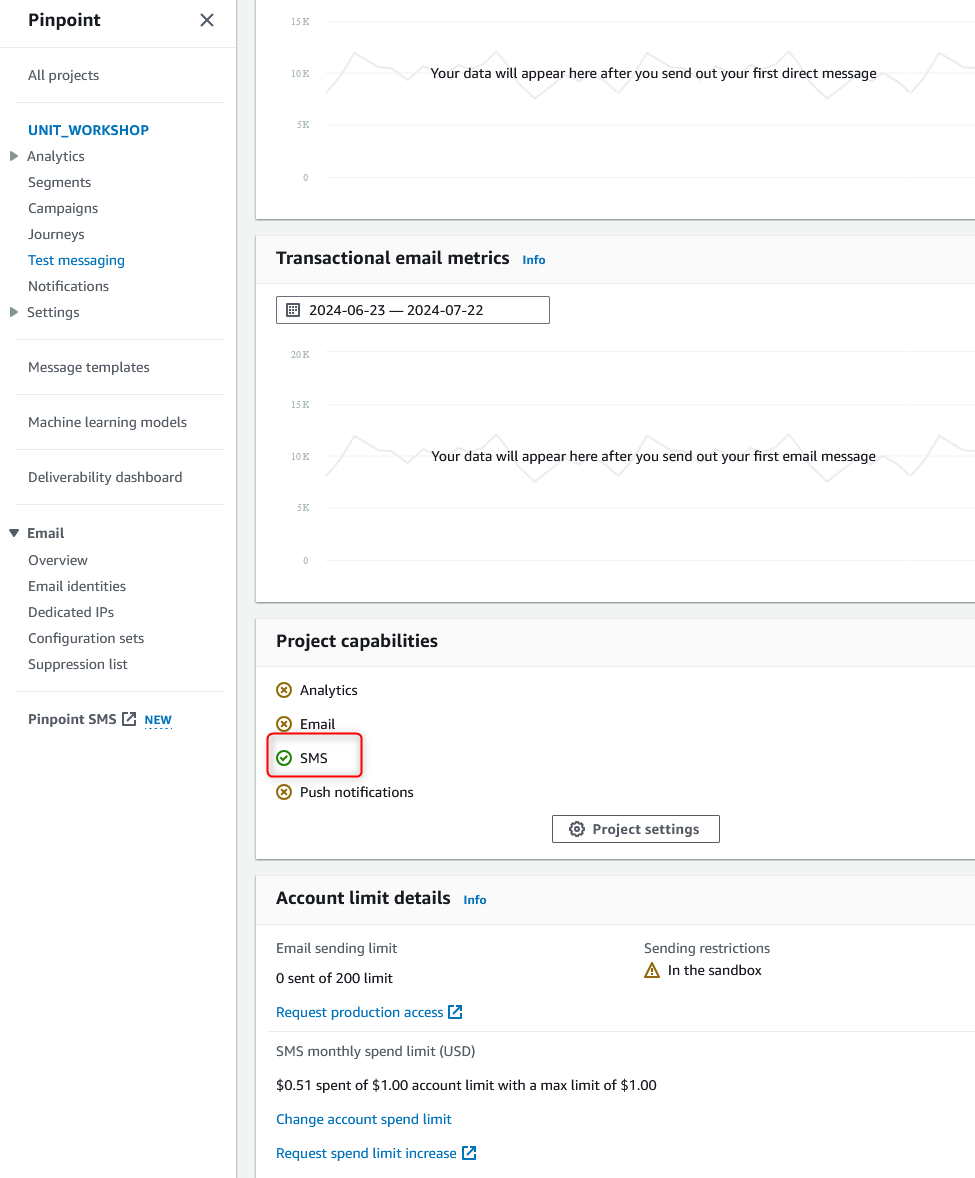


Aquí veamos algunos aspectos importantes, por ejemplo el tipo de mensaje (Default message type). Los mensajes transaccionales (transactional) son usados para enviar contenidos que tienen vencimiento, por ejemplo en códigos de segundo factor o elementos que deben ser enviados con prioridad al tener una duración breve. Los mensajes de tipo promocional (promotional) tienen una prioridad menor, y son los que usaremos en nuestro caso.

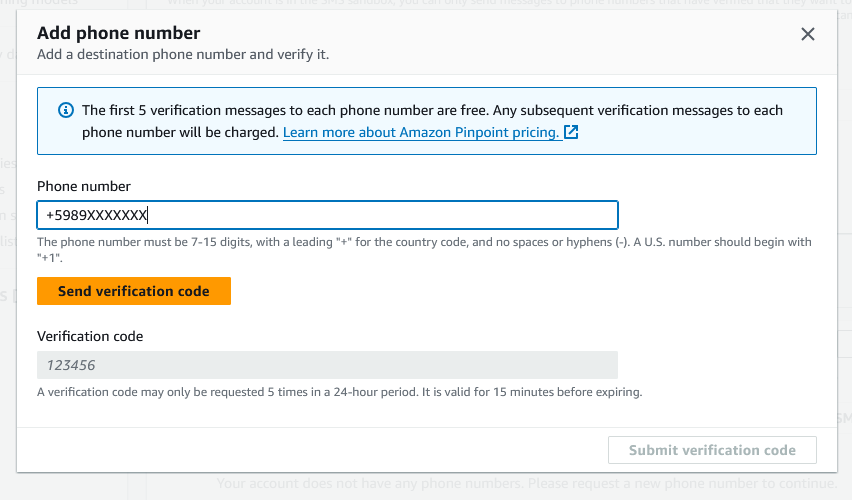
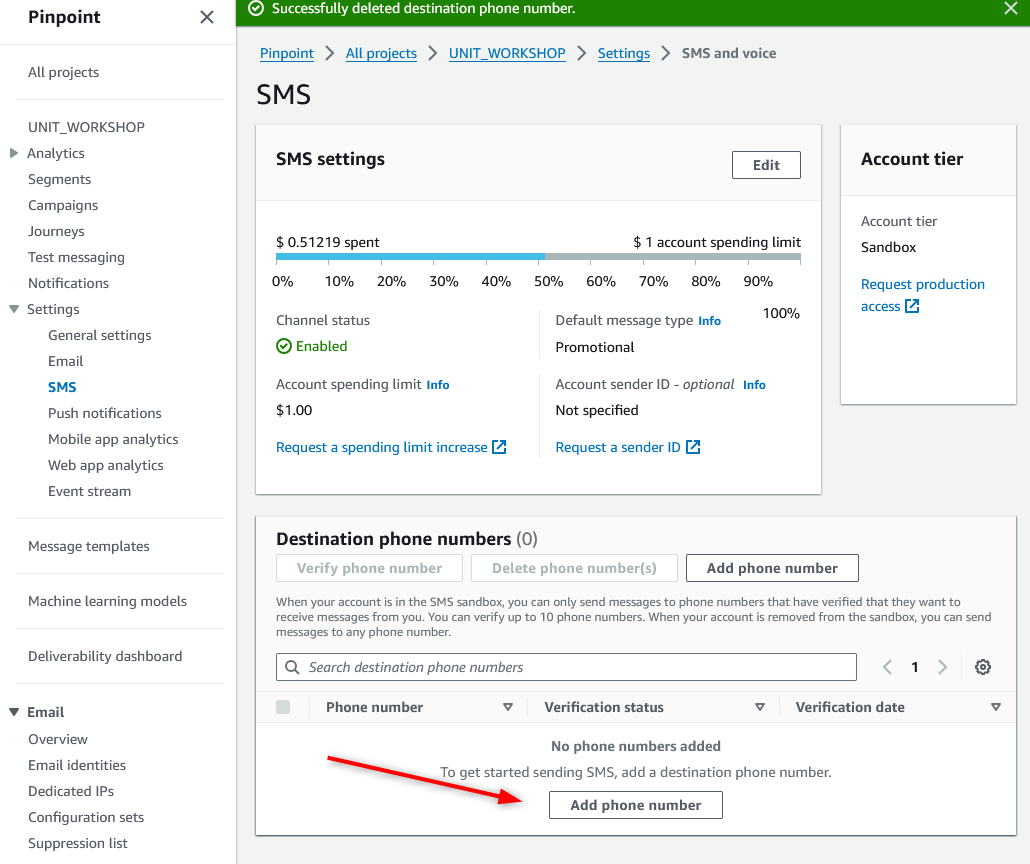
El valor **account spending limit** es el límite de USD que se coloca para presupuesto de SMS, nosotros en este ejemplo lo colocamos en 1 USD.



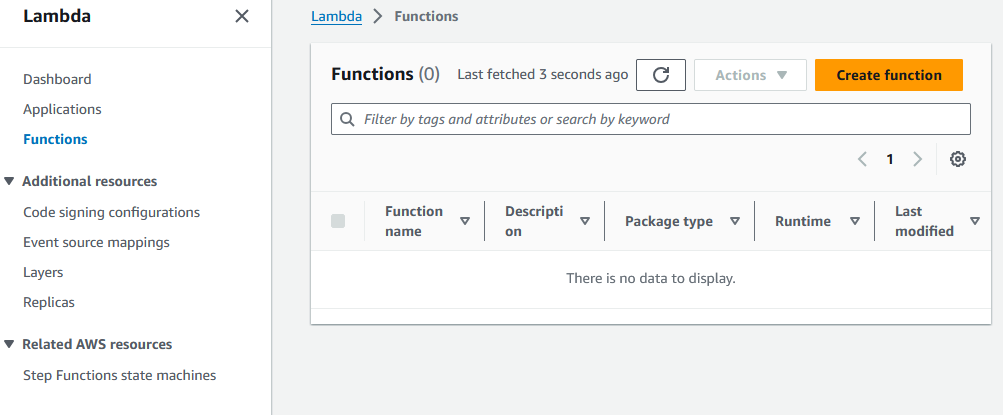
Una vez presionado el botón **Save changes**, verán que el canal SMS del proyecto en **Test Messaging** se marca en verde.



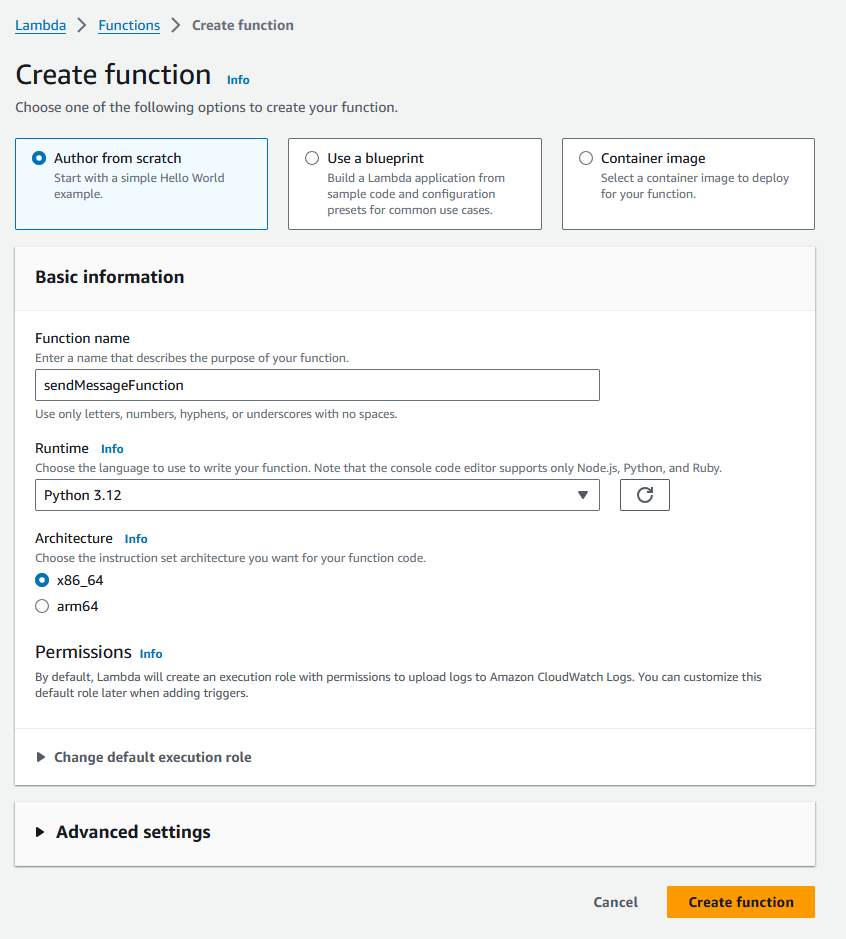
Hasta ahora no le indicamos a qué número se deben enviar las notificaciones, por lo que debemos ir a la sección **Settings – SMS** para agregarlo. Elegimos la opción **Add phone number**, que nos permite ingresarlo y a su vez se envía un código de validación para verificar que el número es correcto, el que debe ser ingresado en una ventana de 15 minutos como máximo.



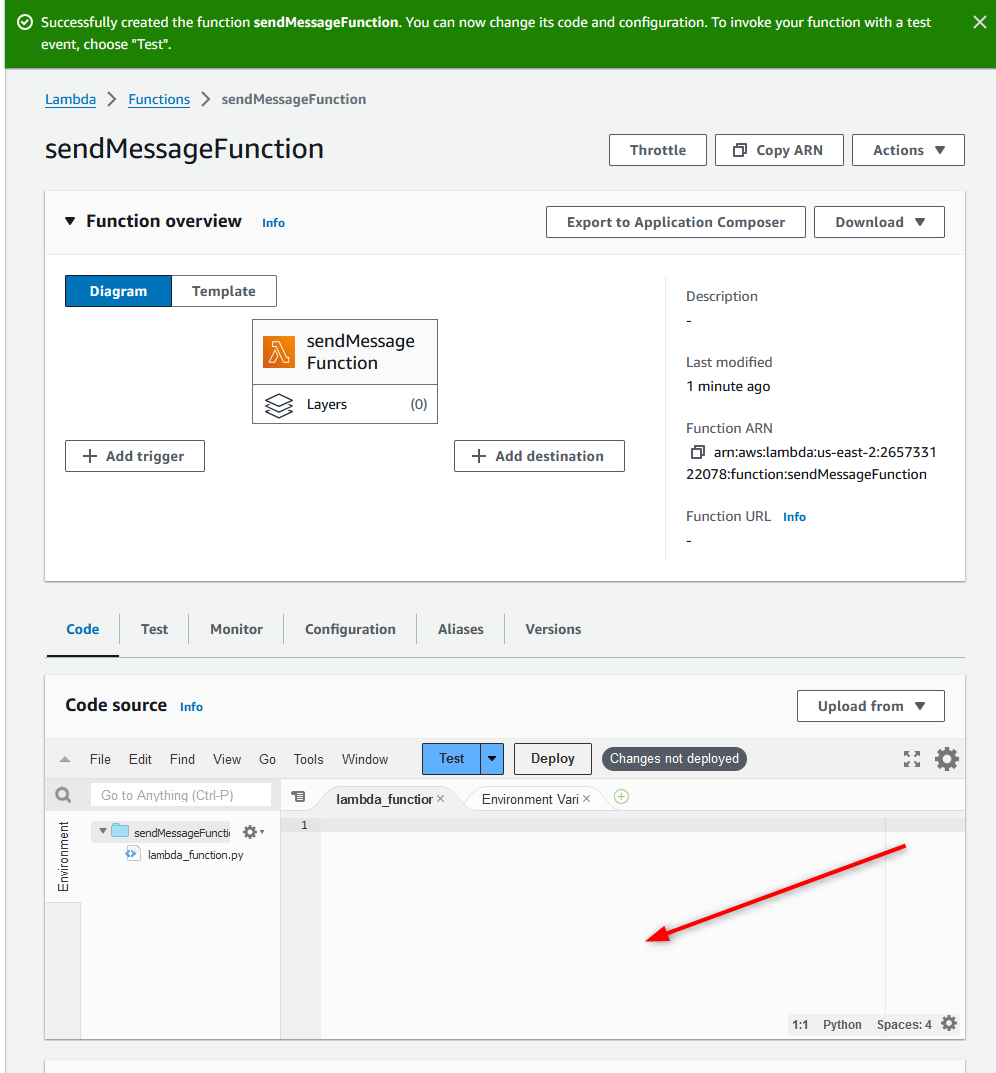
Una vez validado, procedemos a ir a Lambda, para crear una función que una a IoT Core y a Pinpoint. Dentro de Lambda, elegimos la opción **Create function**.



Vamos a utilizar una función desde cero, marcando **Author from scratch**, en Python 3.12, y el nombre será **sendMessageFunction** con arquitectura x86\_64. Una vez finalizado, elegimos **Create function.**



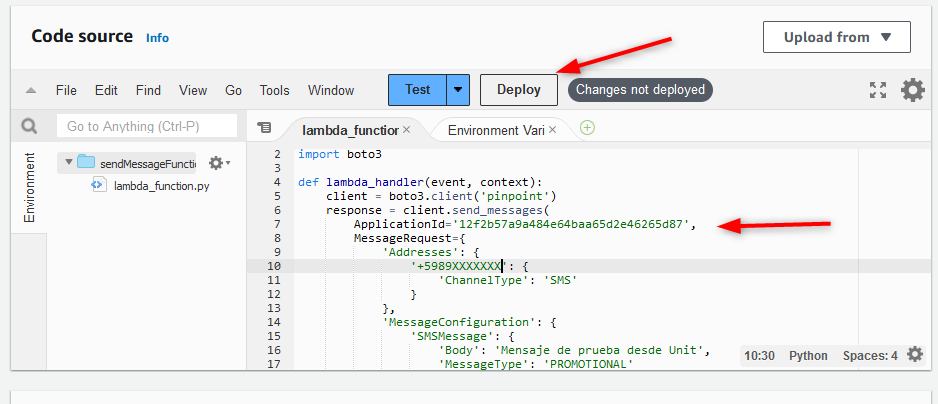
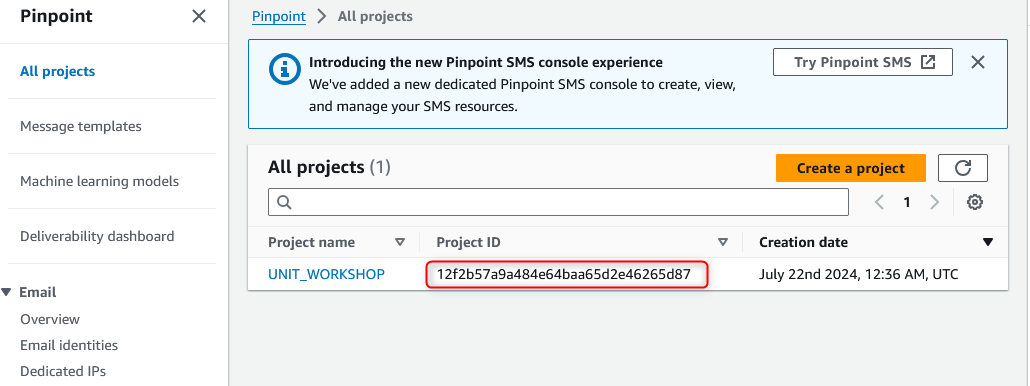
Listo, la función está creada. Ahora en la sección **Code source**, procedemos a copiar el código de ejemplo, al que le haremos algunos cambios:

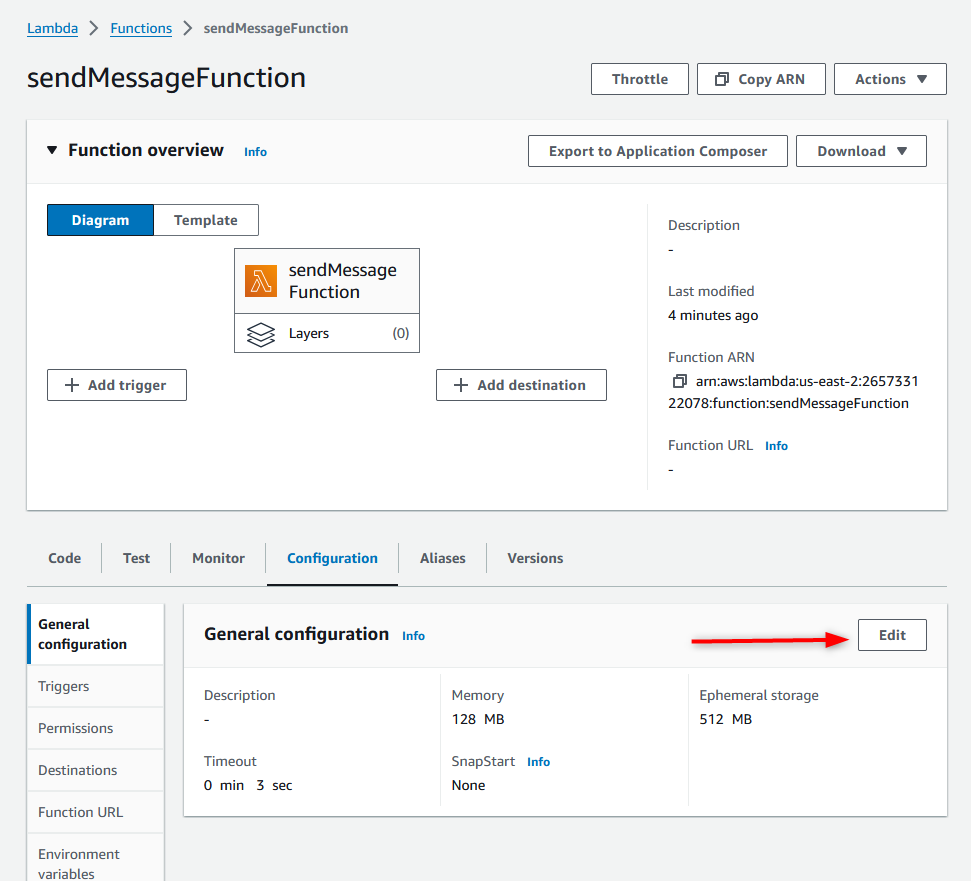


El resumen de modificaciones que deberemos hacer es el siguiente:

-En ApplicationId, debemos indicar el valor del Project ID que es visible en el proyecto de Pinpoint que hemos creado.

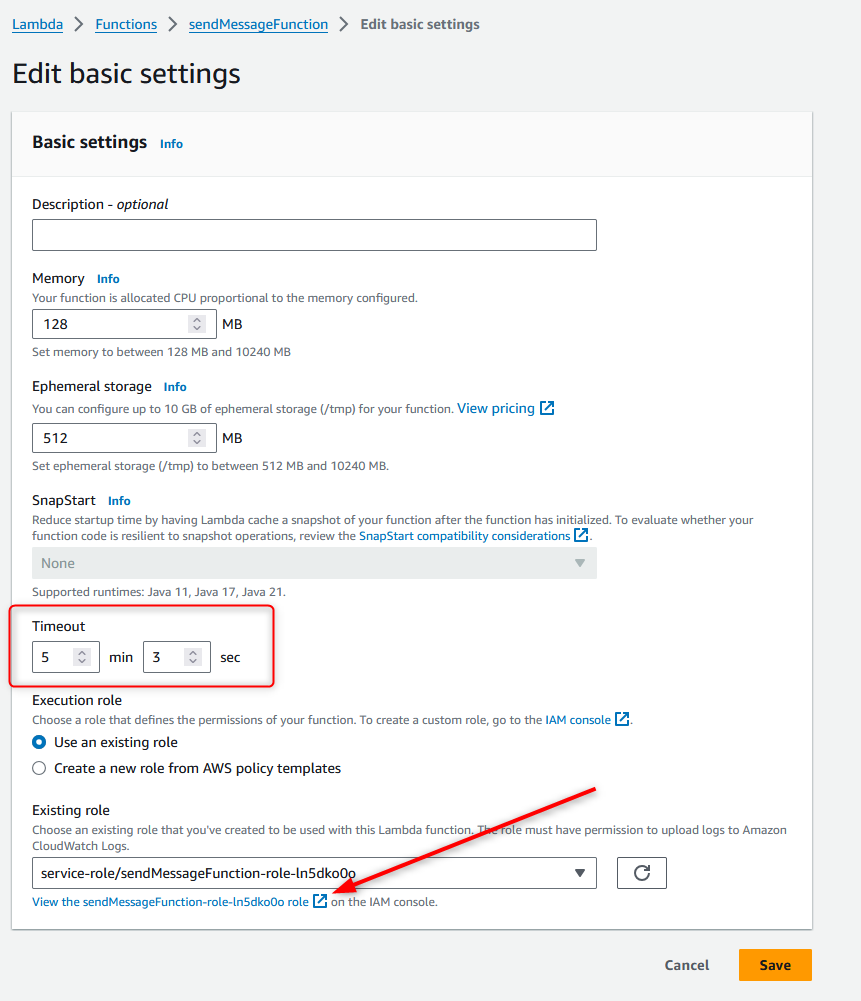
* Dentro de **Addresses**, debemos indicar el número de teléfono que hemos registrado y validado en Pinpoint.
* En **Body** indicamos el contenido del mensaje a enviar.
* En **MessageType** indicamos el tipo de mensaje que configuramos en Pinpoint, ya sea PROMOTIONAL o TRANSACTIONAL.



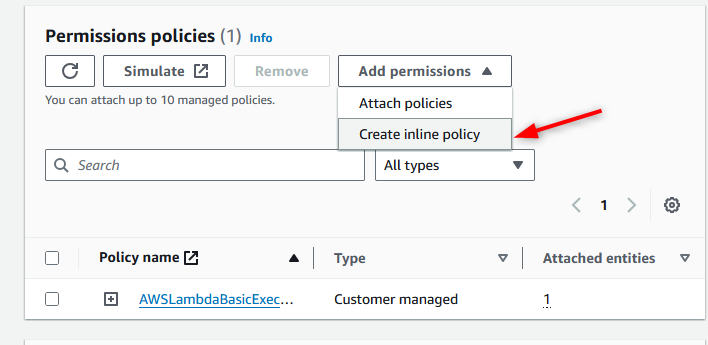
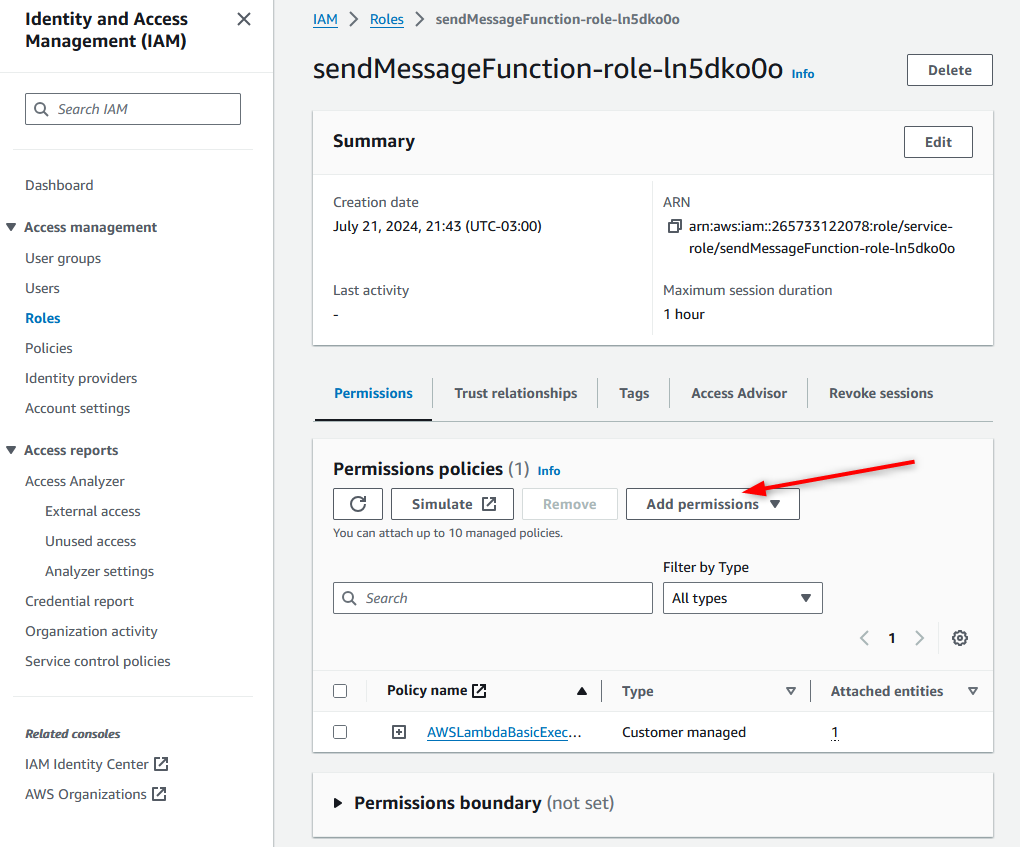
Una vez creada la función y cargado el código, vamos a tener que editarla para darle permisos sobre Pinpoint al rol de seguridad que la va a ejecutar. Para ello, en **Configuration**, vamos a **Edit** dentro de General configuration.

Resulta recomendable colocar un tiempo de ejecución máximo (timeout) en la función lambda, para limitar los costos y establecer un límite a su ejecución.

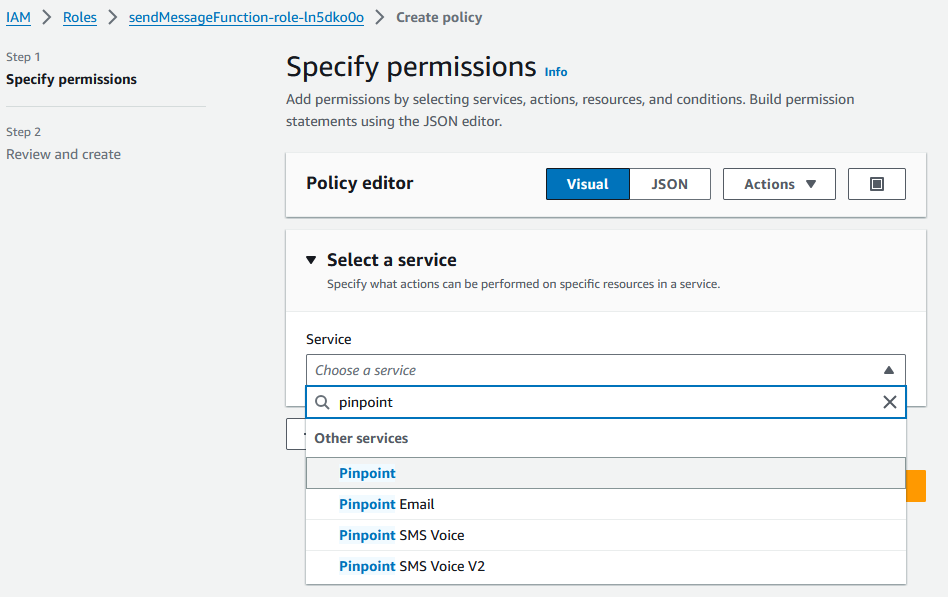
Para revisar el rol de seguridad, existe un enlace al final de la página que permite ver el rol en la consola de IAM. Vamos a proceder a hacer clic sobre él.



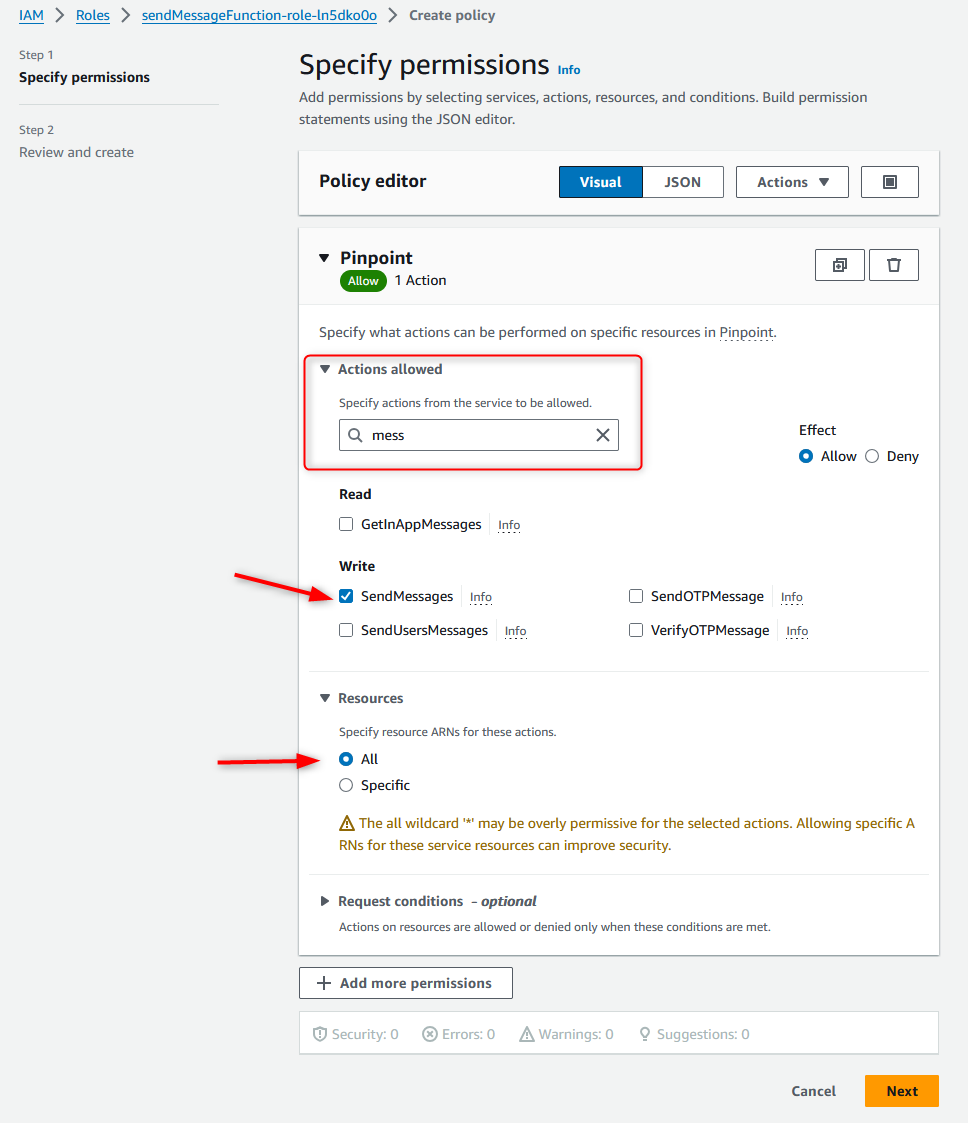
Dentro del rol, vamos a ir a la sección **Permissions policies** y vamos a agregar permisos con el botón **Add permissions** y la opción **Create inline policy.**



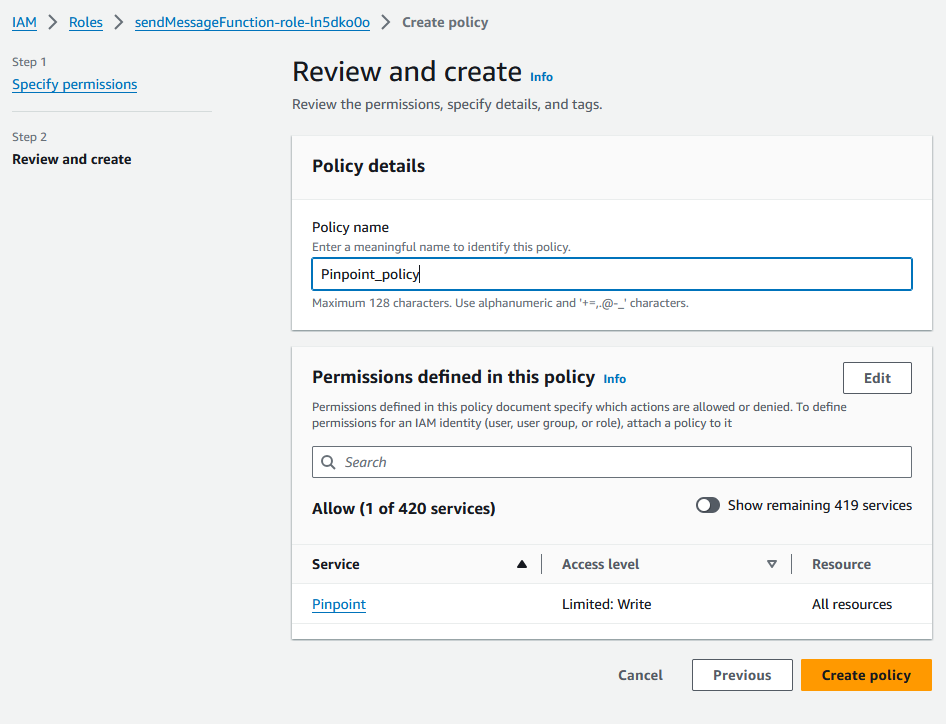
Dentro de los servicios, elegimos Pinpoint. Una vez elegido, avanzamos y debemos elegir la acción que vamos a permitir.



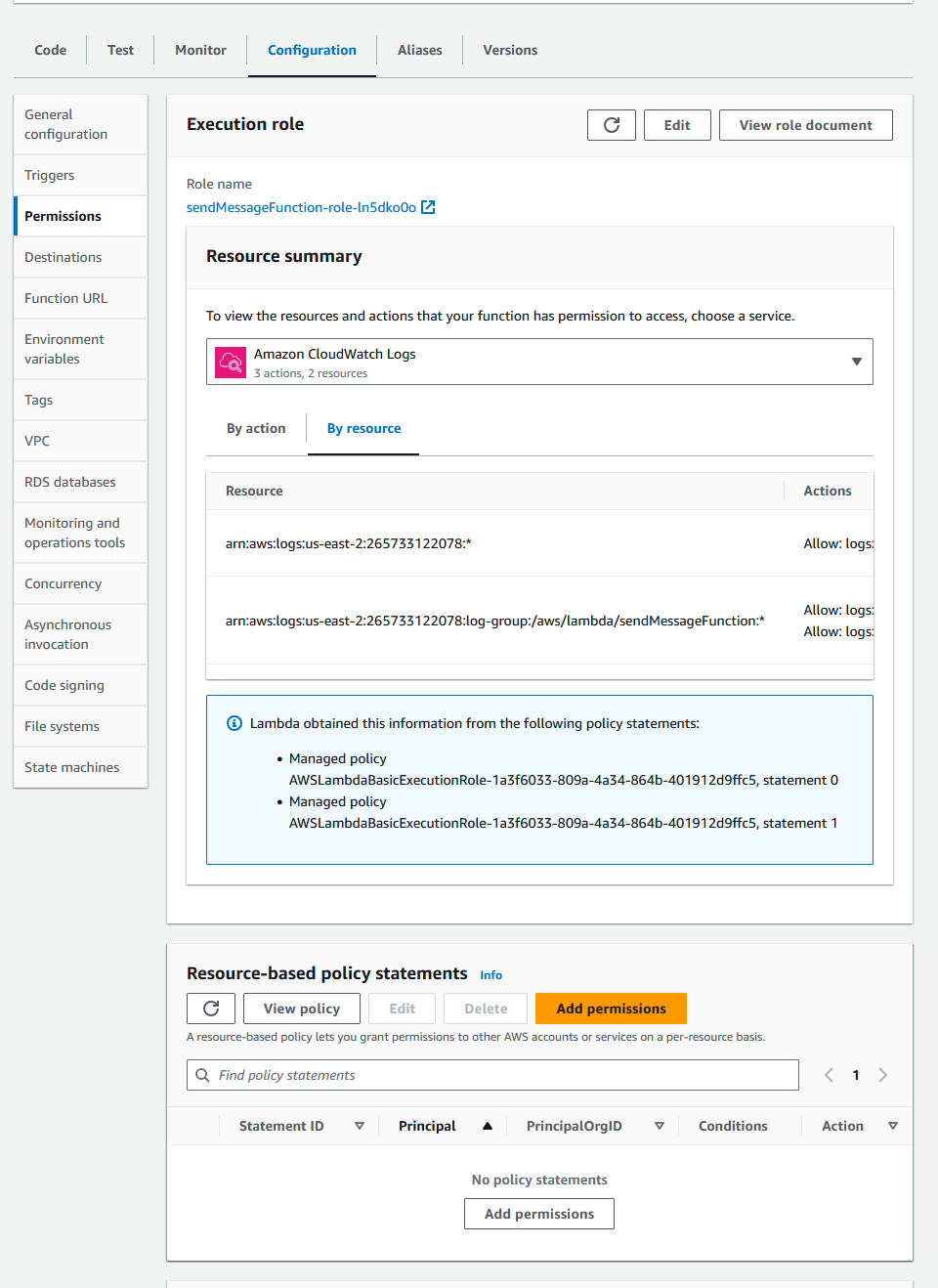
La acción que vamos a permitir es **SendMessages**, sobre todos los recursos, seleccionando en **Resources** el valor **All.** Si quisiéramos especificar recursos puntuales, en este lugar se indican los recursos autorizados sobre los cuales la política tendrá efecto.

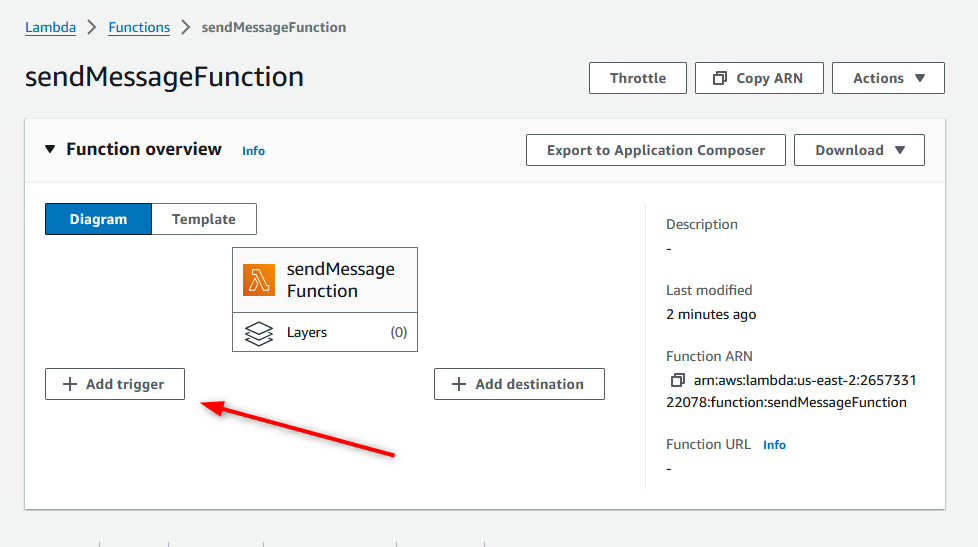


Le damos un nombre a la política, en este caso Pinpoint\_policy, y la creamos. Vemos en el resumen que el permiso es limitado sobre Pinpoint solamente.



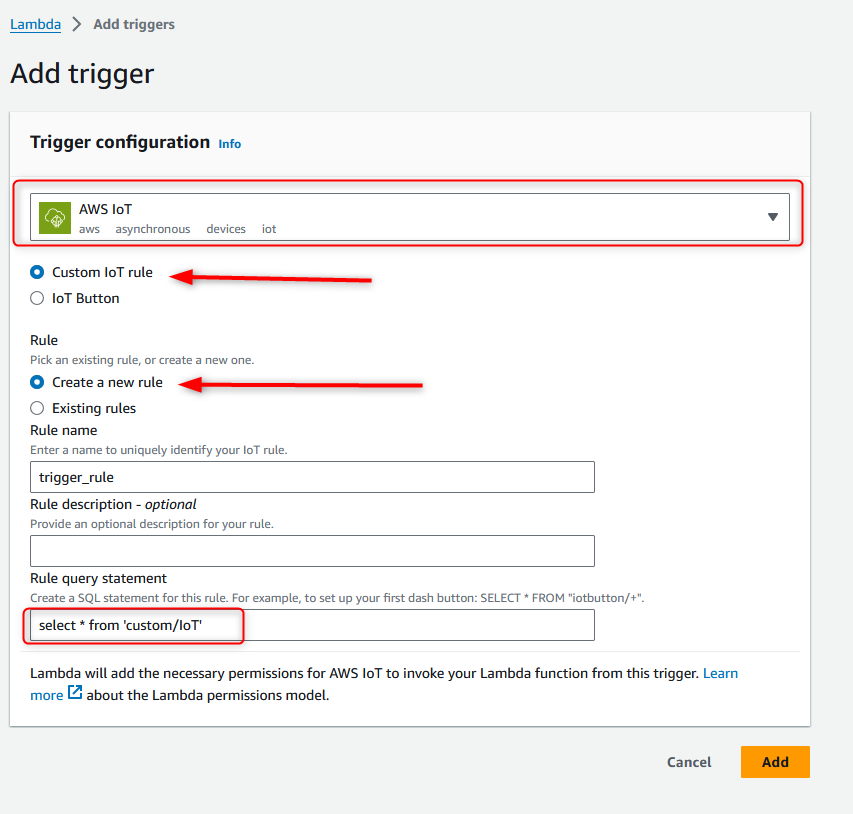
Validamos en la sección Permissions, en la pestaña By Resource, que se muestra la función sendMessage.



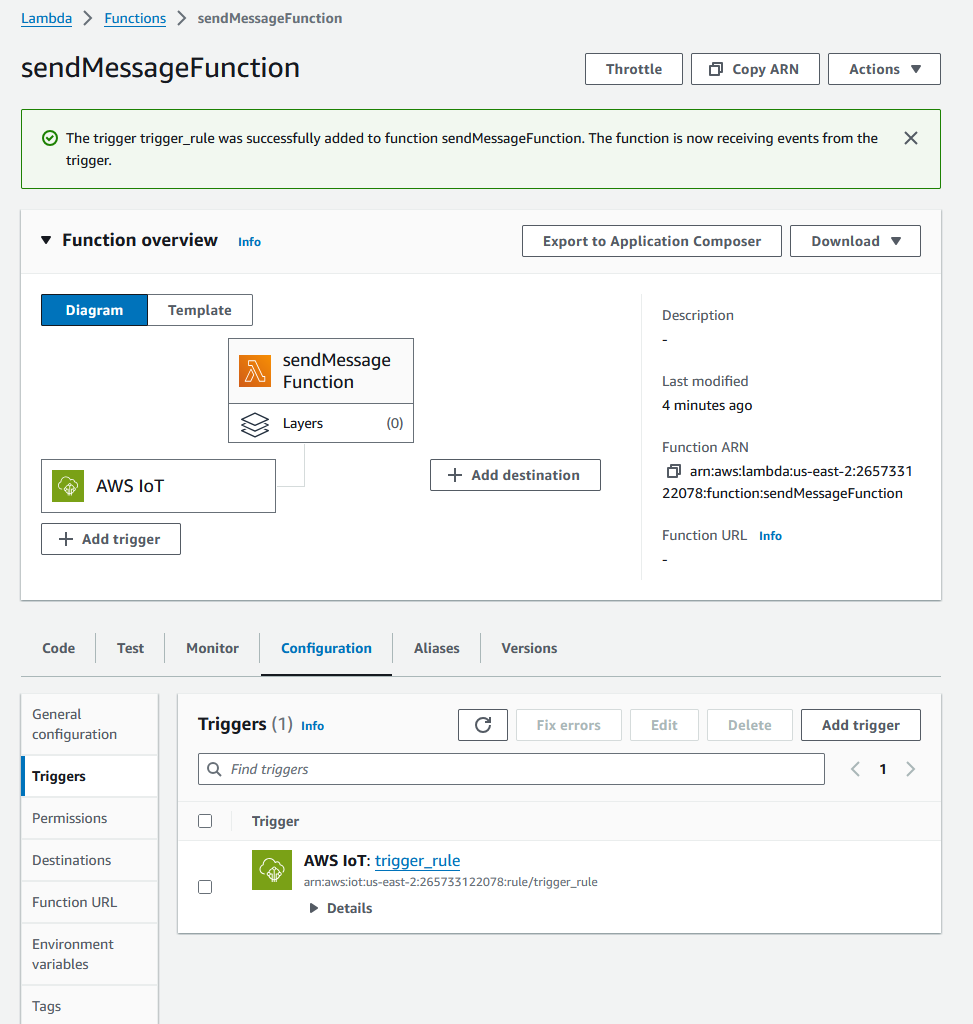
Ahora procedemos a agregar la condición sobre la que se ejecutará la función Lambda, o sea, el **Trigger**. Presionamos sobre el botón **Add trigger**.

Seleccionamos AWS IoT, y creamos una nueva regla personalizada a la que le damos nombre, en este caso **trigger rule**, y le indicamos una consulta SQL para ejecutar. En este caso, queremos que se ejecute ante la recepción de cualquier mensaje en cualquier tópico, por lo que agregamos el texto **select \* from ‘custom/IoT’**

Una vez finalizado, seleccionamos el botón **Add.**



Podemos validar que el trigger ha sido creado, y podemos verificarlo en la pestaña **Configuration**, dentro de la sección **Triggers**.



Listo. A partir de ahora, cada vez que se reciba un mensaje en AWS IoT, o sea, cada vez que el sensor envíe mensajes de proximidad mediante protocolo MQTT a AWS IoT, se ejecutará la función Lambda que enviará un mensaje al celular vía SMS usando Pinpoint. Este ejemplo puede tomarse como base para el desarrollo de otros scripts que utilicen otros servicios, explotando las opciones de integración que AWS tiene disponibles.

**CÓDIGOS COMPLEMENTARIOS**

**Función Lambda:**

import json

import boto3

def lambda\_handler(event, context):

client = boto3.client('pinpoint')

response = client.send\_messages(

ApplicationId='735--------------------7231',

MessageRequest={

'Addresses': {

'+5989XXXXXXX': {

'ChannelType': 'SMS'

}

},

'MessageConfiguration': {

'SMSMessage': {

'Body': 'Mensaje de prueba desde Unit',

'MessageType': 'PROMOTIONAL'

}

}

}

)

return {

'statusCode': 200,

'body': json.dumps(response)

}

**Script python - Lectura del sensor y comunicación AWS IoT**

import time

import json

from awsiot import mqtt\_connection\_builder

from awscrt import io, mqtt, auth, http

# Configuración de IoT Core

endpoint = "axxxxxxx-ats.iot.us-east-2.amazonaws.com"

cert\_filepath = "/home/campus/certs/690axxxxxxxxxx-certificate.pem.crt"

key\_filepath = "/home/campus/certs/690xxxxxxxxxxxx-private.pem.key"

ca\_filepath = "/home/campus/certs/AmazonRootCA1.pem"

client\_id = "raspberry-pi"

topic = "campus/IoT"

# Configuración de conexión

event\_loop\_group = io.EventLoopGroup(1)

host\_resolver = io.DefaultHostResolver(event\_loop\_group)

client\_bootstrap = io.ClientBootstrap(event\_loop\_group, host\_resolver)

mqtt\_connection = mqtt\_connection\_builder.mtls\_from\_path(

endpoint=endpoint,

cert\_filepath=cert\_filepath,

pri\_key\_filepath=key\_filepath,

client\_bootstrap=client\_bootstrap,

ca\_filepath=ca\_filepath,

client\_id=client\_id,

clean\_session=False,

keep\_alive\_secs=6,

)

print("Connecting to {} with client ID '{}'...".format(endpoint, client\_id))

connect\_future = mqtt\_connection.connect()

connect\_future.result()

print("Connected!")

try:

while True:

# Construir el mensaje JSON

message = {

"message": "Hello from Raspberry Pi!",

"timestamp": time.strftime("%Y-%m-%d %H:%M:%S"), # Añadir una marca de tiempo

}

# Convertir el diccionario a una cadena JSON

message\_json = json.dumps(message)

print("Publishing message to topic '{}': {}".format(topic, message\_json))

mqtt\_connection.publish(

topic=topic,

payload=message\_json, # Usar message\_json en lugar de message

qos=mqtt.QoS.AT\_LEAST\_ONCE

)

print("Message published!")

time.sleep(120) # Espera 2 mins antes de enviar el próximo mensaje

except KeyboardInterrupt:

print("Disconnecting...")

disconnect\_future = mqtt\_connection.disconnect()

disconnect\_future.result()

print("Disconnected!")