DESPLIEGUE DE UNA API REST MEDIANTE AMAZON API GATEWAY

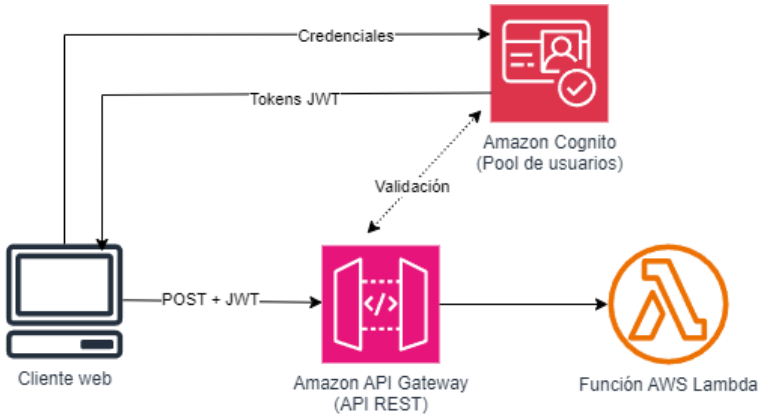
Amazon API Gateway es un servicio administrado que facilita la creación, publicación, mantenimiento, monitoreo y aseguramiento de APIs a cualquier escala. Permite a los desarrolladores crear APIs RESTful y WebSocket para aplicaciones que acceden a servicios de backend como AWS Lambda, Amazon EC2, u otros servicios accesibles por Internet.

Uno de los objetivos del módulo de "Desarrollo Web en Entorno Servidor" es el desarrollo de servicios web, por lo que Amazon API Gateway va a simplificar el proceso de implementación, mantenimiento, escalado y seguridad del servicio para los estudiantes.

Requerimientos:

- Disponer de acceso a los recursos de AWS a través de un *sandbox* de AWS Academy

Arquitectura propuesta:



Realización:

DESPLIEGUE DE LA INFRAESTRUCTURA NECESARIA

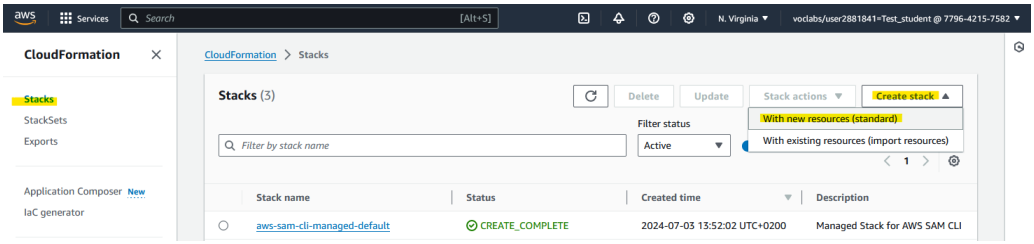
1) En primer lugar, hay que desplegar la función Lambda que implementa la lógica del backend. En este caso será una función llamada *calcularHipoteca* que, a partir de una serie de parámetros (*precio_vivienda*, *entrada_vivienda*, *tasa_diferencial* y *plazo_anos*) obtendrá la cuota mensual de una hipoteca.

Además, se desplegará un *user pool* de Amazon Cognito, que utilizaremos posteriormente para la autorización a nuestra API REST.

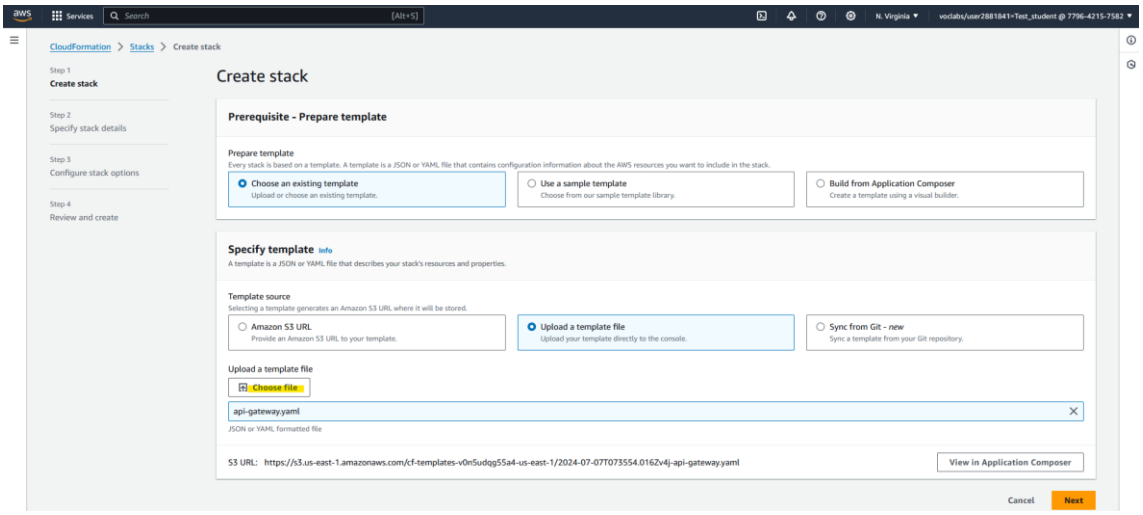
Para desplegar la infraestructura, lo haremos de forma automática mediante el servicio de AWS CloudFormation. Para ello, descargaremos la plantilla desde el siguiente enlace:

https://raw.githubusercontent.com/jose-emilio/aws-academy-fp-daw/main/resources/api-gateway/api-gateway.yaml

2) A continuación, abrimos la consola del servicio de AWS CloudFormation y presionamos en el menú lateral la opción **Stacks** y desde el botón **Create stack** seleccionamos la opción **With new resources (standard)**.

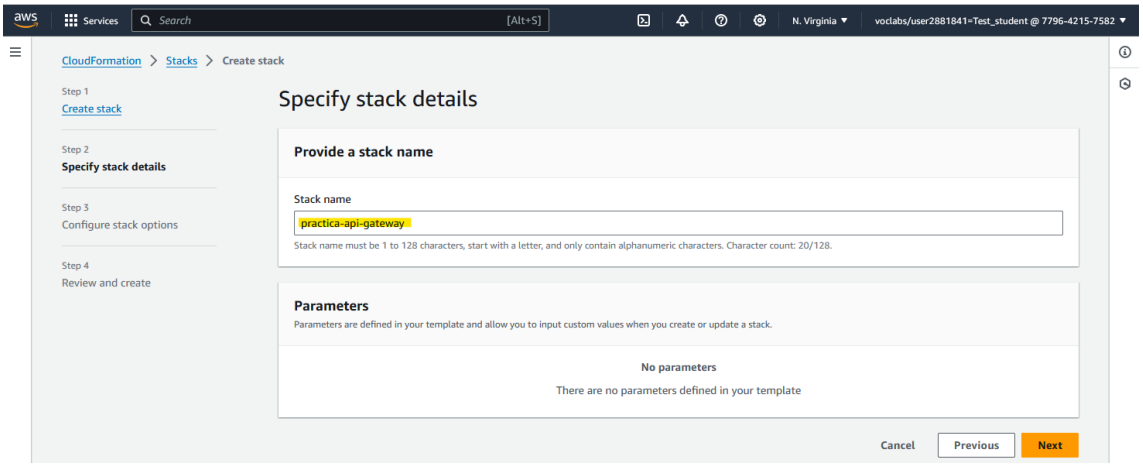


3) Desde el asistente siguiente, en la sección **Prerequisite – Prepare template** se elige la opción **Choose an existing template**. En el apartado **Specify template**, se elige la opción **Upload a template file** y se presiona el botón **Choose file** para elegir el archivo *api-gateway.yaml* descargado en el apartado **1)**:



Por último, se presiona el botón **Next**

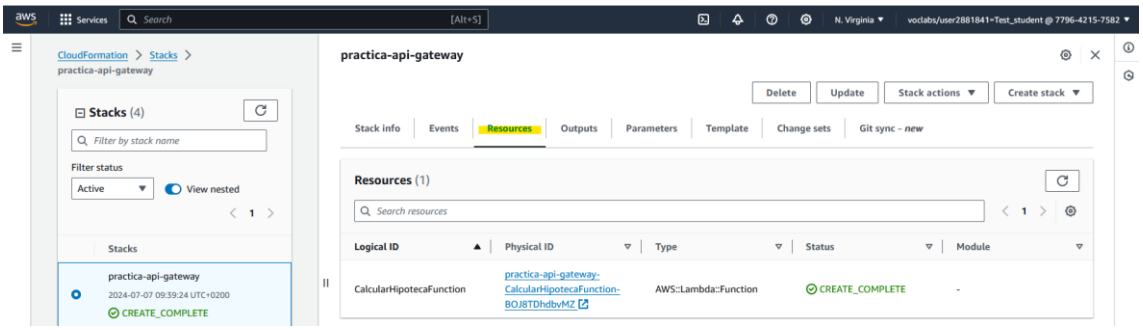
4) En el paso 2 del asistente, introducimos el nombre de la pila de recursos que creará AWS CloudFormation. Indicamos en el campo **Stack name** el valor *practica-api-gateway* y presionamos el botón **Next**:



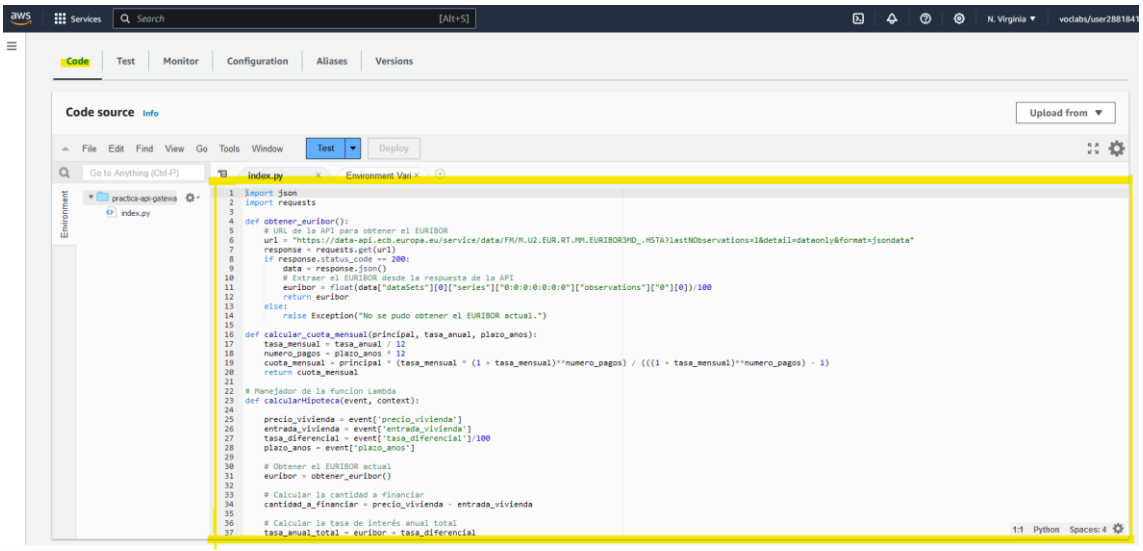
José Emilio Vera

5) En el paso 3 del asistente, dejamos los valores por defecto y presionamos el botón **Next**.

6) En el paso final, presionamos el botón **Submit** que se encuentra al final de la página. Tras unos pocos segundos, la plantilla ya estará creada. Presionamos la pestaña **Resources** y podremos visualizar la función Lambda que se ha creado como parte del proceso:



7) Si presionamos en el botón del enlace de la función Lambda bajo el campo **Physical ID**, accederemos directamente a la función Lambda, donde podremos visualizar el código de dicha función en el editor integrado, bajo la pestaña **Code**:

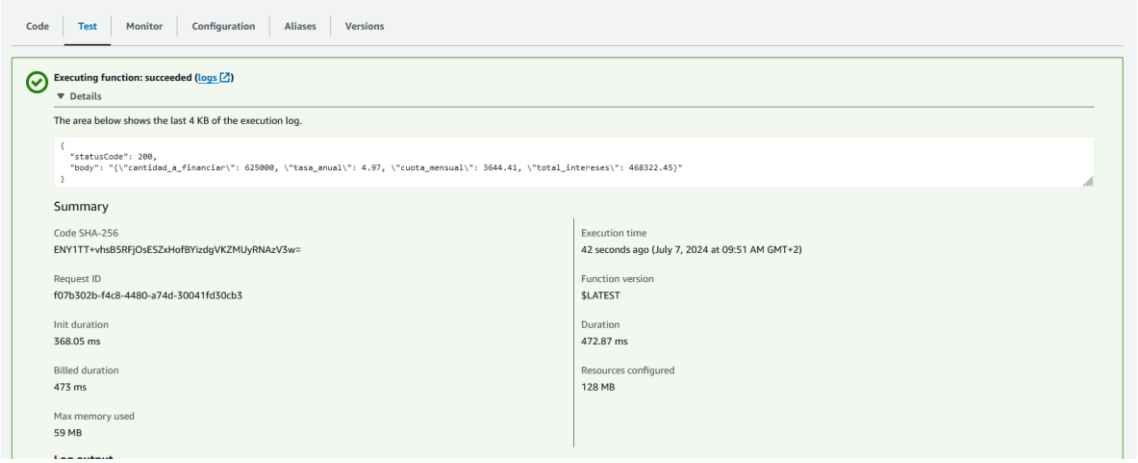


La función anterior, toma los parámetros correspondientes en el objeto *event* y calcula la cuota correspondiente de la hipoteca. Para ello previamente hace una llamada a una API externa para obtener el valor del EURIBOR.

8) Podemos testear el funcionamiento de nuestra función Lambda presionando la pestaña **Test** e introduciendo en el apartado **Event JSON** el siguiente evento de prueba y presionando el botón **Test**:

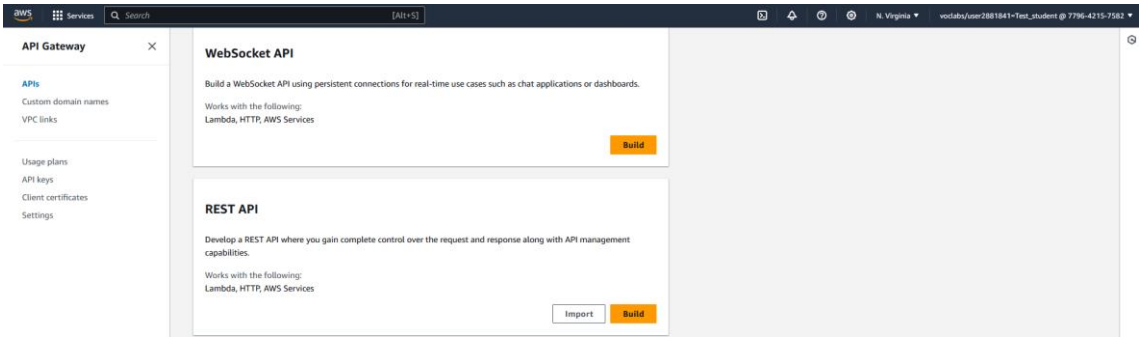
```
{
    "precio_vivienda": 750000,
    "entrada_vivienda": 125000,
    "tasa_diferencial": 1.25,
    "plazo_anos": 25
}
```


José Emilio Vera

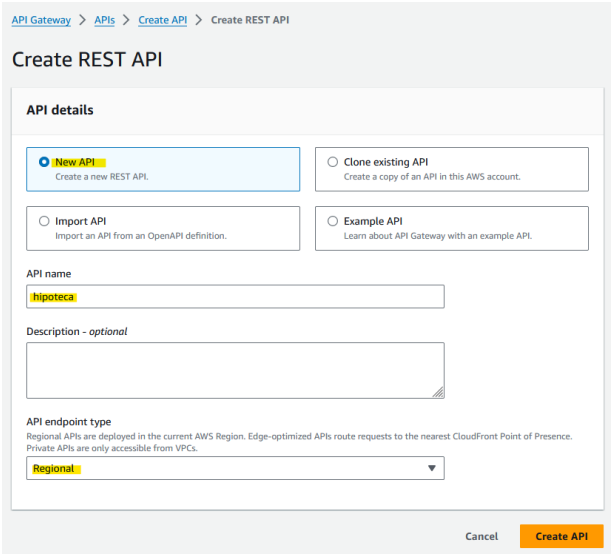


CREACIÓN DE LA API REST MEDIANTE AMAZON API GATEWAY

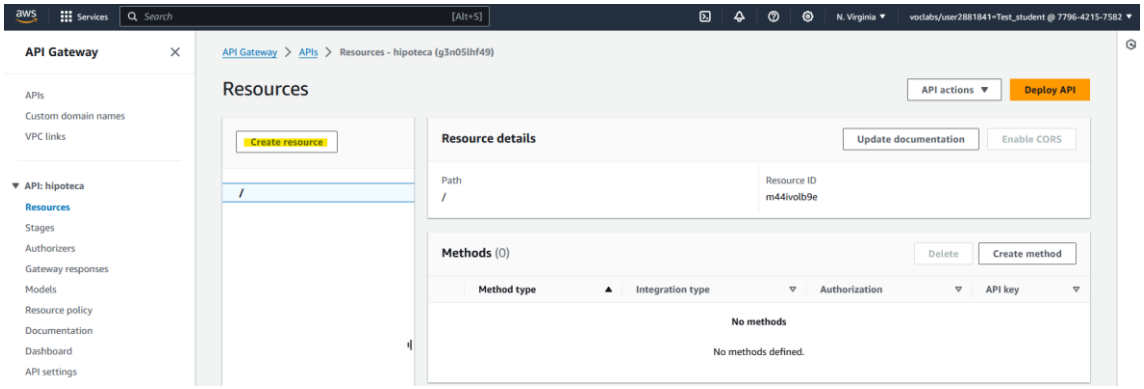
9) Para crear nuestra API REST y exponer la lógica de la función Lambda anterior, accedemos a la consola de Amazon API Gateway. Desde allí, en la sección **REST API**, presionamos el botón **Build**:



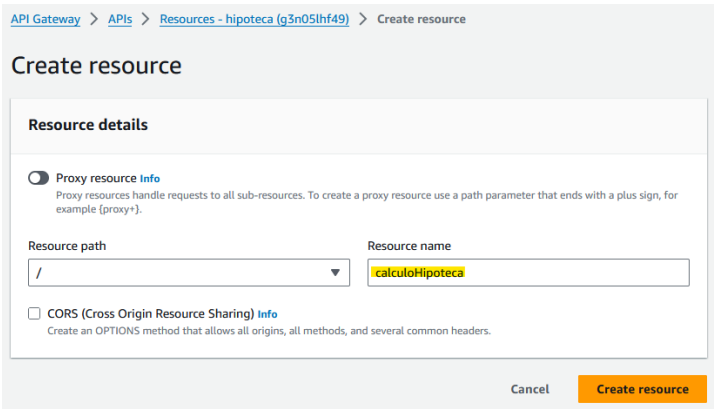
10) En el asistente, seleccionamos la opción **New API**, introducimos como **API name** el valor *hipoteca* y en **API endpoint type** elegimos el valor *Regional*:



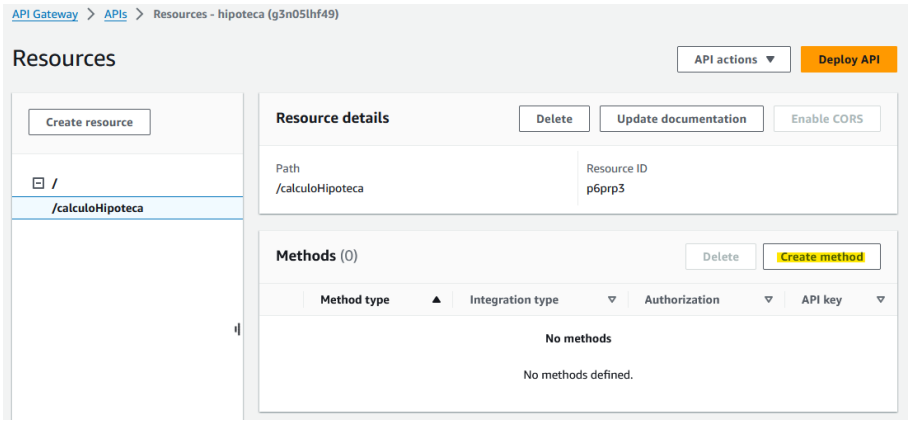
11) A continuación, crearemos un recurso en nuestra API REST, para lo cual presionamos el botón **Create resource**:



12) En la siguiente ventana introducimos en el apartado **Resource name** el valor *calculoHipoteca* y presionamos el botón **Create resource**:



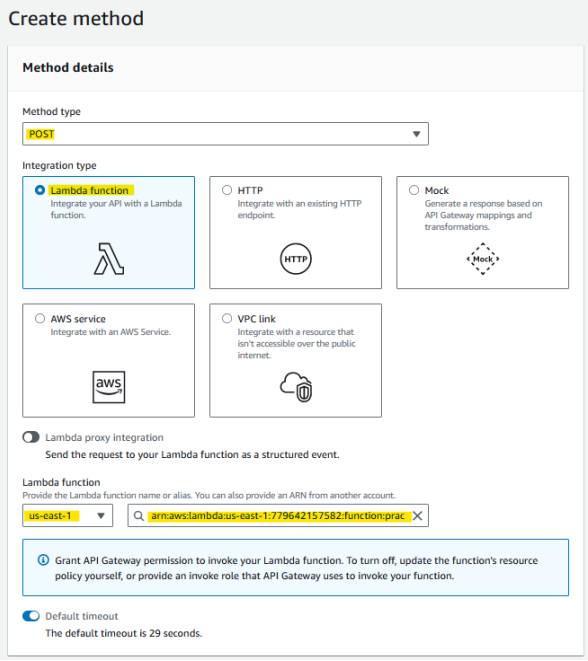
13) En el siguiente paso, crearemos un método sobre dicho recurso. Para ello, seleccionamos el recurso */calculoHipoteca* y presionamos el botón **Create method**:



14) Dentro del apartado **Method details**, seleccionamos las siguientes opciones:

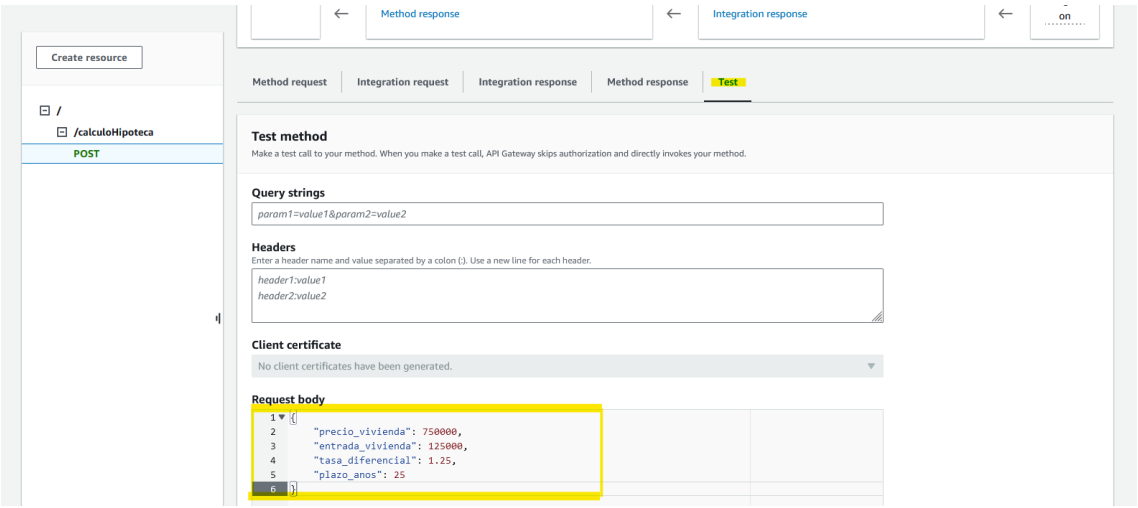
- **Method type**: *POST*
- **Integration type**: *Lambda function*

- **Lambda Region:** *us-east-1*
- **Lambda Function**: Seleccionar la función que tenga como parte del nombre *CalcularHipoteca*



Por último, presionamos el botón **Create method**, dejando el resto de las opciones en sus valores por defecto.

15) A continuación, testearemos el método que hemos definido. Para ello, presionamos la pestaña **Test** e introducimos el mismo documento JSON del apartado **8)** en la sección **Request body**. Por último, presionamos el botón **Test** (en la parte inferior de la pantalla):

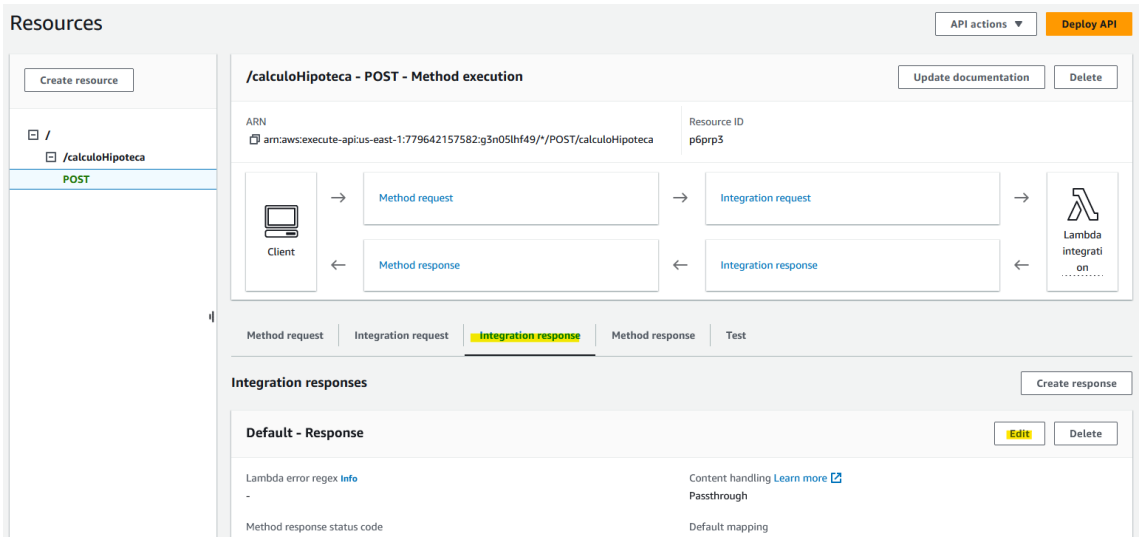


16) Tras lo anterior, podremos visualizar el log de ejecución de la invocación a la función Lambda a través de nuestra API REST:

TRANSFORMACIÓN DE LAS SOLICITUDES DE INTEGRACIÓN Y RESPUESTA DE INTEGRACIÓN

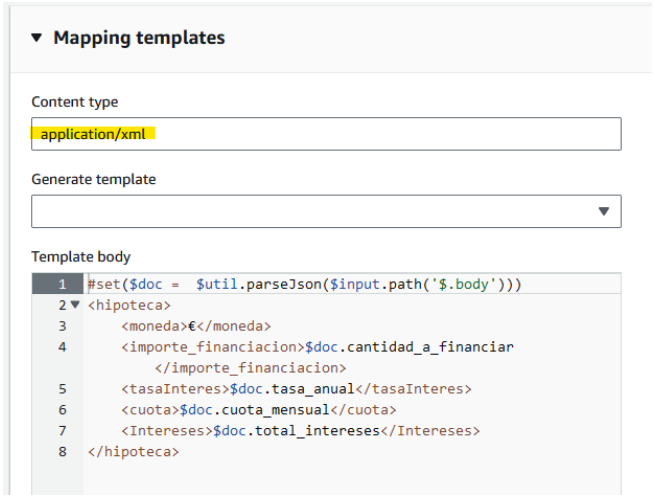
A menudo, las solicitudes que realizan los clientes web a las API REST no se transfieren tal cual al backend, sino que necesitan una transformación. Y viceversa, la respuesta del backend a veces tiene que transformarse para adaptarla a un formato inteligible por el cliente web. En Amazon API Gateway, estas transformaciones se realizan mediante Apache VTL (*Velocity Template Language*), un lenguaje de *scripting* ideal para embeber contenido dinámico en sitios web, o en otras páginas o plantillas.

17) En este apartado, vamos a suponer que el cliente web, espera que nuestra API REST devuelva el resultado en formato XML en lugar de en formato JSON. Para ello, dentro del apartado **Integration response**, presionamos el botón **Edit**:



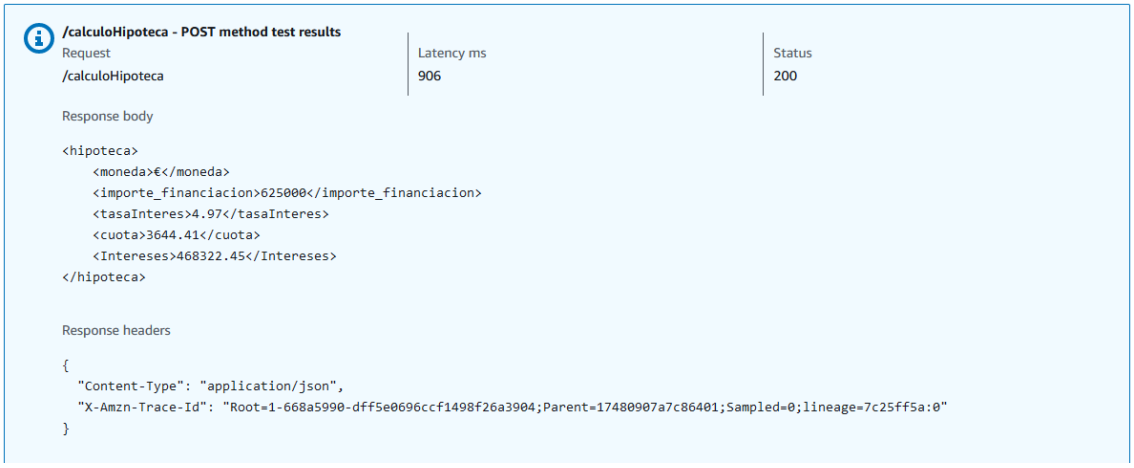
18) En la siguiente ventana, en el apartado **Mapping templates**, indicamos en el campo **Content type** el valor application/xml, e introducimos el siguiente documento en la sección **Template body**. Por último, presionamos el botón **Save**:

```
#set($doc = $util.parseJson($input.path('$.body')))
<hipoteca>
    <moneda>€</moneda>
    <importe_financiacion>$doc.cantidad_a_financiar</importe_financiacion>
    <tasaInteres>$doc.tasa_anual</tasaInteres>
    <cuota>$doc.cuota_mensual</cuota>
    <Intereses>$doc.total_intereses</Intereses>
</hipoteca>
```



El script anterior en VTL mapea los atributos del documento JSON con el que responde nuestra función Lambda a los elementos de un documento XML que espera el cliente web.

19) Si ahora testeamos el método desde el botón **Test**, facilitando como **Request body** el mismo documento JSON del apartado **8)** comprobaremos que la respuesta se transforma correctamente:



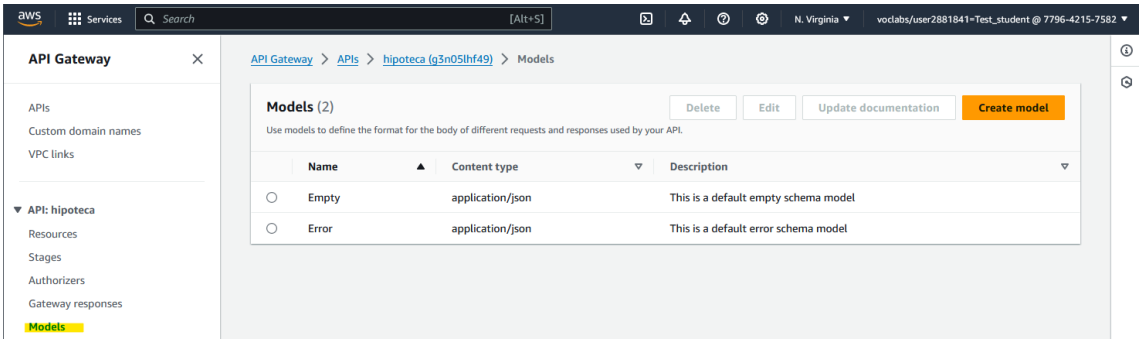
La solicitud de integración también podría transformarse, de manera que si la entrada del cliente web tuviera un formato diferente al esperado por nuestra función Lambda podríamos definir una plantilla en VTL a tal efecto. Esta actividad no se desarrolla en esta práctica.

VALIDACIÓN DE LAS SOLICITUDES

Otra de las características de una API REST definida mediante Amazon API Gateway es la posibilidad de validar las solicitudes HTTP realizadas a la API. Esta solicitud puede validarse requiriendo que ciertas

cabeceras, parámetros de la cadena de consulta estén presentes en la solicitud. Además, también puede validarse el cuerpo de la solicitud, siempre que éste se encuentre en formato JSON, mediante la especificación de un **modelo** en *JSON Schema*. En esta práctica, implementaremos la validación del cuerpo de la solicitud a nuestra API REST.

20) Para validar la solicitud *POST* al recurso */calculoHipoteca* previamente crearemos un modelo en JSON Schema. Para ello, desde el menú lateral accedemos a la opción **API: hipoteca / Models** y presionamos el botón **Create model**:

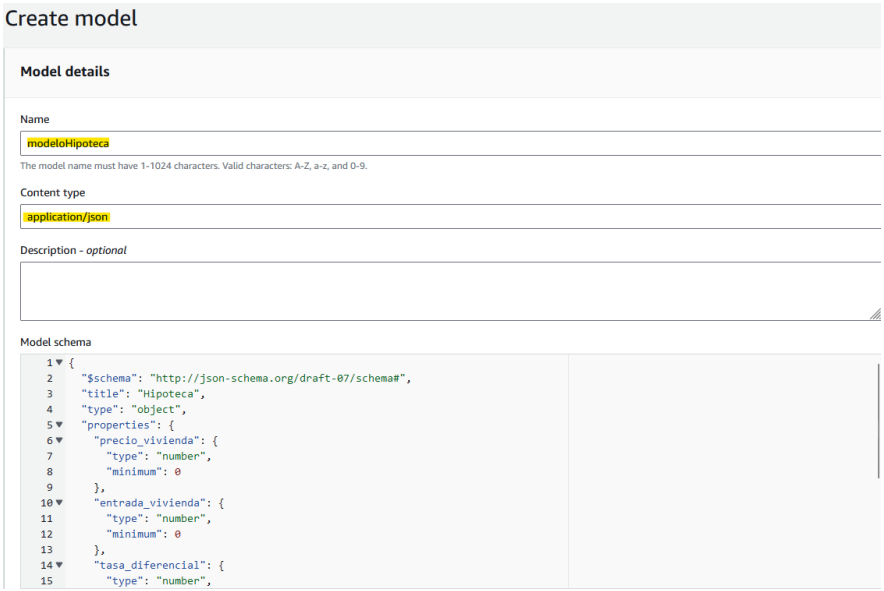


21) En la siguiente ventana, introducimos en el campo **Name** el valor *modeloHipoteca*, en el campo **Content type** el valor *application/json* y en el campo **Model schema** el siguiente documento:

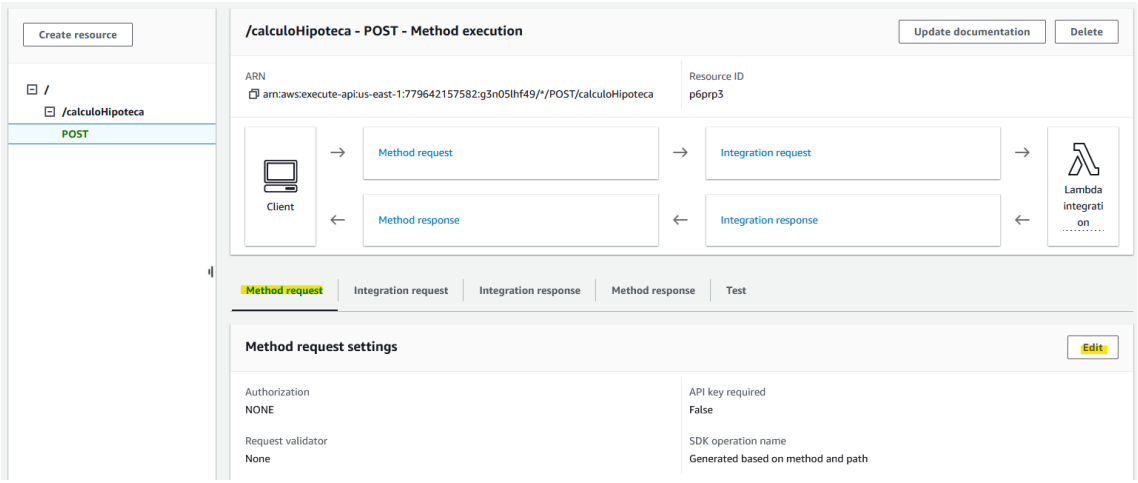
```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Hipoteca",
  "type": "object",
  "properties": {
    "precio_vivienda": {
      "type": "number",
      "minimum": 0
    },
    "entrada_vivienda": {
      "type": "number",
      "minimum": 0
    },
    "tasa_diferencial": {
      "type": "number",
      "minimum": 0
    },
    "plazo_anos": {
      "type": "integer",
      "minimum": 0
    }
  },
  "required": ["precio_vivienda", "entrada_vivienda", "tasa_diferencial", "plazo_anos"],
  "additionalProperties": false
}
```

El modelo anterior establece que todos los campos *precio_vivienda*, *entrada_vivienda*, *tasa_diferencial* y *plazo_anos* son de entrada obligatoria. Además, se impone una restricción de positividad en todos ellos. Además, el campo *plazo_anos* debe ser un valor entero.

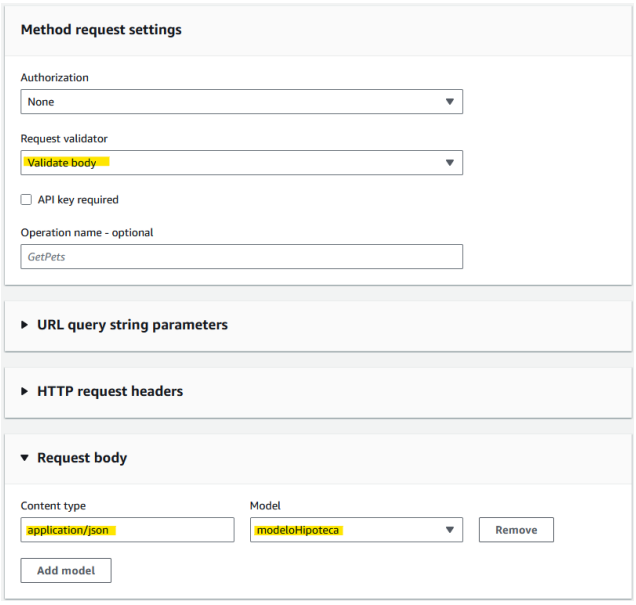
Por último, presionamos el botón **Create**.



22) Ahora deberemos asociar el modelo anterior a la validación del método *POST* sobre nuestro recurso */calculoHipoteca*. Para ello, accedemos al menú lateral **API: hipoteca / Resources**, presionamos el método POST correspondiente accedemos a la pestaña **Method request** y, dentro de la pestaña **Method request** presionamos el botón **Edit**:

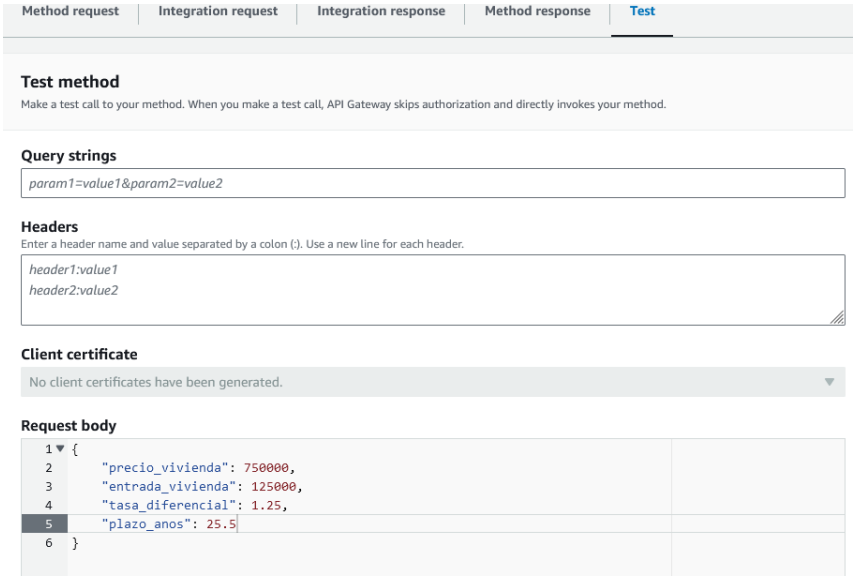


23) En el apartado **Request validator** seleccionamos la opción *Validate body* y en el grupo de opciones **Request body** presionamos el botón **Add model** y seleccionamos el modelo *modeloHipoteca* e introducimos en el campo **Content type** el valor *application/json*. Por último, presionamos el botón **Save**:



24) Para comprobar la validación del cuerpo de la solicitud, accedemos a la pestaña **Test** método e introducimos el siguiente JSON en el campo **Request body** y presionamos el botón **Test**:

```
{
    "precio_vivienda": 750000,
    "entrada_vivienda": 125000,
    "tasa_diferencial": 1.25,
    "plazo_anos": 25.5
}
```



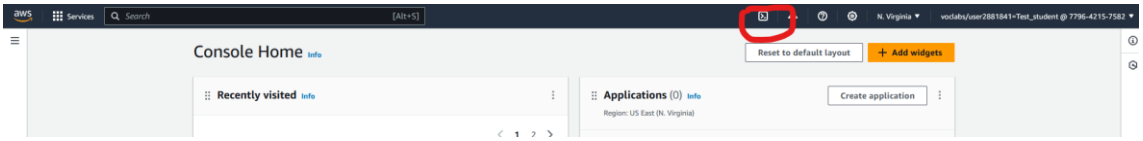
El resultado que API Gateway devuelve al cliente es un código de estado HTTP 400 con una cabecera *s-amzn-ErrorType* y un cuerpo con el mensaje de error de validación:

AUTENTICACIÓN DE LAS SOLICITUDES A UNA API REST

Amazon API Gateway permite definir un proceso de autenticación para determinar si el solicitante de una operación (método-recurso) sobre una API REST es una aplicación o usuario legítimo. Este proceso se puede realizar de varias maneras:

a) **Mediante AWS IAM**. Sólo aquellos usuarios de IAM autorizados podrán realizar solicitudes a la API REST.
b) **Mediante autorizadores Lambda**. En este caso deberíamos implementar una lógica en una función Lambda que, en base a unos parámetros o tokens de identidad, devuelva una política de IAM que evaluará API Gateway para determinar si el usuario de la aplicación tiene permisos o no para invocar el método-recurso de nuestra API REST.
c) **Mediante autorizadores de Amazon Cognito**. Es posible integrar un pool de usuarios de Cognito con una API REST para delegar el proceso de autenticación en Amazon Cognito, que generará un conjunto de JWT (*JSON Web Token*) que se presentarán en API Gateway. Este método será el que se implementará en esta práctica.

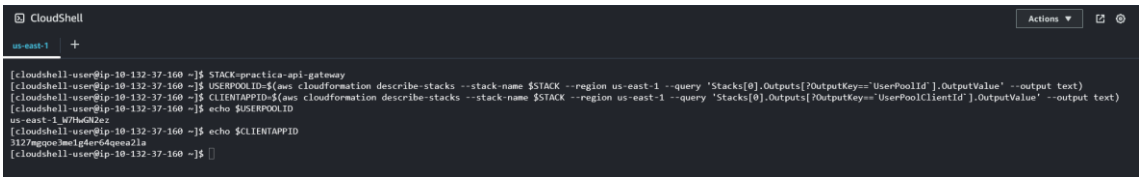
25) Como parte del despliegue realizado en el apartado **1)**, se creó un *user pool* de Amazon Cognito. Previamente necesitamos conocer tanto el ID del *user pool* como de la aplicación cliente creada. Para ello, abrimos un **CloudShell** desde la barra de herramientas de la Consola de Administración de AWS:



26) Desde *CloudShell* ejecutamos la siguiente orden para obtener el ID del *user pool* de Amazon Cognito creado:

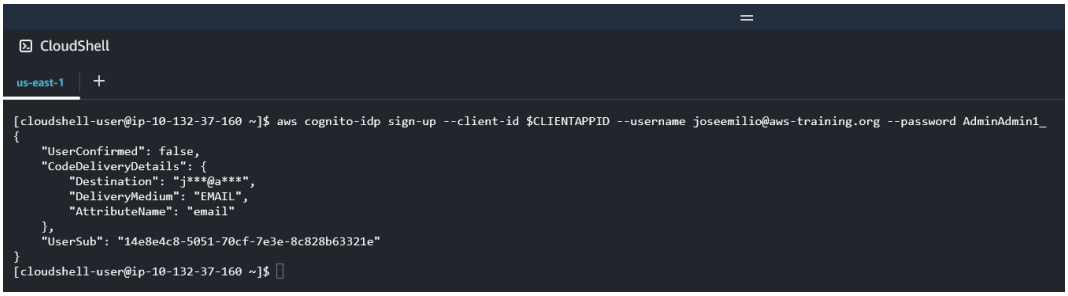
```
$ STACK=practica-api-gateway

$ CLIENTAPPID=$(aws cloudformation describe-stacks --stack-name $STACK --region us-east-
1 --query 'Stacks[0].Outputs[?OutputKey==`UserPoolClientId`].OutputValue' --output text)
```



27) A continuación, ejecutamos la siguiente orden para crear un usuario de prueba en nuestro *user pool*, substituyendo el *placeholder* por un correo electrónico válido:

```
$ aws cognito-idp sign-up --client-id $CLIENTAPPID --username <email> --password AdminAdmin1_
```



Del JSON anterior, obtenemos el nombre del usuario en el campo *UserSub*. Lo almacenamos en una variable mediante la orden, substituyendo el *placeholder* por el valor correspondiente:

```
$ USUARIO=<User>
```

28) Si todo ha ido bien, habremos recibido un correo electrónico desde la dirección *no-reply@verificationemail.com* donde se indica un código de verificación. Ignoramos el correo anterior y verificamos a nuestro nuevo usuario mediante la orden:

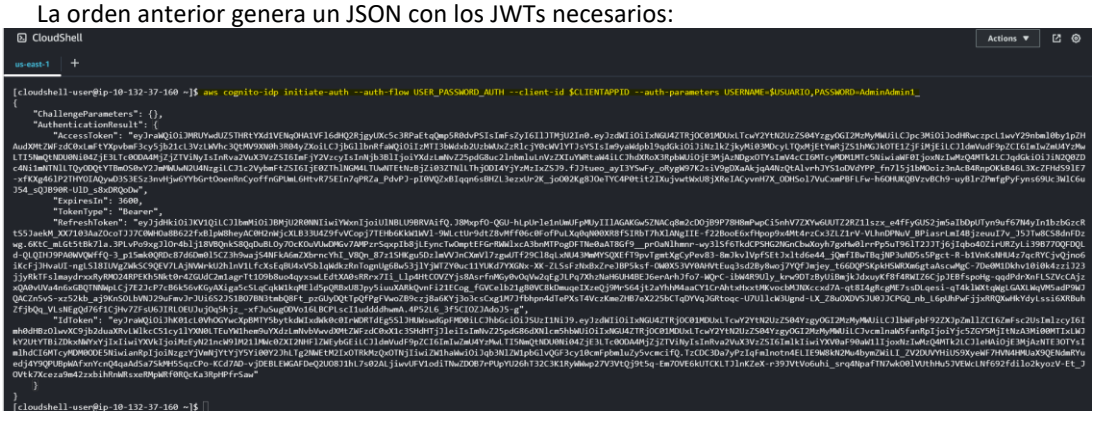
```
$ aws cognito-idp admin-confirm-sign-up --user-pool-id $USERPOOLID --username $USUARIO
```

La orden anterior no produce ningún resultado por la consola. Esto significa que todo ha ido bien.

29) A continuación, vamos a simular el proceso de autenticación de nuestro usuario en el *user pool* de Amazon Cognito. Para ello ejecutamos la siguiente orden:

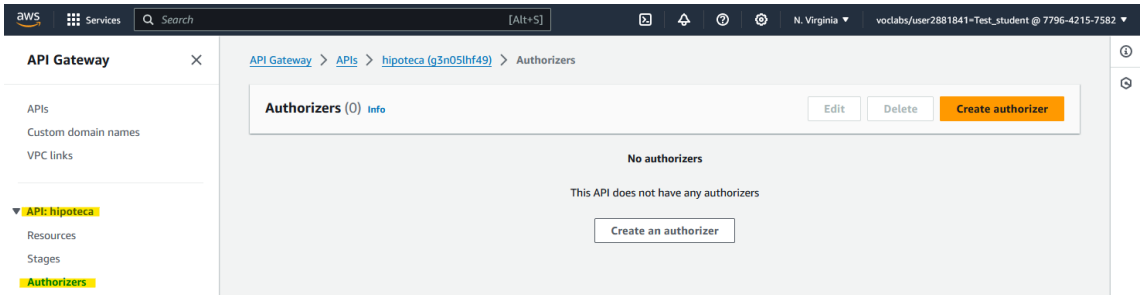
```
$ aws cognito-idp initiate-auth --auth-flow USER_PASSWORD_AUTH --client-id $CLIENTAPPID --auth-parameters USERNAME=$USUARIO,PASSWORD=AdminAdmin1_
```

La orden anterior genera un JSON con los JWTs necesarios:



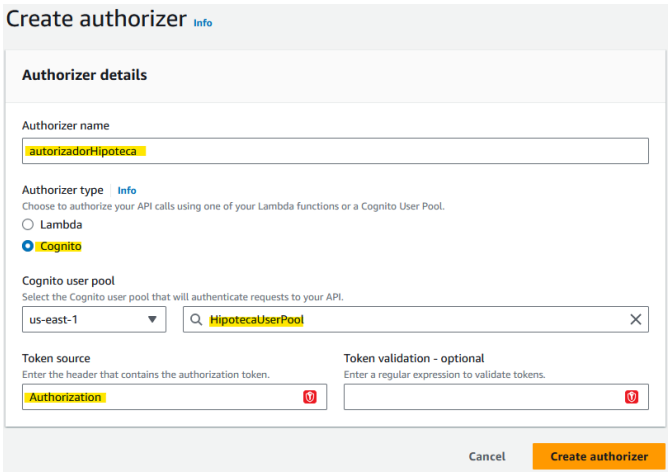
Copiamos el valor del campo **IdToken** del documento anterior (sin incluir las comillas) en el portapapeles.

30) A continuación, crearemos un **autorizador de Cognito** para nuestra API REST. Para ello desde la consola de Amazon API Gateway seleccionamos la opción **API: hipoteca** del menú lateral y presionamos la opción **Authorizers**. Presionamos el botón **Create authorizer**:



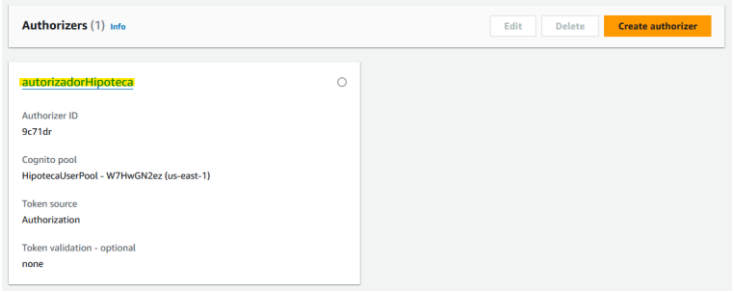
31) En la siguiente ventana introducimos los siguientes valores:

- **Authorizer name**: *autorizadorHipoteca*
- **Authorizer type**: *Cognito*
- **Cognito user pool**: Elegimos el *user pool* llamado *HipotecaUserPool*
- **Token source**: *Authorization*

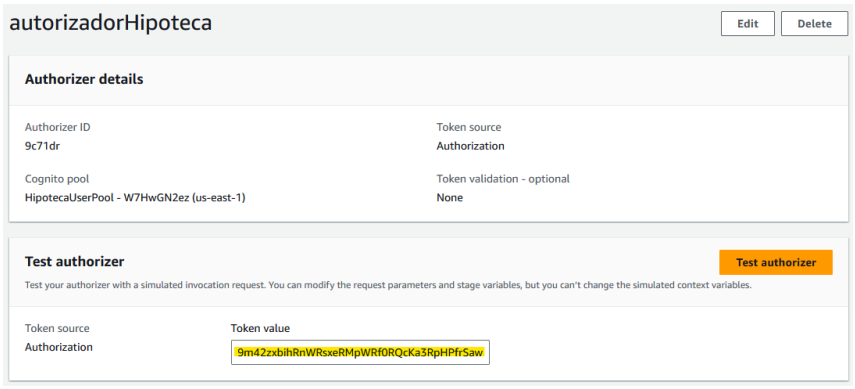


Por último, presionamos el botón **Create authorizer**.

32) A continuación, vamos a probar el correcto funcionamiento de nuestro autorizador. Para ello pinchamos en el enlace indicado en el nombre del autorizador:

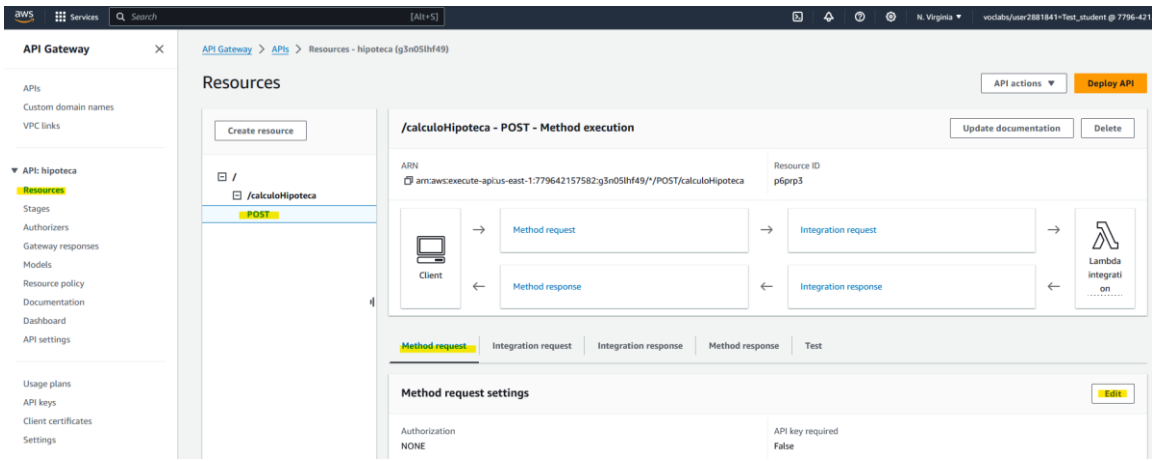


33) En la siguiente ventana, pegamos en el campo **Token value** el valor del token de identidad que hemos copiado en el apartado **29)** y presionamos el botón **Test authorizer**:

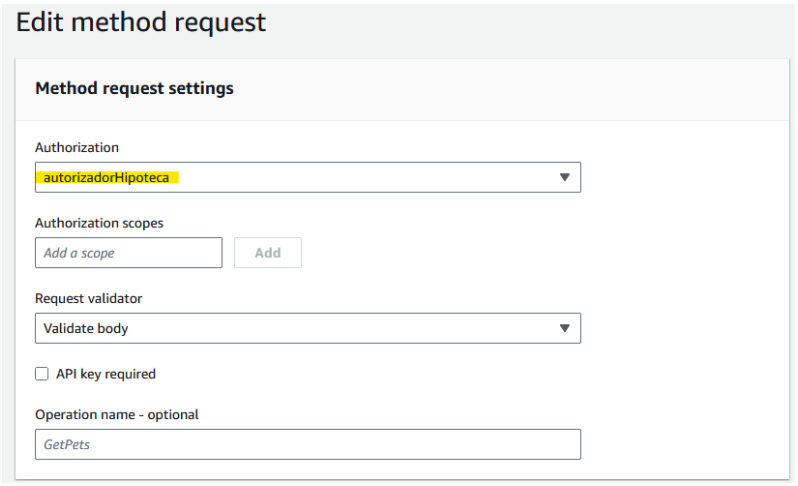


Deberemos obtener una respuesta similar a la siguiente, en el que se nos muestra que Amazon Cognito ha validado el token y nos responde con un código de estado HTTP 200 y como cuerpo de la respuesta nos muestra el *Payload* del JWT, ya decodificado en *Base64url*:

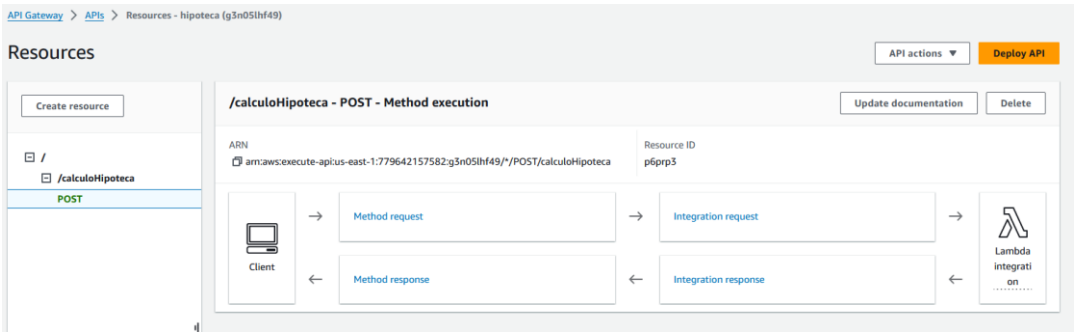
34) Ahora sólo nos resta integrar el autorizador de Cognito que acabamos de crear con el método POST sobre nuestro recurso */calculoHipoteca*. Para ello, desde el menú lateral seleccionamos **API: hipoteca / Resources**, activamos el método POST correspondiente y, desde la pestaña **Method request** presionamos el botón **Edit**:



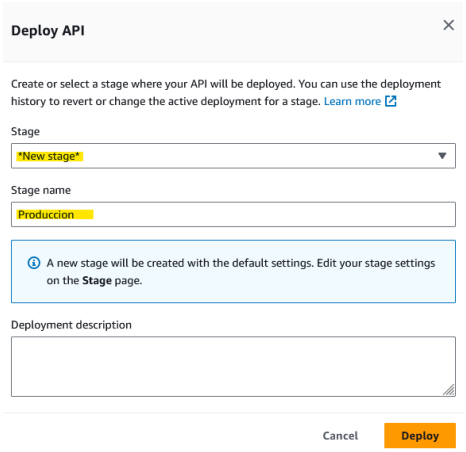
35) Desde la siguiente ventana, en la opción **Authorization** seleccionamos el autorizador de Cognito llamado *autorizadorHipoteca*, creado en el apartado **31)** y presionamos el botón **Save**:



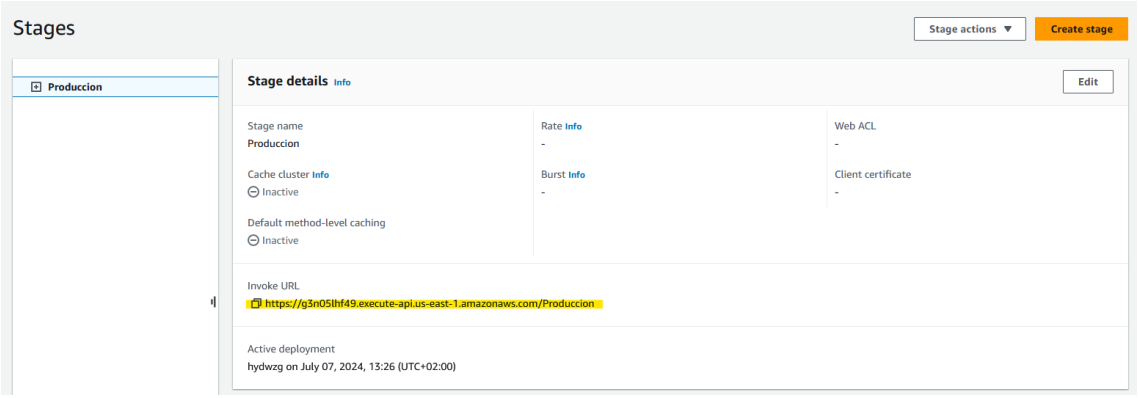
36) Para testear la validación, es preciso desplegar nuestra API REST. Para ello, presionamos el botón **Deploy API**:



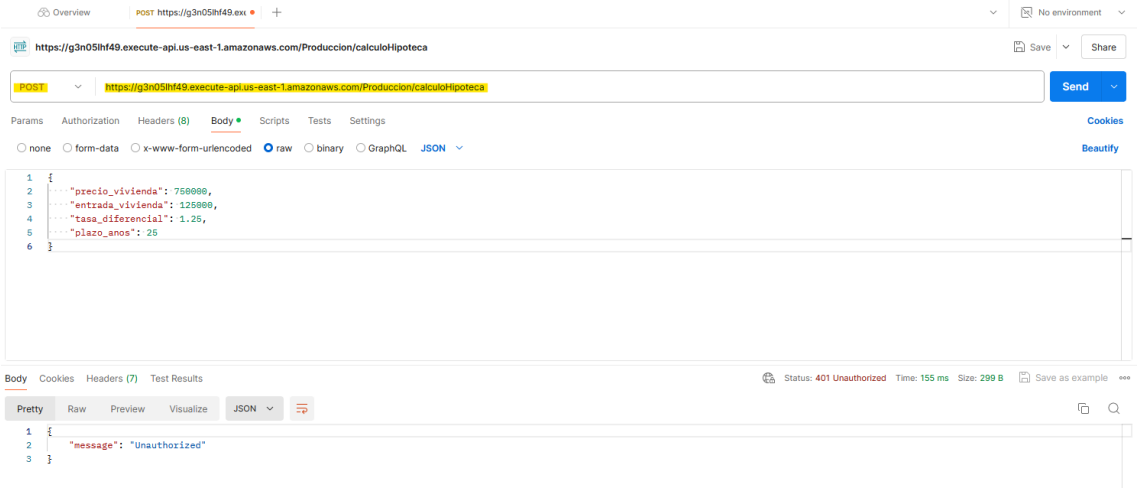
37) En la siguiente ventana, debemos seleccionar una etapa (*stage*). Una **etapa** es el despliegue de la especificación de nuestra API. A partir de una misma especificación, es posible crear diferentes etapas, para Producción, Desarrollo, Pruebas, etc. En esta ventana, seleccionaremos dentro del campo **Stage** el valor **New stage** y en el campo **Stage name** introduciremos el valor *Produccion*. A continuación, presionaremos el botón **Deploy**:



A pesar de obtener un resultado exitoso, el despliegue puede durar algún minuto. Sería prudente esperar hasta que esto ocurra. Mientras tanto, vamos a copiar la URL que aparece en el campo **Invoke URL** de nuestra etapa de *Produccion*:

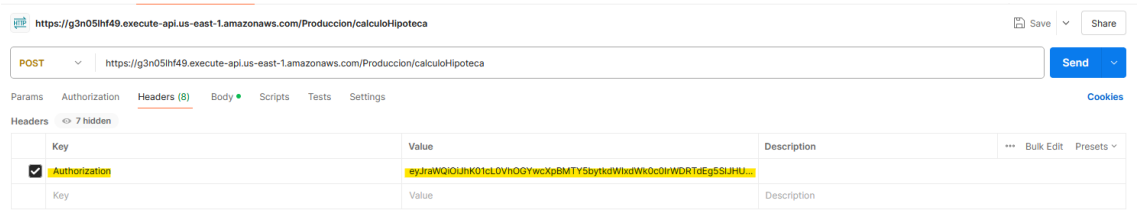


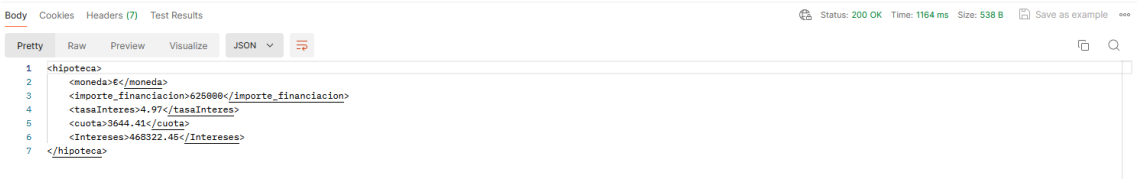
38) A continuación, abriremos una sesión en la herramienta **Postman** (https://www.postman.com) y realizamos una solicitud POST contra la URL copiada en el apartado, añadiendo como ruta a dicha URL el nombre del recurso *calculaHipoteca*. En la pestaña **Body** pegamos el JSON del apartado **8)**



El resultado que se obtiene es una respuesta con un código HTTP 401 *Unauthorized*. Esto es debido a que no se ha incluido el Token de Identidad en la solicitud.

39) Para comprobar el correcto funcionamiento, vamos a modificar la solicitud HTTP. Seleccionamos la pestaña **Headers**. Desde aquí, agregamos una nueva cabecera llamada **Authorization** y, en la columna **Value** pegamos el token de identidad que generamos en el apartado **29)**. Una vez realizado los cambios, presionamos el botón **Send** para enviar la solicitud:



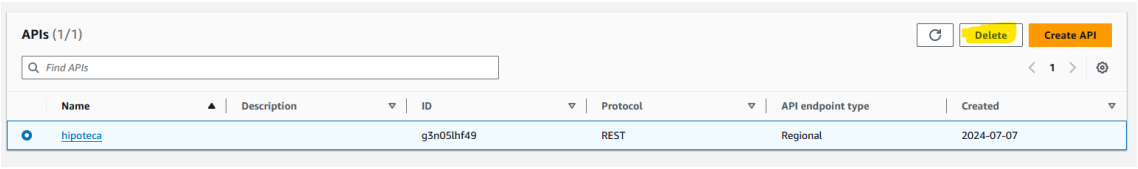


En esta ocasión, el resultado es una respuesta con código de estado HTTP 200 OK, con el resultado esperado.

Limpieza de la Práctica:

Los recursos empleados en esta práctica son completamente *serverless* por lo que, si no se utilizan, no supondrán ningún coste adicional. Para terminar esta práctica y liberar los recursos creados, simplemente debemos dar los siguientes pasos:

- **Eliminación de la API REST creada**. Desde la consola de Amazon API Gateway seleccionamos nuestra API y presionamos el botón **Delete**:



- **Eliminación de la función Lambda y el *user pool* de Amazon Cognito**. Lo haremos desde **Cloudshell**, ejecutando la siguiente orden

```
$ aws cloudformation delete-stack --stack-name practica-api-gateway
```