



Segundo Avance

Diseño y Construcción de Componentes

Esteban Rojas Herrera

Inventario de Armas RPG

Prof. Mario Miguel Agüero Obando

# Inventario\_rpg\_api

## Descripción

Este proyecto implementa un sistema de inventario de armas para un RPG usando:

- **AWS Lambda** como backend serverless
- **API Gateway** para exponer endpoints RESTful
- **DynamoDB** como base de datos nativa en la nube
- **Postman** para pruebas de endpoints

## Descripción general del proyecto

Desarrollo de un sistema backend para la gestión de armas de un RPG utilizando AWS Lambda, DynamoDB y API Gateway. Se creó una función que permite insertar armas en la base de datos a través de una API RESTful. Además, se comenzó a desarrollar una interfaz gráfica con PyQt5 para conectar con la API.

## Funcionalidades actuales

- Función Lambda `inventario_rpg_funcion` desplegada y funcionando.
- Ruta `/inventario` creada en API Gateway, accesible desde Postman.
- Prueba exitosa enviando datos JSON para insertar, editar, ver y eliminar un arma.

## Endpoints disponibles

### POST /inventario

Crea un arma nueva.

- Requiere JSON en el body con los siguientes campos:

```
{
  "arma_id": "verificable001",
  "nombre": "Daga de Prueba",
  "tipo": "Daga",
  "dano": 20,
  "rareza": "Comun",
  "disponible": true
}
```

## **GET /arma/{id}**

Obtiene los datos de un arma usando su arma\_id como parámetro de ruta.

## **PUT /arma/{id}**

Actualiza un arma existente.

- Requiere un body JSON con los datos a modificar: {  
 "nombre": "Espada Renombrada",  
 "tipo": "Espada",  
 "dano": 150,  
 "rareza": "Legendaria",  
 "disponible": false  
}

## **DELETE /arma/{id}**

Elimina un arma según su arma\_id.

## **Tecnologías y servicios usados**

- Python 3.12
- AWS Lambda
- AWS DynamoDB
- API Gateway REST
- Postman
- GitHub (repositorio con código fuente)

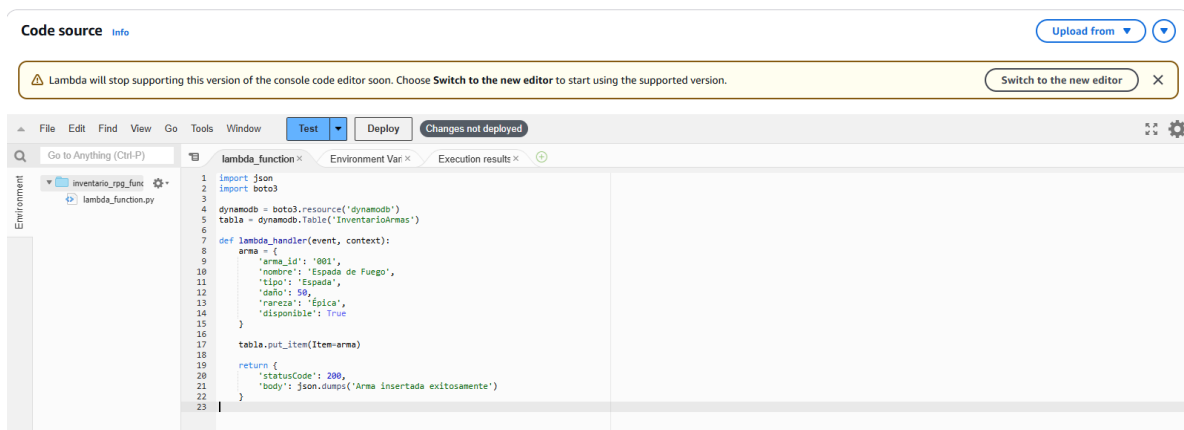
## **Requisitos para correr el proyecto (localmente o para interfaz futura)**

- Python 3.12
- boto3
- Cuenta AWS

## Aprendizajes y retos superados

- Conexión e integración de funciones Lambda con API Gateway
- Manejo de errores comunes como `AccessDeniedException` por falta de permisos IAM
- Validación y limpieza de parámetros `pathParameters` y `body`
- Diferencias entre atributos como `dano` vs `daño` en DynamoDB
- Serialización de tipos `Decimal` para respuestas JSON
- Configuración de rutas dinámicas en API Gateway REST y mapeo correcto de parámetros
- Deploy exitoso desde consola AWS con pruebas en Postman
- Experiencia práctica en arquitectura serverless con AWS

## Test Para agregar arma a la base de datos



The screenshot shows the AWS Lambda console's code editor. At the top, there's a warning banner: "Lambda will stop supporting this version of the console code editor soon. Choose **Switch to the new editor** to start using the supported version." Below the banner, the editor has tabs for "Code source", "Environment Var", and "Execution results". The "Code source" tab is active, showing a Python file named `lambda_function.py`. The code defines a `lambda_handler` function that takes `event` and `context` as arguments. It imports `json` and `boto3`, then creates a `dynamodb` resource and a `tabla` object for the `inventarioArmas` table. The handler function creates a dictionary `arma` with attributes like `arma_id`, `nombre`, `tipo`, `daño`, `rareza`, and `disponible`. It then calls `tabla.put_item(Item=arma)` to insert the item into the database. Finally, it returns a response with `statusCode: 200` and a `body` message: `Arma insertada exitosamente`.

```
1 import json
2 import boto3
3
4 dynamodb = boto3.resource('dynamodb')
5 tabla = dynamodb.Table('inventarioArmas')
6
7 def lambda_handler(event, context):
8     arma = {
9         'arma_id': '001',
10         'nombre': 'Espada de Fuego',
11         'tipo': 'Espada',
12         'daño': 50,
13         'rareza': 'épica',
14         'disponible': True
15     }
16     tabla.put_item(Item=arma)
17
18     return {
19         'statusCode': 200,
20         'body': json.dumps('Arma insertada exitosamente')
21     }
22
23
```

## Segundo experimento

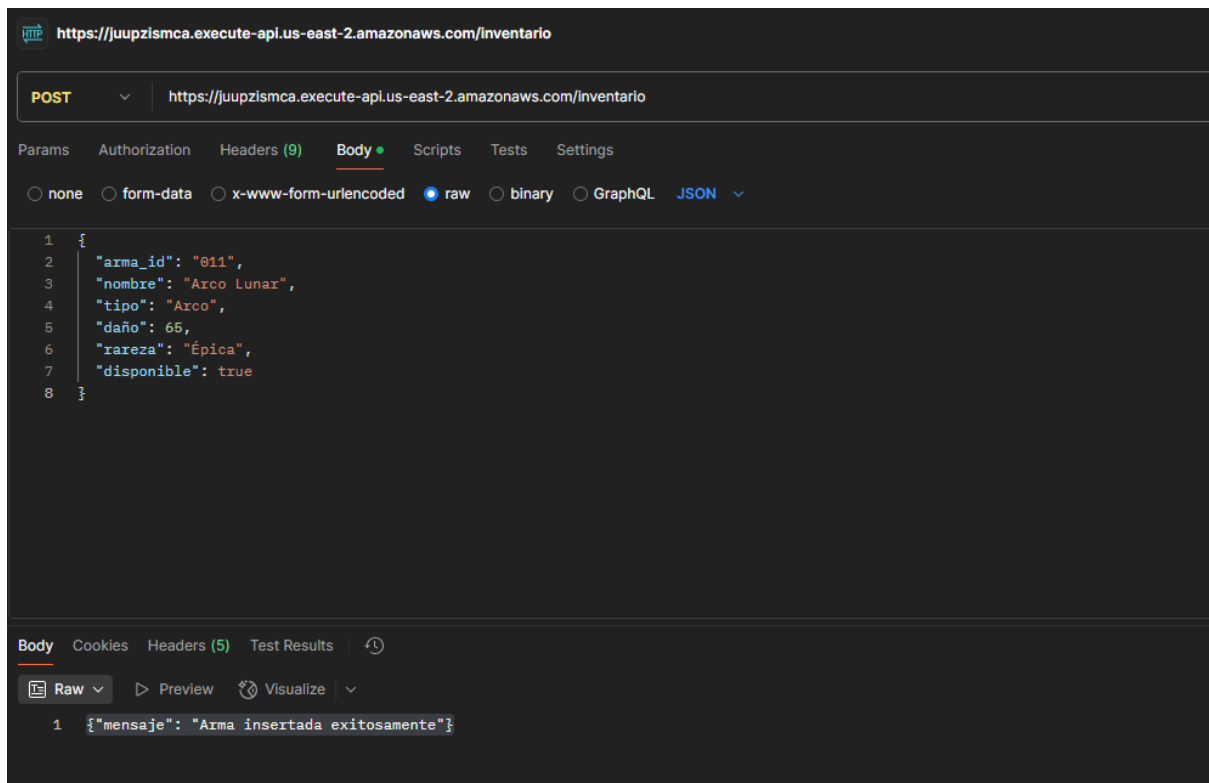


The screenshot shows the AWS Lambda console's code editor with the same warning banner as the previous image. The "Code source" tab is active, showing a Python file named `lambda_function.py`. The code imports `json` and `boto3`, then creates a `dynamodb` resource and a `tabla` object for the `inventarioArmas` table. The handler function calls `tabla.get_item(Key={'arma_id': '001'})` to retrieve an item from the database. It then checks if the `'Item'` is in the response. If it is, it returns a response with `statusCode: 200` and a `body` message: `Arma encontrada`. If not, it returns a response with `statusCode: 404` and a `body` message: `Arma no encontrada`.

```
1 import json
2 import boto3
3
4 dynamodb = boto3.resource('dynamodb')
5 tabla = dynamodb.Table('inventarioArmas')
6
7 def lambda_handler(event, context):
8     respuesta = tabla.get_item(Key={'arma_id': '001'})
9
10    if 'Item' in respuesta:
11        return {
12            'statusCode': 200,
13            'body': json.dumps(respuesta['Item'])
14        }
15    else:
16        return {
17            'statusCode': 404,
18            'body': json.dumps('Arma no encontrada')
19        }
20
```

## Prueba de conexión en Postman.

Ya se puede conectar el API gateway, junto a la función en Lambda y en DynamoDB para insertar armas al inventario desde la función.



## Repositorio del proyecto

GitHub: <https://github.com/tebankai07/inventario-rpg>

Estructura:

- /lambda: funciones AWS Lambda (Python)
- /ui: archivo `.ui` y `main.py` para interfaz PyQt5
- /docs: documentación del proyecto