

CS2032 - Cloud Computing (Ciclo 2024-2)

Multi-tenancy

Semana 9 - Taller 2: Microservicio Multi-tenancy

ELABORADO POR: GERALDO COLCHADO

Contenido

Microservicio Multi-tenancy

1. **Objetivo del taller 2**
2. Ejercicio 1: Crear una función Lambda - ListarAlumnos
3. Ejercicio 2: Crear una función Lambda - CrearAlumno
4. Ejercicio 3: Crear una función Lambda - ModificarAlumno
5. Ejercicio 4: Crear una función Lambda - BuscarAlumno
6. Ejercicio 5: Crear una función Lambda - EliminarAlumno
7. Ejercicio 6: Publicar microservicio como API Rest CRUD con servicio Api Gateway
8. Ejercicio 7: Ejercicio propuesto
9. Cierre

Objetivo del taller 2:

Microservicio Multi-tenancy

- Diseñar e implementar un Microservicio Multi-tenancy con servicio Lambda (Serverless) y Api Gateway (Serverless)

Contenido

Microservicio Multi-tenancy

1. Objetivo del taller 2
2. **Ejercicio 1: Crear una función Lambda - ListarAlumnos**
3. Ejercicio 2: Crear una función Lambda - CrearAlumno
4. Ejercicio 3: Crear una función Lambda - ModificarAlumno
5. Ejercicio 4: Crear una función Lambda - BuscarAlumno
6. Ejercicio 5: Crear una función Lambda - EliminarAlumno
7. Ejercicio 6: Publicar microservicio como API Rest CRUD con servicio Api Gateway
8. Ejercicio 7: Ejercicio propuesto
9. Cierre

Function as a Service (FaaS)

Lambda (Serverless)

AWS Lambda

Ejecute código sin tener que pensar en los servidores o los clústeres.

[Introducción a AWS Lambda](#) [Contacte con un especialista de AWS Lambda](#)

1 millón de solicitudes gratis
al mes con el [nivel gratuito de AWS](#)

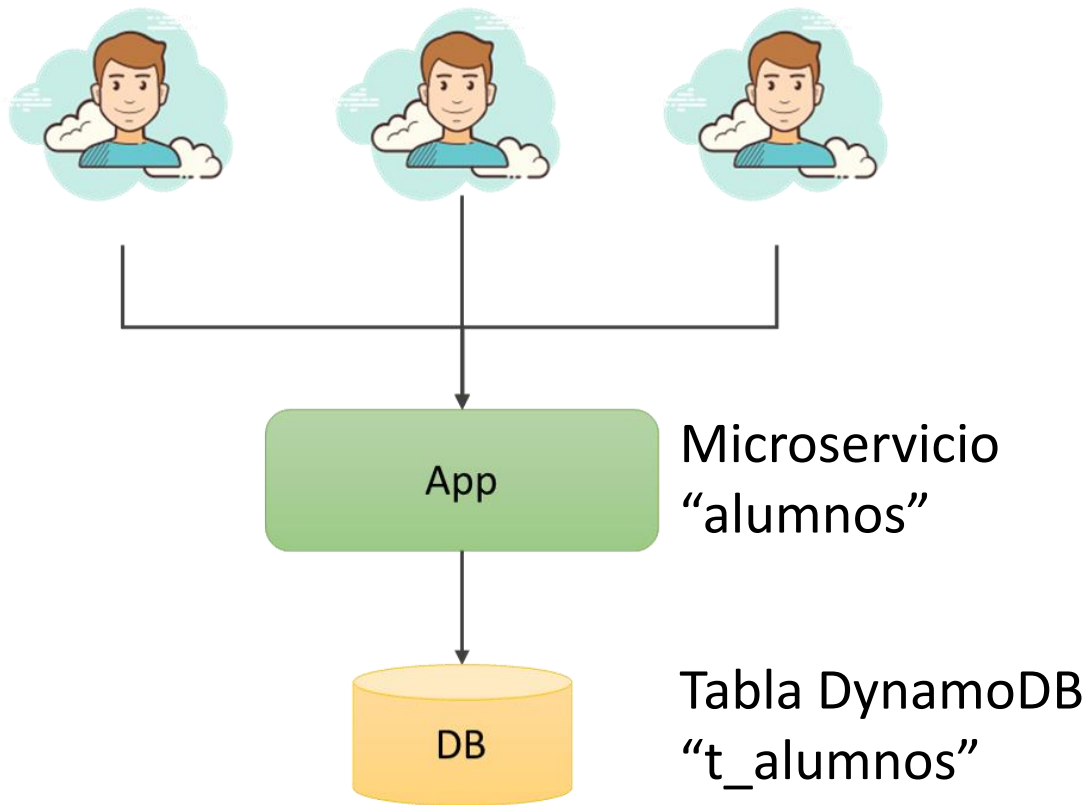
- Ejecute el código sin aprovisionar ni administrar la infraestructura. Simplemente escriba y cargue el código como un archivo .zip o una imagen de contenedor.
- Responda automáticamente a las solicitudes de ejecución de código a cualquier escala, desde una docena de eventos al día hasta cientos de miles por segundo.
- Ahorre costos al pagar solamente por el tiempo de informática que utiliza, por milisegundo, en lugar de aprovisionar la infraestructura por adelantado para la capacidad máxima.
- Optimice el tiempo de ejecución del código y el rendimiento con el tamaño adecuado de la memoria de las funciones. Responda a la alta demanda en milisegundos de dos dígitos con simultaneidad aprovisionada.

Funcionamiento

AWS Lambda es un servicio informático sin servidor y basado en eventos que le permite ejecutar código para prácticamente cualquier tipo de aplicación o servicio backend sin necesidad de aprovisionar o administrar servidores. Puede activar Lambda desde más de 200 servicios de AWS y aplicaciones de software como servicio (SaaS), y solo paga por lo que utiliza.

Patrones de diseño BD Multi-tenancy

Patrón 1: En DynamoDB (NoSQL)



Ejercicio: Diseñe e implemente un **microservicio Multi-tenancy** para almacenar los alumnos de una universidad. Considerar que el `tenant_id` es un código por cada universidad en mayúsculas:

`tenant_id`: UTEC, UNIV2, UNIV3, UNIV4, etc.

Ejercicio 1:

Crear una función Lambda - ListarAlumnos

- **Paso 1:** Requisito: Tener creada la tabla DynamoDB t_alumnos
- **Paso 2:** Crear la función lambda

Crear una función

Seleccione una de las siguientes opciones para crear la función.

Crear desde cero

Empiece con un sencillo ejemplo "Hello World".

Utilizar un proyecto

Cree una aplicación Lambda muestra y los ajustes de casos de uso comunes.

Información básica

Nombre de la función
Escriba un nombre para describir el propósito de la función.

ListarAlumnos

Utilice exclusivamente letras, números, guiones o guiones bajos. No incluya espacios.

Tiempo de ejecución [Información](#)
Elija el lenguaje que desea utilizar para escribir la función. Tenga en cuenta que el editor de código de la consola si

Python 3.9

Arquitectura [Información](#)
Elija la arquitectura del conjunto de instrucciones que desea para el código de la función.

☒ x86_64
☐ arm64

Permisos [Información](#)
De forma predeterminada, Lambda creará un rol de ejecución con permisos para cargar registros en Amazon Cloud

▼ **Cambiar el rol de ejecución predeterminado**

Rol de ejecución
Seleccione un rol que defina los permisos de la función. Para crear un rol personalizado, vaya a la [consola de IAM](#).

☐ Creación de un nuevo rol con permisos básicos de Lambda
☒ **Uso de un rol existente**
☐ Creación de un nuevo rol desde la política de AWS templates

Rol existente
Seleccione un rol existente que haya creado para usarlo con esta función de Lambda. El rol debe tener permiso para

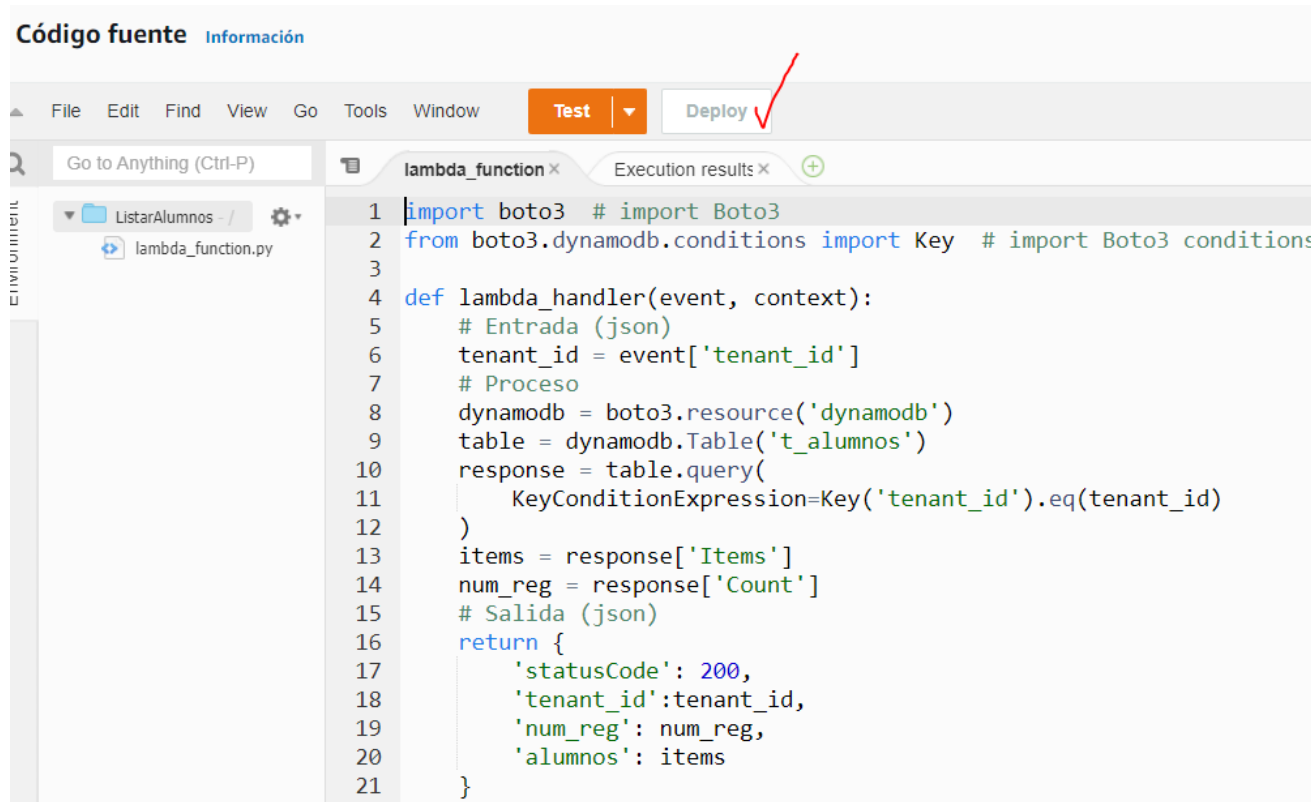
LabRole

Crear una función

Ejercicio 1:

Crear una función Lambda - ListarAlumnos

- **Paso 3:** Escribir el siguiente código fuente y “Deploy”



```
Código fuente Información
File Edit Find View Go Tools Window Test Deploy
Go to Anything (Ctrl-P)
ListarAlumnos /
lambda_function.py
1 import boto3 # import Boto3
2 from boto3.dynamodb.conditions import Key # import Boto3 conditions
3
4 def lambda_handler(event, context):
5     # Entrada (json)
6     tenant_id = event['tenant_id']
7     # Proceso
8     dynamodb = boto3.resource('dynamodb')
9     table = dynamodb.Table('t_alumnos')
10    response = table.query(
11        KeyConditionExpression=Key('tenant_id').eq(tenant_id)
12    )
13    items = response['Items']
14    num_reg = response['Count']
15    # Salida (json)
16    return {
17        'statusCode': 200,
18        'tenant_id': tenant_id,
19        'num_reg': num_reg,
20        'alumnos': items
21    }
```

```
import boto3 # import Boto3
from boto3.dynamodb.conditions import Key # import Boto3 conditions

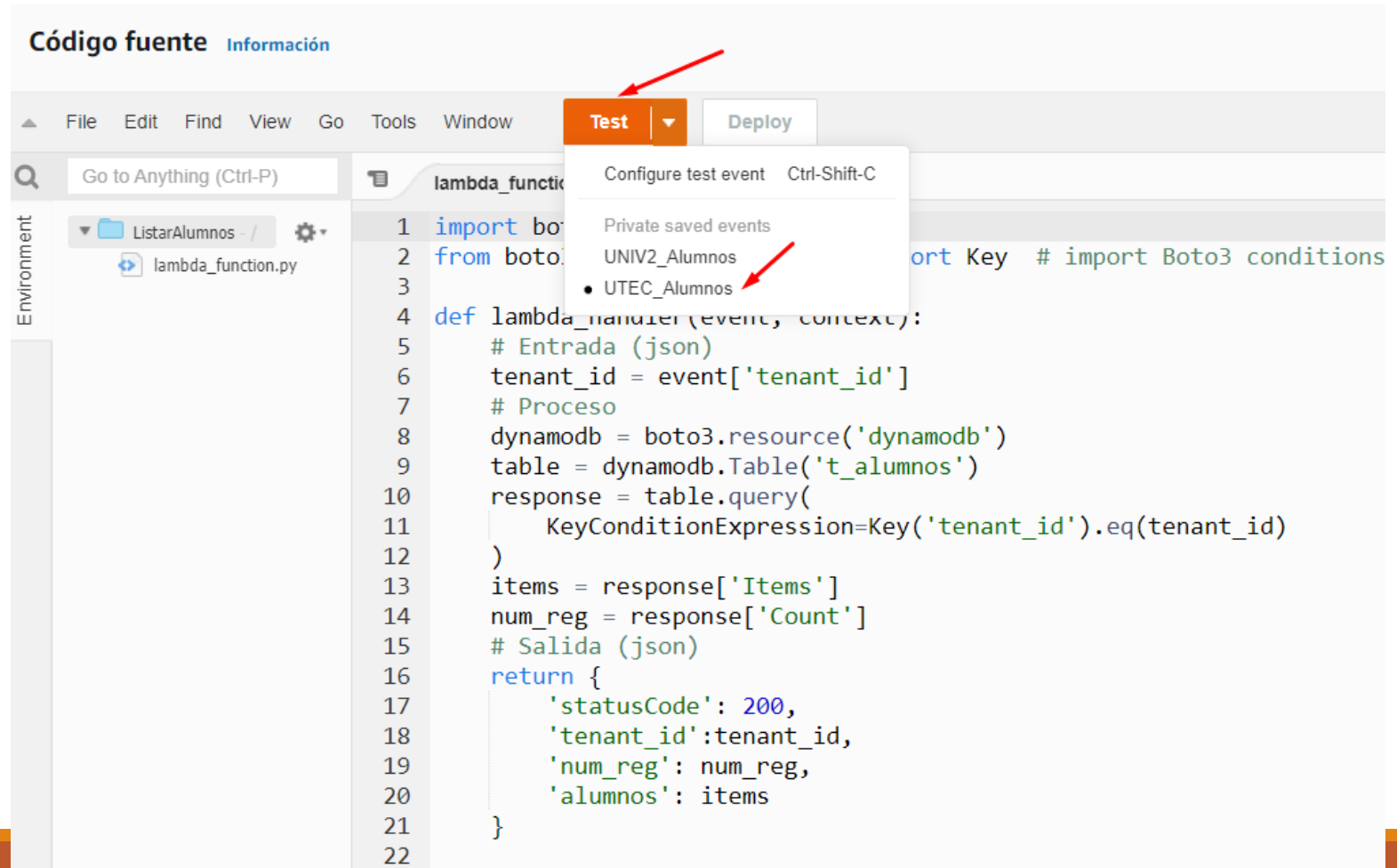
def lambda_handler(event, context):
    # Entrada (json)
    tenant_id = event['tenant_id']
    # Proceso
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('t_alumnos')
    response = table.query(
        KeyConditionExpression=Key('tenant_id').eq(tenant_id)
    )
    items = response['Items']
    num_reg = response['Count']
    # Salida (json)
    return {
        'statusCode': 200,
        'tenant_id': tenant_id,
        'num_reg': num_reg,
        'alumnos': items
    }
```


Ejercicio 1:

Crear una función Lambda - ListarAlumnos

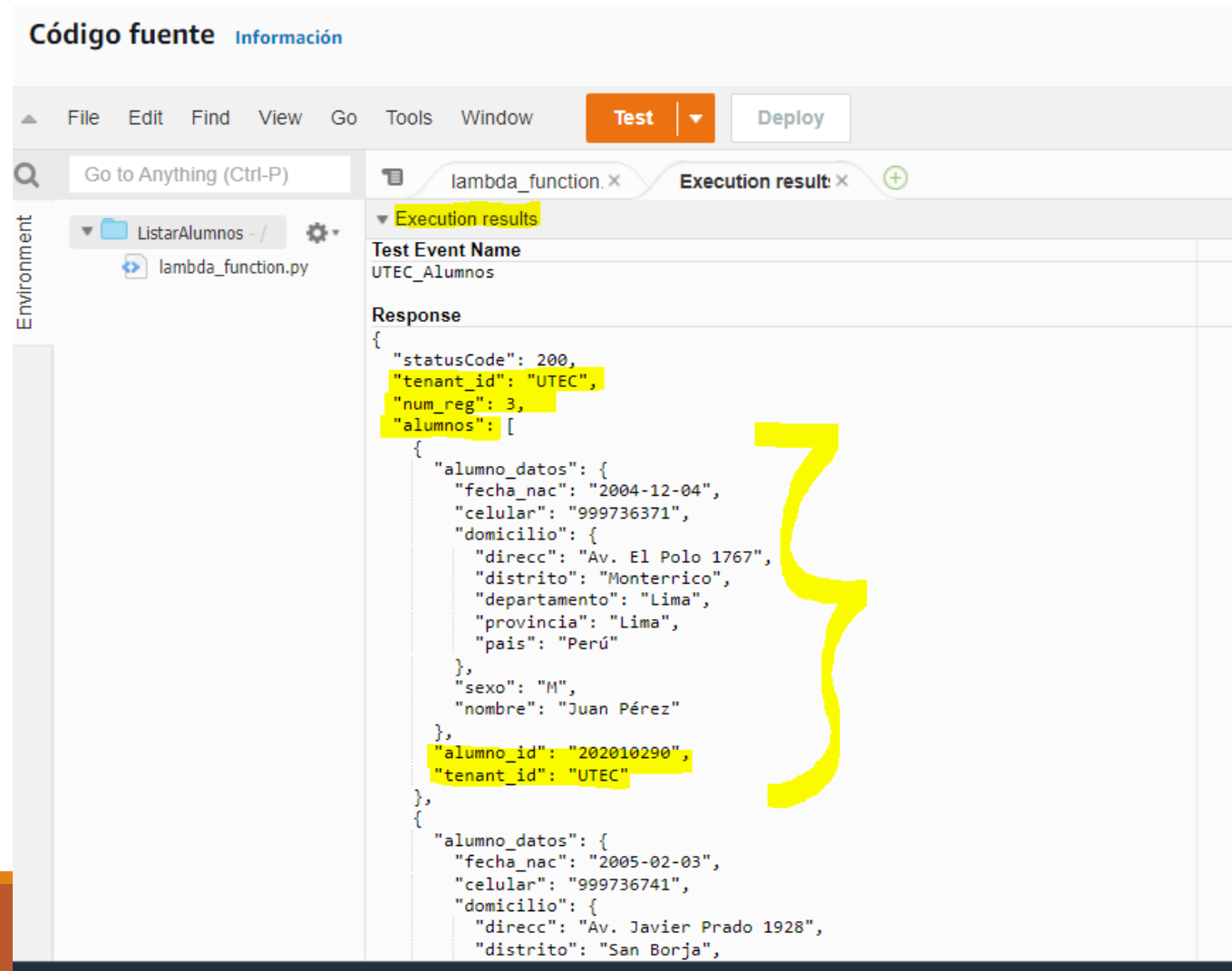
- **Paso 4:** Probar el lambda con esta entrada

```
{  
  "tenant_id": "UTEC"  
}
```



Ejercicio 1:

Crear una función Lambda - ListarAlumnos



The screenshot displays the AWS Lambda console interface. The top navigation bar includes 'Código fuente' and 'Información'. Below this is a menu bar with 'File', 'Edit', 'Find', 'View', 'Go', 'Tools', and 'Window'. A 'Test' button is highlighted in orange, and a 'Deploy' button is visible. The left sidebar shows the 'Environment' section with a folder named 'ListarAlumnos' and a file 'lambda_function.py'. The main area shows the 'Execution results' for a test event named 'UTEC_Alumnos'. The response is a JSON object with the following structure:

```
{
  "statusCode": 200,
  "tenant_id": "UTEC",
  "num_reg": 3,
  "alumnos": [
    {
      "alumno_datos": {
        "fecha_nac": "2004-12-04",
        "celular": "999736371",
        "domicilio": {
          "direcc": "Av. El Polo 1767",
          "distrito": "Monterrico",
          "departamento": "Lima",
          "provincia": "Lima",
          "pais": "Perú"
        },
        "sexo": "M",
        "nombre": "Juan Pérez"
      },
      "alumno_id": "202010290",
      "tenant_id": "UTEC"
    },
    {
      "alumno_datos": {
        "fecha_nac": "2005-02-03",
        "celular": "999736741",
        "domicilio": {
          "direcc": "Av. Javier Prado 1928",
          "distrito": "San Borja",

```

Contenido

Microservicio Multi-tenancy

1. Objetivo del taller 2
2. Ejercicio 1: Crear una función Lambda - ListarAlumnos
3. **Ejercicio 2: Crear una función Lambda - CrearAlumno**
4. Ejercicio 3: Crear una función Lambda - ModificarAlumno
5. Ejercicio 4: Crear una función Lambda - BuscarAlumno
6. Ejercicio 5: Crear una función Lambda - EliminarAlumno
7. Ejercicio 6: Publicar microservicio como API Rest CRUD con servicio Api Gateway
8. Ejercicio 7: Ejercicio propuesto
9. Cierre

Ejercicio 2:

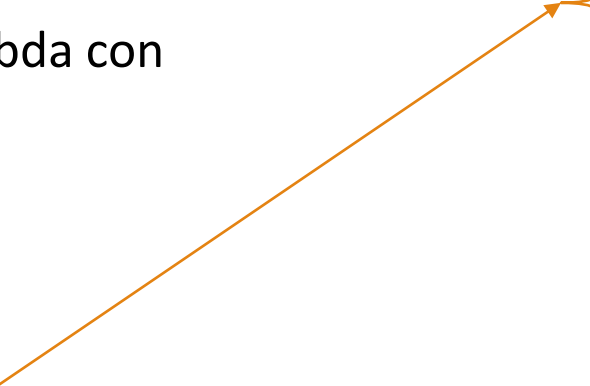
Crear una función Lambda - CrearAlumno

- **Paso 1:** Escribir el siguiente código fuente y “Deploy”
- **Paso 2:** Probar el lambda con esta entrada:

```
{
  "tenant_id": "UTEC",
  "alumno_id": "202099998",
  "alumno_datos": {
    "nombre": "Geraldo Colchado",
    "sexo": "M",
    "fecha_nac": "2000-09-01",
    "celular": "999736332",
    "domicilio": {
      "direcc": "Av. Javier Prado 158",
      "distrito": "San Isidro",
      "provincia": "Lima",
      "departamento": "Lima",
      "país": "Perú"
    }
  }
}
```

```
import boto3
```

```
def lambda_handler(event, context):
    # Entrada (json)
    tenant_id = event['tenant_id']
    alumno_id = event['alumno_id']
    alumno_datos = event['alumno_datos']
    # Proceso
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('t_alumnos')
    alumno = {
        'tenant_id': tenant_id,
        'alumno_id': alumno_id,
        'alumno_datos': alumno_datos
    }
    response = table.put_item(Item=alumno)
    # Salida (json)
    return {
        'statusCode': 200,
        'response': response
    }
```



Contenido

Microservicio Multi-tenancy

1. Objetivo del taller 2
2. Ejercicio 1: Crear una función Lambda - ListarAlumnos
3. Ejercicio 2: Crear una función Lambda - CrearAlumno
4. **Ejercicio 3: Crear una función Lambda - ModificarAlumno**
5. Ejercicio 4: Crear una función Lambda - BuscarAlumno
6. Ejercicio 5: Crear una función Lambda - EliminarAlumno
7. Ejercicio 6: Publicar microservicio como API Rest CRUD con servicio Api Gateway
8. Ejercicio 7: Ejercicio propuesto
9. Cierre

Ejercicio 3:

Crear una función Lambda - ModificarAlumno

- **Paso 1:** Escribir el siguiente código fuente y “Deploy”
- **Paso 2:** Probar el lambda con esta entrada:

```
{
  "tenant_id": "UTEC",
  "alumno_id": "202099998",
  "alumno_datos": {
    "nombre": "Geraldo Colchado Ruiz",
    "sexo": "M",
    "fecha_nac": "2000-09-01",
    "celular": "999736332",
    "domicilio": {
      "direcc": "Av. Javier Prado 158",
      "distrito": "San Borja",
      "provincia": "Lima",
      "departamento": "Lima",
      "pais": "Perú"
    }
  }
}
```

```
import boto3

def lambda_handler(event, context):
    # Entrada (json)
    tenant_id = event['tenant_id']
    alumno_id = event['alumno_id']
    alumno_datos = event['alumno_datos']
    # Proceso
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('t_alumnos')
    response = table.update_item(
        Key={
            'tenant_id': tenant_id,
            'alumno_id': alumno_id
        },
        UpdateExpression="set alumno_datos=:alumno_datos",
        ExpressionAttributeValues={
            ':alumno_datos': alumno_datos
        },
        ReturnValues="UPDATED_NEW"
    )
    # Salida (json)
    return {
        'statusCode': 200,
        'response': response
    }
```

Contenido

Microservicio Multi-tenancy

1. Objetivo del taller 2
2. Ejercicio 1: Crear una función Lambda - ListarAlumnos
3. Ejercicio 2: Crear una función Lambda - CrearAlumno
4. Ejercicio 3: Crear una función Lambda - ModificarAlumno
5. **Ejercicio 4: Crear una función Lambda - BuscarAlumno**
6. Ejercicio 5: Crear una función Lambda - EliminarAlumno
7. Ejercicio 6: Publicar microservicio como API Rest CRUD con servicio Api Gateway
8. Ejercicio 7: Ejercicio propuesto
9. Cierre

Ejercicio 4:

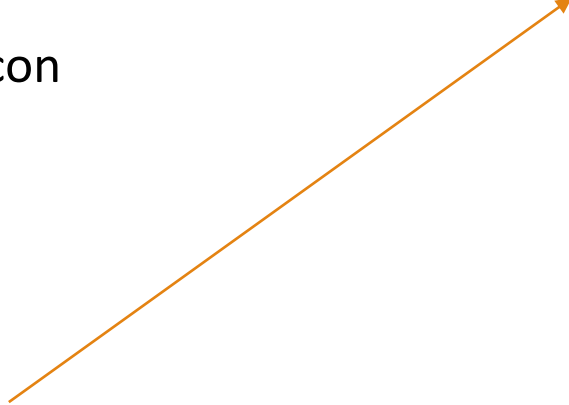
Crear una función Lambda - BuscarAlumno

- **Paso 1:** Escribir el siguiente código fuente y “Deploy”
- **Paso 2:** Probar el lambda con esta entrada:

```
{  
  "tenant_id": "UTEC",  
  "alumno_id": "202099998"  
}
```

```
import boto3
```

```
def lambda_handler(event, context):  
    # Entrada (json)  
    tenant_id = event['tenant_id']  
    alumno_id = event['alumno_id']  
    # Proceso  
    dynamodb = boto3.resource('dynamodb')  
    table = dynamodb.Table('t_alumnos')  
    response = table.get_item(  
        Key={  
            'tenant_id': tenant_id,  
            'alumno_id': alumno_id  
        }  
    )  
    # Salida (json)  
    return {  
        'statusCode': 200,  
        'response': response  
    }
```



Contenido

Microservicio Multi-tenancy

1. Objetivo del taller 2
2. Ejercicio 1: Crear una función Lambda - ListarAlumnos
3. Ejercicio 2: Crear una función Lambda - CrearAlumno
4. Ejercicio 3: Crear una función Lambda - ModificarAlumno
5. Ejercicio 4: Crear una función Lambda - BuscarAlumno
6. **Ejercicio 5: Crear una función Lambda - EliminarAlumno**
7. Ejercicio 6: Publicar microservicio como API Rest CRUD con servicio Api Gateway
8. Ejercicio 7: Ejercicio propuesto
9. Cierre

Ejercicio 5:

Crear una función Lambda - EliminarAlumno

- **Paso 1:** Escribir el siguiente código fuente y “Deploy”
- **Paso 2:** Probar el lambda con esta entrada:

```
{  
  "tenant_id": "UTEC",  
  "alumno_id": "202099998"  
}
```

```
import boto3  
  
def lambda_handler(event, context):  
    # Entrada (json)  
    tenant_id = event['tenant_id']  
    alumno_id = event['alumno_id']  
    # Proceso  
    dynamodb = boto3.resource('dynamodb')  
    table = dynamodb.Table('t_alumnos')  
    response = table.delete_item(  
        Key={  
            'tenant_id': tenant_id,  
            'alumno_id': alumno_id  
        }  
    )  
    # Salida (json)  
    return {  
        'statusCode': 200,  
        'response': response  
    }
```

Contenido

Microservicio Multi-tenancy

1. Objetivo del taller 2
2. Ejercicio 1: Crear una función Lambda - ListarAlumnos
3. Ejercicio 2: Crear una función Lambda - CrearAlumno
4. Ejercicio 3: Crear una función Lambda - ModificarAlumno
5. Ejercicio 4: Crear una función Lambda - BuscarAlumno
6. Ejercicio 5: Crear una función Lambda - EliminarAlumno
7. **Ejercicio 6: Publicar microservicio como API Rest CRUD con servicio Api Gateway**
8. Ejercicio 7: Ejercicio propuesto
9. Cierre

Ejercicio 6:

Servicio Api Gateway (Serverless)

Amazon API Gateway

Cree, conserve y proteja las API a cualquier escala

Empezar a usar Amazon API Gateway

Amazon API Gateway es un servicio completamente administrado que facilita a los desarrolladores la creación, la publicación, el mantenimiento, el monitoreo y la protección de API a cualquier escala. Las API actúan como la "puerta de entrada" para que las aplicaciones accedan a los datos, la lógica empresarial o la funcionalidad de sus servicios de backend. Con API Gateway, puede crear API RESTful y API WebSocket que permiten aplicaciones de comunicación bidireccional en tiempo real. API Gateway admite cargas de trabajo en contenedores y sin servidor, así como aplicaciones web.

API Gateway gestiona todas las tareas implicadas en la aceptación y el procesamiento de hasta cientos de miles de llamadas a API simultáneas, entre ellas, la administración del tráfico, compatibilidad con CORS, el control de autorizaciones y acceso, la limitación controlada, el monitoreo y la administración de versiones de API. API Gateway no requiere pagos mínimos ni costos iniciales. Se paga por las llamadas a las API que se reciben y por la cantidad de datos salientes transferidos; además, con el modelo de precios por niveles de API Gateway, puede reducir sus costos a medida que cambie la escala de uso de las API.

1 millón de llamadas al API recibidas de forma gratuita

por mes, durante 12 meses con la [capa gratuita de AWS](#)

[Comience de forma gratuita »](#)

Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

- **Paso 1:** Crear API Rest en servicio Api Gateway

API REST

Desarrolle una API REST en la que obtenga control total de la solicitud y la respuesta, junto con las capacidades de administración de la API.

Funciona con lo siguiente:
Lambda, HTTP, servicios de AWS

ImportarCrear

Elegir el protocolo

Seleccione si desea crear una API de REST o una API de WebSocket.

☒ REST ☐ WebSocket

Crear API nueva

En Amazon API Gateway, una API de REST hace referencia a una colección de recursos

☒ API nueva ☐ Importar de Swagger u Open API 3 ☐ A

Configuración

Elija un nombre o una descripción fáciles de recordar para su API.

Nombre de API*

alumnos

Descripción

Tipo de punto de enlace

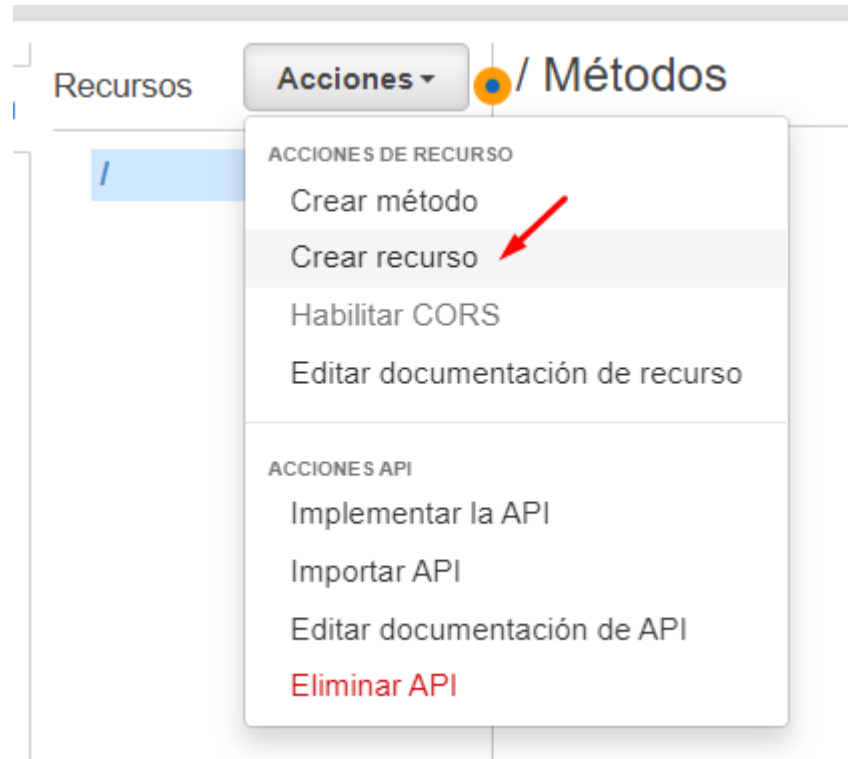
Regional

Crear API

Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

- **Paso 2:** Crear recurso alumnos



Nuevo recurso secundario

Utilice esta página para crear un nuevo recurso secundario para su recurso .

Configurar como [recurso de proxy](#)

Nombre del recurso*

☐ ⓘ

alumnos

Ruta de recurso*

/ alumnos

Puede agregar parámetros de ruta utilizando llaves. Por ejemplo, la ruta de recurso `{username}` representa un parámetro de ruta denominado "username". Al configurar `/ {proxy+}` como un recurso de proxy, se recuperan todas las solicitudes en sus recursos secundarios. Por ejemplo, funciona en una solicitud GET a `/foo`. Para controlar las solicitudes a `/`, agregue un nuevo método ANY en el recurso `/`.

Habilitar API Gateway CORS

☒ ⓘ

* Obligatorio

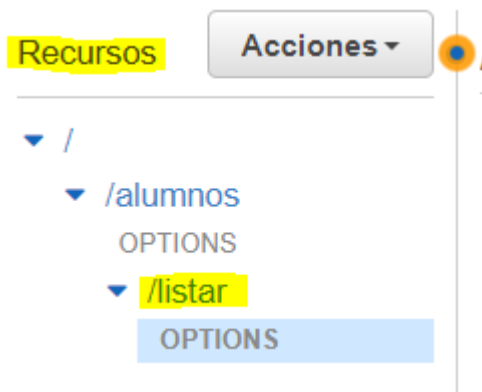
Cancelar

Crear recurso

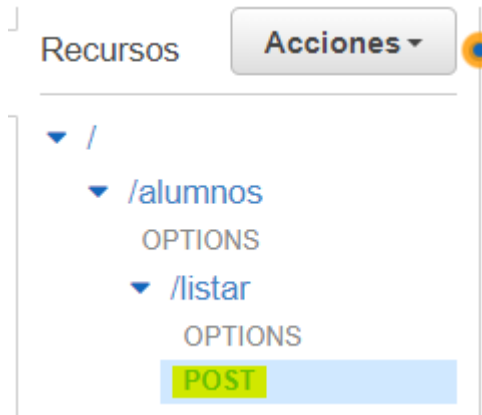
Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

- **Paso 3:** Crear recurso listar



- **Paso 4:** Crear método POST



- **Paso 5:** Asociar a Lambda ListarAlumnos

/alumnos/listar - POST - Configuración

Elija el punto de integración del nuevo método.

Tipo de integración ☒ **Función Lambda** ⓘ

☐ HTTP ⓘ

☐ Simulación ⓘ

☐ Servicio de AWS ⓘ

☐ Enlace de VPC ⓘ

Usar la integración de proxy Lambda ☐ ⓘ

Región Lambda **us-east-1** ▼

Función Lambda

ListarAlumnos

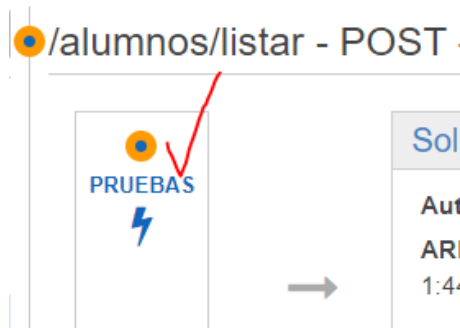
Usar tiempo de espera predeterminado ☒ ⓘ

Guardar

Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

- Paso 6: Probar**



Cuerpo de la solicitud

```
1 {  
2   "tenant_id": "UTEC"  
3 }
```

⚡ Pruebas

Solicitud: /alumnos/listar

Estado: 200

Latencia: 1845 ms

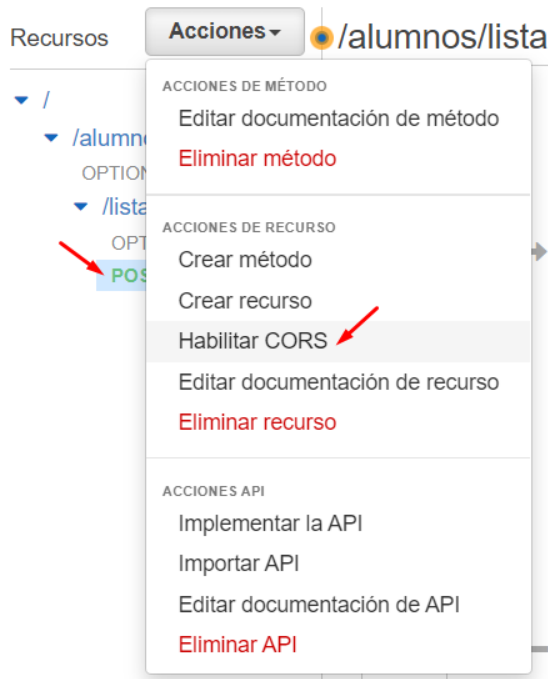
Cuerpo de respuesta

```
{  
  "statusCode": 200, "tenant_id": "UTEC", "num_reg": 4, "alumnos":  
  [{  
    "alumno_datos": {  
      "fecha_nac": "2004-12-04", "celular": "999736371",  
      "domicilio": {  
        "direcc": "Av. El Polo 1767", "distrito": "Monterico",  
        "departamento": "Lima", "provincia": "Lima", "pais": "Per\u0000fa"},  
      "sexo": "M", "nombre": "Juan P\u0000e9rez", "alumno_id": "202010290",  
      "tenant_id": "UTEC"},  
    {  
      "alumno_datos": {  
        "fecha_nac": "2000-09-01", "celular": "999736332",  
        "domicilio": {  
          "direcc": "Av. Javier Prado 158", "distrito": "San Isidro",  
          "departamento": "Lima", "provincia": "Lima", "pais": "Per\u0000fa"},  
        "sexo": "M", "nombre": "Carlos de las Casas", "alumno_id": "202099977",  
        "tenant_id": "UTEC"},  
    {  
      "alumno_datos": {  
        "fecha_nac": "2000-09-01", "celular": "999736332",  
        "domicilio": {  
          "direcc": "Av. Javier Prado 158", "distrito": "San Isidro",  
          "departamento": "Lima", "provincia": "Lima", "pais": "Per\u0000fa"},  
        "sexo": "M", "nombre": "Geraldito Colchado", "alumno_id": "202099998",  
        "tenant_id": "UTEC"}  
  ]  
}
```


Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

- Paso 7: Habilitar CORS**



Habilitar CORS y reemplazar los encabezados CORS existentes

- Paso 8: Implementar la Api**



Implementar la API

Elija una etapa donde se implementará su API. Por ejemplo, una versión de prueba de su API se podría implementar en una etapa denominada beta.

Etapas de implementación [Nueva etapa]

Nombre de la fase* prod

Descripción de etapa

Descripción de implementación

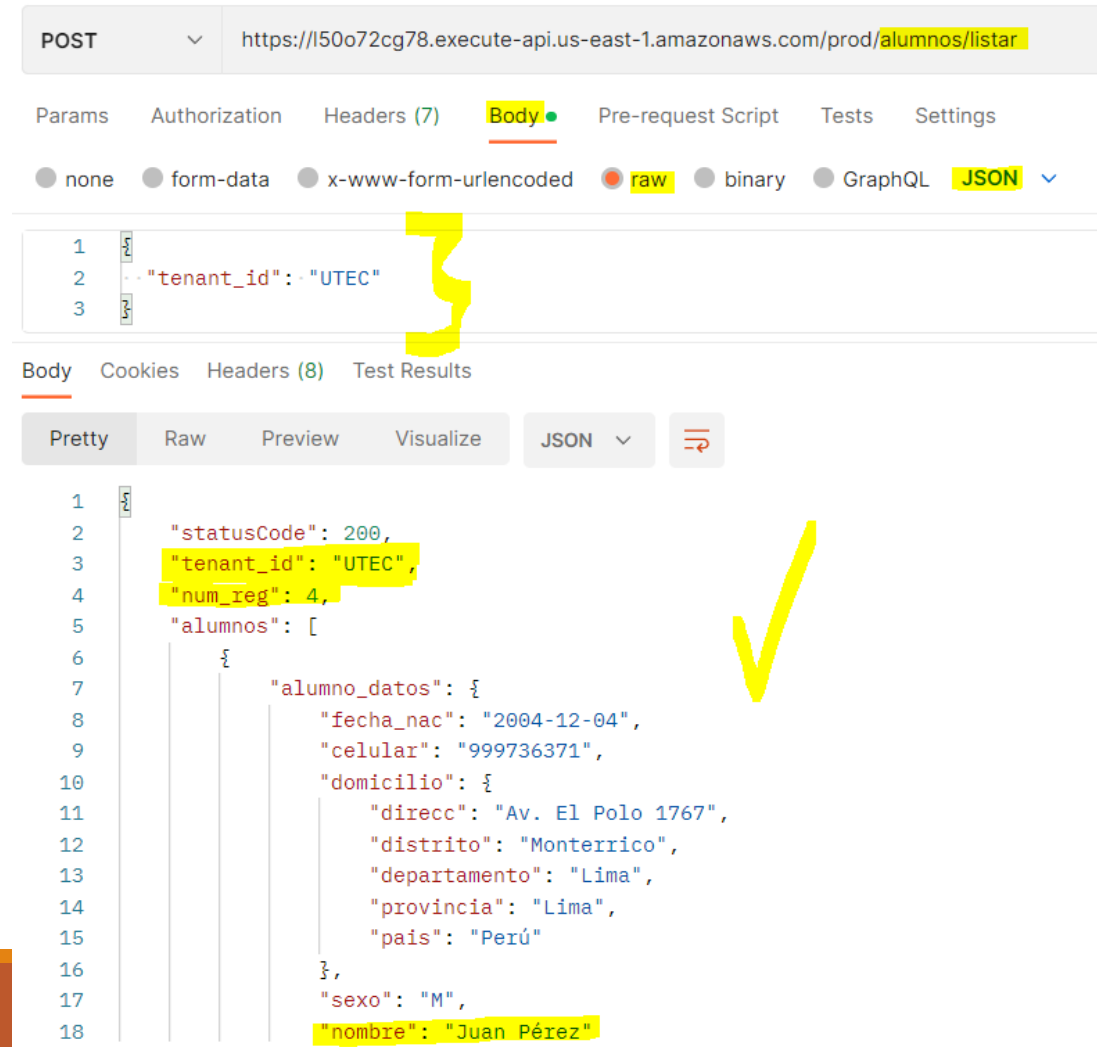
Cancelar **Implementación**

Invocar URL: <https://150o72cg78.execute-api.us-east-1.amazonaws.com/prod>

Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

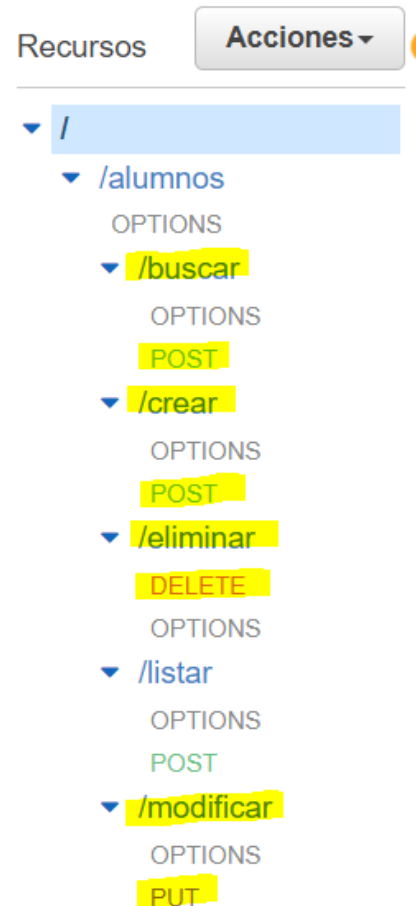
- **Paso 9:** Probar la api ListarAlumnos en postman



Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

- **Paso 10:** Repetir Pasos 3 al 8 para el resto de recursos según esta imagen



Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

- **Paso 11:** Probar la api CrearAlumno en postman

The screenshot shows a Postman interface for a POST request. The URL is `https://f50a72cg78.execute-api.us-east-1.amazonaws.com/prod/alumnos/crear`. The 'Body' tab is selected, and the 'raw' radio button is chosen with 'JSON' as the format. The request body is a JSON object with the following structure:

```
1 {
2   "tenant_id": "UTEC",
3   "alumno_id": "202199998",
4   "alumno_datos": {
5     "nombre": "Geraldo Colchado",
6     "sexo": "M",
7     "fecha_nac": "2000-09-01",
8     "celular": "999736332",
9     "domicilio": {
10      "direcc": "Av. Javier Prado 158",
11      "distrito": "San Isidro",
12      "provincia": "Lima",
13      "departamento": "Lima",
14      "pais": "Perú"
15    }
16  }
17 }
```

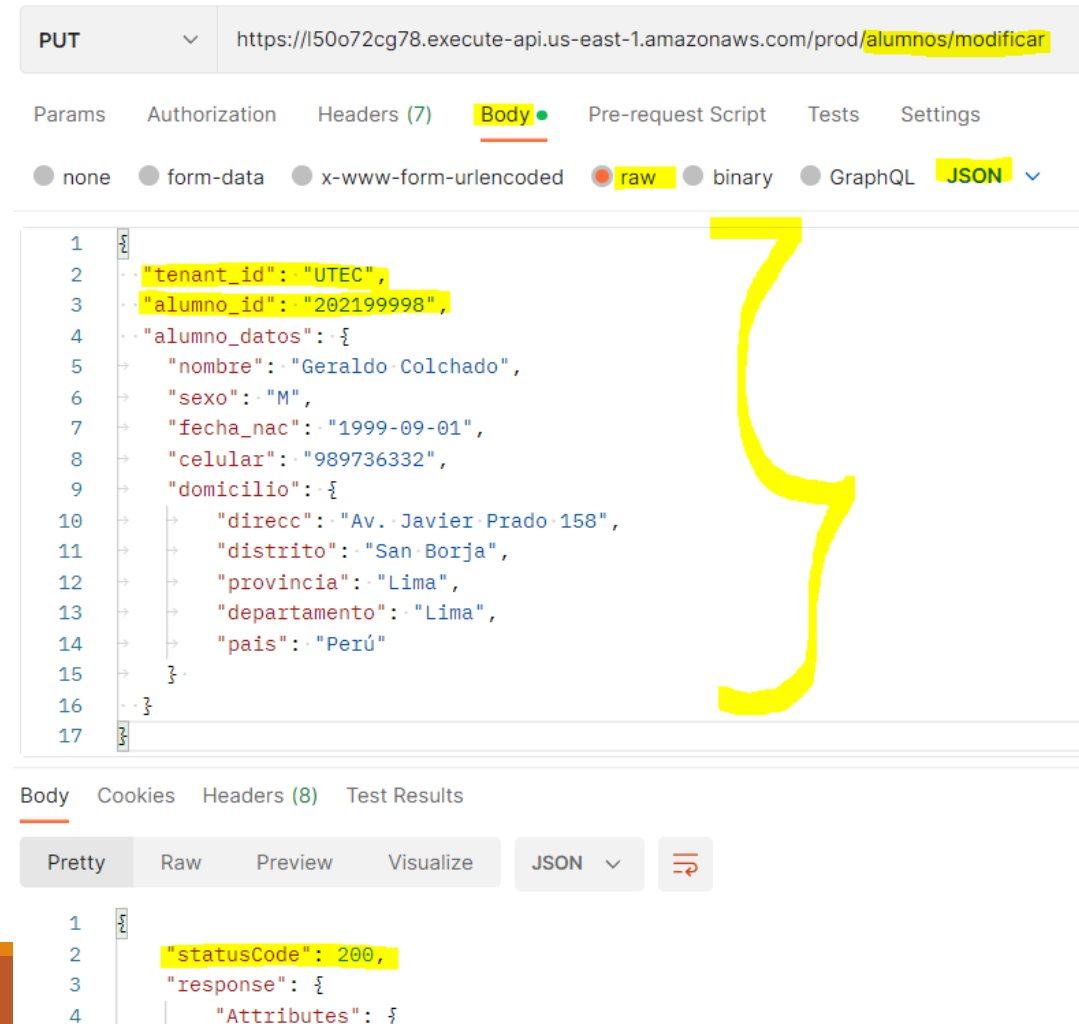
Below the request, the 'Body' tab of the response is shown, displaying the following JSON:

```
1 {
2   "statusCode": 200,
3   "response": {
```

Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

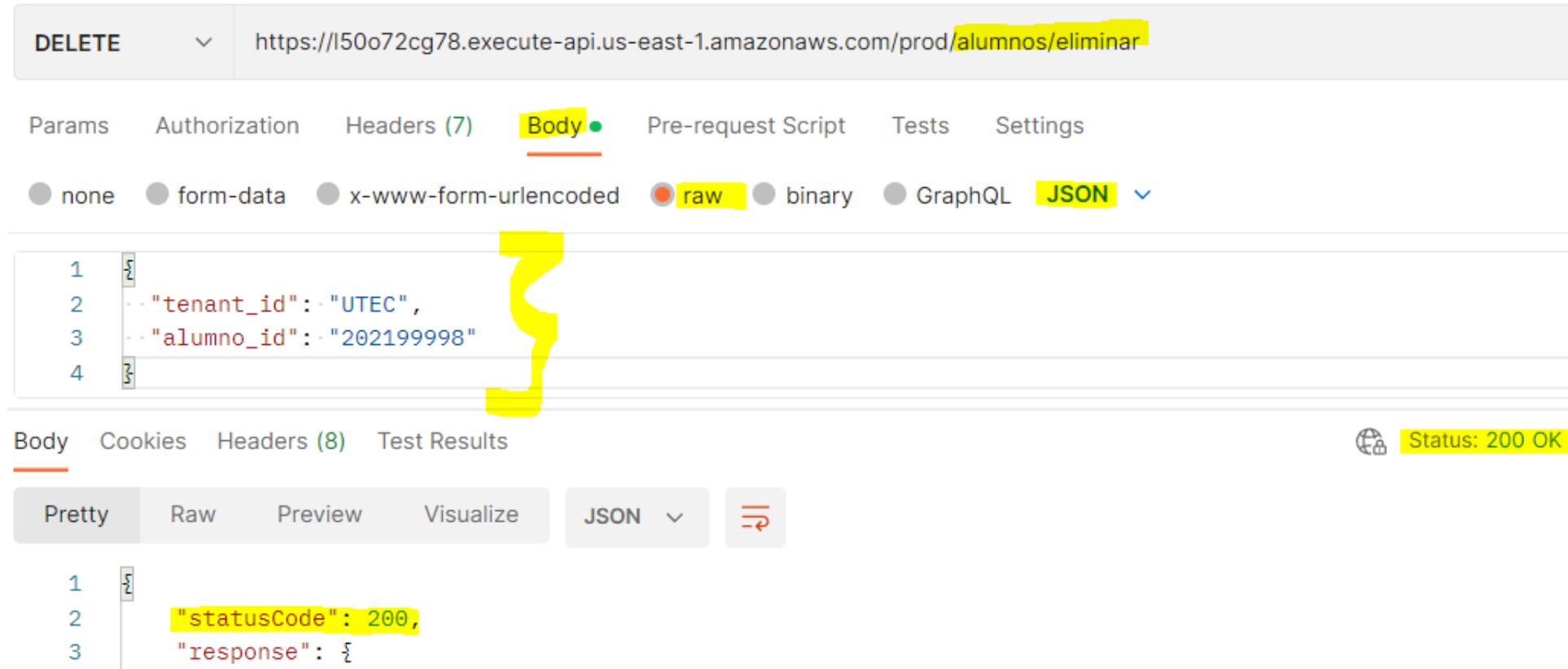
- **Paso 12:** Probar la api ModificarAlumno en postman



Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

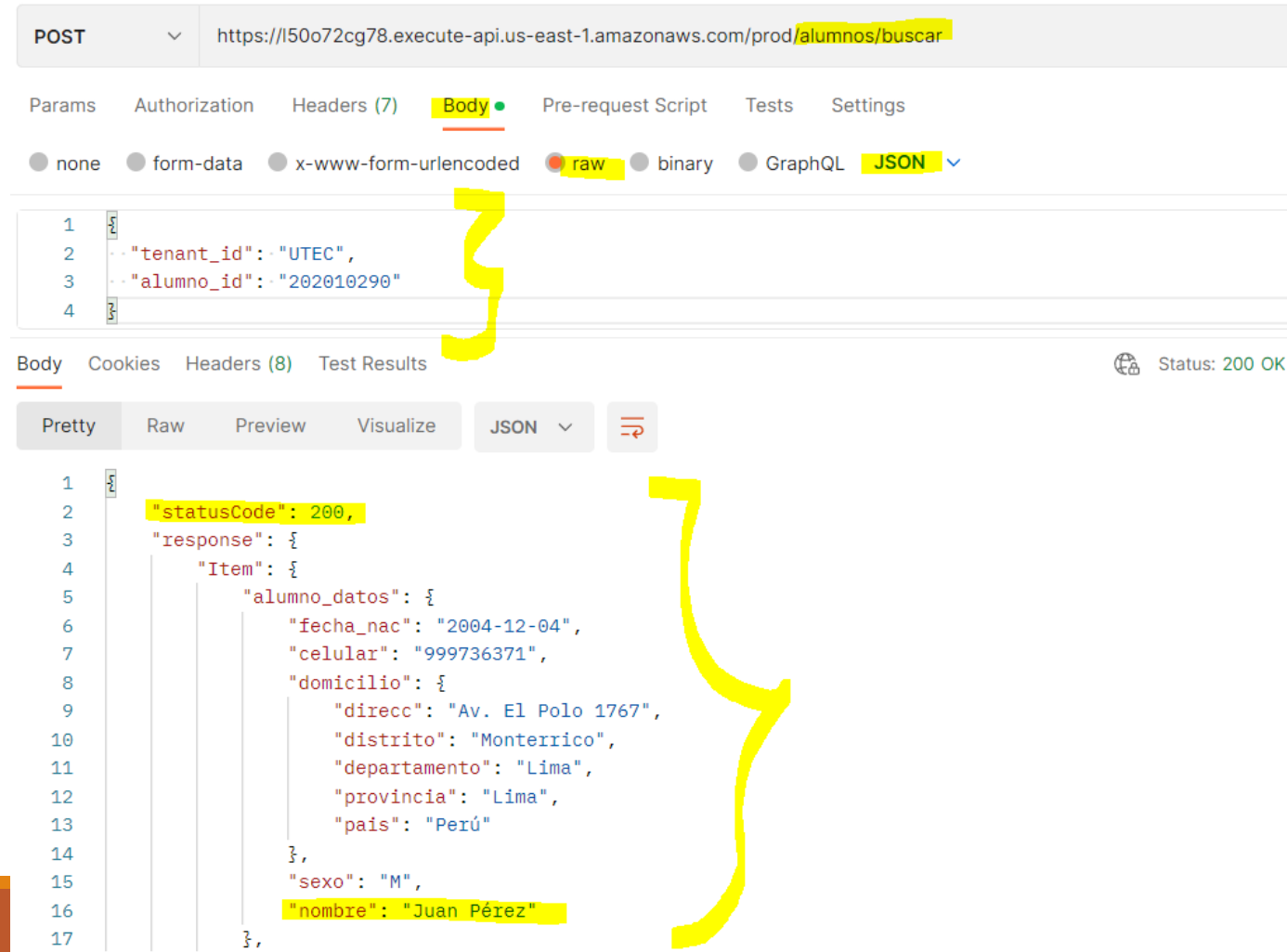
- **Paso 13:** Probar la api EliminarAlumno en postman



Ejercicio 6:

Publicar microservicio como API Rest CRUD con servicio Api Gateway

- **Paso 14:** Probar la api BuscarAlumno en postman



Contenido

Microservicio Multi-tenancy

1. Objetivo del taller 2
2. Ejercicio 1: Crear una función Lambda - ListarAlumnos
3. Ejercicio 2: Crear una función Lambda - CrearAlumno
4. Ejercicio 3: Crear una función Lambda - ModificarAlumno
5. Ejercicio 4: Crear una función Lambda - BuscarAlumno
6. Ejercicio 5: Crear una función Lambda - EliminarAlumno
7. Ejercicio 6: Publicar microservicio como API Rest CRUD con servicio Api Gateway
8. **Ejercicio 7: Ejercicio propuesto**
9. Cierre

Ejercicio 7:

Ejercicio propuesto para la casa

- Implemente **una sola función Lambda** en Python 3 que reciba un campo adicional “action” y en base a ello ejecute las instrucciones para Listar Alumnos y para Crear, Modificar, Eliminar y Buscar un alumno.
- Publicar como Api REST CRUD usando Api Gateway

json de entrada a lambda	Funcionalidad para:
<pre>{ "tenant_id": "UTEC", "action": "listar" }</pre>	Listar alumnos
<pre>... "action": "crear" ...</pre>	Crear alumno
<pre>... "action": "modificar" ...</pre>	Modificar alumno
<pre>... "action": "eliminar" ...</pre>	Eliminar alumno
<pre>... "action": "buscar" ...</pre>	Buscar alumno

Contenido

Microservicio Multi-tenancy

1. Objetivo del taller 2
2. Ejercicio 1: Crear una función Lambda - ListarAlumnos
3. Ejercicio 2: Crear una función Lambda - CrearAlumno
4. Ejercicio 3: Crear una función Lambda - ModificarAlumno
5. Ejercicio 4: Crear una función Lambda - BuscarAlumno
6. Ejercicio 5: Crear una función Lambda - EliminarAlumno
7. Ejercicio 6: Publicar microservicio como API Rest CRUD con servicio Api Gateway
8. Ejercicio 7: Ejercicio propuesto
9. Cierre

Cierre:

Microservicio Multi-tenancy - Qué aprendimos?

- Diseño e implementación de un Microservicio Multi-tenancy con servicio Lambda (Serverless) y Api Gateway (Serverless)

Gracias

Elaborado por docente: Geraldo Colchado