**API Gateway Integration Types**

API Gateway acts as a middleman between your client application and backend services. When a client makes a request, API Gateway routes it to the appropriate backend service and then returns the response. To achieve this, API Gateway uses different integration types depending on the backend you're connecting to. Here's a breakdown of the common ones:

**1. MOCK Integration:**

* This is for testing purposes. API Gateway simply returns a pre-defined response without even calling the backend service.

**2. HTTP/AWS Integration:**

* This is for connecting to various backend options:
  + **HTTP endpoints:** You configure the integration request (what data to send) and integration response (how to handle the response). You can also use mapping templates to transform data between request and response formats (e.g., JSON to XML).
  + **AWS Services:** Similar to HTTP endpoints, you configure request and response but with additional options specific to the AWS service you're integrating with (e.g., SQS queues).

**3. Integration Types for Lambda Functions:**

* These are specifically for invoking AWS Lambda functions:

**AWS\_PROXY Integration (Lambda Proxy):**

* This is the simplest approach for integrating with Lambda functions.
* The entire incoming client request is passed directly to the Lambda function as input.
* The Lambda function is responsible for handling the request logic and generating the response.
* No mapping templates are needed as headers, query strings, etc., are automatically available to the Lambda function.

**HTTP\_PROXY Integration:**

* This type forwards the entire HTTP request received from the client directly to the backend service (like an application load balancer).
* The backend service's response is then forwarded back to the client by API Gateway.
* You can optionally add custom HTTP headers to the request, like an API key for authorization.

**Mapping Templates:**

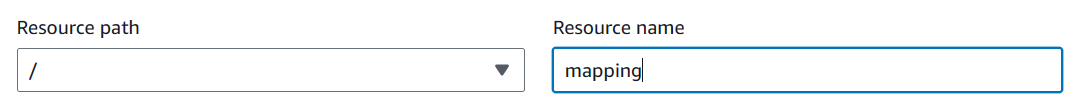
* Both HTTP and AWS integrations can leverage mapping templates to manipulate request and response data.
* These templates use the Velocity Template Language (VTL) and allow you to:
  + Rename or modify data (e.g., query string parameters, body content).
  + Add headers to the request or response.
  + Filter data (remove unnecessary information).
  + Transform data formats (e.g., JSON to XML for SOAP APIs).

**Example: Transforming JSON to XML for a SOAP API:**

* SOAP APIs use XML, while REST APIs typically use JSON.
* Here's how API Gateway with a mapping template can handle this conversion:
  1. Extract data from the client's request (path, payload, or header).
  2. Use the mapping template to build a SOAP message based on the extracted data.
  3. Call the SOAP service and receive the XML response.
  4. Transform the XML response to the desired format (e.g., JSON) using the mapping template again before sending it back to the client.

## API Gateway Mapping Templates Hands-On Guide

### Setting Up the API

1. **Create a Resource:** In your API Gateway console, create a new resource named mapping.  
     
   
2. **Create a GET Method:** Under the mapping resource, create a new GET method.
3. **Create Lambda Function:**

* Choose "Create Lambda Function" during method creation.
* Name the function “api-gateway-mapping-get” and select Python 3.12 as the runtime.
* Within the function code, paste the following:

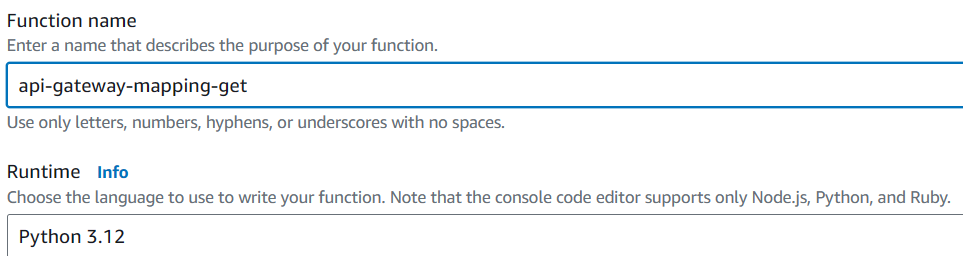
Python

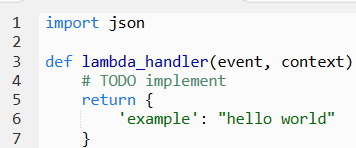
def lambda\_handler(event, context):

return {

'example': "hello world"

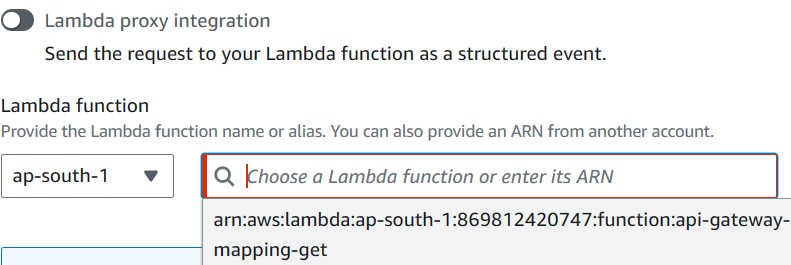
}



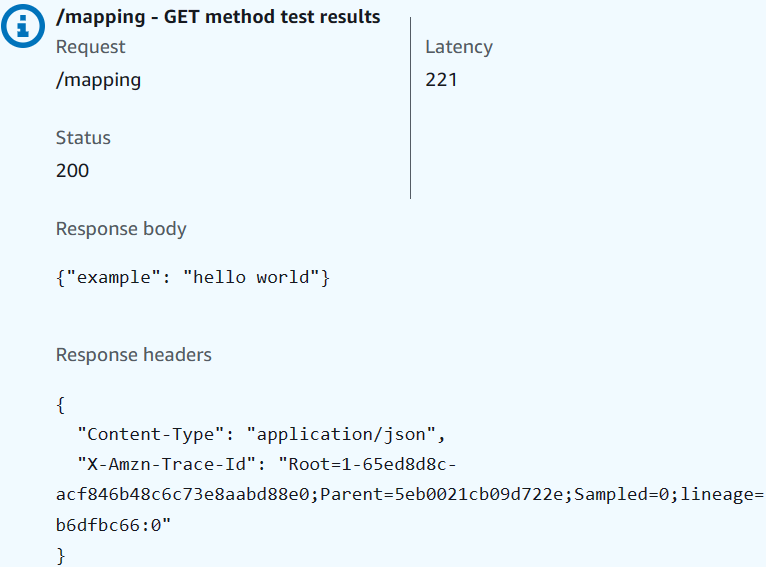


Deploy the code and test it.

1. **Copy Lambda Function ARN:** Locate the ARN (Amazon Resource Name) of the newly created Lambda function. You'll need it for integration.
2. **Integrate with Lambda:**

* Go back to the GET method configuration in API Gateway.
* Under "Integration Type," choose "AWS - Lambda".
* Paste the copied Lambda function ARN into the "Integration target" field.
* **Important:** Disable the "Use Lambda Proxy integration" option.  
    
  

1. **Test the Integration:** In the "Test" tab, send a GET request. You should receive a response with the body "example, hello world" and status code 200.



### **Creating the Mapping Template**

Now, let's modify the response using a mapping template:

1. **Access Integration Response:** In the GET method configuration, navigate to "Integration Response."



1. **Create Mapping Template:** Edit integration response.
2. **Define Content Type:** Set the "Content-Type" to application/json.
3. **Create Template Body:** Paste the following code into the "Template body" field:

JSON

{

"my-key": "my-value",

"renamed-key": $input.json('$.example')

}



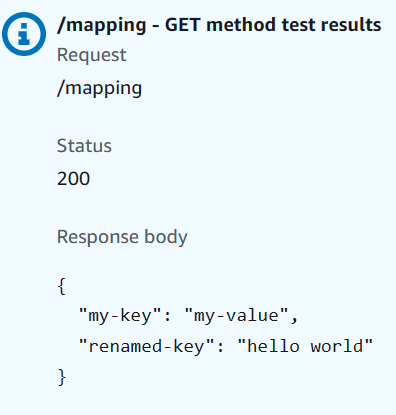
**Explanation:**

* **$input.json('$.example'):** This expression accesses the value of the "example" key within the original JSON payload received from the Lambda function.

1. **Save the Template.**

### Testing with the Template

Now, when you test the GET request again, the response body will be:



The mapping template intercepted the Lambda function's output, added the "my-key" key-value pair, and renamed the "example" key to "renamed-key" with its original value.

**Key Points:**

* Mapping templates allow you to transform data between your API and backend services.
* You can use them on both integration requests and responses.
* They offer functionalities like renaming keys, adding data, and filtering information.

This is a basic example, but it demonstrates the power of mapping templates in shaping your API Gateway responses.