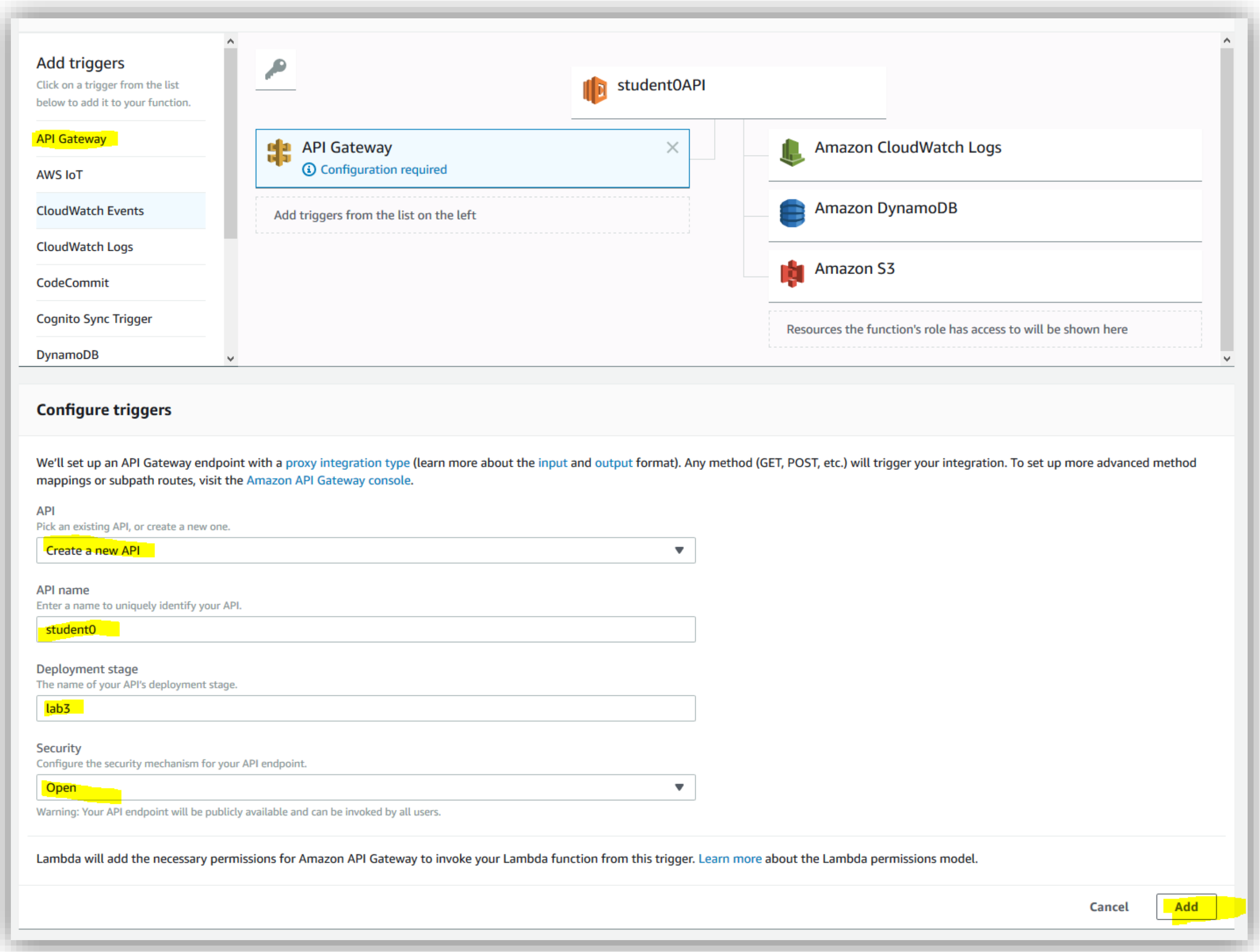
Lab 3: Connect your Lambda function to an API Gateway

Create a new Lambda function named Student#API matching your student number

Select the API Gateway trigger in your Lambda function,

Create a new API, Name it to match your student#, Name the deployment stage “lab3”, Set the security to “Open” and click Add :



Configure triggers

We'll set up an API Gateway endpoint with a proxy integration type (learn more about the input and output format). Any method (GET, POST, etc.) will trigger your integration. To set up more advanced method mappings or subpath routes, visit the Amazon API Gateway console.

API
Pick an existing API, or create a new one.
Create a new API

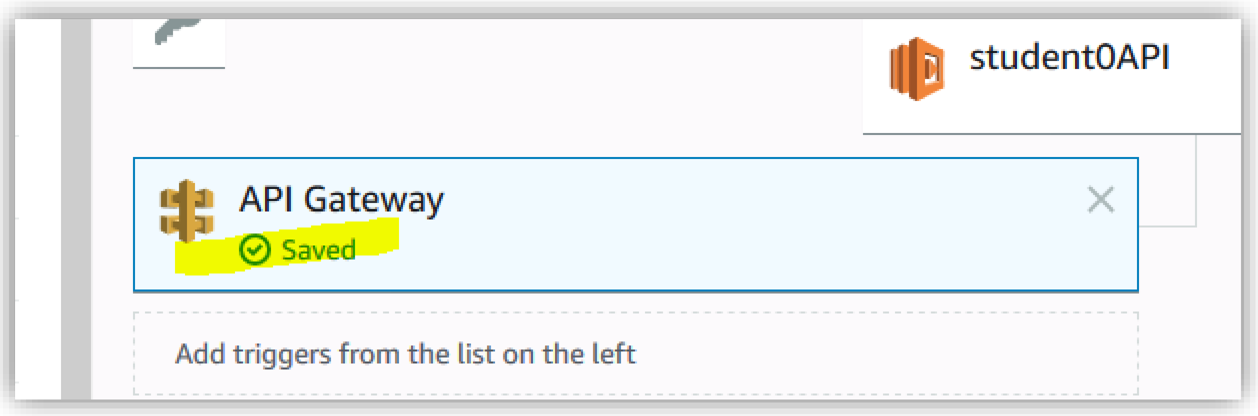
API name
Enter a name to uniquely identify your API.
student0

Deployment stage
The name of your API's deployment stage.
lab3

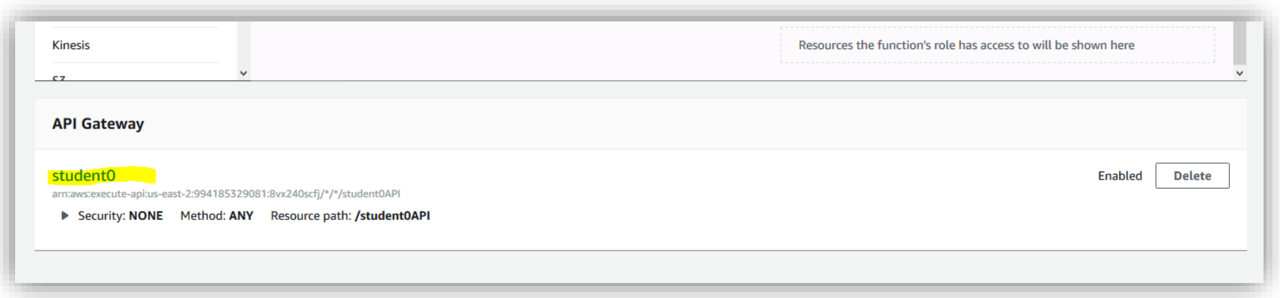
Security
Configure the security mechanism for your API endpoint.
Open
Warning: Your API endpoint will be publicly available and can be invoked by all users.

Lambda will add the necessary permissions for Amazon API Gateway to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

Your API Gateway should be automatically configured once you click save in the upper right hand corner:

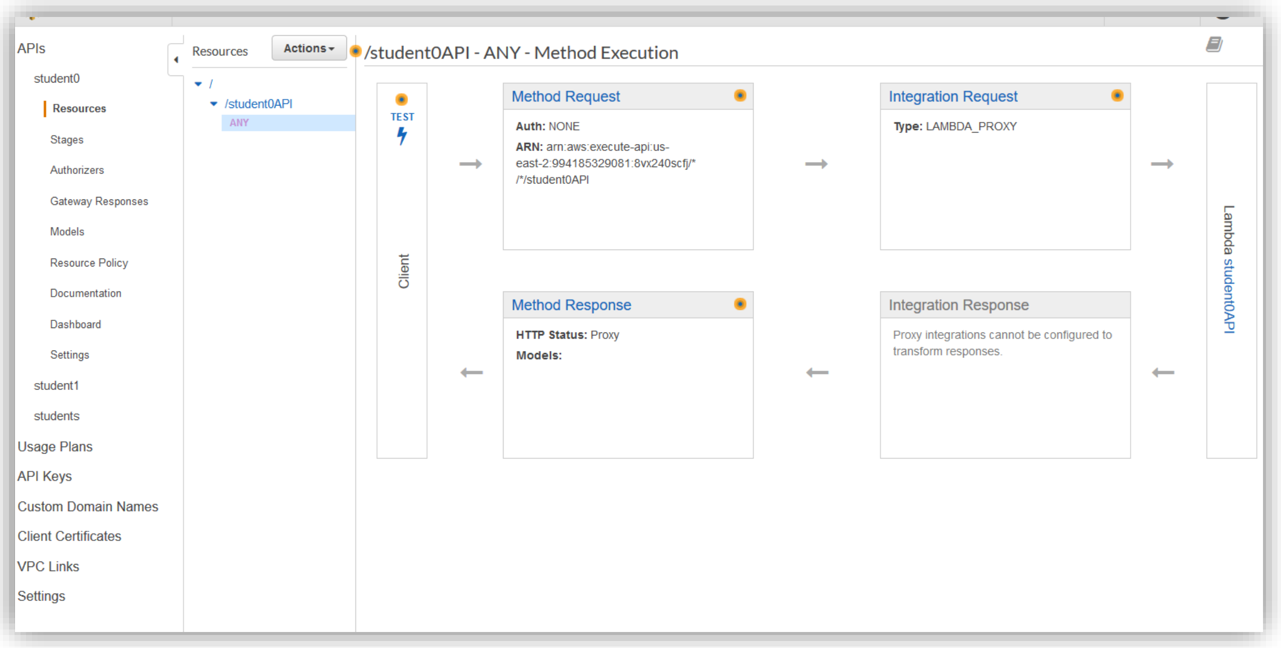


Click on the API Gateway trigger that is attached to the function in the designer, it will have a link matching what you named the gateway. Clicking it will take you to the API Gateway management console:


Kinesis
Resources the function's role has access to will be shown here
API Gateway
Enabled
Delete

student0
arn:aws:execute-api:us-east-2:994185329081:8vx240scfj/*/*/student0API
Security: **NONE** Method: **ANY** Resource path: **/student0API**

In the API management console, you can see the flow and mappings for the requests, and the functions:



/student0API - ANY - Method Execution

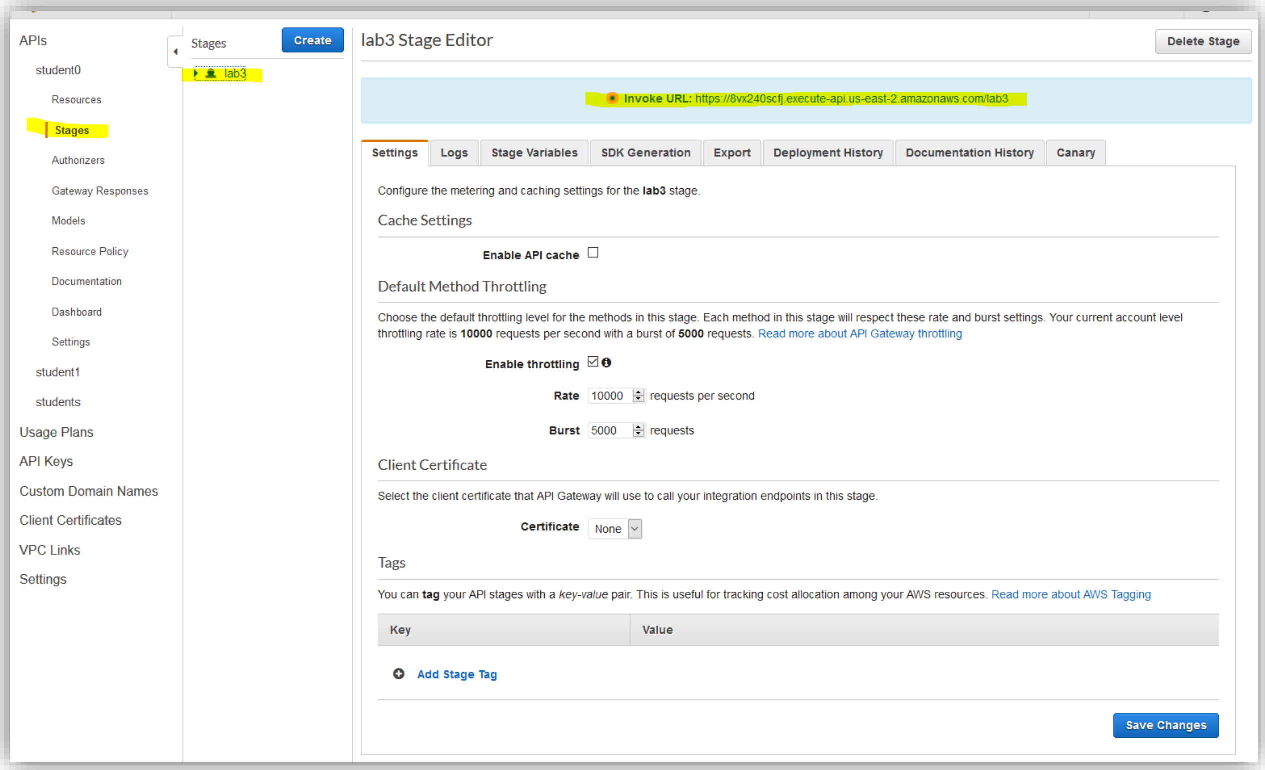
Auth: NONE
ARN: arn:aws:execute-api:us-east-2:994185329081:8vx240scfj/*/*/student0API

Type: LAMBDA_PROXY

HTTP Status: Proxy
Models:

Proxy integrations cannot be configured to transform responses.

To be able to access your API, you will need to find the invocation URL. Click on Stages, then “lab3”. You will see the invocation URL at the top. Click on it:



lab3 Stage Editor

Invoke URL: https://8vx240scfj.execute-api.us-east-2.amazonaws.com/lab3

Configure the metering and caching settings for the **lab3** stage.

Cache Settings

Enable API cache ☐

Default Method Throttling

Choose the default throttling level for the methods in this stage. Each method in this stage will respect these rate and burst settings. Your current account level throttling rate is **10000** requests per second with a burst of **5000** requests. Read more about API Gateway throttling

Enable throttling ☑

Rate 10000 requests per second

Burst 5000 requests

Client Certificate

Select the client certificate that API Gateway will use to call your integration endpoints in this stage.

Certificate None

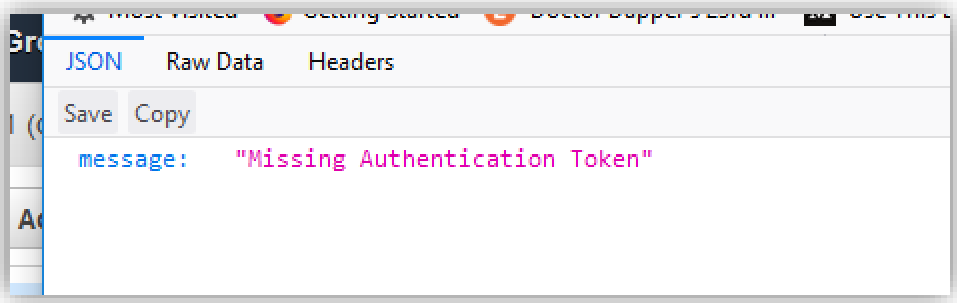
Tags

You can **tag** your API stages with a *key-value* pair. This is useful for tracking cost allocation among your AWS resources. Read more about AWS Tagging

Key	Value

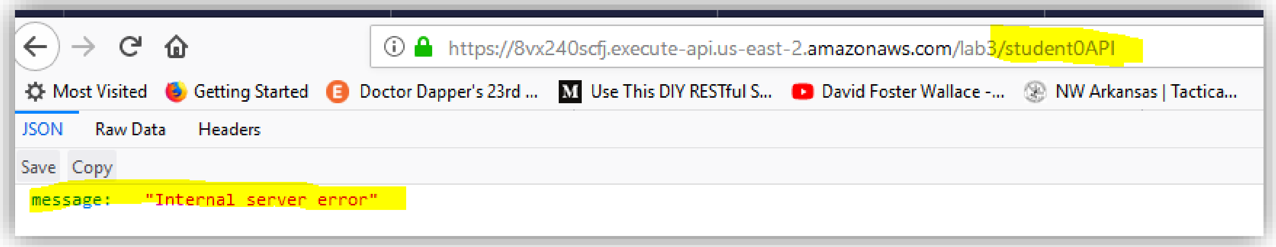
Add Stage Tag

Going to the root URL of an API gateway will give you a not very helpful error that you are missing an auth token. In fact the issue is that you haven’t gone to an actual API endpoint:



```
message:   "Missing Authentication Token"
```

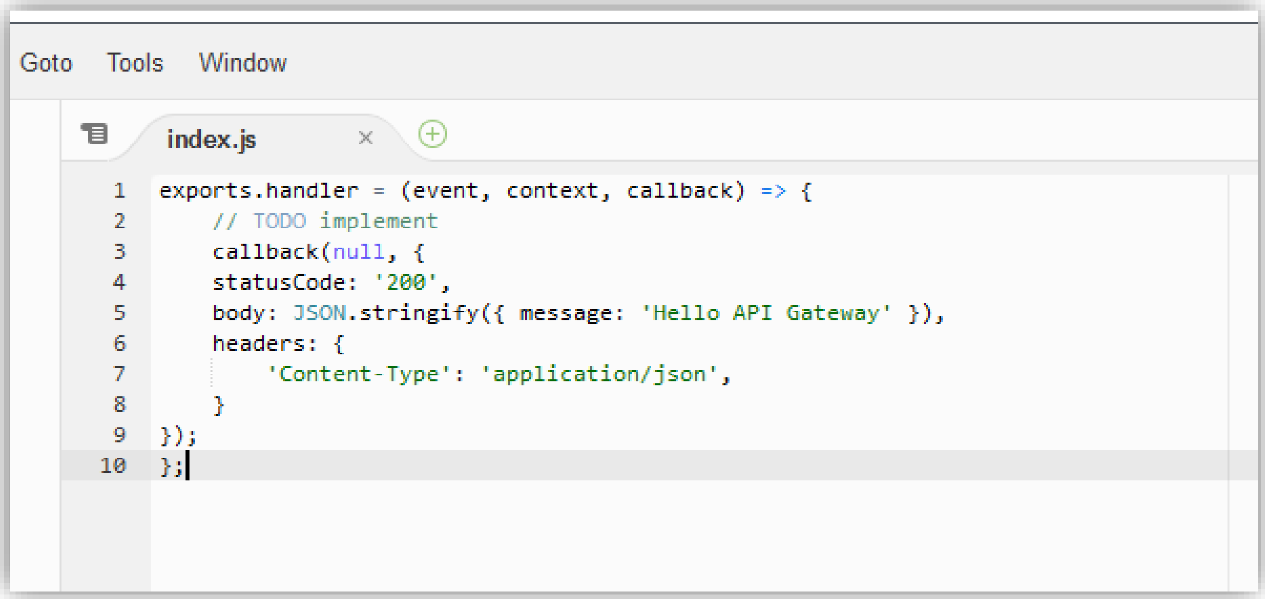
Add /student#API or whatever you named your function to the end of the URL and you should be presented with a new error message:


https://8vx240scfj.execute-api.us-east-2.amazonaws.com/lab3/student0API
Most Visited
Getting Started
Doctor Dapper's 23rd ...
Use This DIY RESTful S...
David Foster Wallace -...
NW Arkansas | Tactica...
JSON
Raw Data
Headers
Save
Copy

```
message:   "Internal server error"
```

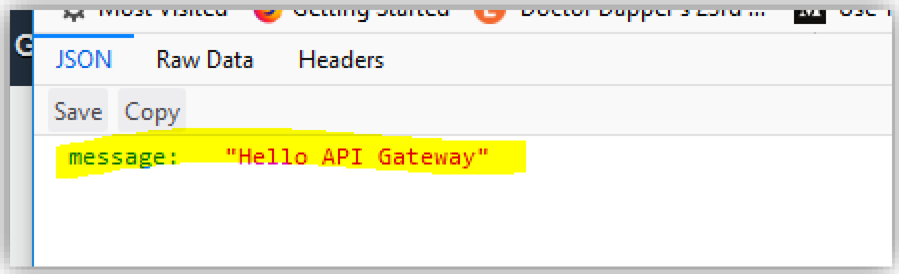
Lambda functions have to return at minimum a status code, otherwise the API gateway defaults to a gateway error 502. Modify our new lambda function to have the following code:

```
exports.handler = (event, context, callback) => {
    const response = {
    statusCode: 200,
    body: JSON.stringify({ message: 'Hello API Gateway' }),
    headers: {
        'Access-Control-Allow-Origin' : '*',
        'Content-Type': 'application/json'
        }
    };
    console.log(response);
    callback(null, response);
};
```



```
exports.handler = (event, context, callback) => {
    // TODO implement
    callback(null, {
    statusCode: '200',
    body: JSON.stringify({ message: 'Hello API Gateway' }),
    headers: {
        'Content-Type': 'application/json',
    }
});
};
```

Save the function, and try hitting the API again:



```
message:    "Hello API Gateway"
```

Congratulations, you now have a Lambda service that is accessible from the outside world.