



School Slack App Documentation

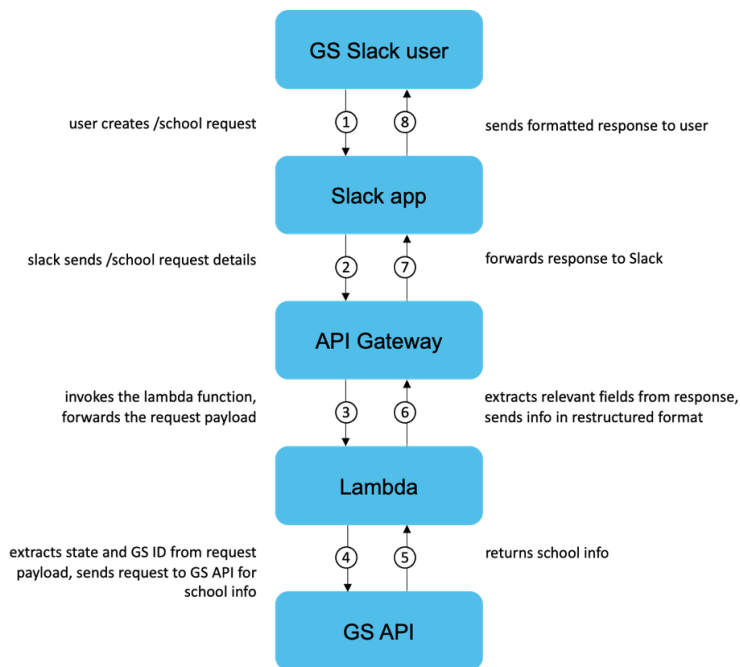
Overview

With the School Slack App, we are now able to get schools' info within one click. All we have to do is type `/school1` and the corresponding GS ID in Slack, and then we would be able to get the name, type, state ID, NCES ID, summary and link of that school!

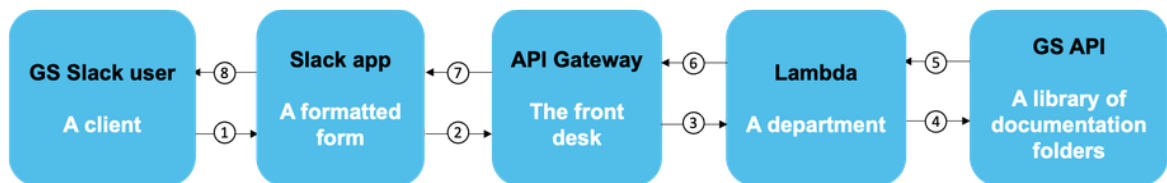
**Serena Huang** 12:32 AM
`/school KS-199`

**omni** APP 12:32 AM
Name: Frankfort High School
Type: public
State ID: 5038
NCES ID: 200456000833
Summary: Frankfort High School, a public school located in Frankfort, KS, serves grades 7-12 & Ungraded in the Vermillion School District. It has received a GreatSchools Rating of 8 out of 10, based on a variety of school quality measures.
Link: <https://www.greatschools.org/kansas/frankfort/199-Frankfort-High-School/>

Below is an overview of how the process works behind the screens:



✓ Easier interpretation of the process



The **client** (GS Slack user) makes a request to get information on a particular school. They write down the ID of the school on a **request form** (Slack app) and hands it over to the **front desk** (API Gateway). The front desk then calls the **relevant department** (Lambda) and tells them the ID of the school. This specific department is an expert at finding schools based on ID, while other departments are skilled at finding different information based on different requests (*thus it depends on which Lambda function we use*). The department then searches up a **specific documentation folder** (GS API) that contains all the information on the schools and its ID, and calls back the **front desk** (API Gateway) with the corresponding information. The front desk writes the information down on a **formatted form** and hands it back to the **client** (GS Slack user).

🔑 Credentials

Before we get started, we first have to set up all of our credentials.

Greatschools API key

To get information from the Greatschools' database, we would require a Greatschools API key. We would have to request access from Anthony, who will provide us with an internal or external administrative key based on the endpoints we want and the estimated monthly volume of requests.

Once we have been granted access, we can use this [documentation](#) to see how the authentication works and which endpoints and parameters to use.

▼ How to use and test our Greatschools API key

We can use the application Postman to test our API key, or we could test it directly through Python when we are building our code.

Testing through Postman

- Download Postman
- Navigate to `My Workspace` → `New` → `HTTP Request`
- In the `Headers` tab, input `x-api-key` under `Key` and the value under `Value`
- In the search bar, make sure the dropdown value is `GET`
- In the search bar, input `https://gs-api.greatschools.org/schools/{universal_id}`, where we put the 7-digit `Universal ID` in the last part of the link. This will request a single school based on the `Universal ID` using the endpoint from the documentation
- Click `Send` and return response

GET <https://gs-api.greatschools.org/schools/2000199> Send

Params Authorization **Headers (7)** Body Pre-request Script Tests Settings Cookies

Key	Value	Description
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.31.3	
<input checked="" type="checkbox"/> Accept	/*/*	
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/> Connection	keep-alive	
<input checked="" type="checkbox"/> x-api-key	XmBWY [REDACTED]	

Body Cookies Headers (7) Test Results 200 OK 112 ms 1.66 KB Save as Example

Pretty Raw Preview Visualize JSON

```

1 {
2   "universal-id": "2000199",
3   "nces-id": "200456000833",
4   "state-id": "5038",
5   "name": "Frankfort High School",
6   "school-summary": "Frankfort High School, a public school located in Frankfort, KS, serves grades
7     7-12 & Ungraded in the Vermillion School District. It has received a GreatSchools Rating of 8 out
8     of 10, based on a variety of school quality measures.",
9   "type": "public",
10  "level-codes": "m,h",
11  "level": "7,8,9,10,11,12,UG",
12  "street": "604 North Kansas Avenue",
13  "city": "Frankfort",
14  "state": "KS",
15  "fipscounty": "20117",
16  "zip": "66427",
17  "phone": "(785) 292-4486",
18  "fax": null,
19  "county": "Marshall County"
20 }
```

Testing through Python

- Input Unique ID (e.g. KS-199)
- Convert Unique ID to Universal ID using our own function `unique_to_universal_id()`
- Request a single school based on Universal ID using the endpoint from the documentation
- Use the API Key given to us
- Return response in json format

```

1 import requests
2
3 def get_school(unique_id):
4     universal_id = unique_to_universal_id(unique_id)
5     response = requests.get(f'https://gs-api.greatschools.org/schools/{universal_id}',
6                             headers = {'x-api-key': 'thiswouldbeourAPIkey'})
7     return response
8
9 print(get_school('KS-199').json())
```

Now we can see the parameters and available information for school KS-199:

```
{
  'universal-id': '2000199', 'nces-id': '20045600833', 'state-id': '5030', 'name': 'Frankfort High School', 'school-summary': 'Frankfort High School, a public school located in Frankfort, KS, serves grades 7-12 & Ungraded in the Vermillion School District. It has received a GreatSchools Rating of 9 out of 10, based on a variety of school quality measures.', 'type': 'public', 'level-codes': 'm,h', 'level': '7,8,9,10,11,12,UG', 'street': '604 North Kansas Avenue', 'city': 'Frankfort', 'state': 'KS', 'fipscounty': '20117', 'zip': '66427', 'phone': '(785) 292-4486', 'fax': None, 'county': 'Marshall County', 'lat': 39.707863, 'lon': -96.41922, 'district-name': 'Vermillion School District', 'district-id': 55, 'web-site': 'http://www.usd380.com', 'overview-url': 'https://www.greatschools.org/kansas/frankfort/199-Frankfort-High-School/', 'rating': '9', 'year': 2020, 'rating-description': 'The GreatSchools Rating helps parents compare schools within a state based on a variety of school quality indicators and provides a helpful picture of how effectively each school serves all of its students. Ratings are on a scale of 1 (below average) to 10 (above average) and can include test scores, college readiness, academic progress, advanced courses, equity, discipline and attendance data. We also advise parents to visit schools, consider other information on school performance and programs, and consider family needs as part of the school selection process.'}
```

AWS Sandbox Engineer Role

To access Lambda and API Gateway, we would need specific credentials. GreatSchools has a Engineer role within the Sandbox account in AWS that we use for this Slack App because it gives us access to the services we need and we don't expect high usage volume. We can get the Sandbox account ID [here](#) but we would need Anthony to assign us the Engineer role in order for us to use it.

Slack

To input commands and receive information in Slack, we will first need to create a Slack app and a slash command. We will be using a few components from Slack to connect with Lambda and API Gateway.

Slack Signing Secret

After we create a slack app, we will be given a unique **Slack Signing Secret**, which can be found in **Basic Information** → **App Credentials**. We will use this later for when we create our Lambda function.

omni

Settings

Basic Information

Collaborators

Socket Mode

Install App

Manage Distribution

Features

App Home

Org Level Apps

Incoming Webhooks

Interactivity & Shortcuts

Slash Commands

Workflow Steps

OAuth & Permissions

Event Subscriptions

User ID Translation

App Manifest **NEW**

Beta Features

App Credentials

These credentials allow your app to access the Slack API. They are secret. Please don't share your app credentials with anyone, include them in public code repositories, or store them in insecure ways.

App ID

Date of App Creation

February 15, 2022

Client ID

Client Secret

.....

Show

Regenerate

You'll need to send this secret along with your client ID when making your [oauth.v2.access](#) request.

Signing Secret

.....

Show

Regenerate

Slack signs the requests we send you using this secret. Confirm that each request comes from Slack by verifying its unique signature.

Verification Token

Regenerate

This deprecated Verification Token can still be used to verify that requests come from Slack, but we strongly recommend using the above, more secure, signing secret instead.

Request URL

When we create a slash command, we would have to input a **Request URL**, which will be the **Invoke URL** in our API Gateway. Details on how to set that up will be in the **API Gateway** section.

API Gateway

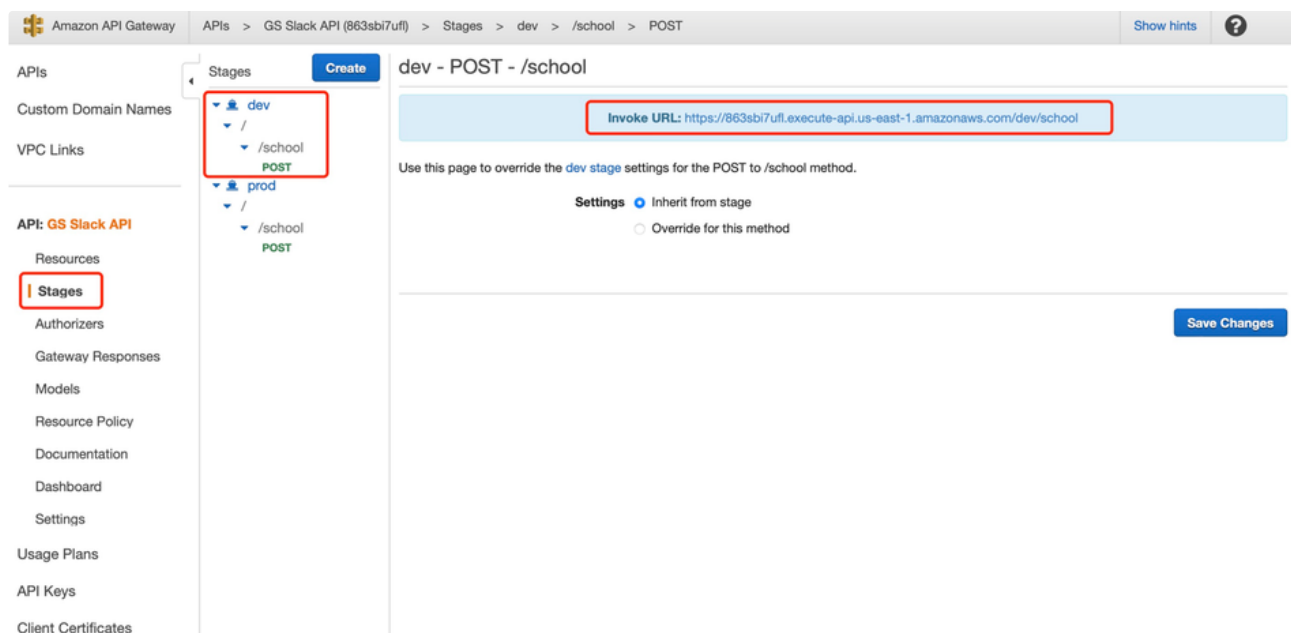
AWS API Gateway is a fully managed service that enables us to create, deploy, and manage APIs at scale. We can create RESTful APIs that connect to backend services running on AWS or any publicly available HTTP endpoint.

Creating APIs, Resources, Methods and Stages

When creating APIs, we will choose the **REST** API, which is an architectural style for building web services that use HTTP requests to access and manipulate data, typically using standard HTTP methods like GET, POST, PUT, and DELETE. Then, for Resources and Methods, we will choose **/school** to be our endpoint and **POST** to be our method. Last, we will create a **dev** stage when testing out new changes, then deploy changes to the **prod** stage when we're ready.

Request URL

Once deployed, we will obtain the **Invoke URL** of the API's endpoints. As we have mentioned before, this will be the **Request URL** for when we create the slash command.



Lambda

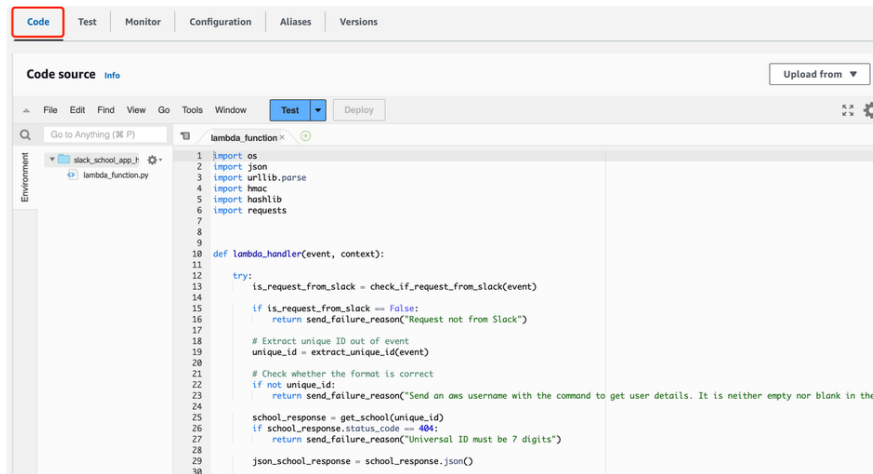
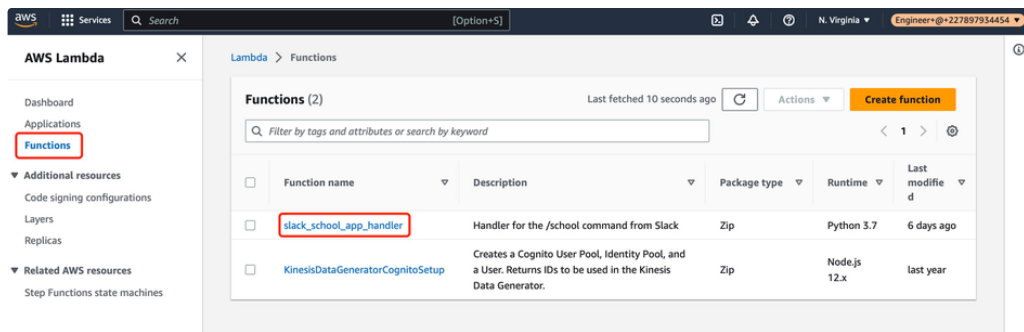
AWS Lambda is a serverless computing service offered by AWS that allows us to run code in response to events or triggers without the need to manage or provision servers. Lambda supports many coding languages, such as Python, Java, etc.

Creating a Lambda function

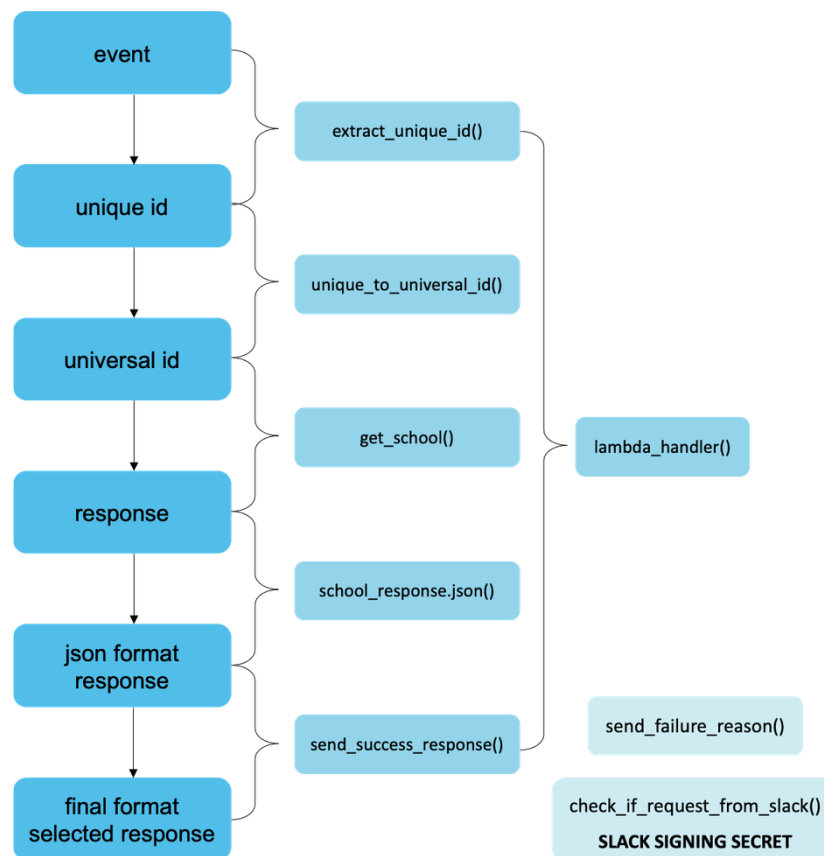
When creating a Lambda function, we will be choosing **Python 3.7** for runtime info instead of Python 3.8, because we use the `requests` Python module to make calls to the GreatSchools API, which only works with Python 3.7 in Lambda. More info can be found [here](#).

Implementing our Python code

Python enables us to handle the text from Slack and convert it to useful information. To input our Python code, we would have to navigate to `Function` → `Code` in the Lambda console.



This is a basic visual of how our code and functions work:



```

1  import os
2  import json
3  import urllib.parse
4  import hmac
5  import hashlib
6  import requests
7
8  def lambda_handler(event, context):
9      try:
10         is_request_from_slack = check_if_request_from_slack(event)
11
12         if is_request_from_slack == False:
13             return send_failure_reason("Request not from Slack")
14
15         # Extract unique ID out of event
16         unique_id = extract_unique_id(event)
17
18         # Check whether the format is correct
19         if not unique_id:
20             return send_failure_reason("Send an aws username with the command to get user details. It is ne
21
22         school_response = get_school(unique_id)
23         if school_response.status_code == 404:
24             return send_failure_reason("Universal ID must be 7 digits")
25
26         json_school_response = school_response.json()
27
28         return send_success_response(json_school_response)
29
30     except Exception as e:
31         print("Error:", e)
32         return send_failure_reason(
33             "Unable to process your request at this time. Please try again later or contact the admin.")
34
35 # Send success response to the Slack user
36 def send_success_response(json_school_response):
37     return {
38         'statusCode': 200,
39         'body': json.dumps({
40             "response_type": "in_channel",
41             "blocks": [
42                 {
43                     "type": "section",
44                     "text": {
45                         "type": "mrkdn",
46                         "text": f"*Name:* {json_school_response['name']}"
47                     }
48                 },
49                 {
50                     "type": "section",
51                     "text": {
52                         "type": "mrkdn",
53                         "text": f"*Type:* {json_school_response['type']}"
54                     }
55                 },
56                 {
57                     "type": "section",
58                     "text": {

```

```

59         "type": "mrkdwn",
60         "text": f"*State ID:* {json_school_response['state-id']}"
61     }
62 },
63 {
64     "type": "section",
65     "text": {
66         "type": "mrkdwn",
67         "text": f"*NCES ID:* {json_school_response['nces-id']}"
68     }
69 },
70 {
71     "type": "section",
72     "text": {
73         "type": "mrkdwn",
74         "text": f"*Summary:* {json_school_response['school-summary']}"
75     }
76 },
77 {
78     "type": "section",
79     "text": {
80         "type": "mrkdwn",
81         "text": f"*Link:* {json_school_response['overview-url']}"
82     }
83 }
84 ]
85 }},
86 'headers': {
87     'Content-Type': 'application/json'
88 }
89 }
90
91 # Send failure reason to the Slack user
92 def send_failure_reason(message):
93     return {
94         'statusCode': 200,
95         'body': json.dumps(
96             {
97                 "response_type": "in_channel",
98                 "text": message
99             }
100         ),
101         'headers': {
102             'Content-Type': 'application/json'
103         }
104     }
105
106 # Parse out command input from the posted message
107 def extract_unique_id(event):
108     body = event['body']
109     command = body.split("text=")[1]
110     command_input = urllib.parse.unquote(command.split("&")[0])
111     return command_input.strip()
112
113 state_fips = {}
114 state_fips['AL'] = '01'
115 state_fips['AK'] = '02'
116 state_fips['AZ'] = '04'

```



```
117 state_fips['AR'] = '05'
118 state_fips['CA'] = '06'
119 state_fips['CO'] = '08'
120 state_fips['CT'] = '09'
121 state_fips['DE'] = '10'
122 state_fips['DC'] = '11'
123 state_fips['FL'] = '12'
124 state_fips['GA'] = '13'
125 state_fips['HI'] = '15'
126 state_fips['ID'] = '16'
127 state_fips['IL'] = '17'
128 state_fips['IN'] = '18'
129 state_fips['IA'] = '19'
130 state_fips['KS'] = '20'
131 state_fips['KY'] = '21'
132 state_fips['LA'] = '22'
133 state_fips['ME'] = '23'
134 state_fips['MD'] = '24'
135 state_fips['MA'] = '25'
136 state_fips['MI'] = '26'
137 state_fips['MN'] = '27'
138 state_fips['MS'] = '28'
139 state_fips['MO'] = '29'
140 state_fips['MT'] = '30'
141 state_fips['NE'] = '31'
142 state_fips['NV'] = '32'
143 state_fips['NH'] = '33'
144 state_fips['NJ'] = '34'
145 state_fips['NM'] = '35'
146 state_fips['NY'] = '36'
147 state_fips['NC'] = '37'
148 state_fips['ND'] = '38'
149 state_fips['OH'] = '39'
150 state_fips['OK'] = '40'
151 state_fips['OR'] = '41'
152 state_fips['PA'] = '42'
153 state_fips['RI'] = '44'
154 state_fips['SC'] = '45'
155 state_fips['SD'] = '46'
156 state_fips['TN'] = '47'
157 state_fips['TX'] = '48'
158 state_fips['UT'] = '49'
159 state_fips['VT'] = '50'
160 state_fips['VA'] = '51'
161 state_fips['WA'] = '53'
162 state_fips['WV'] = '54'
163 state_fips['WI'] = '55'
164 state_fips['WY'] = '56'
165
166 def unique_to_universal_id(unique_id):
167     splitted_unique_id = unique_id.split("-")
168     universal_id = state_fips[splitted_unique_id[0].upper()] + splitted_unique_id[1].zfill(5)
169     return universal_id
170
171 def get_school(unique_id):
172     universal_id = unique_to_universal_id(unique_id)
173     response = requests.get(f'https://gs-api.greatschools.org/schools/{universal_id}',
174                             headers = {'x-api-key': 'thiswouldbeourapikey'})
```

```

175     return response
176
177 # Verify the request is from Slack
178 def check_if_request_from_slack(event):
179     body = event['body']
180     timestamp = event['headers']['X-Slack-Request-Timestamp']
181     slack_signature = event['headers']['X-Slack-Signature']
182     slack_signature_basestring = "v0:" + timestamp + ":" + body
183     slack_signature_hash = "v0=" + hmac.new(os.environ['SLACK_SIGNING_SECRET'].encode(
184         'utf-8'), slack_signature_basestring.encode('utf-8'), hashlib.sha256).hexdigest()
185
186     if not hmac.compare_digest(slack_signature_hash, slack_signature):
187         return False
188     else:
189         return True

```

Slack Signing Secret

As we mentioned before, we have a unique **Slack Signing Secret** from Slack. This verifies requests from Slack by verifying signatures from our signing secret. To add this to Lambda, we would use the Lambda console and navigate to `Function` → `Configuration` → `Environment variables` → `Edit` → `Add environment variable`. Our key would be `SLACK_SIGNING_SECRET` (as from the Python code), while our value would be the generated code from Slack.

CloudWatch

AWS CloudWatch is a monitoring and observability service that allows us to collect, analyze, and act on metrics and logs from various resources within our AWS environment. It is particularly useful in debugging our code in beginning stages.

Viewing and debugging

We can navigate to `Logs` → `Log groups` → `/aws/lambda/slack_school_app_handler` to see the Log Streams. By clicking on the links of the log streams, we will be able to see how the slack request was made and how it responded.

Example of debugging

Example 1: Incorrect parameter name

Under the Message column, we can see that there was an error of 'nces_id', this was because in my Python function `send_success_response()`, I wrote `json_school_response['nces_id']` instead of `json_school_response['nces-id']`. Using 'nces-id' is essential since this is our parameter name from GreatSchool's API.

Timestamp	Message
	There are older events to load. Load more .
2023-04-14T16:46:49.649-05:00	INIT_START Runtime Version: python:3.7.v24 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:db8ce...
2023-04-14T16:46:49.909-05:00	START RequestId: 99f1496b-45c5-421e-a9ea-a8ae5a6fe23a Version: \$LATEST
2023-04-14T16:46:50.237-05:00	Error: 'nces_id'
	Error: 'nces_id' Copy
2023-04-14T16:46:50.265-05:00	END RequestId: 99f1496b-45c5-421e-a9ea-a8ae5a6fe23a
2023-04-14T16:46:50.265-05:00	REPORT RequestId: 99f1496b-45c5-421e-a9ea-a8ae5a6fe23a Duration: 356.22 ms Billed Duration: 357 ms Memo...
	No newer events at this moment. Auto retry paused. Resume

Example 2: Python module is incompatible with Python version

As we mentioned before in the Lambda section, the `request` module we used isn't compatible with Python 3.8, only Python 3.7.

Create Slash Commands using AWS Serverless Services