

TEKLINKS

Code Camp

Lab 4 - Serverless Platforms

Jason Barbee
Solutions Architect
CCIE #18039

Agenda

1. Sign up for Hook.io
2. Build a simple Hook.io endpoint
 1. This task will send an API call to a Spark Room
3. AWS
 1. Sign up for AWS
 2. Install our example API
 3. Test POST
 4. Test GET
 5. Remove the API

Simple Microservices @ Hook.io

Create a Hook.io Account

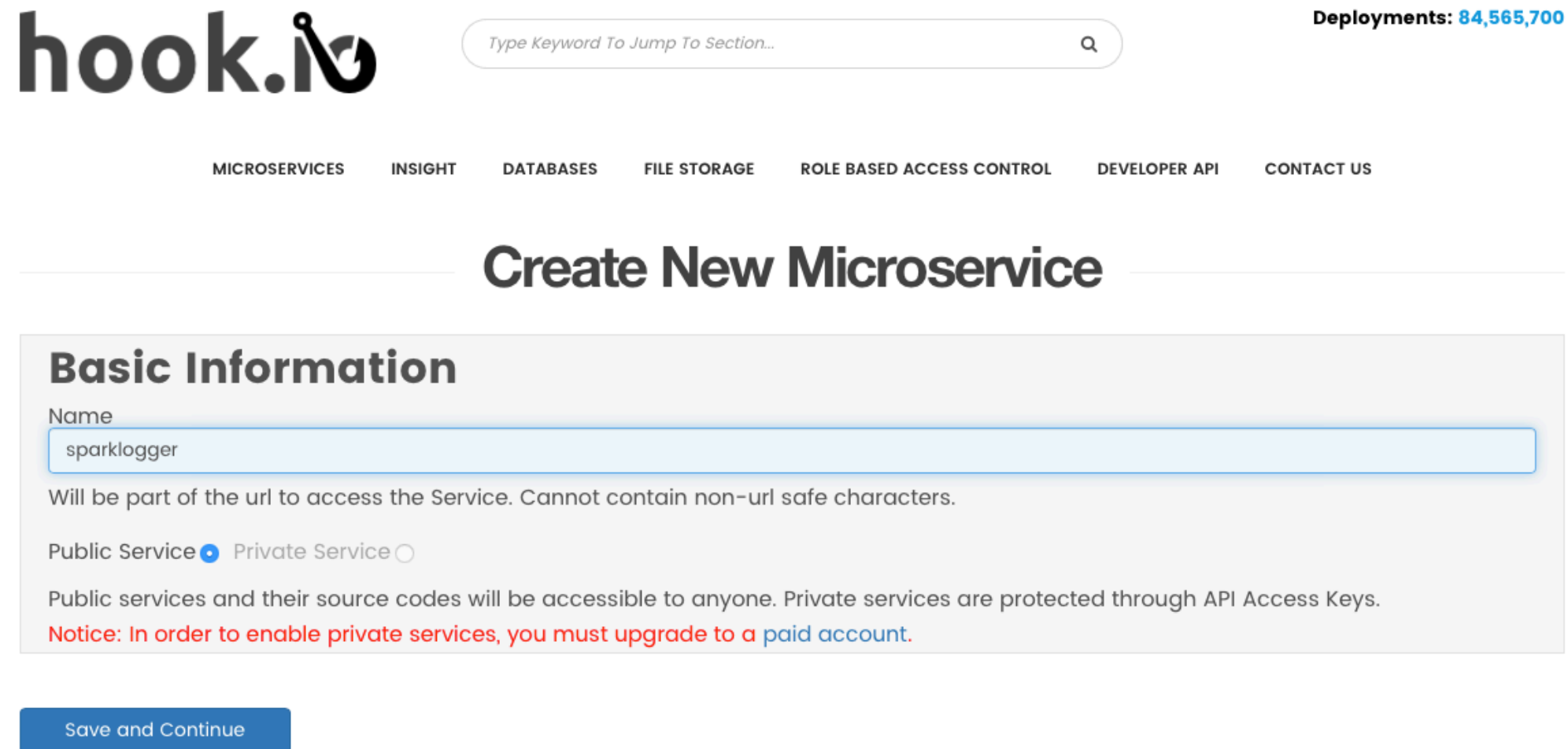
Make a function as a service process to call our Spark Logger.

Sign up for an account at Hook.io

Hook.io will give you a very simple click to start web hook service.

Add a new service at Hook.io

Click Create MicroService at the top navigation bar. - Call it **sparklogger**



The screenshot shows the Hook.io website's 'Create New Microservice' form. At the top, the Hook.io logo is on the left, a search bar with the placeholder 'Type Keyword To Jump To Section...' is in the center, and the text 'Deployments: 84,565,700' is on the right. Below the logo is a navigation bar with links: MICROSERVICES, INSIGHT, DATABASES, FILE STORAGE, ROLE BASED ACCESS CONTROL, DEVELOPER API, and CONTACT US. The main heading is 'Create New Microservice'. The form is titled 'Basic Information' and contains a 'Name' field with the value 'sparklogger'. Below the field is a note: 'Will be part of the url to access the Service. Cannot contain non-url safe characters.' There are two radio buttons: 'Public Service' (selected) and 'Private Service'. Below these is a note: 'Public services and their source codes will be accessible to anyone. Private services are protected through API Access Keys.' A red notice states: 'Notice: In order to enable private services, you must upgrade to a paid account.' At the bottom of the form is a blue button labeled 'Save and Continue'.

hook.io

Type Keyword To Jump To Section...

Deployments: 84,565,700

MICROSERVICES INSIGHT DATABASES FILE STORAGE ROLE BASED ACCESS CONTROL DEVELOPER API CONTACT US

Create New Microservice

Basic Information

Name

sparklogger

Will be part of the url to access the Service. Cannot contain non-url safe characters.

Public Service ☒ Private Service ☐

Public services and their source codes will be accessible to anyone. Private services are protected through API Access Keys.

Notice: In order to enable private services, you must upgrade to a paid account.

Save and Continue

Copy Spark Logging Code

Copy `sparklogger.js` (in the Lab 4 Serverless Folder)

Paste/Save the content into a new hook at Hook.io called "sparklogger")

Your Hook.io URL for should look like this

<https://hook.io/jasonbarbee/sparklogger>

It requires these parameters

- `bottoken` - your authentication bot/person tokens
- `roomid` - the roomID in Spark that you want to post into.
- `message` - the content you want to post

Spark Logger Code

```
module['exports'] = function simpleHttpRequest (hook) {
  // npm modules available, see: http://hook.io/modules
  var request = require('request');
  var botToken= hook.params.bottoken;
  var roomId= hook.params.roomid;
  var text= hook.params.message;
  var body={"roomId": roomId , "text": text};
  var postReq = {
    url: "https://api.ciscospark.com/v1/messages",
    method: "POST",
    headers: {
      'Accepts': 'application/json',
      'Content-type': 'application/json',
      'Authorization': "Bearer " + botToken
    },
    json: body,
  };

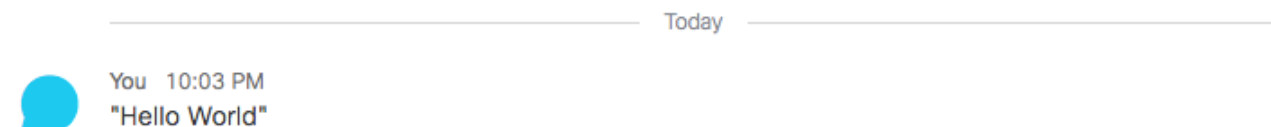
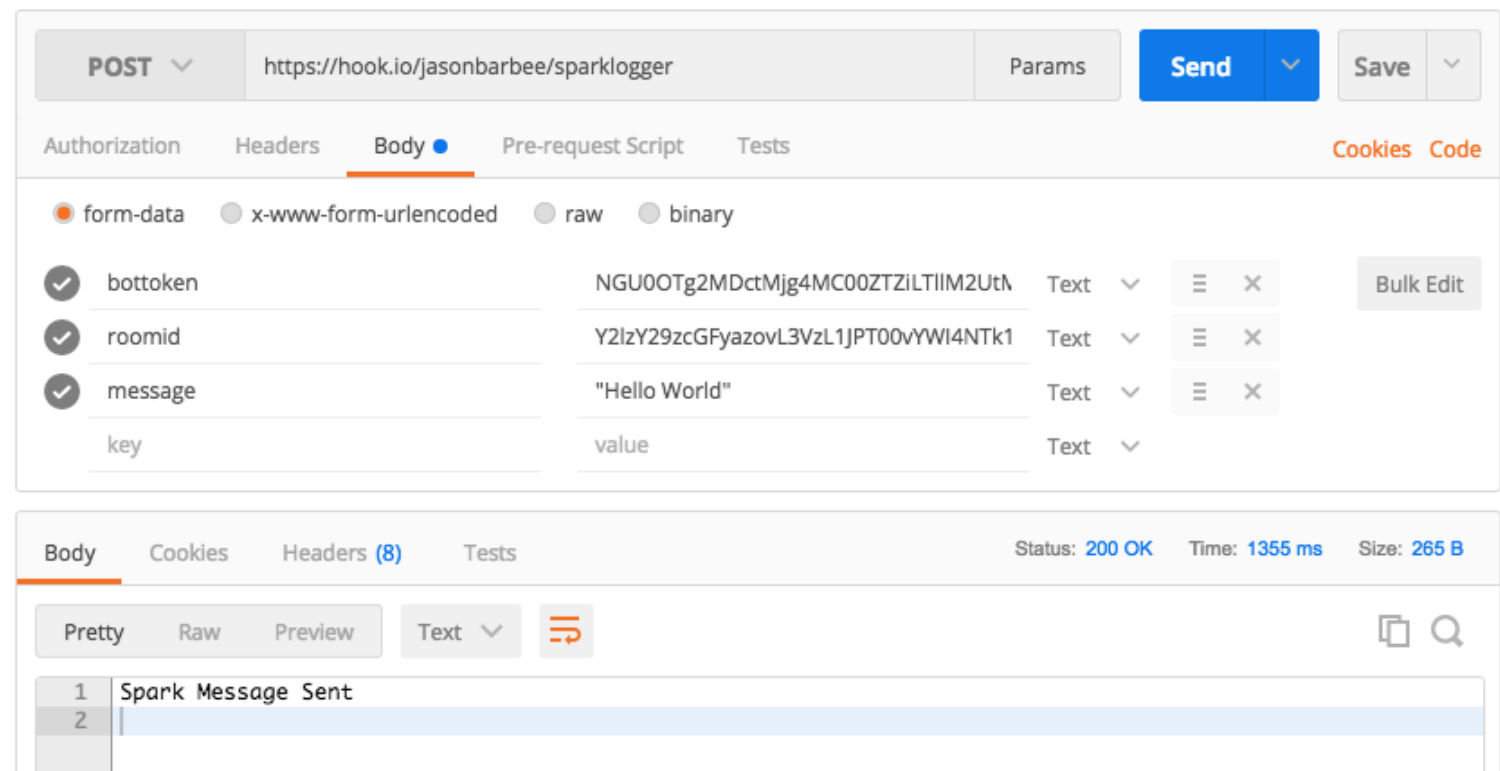
  request.post(postReq, function(err, res, body){

    if (err) {
      console.log("Error", err.message);
      return hook.res.end(err.message);
    }
    //Check for right status code
    if(res.statusCode !== 200){
      console.log('Invalid Status Code Returned:', res.statusCode);
      return hook.res.end("Spark API Error " + res.statusCode);
    }

    //All is good. Print the body
    return hook.res.end("Spark Message Sent");
  });
};
```

Test it in Postman

Now if you call your URL
`https://hook.io/jasonbarbee/sparklogger`
with parameters `bottoken`, `roomid`, `message`
it should post our message...



Making a useful API

Let's make an API that creates, updates, deletes a router inventory.

We will use AWS API Gateway, AWS DynamoDB, and AWS Lambda.

And some Serverless Framework Magic.

Setup an account for Serverless to use your AWS

Create or login to your Amazon Web Services Account and go to the Identity & Access Management (IAM) page.

Click on Users and then Create New Users. Enter a name in the first field to remind you this User is the Framework, like serverless-admin. Then click Create. Later, you can create different IAM Users for different apps and different stages of those apps. That is, if you don't use separate AWS accounts for stages/apps, which is most common.

View and copy the API Key & Secret to a temporary place. You'll need it in the next step.

In the User record in the AWS IAM Dashboard, look for Managed Policies on the Permissions tab and click Attach Policy.

In the next screen, search for and select AdministratorAccess then click Attach.

Vagrant Check

This lab is designed to be run inside the Vagrant profile provided in the Code Camp Repo.

Make sure you have

CD to the vagrant-code-camp folder and run

"vagrant ssh" to access the Vagrant VM.

This Lab 4 is designed to run inside the Serverless Folder within that.

Give Serverless AWS Access

Replace the keys below with your own.

```
serverless config credentials --provider aws --key myawesomekey --secret myawesomesecret
```

Let's deploy our prebuilt API

Change Directory to Serverless example

Code Camp Repo / Vagrant/Serverless

This next step will load the dependency packages to the folder. If you don't do this step it will fail.

```
npm install
```

```
serverless deploy
```

Deployed!

This also shows you your REST endpoint URLs!

```
Serverless: Stack update finished...  
Service Information  
service: serverless-rest-api-with-dynamodb  
stage: dev  
region: us-east-1  
api keys:  
  None  
endpoints:  
  POST - https://3snrpqj7tj.execute-api.us-east-1.amazonaws.com/dev/routers  
  GET - https://3snrpqj7tj.execute-api.us-east-1.amazonaws.com/dev/routers  
  GET - https://3snrpqj7tj.execute-api.us-east-1.amazonaws.com/dev/routers/{id}  
  PUT - https://3snrpqj7tj.execute-api.us-east-1.amazonaws.com/dev/routers/{id}  
  DELETE - https://3snrpqj7tj.execute-api.us-east-1.amazonaws.com/dev/routers/{id}  
functions:  
  serverless-rest-api-with-dynamodb-dev-update: arn:aws:lambda:us-east-1:062829191412:  
function:serverless-rest-api-with-dynamodb-dev-update  
  serverless-rest-api-with-dynamodb-dev-get: arn:aws:lambda:us-east-1:062829191412:fun  
ction:serverless-rest-api-with-dynamodb-dev-get
```

Open the API in your AWS Console

Make sure to choose "N. Virginia" on the top right of AWS console!

Select Amazon API Gateway service.

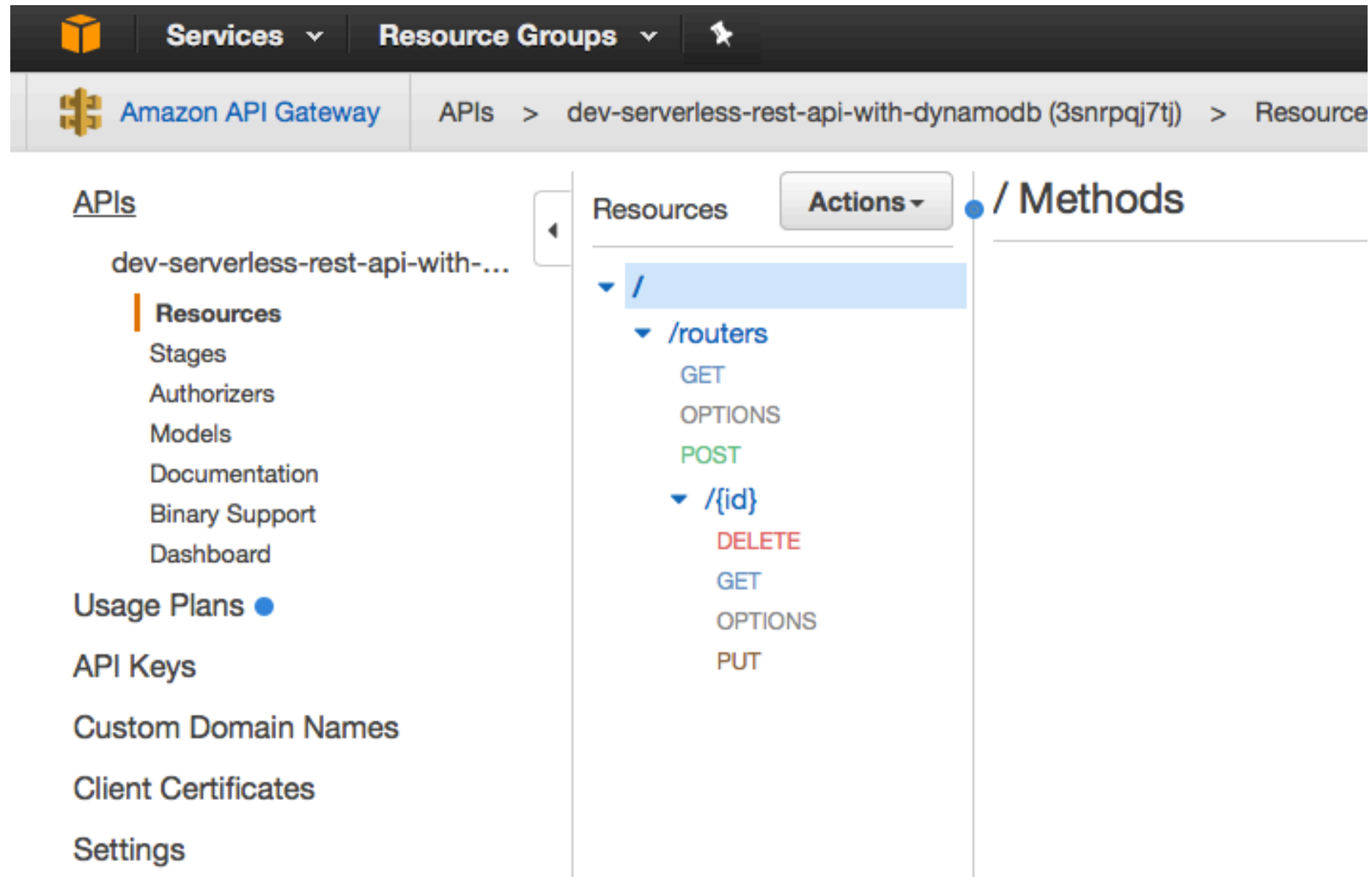
You should see our new service

"dev-serverless-rest-api-with-dynamodb"

API Gateway Screenshot

The screenshot displays the Amazon API Gateway console interface. At the top, a dark navigation bar includes the AWS logo, 'Services' with a dropdown arrow, 'Resource Groups' with a dropdown arrow, a search icon, a notification bell, and user information: 'Jason Barbee - TekLinks', 'N. Virginia', and 'Support' with a dropdown arrow. Below this, a light gray breadcrumb bar shows 'Amazon API Gateway' with a dollar-sign icon and 'APIs', followed by a 'Show all hints' link and a help icon. The main content area features a left-hand sidebar with a vertical orange bar next to the 'APIs' section header. Other sidebar items include 'Usage Plans' (with a blue dot), 'API Keys', 'Custom Domain Names', 'Client Certificates', and 'Settings'. The main panel has a blue 'Create API' button at the top left. Below it, a list of APIs is shown, with 'dev-serverless-rest-api-with-dynamodb' selected and highlighted in blue. The details for this API are displayed in a box, showing the name 'dev-serverless-rest-api-with-dynamodb' and the description 'No description.'

API Methods



The screenshot displays the Amazon API Gateway console interface. At the top, there's a navigation bar with 'Services' and 'Resource Groups' dropdowns. Below this, a breadcrumb trail shows 'Amazon API Gateway' > 'APIs' > 'dev-serverless-rest-api-with-dynamodb (3snrpqj7tj)' > 'Resource'. The main content area is divided into three panes: 'APIs' on the left, 'Resources' in the middle, and 'Methods' on the right. The 'APIs' pane lists various API components like 'Resources', 'Stages', 'Authorizers', 'Models', 'Documentation', 'Binary Support', and 'Dashboard'. The 'Resources' pane shows a tree structure with a root '/' resource expanded, revealing sub-resources '/routers' and '/{id}'. The 'Methods' pane is currently empty, indicating that no methods have been defined for the selected resource. A blue dot in the breadcrumb trail highlights the current location.

Test our API

Click GET and click "TEST"

Amazon API Gateway

APIs > dev-serverless-rest-api-with-dynamodb (3snrpqj7tj) > Resources > /routers (98o8db) > GET

Show all hints ?

APIs

- dev-serverless-rest-api-with-...
- Resources
- Stages
- Authorizers
- Models
- Documentation
- Binary Support
- Dashboard

Usage Plans ●

API Keys

Custom Domain Names

Client Certificates

Settings

Resources

Actions ▾

/routers - GET - Method Execution

/

- /routers
 - GET
 - OPTIONS
 - POST
 - /id
 - DELETE
 - GET
 - OPTIONS
 - PUT

Client

TEST

Test

Method Request

Auth: NONE

ARN: arn:aws:execute-api:us-east-1:062829191412:3snrpqj7tj/*/GET/routers

Integration Request

Type: LAMBDA_PROXY

Method Response

Select an integration response.

Integration Response

Proxy integrations cannot be configured to transform responses.

Test GET Results

Top Right Status 200 is good. Right now there are no results in the right box. It's just a blank JSON object {}
We also get a stack trace of the console logs that happened during the method.

The screenshot shows the Amazon API Gateway console interface. The breadcrumb navigation at the top reads: Amazon API Gateway > APIs > dev-serverless-rest-api-with-dynamodb (3snrpqj7ti) > Resources > /routers (98o8db) > GET. The left sidebar contains navigation links: APIs, Resources, Stages, Authorizers, Models, Documentation, Binary Support, Dashboard, Usage Plans, API Keys, Custom Domain Names, Client Certificates, and Settings. The 'Resources' section is expanded, showing the hierarchy: / > /routers > GET. The main content area is titled 'Method Execution /routers - GET - Method Test'. It includes a 'Test' button at the bottom right. The page is divided into several sections: Path (No path parameters exist for this resource. You can define path parameters by using the syntax {myPathParam} in a resource path.), Query Strings (A text input field contains 'param1=value1¶m2=value2'), Headers (A text area contains 'Accept:application/json.'), Stage Variables (No stage variables exist for this method.), Client Certificate (No client certificates have been generated.), Request Body (Request Body is not supported for GET methods.), Request (Request: /routers), Status (Status: 200), Latency (Latency: 94 ms), Response Body (A text box contains an empty JSON array '[]'), Response Headers (A text box contains '{"X-Amzn-Trace-Id": "Root=1-58802be2-d2e6931d0f46b38a3fe4108d"}'), and Logs (An execution log for request test-request, dated Thu Jan 19 03:00:50 UTC 2017, showing the request details and the endpoint URI).

POST a Router manually

Use the POST/ method - click TEST, and use this as a template the body

```
{  
  "customer" : "Jason",  
  "ip" : "1.1.1.1",  
  "os" : "VyOS",  
  "hostname" : "VyOS Router",  
  "version" : "12.2"  
}
```

You should get Status 200 (OK) - that means it posted correctly to the database.

Let's post some real data to the API

Your GET request will return all the routers in the inventory.
Let's add a router to the database using Ansible.

Build an Ansible Playbook

Use the template "aws-inventory.yml" under the Vagrant/Ansible folder.

We use a method called URI to POST data to a URL after collecting the inventory.

make sure to change this line in aws-inventory.yml

url: "https://3snrpqj7tj.execute-api.us-east-1.amazonaws.com/dev/routers"

to **YOUR URL** reported by Serverless so that the data gets posted to YOUR API (not mine)

Ansible AWS Tasks

tasks:

- name: collect all facts from the device

vyos_facts:

gather_subset: all

provider: "{{ cli }}"

register: result

- name: Write a record to AWS API

uri:

url: "https://3snrpqj7tj.execute-api.us-east-1.amazonaws.com/dev/routers"

method: POST

HEADER_Content-Type: application/json

body: '{

"ip" : "{{ inventory_hostname }}",

"version" : "{{ result.ansible_facts.ansible_net_version }}",

"hostname" : "{{ result.ansible_facts.ansible_net_hostname }}",

"customer" : "{{ customername }}"

}'

body_format: json

validate_certs: no

Run Ansible AWS inventory

ansible-playbook -i inventory aws-inventory.yml

```
vagrant@vagrant:/vagrant/Ansible$ ansible-playbook -i inventory aws-inventory.yml

PLAY [VyOS Inventory Collector] *****

TASK [setup] *****
ok: [35.166.172.203]

TASK [collect all facts from the device] *****
ok: [35.166.172.203]

TASK [Write a record to AWS API] *****
ok: [35.166.172.203]

PLAY RECAP *****
35.166.172.203      : ok=3    changed=0    unreachable=0    failed=0

vagrant@vagrant:/vagrant/Ansible$
```


Query our API/database

On the right we see both entries, our custom posted entry, and the Ansible entry.

← Method Execution

/routers - GET - Method Test

Make a test call to your method with the provided input

Path

No path parameters exist for this resource. You can define path parameters by using the syntax {myPathParam} in a resource path.

Query Strings

{routers}

param1=value1¶m2=value2

Headers

{routers}

Use a colon (:) to separate header name and value, and new lines to declare multiple headers. eg. Accept:application/json.

Stage Variables

No [stage variables](#) exist for this method.

Client Certificate

No client certificates have been generated.

Request: /routers

Status: 200

Latency: 124 ms

Response Body

```
[
  {
    "hostname": "VyOS Router",
    "version": "12.2",
    "os": "VyOS",
    "updatedAt": 1484794975466,
    "ip": "1.1.1.1",
    "createdAt": 1484794975466,
    "customer": "Jason",
    "id": "c64634a0-ddf3-11e6-b422-4185d1ed2616"
  },
  {
    "hostname": "AWS-CodeCamp",
    "version": "VyOS",
    "updatedAt": 1484796176710,
    "ip": "35.166.172.203",
    "createdAt": 1484796176710,
    "customer": "Example Customer",
    "id": "92457e60-ddf6-11e6-b422-4185d1ed2616"
  }
]
```

Caution if you run Ansible more than once, it will upload duplicates.
If you feel like rewriting some code to prevent that - go for it.

End of Lab

Thanks!