

# Documentation

## Back-end\_RAD\_Event\_API\_and\_Database

REST API with one endpoint named: /events

**read:** <https://boee9mgtbe.execute-api.us-east-1.amazonaws.com/events/read>

**delete:** <https://boee9mgtbe.execute-api.us-east-1.amazonaws.com/events/{id}/get?EventID=1>

**get:** <https://boee9mgtbe.execute-api.us-east-1.amazonaws.com/events/{id}/get?EventID=1>

Need to test in API proxy (postman)

**create:** <https://boee9mgtbe.execute-api.us-east-1.amazonaws.com/events/create/{data}?Organizer=Mississippi Chapter of Street Dreams Car club&Venue=Meridian&EventDate=June 13, 2020>

**update:** <https://boee9mgtbe.execute-api.us-east-1.amazonaws.com/events/update/{id}?EventID=1>

Request Body

```
{
  "Organizer": "Mississippi Chapter of Street Dreams Car club",
  "Venue": "Meridian",
  "EventDate": "June 13, 2020"
}
```

## Create RDS MySQL — AWS Serverless to MySQL

1. Create MySQL database
2. Select the **MySQL Database**
3. Select the **Free Tire** for MySQL
4. Setting on the **DB cluster identifier**, **Master Username** and **Master Password**

For your local MySQL:

DB cluster identifier as the database name,

Master Username as Username,

Master Password as password

Endpoint as Hostname

(After the Database is created, you will get an EndPoint at RDS Database

Connectivity & security)

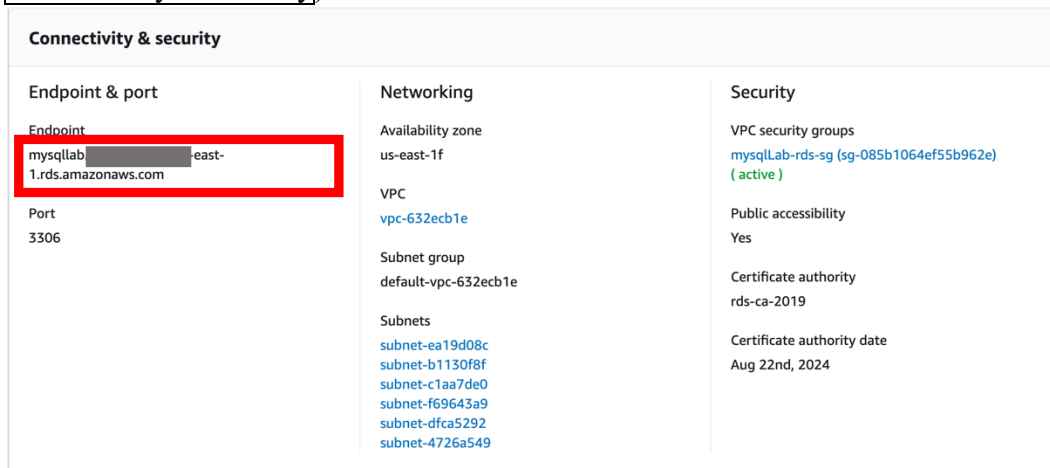


Fig. 1

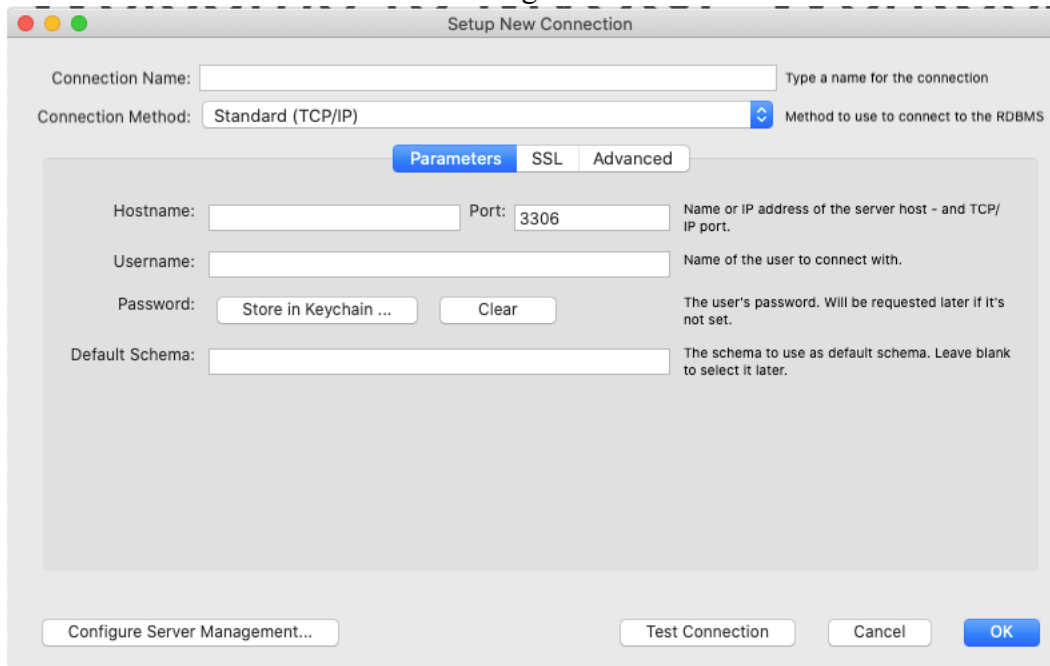


Fig. 2

## 5. Connectivity for MySQL

At **Additional connectivity configuration**


- Publicly accessible

Select **Yes**

(We can improve the security by setting VPC to connect your local MySQL with AWS RDS)

- VPC security group


Select **Create new**

**Connectivity** 

**Virtual private cloud (VPC)** [Info](#)  
VPC that defines the virtual networking environment for this DB cluster.

Default VPC (vpc-632ecb1e) ▼

Only VPCs with a corresponding DB subnet group are listed.

 After a database is created, you can't change the VPC selection.

▼ **Additional connectivity configuration**

**Subnet group** [Info](#)  
DB subnet group that defines which subnets and IP ranges the DB instance can use in the VPC you selected.

default-vpc-632ecb1e ▼

**Publicly accessible** [Info](#)

☐ Yes  
Amazon EC2 instances and devices outside the VPC can connect to your database. Choose one or more VPC security groups that specify which EC2 instances and devices inside the VPC can connect to the database.

☒ No  
RDS will not assign a public IP address to the database. Only Amazon EC2 instances and devices inside the VPC can connect to your database.

**VPC security group**  
Choose one or more RDS security groups to allow access to your database. Ensure that the security group rules allow incoming traffic from EC2 instances and devices outside your VPC. (Security groups are required for publicly accessible databases.)

☐ Choose existing  
Choose existing VPC security groups

☒ Create new  
Create new VPC security group

**New VPC security group name**

rds-sg

**Availability Zone** [Info](#)

No preference ▼

**Database port** [Info](#)  
TCP/IP port that the database will use for application connections.

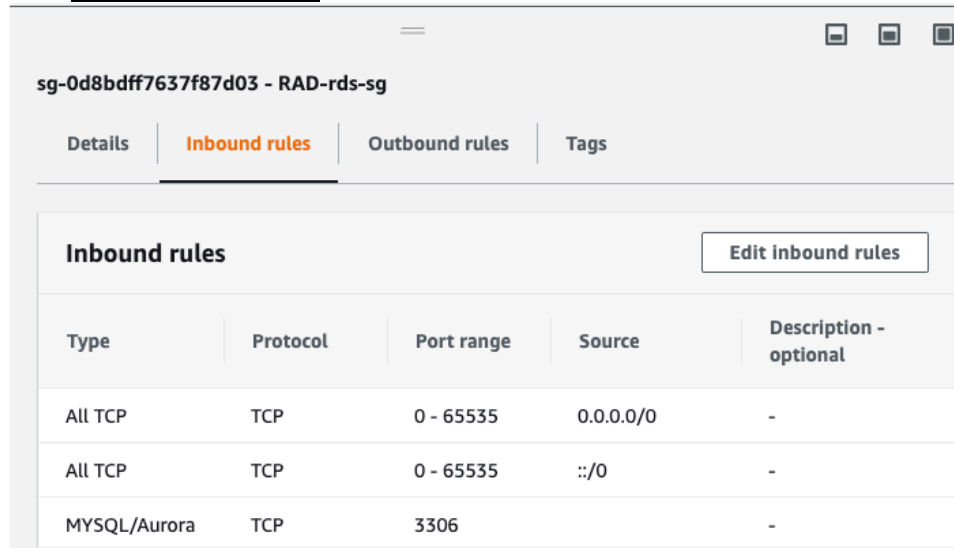
3306

Fig. 3

6. Click “Create database”
7. After the Database is created, checking the details of your MySQL.

### Connect to Your Local MySQL (from your computer)

- At AWS RDS (as show in Fig. 1) Select **Connectivity & security** and Click **VPC security groups**
- Click **Edit Inbound rules**, beside your default Inbound roles.



- Then Click **Add rule**, Select **My IP** For Source type.  
(Allow you to connect AWS RDS through local MySQL from your computer)

The screenshot shows the 'Add rule' dialog in the AWS Management Console. The 'Inbound rule 3' header is at the top. The 'Type' is set to 'MYSQL/Aurora', 'Port range' is '3306', 'Protocol' is 'TCP', and 'Source type' is 'My IP'. The 'Source' field is empty. An 'Add rule' button is at the bottom.

**Inbound rule 3** Delete

Type [Info](#)  
MYSQL/Aurora ▼

Port range [Info](#)  
3306

Source [Info](#)  
Q

Protocol [Info](#)  
TCP

**Source type** [Info](#)  
My IP ▼

Description - optional [Info](#)

**Add rule**

- Create the connections from your Local MySQL Workbench

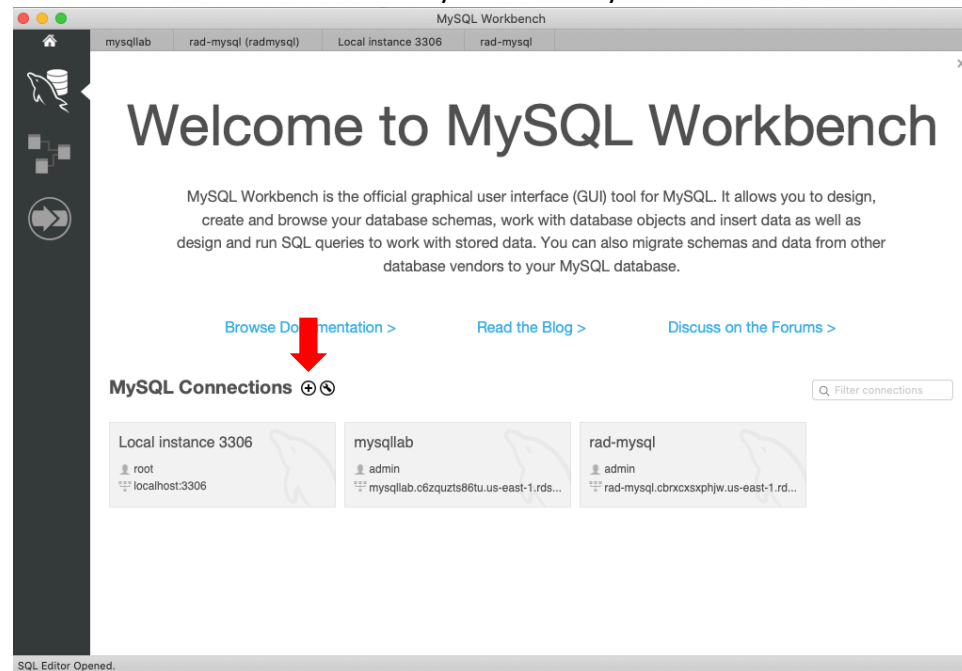


Fig. 4

- Sign in with your AWS RDS Service in Fig. 1 and Fig. 2.

## Select the database and Create the table

- Query For creating table and data

```
USE radmysql;

#DROP TABLE EventsTable;

CREATE TABLE EventsTable (
  EventID int PRIMARY KEY NOT NULL AUTO_INCREMENT,
  Organizer varchar(255) NULL,
  Venue varchar(255) NULL,
  EventDate varchar(255) NULL
);

SELECT * FROM EventsTable;

INSERT INTO EventsTable (Organizer, Venue, EventDate)
VALUE ('Plug In America', 'New York Auto Show', 'June 1, 2020');

SELECT * FROM EventsTable;
```

## Create Lambda and API Gateway (Nodejs) — AWS Serverless to RDS MySQL

### 1. Create NodeJS Code

- Create package.json

```
npm init
```

- The package.json for your reference.

```
{
  "name": "mysql-lambda",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Andrea Chang",
  "license": "ISC",
  "dependencies": {
    "mysql": "^2.18.1"
  }
}
```

- Install the “mysql” library for connecting the MySQL

```
npm install --save mysql
```

- Create Lambda function in index.js  
(example for the lambda update function, more examples for implementing REST api function at [Github link](#))

```
const mysql = require('mysql');

// connect Lambda with AWS RDS MySQL
const db = mysql.createConnection({
  host    : process.env.RDS_HOSTNAME,
  user    : process.env.RDS_USERNAME,
  password : process.env.RDS_PASSWORD,
  port    : process.env.RDS_PORT,
  database:'radmysql'
});

exports.handler = (event, context, callback) => {

  context.callbackWaitsForEmptyEventLoop = false;

  // get the dynamic parameters from the routing input
  // passing the default value to
  // prevent the error 'Internal server error' for
  // cannot destructure property 'undefined' or 'null'
  const id = event.queryStringParameters.EventID || 1;
  const Organizer = event.queryStringParameters.Organizer || "Plug In America";
  const Venue = event.queryStringParameters.Venue || "New York Auto Show";
  const EventDate = event.queryStringParameters.EventDate || "June 1, 2020";

  // create the MySQL query for updating data in MySQL from Node.js
  const sql = `UPDATE EventsTable SET Organizer= ?, Venue= ?, EventDate= ? WHERE EventID
= ?`;

  db.query(sql, [`${Organizer}`, `${Venue}`, `${EventDate}`, `${id}`], function (err, result) {
    if (err) throw err;

    // check the output is matching
    // the id we pass in, for
    // updating it with the new values (development purpose)
    console.log('Update EventID : ' + `${id}` + ', Organizer : ' + `${Organizer}` + ', Venue : ' + `${Venue}` + ', EventDate : ' + `${EventDate}`);

    // have to return the status code and the body with 'response' for
    // configuring it with GatewayAPI
    callback(null, {
      statusCode: 200,
      body: JSON.stringify({response: result})
    });
  });
};
```

- Zip all the node\_modules, index.js and package.json  
Upload the zip file  
Save the zip file  
The library and source code are uploaded.

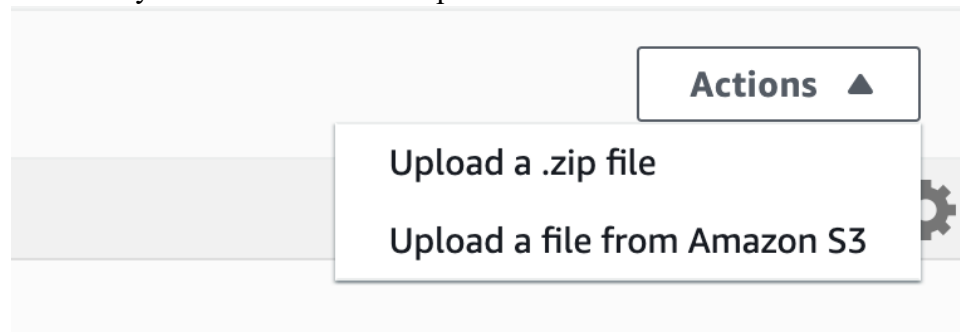


Fig. 5

- Setup the **Environment Variables** and Save

Environment variables (4)	
The environment variables below are encrypted at rest with the default Lambda service key.	
Key	Value
RDS_HOSTNAME	rad-mysql.██████████-east-1.rds.amazonaws.com
RDS_PASSWORD	password
RDS_PORT	3306
RDS_USERNAME	admin

Fig. 6



## 2. Create Lambda

- Click **Create function**
- We selected the “Author from scratch” and you can follow the setting.  
For Permission:
- Select **Create a new role from AWS policy templates**  
(Role name as you IAM, Identity and Access Management, roles name)
- Policy templates Select **Basic Lambda@Edge permissions (for CloudFront trigger)**

**Basic information**

Function name  
Enter a name that describes the purpose of your function.  
lambda\_gateway\_rest\_api\_mysql\_create  
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)  
Choose the language to use to write your function.  
Node.js 12.x

Permissions [Info](#)  
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

▼ Choose or create an execution role

Execution role  
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions

☐ Use an existing role

☒ Create a new role from AWS policy templates

**Role creation might take a few minutes. Please do not delete the role or edit the trust or permissions policies in this role.**

Role name  
Enter a name for your new role.  
RAD\_service  
Use only letters, numbers, hyphens, or underscores with no spaces.

Policy templates - *optional* [Info](#)  
Choose one or more policy templates.

Basic Lambda@Edge permissions (for CloudFront trigger) X  
CloudWatch Logs

Fig. 7

- Click “Create function”
- After the lambda is created, we need to Attach new policies for your IAM role("service-role/RAD\_service)
- Select **Edit** Basic settings

**Basic settings** [Edit](#)

Description  
-

Runtime  
Node.js 12.x

Handler [Info](#)  
index.handler

Memory (MB) [Info](#)  
128

Timeout [Info](#)  
0 min 3 sec

Fig.8

- After opening the Basic settings, at the bottom, Click **View the RAD\_service role**

**Basic settings**

Description - *optional*

Runtime

Node.js 12.x

Handler [Info](#)

index.handler

Memory (MB) [Info](#)

Your function is allocated CPU proportional to the memory configured.

128 MB

Timeout [Info](#)

0 min 3 sec

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☒ Use an existing role

☐ Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

service-role/RAD\_service

[View the RAD\\_service role](#) on the IAM console.

Fig. 9

- Select **Attach policies**, Search **AWSLambdaVPCAccessExecutionRole**

Summary

Role ARN	arn:aws:iam::309914797930:role/service-role/RAD_service
Role description	<a href="#">Edit</a>
Instance Profile ARNs	
Path	/service-role/
Creation time	2020-06-13 21:28 EDT
Last activity	2020-06-14 00:00 EDT (Today)
Maximum CLI/API session duration	1 hour <a href="#">Edit</a>

Permissions

Trust relationships

Tags

Access Advisor

Revoke sessions

▼ Permissions policies (1 policy applied)

[Attach policies](#)

Policy name ▼
<div> <div></div> <div>AWSLambdaVPCAccessExecutionRole</div> </div>

Fig. 10

- And then configure the security group For VPC
- Select **Custom VPC** , Select **sg-1a88b03f (default)** For Security group  
(We allow all the TCP access to our Lambda function, for demo purpose)

## VPC

ⓘ When you connect a function to a VPC in your account, it does not have access to the internet unless your VPC provides access. To give your function access to the internet, route outbound traffic to a NAT gateway in a public subnet. [Learn more](#)

### VPC connection [Info](#)

Connect to a virtual private cloud (VPC) to access network resources without exposing them to the internet.

☐ None

☒ Custom VPC

### VPC

Choose a VPC for your function to access.

vpc-a5b451d8 (172.31.0.0/16)

### Subnets

Select the VPC subnets for Lambda to use to set up your VPC configuration.

subnet-bb50849a (172.31.80.0/20) us-east-1a ✕

subnet-0140d94c (172.31.16.0/20) us-east-1b ✕

subnet-3c00ca63 (172.31.32.0/20) us-east-1c ✕

### Security groups

Choose the VPC security groups for Lambda to use to set up your VPC configuration. The table below shows the inbound and outbound rules for the security groups that you choose.

sg-1a88b03f (default) ✕  
default VPC security group

### Inbound rules

### Outbound rules

< 1 >			
Security group ID	Protocol	Ports	Source
sg-1a88b03f	All	All	sg-1a88b03f

Fig. 11

### 3. Test Lambda

- Testing Lambda function For each REST API function
- Select **Configure events**

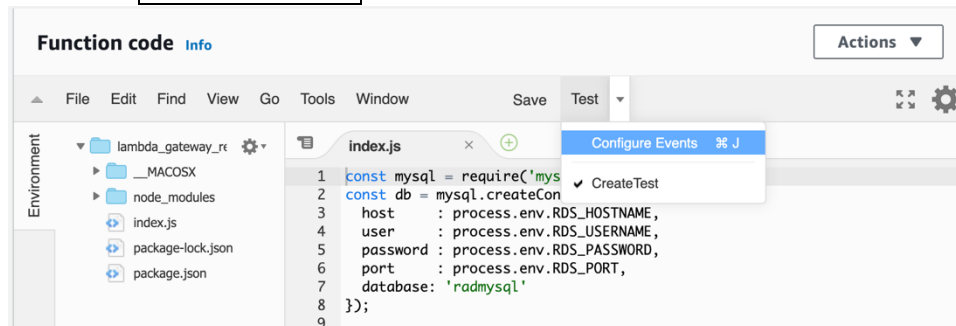


Fig. 12

- Click **Create new test event**
- Select **Amazon API Gateway AWS Proxy**

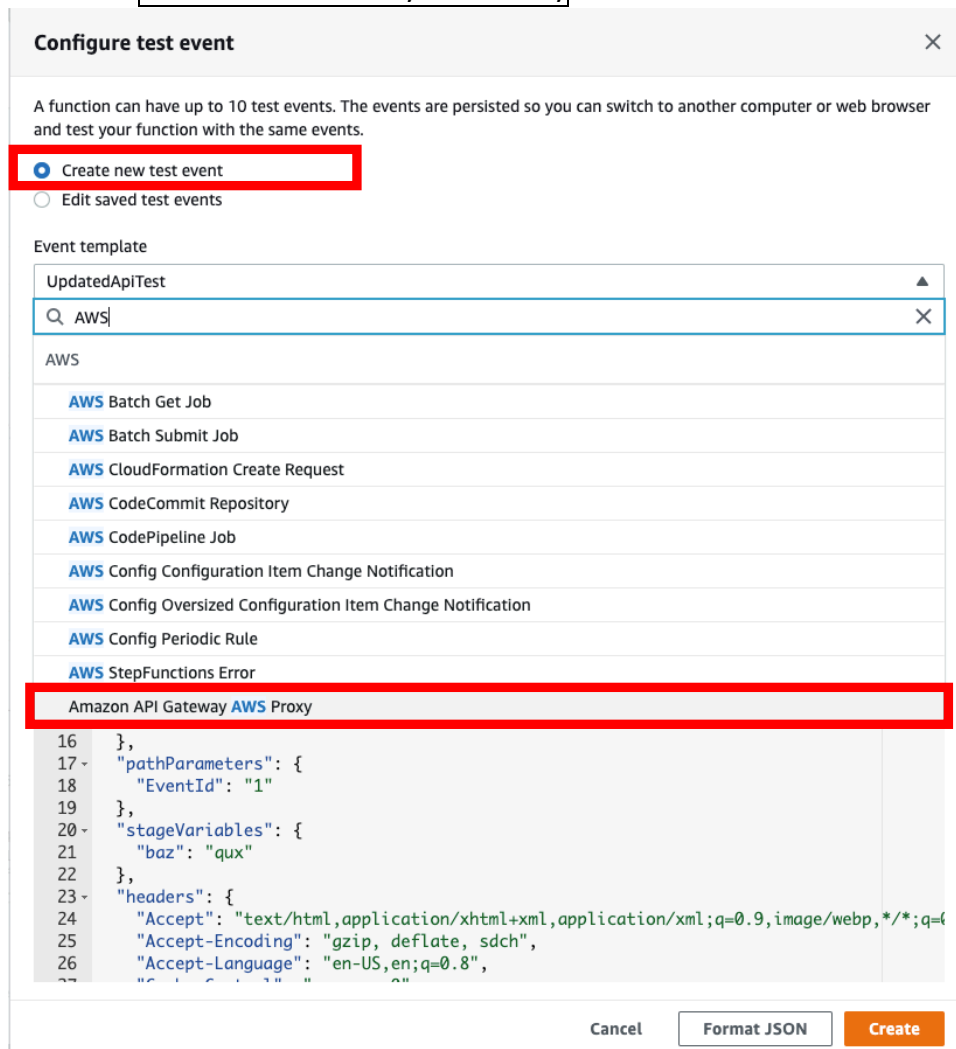


Fig. 13

- Set the routing parameters for testing Create API

```

1- {
2   "body": "eyJ0ZXN0IjoIYm9keSJ9",
3   "resource": "/{proxy+}",
4   "path": "/path/to/resource",
5   "httpMethod": "POST",
6   "isBase64Encoded": true
7   "queryStringParameters": {
8     "id": "1",
9     "Organizer": "Plug In",
10    "EventDate": "June 3, 2020"
11  },
12  "multiValueQueryStringParameters": {
13    "foo": [
14      "bar"
15    ]
16  },
17  "pathParameters": {
18    "EventId": "1"
19  },
20  "stageVariables": {
21    "baz": "qux"
22  }

```

- Listing all the testing parameters
  - CreateApiTest:

```

"queryStringParameters": {
  "Organizer": "Plug In America",
  "Venue": "New York Auto Show",
  "EventDate": "June 1, 2020"
},
"pathParameters": {
  "Organizer": "Plug In America",
  "Venue": "New York Auto Show",
  "EventDate": "June 1, 2020"
}

```

- GetApiTest
- DeleteApiTest

```

"queryStringParameters": {
  "id": "1"
},
"pathParameters": {
  "EventId": "1"
}

```

#### 4. Create Gateway API

Create update API routing

- Click **Actions**, Select **Create Resource**
- Type **update** For Resource Name and Resource Path
- Click **Enable API Gateway CORS**
- Click **Create Resource**

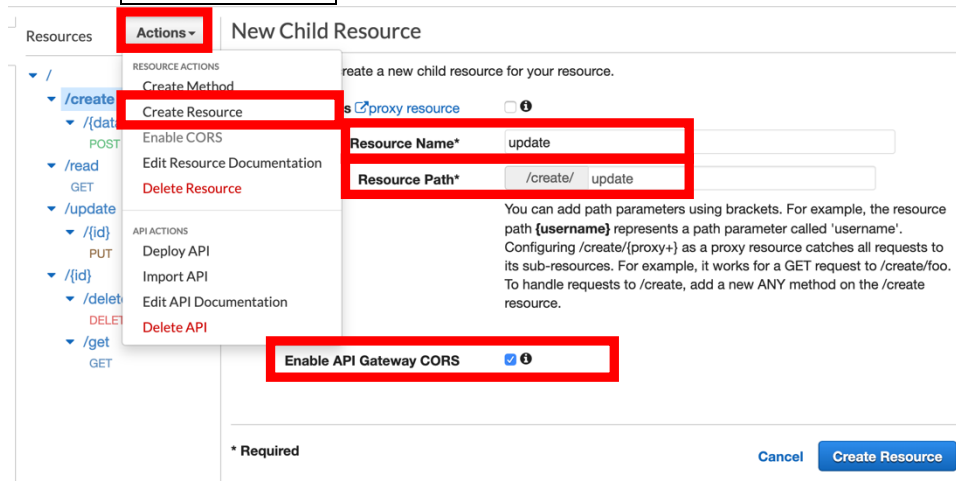
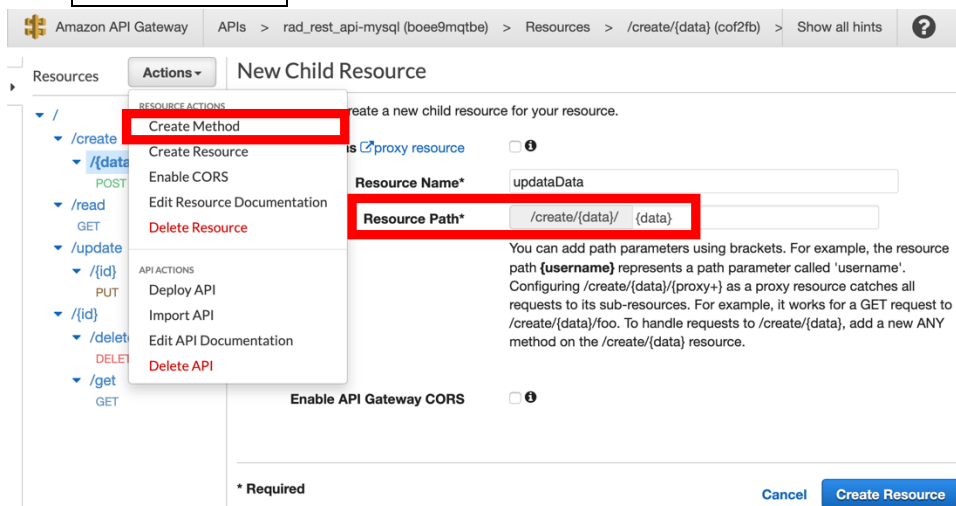


Fig. 14

Create routing parameters for input data

- Click **Actions**, Select **Create Resource** again
- Type **updateData** For Resource Name and **{data}** For Resource Path
- Click **Enable API Gateway CORS**
- Click **Create Resource**



Fir. 15

### Create POST (REST API end point)

- Click **Actions**, Select **Create Method**
- Select **PUT** Method
- Select Integration type **Lambda Function**
- Click **Use Lambda Proxy integration**
- Select update Lambda function you initiate it in the AWS Lambda

Resources **Actions** Choose the integration point for your new method.

**Integration type** ☒ Lambda Function ⓘ

☐ HTTP ⓘ  
☐ Mock ⓘ  
☐ AWS Service ⓘ  
☐ VPC Link ⓘ

**Use Lambda Proxy integration** ☒ ⓘ

**Lambda Region** us-east-1

**Lambda Function**  
lambda\_gateway\_rest\_api\_with\_mysql\_update ⓘ

**Use Default Timeout** ☒ ⓘ

**Save**

Fig. 16

- Following the fig. to create the rest of the routing path

Resources **Actions** **New Child Resource**

Use this page to create a new child resource for your resource.

**Configure as proxy resource** ☒ ⓘ

**Resource Name\*** update

**Resource Path\*** /create/ update

You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'.  
Configuring `/create/{proxy+}` as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to `/create/foo`.  
To handle requests to `/create`, add a new ANY method on the `/create` resource.

**Enable API Gateway CORS** ☒ ⓘ

\* Required

**Cancel** **Create Resource**

Fig. 17

## 5. Test Gateway API

- At Method Execution, Click **test** For API Gateway

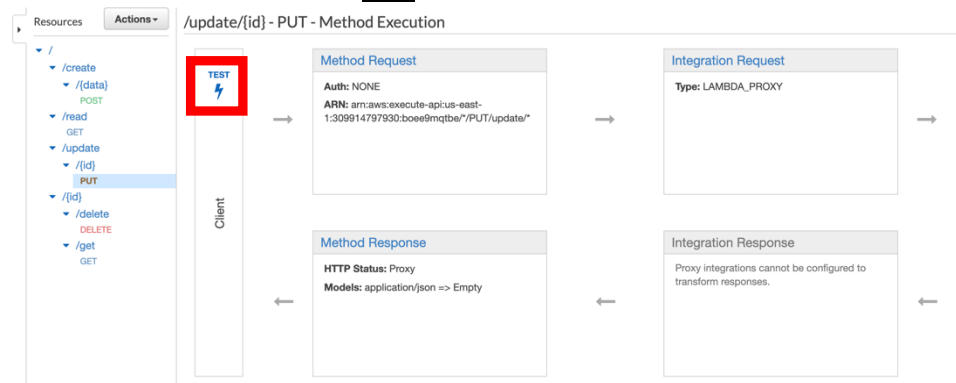


Fig. 18

- Passing the Query Strings and Request Body for testing

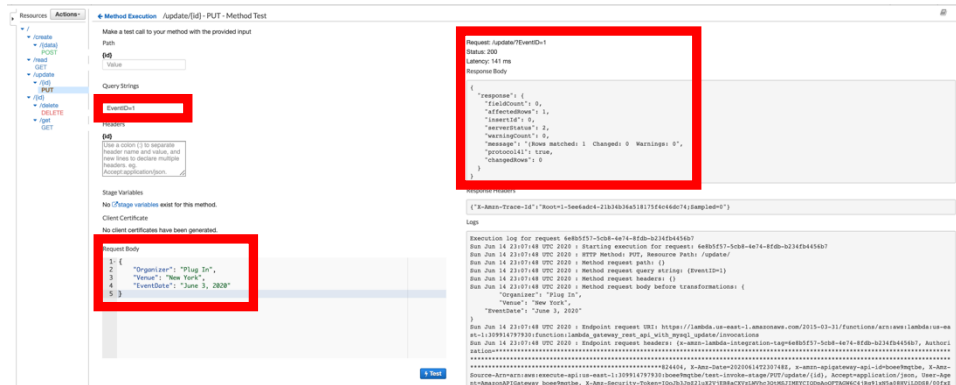


Fig. 19

- List all the Gateway API testing parameters

-GetApiTest:

-DeleteApiTest:

Query Strings

Input: **EventID= 1**

-CreateApiTest:

Query Strings

Input:

**Organizer=Plug In America&Venue=New York Auto Show&EventDate=June 1, 2020**

-UpdatedApiTest:

Query Strings:

Input: **EventID= 1**

Request Body:

```

{
  "Organizer": "Plug In",
  "Venue": "New York",
  "EventDate": "June 3, 2020"
}
  
```



- After testing, Click **Action**, Select **Deploy API**

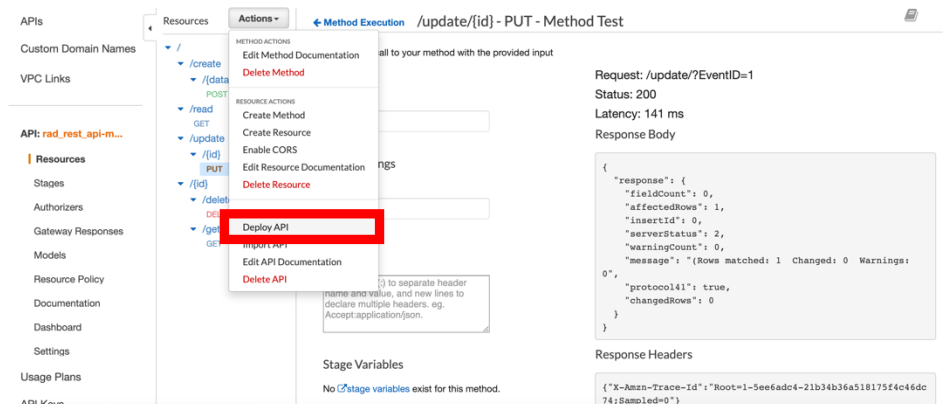


Fig. 20

- Select **Stages** at the left bar, Click the Invoke URL as endpoint

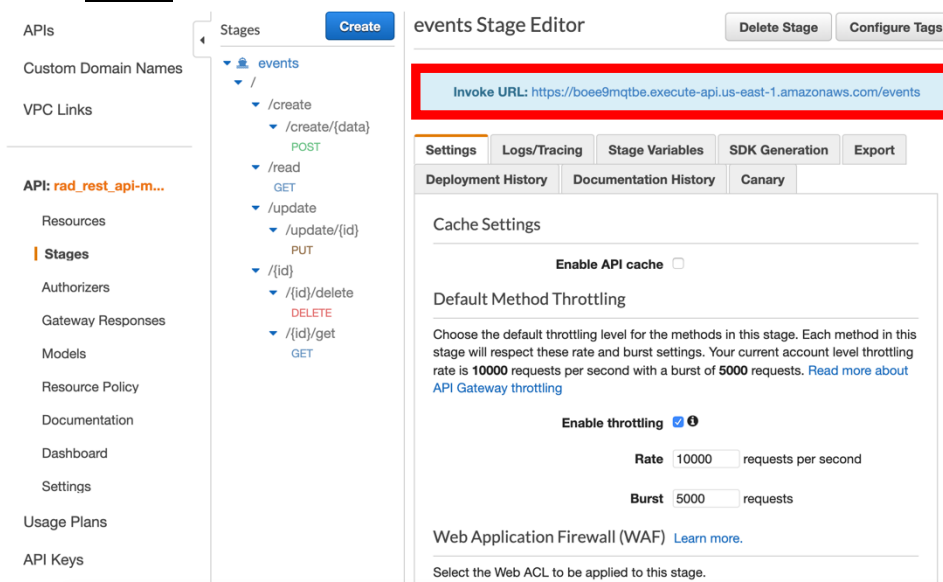


Fig.21