

# How to Build a RESTful API with a Serverless Framework on AWS Lambda

I'll take you through the entire process of implementing RESTful API service on Serverless Framework using AWS Lambda, a serverless compute service and you better believe it's going to be awesome.

If you are going to use Serverless functionality offered by [Azure Functions](#), and [Google Cloud Functions](#), be sure to check the documentation.

If you don't know about the serverless computing then I strongly recommended watching the following video before getting started.

---

## Build RESTful API with a Serverless Framework

With a "Serverless Framework", we can quickly build, configure and deploy resources within few commands. We can store our code and configuration into a centralized repository so we can design proper workflow and developers can later write, reuse and refer other developers codebase.

Let's walk the walk together and to build a Pokemon RESTful API services with a "Serverless Framework". Please check out the table below for reference.

#	ENDPOINT	METHOD	DESCRIPTION
1	pokemon/	GET	Get a list of all pokemon from the database
2	pokemon/{id}	GET	Get a specific pokemon.
3	pokemon/	POST	Add new pokemon to the database.
4	pokemon/{id}	PUT	Update existing pokemon.
5	pokemon/{id}	DELETE	Delete existing pokemon.

You can find the code for this article here: <https://github.com/sagar-gavhane/pokemon-app>

## Prerequisites

We need to install the following tools and frameworks:

1. Node.js 8.10 or above
2. MySQL
3. Visual Studio Code (preferred) or any code editor
4. Postman

Note: This guide is simply to give you an idea of how to build an API and should not be mistaken as a guide for creating Productive API and we strongly suggest that you write your own functions for efficient serverless autoscaling.

---

## Serverless Setup

We have to install Serverless globally, so fire up a terminal window and run:

```
npm install -g serverless
```

Note: You may need to run the command as sudo.

Now, let's install plugins and libraries step by step.

---

## Install Dependencies

Install the following packages to work with "Serverless Framework"

- [express](#) - Fast, unopinionated, minimalist web framework for Node.js.
- [body-parser](#) - Parse incoming request bodies in a middleware before your handlers, available under the req.body property.
- [mysql](#) - A pure node.js JavaScript Client implementing the MySQL protocol.
- [serverless-http](#) - Plugin allows you to wrap express API for serverless use.

- serverless-offline - Plugin to emulate AWS Lambda and API Gateway for speed up local development.
- 

## App structure

Before we start writing the handler code, we're going to structure the project folder and configure our tools.

Create the following structure at the root level:

Make sure to list private files into a file `.gitignore` so that we don't accidentally commit it to the public repository. Copy paste raw material from <https://www.gitignore.io/api/node> to file `.gitignore`.

`serverless.yml` file serves as a manifest for our RESTful API service. Where we define our functions, events, and necessary resources. Later, with serverless CLI we configure and deploy our service to AWS infrastructure.

We are doing a few things here:

1. `service`: `pokemon-service` is the name of the service. You can give any type name for your service.
2. `provider`: This is where we specify the name of the provider we are using (AWS as cloud service provider) and configurations specific to it. In our case, we've configured the runtime (Node.js) with 8.10 version and region `toous-east-1`.
3. `functions`: We specify the functions provided by our service, Here I'm specifying `aspokemonFunc` function name with `eventshttp`. We can also say that this is our AWS Lambda function.

We have to store our pokemon somewhere, for sake of simplicity I'm chosen MySQL but you can also use another type database. I have already created a database with name `pokemon_db` and inside a database created table `pokemon_tb` with `id`, `name`, `height`, `weight`,

avatar, and createAt columns.

Rather than creating and managing connections every time, we configure pool connections once inside a file dbConfig.js and reused it multiple times.

---

## Writing the handler function

Let's focus on handling RESTful API route inside the index.js file with express. First, we imported the package serverless-http at the top. Second, we exported a handler function which is our application wrapped in the serverless package.

Here, we're implementing basic five routes for handling operation crud with pokemon (without any validation).

## Terminal snapshot:

```
> pokemon-app@1.0.0 start /home/sagar/workspace/pokemon-app
> serverless offline start

Serverless: Starting Offline: dev/us-east-1.

Serverless: Routes for pokemonFunc:
Serverless: GET /pokemon
Serverless: GET /pokemon/{id}
Serverless: POST /pokemon
Serverless: PUT /pokemon/{id}
Serverless: DELETE /pokemon/{id}

Serverless: Offline listening on http://localhost:3000
```

Get all pokemon:

http://localhost:3000/ | No Environment

GET http://localhost:3000/pokemon | Params | Send | Save

Pretty Raw Preview JSON

```

1 {
2   "data": [
3     {
4       "id": 1,
5       "name": "pikachu",
6       "height": 4,
7       "weight": 60,
8       "avatar": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/120.png",
9       "createdAt": "2018-10-18T08:26:57.000Z"
10    },
11   {
12     "id": 2,
13     "name": "Charizard",
14     "height": 5,
15     "weight": 89,
16     "avatar": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/100.png",
17     "createdAt": "2018-10-18T10:05:29.000Z"
18   },
19   {
20     "id": 3,
21     "name": "raychu",
22     "height": 3,
23     "weight": 24,
24     "avatar": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/144.png",
25     "createdAt": "2018-10-18T14:03:13.000Z"
26   }
27 ],
28 "message": "All pokemons successfully retrieved."
29 }

```

## Get pokemon by id:

http://localhost:3000/ | No Environment

GET http://localhost:3000/pokemon/2 | Params | Send | Save

Pretty Raw Preview JSON

```

1 {
2   "data": {
3     "id": 2,
4     "name": "Charizard",
5     "height": 5,
6     "weight": 89,
7     "avatar": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/100.png",
8     "createdAt": "2018-10-18T10:05:29.000Z"
9   },
10  "message": "Pokemon Charizard successfully retrieved."
11 }

```

## Add new pokemon:

http://localhost:3000/ | No Environment

POST http://localhost:3000/pokemon | Params | Send | Save | Status: 201 Created | Time: 169 ms | Size: 497 B

Auth Headers (1) Body Pre-req. Tests Cookies Code | Body Cookies Headers (9) Test Results

form-data x-www-form-urlencoded raw binary

JSON (application/json)

```

1 {
2   "name": "Bulbasaur",
3   "height": 3,
4   "weight": 67,
5   "avatar": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/78.png"
6 }

```

Pretty Raw Preview JSON

```

1 {
2   "data": {
3     "id": 7,
4     "name": "Bulbasaur",
5     "height": 3,
6     "weight": 67,
7     "avatar": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/78.png"
8   },
9   "message": "Pokemon Bulbasaur successfully added."
10 }

```

## Update existing pokemon:

The screenshot shows a REST client interface for a PUT request to `http://localhost:3000/pokemon/1`. The request body is a JSON object: 

```
{ 1: { 2: "name": "Pikachu - Name changed", 3: "height": 3, 4: "weight": 55 5: }
```

. The response status is 200 OK, and the response body is a JSON object: 

```
{ 1: { 2: "data": { 3: "id": 1, 4: "name": "Pikachu - Name changed", 5: "height": 3, 6: "weight": 55, 7: "avatar": "https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/128.png", 8: }, 9: "message": "Pokemon Pikachu - Name changed is successfully updated." 10: }
```

## Delete existing pokemon:

The screenshot shows a REST client interface for a DELETE request to `http://localhost:3000/pokemon/2`. The response status is 200 OK, and the response body is a JSON object: 

```
{ 1: { 2: "data": null, 3: "message": "Pokemon with id: 2 successfully deleted." 4: }
```

## MySQL

For simplicity I choose MySQL but you can use any NoSQL database.

Here's a [simple tutorial](#) provided by AWS that will help you to configure and Connect to Serverless MySQL Database

## Deployment

Deploying services with the serverless framework is so simple, we require to just hit deploy command.

serverless deploy

Creating RESTful API with a serverless framework is pretty straightforward. For serverless, we have to switch our development workflow. I found that lots of companies are moving towards creating and managing micro-services architecture instead of the monolithic app.

---

That's it!

You've successfully created a RESTful API with a Serverless Framework.

You may also be interested in learning How to build a [Serverless Website with AWS Lambda](#) in 7 Easy Steps.

This post was originally published on [dev.to](#) by [@sagar](#) on 19 Oct 2018 and reposted with permitted edits.