

Construindo uma API com NodeJS e Serverless

Carlos Santos



O que é Serverless e como funciona

- “Computação sem servidores” (não é bem assim)
- Foco apenas no core do negócio (código)
- Existem servidores mas o desenvolvedor ou o operador não precisa gerenciá-lo
- Quando utilizar serverless? (eventos)

Vantagens de utilizar serverless

- Acionado por eventos
- Auto escalável
- Foco apenas no core do seu negócio (não precisa gerenciar servidores)
- Redução de custos (paga por evento)

Framework Serverless, por que utilizar?

- Facilidade no deploy da sua aplicação.
- Suporte em diferentes providers: Google, AWS, Azure..
- Plugins
- Suporte a diversas linguagens: nodejs, python, go, java..

Instalando o serverless e o template do google

Installing the serverless cli

```
npm install -g serverless
```

#Instalando o Google template project

```
serverless create --template google-nodejs --path my-service
```

#Entrar na pasta e instalar

```
cd my-service
```

```
npm install
```

▲ MY-SERVICE

▸ node_modules

📁 .gitignore

JS index.js

{ } package.json

! serverless.yml

```
service: my-service # NOTE: Don't put the word "google" in here
```

```
provider:
```

```
  name: google
```

```
  stage: dev
```

```
  runtime: nodejs8
```

```
  region: us-central1
```

```
  project: my-project
```

```
  # The GCF credentials can be a little tricky to set up. Luckily we've documented this for you here:
```

```
  # https://serverless.com/framework/docs/providers/google/guide/credentials/
```

```
  #
```

```
  # the path to the credentials file needs to be absolute
```

```
  credentials: ~/.gcloud/keyfile.json
```

```
plugins:
```

```
  - serverless-google-cloudfunctions
```

```
# needs more granular excluding in production as only the serverless provider npm
```

```
# package should be excluded (and not the whole node_modules directory)
```

```
package:
```

```
  exclude:
```

```
    - node_modules/**
```

```
    - .gitignore
```

```
    - .git/**
```

```
functions:
```

```
  first:
```

```
    handler: http
```

```
    events:
```

```
      - http: path
```

Variáveis de ambiente

```
provider:  
  name: google  
  runtime: nodejs  
  project: projeto-cf  
  # the path to the credentials file needs to be absolute  
  credentials: /home/carlos/projeto/projeto.json  
  environment:  
    DB_HOST: ${file(config/env.js):env.DB_HOST}  
    DB_USER: ${file(config/env.js):env.DB_USER}  
    DB_PASS: ${file(config/env.js):env.DB_PASS}  
    DB_DATABASE: ${file(config/env.js):env.DB_DATABASE}
```


Deploy

Rodando comando de deploy
`serverless deploy`

```
PS C:\Users\Carlos\Documents\app\serverless\my-service> serverless deploy
Serverless: Packaging service...
Serverless: Excluding development dependencies...
Serverless: Compiling function "first"...
Serverless: Creating deployment...
Serverless: Checking deployment create progress...
..
Serverless: Done...
Serverless: Uploading artifacts...
Serverless: Artifacts successfully uploaded...
Serverless: Updating deployment...
Serverless: Checking deployment update progress...
.....
Serverless: Done...
Service Information
service: my-service
project: 
stage: dev
region: us-central1

Deployed functions
first
https://us-central1-projeto-cf.cloudfunctions.net/http
```

Tratando a request no Google Cloud Functions

```
exports.cliente = (req, res) => {  
  switch (req.method) {  
    case 'GET':  
      buscarCliente(req, res);  
      break;  
    case 'PUT':  
      editarCliente(req, res);  
      break;  
    case 'POST':  
      criarCliente(req, res);  
      break;  
    case 'DELETE':  
      removerCliente(req, res);  
      break;  
    default:  
      res.status(405).send({error: 'Something blew up!'});  
      break;  
  }  
};
```

GET cliente

```
function buscarCliente(req, res) {  
  let clientes = ['Alita', 'Kailani', 'Lindinalva', 'Ivaneide', 'Lucinete', 'Amber', 'Shakira',  
  'Brisa', 'Indianara', 'Deisiane'];  
  res.status(200).send(JSON.stringify(clientes));  
}
```

GET response

The screenshot shows the developer tools interface for a GET request. The 'Params' tab is selected, displaying a table with the following structure:

KEY	VALUE	DESCRIPTION	••
Key	Value	Description	

Below the table, the 'Body' tab is selected, showing the response data in JSON format:

```
i 1 [{"Alita", "Kailani", "Lindinalva", "Ivaneide", "Lucinete", "Amber", "Shakira", "Brisa", "Indianara", "Deisiane"}]
```

At the top right of the response area, the status is 200 OK, the time taken is 503 ms, and the size is 459 B. The interface also includes tabs for Params, Authorization, Headers, Body, Pre-request Script, Tests, Cookies, and Code. The Body tab has sub-tabs for Pretty, Raw, Preview, and HTML, along with a menu icon.

POST cliente

```
function criarCliente(req, res) {  
  if(!req.body.name){  
    res.status(405).send('O parâmetro nome é obrigatório!');  
  }  
  
  //Salva cliente  
  let clienteId = Math.floor(Math.random() * 100);  
  let response = {"id": clienteId, "nome": req.body.name,  
    "mensagem": "Cliente criado com sucesso"}  
  res.status(200).send(JSON.stringify(response));  
}
```

POST response

The screenshot displays a REST client interface with two main sections. The top section shows the request configuration, and the bottom section shows the response details and body.

Request Section:

- Params
- Authorization
- Headers (1)
- Body** (selected)
- Pre-request Script
- Tests
- Cookies
- Code

Body format options: none, form-data, x-www-form-urlencoded, **raw** (selected), binary, JSON (application/json)

Request Body (Line 1): `{"name": "Carlos Santos"}`

Response Section:

- Body (selected)
- Cookies
- Headers (9)
- Test Results

Status: 200 OK | Time: 199 ms | Size: 423 B

Response Body (Line 1): `{ "id": 14, "nome": "Carlos Santos", "mensagem": "Cliente criado com sucesso" }`

PUT cliente

```
function editarCliente(req, res) {
  let id = getIdFromUrl(req.url);

  if(!req.body.name){
    res.status(405).send('O parâmetro nome é obrigatório!');
  }

  if(!id){
    res.status(405).send('O parâmetro id é obrigatório para editar o registro!');
  }

  //Edita o seu cliente no banco
  res.status(200).send("Cliente editado com sucesso id: "+id);
}
```

PUT response

The screenshot displays a REST client interface with the following details:

- Method:** PUT
- URL:** `https://us-central1-projeto-cf.cloudfunctions.net/helloHttp/33`
- Send Button:** A blue button labeled "Send".
- Request Body:** The "Body" tab is selected, showing a JSON payload: `{"name": "Carlos Santos"}`. The content type is set to "JSON (application/json)".
- Response:** The "Body" tab shows the response: `Cliente editado com sucesso id: 33`. The status is "200 OK", the time is "311 ms", and the size is "389 B".

DELETE cliente

```
function removerCliente(req, res) {  
  let id = getIdFromUrl(req.url);  
  
  if(!id){  
    res.status(405).send('O parâmetro id é obrigatório para deletar o registro!');  
  }  
  
  //deleta o registro do seu db  
  res.status(200).send("Cliente removido com sucesso id: "+id);  
}
```

DELETE response

The screenshot displays a REST client interface with the following components:

- Request Section:**
 - Tab: **Body** (selected)
 - Content Type: **JSON (application/json)**
 - Request Body:

```
1 [{"name": "Carlos Santos"}]
```
- Response Section:**
 - Tab: **Body** (selected)
 - Status: **200 OK**
 - Time: **456 ms**
 - Size: **390 B**
 - Content Type: **HTML**
 - Response Body:

```
i 1 Cliente removido com sucesso id: 33
```

Perguntas?

carlos@totalvoice.com.br



Obrigado!

