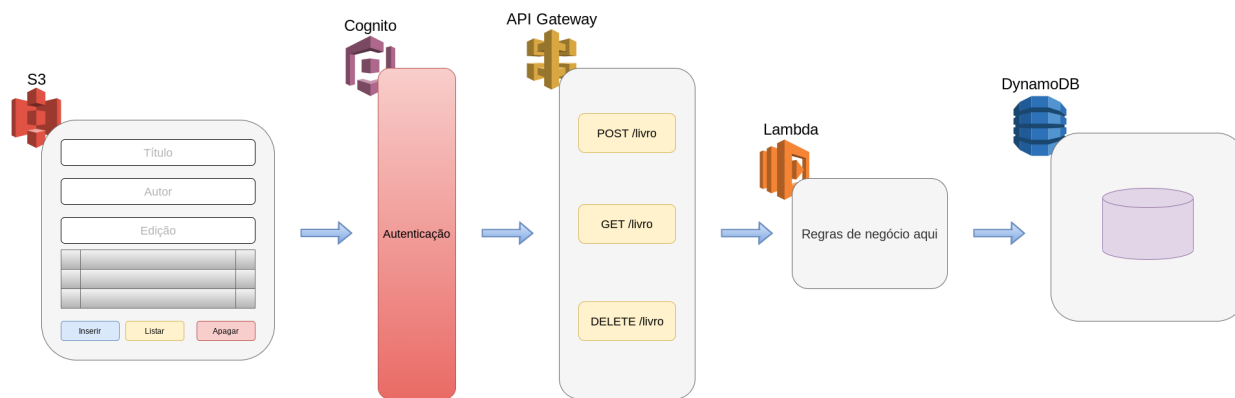
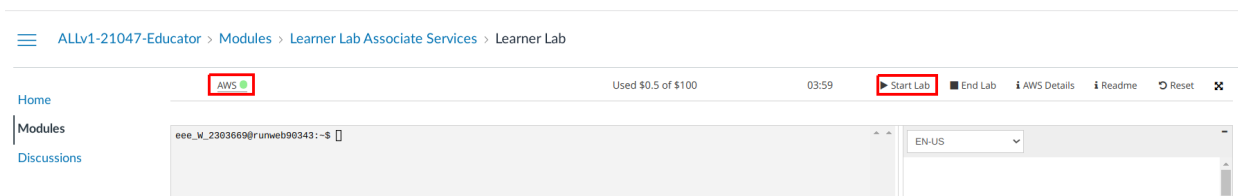


## 1 Introdução

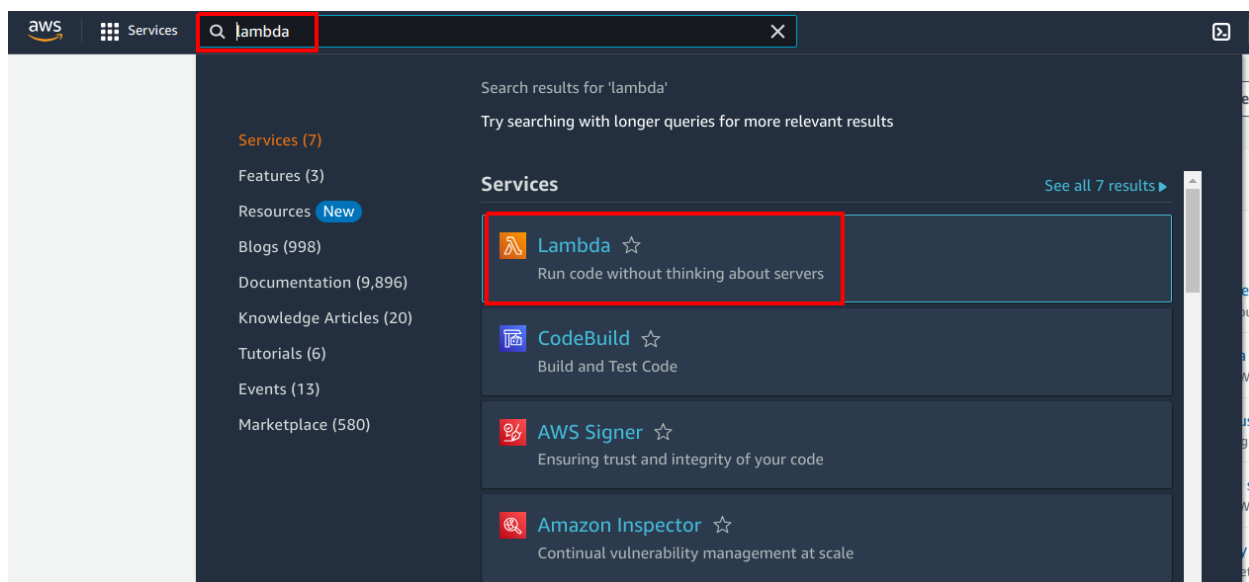
Prosseguimos com a implementação da aplicação retratada a seguir.



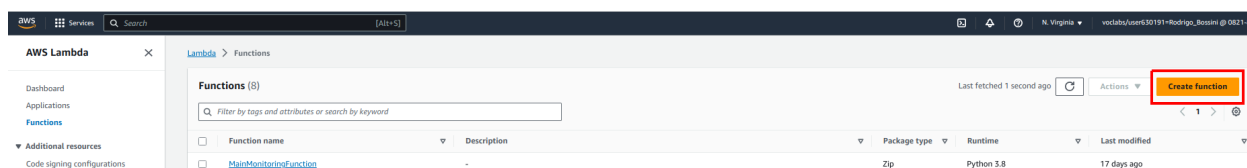
**(Criando uma função Lambda)** O próximo passo é criar uma função Lambda que nos servirá de Back End. Sua existência nos permitirá implementar nossas regras de negócio. Visite seu Learner Lab, inicie o Lab e clique em AWS quando a bolinha ficar verde.



No console AWS, busque pelo serviço Lambda e clique em seu ícone.



Clique em **Create Function**.



Preencha os campos como a seguir e clique em **Create Function**.

**Nota.** **LabRole** é uma Role previamente criada no ambiente Learner Lab. As permissões associadas a ela são suficientes para aquilo que pretendemos implementar.

**Basic information**

**Function name**  
Enter a name that describes the purpose of your function.  
lambda-pdfs

**Runtime**  
Choose the runtime environment to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.  
Node.js 18.x

**Architecture**  
Choose the architecture you want for your function code.  
x86\_64

**Permissions**  
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

**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

**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.  
LabRole

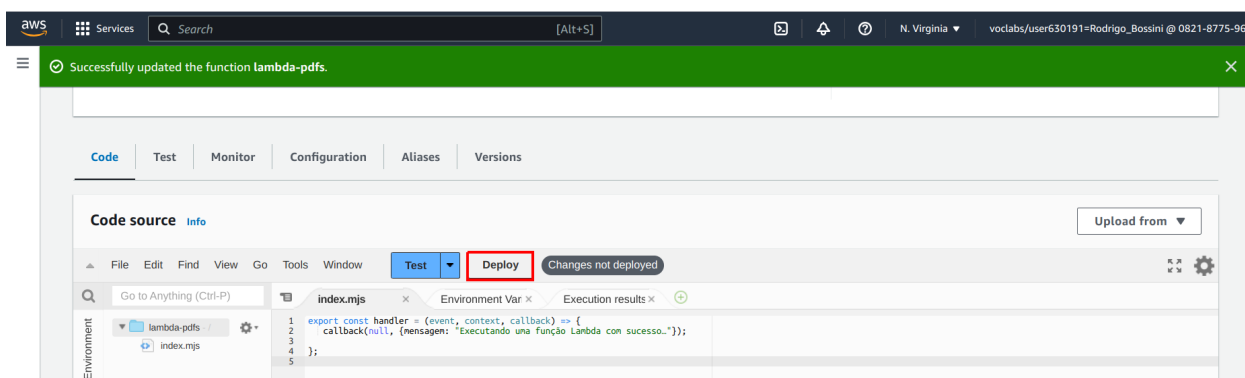
**Advanced settings**

Cancel **Create function**

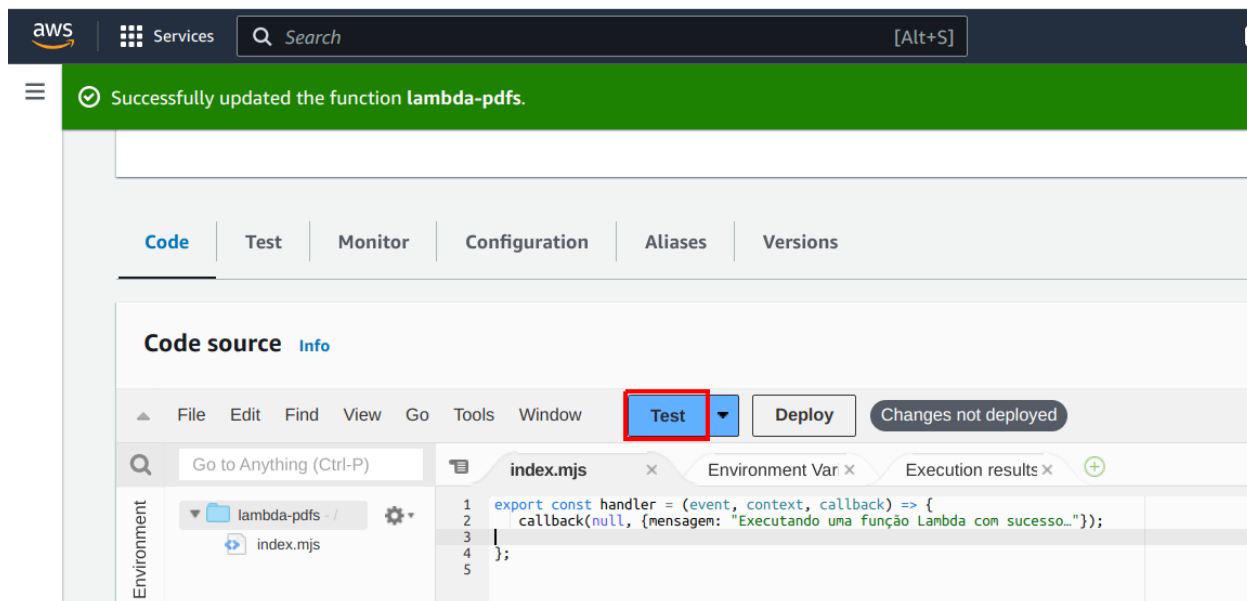
Na tela resultante, ajuste o código inicial como a seguir. **callback** é o nome de uma função que é colocada em execução automaticamente quando a função lambda é chamada. Nossa implementação será feita nesta função. Aprenderemos mais sobre os outros objetos em breve.

```
export const handler = (event, context, callback) => {
  callback(null, {mensagem: "Executando uma função Lambda com sucesso..."});
};
```

Clique em **Deploy**.



Podemos testar a função Lambda criando um evento de teste “Mock”. Para tal, clique em **Test**.



Escolha a opção **Create new Event**. Dê um nome para o seu teste. Mantenha as demais opções com seu valor padrão e clique em **Save**.

### Configure test event

A test event is a JSON object that mocks the structure of requests emitted by AWS services to invoke a Lambda function. Use it to see the function's invocation result.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

☒ Create new event
 ☐ Edit saved event

Event name

test-lambda-pdfjs

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

☒ Private
 

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

☐ Shareable
 

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

hello-world

Event JSON

```

1 {
2   "key1": "value1",
3   "key2": "value2",
4   "key3": "value3",
5 }

```

Format JSON

Cancel

Invoke

Save

Clique em **Test** para obter o resultado a seguir.

Code source info

Test

Deploy

index.mjs

Environment Vari

Execution result

lambda-pdfjs

index.mjs

Test Event Name

(unsaved) test event

Response

{ "mensagem": "Executando uma função Lambda com sucesso.." }

Function Logs

START RequestId: b8375585-5697-4ecd-96b3-be2b88ef03f3 Version: SLATEST  
 END RequestId: b8375585-5697-4ecd-96b3-be2b88ef03f3 Duration: 162.26 ms Billed Duration: 163 ms Memory Size: 128 MB Max Memory Used: 67 MB  
 REPORT RequestId: b8375585-5697-4ecd-96b3-be2b88ef03f3

Request ID

b8375585-5697-4ecd-96b3-be2b88ef03f3

Status: Succeeded

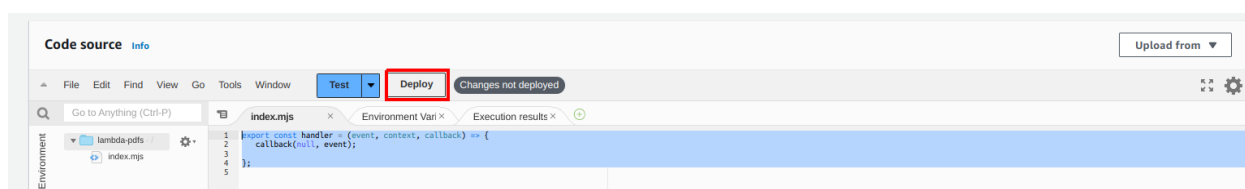
Max memory used: 67 MB

Time: 162.26 ms

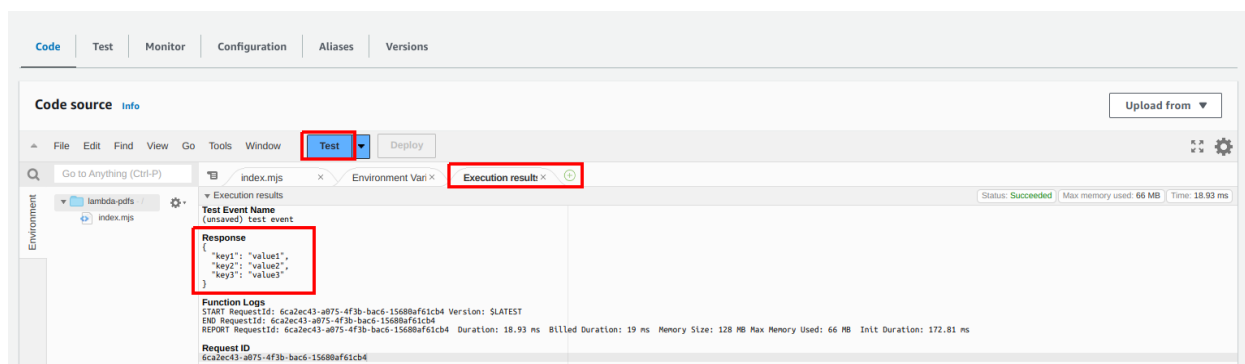
Para visualizar o objeto de teste especificado (o json com três pares/chave valor key: value), volte ao arquivo **index.mjs** e ajuste o código como a seguir. Observe que a nossa função agora devolve o objeto **event**.

```
export const handler = (event, context, callback) => {
  callback(null, event);
};
```

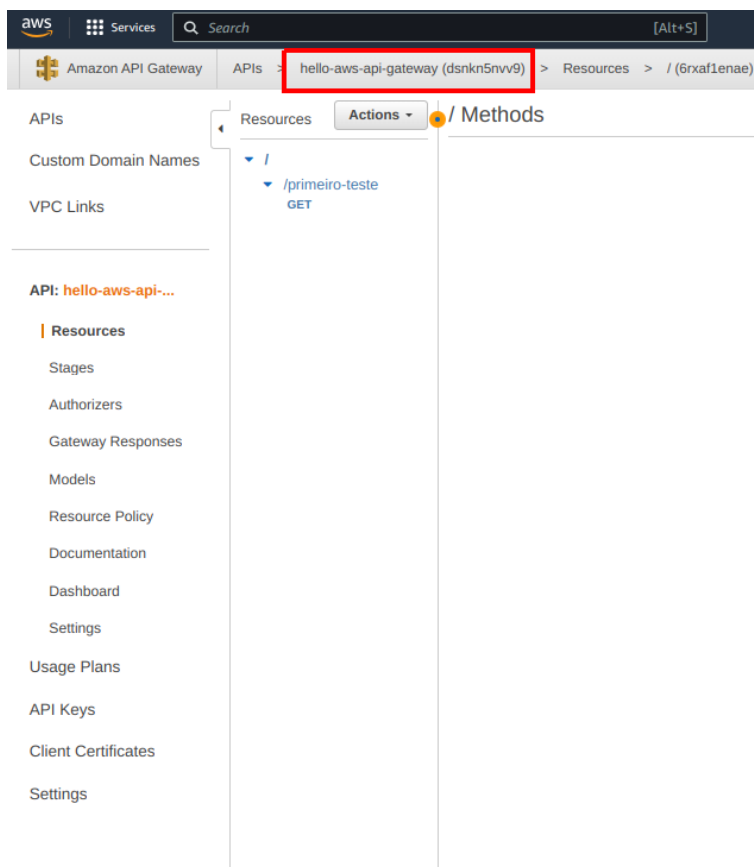
Clique em **Deploy** a seguir.



Depois, clique em **Test**. Veja o resultado esperado.



**(Mais funcionalidades do API Gateway e nossa nova API)** Como vimos, o serviço AWS API Gateway oferece diferentes funcionalidades. Nesta seção estudaremos as principais. Depois disso, criaremos a API da solução computacional que estamos implementando. Certifique-se de que você está na página inicial da sua API. Para tal, você pode clicar no nome dela.



**(API Keys)** Essa feature nos permite criar chaves de acesso para nossa API. Tratam-se de chaves que poderíamos entregar para nossos clientes e restringir o acesso aos serviços somente àqueles que enviem uma chave válida. Além disso, elas podem ser usadas para garantir que um usuário não ultrapasse os limites de uso impostos pelo plano que tiver escolhido. À esquerda, clique **API Keys** e então clique **Actions >> Create API Key**. Dê um nome qualquer para a chave, talvez **teste**, e escolha **Auto Generate**. Na tela a seguir, clique **Show** para ver a chave gerada.

- **(Custom Domain Name)** Como o nome sugere, com esta opção podemos escolher um domínio personalizado para nossa api, como **api.meudominio.com**.

- **(Client Certificates)** Aqui podemos criar um certificado que pode ser enviado para outros serviços a fim de garantir a autenticidade das requisições. Ou seja, para garantir que elas estão sendo, de fato, originadas a partir do API Gateway.

- **Authorizers** Aqui podemos utilizar um serviço de autenticação (Cognito ou mesmo uma função lambda) para restringir à API.

- **Models:** Aqui podemos definir a estrutura esperada dos dados recebidos na requisição.

- **Documentation:** Permite que especifiquemos documentações para os endpoints de nossa API.

- **Dashboard:** Informações sobre uso.

**(Manipulação de requisição e resposta)** Como vimos, ao especificar um método para um endpoint, temos quatro quadros. Veja a finalidade de cada um a seguir. Para visualizar os quadros, certifique-se de selecionar o método (GET, neste exemplo que criamos) no menu Resources.

- **Method Request:** Essa fase do fluxo pode filtrar requisições e bloquear aquelas que não estiverem de acordo com critérios pré definidos. Por exemplo, aqui especificamos se uma chave de API é requerida, se parâmetros na URL são requeridos (e se sim, quais são) entre outras coisas.

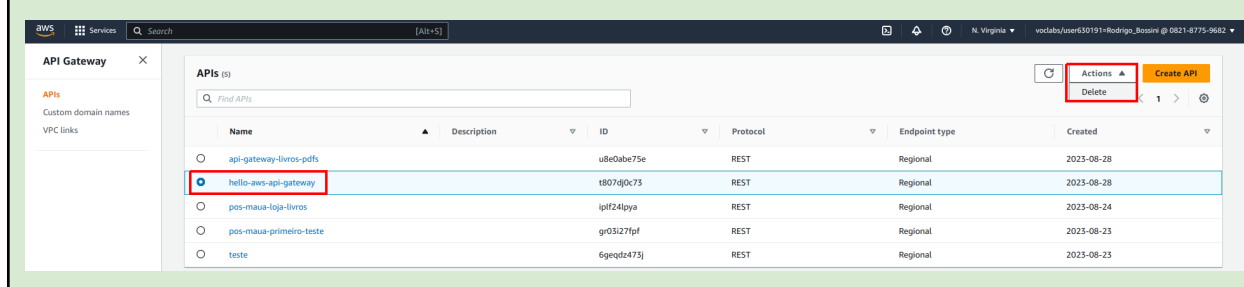
- **Integration Request:** Caso a requisição passe pela fase **Method Request**, ela é entregue para a fase Integration Request. Nesta fase, podemos realizar transformações em geral envolvendo os dados recebidos na requisição para entregar para o componente responsável por executar nossas regras de negócio. Por exemplo, podemos extrair os parâmetros de uma requisição GET e montar um objeto JSON para que ele possa ser entregue para uma função Lambda.

- **Integration Response** Esta fase entra em cena uma vez que a requisição tenha sido processada. Aqui podemos, por exemplo, **mapear códigos de status vindos do Back End para código que desejamos entregar para o cliente**. Por exemplo, verificar se o código de resposta do back end está de acordo com algum padrão (2xx, por exemplo) e mapeá-lo para um código que faça sentido para a nossa API. **Também podemos adicionar pares chave/valor ao cabeçalho da resposta**. Também é possível aplicar transformações aos dados recebidos do back end antes de entregá-los ao cliente.

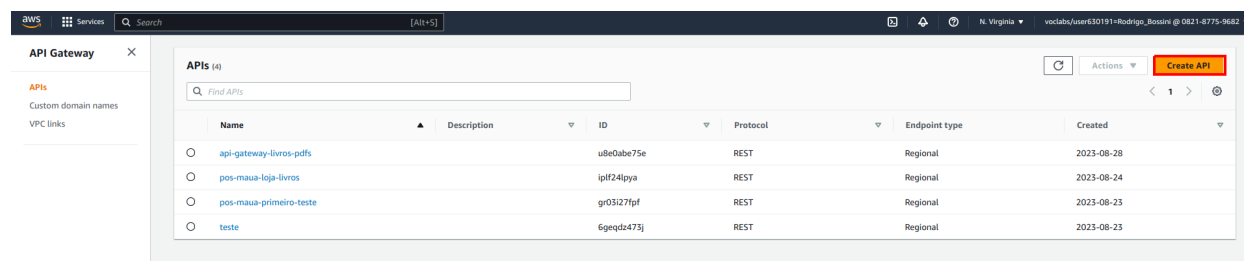


- **Method Response:** Esta fase define o que é público para o cliente. Por exemplo, quais headers, quais códigos, quais MIME types o endpoint pode devolver. É importante entender que nesta fase são feitas as **definições das características da resposta. Somente seu formato.** O preenchimento com os dados de fato ocorre na fase **Integration Response**.

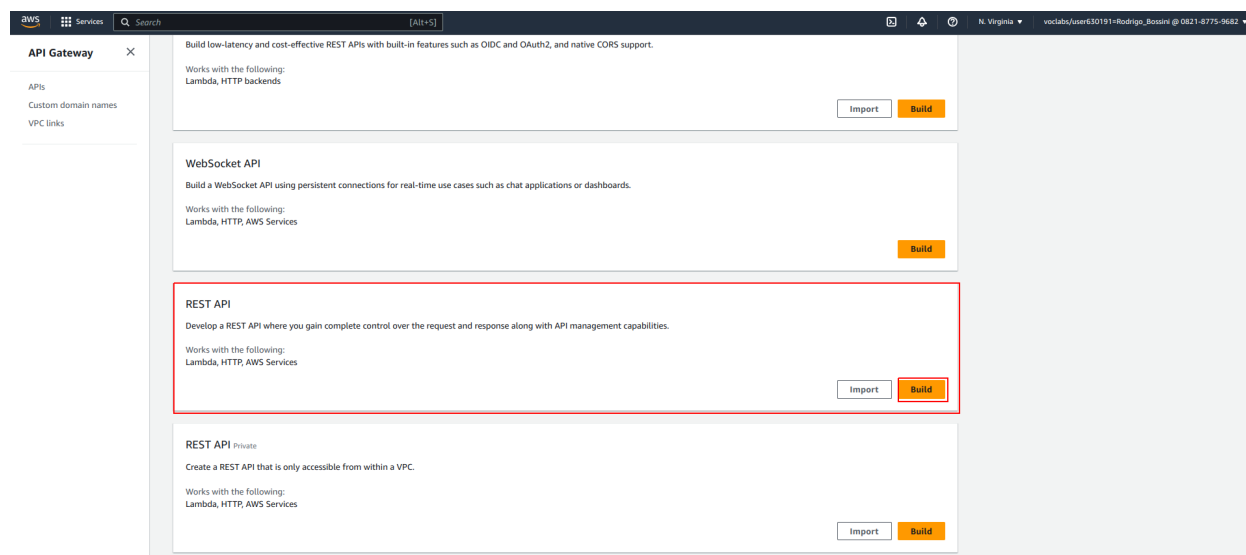
**Nota: Agora vamos criar uma nova API.** Caso queira apagar a API de teste que criamos, visite a página do API Gateway, selecione a API e clique em **Actions >> Delete**. Para encontrar a página inicial, você pode sempre buscar pelo nome do serviço - API Gateway - no Console AWS.



**(Criando uma nova API)** Na página principal do API Gateway, clique em **Create API**.



Clique na opção **Build de REST API** (não private).



A tela seguinte mostra quatro opções.

- **New API:** Já utilizamos uma vez e vamos utilizar novamente. Usada para criar uma API “do zero”.
- **Clone from existing API:** Permite fazer uma cópia de uma API existente.
- **Import from Swagger:** Permite importar um arquivo textual que descreve uma API criada pelo Swagger. Veja um exemplo a seguir. Basta clicar no pequeno link com extensão json nesta página.

<https://petstore.swagger.io/>

The screenshot displays the Swagger Petstore API interface. At the top, the Swagger logo is visible, followed by the URL `https://petstore.swagger.io/v2/swagger.json` and an **Explore** button. The main heading is **Swagger Petstore** with a version badge **1.0.6**. Below this, a red box highlights the `Base URL: petstore.swagger.io/v2` and the `https://petstore.swagger.io/v2/swagger.json` link. A paragraph explains that this is a sample server and provides links for more information, including `http://swagger.io`, `irc.freenode.net, #swagger`, and an API key `special-key` for testing authorization filters. Links for [Terms of service](#), [Contact the developer](#), [Apache 2.0](#), and [Find out more about Swagger](#) are also present.

Under the **Schemes** section, **HTTPS** is selected, and an **Authorize** button is available. The **pet** section, titled "Everything about your Pets", lists three endpoints:

- POST** `/pet/{petId}/uploadImage` uploads an image
- POST** `/pet` Add a new pet to the store
- PUT** `/pet` Update an existing pet

At the bottom, the Swagger JSON definition is shown in a code editor. The JSON structure includes:

```
// 20230828175406
// https://petstore.swagger.io/v2/swagger.json

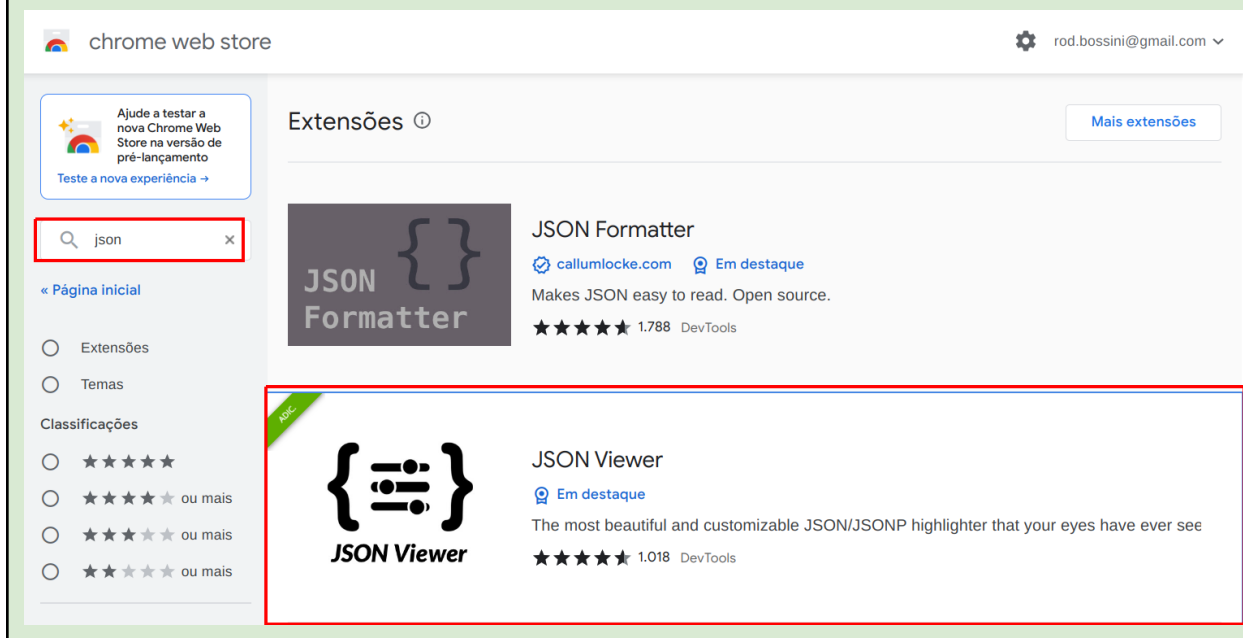
{
  "swagger": "2.0",
  "info": {
    "description": "This is a sample server Petstore server. You can find out more about Swagger at [http://swagger.io](http://swagger.io) or on [irc.freenode.net, #swagger](http://swagger.io/irc/). For this sample, you can use the api key `special-key` to test the authorization filters.",
    "version": "1.0.6",
    "title": "Swagger Petstore",
    "termsOfService": "http://swagger.io/terms/",
    "contact": {
      "email": "apiteam@swagger.io"
    },
    "license": {
      "name": "Apache 2.0",
      "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
    }
  },
  "host": "petstore.swagger.io",
  "basePath": "/v2",
  "tags": [
    {
      "name": "pet",
      "description": "Everything about your Pets",
      "externalDocs": {
        "description": "Find out more",
        "url": "http://swagger.io"
      }
    }
  ]
}
```

Lembre-se que esse é apenas um exemplo usando Swagger, ferramenta que não estamos utilizando neste material.

**Nota.** É possível que o seu objeto JSON não seja exibido formatado como na figura. Para que essa formatação seja realizada, você pode instalar uma extensão no Google Chrome. Comece visitando a Chrome Web Store.

<https://chrome.google.com/webstore/>

Busque por JSON e instale a extensão JSON Viewer.



- **Example API:** Mostra uma API de exemplo em que você pode se basear.

Escolha **New API**. Preencha os campos como a seguir.

**API name:** loja-livros

**Description:** Loja de livros

**Endpoint Type:** Regional (otimizado para funcionar dentro da região escolhida. Edge Optimized faria com que ele fosse distribuído em regiões diferentes usando o CloudFront, visando reduzir a latência).

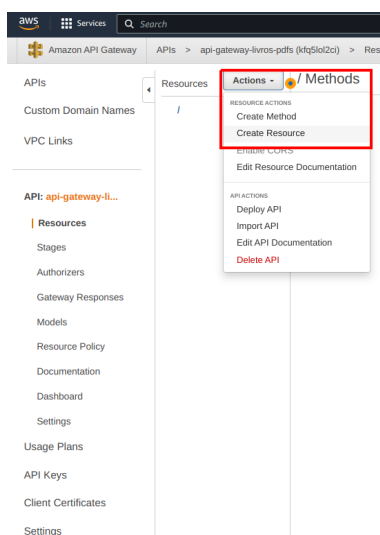
The screenshot shows the AWS API Gateway 'Create' console. The 'Choose the protocol' section has 'REST' selected. The 'Create new API' section has 'New API' selected. The 'Settings' section shows the following fields:

API name*	api-gateway-livros-pdfs
Description	Loja de livros
Endpoint Type	Regional

A 'Create API' button is visible at the bottom right.

**(Adicionando o recurso /livros, CORS e OPTIONS)** O próximo passo é adicionar os recursos da API.

- Clique **Actions >> Create Resource**.



Preencha os campos como a seguir e clique em **Create Resource**.

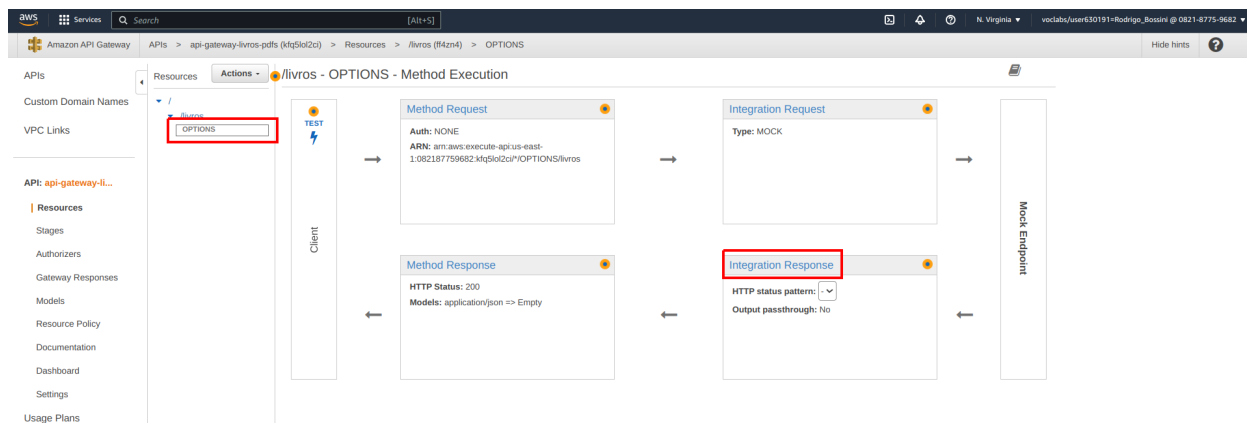
**Resource Name:** livros

**Resource path:** livros (name e path não precisam ser iguais mas pode simplificar)

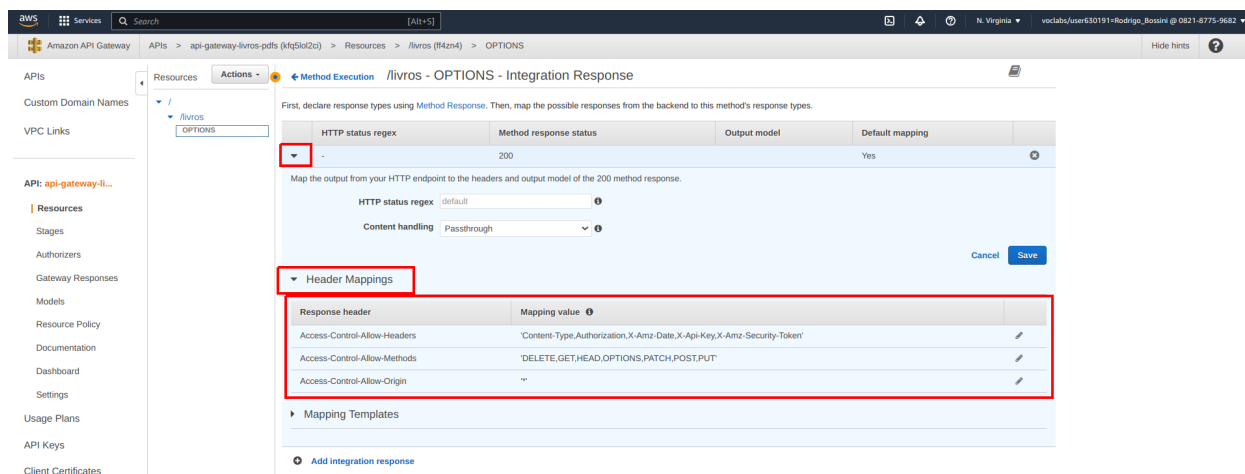
**Proxy Resource:** não marcar

**Enable API Gateway CORS:** marcar

A configuração do mecanismo CORS é feita no cabeçalho da resposta. Podemos verificar mantendo o método OPTIONS selecionado e clicando em **Integration Response**.

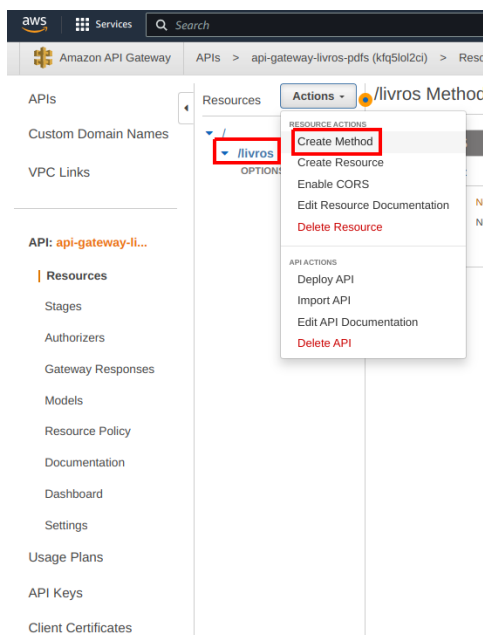


Clique na pequena seta para expandir e, então, clique em **Header Mappings**. Veja, ali, os pares chave/valor de configuração do mecanismo CORS.

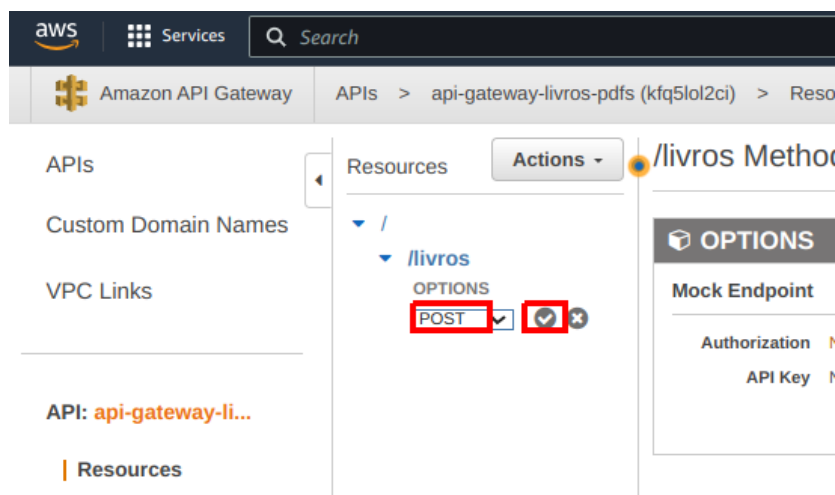


Apenas observe essas configurações. Não há nada a fazer nesta área, por enquanto.

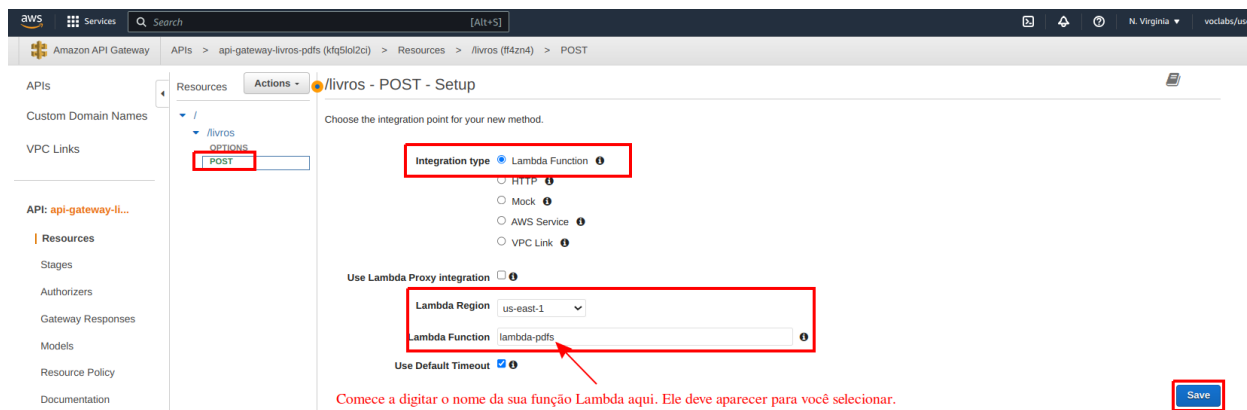
**(Armazenamento de um livro com o método POST e uma função Lambda) POST** é o método do protocolo HTTP apropriado para realizar operações de criação de conteúdo. Façamos a sua criação, associada ao recurso **livros**, já criado. **Certifique-se de que o endpoint /livros está selecionado** (e não a raiz /) e clique **Actions >> Create Method**.



Escolha o método **POST** no pequeno menu que aparecerá e clique no botão logo à frente dele para confirmar.

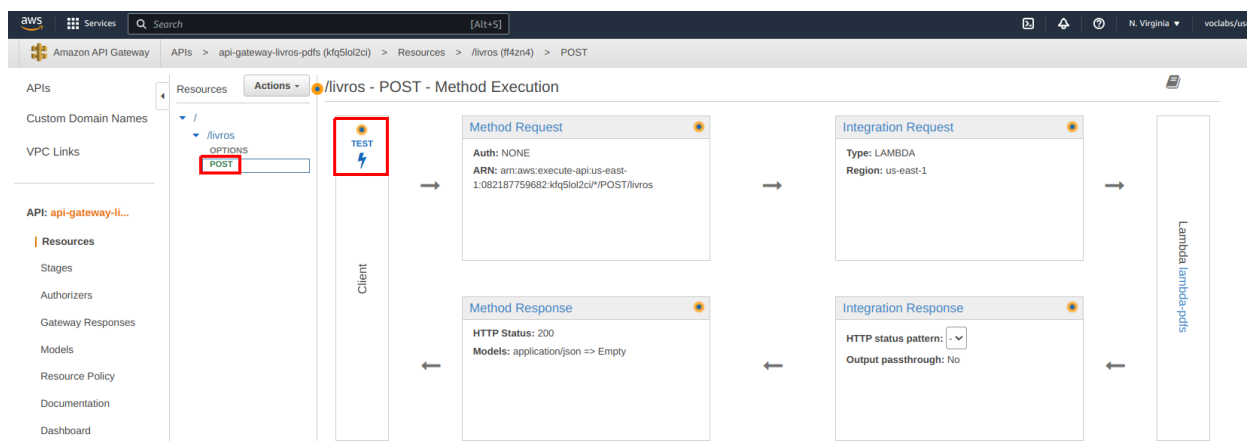


Mantenha o método **POST** selecionado para podermos escolher o **Integration Type**. Ou seja, mediante uma requisição POST neste Endpoint, precisamos especificar o que desejamos fazer. Neste caso, desejamos colocar a função lambda em funcionamento. Faça as escolhas como a seguir. Observe que você precisa começar a digitar o nome de sua função lambda para que ela apareça e possa ser selecionada.

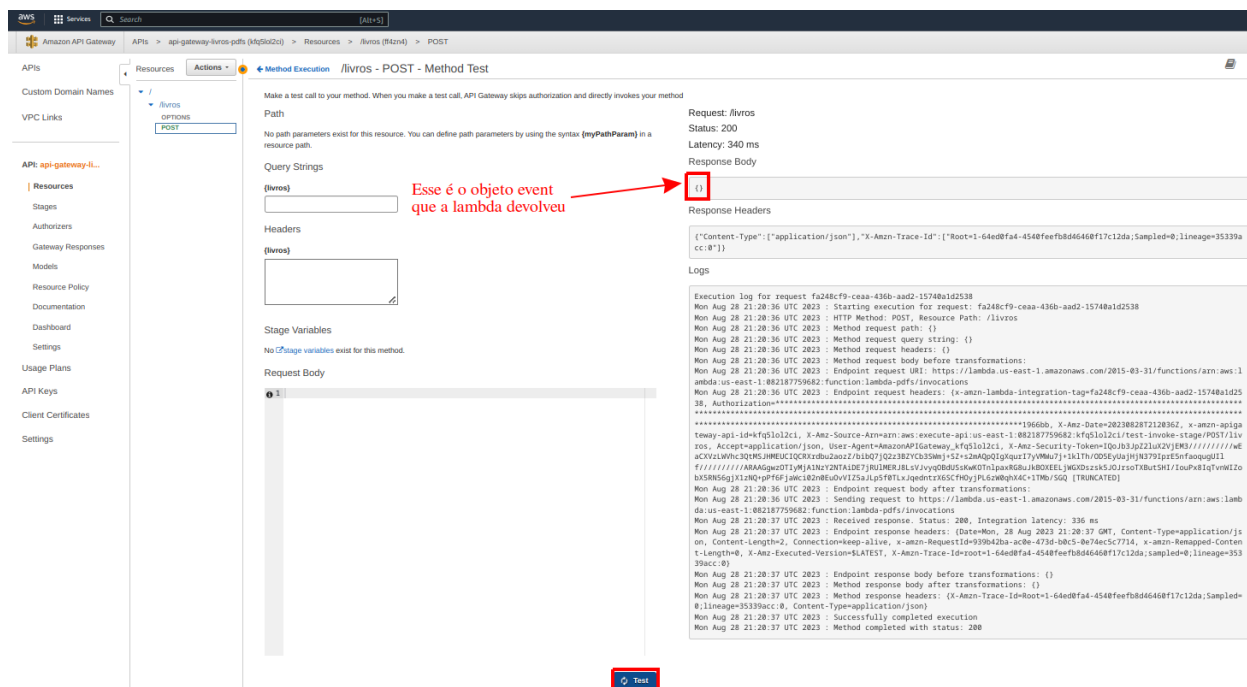




A seguir, mantendo POST selecionado, clique em **Test**.



Clique novamente **Test**, agora na parte inferior da tela e observe o resultado. A função Lambda está devolvendo o objeto **event**. Ele representa aquilo que enviamos no corpo da requisição quando ela foi realizada. Neste momento, não enviamos coisa alguma. Por isso, o objeto está vazio, sem nenhum par chave/valor.



### ***Referências***

- [1] Amazon Web Services (AWS) - Cloud Computing Services. 2023. Disponível em <<https://aws.amazon.com/>>. Acesso em agosto de 2023.
- [2] PiCloud Launches Serverless Computing Platform To The Public | TechCrunch. 2023. Disponível em <<https://techcrunch.com/2010/07/19/picloud-launches-serverless-computing-platform-to-the-public/>>. Acesso em agosto de 2023.
- [3] Serverless Architectures. 2023. Disponível em <<https://martinfowler.com/articles/serverless.html>>. Acesso em agosto de 2023.
- [4] Who coined the term ‘serverless’?. 2023. Disponível em <<https://www.quora.com/Who-coined-the-term-serverless>>. Acesso em agosto de 2023.