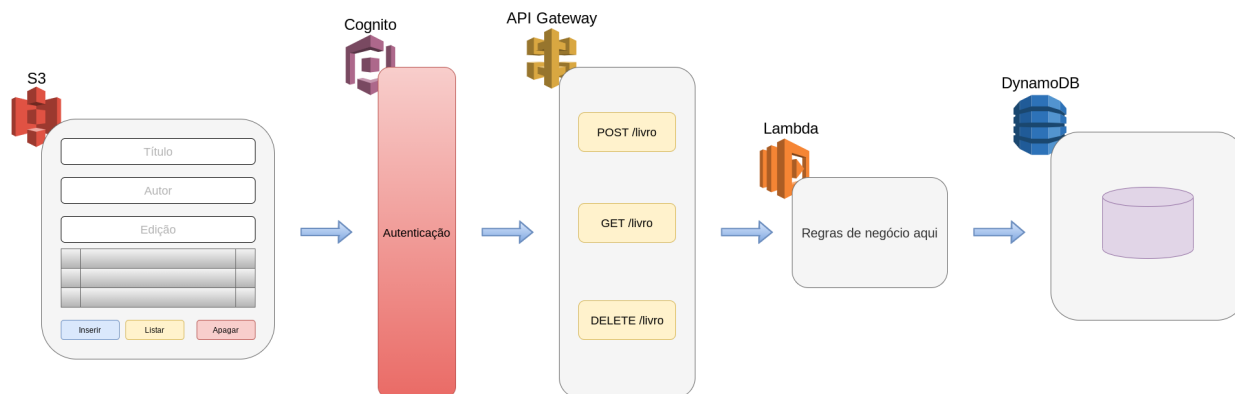


1 Introdução

Neste material, prosseguimos com o desenvolvimento da solução retratada a seguir.



2 Desenvolvimento

(Sobre o DynamoDB) O DynamoDB é um serviço de armazenamento de dados gerenciado da Amazon. Algumas de suas características são:

- Por ser um serviço gerenciado, não nos cabe fazer atualizações de sistema operacional ou mesmo do próprio sistema gerenciador de banco de dados. A escalabilidade também fica por conta da Amazon.
- Os dados que armazenamos são **criptografados** automaticamente.
- Todos os dados são armazenados em **SSDs**.
- Todos os dados são automaticamente **replicados** em **múltiplas zonas de disponibilidade** dentro de uma mesma região.
- **Podemos** escolher replicar os dados **globalmente**, em **diversas regiões**.
- Seu **modelo de dados** baseia-se no seguinte:
- **Tabelas**: Similar a tabelas de outros sistemas de bancos de dados.
- **Itens**: Cada tabela contém zero ou mais itens. Um item é um grupo de atributos que pode ser diferenciado de todos os demais. Em uma tabela que armazena pessoas, por exemplo, um item é uma pessoa.

- **Atributos:** Um dado “**escalar**”, por exemplo uma string, um número inteiro, um valor booleano etc.

- **Não** há “**schema**” pré definido. Isso quer dizer que não é necessário definir os atributos que os itens de uma tabela têm e nem o seu tipo.

- As tabelas necessariamente possuem **chave primária**. A chave primária de uma tabela pode ser simples (um único atributo) ou composta (dois atributos).

- Uma chave primária **simples** leva o nome de **Partition Key**. Em função de seu valor, o DynamoDB calcula um código hash que serve para determinar a localização física daquele item no mecanismo interno do sistema.

- Uma chave primária **composta** possui uma **Partition Key** e uma **Sort Key**. O código hash calculado em função de sua Partition Key também serve para determinar a localização física do item. Todos os itens com Partition Key igual são armazenados juntos, ordenados de acordo com a Sort Key. Uma tabela que possui chave primária composta pode conter valores repetidos na Partition Key, desde que os valores de Sort Key sejam diferentes.

As figuras a seguir mostram exemplos de tabelas com chaves primária simples e composta.

Chave primária simples

People

<pre>{ "PersonID": 101, "LastName": "Smith", "FirstName": "Fred", "Phone": "555-4321" }</pre>
<pre>{ "PersonID": 102, "LastName": "Jones", "FirstName": "Mary", "Address": { "Street": "123 Main", "City": "Anytown", "State": "OH", "ZipCode": 12345 } }</pre>
<pre>{ "PersonID": 103, "LastName": "Stephens", "FirstName": "Howard", "Address": { "Street": "123 Main", "City": "London", "PostalCode": "ER3 5K8" }, "FavoriteColor": "Blue" }</pre>

Chave primária composta

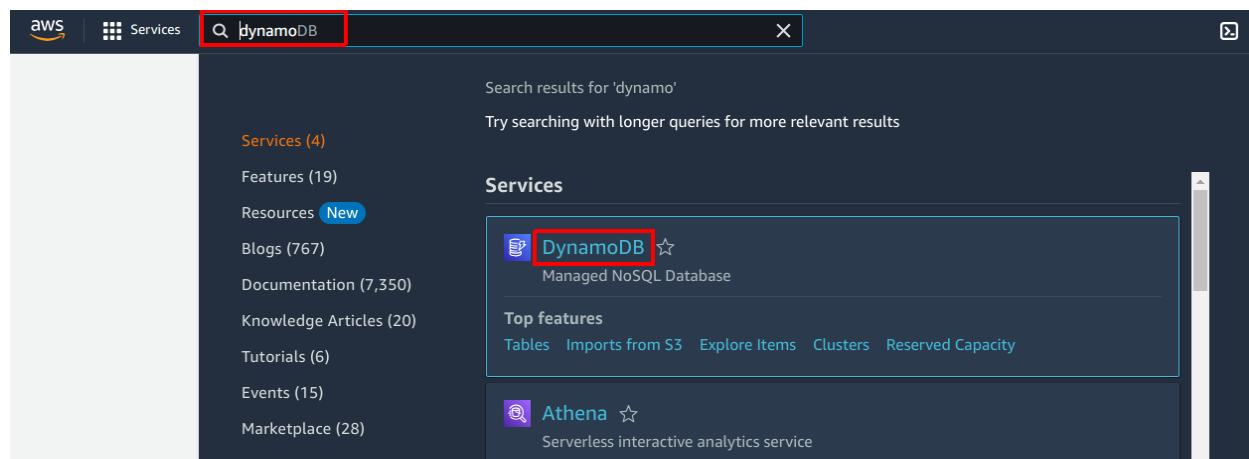
Music

<pre>{ "Artist": "No One You Know", "SongTitle": "My Dog Spot", "AlbumTitle": "Hey Now", "Price": 1.98, "Genre": "Country", "CriticRating": 8.4 }</pre>
<pre>{ "Artist": "No One You Know", "SongTitle": "Somewhere Down The Road", "AlbumTitle": "Somewhat Famous", "Genre": "Country", "CriticRating": 8.4, "Year": 1984 }</pre>
<pre>{ "Artist": "The Acme Band", "SongTitle": "Still in Love", "AlbumTitle": "The Buck Starts Here", "Price": 2.47, "Genre": "Rock", "PromotionInfo": { "RadioStationsPlaying": ["KHCR", "KQBX", "WTNR", "WJJH"], "TourDates": { "Seattle": "20150625", "Cleveland": "20150630" }, "Rotation": "Heavy" } }</pre>
<pre>{ "Artist": "The Acme Band", "SongTitle": "Look Out, World", "AlbumTitle": "The Buck Starts Here", "Price": 0.99, "Genre": "Rock" }</pre>

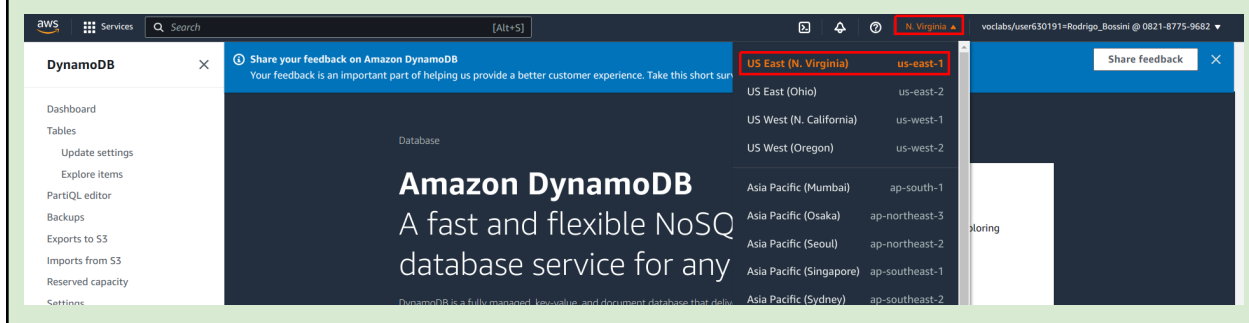
Leia mais sobre o DynamoDB na documentação.

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html>

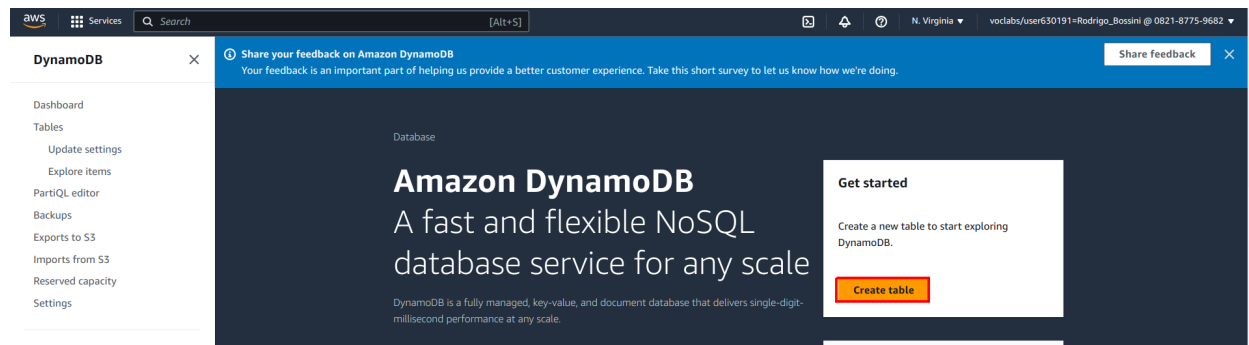
Utilizaremos o DynamoDB para armazenar os dados envolvidos em nossa aplicação em meio persistente. No console AWS, busque por DynamoDB. **Clique em seu nome segurando CTRL no teclado. Assim você terá uma aba exclusiva para manipular o DynamoDB.**



Nota: O DynamoDB é um serviço regional. Por isso, verifique no canto superior direito se a região selecionada é aquela que você deseja utilizar.



Para criar uma nova tabela, clique em **Create table.**



Use **Livro** como nome da tabela e **LivroID** como a sua chave primária. A tabela terá uma chave primária simples. Portanto, use apenas o campo **Partition Key**. O tipo dela é **String**.

The screenshot shows the AWS Management Console for DynamoDB, specifically the 'Create table' page. The breadcrumb navigation at the top indicates the path: **DynamoDB** > **Tables** > **Create table**. The main heading is 'Create table'. Below this, there's a section titled 'Table details' with an 'Info' link. A note states: 'DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.'

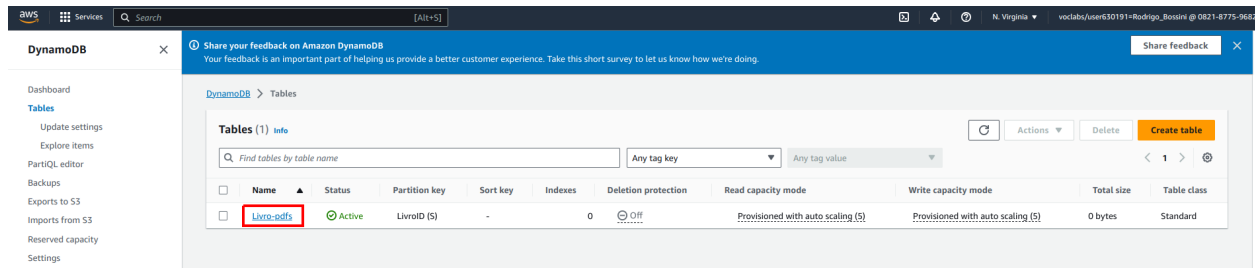
The 'Table name' section includes a text box containing 'Livro-pdfs'. A note below it says: 'This will be used to identify your table.' and 'Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.)'.

The 'Partition key' section includes a text box containing 'LivroID' and a dropdown menu set to 'String'. A note below it says: 'The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.' and '1 to 255 characters and case sensitive.'

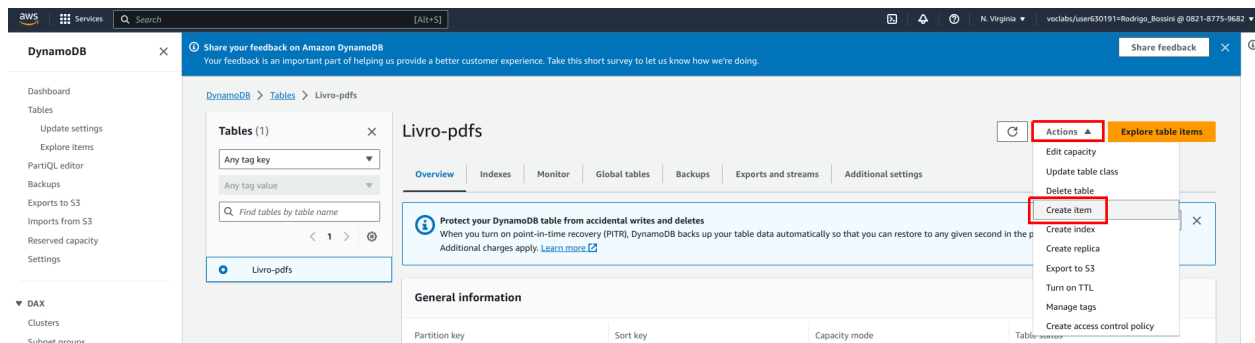
The 'Sort key - optional' section includes a text box with the placeholder 'Enter the sort key name' and a dropdown menu set to 'String'. A note below it says: 'You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.' and '1 to 255 characters and case sensitive.'

Mais abaixo, mantenha a opção **Use default settings**. Clique em **Create table**.

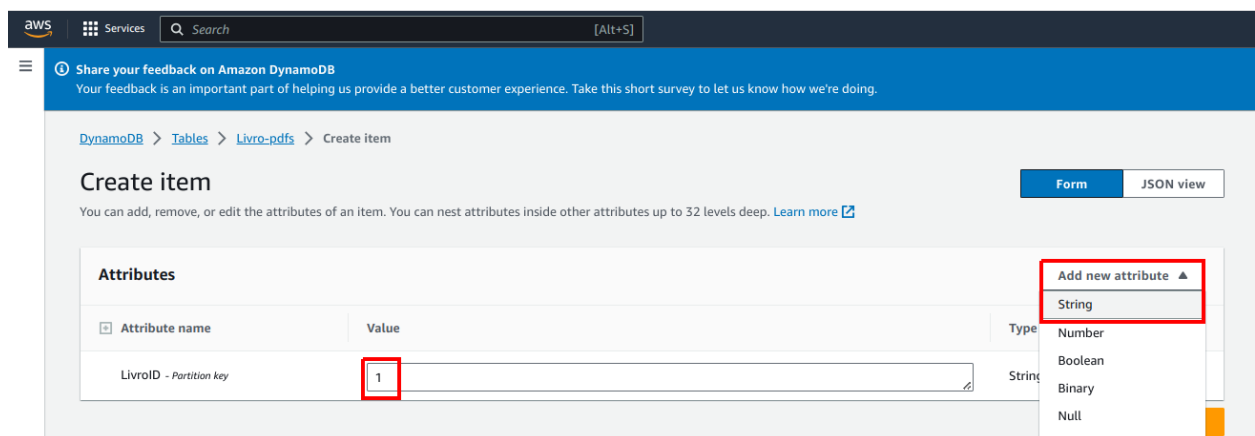
Clique no nome da tabela.



(Criando um item) Um item é semelhante a uma linha de uma tabela no modelo relacional. É uma instância da entidade representada por aquela tabela. Criemos um item manualmente em nossa tabela de livros. Para isso, clique em **Actions >> Create Item**.



Coloque o número 1 (ele será armazenado como String) como chave primária. Depois, clique em **Add new attribute** para adicionar um novo atributo. Ele será do tipo String. Um atributo é semelhante a uma coluna no modelo relacional.



Repita o passo a passo para criação de novos atributos (**Add new attribute >> String**) que caracterizam um livro. Todos eles serão do tipo String. Ao final, clique em **Create item**.

aws Services Search [Alt+S]

Share your feedback on Amazon DynamoDB
Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.

DynamoDB > Tables > Livro-pdfs > Create item

Create item

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)

Form JSON view

Attributes Add new attribute

Attribute name	Value	Type	
LivroID - Partition key	1	String	
titulo	Concrete Mathematics	String	Remove
autor	Donald Knuth	String	Remove
edicao	1	String	Remove

Cancel Create item

Clique sobre a chave primária do item para visualizá-lo.

aws Services Search [Alt+S]

DynamoDB

Share your feedback on Amazon DynamoDB
Your feedback is an important part of helping us provide a better customer experience. Take this short survey to let us know how we're doing.

DynamoDB > Explore items > Livro-pdfs

Tables (1) ×

Any tag key

Any tag value

Find tables by table name

1

Livro-pdfs

Livro-pdfs

Autopreview View table details

Scan or query items
Expand to query or scan items.

Completed. Read capacity units consumed: 0.5

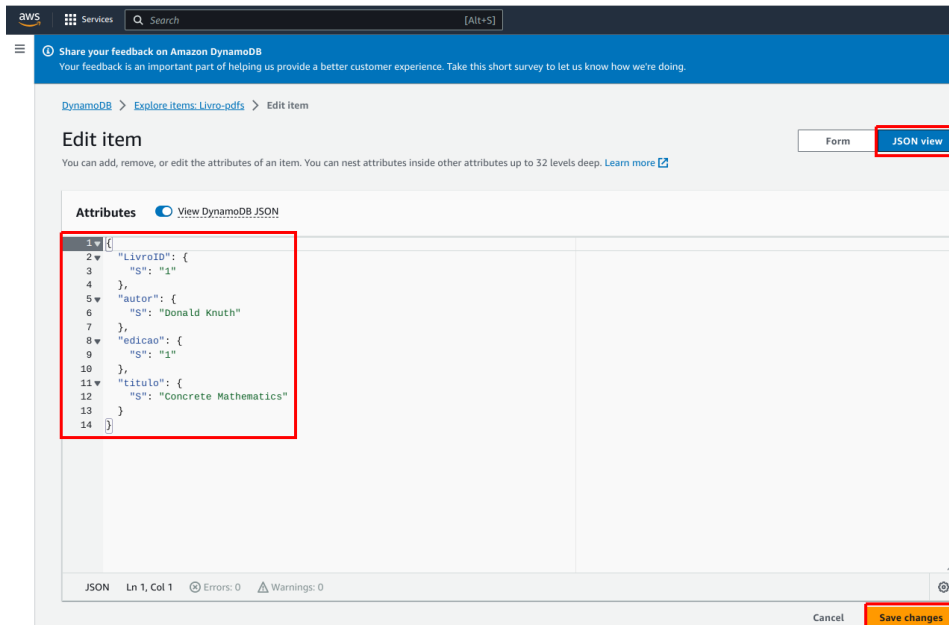
Items returned (1)

Actions Create item

1

	LivroID (String)	autor	edicao	titulo
	1	Donald Knuth	1	Concrete Mathematics

Observe que ele pode ser visualizado como um objeto JSON.



Clique em **Cancel** quando terminar de visualizar o item.

Nota. A letra S representa o tipo do atributo. Neste caso, String. Outros possíveis valores são

N: Number

B: Binary

BOOL: booleano

M: Mapa (coleção de pares chave/valor que fica aninhada no item)

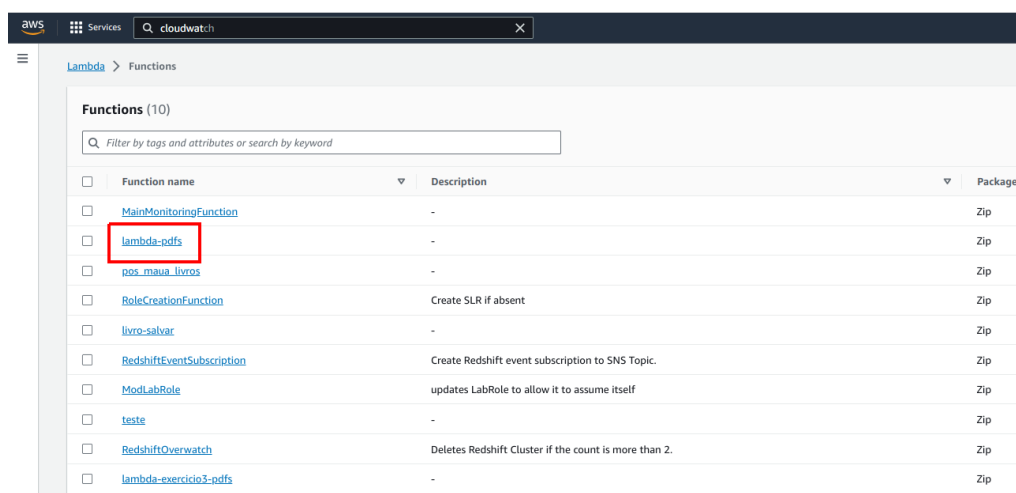
L: Lista

Veja mais em

https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_AttributeValue.html

(Acessando o DynamoDB a partir de uma função Lambda) Os serviços AWS podem ser acessados “programaticamente” por meio de SDKs que a Amazon disponibiliza para diferentes linguagens. Quando vamos acessar um dos serviços em uma função Lambda, as funcionalidades dos SDKs já estão automaticamente disponíveis, não há a necessidade de fazer qualquer instalação. Neste passo, vamos editar a nossa função Lambda. Ela será responsável por fazer a inserção de um livro. Lembre-se que ela está sendo associada pelo API Gateway quando ele recebe uma requisição do tipo POST direcionada ao recurso /livros.

Comece visitando a sua lista de funções Lambda e escolhendo aquela em que temos trabalhado até então.



A seguir, vamos ajustar a nossa função para que ela deixe de operar utilizando **callbacks** e passe a utilizar a construção **async/await** do Javascript. Dessa forma, o código tende a ficar mais fácil de ler, manter e “debugar”.

Nota. O uso de funções callback tende a dar origem ao conhecido **inferno de callbacks**. Veja mais em

https://developer.mozilla.org/pt-BR/docs/Learn/Server-side/Express_Nodejs/Introduction

e aqui também

<http://callbackhell.com/>

Veja como ela fica.

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

O próximo passo é **importar**

DynamoDbClient: como o nome sugere, representa um cliente DynamoDB. Permite que executemos comandos para a criação de tabelas, criação de itens, remoção de itens etc.

DynamoDBDocumentClient: Empacota o DynamoDBClient. Ele se encarrega de fazer conversões entre tipos nativos do Javascript e do DynamoDB. Veja o que a documentação oficial fala sobre ele.

"The document client simplifies working with items in Amazon DynamoDB by abstracting away the notion of attribute values. This abstraction annotates native JavaScript types supplied as input parameters, as well as converts annotated response data to native JavaScript types."

<https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB/DocumentClient.html>

PutCommand: Representa um comando de inserção de item no DynamoDB. Ele será construído com informações como o nome da tabela e os valores que caracterizam um item a ser inserido. Depois, utilizamos um cliente para enviá-lo ao servidor.

Nota. Lembre-se da especificação de tipo no DynamoDB usando as letras S, M, N etc? O objeto PutCommand abstrai essa especificação. Ou seja, não temos de digitar esses tipos explicitamente.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";  
import { PutCommand, DynamoDBDocumentClient } from  
"@aws-sdk/lib-dynamodb";  
  
export const handler = async (event, context) => {  
  
};
```

A seguir, construímos o objeto `DynamoDBClient`. Observe que entregamos ao seu construtor um objeto em que poderíamos personalizar opções diversas. Um exemplo é a região. Neste momento, no entanto, não temos nada a personalizar. Por isso, passamos um objeto vazio.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from
"@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});

export const handler = async (event, context) => {

};
```

A seguir, construímos o objeto `DynamoDBDocumentClient`. Observe como ele “empacota” o objeto `DynamoDBClient`, provendo mais funcionalidades, como a conversão de tipos que mencionamos.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from
"@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const handler = async (event, context) => {

};
```

Precisamos de um id para o livro. Há diferentes formas para se obter um. Neste exemplo, observe que, a cada requisição recebida pela função Lambda, um identificador de requisição é gerado. Vamos utilizar este mesmo identificador como identificador do livro sendo inserido no momento. Ele pode ser obtido a partir do objeto **context** que a nossa função recebe e seu nome é **awsRequestId**.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from
"@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const handler = async (event, context) => {
  const id = context.awsRequestId;
};
```

A seguir, construímos o objeto **PutCommand** especificando o nome da tabela e os dados do item (o livro) a ser inserido. Observe como utilizamos o id da requisição e também os dados do objeto **event**.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from
"@aws-sdk/lib-dynamodb";

const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const handler = async (event, context) => {
  const id = context.awsRequestId;
  const command = new PutCommand({
    TableName: "Livro-pdfs",
    Item: {
      LivroID: context.awsRequestId,
      titulo: event.titulo,
      autor: event.autor,
      edicao: event.edicao
    },
  });
};
```

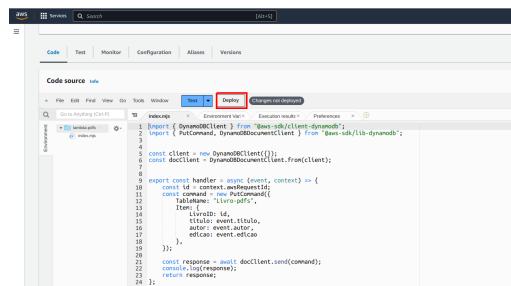
A seguir, podemos enviar o comando e “aguardar” a resposta com a construção **await**. Depois disso, devolvemos um objeto com a instrução **return**. O que ela devolve é equivalente àquilo que devolvíamos quando utilizávamos a função callback, usando seu segundo parâmetro.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from
"@aws-sdk/lib-dynamodb";

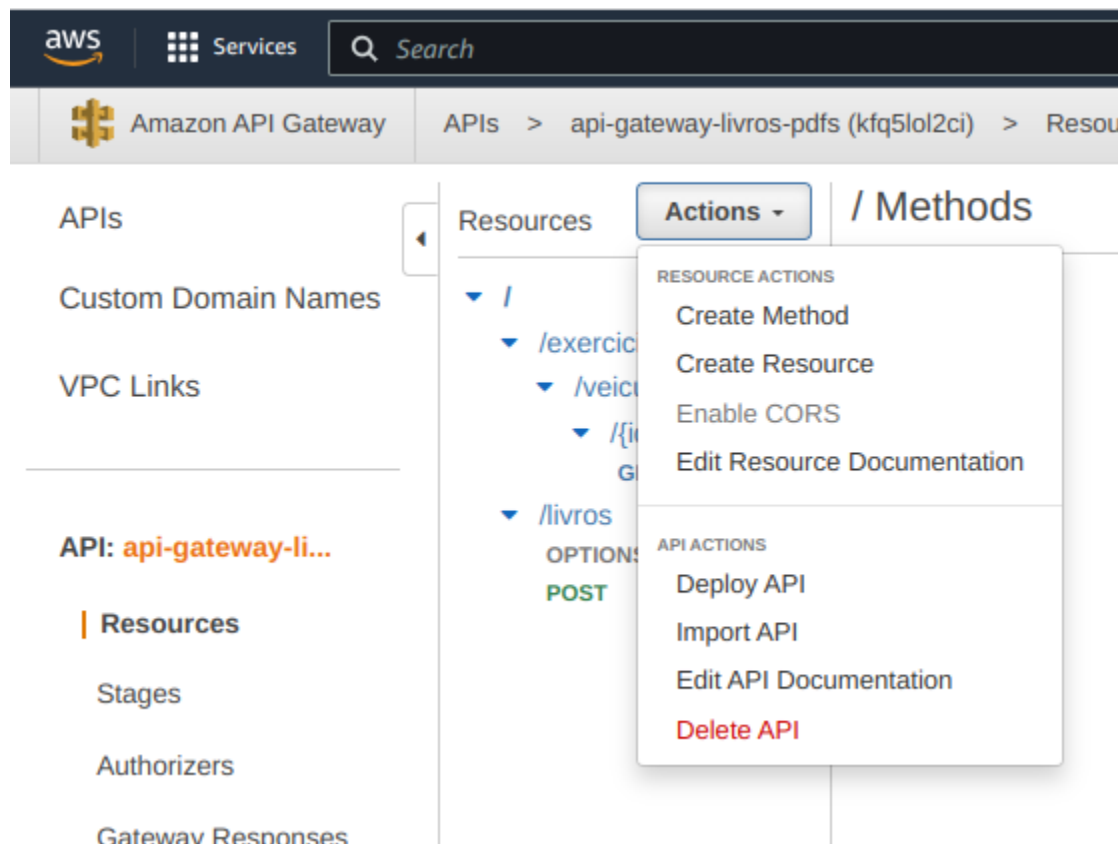
const client = new DynamoDBClient({});
const docClient = DynamoDBDocumentClient.from(client);

export const handler = async (event, context) => {
  const id = context.awsRequestId;
  const command = new PutCommand({
    TableName: "Livro-pdfs",
    Item: {
      LivroID: context.awsRequestId,
      titulo: event.titulo,
      autor: event.autor,
      edicao: event.edicao
    },
  });
  const response = await docClient.send(command);
  return response;
};
```

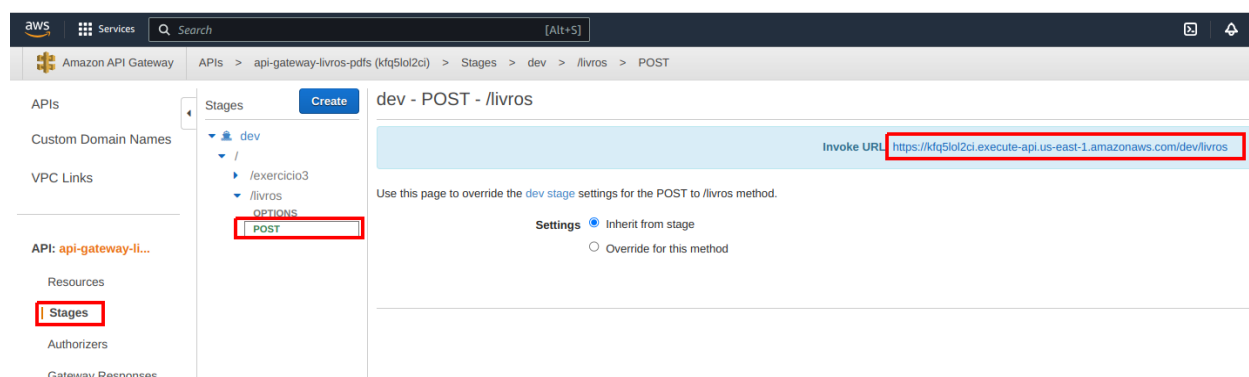
A função Lambda está pronta. Não se esqueça de clicar em **Deploy**.



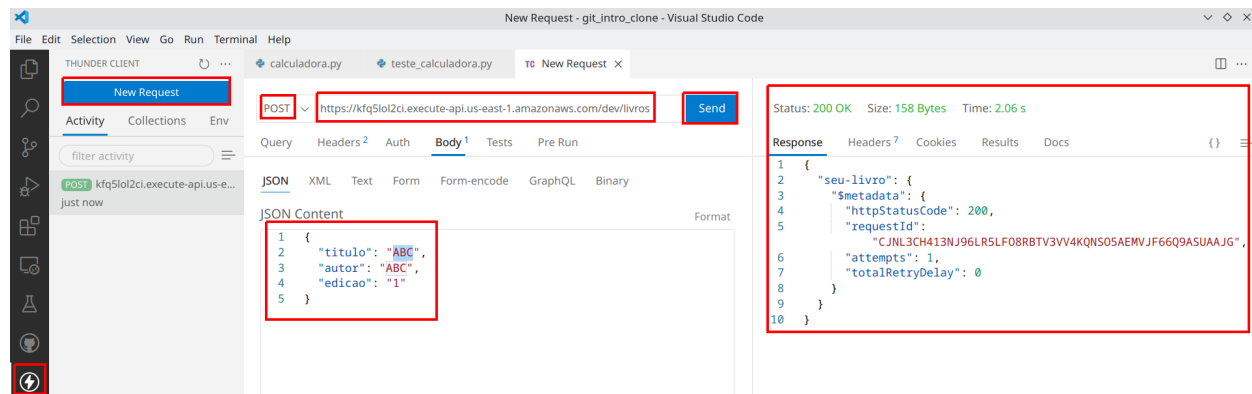
(**Teste com a Thunder Client**) Caso tenha realizado alguma alteração na sua API no API Gateway, faça novo deploy dela. Se estiver na dúvida, faça mesmo assim.



Depois disso, pegue o link de seu stage no API Gateway.

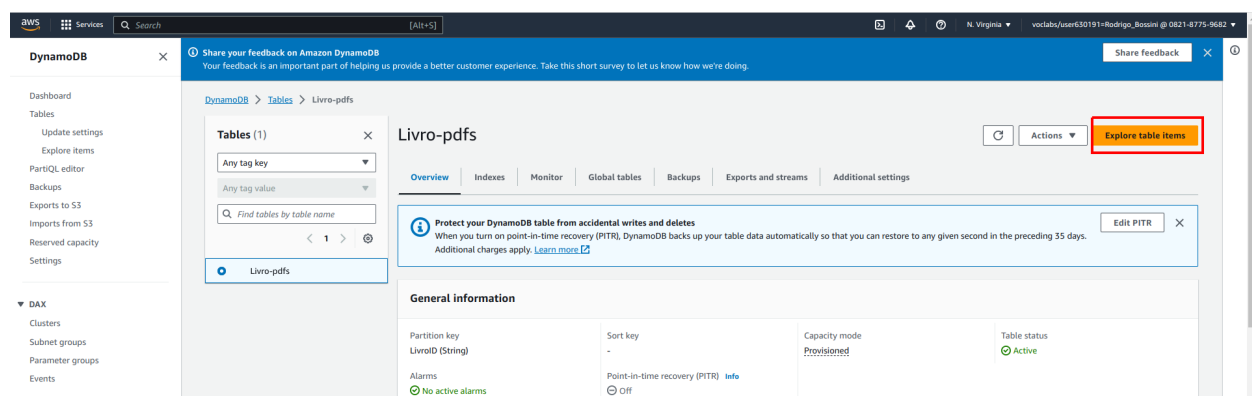
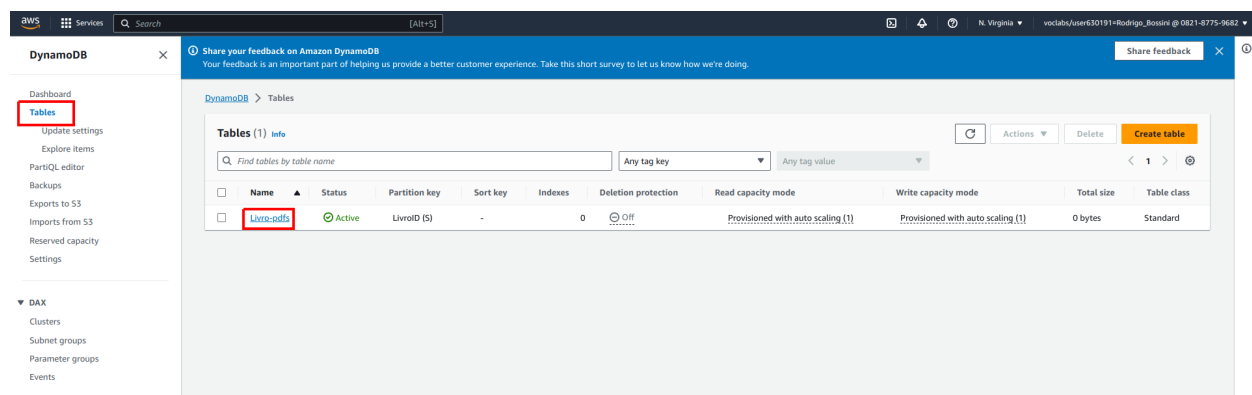


Faça uma nova requisição na Thunder Client.



Observe que o resultado inclui detalhes técnicos da requisição enviada ao DynamoDB. Isso está acontecendo pois a função Lambda realiza a requisição ao DynamoDB e, quando recebe a resposta, apenas a devolve ao API Gateway.

Visite também o DynamoDB e verifique se o novo livro foi cadastrado.



BWS Services [Alt+S] N. Virginia voclabs/user650191-Rodrigo_Bossini @ 0621-4775-9652

DynamoDB

Dashboard
Tables
Update settings
Explore items
 PartiQL editor
Backups
Exports to S3
Imports from S3
Reserved capacity
Settings

▼ DAX
Clusters
Subnet groups
Parameter groups
Events

DynamoDB > Explore items > Livro-pdfs

Tables (1)

Any tag key
Any tag value
Find tables by table name

Livro-pdfs

Livro-pdfs

Autopreview View table details

▼ Scan or query items

Scan Query

Select a table or index Table - Livro-pdfs Select attribute projection All attributes

Filters

Run Reset

Completed. Read capacity units consumed: 0.5

Items returned (3)

Copy Actions Create item

	LivroID (String)	autor	edicao	titulo
<input type="checkbox"/>	68252363-3876-4312-a4b8-dae0209837e	ABC	1	ABC
<input type="checkbox"/>	1	Donald Knuth	1	Concrete Mathematics
<input type="checkbox"/>	9a4871f4-f780-4a64-9e1b-29d11c20651d	abc	abc	abcd

Referências

- [1] Amazon Web Services (AWS) - Cloud Computing Services. 2023. Disponível em <<https://aws.amazon.com/>>. Acesso em setembro 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 setembro de 2023.
- [3] Serverless Architectures. 2023. Disponível em <<https://martinfowler.com/articles/serverless.html>>. Acesso em setembro de 2023.
- [4] Who coined the term 'serverless'?. 2023. Disponível em <<https://www.quora.com/Who-coined-the-term-serverless>>. Acesso em setembro de 2023.