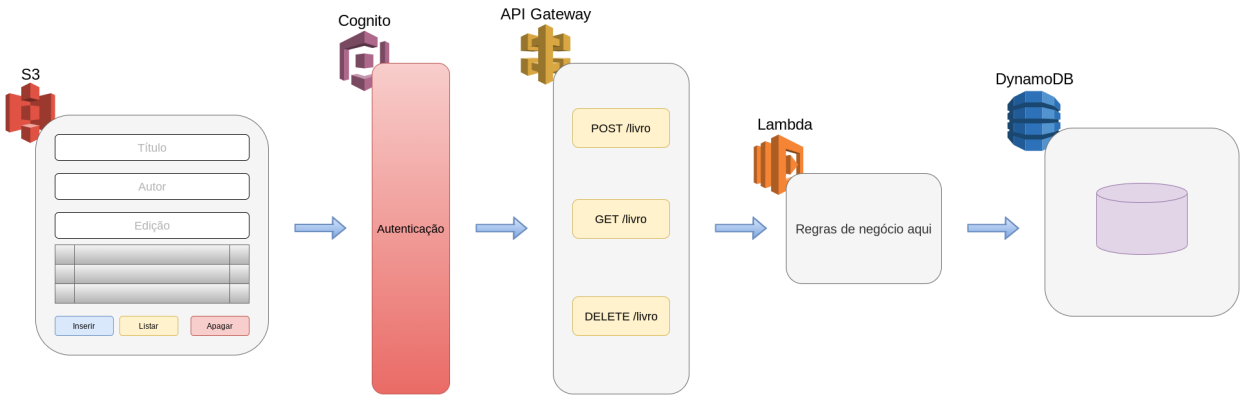
1 Introdução

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.

Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

- **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

Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

Chave primária composta

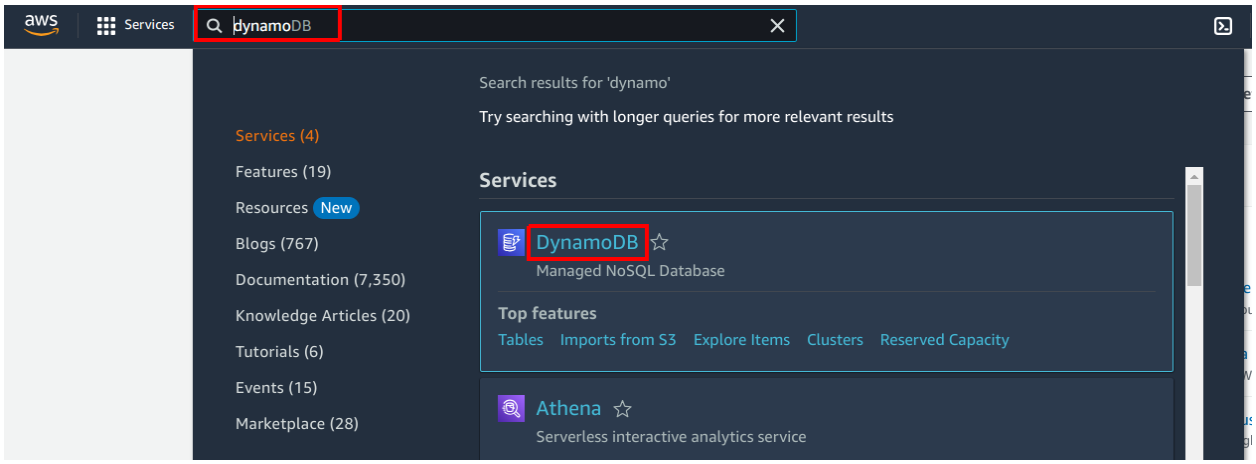
Music

Leia mais sobre o DynamoDB na documentação.

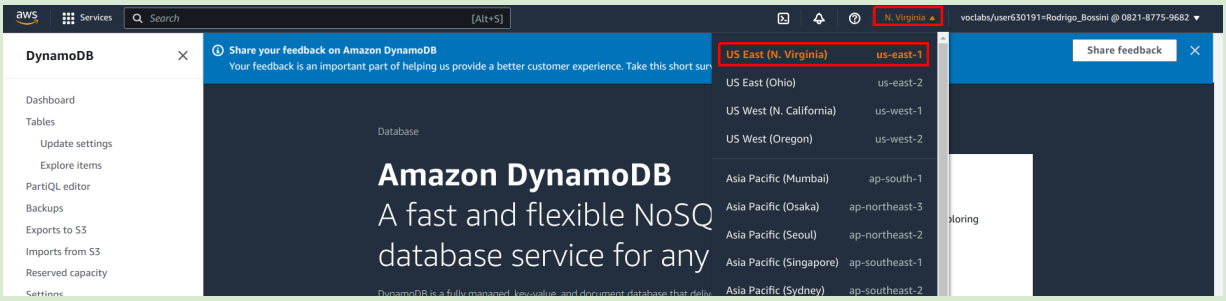
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/HowItWorks.CoreComponents.html

Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

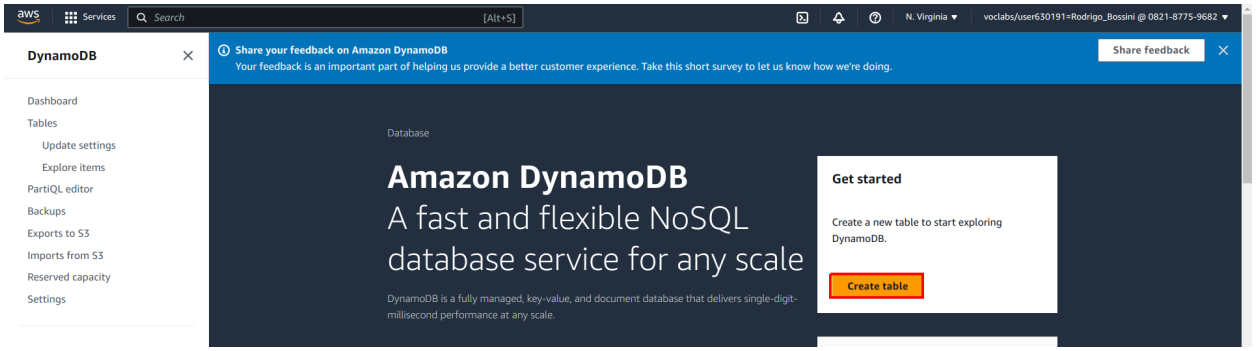
Utilizaremos o DynamoDB para armazenar os dados envolvidos em nossa aplicação em meio persistente. No console AWS, busque por DynamoDB. **Clique em seu nome segurando CTRL no teclado. Assim você terá uma aba exclusiva para manipular o DynamoDB.**



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

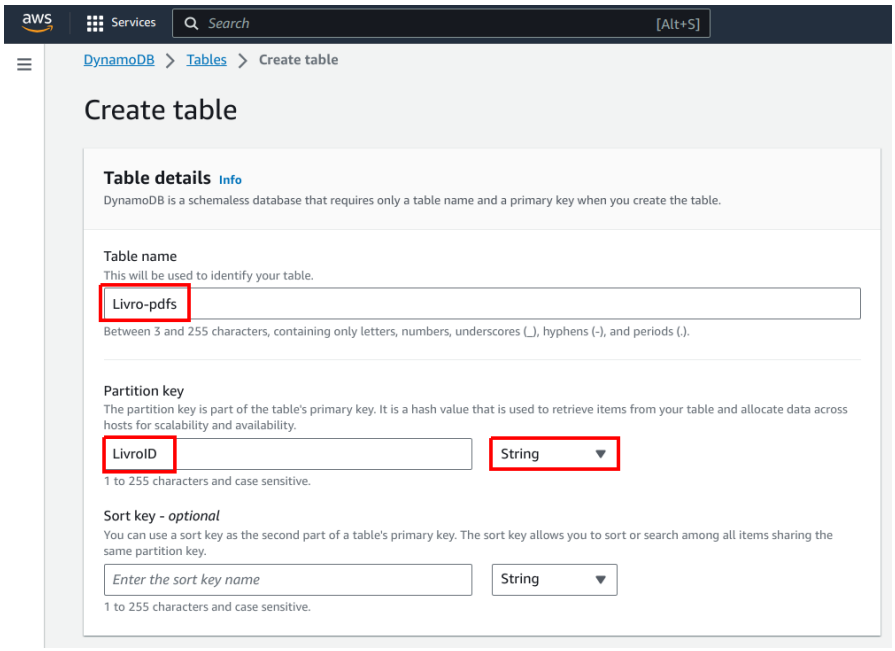


Para criar uma nova tabela, clique em **Create table**.



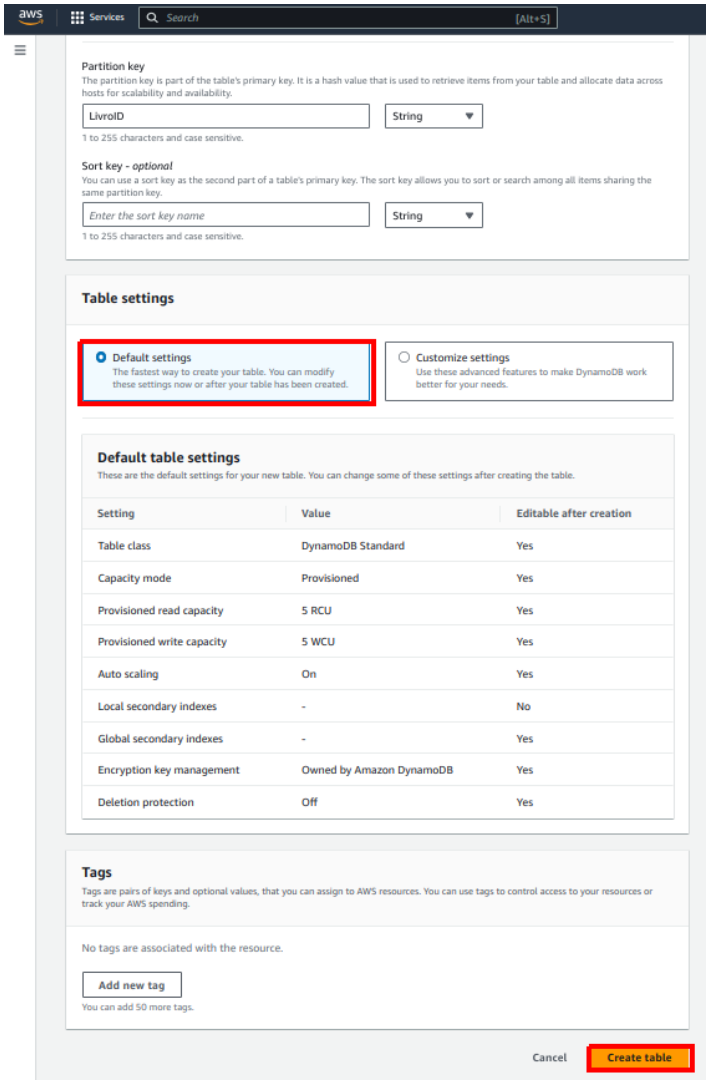
Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

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**.

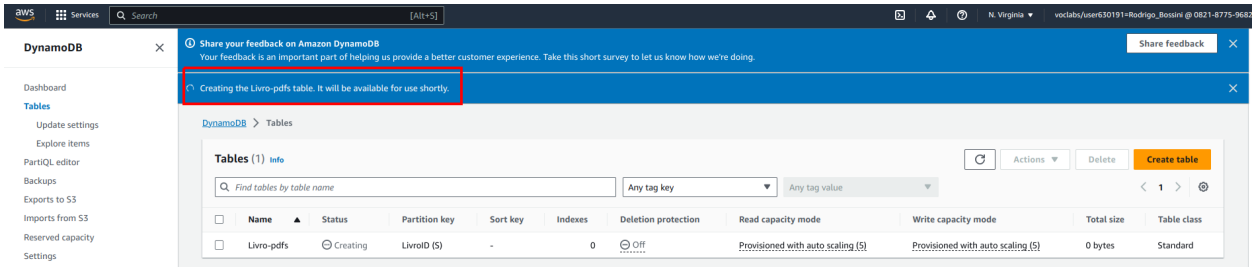


Mais abaixo, mantenha a opção **Use default settings**. Clique em **Create table.**

Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

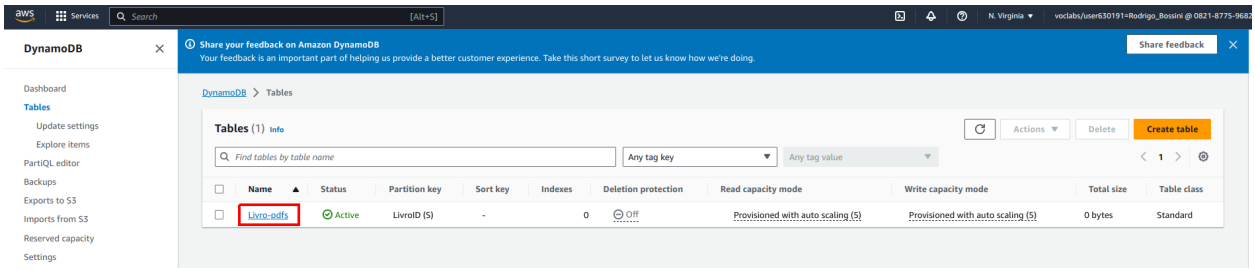


Segundo a mensagem a seguir, a criação da tabela pode demorar um pouco. Aguarde.

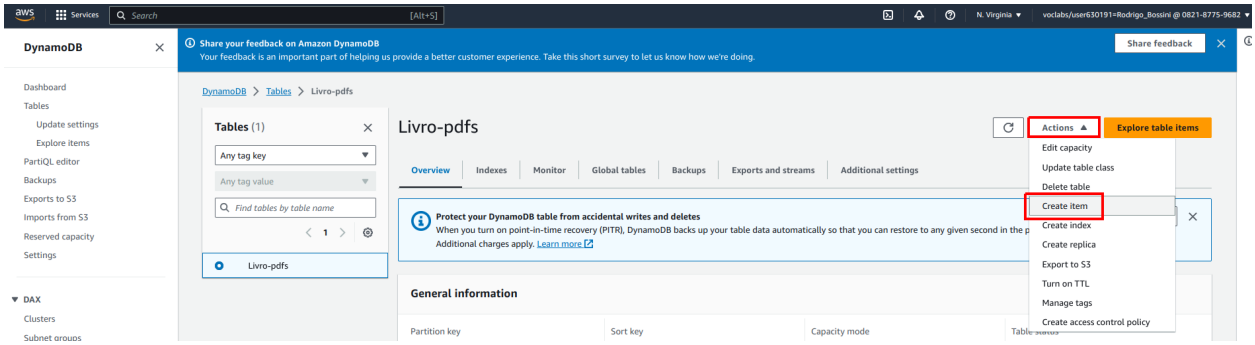


Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

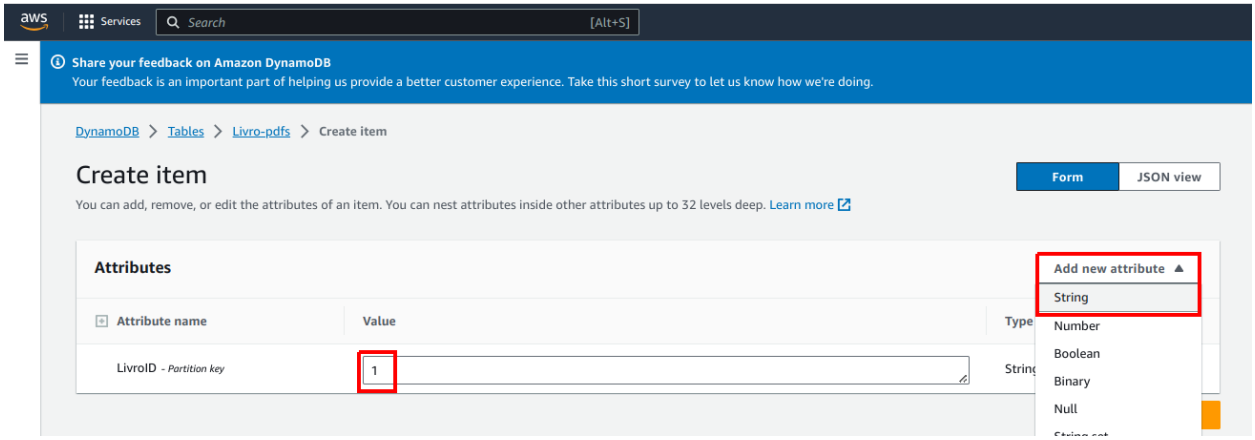
Clique no nome da tabela.



(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**.

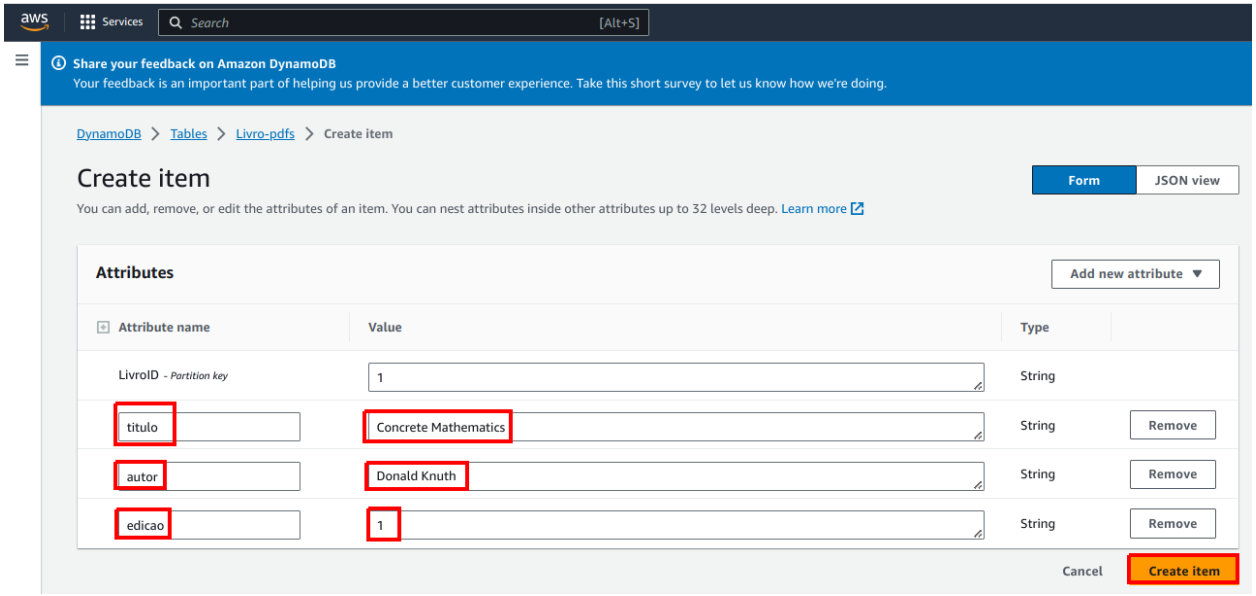


Coloque o número 1 (ele será armazenado como String) como chave primária. Depois, clique em **Add new attribute** para adicionar um novo atributo. Ele será do tipo String. Um atributo é semelhante a uma coluna no modelo relacional.

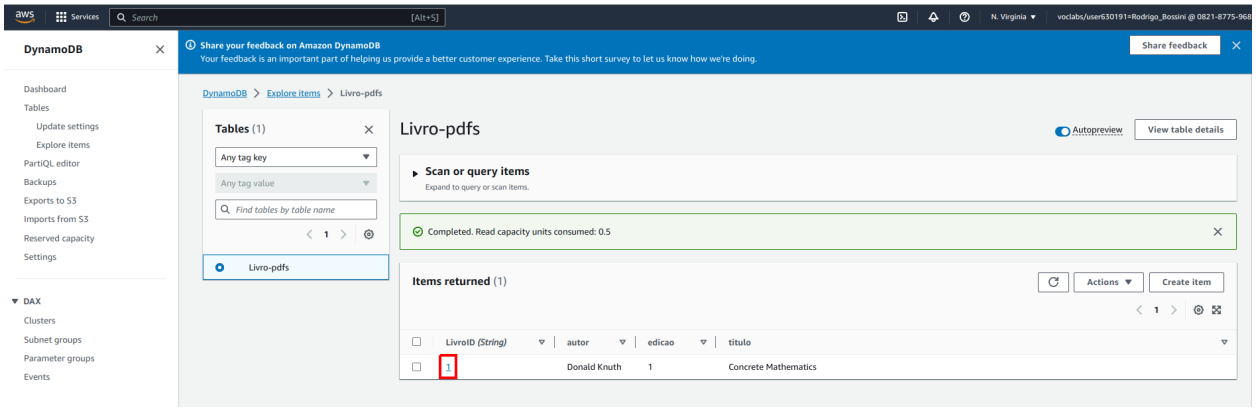


Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

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**.

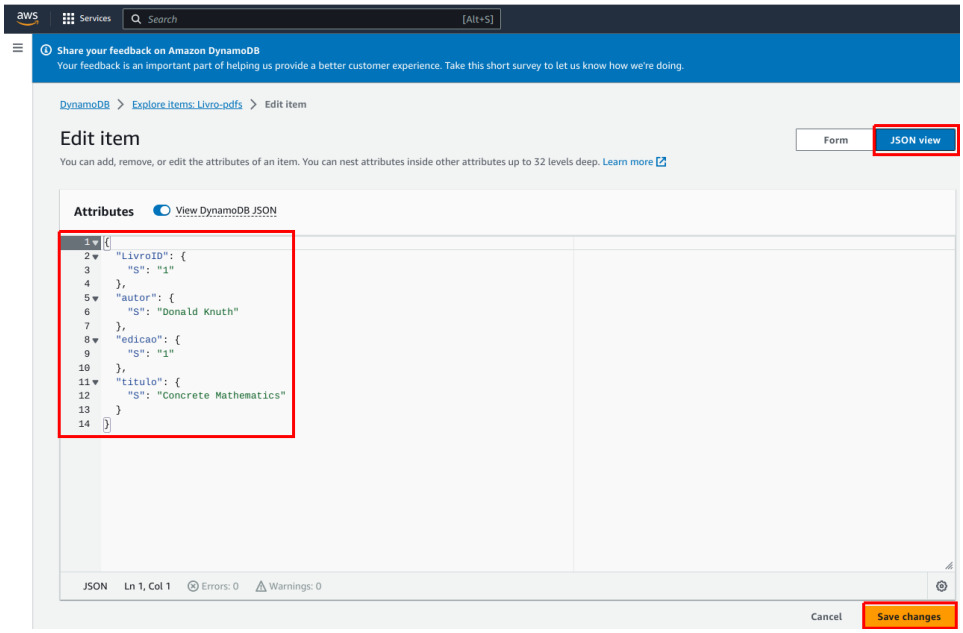


Clique sobre a chave primária do item para visualizá-lo.



Observe que ele pode ser visualizado como um objeto JSON.

Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023



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

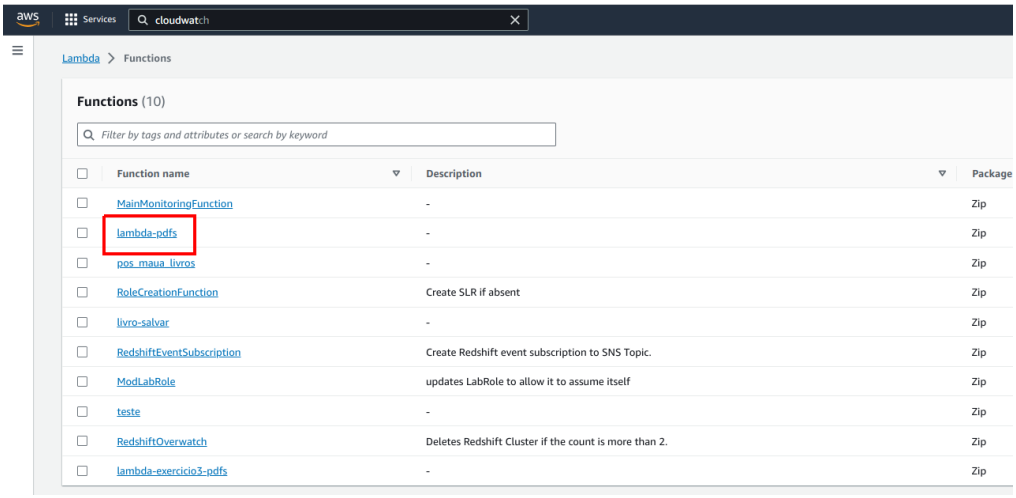
Veja mais em

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

Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

(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/

Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

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. Lembra-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;
};
```

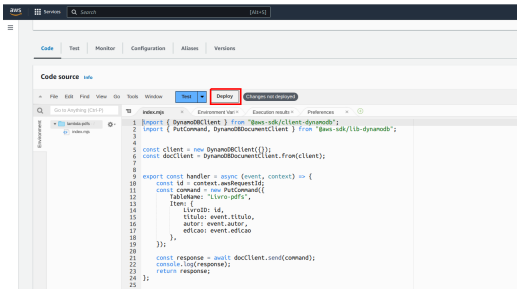
A seguir, construímos o objeto **PutCommand** especificando o nome da tabela e os dados do item (o livro) a ser inserido. Observe como utilizamos o id da requisição e também os dados do objeto **event**.

```
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.

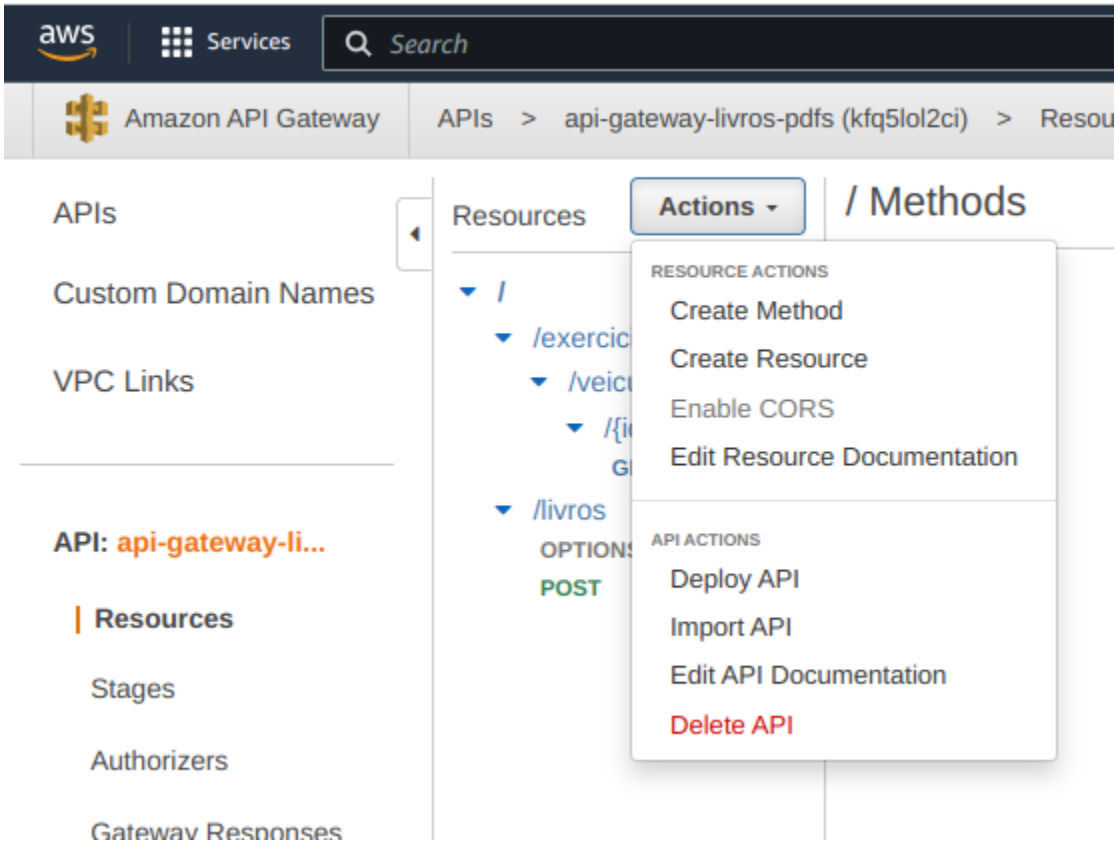
```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import { PutCommand, DynamoDBDocumentClient } from
"@aws-sdk/lib-dynamodb";

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**.

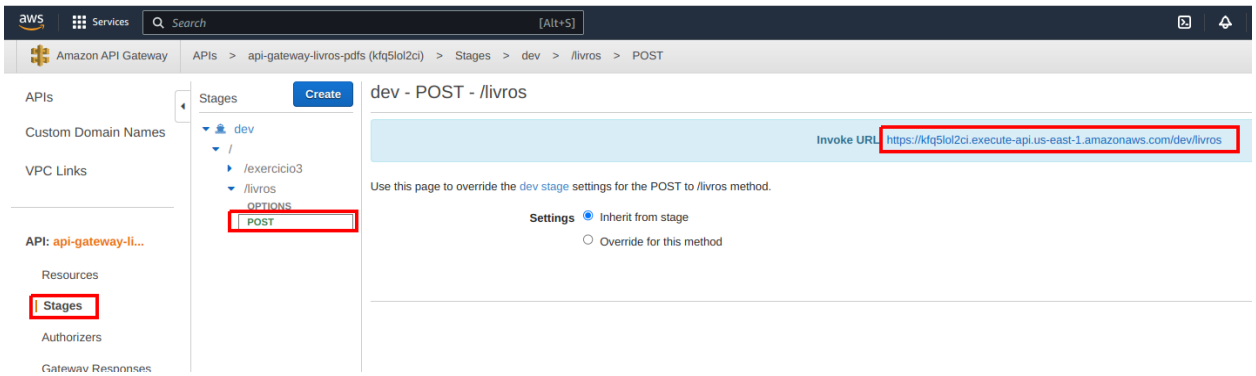


Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

(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.

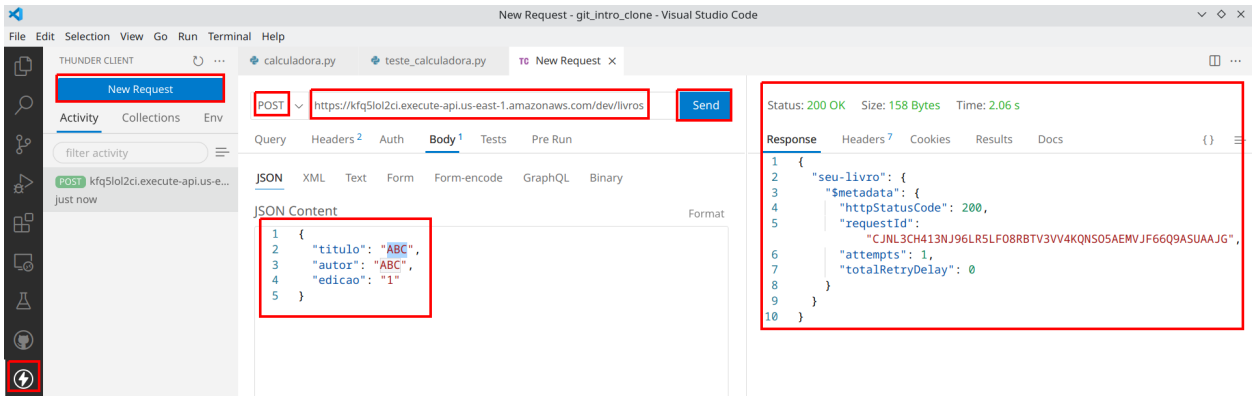


Depois disso, pegue o link de seu stage no API Gateway.



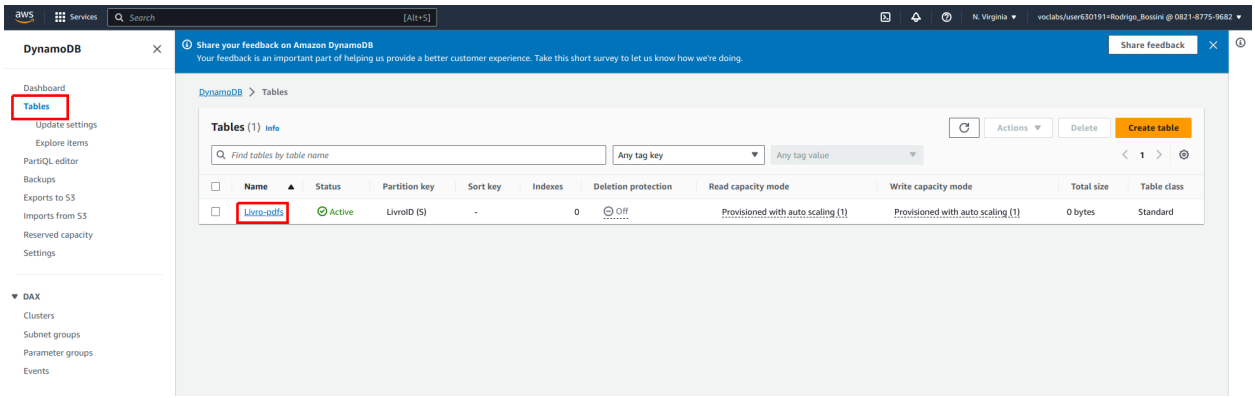
Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

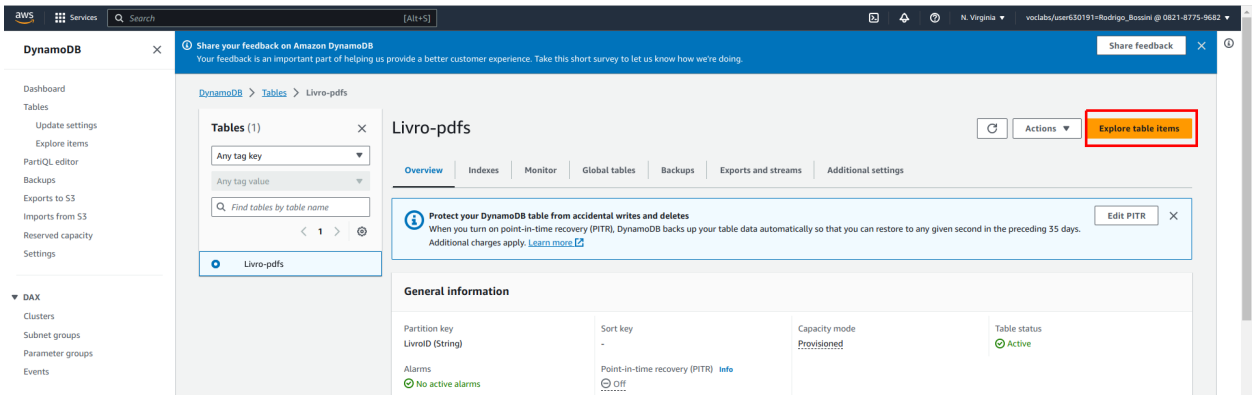
Faça uma nova requisição na Thunder Client.



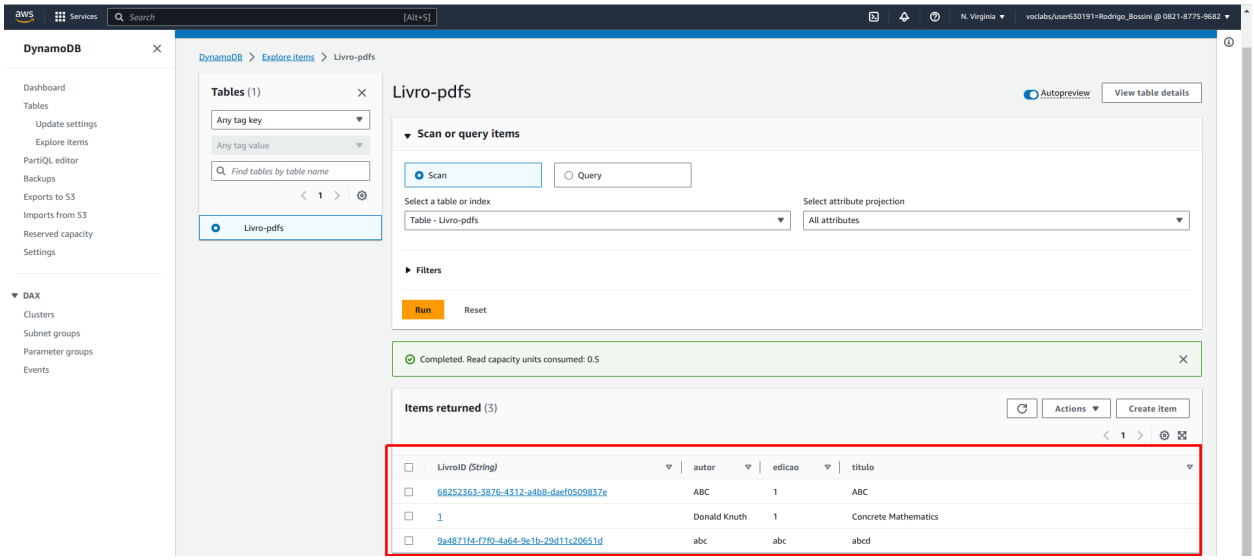
Observe que o resultado inclui detalhes técnicos da requisição enviada ao DynamoDB. Isso está acontecendo pois a função Lambda realiza a requisição ao DynamoDB e, quando recebe a resposta, apenas a devolve ao API Gateway.

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





Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023



Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: setembro de 2023

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.

Autoria: Professor Rodrigo Bossini <professorbossini@gmail.com>

Produção: setembro de 2023