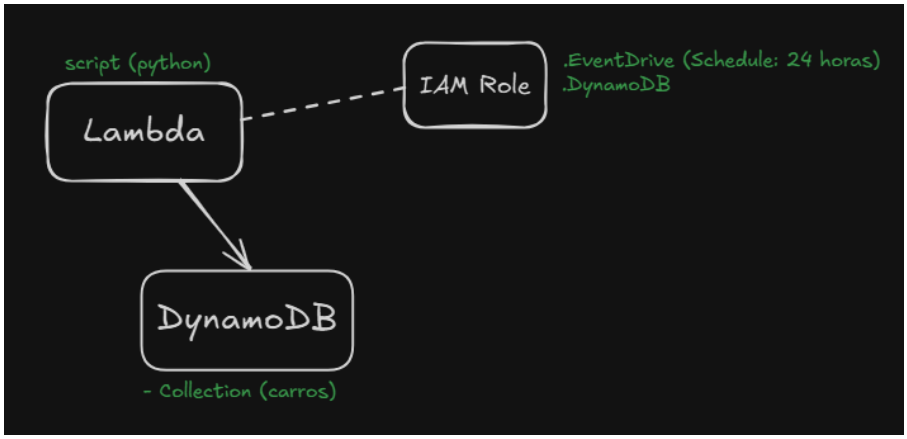
DynamoDB Backup

Apresentação Projeto

Nesse projeto, iremos automatizar o **backup** de um banco de dados **NSql**, que nesse caso será o **DynamoDB**!!

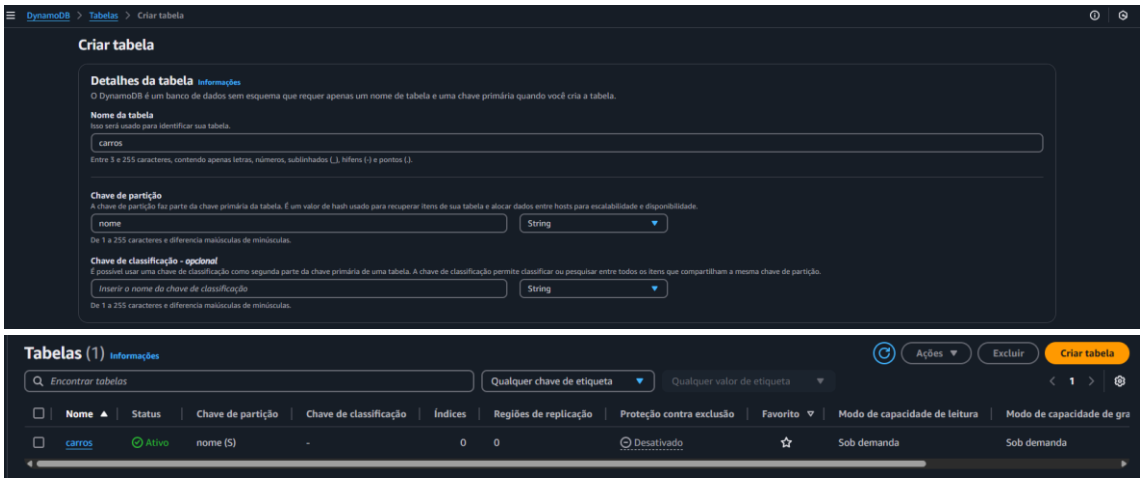
Pois caso der algum problema, ou até mesmo algum ataque no **banco de dados**, precisamos de **backups** para termos a segurança de restaurar o nosso **banco de dados** através desse **backup**!!

Time Line Projeto



Criação banco de dados (DynamoDB)

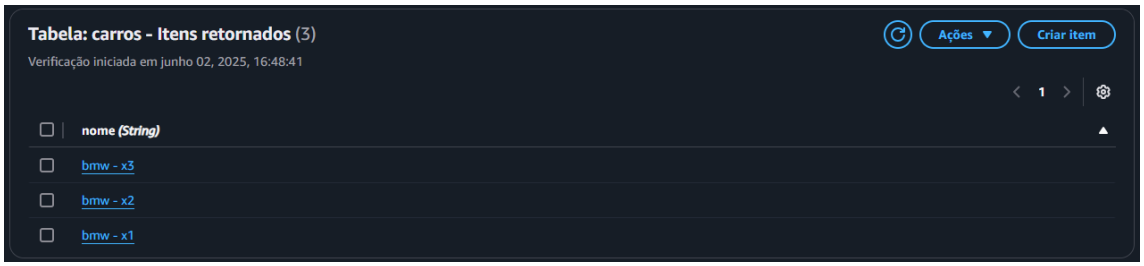
Primeiramente precisamos criar o nosso banco de dados **NSQL**, que seria o **dynamoDB**!! Dito isso, vamos criar uma tabela no **DynamoDB** chamado **carros**, onde a chave primaria será o campo **nome**.



Obs: Vale destacar que o DynamoDB mesmo que seja NoSql, é diferente do Mongodb por exemplo! Pois o mongodb usa estrutura de documentos "sem chave primaria", já o DynamoDB usa estrutura de tabelas que não relacionais, porém possuem chaves primarias!

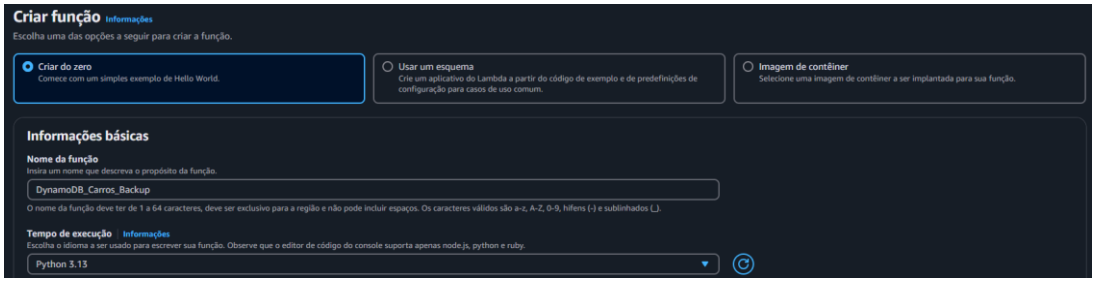
Inserindo dados no DynamoDB

Após a criação da tabela, vamos inserir alguns dados para podermos prosseguir nos testes!!



Criando função lambda

Após a criação da tabela do **DynamoDB**, vamos criar a nossa **função lambda**, onde teremos o **script** para automatizar o **backup do DynamoDB**!!

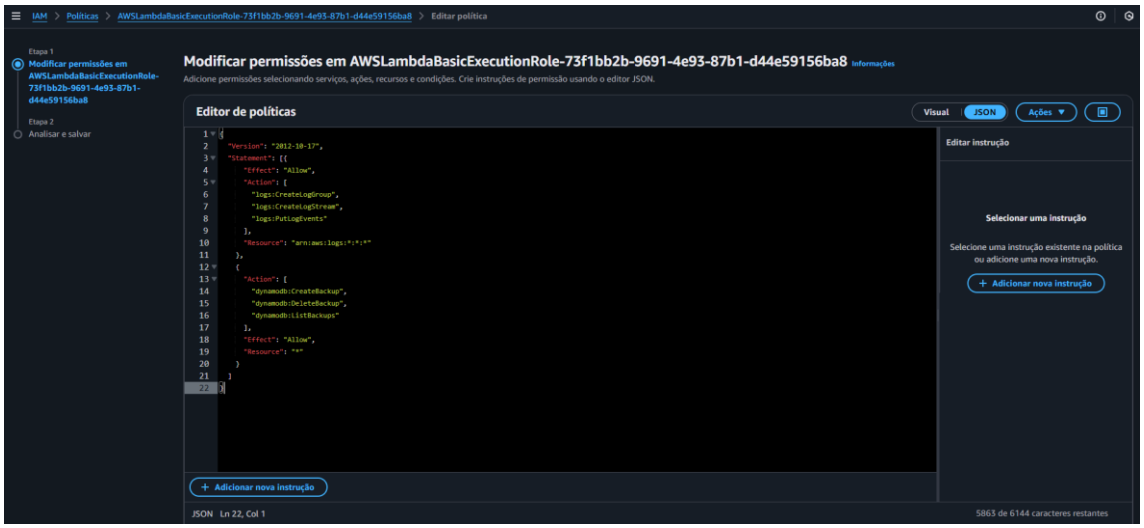


Permissão função lambda

Agora precisamos ajustar as regras de permissões do **IAM** de nossa **função lambda**!!

Pois como já sabemos por padrão a única permissão que as **funções lambda** possui é referente ao serviço do **cloudwatch**, logo teremos que adicionar as permissões referente ao serviço do **DynamoDB** referente aos **backups**!!

Como podemos ver a seguir:



Script (Python) - Funcao lambda (DynamoDB_Carros_Backup)

Vamos agora criar o nosso **script python** referente a nossa função lambda, que será responsável por realizar o **backup** de nossa tabela **carros**!

E logo de início vamos importar as seguintes bibliotecas:

- boto3: library para integrar de forma correta com os serviços da aws.
- datetime: library para de data, pois queremos nomear os backups de acordo com a data e hora naquele momento.

```
lambda_function.py
1   import boto3
2   from datetime import datetime
3
4   def lambda_handler(event, context):
5       pass
```

Em seguida vamos já declarar o nosso objeto referente ao **dynamo_client**, onde posteriormente iremos integrar com o nosso serviço do **DynamoDB**!!

```
1   import boto3
2   from datetime import datetime
3
4   dynamo = boto3.client('dynamodb')
5
6   def lambda_handler(event, context):
7       pass
```

Logo no início da implementação da função principal **lambda_handler(event, context)**, iremos validar se no evento que foi invocado nossa função lambda, temos um atributo com o nome **"table_name"**. Pois queremos **parametrizar o backup** em si correspondente com a tabela correta informada, durante a **invocação do evento**!!

```
def lambda_handler(event, context):
    if 'TableName' not in event:
        raise Exception('No table name specified.')
```

Dito isso, iremos chamar a função referente a criação do **backup** em si, **create_backup()**, que será implementada ainda!

```
def lambda_handler(event, context):
    if 'TableName' not in event:
        raise Exception('No table name specified.')

    create_backup(event['TableName'])
```

Agora sim iremos implementar a função **create_backup()**, e de início vamos declarar o nome do nosso backup em si!

```
def create_backup(table_name):
    print("Backing up table:", table_name)
    backup_name = table_name + '-' + datetime.now().strftime('%Y%m%d%H%M%S')
```

Veja que o nome do backup que iremos gerar, será formado pelo **nome da tabela** mais a **data atual**!

Após a declarar a variável do nome do backup, vamos realizar o **backup do DynamoDB em si**, e isso será feito pelo método **create_backup()** do objeto **dynamo_client**.

```
def create_backup(table_name):
    print("Backing up table:", table_name)
    backup_name = table_name + '-' + datetime.now().strftime('%Y%m%d%H%M%S')

    response = dynamo.create_backup(TableName=table_name, BackupName=backup_name)
    print(f"Created backup {response['BackupDetails']['BackupName']}")
```

E pronto, segue a versão final de nosso script:

lambda_function.py

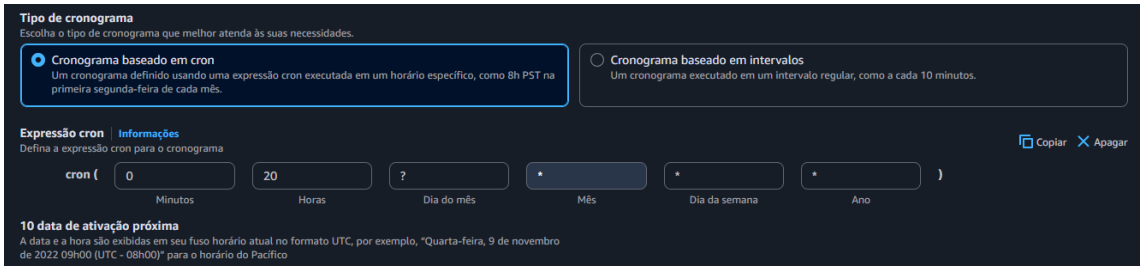
```
import boto3
from datetime import datetime

dynamo = boto3.client('dynamodb')


def lambda_handler(event, context):
    if 'TableName' not in event:
        raise Exception('No table name specified.')

    create_backup(event['TableName'])


def create_backup(table_name):
    print("Backing up table:", table_name)
    backup_name = table_name + '-' + datetime.now().strftime('%Y%m%d%H%M%S')

    response = dynamo.create_backup(TableName=table_name, BackupName=backup_name)
    print(f"Created backup {response['BackupDetails']['BackupName']}")
```

Configurando schedule pelo Amazon Event Bridge

E para finalizar, vamos agora configurar o **schedule(agendamento)** da execucao da nossa **funcao lambda**!! E para isso precisamos ir no servico do **EventBridge**!!

E para a criação de nossas regras de agendamento, vamos utilizar **expressao cron**!! E como foi citado na **time-line do projeto**, a execução de nossa **função lambda** será feita todos os dias as **20:00:00**!!

Dito isso, segue a configuração da nova regra de nosso schedule:



Vale destacar que precisamos adicionar **parâmetros** nessa **configuração do schedule**!!

Pois como vimos no **script** da **função lambda**, aguardaremos um atributo associado ao **evento de execução** da função referente ao **nome da tabela** que queremos realizar o **backup**!! Ficando assim, mais automatizado o processo de **backup do DynamoDB**!!

E pronto, com isso criamos o nosso schedule necessário para rodar a nossa função lambda!