

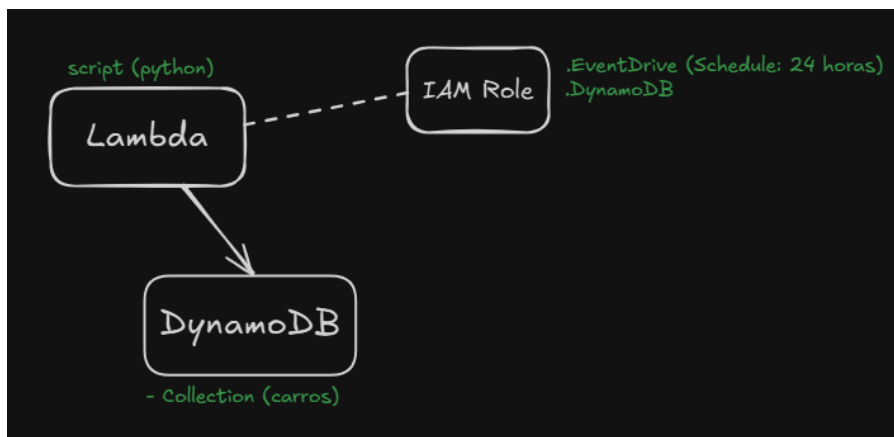
# DynamoDB Backup

## 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**.

A imagem mostra a interface de usuário do console do AWS para criar uma nova tabela no DynamoDB. O formulário está preenchido com os seguintes dados:

- Nome da tabela:** carros
- Chave de partição:** nome (tipo String)
- Chave de classificação - opcional:** Insere o nome da chave de classificação (tipo String)

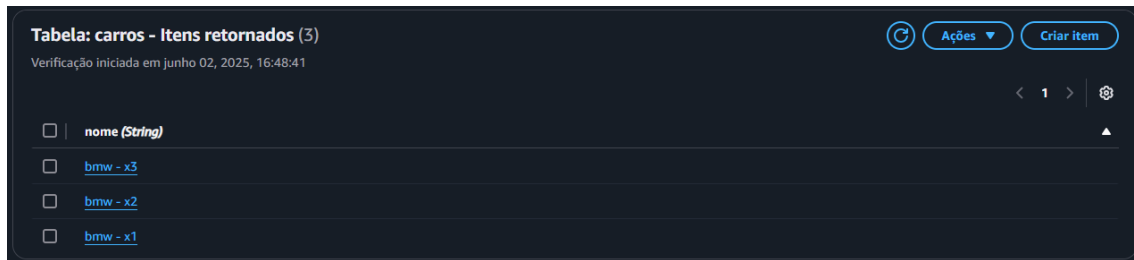
Abaixo do formulário, há uma tabela de resumo das configurações:

Nome	Status	Chave de partição	Chave de classificação	Índices	Regras de replicação	Proteção contra exclusão	Favorito	Modo de capacidade de leitura	Modo de capacidade de gr...
carros	Ativo	nome (S)	-	0	0	Desativado	☆	Sob demanda	Sob demanda

**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!**

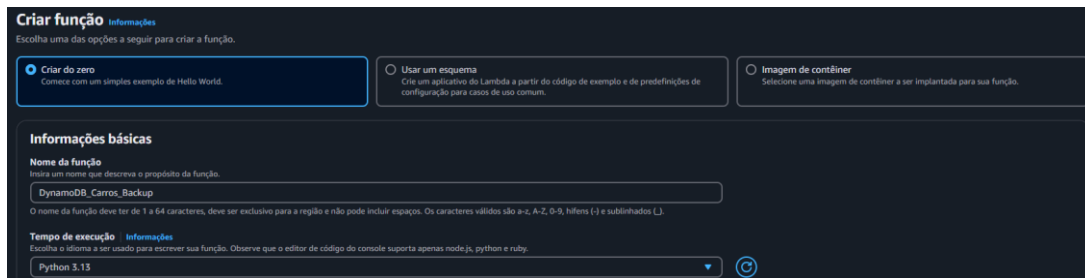
## Inserindo dados no DynamoDB

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**!!

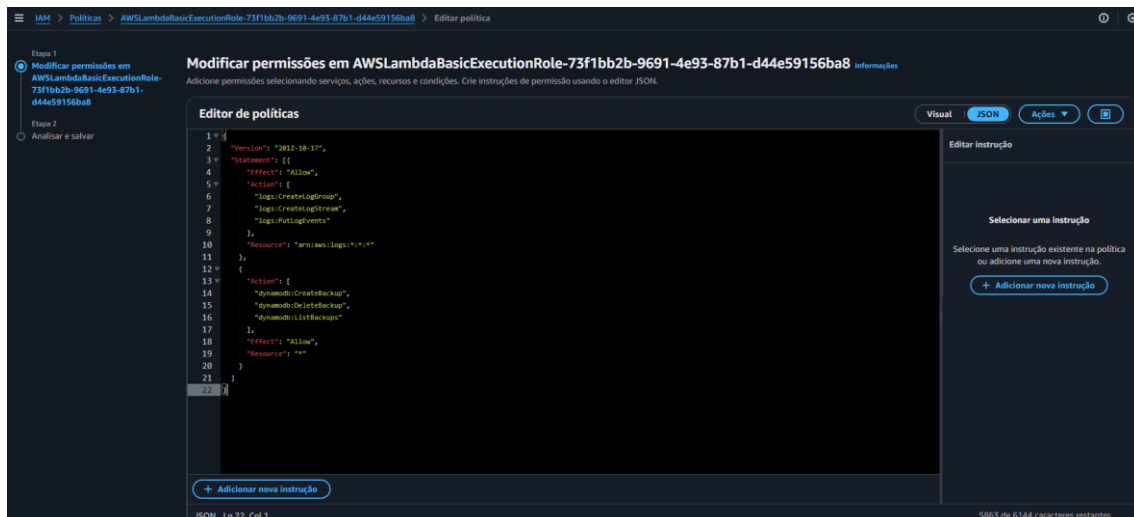


## Permissão função lambda

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  
1  import boto3  
2  from datetime import datetime  
3  
4  dynamo = boto3.client('dynamodb')  
5  
6  
7  def lambda_handler(event, context):  
8      if 'TableName' not in event:  
9          raise Exception('No table name specified.')  
10  
11      create_backup(event['TableName'])  
12  
13  
14  def create_backup(table_name):  
15      print("Backing up table:", table_name)  
16      backup_name = table_name + '-' + datetime.now().strftime('%Y%m%d%H%M%S')  
17  
18      response = dynamo.create_backup(TableName=table_name, BackupName=backup_name)  
19      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:

**Tipo de cronograma**  
Escolha o tipo de cronograma que melhor atenda às suas necessidades.

☒ Cronograma baseado em cron  
Um cronograma definido usando uma expressão cron executada em um horário específico, como 8h PST na primeira segunda-feira de cada mês.

☐ Cronograma baseado em intervalos  
Um cronograma executado em um intervalo regular, como a cada 10 minutos.

**Expressão cron** **Informações**  
Defina a expressão cron para o cronograma

cron ( 0 20 ? \* \* )

Minutos Horas Dia do mês Mês Dia da semana Ano

**10 data de ativação próxima**  
A data e a hora são exibidas em seu fuso horário atual no formato UTC, por exemplo, "Quarta-feira, 9 de novembro de 2022 09H00 (UTC - 08H00)" para o horário do Pacífico

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!!**

**Invoke** Definição de alvo universal

AWS Lambda

**Função do Lambda**  
DynamoDB\_Carros\_Backup

**Configurar versão/alias**

**Carga útil**  
O JSON que você deseja fornecer à sua função do Lambda como entrada. Por exemplo, `--payload "{"key": "value"}"` Saiba mais

```
{
  "TableName": "carros"
}
```

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