MIT Big Data

Título: **Amazon DynamoDB**

Autores: **Cesar Mansur**
Günther Becker
Leonardo Arariba
Romario Gomes

Coordenador: **Eduardo Morelli**

Rio de Janeiro, 17 de setembro de 2020

Sumário

1 Introdução

1.1 Bancos de dados de documentos

Os bancos de documentos, também chamados de sistemas de banco de dados orientados a documentos, são caracterizados por sua organização de dados sem esquemas. Os registros não precisam ter uma estrutura uniforme, ou seja, registros diferentes podem ter tipos de colunas diferentes. As colunas podem ter mais de um valor (matrizes), e os registros podem ter uma estrutura aninhada.

Os armazenamentos de documentos costumam usar notações internas, que podem ser processadas diretamente em aplicativos, principalmente JSON. Bancos de dados de documentos tornam mais fácil para os desenvolvedores armazenar e consultar dados em um banco de dados usando o mesmo formato de modelo de documento que eles usam em seu código de aplicativo. A natureza flexível, semiestruturada e hierárquica de documentos e bancos de dados de documentos permite que eles evoluam com as necessidades dos aplicativos.

Os documentos JSON, é claro, também podem ser armazenados como texto puro em armazenamentos de chave-valor ou sistemas de banco de dados relacionais, o que exige uma carga de processamento para converter os documentos para o modelo estruturado.

Exemplos mais populares:
MongoDB
Amazon DynamoDB
Microsoft Azure Cosmos DB
Couchbase
CouchDB

1.2 Bancos de dados de chave-valor

Os bancos de dados de valor-chave não estabelecem um schema específico. Os bancos de dados relacionais tradicionais pré-definem suas estruturas dentro do banco de dados, usando tabelas que contêm campos com tipos de dados bem definidos. Os sistemas de valor-chave, por outro lado, tratam os dados como uma única coleção com a chave representando uma string arbitrária - por exemplo, um nome de arquivo, hash ou identificador de recurso uniforme (URI - Uniform Resource Identifier).

Esses simples sistemas normalmente não são adequados para aplicativos complexos. Por outro lado, é exatamente essa simplicidade que torna esses sistemas atraentes em certas circunstâncias. Os armazenamentos de valores-chave geralmente usam muito menos memória enquanto salvam e armazenam a mesma quantidade de dados, aumentando o desempenho para certos tipos de cargas de trabalho. Por exemplo, em sistemas incorporados ou como banco de dados interno de alguns processos de alta performance.

Diferentes bancos de dados de valor-chave usam técnicas diferentes para melhorar o modelo básico de valor-chave. Alguns armazenam todos os seus dados na RAM, enquanto outros trabalham com uma combinação de SSDs e RAM. Outros ainda combinam suporte para discos rotativos e RAM.

Um banco de dados de valor-chave é fácil de construir e escalar. Normalmente oferece excelente desempenho e pode ser otimizado para atender às necessidades de uma organização.

Exemplos mais populares:
Redis
Amazon DynamoDB
Microsoft Azure Cosmos DB
Memcached
Hazelcast

1.3 Amazon DynamoDB

O Amazon DynamoDB é um banco de dados de documentos e chave-valor com suporte a tabelas de praticamente qualquer tamanho com dimensionamento horizontal, alto desempenho e escalabilidade automática, totalmente gerenciado, com segurança, backup e restauração integrados e armazenamento em cache na memória para aplicativos em escala. **O DynamoDB pode processar mais de 10 trilhões de solicitações por dia e comportar picos de mais de 20 milhões de solicitações por segundo.**

Muitas das empresas que mais crescem no mundo, como Lyft, Airbnb e Redfin, bem como Samsung, Toyota, e Capital One, dependem da escala e do desempenho do DynamoDB para comportar suas cargas de trabalho de missão crítica.

Centenas de milhares usuários utilizam o DynamoDB como banco de dados de documentos e chave-valor para aplicativos móveis, web, jogos, tecnologia de anúncios, IoT e de várias outras áreas que precisam de acesso a dados com baixa latência em qualquer escala. Basta criar uma tabela para o aplicativo e deixar que o DynamoDB se encarregue do resto.

Benefícios

O DynamoDB oferece suporte a alguns dos maiores aplicativos em escala do mundo ao fornecer tempos de resposta consistentes abaixo de 10 milissegundos, em qualquer escala. Você pode criar aplicativos com taxa de transferência e armazenamento praticamente ilimitados. As tabelas globais do DynamoDB replicam seus dados em várias regiões da AWS para oferecer a você acesso rápido e local a dados para seus aplicativos distribuídos globalmente. Para casos de uso que exigem acesso ainda mais rápido com latência de microssegundos, o DynamoDB Accelerator (DAX) oferece um cache de memória totalmente gerenciado.

Não há servidores para gerenciar

O DynamoDB é serverless, não há servidores para provisionar, aplicar patches ou gerenciar nem softwares para instalar, manter ou operar. O DynamoDB expande e reduz tabelas automaticamente para ajustar de acordo com a capacidade e manter o desempenho. A disponibilidade e a tolerância a falhas são incorporadas, eliminando a necessidade de projetar esses recursos em seus aplicativos. O DynamoDB oferece modos de capacidade provisionada e sob demanda para que você possa otimizar custos especificando a capacidade por carga de trabalho ou pagando somente pelos recursos que consumir.

Pronto para uso empresarial

O DynamoDB oferece suporte a transações ACID para permitir que você crie aplicativos de missão crítica em grande escala. O DynamoDB criptografa todos os dados por padrão e oferece controle refinado de acesso e identidade em todas as suas tabelas. Você pode criar backups completos de centenas de terabytes de dados instantaneamente, sem impacto no desempenho de suas tabelas, e recuperar qualquer momento dos 35 dias anteriores sem tempo de inatividade. O DynamoDB também tem o apoio de um acordo de nível de serviço para disponibilidade garantida.

Precificação

O DynamoDB cobra pela leitura, gravação e armazenamento de dados em suas tabelas do DynamoDB, juntamente com quaisquer recursos adicionais que você opte por habilitar. O DynamoDB tem dois modos de capacidade e eles vêm com duas opções específicas de faturamento para o processamento de leituras e gravações em suas tabelas: sob demanda e provisionada.

Com o modo de capacidade sob demanda, você paga pelas leituras e gravações de dados efetuadas pelo aplicativo nas tabelas. Você não precisa especificar a quantidade de throughput de leitura e gravação que espera que o aplicativo execute, pois o DynamoDB acomoda instantaneamente o aumento e a redução das cargas de trabalho.

Com o modo de capacidade provisionada, você especifica o número de leituras e gravações de dados por segundo exigidas pelo aplicativo. Esse modo se dividem de maneira diferente porque você está pagando pela capacidade, quer ela seja usada ou não. Você também pode usar o Auto Scaling para ajustar automaticamente a capacidade da tabela de acordo com a taxa de utilização especificada para garantir a performance do aplicativo e reduzir os custos.

Também é possível reservar uma capacidade gerando uma economia considerável em relação ao preço padrão da capacidade provisionada do DynamoDB. Você paga uma única taxa adiantada e se compromete a pagar a taxa por hora por blocos de 100 WCU e/ou 100 RCU durante o período de 1 ano da capacidade reservada. Qualquer capacidade que você exceda sua capacidade reservada é cobrada de acordo com as taxas de capacidade provisionada padrão.

A AWS oferece o Free Tier do DynamoDB sem prazo de validade, com os seguintes benefícios:

- 25 WCUs e 25 RCUs de capacidade provisionada
- 25 GB de armazenamento físico de dados
- 25 rWCUs para tabelas globais implantadas em duas regiões da AWS
- 2,5 milhões de solicitações de leitura do Streams do DynamoDB
- 1 GB de transferência de dados para fora (15GB nos primeiros 12 meses), agregados em todos os serviços da AWS

A tabela abaixo informa os valores das taxas cobradas pelo uso do Dynamodb nas modalidades On-Demand e Provisionado acima do free tier na região US-EAST-1.

	On-Demand	Provisionado
Write	$1.25 por milhão de requisições	$0.00065 per WCU
Read	$0.25 por milhão de requisições	$0.00013 per RCU
Data Storage	$0.25 por GB-mês	$0.25 por GB-mês
Continuous Backup (PITR)	$0.20 por GB-mês	$0.20 por GB-mês
On-Demand Backup	$0.10 por GB	$0.10 por GB
Restoring Table	$0.15 por GB	$0.15 por GB
Global Tables	$1.875 por milhões de escritas replicadas	$0.000975 por rWCU por hora
Dynamodb Accelerator (DAX)	$0.04 por hora - $12.23 por hora	$0.04 por hora - $12.23 por hora
Dynamodb Streams	$0.02 per 100,000 streams	$0.02 per 100,000 streams
Data Transfer	$0.09 por GB (até 10 TB)	$0.09 por GB (até 10 TB)

Na tabela abaixo podemos observar a comparação dos custos do Dynamodb provisionado com e sem Capacidade Reservada (RC - Reserved Capacity).

	Provisionado com RC			**Provisionado sem RC**	**Economia Total**
Comprometimento Mensal	**Upfront: 1 ano**	**Hourly: 1 ano**	**Total Anual**	**Total Anual**	
100 WCU	$150.00	$0.0128	$262.13	$569.40	46.0%
100 RCU	$30.00	$0.0025	$51,90	$113.88	45.6%

2 Posicionamento no Mercado

A DB-Engines é uma iniciativa que ranqueia os sistemas de gerenciamento de banco de dados por popularidade, cobrindo mais de 350 sistemas. Os critérios de classificação [1] incluem o número de resultados do motor de busca ao pesquisar os nomes dos sistemas, Google Trends, discussões Stack Overflow, ofertas de emprego com menções aos sistemas, número de perfis em redes profissionais como LinkedIn, menções em redes sociais como Twitter. O ranking é atualizado mensalmente.

As propriedades mais importantes são mostradas na visão geral dos sistemas de gerenciamento de banco de dados. É possível examinar as propriedades de cada sistema e compará-las lado a lado. O Amazon DynamoDB ocupa atualmente a décima sexta posição no ranking global do DB-Engines e ocupa a segunda posição nas categorias documento e chave-valor.

2.1 Visão geral do Amazon DynamoDB

Nome	Amazon DynamoDB
Modelo de Banco de Dados	Documento, Chave-Valor
Desenvolvedor	Amazon
Release Inicial	2012
Tipo de Licença	Comercial
Somente na Nuvem	Sim
Sistema Operacional do Servidor	Hosted
Índice Secundário	Sim
SQL	Não
APIs e Outros Métodos de Acesso	RESTful HTTP API
Linguagens de Programação Suportadas	.Net, ColdFusion, Erlang, Groovy, Java, JavaScript, Perl, PHP, Python, Ruby
Server-side scripts (Stored Procedures)	Não
Triggers	Sim
Métodos de Partição	Sharding
Métodos de Replicação	Sim
Conceitos de Consistência	Consistência Eventual, Consistência Imediata
Chave Estrangeira	Não
Conceitos de Transição	ACID
Concorrência	Sim
Controle de Acesso	Regras de usuário pelo AWS IAM

Fonte: https://db-engines.com/en/system/Amazon+DynamoDB

2.2 Ranking DB-Engines para banco de dados de documentos

Rank			DBMS	Database Model	Score		
Sep 2020	Aug 2020	Sep 2019			Sep 2020	Aug 2020	Sep 2019
1.	1.	1.	MongoDB	Document, Multi-model	446.48	+2.92	+36.42
2.	2.	2.	Amazon DynamoDB	Multi-model	66.18	+1.43	+8.36
3.	3.	↑ 4.	Microsoft Azure Cosmos DB	Multi-model	31.67	+0.94	+0.80
4.	4.	↓ 3.	Couchbase	Document, Multi-model	30.60	+1.54	-0.69
5.	5.	5.	CouchDB	Document	17.25	-0.14	-1.15
6.	6.	↑ 7.	Firebase Realtime Database	Document	15.61	+0.30	+4.36
7.	7.	↓ 6.	MarkLogic	Multi-model	11.94	-0.28	-1.48
8.	8.	8.	Realm	Document	8.75	+0.14	+1.01
9.	9.	↑ 10.	Google Cloud Firestore	Document	8.05	+0.23	+3.11
10.	10.	↑ 12.	ArangoDB	Multi-model	5.80	+0.06	+1.44

Fonte: https://db-engines.com/en/ranking/document+store

2.3 Ranking DB-Engines para banco de dados de chave-valor

Rank			DBMS	Database Model	Score		
Sep 2020	Aug 2020	Sep 2019			Sep 2020	Aug 2020	Sep 2019
1.	1.	1.	Redis	Key-value, Multi-model	151.86	-1.02	+9.95
2.	2.	2.	Amazon DynamoDB	Multi-model	66.18	+1.43	+8.36
3.	3.	3.	Microsoft Azure Cosmos DB	Multi-model	31.67	+0.94	+0.80
4.	4.	4.	Memcached	Key-value	26.34	+0.38	-0.12
5.	5.	5.	Hazelcast	Key-value, Multi-model	9.47	+0.18	+1.93
6.	6.		etcd	Key-value	8.85	+0.27	
7.	7.	↓ 6.	Ehcache	Key-value	7.32	+0.24	+1.25
8.	8.	8.	Aerospike	Key-value, Multi-model	6.70	-0.13	+1.04
9.	↑ 10.	↓ 7.	Riak KV	Key-value	5.92	+0.43	+0.24
10.	↓ 9.	10.	ArangoDB	Multi-model	5.80	+0.06	+1.44

Fonte: https://db-engines.com/en/ranking/key-value+store

2.4 DynamoDB versus MongoDB

MongoDB é um banco de dados NoSQL open-source que fornece suporte para sistemas de armazenamento orientados a documentos no estilo JSON. Segundo a db-engines, é líder de mercado na categoria NoSQL. Ele oferece suporte a um modelo de dados flexível que permite armazenar dados de qualquer estrutura e fornece um rico conjunto de recursos, incluindo suporte a índice, fragmentação e replicação. Está disponível nas principais clouds do mercado, incluindo AWS.

Vantagens de Cada Engine	
DynamoDB	**MongoDB**
Não há servidores para gerenciar	Implantação em qualquer ambiente
Performance em grande escala	Baseado em documentos JSON (até 16MB)
Escalabilidade automática	Linguagem de consultas mais completa
Fácil integração com outros serviços da AWS	Index mais robusto
Sistema de armazenamento simples	Mais fácil de desenvolver
Segurança	Mais consistente
Baixíssima Latência de Leitura (10ms - Cache)	Mais transparente
Geo-Distribuição	Precificação mais simples

2.5 DynamoDB versus Redis (visão DB-Engines):

Redis é um rápido armazenador de estrutura de dados em memória de código aberto, para uso como banco de dados, cache, agente de mensagens e fila. Segundo a db-engines, é líder de mercado no modelo chave-valor. Ele suporta estruturas de dados, como strings, hashes, listas, conjuntos, conjuntos classificados com consultas de intervalo, bitmaps, hiperloglog, índices geoespaciais com consultas de raio e fluxos. Na AWS está disponível no serviço AWS ElastiCache.

Vantagens de Cada Engine	
DynamoDB	**Redis**
Não há servidores para gerenciar	Fácil configuração
Performance em grande escala	Baixo custo
Pode armazenar dados estruturados e não estruturados	Ótimo para cache de consultas e respostas caras
Baseado em documentos JSON (até 400kb)	

Fonte: https://www.trustradius.com/compare-products/amazon-dynamodb-vs-amazon-elasticache

3 Tutorial DynamoDB

3.1 Conceitos chave

3.1.1 Tabelas, itens e atributos

Tabelas, itens e atributos são os principais blocos de construção do DynamoDB.

Uma tabela é um agrupamento de registros de dados. Por exemplo, você pode ter uma tabela usuários para armazenar dados sobre seus usuários e uma tabela pedidos para armazenar dados sobre os pedidos de seus usuários. Este conceito é semelhante a uma tabela em um banco de dados relacional ou uma coleção no MongoDB.

Um item é um único registro de dados em uma tabela. Cada item em uma tabela é identificado exclusivamente pela chave primária declarada da tabela. Em sua tabela de usuários, um item seria um determinado usuário. Um item é semelhante a uma linha em um banco de dados relacional ou um documento no MongoDB.

Atributos são pedaços de dados anexados a um único item. Este poderia ser um atributo simples de idade que armazena a idade de um usuário. Um atributo é comparável a uma coluna em um banco de dados relacional ou a um campo no MongoDB. O DynamoDB não requer atributos em itens, exceto para atributos que constituem sua chave primária.

3.1.2 Chave primária

Cada item em uma tabela é identificado exclusivamente por uma chave primária. A definição da chave primária deve ser definida na criação da tabela, e a chave primária deve ser fornecida ao inserir um novo item.

Existem dois tipos de chave primária: uma chave primária simples composta apenas por uma chave de partição e uma chave primária composta, que é formada por uma chave de partição e uma chave de classificação.

Usar uma chave primária simples é semelhante a armazenamentos de chave-valor padrão como Memcached ou acessar linhas em uma tabela SQL por uma chave primária. Um exemplo seria uma tabela de usuários com uma chave primária de nome de usuário.

A chave primária composta é mais complexa. Com uma chave primária composta, você especifica uma chave de partição e uma chave de classificação. A chave de classificação é usada para classificar itens com a mesma partição. Um exemplo poderia ser uma tabela de pedidos para registrar pedidos de clientes em um site de comércio eletrônico. A chave de partição seria CustomerId e a chave de classificação seria OrderId.

Cada item em uma tabela é identificado exclusivamente por uma chave primária, mesmo com a chave composta. Ao usar uma tabela com uma chave primária composta, você pode ter vários itens com a mesma chave de partição, mas chaves de classificação diferentes. Você só pode ter um item com uma combinação específica de chave de partição e chave de classificação.

A chave primária composta permite padrões de consulta sofisticados, incluindo pegar todos os itens com a chave de partição fornecida ou usar a chave de classificação para restringir os itens relevantes para uma consulta específica.

3.1.3 Índices Secundários

A chave primária identifica exclusivamente um item em uma tabela, e você pode fazer consultas na tabela usando a chave primária. No entanto, às vezes você tem padrões de acesso adicionais que seriam ineficientes com sua chave primária. O DynamoDB tem a noção de índices secundários para habilitar esses padrões de acesso adicionais.

O primeiro tipo de índice secundário é um índice secundário local. Um índice secundário local usa a mesma chave de partição que a tabela subjacente, mas uma chave de classificação diferente. Utilizando o exemplo de tabela de pedidos, imagine que você deseja acessar rapidamente os pedidos de um cliente em ordem decrescente do valor que eles gastaram no pedido. Você pode adicionar um índice secundário local com uma chave de partição CustomerId e uma chave de classificação Amount, permitindo consultas eficientes nos pedidos de um cliente por quantidade.

O segundo tipo de índice secundário é um índice secundário global. Um índice secundário global pode definir uma chave primária totalmente diferente para uma tabela. Isso pode significar definir um índice com apenas uma chave de partição para uma tabela com uma chave primária composta. Também pode significar o uso de atributos completamente diferentes para preencher uma chave de partição e uma chave de classificação. Com o exemplo da tabela de pedidos, poderíamos ter um índice secundário global com uma chave de partição OrderId para que pudéssemos recuperar um pedido específico sem saber o CustomerId que fez o pedido.

Os índices secundários são um tópico complexo, mas extremamente útil para obter o máximo do DynamoDB.

3.1.4 Capacidade de leitura e gravação

Quando você usa um banco de dados como MySQL, Postgres ou MongoDB, provisiona um servidor específico para executar seu banco de dados. Você precisará escolher o tamanho da instância - quantas CPUs você precisa, quanta RAM, quantos GBs de armazenamento, etc.

Não é assim com o DynamoDB. Em vez disso, você provisiona unidades de capacidade de leitura e gravação. Essas unidades permitem um determinado número de operações por segundo. Este é um paradigma de precificação fundamentalmente diferente do mundo baseado em instância (a precificação pode refletir mais de perto o uso real).

O DynamoDB também possui escalonamento automático de suas unidades de capacidade de leitura e gravação. Isso torna muito mais fácil dimensionar seu aplicativo durante os horários de pico, enquanto economiza dinheiro ao reduzir quando seus usuários estão dormindo. Por outro lado, oferece um risco alto vinculado ao custo, caso a demanda aumente.

3.2 Preparação do ambiente

Optamos pela utilização do Docker, pela simplicidade e praticidade.

3.2.1 Instalação do Docker

Como exemplo, apresentamos um passo a passo para instalação do Docker em um ambiente Linux Red Hat.

Executar os seguintes comandos:

```
# yum install -y yum-utils

# yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo

# yum install docker-ce docker-ce-cli containerd.io

# systemctl start docker
```

Caso seu ambiente utilize um proxy HTTP/HTTPS, execute os quatro passos abaixo:

1) Criar o diretório "/etc/systemd/system/docker.service.d":

```
# mkdir -p /etc/systemd/system/docker.service.d
```

2) Criar o arquivo "/etc/systemd/system/docker.service.d/http-proxy.conf" e adicionar as linhas abaixo:

```
[Service]
Environment="HTTP_PROXY=http://proxy_user:proxy_password@proxy_URL:PORT"
Environment="HTTPS_PROXY= http://proxy_user:proxy_password@proxy_URL:PORT"
Environment="NO_PROXY=localhost,127.0.0.1,10.*"
```

3) Persistir mudanças e reiniciar a engine Docker:

```
# systemctl daemon-reload
# systemctl restart docker
```

4) Conferir se a configuração de proxy foi carregada:

```
# systemctl show --property=Environment docker
```

Conferir se a engine Docker está instalada corretamente, executando a imagem "hello-world":

```
# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:d58e752213a51785838f9eed2b7a498ffa1cb3aa7f946dda11af39286c3db9a9
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
```

3.2.2 Instalação Docker Compose

Verificar a release atual em:
https://github.com/docker/compose/releases

Executar os comandos abaixo, alterando a release (conforme verificado acima):

```
# sudo curl -L "https://github.com/docker/compose/releases/download/1.26.0/docker-compose-
$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# sudo chmod +x /usr/local/bin/docker-compose
```

Conferir se o docker-compose está instalado corretamente, executando o comando abaixo:

```
# docker-compose --version
docker-compose version 1.26.0, build d4451659
```

3.3 Instalação do DynamoDB local

Mesmo sendo um serviço na AWS, o DynamoDB é oferecido pela Amazon em uma versão para instalação local, que você pode executar em seu computador e é perfeita para desenvolvimento e teste de seu código. A versão para download permite que você escreva e teste aplicativos localmente sem acessar o serviço da AWS. Excelente opção para aprender, praticar e desenvolver, sem custo.

3.3.1 Instalando a imagem Docker do DynamoDB

Criar o arquivo "docker-compose.yml" com o conteúdo abaixo:

```
# vi docker-compose.yml

version: '3.7'
services:
 dynamodb-local:
   image: amazon/dynamodb-local:latest
   container_name: dynamodb-local
   ports:
    - "8000:8000"
   command: ["-jar", "DynamoDBLocal.jar", "-sharedDb", "-inMemory"]
```

PS: Se necessário, altere a porta de mapeamento destacada em amarelo (local:contêiner).
Imprescindível a utilização do parâmetro "-sharedDb", para viabilizar o acesso ao mesmo database local através do "AWS CLI", "DynamoDB local shell" (Javascript SDK) e "NoSQL Workbench for Amazon DynamoDB".

Download da imagem e inicialização do DynamoDB local:

```
# docker-compose up

Creating dynamodb-local ... done
Attaching to dynamodb-local
dynamodb-local    | Initializing DynamoDB Local with the following configuration:
dynamodb-local    | Port:       8000
dynamodb-local    | InMemory:   true
dynamodb-local    | DbPath:     null
dynamodb-local    | SharedDb:   true
dynamodb-local    | shouldDelayTransientStatuses:       false
dynamodb-local    | CorsParams: *
dynamodb-local    |
```

Verificar se a imagem foi instalada e está em execução:

```
# docker ps -a | grep -i dynamodb
b4e25ce1839a        amazon/dynamodb-local:latest   "java -jar DynamoDBL…"   8 days ago
Up 8 days                 0.0.0.0:8000->8000/tcp   dynamodb-local
```

3.4 Instalação do AWS Command Line Interface (AWS CLI)

3.4.1 Utilizando a imagem Docker do AWS CLI

Utilizando a AWS CLI version 2 Docker image:

```
# docker run --rm -it amazon/aws-cli command
```

PS: Essa imagem é equivalente ao executável 'aws'. Cada vez que você executa este comando, o Docker prepara um contêiner de sua imagem amazon/aws-cli baixada e executa seu comando aws. Por padrão, a imagem Docker usa a versão mais recente do AWS CLI (versão 2).
A opção '`--rm`' especifica a limpeza do contêiner após a saída do comando.
A opção '`-it`' especifica a abertura de um pseudo-TTY com stdin. Isso permite que você forneça dados para o AWS CLI versão 2 enquanto ele está sendo executado em um contêiner, por exemplo, usando os comandos aws configure e aws help.

Verificando a versão do AWS CLI

```
# docker run --rm -it amazon/aws-cli --version
aws-cli/2.0.26 Python/3.7.3 Linux/3.10.0-862.14.4.el7.x86_64 botocore/2.0.0dev30
```

Criando um ALIAS "aws" para a execução via Docker image

```
# alias aws='docker run --rm -it amazon/aws-cli'
```

Testando o ALIAS "aws"

```
# aws --version
aws-cli/2.0.26 Python/3.7.3 Linux/3.10.0-862.14.4.el7.x86_64 botocore/2.0.0dev30
```

3.5 Configuração do AWS CLI

Para interagir com o serviço na nuvem, é necessário configurar o AWS CLI com a chave de acesso, conforme determina a documentação da Amazon:

To create access keys for an IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose **Users**.
3. Choose the name of the user whose access keys you want to create, and then choose the **Security credentials** tab.
4. In the **Access keys** section, choose **Create access key**.
5. To view the new access key pair, choose **Show**. You will not have access to the secret access key again after this dialog box closes. Your credentials will look something like this:
   - Access key ID: AKIAIOSFODNN7EXAMPLE
   - Secret access key: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location. You will not have access to the secret access key again after this dialog box closes.

   Keep the keys confidential in order to protect your AWS account and never email them. Do not share them outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.
7. After you download the `.csv` file, choose **Close**. When you create an access key, the key pair is active by default, and you can use the pair right away.

PS: Neste exemplo, como estamos utilizando uma imagem Docker do DynamoDB Local, podemos pular os 7 passos acima.

Criando um ALIAS "aws" para a execução via Docker image, permitindo salvar as configurações localmente:

```
# alias aws='docker run --rm -ti -v ~/.aws:/root/.aws -v $(pwd):/aws amazon/aws-cli'
```

Configurando o AWS CLI

```
# aws configure
AWS Access Key ID [None]: fakeAWSAccessKeyID
AWS Secret Access Key [None]: fakeAWSSecretAccessKey
Default region name [None]: us-west-2
Default output format [None]: json
```

PS: `us-west-2` é a região utilizada pelo JavaScript Shell do DynamoDB Local. O DynamoDB Local utiliza um DB file no padrão "<SecretKey>_us-west-2.db".

Conferindo a configuração salva:

```
# cat /root/.aws/config
[default]
region = us-west-2
output = json

# cat /root/.aws/credentials
[default]
aws_access_key_id = fakeAWSAccessKeyID
aws_secret_access_key = fakeAWSSecretAccessKey
```

Como estamos usando o DynamoDB Local, a partir deste ponto será necessário acrescentar ao final de todos os comandos o parâmetro "`--endpoint-url http://IP:PORTA`":

```
 --endpoint-url http://127.0.0.1:8000
```

3.6 Utilizando o DynamoDB – AWS CLI

Diferente do MongoDB, que cria a coleção (tabela) automaticamente quando o primeiro documento (registro) é inserido, no DynamoDB é necessário criar a tabela antes de inserir os registros. Se a tabela não for criada, ele retornará a seguinte mensagem de erro quando for feita uma tentativa de inserção:

```
An error occurred (ResourceNotFoundException) when calling the PutItem operation: Cannot do
operations on a non-existent table
```

3.6.1 Criação de tabelas

"create-table": Cria uma tabela.

```
# aws dynamodb create-table \
     --table-name Music \
     --attribute-definitions \
         AttributeName=Artist,AttributeType=S \
         AttributeName=SongTitle,AttributeType=S \
     --key-schema AttributeName=Artist,KeyType=HASH AttributeName=SongTitle,KeyType=RANGE \
     --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1 \
     --endpoint-url http://127.0.0.1:8000

{
    "TableDescription": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "Music",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": "2020-07-23T02:58:43.719000+00:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "1970-01-01T00:00:00+00:00",
            "LastDecreaseDateTime": "1970-01-01T00:00:00+00:00",
```

```
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 1,
            "WriteCapacityUnits": 1
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:ddblocal:000000000000:table/Music"
    }
}
```

"list-tables": Lista as tabelas existentes.

```
# aws dynamodb list-tables --endpoint-url http://127.0.0.1:8000
{
    "TableNames": [
        "Music"
    ]
}
```

"describe-table": Mostra a descrição de uma tabela.

```
# aws dynamodb describe-table --table-name Music --endpoint-url http://127.0.0.1:8000
{
    "Table": {
        "AttributeDefinitions": [
            {
                "AttributeName": "Artist",
                "AttributeType": "S"
            },
            {
                "AttributeName": "SongTitle",
                "AttributeType": "S"
            }
        ],
        "TableName": "Music",
        "KeySchema": [
            {
                "AttributeName": "Artist",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "SongTitle",
                "KeyType": "RANGE"
            }
        ],
        "TableStatus": "ACTIVE",
        "CreationDateTime": "2020-07-23T02:58:43.719000+00:00",
        "ProvisionedThroughput": {
            "LastIncreaseDateTime": "1970-01-01T00:00:00+00:00",
            "LastDecreaseDateTime": "1970-01-01T00:00:00+00:00",
            "NumberOfDecreasesToday": 0,
            "ReadCapacityUnits": 1,
            "WriteCapacityUnits": 1
        },
        "TableSizeBytes": 0,
        "ItemCount": 0,
        "TableArn": "arn:aws:dynamodb:ddblocal:000000000000:table/Music"
    }
}
```

3.6.2 Inserção de itens

"put-item": Insere um item na tabela.

```
# aws dynamodb put-item \
    --table-name Music \
    --item '{
        "Artist": {"S": "Pink Floyd"},
        "SongTitle": {"S": "Shine On You Crazy Diamond"}
      }' \
    --return-consumed-capacity TOTAL \
```

```
    --endpoint-url http://127.0.0.1:8000

{
    "ConsumedCapacity": {
        "TableName": "Music",
        "CapacityUnits": 1.0
    }
}


# aws dynamodb put-item \
    --table-name Music \
    --item '{
        "Artist": {"S": "Pink Floyd"},
        "SongTitle": {"S": "Another Brick in the Wall"}
      }' \
    --return-consumed-capacity TOTAL \
    --endpoint-url http://127.0.0.1:8000

{
    "ConsumedCapacity": {
        "TableName": "Music",
        "CapacityUnits": 1.0
    }
}


# aws dynamodb put-item \
    --table-name Music \
    --item '{
        "Artist": {"S": "Dire Straits"},
        "SongTitle": {"S": "Sultans of Swing"}
      }' \
    --return-consumed-capacity TOTAL \
    --endpoint-url http://127.0.0.1:8000


{
    "ConsumedCapacity": {
        "TableName": "Music",
        "CapacityUnits": 1.0
    }
}


# aws dynamodb put-item \
    --table-name Music \
    --item '{
        "Artist": {"S": "Led Zeppelin"},
        "SongTitle": {"S": "Stairway to Heaven"}
      }' \
    --return-consumed-capacity TOTAL \
    --endpoint-url http://127.0.0.1:8000

{
    "ConsumedCapacity": {
        "TableName": "Music",
        "CapacityUnits": 1.0
    }
}
```

PS: Note que é necessário especificar o tipo do atributo.

3.6.3 Consulta de itens

“get-item”: Faz a consulta de um item pela chave primária. Se não existir chave primária na tabela, é necessário utilizar todos os atributos como chave na consulta.

```
# aws dynamodb get-item \
    --table-name Music \
    --endpoint-url http://127.0.0.1:8000 \
    --key '{"Artist": {"S": "Pink Floyd"}, "SongTitle": {"S": "Shine On You Crazy Diamond"}}'
```

```
{
    "Item": {
        "Artist": {
            "S": "Pink Floyd"
        },
        "SongTitle": {
            "S": "Shine On You Crazy Diamond"
        }
    }
}
```

“scan”: Permite consultar todos os itens de uma tabela, ou aplicar uma expressão de filtro.

```
# aws dynamodb scan \
     --table-name Music \
     --endpoint-url http://127.0.0.1:8000

{
    "Items": [
        {
            "Artist": {
                "S": "Dire Straits"
            },
            "SongTitle": {
                "S": "Sultans of Swing"
            }
        },
        {
            "Artist": {
                "S": "Pink Floyd"
            },
            "SongTitle": {
                "S": "Another Brick in the Wall"
            }
        },
        {
            "Artist": {
                "S": "Pink Floyd"
            },
            "SongTitle": {
                "S": "Shine On You Crazy Diamond"
            }
        },
        {
            "Artist": {
                "S": "Led Zeppelin"
            },
            "SongTitle": {
                "S": "Stairway to Heaven"
            }
        }
    ],
    "Count": 4,
    "ScannedCount": 4,
    "ConsumedCapacity": null
}


# aws dynamodb scan \
     --table-name Music \
     --endpoint-url http://127.0.0.1:8000 \
     --filter-expression "Artist = :artist1 or Artist = :artist2" \
     --expression-attribute-values '{":artist1":{"S":"Pink Floyd"},":artist2":{"S":"Led
Zeppelin"}}'

{
    "Items": [
        {
            "Artist": {
                "S": "Pink Floyd"
            },
            "SongTitle": {
                "S": "Another Brick in the Wall"
            }
        },
```

```
        {
            "Artist": {
                "S": "Pink Floyd"
            },
            "SongTitle": {
                "S": "Shine On You Crazy Diamond"
            }
        },
        {
            "Artist": {
                "S": "Led Zeppelin"
            },
            "SongTitle": {
                "S": "Stairway to Heaven"
            }
        }
    ],
    "Count": 3,
    "ScannedCount": 4,
    "ConsumedCapacity": null
}
```

"query": Permite aplicar uma expressão de filtro na consulta.

```
# aws dynamodb query --table-name Music --endpoint-url http://127.0.0.1:8000 \
    --key-condition-expression "Artist = :artist and begins_with(SongTitle, :song)" \
    --expression-attribute-values '{":artist":{"S":"Pink Floyd"},":song":{"S":"Sh"}}'

{
    "Items": [
        {
            "Artist": {
                "S": "Pink Floyd"
            },
            "SongTitle": {
                "S": "Shine On You Crazy Diamond"
            }
        }
    ],
    "Count": 1,
    "ScannedCount": 1,
    "ConsumedCapacity": null
}
```

3.6.4 Atualização de itens

"update-item": Permite atualizar um item. Assim como o "get-item", se não existir chave primária na tabela, é necessário utilizar todos os atributos como chave no update.

```
# aws dynamodb update-item \
    --table-name Music \
    --endpoint-url http://127.0.0.1:8000 \
    --key '{"Artist": {"S": "Pink Floyd"},
            "SongTitle": {"S": "Shine On You Crazy Diamond"}}' \
    --update-expression "SET ReleasedYear = :year" \
    --expression-attribute-values '{":year":{"N":"1975"}}' \
    --return-values UPDATED_NEW

{
    "Attributes": {
        "ReleasedYear": {
            "N": "1975"
        }
    }
}
```

3.6.5 Deleção de itens

“delete-item”: Deleta um item.

```
# aws dynamodb delete-item \
    --table-name Music \
    --endpoint-url http://127.0.0.1:8000 \
    --key '{"Artist": {"S": "Pink Floyd"}, "SongTitle": {"S": "Another Brick in the Wall"}}'
```

3.6.6 Deleção de tabelas

“delete-table”: Deleta uma tabela.

```
# aws dynamodb create-table \
     --table-name Test \
     --attribute-definitions \
         AttributeName=Att1,AttributeType=S \
         AttributeName=Att2,AttributeType=S \
     --key-schema AttributeName=Att1,KeyType=HASH AttributeName=Att2,KeyType=RANGE \
     --provisioned-throughput ReadCapacityUnits=1,WriteCapacityUnits=1 \
     --endpoint-url http://127.0.0.1:8000

# aws dynamodb delete-table --table-name Test --endpoint-url http://127.0.0.1:8000
```

3.7 Utilizando o DynamoDB – DynamoDB Local Shell (Javascript SDK)

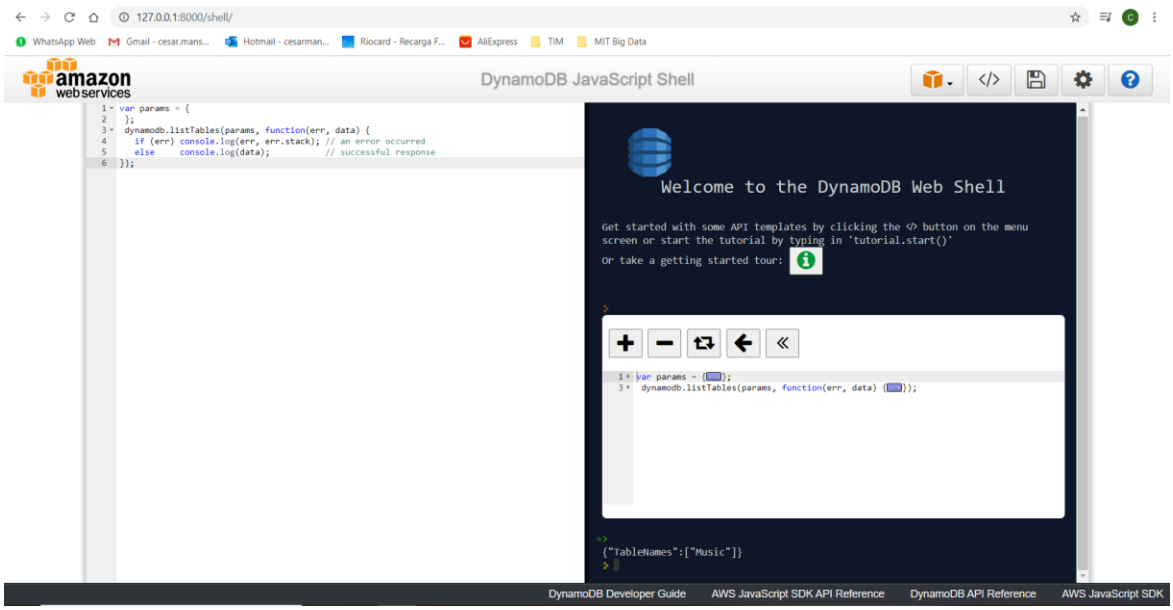
O shell do DynamoDB é uma forma interativa de experimentar e testar o serviço DynamoDB por meio do DynamoDB Local. É usado para fins de aprendizagem e teste.

Verificar acesso ao shell do DynamoDB local via browser:

http://127.0.0.1:8000/shell/

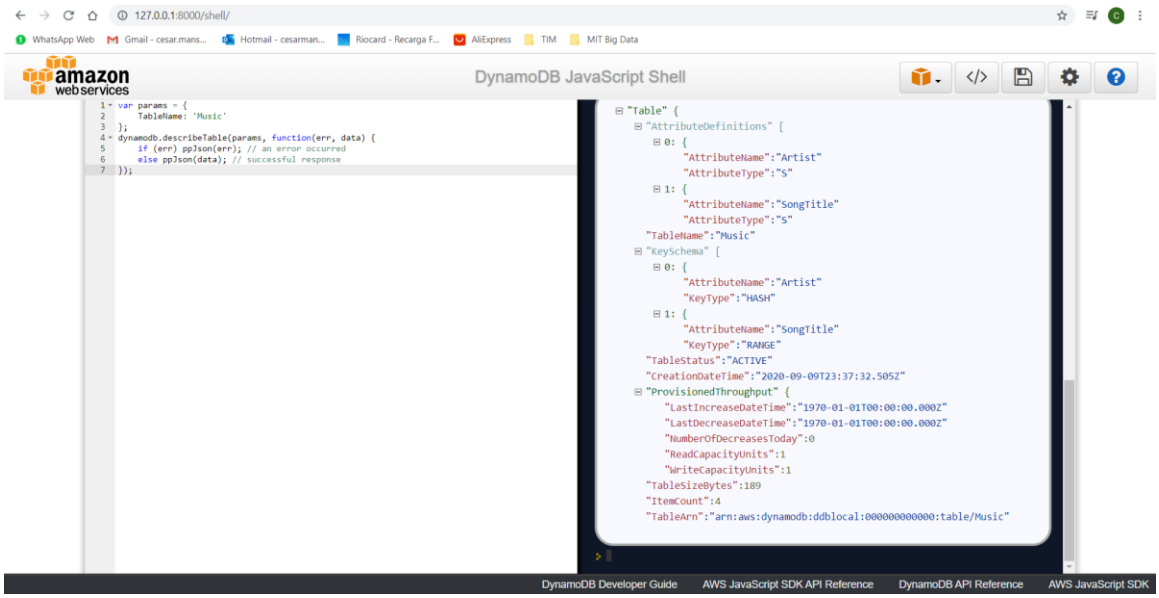
“listTables”: Lista as tabelas existentes.

```
var params = {
 };
 dynamodb.listTables(params, function(err, data) {
   if (err) console.log(err, err.stack); // an error occurred
   else     console.log(data);           // successful response
});
```



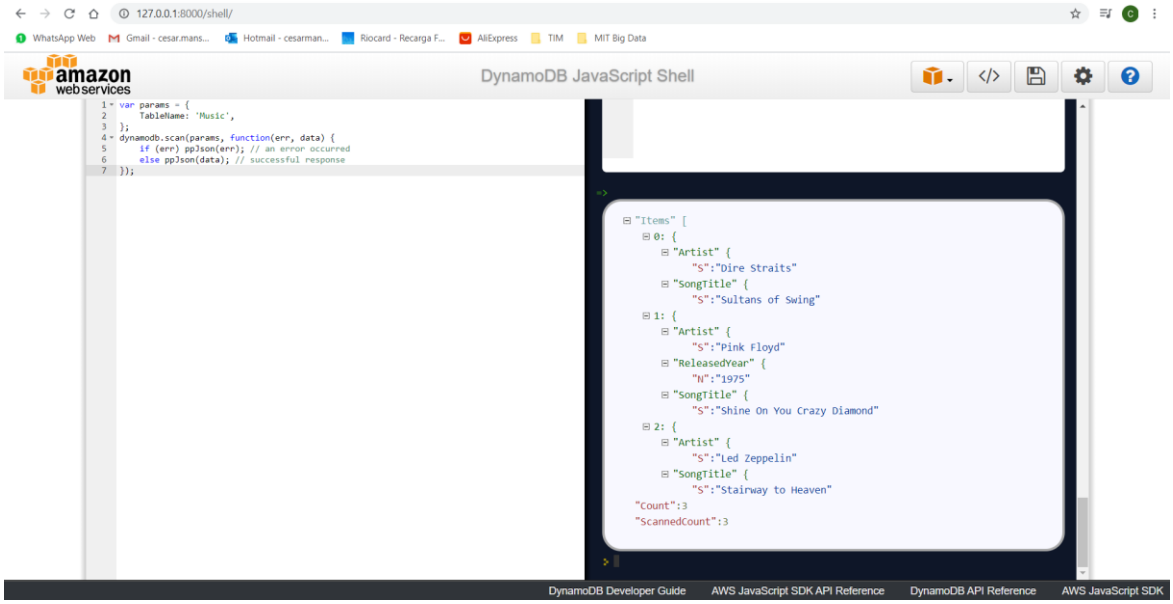
“describeTable”: Mostra a descrição de uma tabela.

```
var params = {
    TableName: 'Music'
};
dynamodb.describeTable(params, function(err, data) {
    if (err) ppJson(err); // an error occurred
    else ppJson(data); // successful response
});
```



“scan”: Permite consultar todos os itens de uma tabela, ou aplicar uma expressão de filtro.

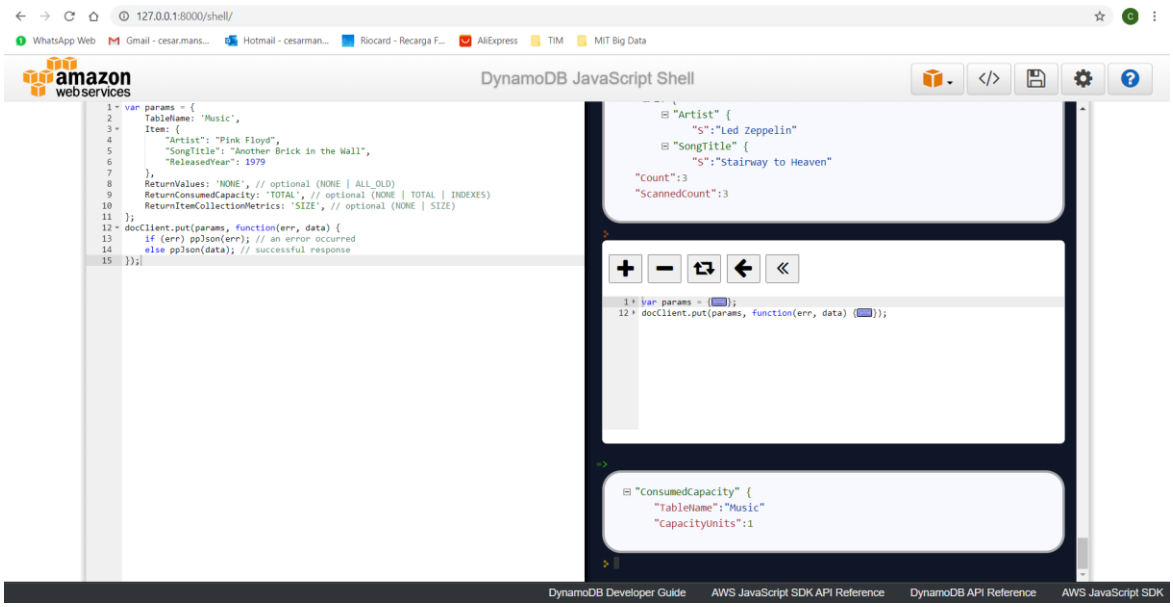
```
var params = {
    TableName: 'Music',
};
dynamodb.scan(params, function(err, data) {
    if (err) ppJson(err); // an error occurred
    else ppJson(data); // successful response
});
```



“put”: Insere um item na tabela.

```
var params = {
    TableName: 'Music',
    Item: {
        "Artist": "Pink Floyd",
        "SongTitle": "Another Brick in the Wall",
        "ReleasedYear": 1979
    },
    ReturnValues: 'NONE', // optional (NONE | ALL_OLD)
    ReturnConsumedCapacity: 'TOTAL', // optional (NONE | TOTAL | INDEXES)
    ReturnItemCollectionMetrics: 'SIZE', // optional (NONE | SIZE)
};
docClient.put(params, function(err, data) {
    if (err) ppJson(err); // an error occurred
    else ppJson(data); // successful response
});
```

PS: Note que nesta interface não é especificado o tipo do atributo.



3.8 Utilizando o DynamoDB – NoSQL Workbench for DynamoDB

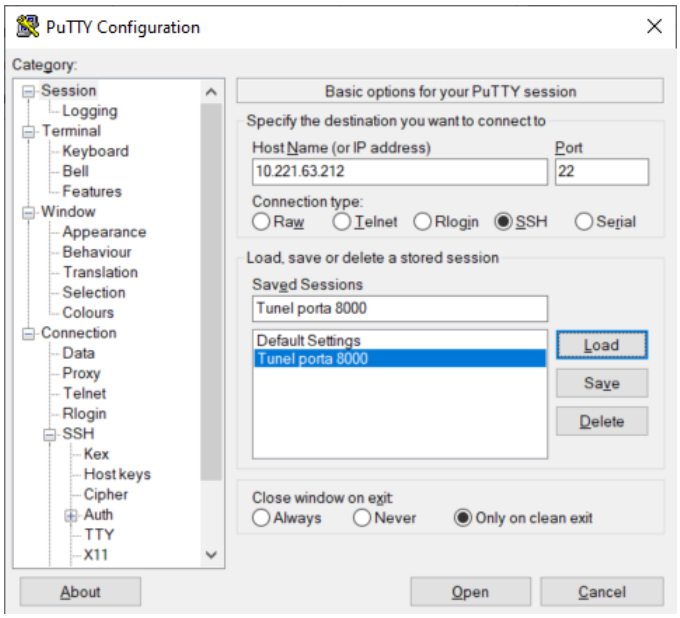
O “NoSQL Workbench for DynamoDB” é uma interface gráfica, para interação com o DynamoDB, extremamente fácil e intuitiva.

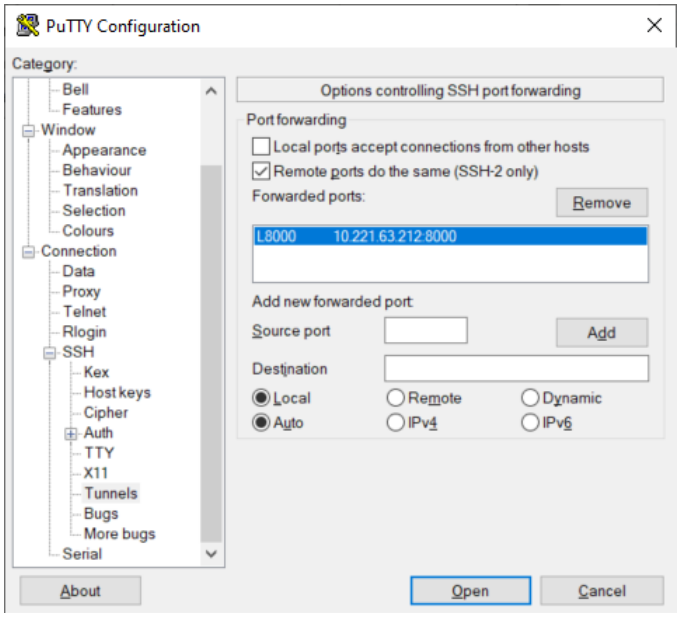
Download:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/workbench.settingup.html

3.8.1 Criando uma conexão com o DynamoDB Local

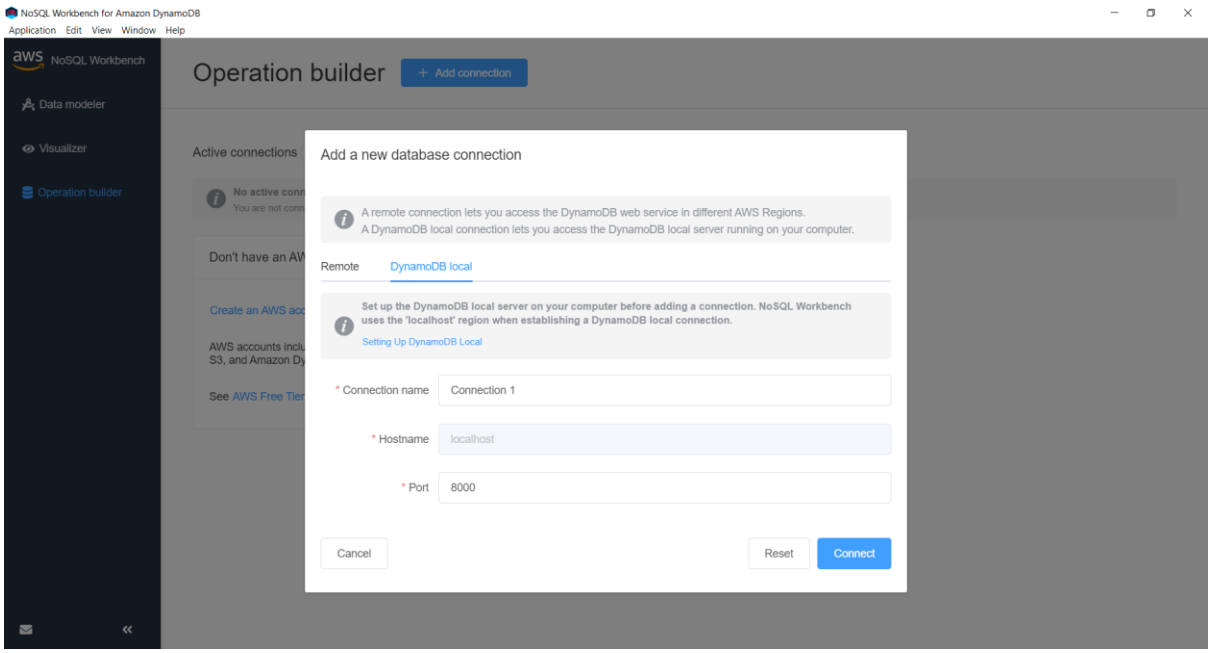
O “NoSQL Workbench for DynamoDB” vai tentar acessar o IP 127.0.0.1 para conexão com o DynamoDB Local, partindo da premissa que o Dynamodb Local está instalado na mesma máquina que o “NoSQL Workbench for DynamoDB”. Não é possível alterar essa configuração. Se você instalou o DynamoDB Local em outra máquina (ou em uma máquina virtual), será necessário criar um túnel SSH para viabilizar o acesso via “NoSQL Workbench for DynamoDB”.

Neste exemplo, o DynamoDB está instalado em outra máquina: IP 10.221.63.212 – porta 8000. Devemos então executar as configurações abaixo no PuTTY para estabelecer um túnel SSH entre a máquina local (IP 127.0.0.1 – porta 8000) e a máquina onde o DynamoDB está instalado (10.221.63.212 – porta 8000). Com a configuração feita, devemos realizar a conexão SSH na máquina onde o DynamoDB está instalado, para ativar o túnel. Feito isto, podemos o acessar o DynamoDB Local através do "NoSQL Workbench for DynamoDB".

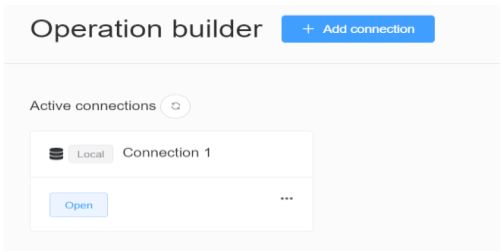




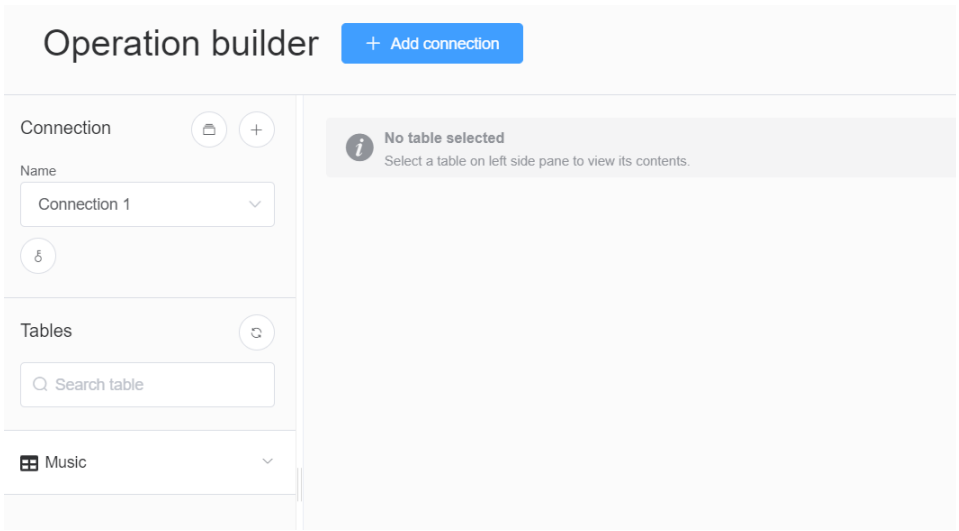
Na tela inicial do “NoSQL Workbench for DynamoDB”, clicar em “Operation Builder” e depois em “Add connection”. Na janela de criação da conexão, clicar em “DynamoDB local”:



Após criação da conexão, clicar em “Open”:

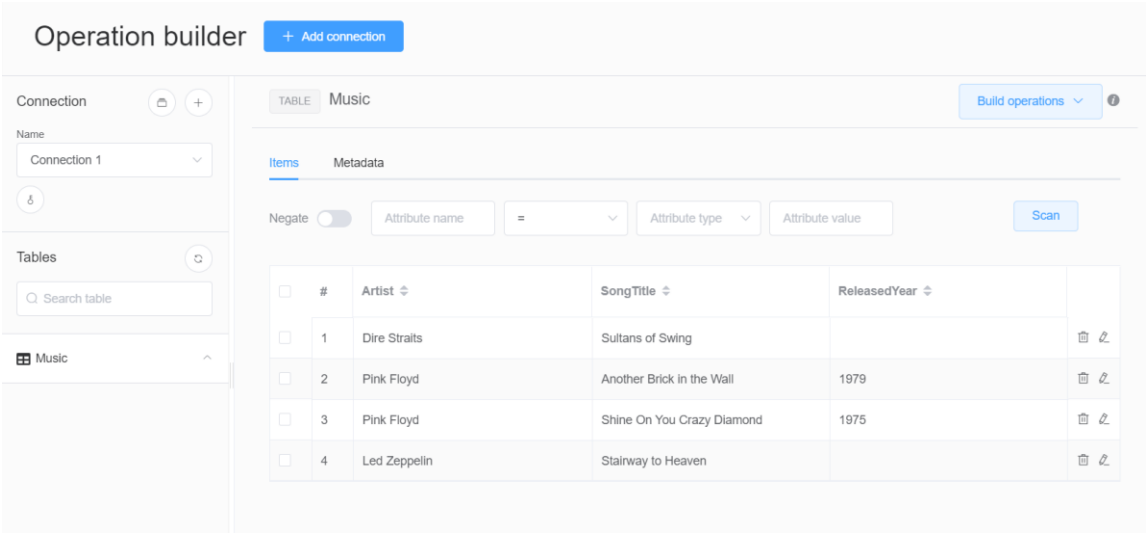


Nesta etapa, estamos conectados no “DynamoDB Local”:



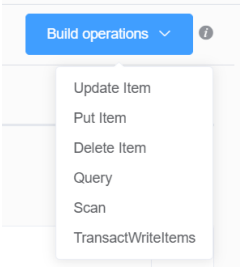
3.8.2 Tabelas

Visualizando o conteúdo de uma tabela:

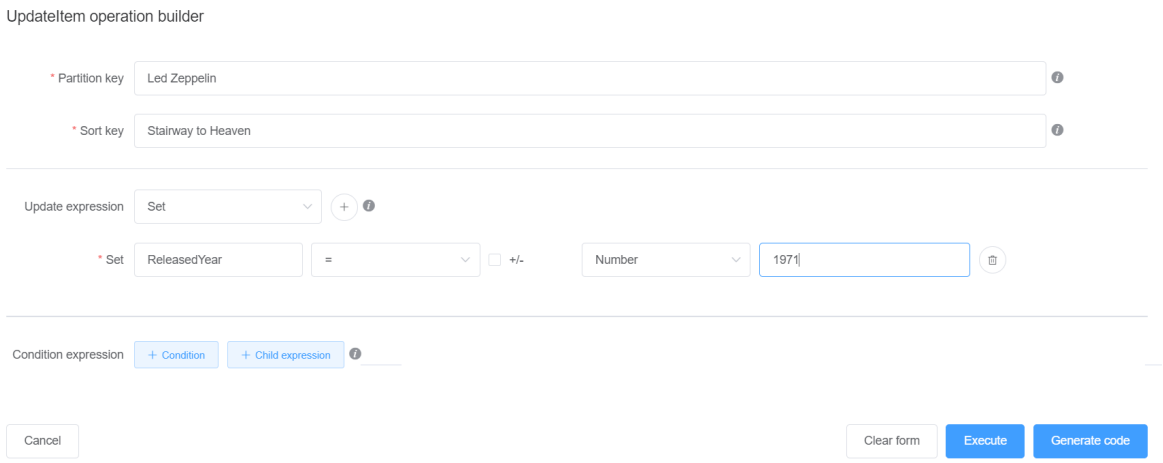


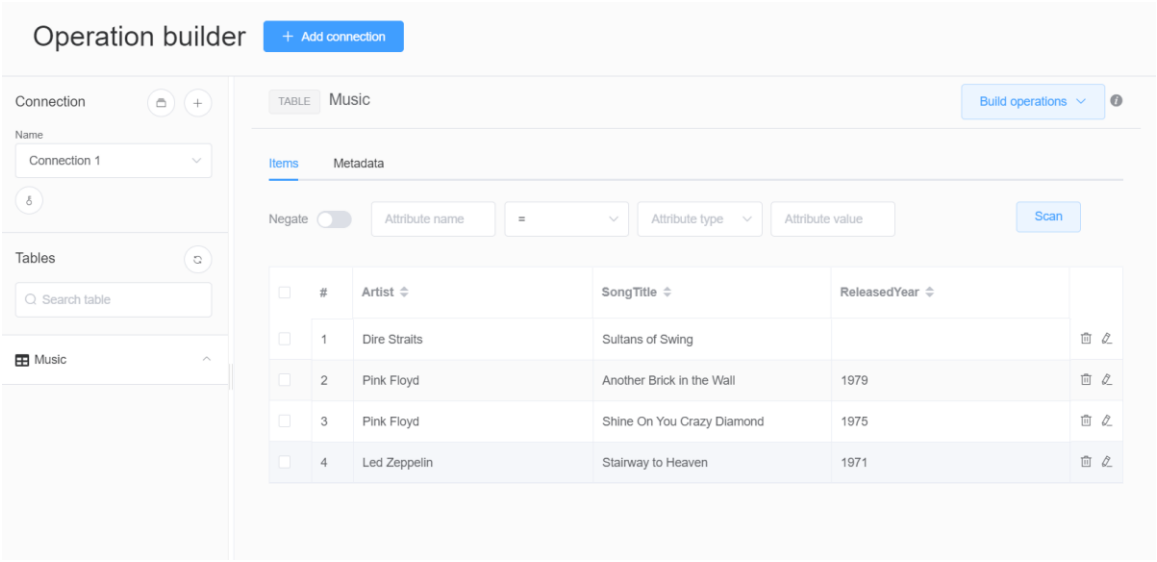
3.8.3 Itens

Operações permitidas:

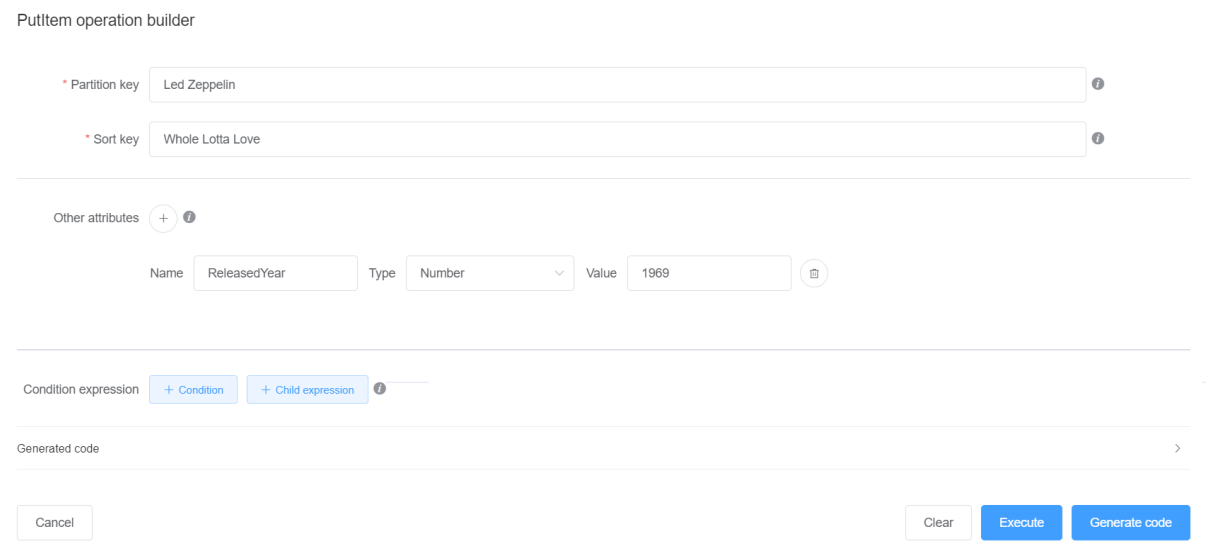


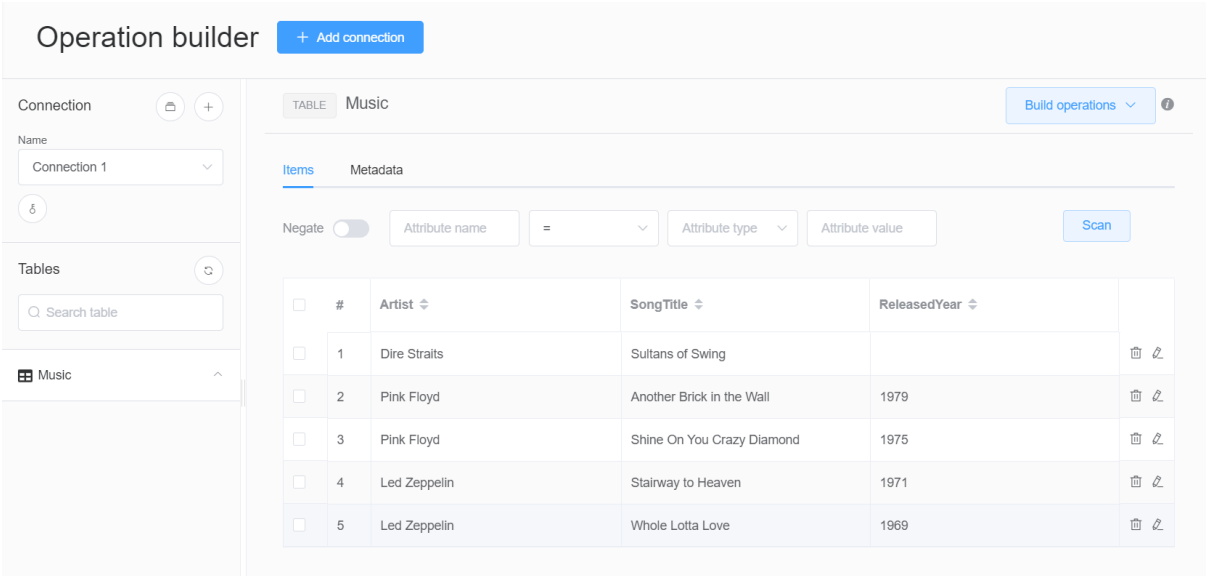
Atualizando um item:





Inserindo um item:

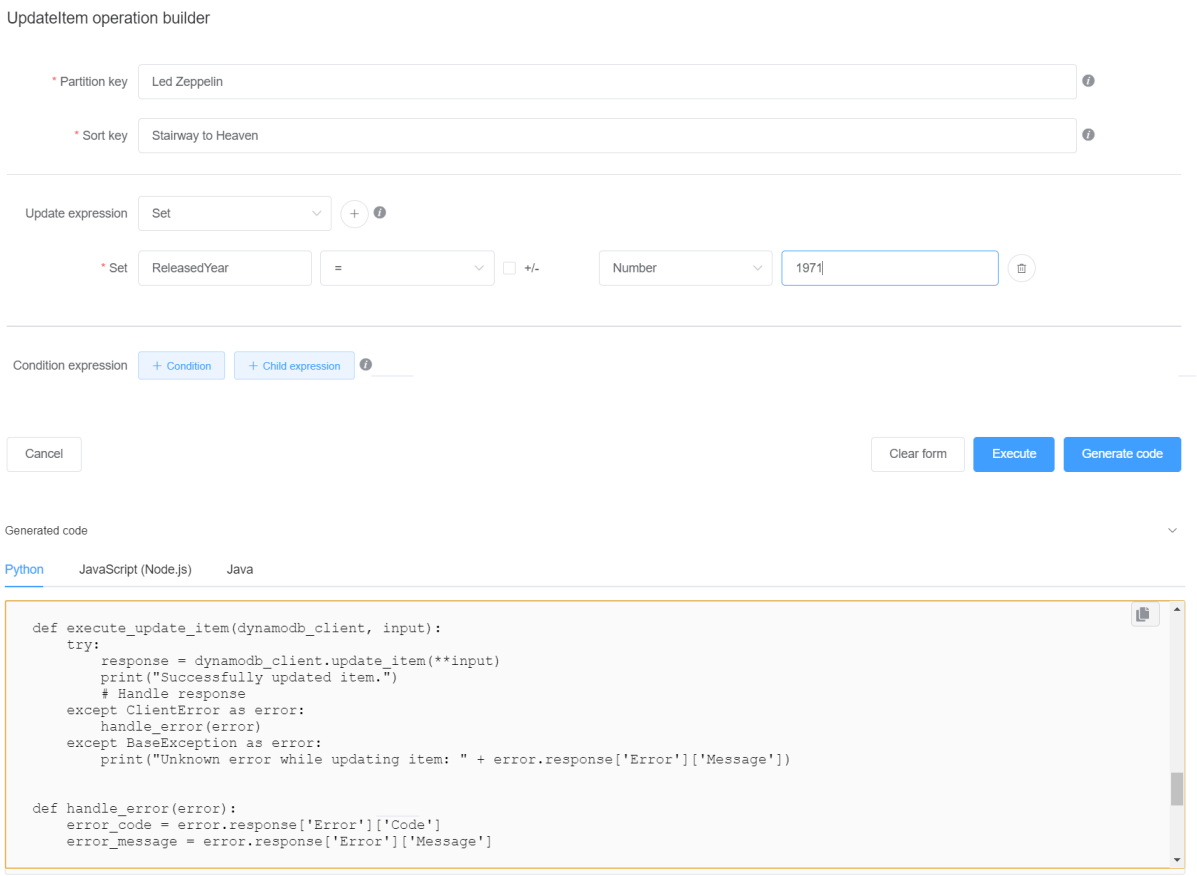




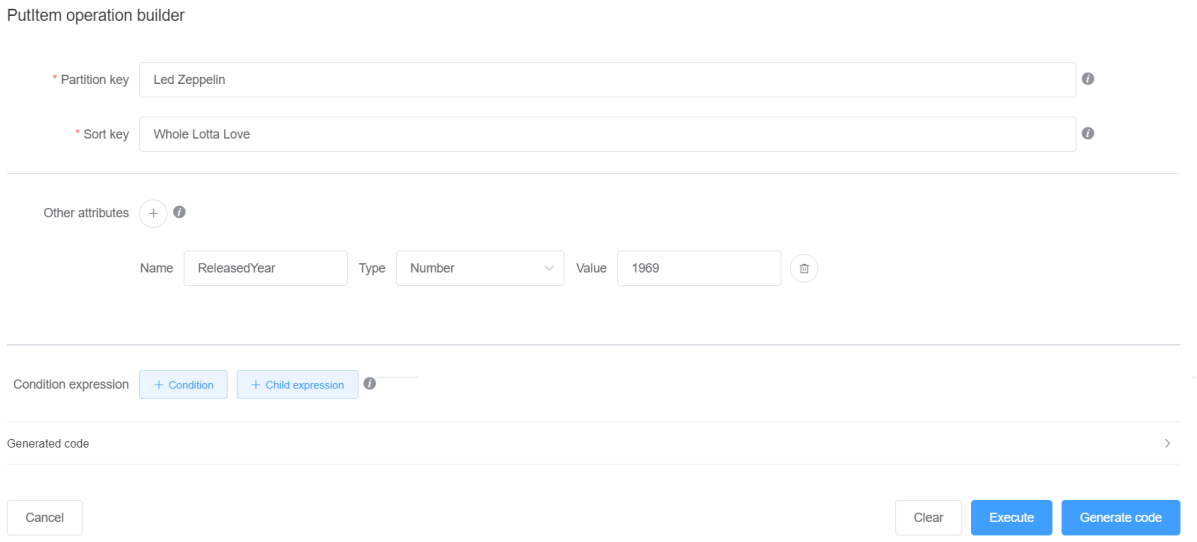
3.8.4 Geração automática de códigos (Python, Javascript e Java)

Clicando no botão “Generate code”, o “NoSQL Workbench for DynamoDB” gera automaticamente o código referente à ação, em qualquer uma das linguagens listadas (Python, Node.js e Java).

Exemplo de código Python gerado automaticamente para atualizar um item:



Exemplo de código Python gerado automaticamente para inserir um item:



Generated code

Python JavaScript (Node.js) Java

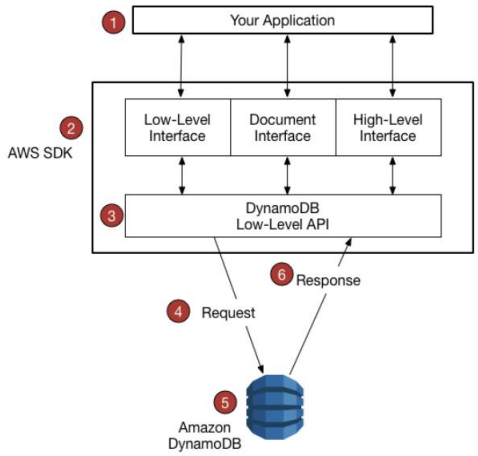
```
def execute_put_item(dynamodb_client, input):
    try:
        response = dynamodb_client.put_item(**input)
        print("Successfully put item.")
        # Handle response
    except ClientError as error:
        handle_error(error)
    except BaseException as error:
        print("Unknown error while putting item: " + error.response['Error']['Message'])


def handle_error(error):
    error_code = error.response['Error']['Code']
    error_message = error.response['Error']['Message']
```

3.9 Utilizando o DynamoDB – APIs REST

Conforme visto nos itens anteriores, presentamos algumas opções que podem ser utilizadas para trabalhar interativamente com o Amazon DynamoDB. No entanto, para obter o máximo do DynamoDB, podemos escrever códigos usando os SDKs da AWS.

Os SDKs da AWS fornecem amplo suporte para DynamoDB nas seguintes linguagens: Java, JavaScript, .NET, Node.js, PHP, Python, Ruby, C++, Go, Android e iOS.

Visão geral SDK da AWS para DynamoDB:



1. Você escreve um aplicativo usando um SDK da AWS para sua linguagem de programação.
2. Cada AWS SDK fornece uma ou mais interfaces programáticas para trabalhar com o DynamoDB.
3. O AWS SDK constrói solicitações HTTP(S) para uso com a API DynamoDB de baixo nível.
4. O AWS SDK envia a solicitação ao endpoint do DynamoDB.
5. O DynamoDB executa a solicitação. Se a solicitação for bem-sucedida, o DynamoDB retorna um código de resposta HTTP 200 (OK). Se a solicitação não for bem-sucedida, o DynamoDB retornará um código de erro HTTP e uma mensagem de erro.
6. O SDK da AWS processa a resposta e a propaga de volta para seu aplicativo.

Os serviços a seguir já são implementados pelo SDK da AWS, dispensando qualquer desenvolvimento adicional neste sentido:

- Formatar solicitações HTTP(S) e serializar parâmetros de solicitação.
- Gerar uma assinatura criptográfica para cada solicitação.

- Encaminhar solicitações para um endpoint do DynamoDB e receber respostas do DynamoDB.
- Extrair os resultados dessas respostas.
- Implementar a lógica de repetição básica em caso de erros.

Low-Level Interfaces

Todo SDK da AWS específico de cada linguagem fornece uma interface de baixo nível para o DynamoDB, com métodos que se assemelham a solicitações de API do DynamoDB de baixo nível.

Document Interfaces

Muitos SDKs da AWS fornecem uma interface de documento, permitindo que você execute operações (criar, consultar, atualizar, excluir) em tabelas e índices. Com uma interface de documento, você não precisa especificar os descritores de tipo de dados. Os tipos de dados estão implícitos na semântica dos próprios dados. Esses SDKs da AWS também fornecem métodos para converter facilmente documentos JSON de e para tipos de dados nativos do Amazon DynamoDB.

High-Level Interfaces

Os SDKs da AWS para Java e .NET fornecem interfaces adicionais com níveis mais altos de abstração. As interfaces de alto nível para o DynamoDB permitem definir as relações entre os objetos em seu programa e as tabelas de banco de dados que armazenam os dados desses objetos. Depois de definir esse mapeamento, você chama métodos de objeto simples, como salvar, carregar ou excluir, e as operações subjacentes de baixo nível do DynamoDB são chamadas automaticamente em seu nome. Isso permite que você escreva código centrado no objeto, em vez de código centrado no banco de dados.

DynamoDB Low-Level API

Você pode construir sua própria API RESTful (ou seja, usar o SDK da AWS para sua linguagem de programação) que atinge o Dynamo, reformata os dados e os retorna.

Ou você pode utilizar os serviços da AWS para construir sua API RESTful de forma muito mais prática, através do API Gateway da AWS, que aponta para uma função Lambda, para buscar os dados do DynamoDB. A URL do API Gateway será o endpoint da sua API REST.

4 Casos de uso

4.1 Ad Tech

As empresas do setor de tecnologia de anúncios (ad tech) usam o DynamoDB como armazenamento de chave-valor para armazenar diversos tipos de dados de marketing, como perfis de usuário e eventos, cliques e links visitados de usuários. Os casos de uso aplicáveis incluem Real-Time Bidding (RTB – Ofertas em tempo real), direcionamento de anúncios e atribuição. Esses casos de uso exigem uma alta taxa de solicitações (milhões de solicitações por segundo), baixa latência previsível e confiabilidade. As empresas usam armazenamento em cache por meio do DynamoDB Accelerator (DAX) quando têm altos volumes de leitura ou precisam de latência de leitura inferior a um milissegundo. Cada vez mais, as empresas de ad tech precisam implantar plataformas de RTB e direcionamento de anúncios em mais de uma região geográfica da AWS, o que demanda replicação de dados entre regiões.

Casos de uso comuns:
Armazenamentos de perfis de usuários para RTB e direcionamento de anúncios.
Eventos de usuários, clickstreams de datastore de impressões.
Datastores de metadados de ativos.
Caches de itens populares.

4.2 Jogos

As empresas do setor de jogos usam o DynamoDB em todos os recursos das plataformas de jogos, incluindo estado dos jogos, dados dos jogadores, histórico de sessões e tabelas de classificação. Os principais benefícios oferecidos pelo DynamoDB a essas empresas são a capacidade de escalar com confiabilidade para milhões de usuários concorrentes e solicitações e a garantia de uma baixa latência consistente, abaixo de 10 milissegundos. Além disso, como um serviço gerenciado, o DynamoDB não tem sobrecarga operacional. Assim, os desenvolvedores podem se concentrar no desenvolvimento de jogos em vez de no gerenciamento de banco de dados. À medida que os desenvolvedores de jogos procuram cada vez mais ampliar a disponibilidade de uma única região para várias regiões da AWS, podem confiar nas tabelas globais do DynamoDB para replicação multirregião de dados no modo ativo-ativo.

Casos de uso comuns:
Estados de jogos
Datastores de jogadores
Datastores de histórico de sessões de jogadores
Tabelas de classificação

4.3 Varejo

Muitas empresas do setor de varejo usam padrões de projeto comuns do DynamoDB para entregar baixa latência consistente em casos de uso de missão crítica. Não estar preso a preocupações de escalabilidade e sobrecargas operacionais é uma vantagem competitiva essencial e um fator habilitador de eventos de alta velocidade em escala extrema, como o Amazon Prime Day, cuja magnitude é difícil de prever. A capacidade de aumentar e reduzir a escala permite que esses clientes paguem apenas pela capacidade necessária e concentrem recursos técnicos valiosos na inovação e não nas operações.

Casos de uso comuns:
Carrinhos de compras
Mecanismos de fluxo de trabalho
Controle de estoque e atendimento
Perfis e contas de usuários

4.4 Serviços bancários e financeiros

À medida que as empresas de serviços bancários e financeiros criam mais aplicativos nativos da nuvem, procuram usar serviços gerenciados para aumentar a agilidade, reduzir o tempo de introdução no mercado e minimizar a sobrecarga operacional. Ao mesmo tempo, precisam garantir a segurança, a confiabilidade e a alta disponibilidade dos aplicativos. Quando essas empresas ampliam serviços atuais, baseados em sistemas de mainframe legados, constatam que esses sistemas não conseguem atender às demandas de escalabilidade decorrentes da crescente base de usuários; das novas plataformas, como aplicativos móveis; e do crescimento de tráfego resultante. Para resolver esses problemas, as empresas replicam dados dos mainframes para a nuvem com o intuito de transferir o tráfego.

Casos de uso comuns:
Transações de usuários
Processamento de transações orientadas a eventos
Detecção de fraudes
Transferência do mainframe e captura de alterações de dados

4.5 Mídia e entretenimento

As empresas de mídia e entretenimento usam o DynamoDB quando precisam de throughput e simultaneidade, baixa latência e confiabilidade em escalas extremas. O DynamoDB escala de maneira elástica para absorver a carga e mantém a baixa latência essencial para cenários em tempo real, como streaming de vídeo e conteúdo interativo. Nesses cenários, o número de usuários simultâneos pode atingir milhões. Nenhum banco de dados comporta esse nível de simultaneidade tão bem quanto o DynamoDB. Apesar dessa alta simultaneidade, a latência permanece baixa, proporcionado uma experiência de usuário ideal aos indivíduos que acessam sua mídia ou participam de um evento interativo em tempo real. Essas empresas usam o DynamoDB para superar desafios de escalabilidade e manter o foco no desenvolvimento de recursos e não no gerenciamento de bancos de dados.

Casos de uso comuns:
Datastores de metadados de mídia
Datastores de usuários
Datastores de gerenciamento de direitos digitais

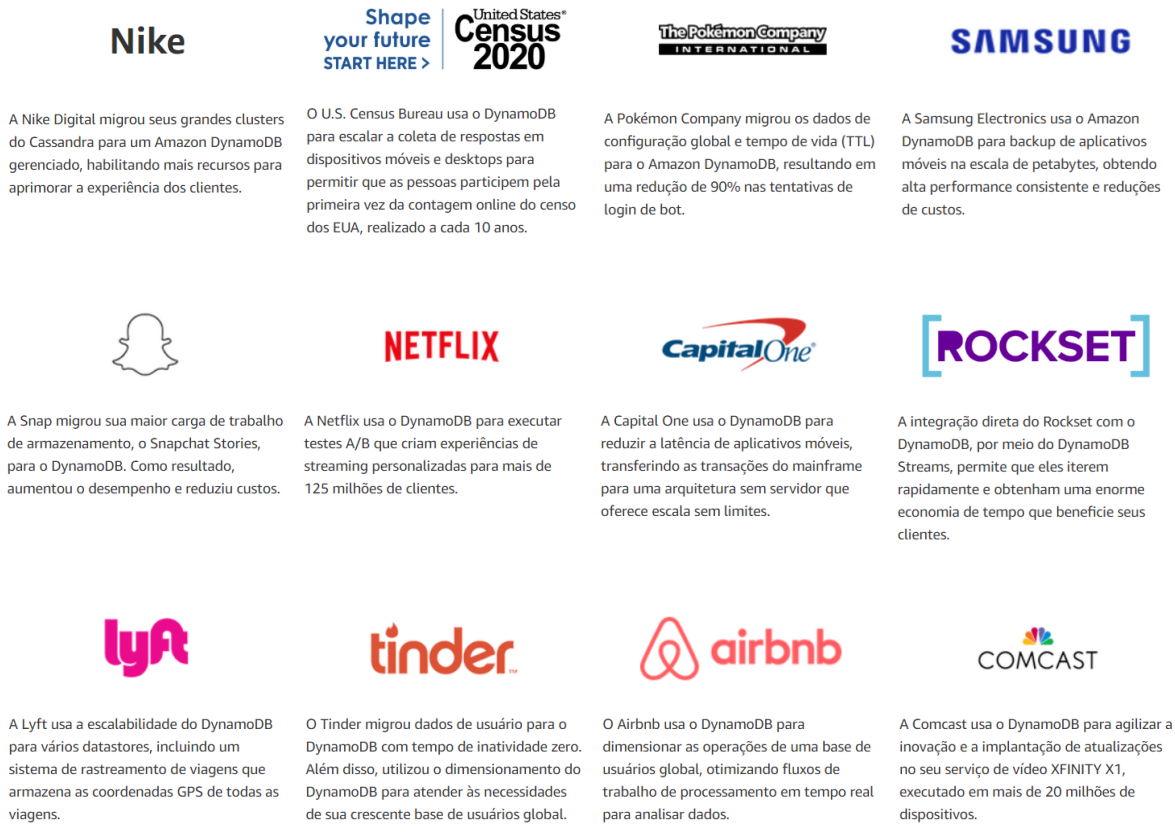
4.6 Software e Internet

Um fator comum importante entre empresas de software e vários outros clientes do DynamoDB é a escala da Internet. Os casos de uso dessas empresas exigem a capacidade de acomodar simultaneidade, taxas de solicitação e picos de tráfego em níveis extremos. Essa simultaneidade é

medida em milhões de usuários e conexões, e as taxas de solicitação podem atingir facilmente milhões por segundo. O DynamoDB tem um histórico comprovado de comportar casos de uso na escala da Internet e seus requisitos ao mesmo tempo que mantém uma latência consistente e inferior a 10 milissegundos. Com as tabelas globais, os clientes do DynamoDB podem ampliar facilmente os aplicativos para várias regiões da AWS, obtendo alcance global e continuidade dos negócios.

Casos de uso comuns:
Datastores de metadados de conteúdo de usuários
Datastores de gráficos de relacionamentos
Caches de metadados
Datastores de controle de transporte por aplicativo
Datastores de usuários, veículos e motoristas
Datastores de vocabulário de usuários

4.7 Principais clientes



Mais alguns exemplos:

Duolingo
Site de aprendizado online, que utiliza o DynamoDB para armazenar aproximadamente 31 bilhões de objetos de dados em seu servidor web.

Esta startup tem cerca de 18 milhões de usuários mensais que realizam cerca de seis bilhões de exercícios usando o aplicativo Duolingo.
Como seu aplicativo tem 24.000 unidades de leitura por segundo e 3.300 unidades de gravação por segundo, o DynamoDB acabou sendo a escolha certa para eles. A equipe tinha muito pouco conhecimento sobre DevOps e gerenciamento de sistemas de grande escala quando começou. Devido ao uso global do Duolingo e à necessidade de dados personalizados, o DynamoDB é o único banco de dados que foi capaz de atender às suas necessidades, tanto em termos de armazenamento de dados quanto de DevOps.
Além disso, o fato de o DynamoDB ser dimensionado automaticamente significa que essa pequena inicialização não precisou usar seus desenvolvedores para ajustar o tamanho manualmente. O DynamoDB foi simplificado e também dimensionado para atender às suas necessidades.

Major League Baseball (MLB)
Há muitas coisas que consideramos certas quando assistimos a um jogo de beisebol.
Por exemplo, você sabia que existe um sistema de radar Doppler que fica atrás da placa base, amostrando a posição da bola 2.000 vezes por segundo? Ou que existem dois dispositivos de imagem estereoscópica, geralmente posicionados acima da linha da terceira base, que amostram as posições dos jogadores em campo 30 vezes por segundo?
Todas essas transações de dados requerem um sistema rápido em leituras e gravações. A MLB usa uma combinação de componentes da AWS para ajudar a processar todos esses dados. O DynamoDB desempenha um papel fundamental em garantir que as consultas sejam rápidas e confiáveis.

Bemobi
A Bemobi é uma empresa multinacional de mídia e entretenimento mobile com escritórios no Brasil, Ucrânia, Noruega, Índia e presença em diversos outros países. Integra clientes e conteúdo mobile, através de tecnologias e modelos de negócio inovadores, sendo reconhecida pelo Instituto Great Place To Work como uma das melhores empresas para se trabalhar no Rio de Janeiro.
Recentemente a Bemobi iniciou seu projeto de migração para o DynamoDB.

4.8 Exemplos de scripts utilizados na Bemobi

4.8.1 Importação de dados para o DynamoDB

Script Python que cria o cluster EMR:


emr_create_cluster.py

Script de importação de dados para o DynamoDB usando HIVE:


Dynamo_migrate.txt

4.8.2 Extração de dados do DynamoDB

Script Python para subir o cluster EMR na AWS:

Cluster_EMR.py

Script Python para extrair dados do DynamoDB:

datalake_dynamodb_
extract_process_emr.p

5 Conclusão

O DynamoDB foi desenvolvido para entregar um alto desempenho podendo atender desde os menores aplicativos da internet, escalando até os maiores. Essa versatilidade, juntamente com o fato de ser serverless, o coloca em uma posição de destaque no mercado permitindo escalar computacionalmente de forma automática, sem downtime, com baixa latência, e com uma baixa intervenção humana. Também vale destacar toda a segurança fornecida pela AWS desde as instalações de infraestrutura até a proteção dos dados.

Seu grande diferencial, comparado com outras soluções escaláveis, é o fato de oferecer escalabilidade automática. Não há necessidade de ampliar previamente o cluster ou preparar com antecedência o ambiente para uma sazonalidade ou um grande evento programado, o que é fantástico. O viés dessa funcionalidade pode ser o custo.

Apesar do DynamoDB ter uma fatura flexível, podendo escolher entre on-demand e provisionado, os custos têm que receber uma atenção extra, já que escritas e leituras desnecessárias irão inflacionar a fatura. E para aplicações que necessitem de muitas leituras, deve ser avaliado o uso de DynamoDB Accelerator (DAX) para cachear os dados. Outro ponto que deve ser destacado é o elevado custo de armazenamento por GB, que é aproximadamente 40% a mais do que um banco relacional.

Portanto, o DynamoDB é capaz de sustentar a maioria das aplicações da internet com altíssimas taxas de requisições, mas sempre tendo muita atenção aos custos.

6 Referências

https://aws.amazon.com/dynamodb/

https://aws.amazon.com/documentdb/

https://aws.amazon.com/redis/

https://www.dynamodbguide.com

https://db-engines.com

https://db-engines.com/en/system/Amazon+DynamoDB%3BMongoDB

https://www.mongodb.com/compare/mongodb-dynamodb

https://www.educba.com/mongodb-vs-dynamodb/

https://redis.io/

https://www.dataversity.net/understanding-key-value-databases/

https://medium.com/@caseygibson_42696/difference-between-aws-dynamodb-vs-aws-documentdb-vs-mongodb-9cb026a94767

https://medium.com/better-programming/5-real-life-use-cases-for-dynamodb-a152a9d152e2

https://blog.yugabyte.com/11-things-you-wish-you-knew-before-starting-with-dynamodb/