DynamoDB

Dr Peadar Grant

February 9, 2024

Contents

Required reading

Docs: `https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/`

Inbuilt help for dynamodb:

- `aws dynamodb help` to see commnads
- `aws dynamodb create-table help` for the description, synopsis and options

1 Cloud databases

The cloud offers a number of persistence options, including both relational and non-relational databases. The array of choices can be confusing. Often a number of possible solutions exist for any given problem. Choosing the most suitable is not straightforward.

We will look at a simple database today, DynamoBD. According to AWS:

> DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability

1.1 Use cases

DynamoDBis a useful PaaS tool for simple data persistence that scales well:

- It is *NOT* a replacement for relational or other databases.
- There is much ill-informed comment on the internet touting the merits of different database technologies without any context or objective basis.
- DynamoDB can be used as a persistence store for:
  - applications themselves running with AWS (Lambda, EC2)
  - applications that run elsewhere (own server, own laptop).

This class provides a *basic* introduction to DynamoDB.

2 DynamoDB

2.1 Key components

A DynamoDB table is a collection of *items*. (e.g. People) Note that there is no equivalent of “database” grouping multiple tables!

2.2 Item

An item items is a collection of *attributes*. Each item is a member of a table. (e.g. a person) Similar to a row in a relational DB, CSV file or spreadsheet.

2.3 Attributes

Attributes are the fundamental data element. An attribute maps a key to a value for that item. (e.g. name = John) Similar to a column in a relational DB, CSV file or spreadsheet. Attributes can be:

Scalar: number, string, binary, Boolean, and null.

Set: multiple scalar values in a set. Allowed types are string set, number set, and binary set

Document types are list and map, roughly mapping to JSON document types.

Unlike a traditional DB, different items may have a differing set of attributes. Only the primary key attribute is required, .

2.4 Primary key

Every item must have a primary key. The primary key is either:

Simple primary key (Partition key) must uniquely identify each item. Name comes from its internal use in a hash function to distribute table contents among physical storage.

- If data already has a unique ID then it should be the simple primary key.

Composite primary key (Partition key and sort key): must together uniquely identify each item. Partition keys of 2 items can be same if sort keys differ.

- Partition key determins physical storage location.
- Items with same partition key are stored in ascending order of sort key.
- In larger workloads, must ensure partition key is not the same across large portions of data set.

Only scalar types can be used in primary keys.

3 Access pattern

Most server-side databases use a custom binary or text protocol on a specific port number. DynamoDB is HTTP based and has a web-service API. This can be used from the AWS CLI or any language that the AWS SDK supports.

4 Suitability

DynamoDB is a good introduction to cloud-based PaaS database services. However, its suitability for any given application needs to be considered carefully:

- Good for single-table applications where language used supports its API.
- It is NOT a drop-in replacement for an SQL DB.
- NOT a relational database: Has no foreign keys, no JOINs, no GROUP BY, no unique keys.
- Has single-digit millisecond response time.
- Not really designed for ad-hoc queries.

5 Operations

Assume we want to create a table `players`. Each item has an attribute `name` (string) and will have a second attribute, `points` (numeric). The handout assumes that you are looking at the help text for each command - explanations there will not be repeated here.

In practice, data manipulation operations (like inserting new data, deleting data, querying) are likely to come from application code in Java, C#, C++ via the AWS SDK rather than via direct CLI / AWS Console.

5.1 Choice of primary key

Here we will have a simple primary key, consisting of the partition key `name`. This means that every item in the `players` table *must* have a `name` attribute.

5.2 Table creation

```
$TableName="players"

# create the table
aws dynamodb create-table --table-name $TableName `
--attribute-definitions AttributeName=name,AttributeType=S `
--key-schema AttributeName=name,KeyType=HASH `
--billing-mode PAY_PER_REQUEST
```

The table creation command is asynchronous:

- Although it returns, the table may stay in the `CREATING` status for some time before it becomes `ACTIVE`.

5.3 Putting item into table

```
# basic usage
aws dynamodb put-item --table-name $TableName
--item '{\"name\": {\"S\": \"John\"}, \"points\": {\"N\": \"10\"}}'
```

The extra quotation marks are because of how strings need to be encoded within other strings. See:

`https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-parameters-quoting-strings.html`

5.4 Reading single item

```
# retrieve value based on key
aws dynamodb get-item --table-name $TableName --item '{\"name\": {\"S\": \"John\"}}'
```

The output is returned as a JSON dictionary representing the item's attributes:

```
# can get as PowerShell objects in usual way
$Item = ( aws dynamodb get-item --table-name $TableName --item '{\"name\": {\"S\": \"Jo
| ConvertFrom-Json ).Item
```

5.5 Getting all items

```
aws dynamodb scan --table-name $TableName
```

We could for example iterate over the returned items:

```
$Items = (aws dynamodb scan --table-name $TableName | ConvertFrom-Json).Items
foreach ( $Item in $Items ) {
    Write-Host "$($Item.name.S ) has $($Item.points.N) points"
}
```

5.6 Table deletion

```
aws dynamodb delete-table --table-name $TableName
```

6 CloudFormation

```yaml
Resources:
  Table:
    Type: AWS::DynamoDB::Table
    Properties:
      AttributeDefinitions:
        -
          AttributeName: id
          AttributeType: S
      BillingMode: PAY_PER_REQUEST
      KeySchema:
        -
          AttributeName: id
          KeyType: HASH
```