LEARNING amazon-dynamodb

#amazon-dynamodb

Table of Contents

About

Chapter 1: Getting started with amazon-dynamodb

Remarks

This section provides an overview of what amazon-dynamodb is, and why a developer might want to use it.

It should also mention any large subjects within amazon-dynamodb, and link out to the related topics. Since the Documentation for amazon-dynamodb is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

DynamoDB is a fully managed service provided by AWS. It does not need to be installed or configured. AWS is responsible for all administrative burdens of operating, scalling and backup/restore of the distributed database.

Introduction

DynamoDB is a Distributed NoSQL database, based on key-value architecture, fully managed by Amazon Web Services. It was designed to provide scalability, redundancy and failover in predictable performance.

Read Getting started with amazon-dynamodb online: https://riptutorial.com/amazon-dynamodb/topic/2085/getting-started-with-amazon-dynamodb

Chapter 2: Batch Operations: Things to know

Introduction

Database is an integral part of any application and performance and persistance are real challenges faced by any web application. NoSql databases are no different in this matter and need to be dealt carefully. DynamoDB being one of the NoSQL database that is provided by Amazon Web Services support batch operations in addition to the CRUD operations. Lets start with Batch Operations. In this example we will learn how we can make use of Dynamo DB's JAVA SDK to perform Batch Inserts.

Remarks

Good to know about Batch operations

1. Batch operation doesn't minimize the HTTP request count, rather it has more features to handle when we receive a throttling error. To put it simple, each record that we insert into dynamo db will consume one http request.
2. Sound implementation of batch operation is to first to cache data temporarily and once we have the threshold number, ie 25, is the correct time to fire a batch request.
3. Any request that fails due to throttling will be retried between(500-999) milliseconds, as recommended by Amazon best practices.

Examples

How to code the BatchWriteItemRequest and save data

```
private static void saveItem(List<EventTracker> items) {
    List<WriteRequest> wrList = new ArrayList<>();
    try {

        for (EventTracker item : items) {
            WriteRequest wreqItem;
            wreqItem = getWriteRequest(item);
            wrList.add(wreqItem);
        }

        try {

            BatchWriteItemResult batchWriteItemResult = new BatchWriteItemResult();
            do {
                BatchWriteItemRequest batchWriteItemRequest = new BatchWriteItemRequest();
                batchWriteItemRequest.addRequestItemsEntry(forumTableName, wrList);//
setRequestItems(writeRequestitems);
                batchWriteItemResult = amazonDynamoDB.batchWriteItem(batchWriteItemRequest);
                // Check for unprocessed keys which could happen if you
                // exceed
                // provisioned throughput
                Map<String, List<WriteRequest>> unprocessedItems =
batchWriteItemResult.getUnprocessedItems();
                if (unprocessedItems.size() == 0) {
                    System.out.println("No unprocessed items found");
                } else {
                    System.out.println("Sleeping for: " +
ThreadLocalRandom.current().nextInt(500, 999 + 1));
                    Thread.sleep(ThreadLocalRandom.current().nextInt(500, 999 + 1));
                    wrList = unprocessedItems.get(forumTableName);
                    System.out.println("Retrieving the unprocessed items");
                }

            } while (batchWriteItemResult.getUnprocessedItems().size() > 0);

        } catch (Exception e) {
            System.err.println("Failed to retrieve items: ");
            e.printStackTrace(System.err);
        }

    } catch (Exception e) {

    }
}
```

This is what we need to know if we want to make use of batch operations to put data to a dynamo db table. Lets see the steps that need to be followed to accomplish this. In this example, we are trying to persist a list of EventTracker data, which is my POJO.

1. For each record to be inserted, we need to create a PUT Request.
2. Each PUT Request is wrapped to a Write Request.
3. All Write Request are bundled into a List.
4. The WriteRequest List is then added to the BatchWriteItemRequest and executed.

The below code will show how we create write requests.

How to create WriteRequest

```
private static WriteRequest getWriteRequest(EventTracker event) {

    WriteRequest wreq = null;// = new WriteRequest();

    if (event != null) {
        Map<String, AttributeValue> attributeMap = new HashMap<String, AttributeValue>();

        addAttribute(attributeMap, "event_id", event.getEventId());
        addAttribute(attributeMap, "created_datetime", event.getCreatedDatetime());
        addAttribute(attributeMap, "event", event.getEvent());
        addAttribute(attributeMap, "event_type", event.getEventType());
        addAttribute(attributeMap, "response_id", "NULL");

        wreq = new WriteRequest(new PutRequest(attributeMap));
    }
    return wreq;
}

private static void addAttribute(Map<String, AttributeValue> item, String attributeName,
String value) {
    AttributeValue attributeValue = new AttributeValue(value);
    item.put(attributeName, attributeValue);
}
```

Read Batch Operations: Things to know online: https://riptutorial.com/amazon-dynamodb/topic/8675/batch-operations--things-to-know

Chapter 3: Dynamodb delete data over time

Introduction

Removing old data from dynamodb using a date attribute.

Remarks

My use case: removing old data from dynamodb using a date attribute.

Important things to know:

- You can't query a table with using only range key attribute (date for example).
- You can only query a table using hash or hash+range key.
- You can't query a table using a hash key with '<' / '>' operations, only '='.

Possible Solutions:

- Scanning the whole table - this could be very costly
- My chosen solution - Defining an index with range key for the date and with a hash key that would be pretty decent such as the day of year.

Eventually batch delete the result set.

Notes: Building the entity I was using the amazon dynamo annotations. I was using DynamoDBQueryExpression to query, getting the result page with the defined Class object.

Examples

cleanUpOldData

```
public static void cleanUpOldData(AmazonDynamoDB amazonDynamoDB, String dynamoDBTablesPrefix,
String tableName,
                                  String dateRangeField, String dateHashKey, String dateIndex,
Class clazz) {
    log.info(String.format("Cleaning old data from table: %s", tableName));

    long cleanUpDateInMillis = (new Date()).getTime() - CLEAN_UP_TIME_MILLIS;
    SimpleDateFormat dateFormatter = new SimpleDateFormat(DYNAMO_DATE_FORMAT);
    final TimeZone utcTimeZone = TimeZone.getTimeZone("UTC");
    dateFormatter.setTimeZone(utcTimeZone);
    String cleanUpDate = dateFormatter.format(cleanUpDateInMillis);

    Calendar calendar = Calendar.getInstance(utcTimeZone);
    calendar.setTimeInMillis(cleanUpDateInMillis);

    final String dailyHashKey = String.format("%s_%s", calendar.get(Calendar.YEAR),
calendar.get(Calendar.DAY_OF_YEAR));
    final String pastDayHashKey = String.format("%s_%s", calendar.get(Calendar.YEAR),
calendar.get(Calendar.DAY_OF_YEAR)-1);

    final String fullTableName = dynamoDBTablesPrefix + "_" + tableName;
    final DynamoDBMapperConfig dbMapperConfig = new DynamoDBMapperConfig(new
DynamoDBMapperConfig.TableNameOverride(fullTableName));
    DynamoDBMapper mapper = new DynamoDBMapper(amazonDynamoDB, dbMapperConfig);
    DynamoDBTableMapper dbTableMapper = mapper.newTableMapper(clazz);

    final QueryResultPage dailyResultPage = getDailyQueryResultPage(dateRangeField,
dateHashKey, dateIndex, cleanUpDate, dailyHashKey, dbTableMapper);
    final QueryResultPage pastDayResultPage = getDailyQueryResultPage(dateRangeField,
dateHashKey, dateIndex, cleanUpDate, pastDayHashKey, dbTableMapper);

    deleteOldData(dbTableMapper, dailyResultPage, pastDayResultPage);

    log.info(String.format("Completed cleaning old data from table: %s, %s items were
deleted", tableName,
            dailyResultPage.getCount() + pastDayResultPage.getCount()));
}

private static QueryResultPage getDailyQueryResultPage(String dateRangeField, String
dateHashKey, String dateIndex,
                                                       String cleanUpDate, String dayHashKey,
DynamoDBTableMapper dbTableMapper) {
    HashMap<String, String > nameMap = new HashMap<>();
    nameMap.put("#date", dateRangeField);
    nameMap.put("#day", dateHashKey);
    HashMap<String, AttributeValue> valueMap = new HashMap<>();
    valueMap.put(":date", new AttributeValue().withS(cleanUpDate)) ;
    valueMap.put(":day", new AttributeValue().withS(dayHashKey));

    final DynamoDBQueryExpression dbQueryExpression = new DynamoDBQueryExpression()
            .withIndexName(dateIndex)
            .withConsistentRead(false)
            .withKeyConditionExpression("#day = :day and #date < :date")
            .withExpressionAttributeNames(nameMap)
            .withExpressionAttributeValues(valueMap);
    return dbTableMapper.query(dbQueryExpression);
}

private static void deleteOldData(DynamoDBTableMapper dbTableMapper, QueryResultPage
dailyResultPage, QueryResultPage pastDayResultPage) {
    if (dailyResultPage.getCount() > 0) {
        dbTableMapper.batchDelete(dailyResultPage.getResults());
    }
    if (pastDayResultPage.getCount() > 0) {
        dbTableMapper.batchDelete(pastDayResultPage.getResults());
    }
}
```

Read Dynamodb delete data over time online: https://riptutorial.com/amazon-dynamodb/topic/10889/dynamodb-delete-data-over-time

Chapter 4: How to create a DynamoDB Table

Remarks

When creating tables make sure to pay attention to the choice of attributes for the partition and sort keys. See the published guidelines for working with tables.

Examples

Create Table in Java using Document API

In the following example we will be creating a table called `Membership` using the AWS Java SDK for DynamoDB. The table will consist of items that represent team assignments. The table will be partitioned by TeamID. Each team will have multiple members, identified by the MemberID (as a sort key).

```
AWSCredentials credentials = new BasicAWSCredentials("access_key", "secret_key");
DynamoDB dynamoDB = new DynamoDB(new AmazonDynamoDBClient(credentials));

try {
      // every DynamoDB table must have basic schema that determines
      //   which attributes are to be used as partition key (and optionally sort key)
  List<KeySchemaElement> keySchema = new ArrayList<~>();
      // required: specify the partition key (also called hash key)
  keySchema.add(new KeySchemaElement()
              .withAttributeName("TeamID")
              .withKeyType(KeyType.HASH));
      // optionally: add a sort key (also called a range key)
  keySchema.add(new KeySchemaElement()
              .withAttributeName("MemberID")
              .withKeyType(KeyType.RANGE));

      // after defining the key schema - the attributes that will be used as partition and
range key
      //  we need to specify these attributes' type
  List<AttributeDefinition> attrDefinitions = new ArrayList<~>();
      //  we must specify the type for the TeamID attribute; suppose it's a string
  attrDefinitions.add(new AttributeDefinition()
                   .withAttributeName("TeamID")
                   .withAttributeType("S"));
      //  if using a sort key we need to specify its type; suppose that it's numeric
  attrDefinitions.add(new AttributeDefinition()
                   .withAttributeName("MemberID")
                   .withAttributeType("N"));


      // after defining the attributes making up the schema and their type
      // we build a request, specifying the table name and the provisioned capacity
  CreateTableRequest request = new CreateTableRequest()
      .withTableName("Membership")
      .withKeySchema(keySchema)
      .withAttributeDefinitions(attrDefinitions)
      .withProvisionedThroughput(new ProvisionedThroughput()
            .withReadCapacityUnits(30L)
            .withWriteCapacityUnits(10L));

      // now submit the request to create the table in DynamoDB
  Table table = dynamoDB.createTable(request);

      // creating a table is an asynchronous operation
      // we can wait for the table to be created
  table.waitForActive();

      // the table is now ready and can be used

} catch (Exception e) {
      // the table creation failed.
      //   the reason for failure may be determined from the exception
}
```

Create Table in Ruby using AWS SDK v2

In following example we will create table `movies` with AWS Ruby SDK v2. Here, each Movie as one unique Partition Key as `id`, and Range Key `year`. Apart from this we want to be able to query movies with their name, hence we will create a Global Secondary Index (GSI) `name-year-index` with `name` as Hash Key and `year` as Range Key. Movie can have other attributes such as `released`, `created_at`, `actor` and `actress`. Schema for the table is shown below:

Table Name: movies

Partition Key	Range Key	Global Secondary Index	Attributes
id	year	name	released , created_at, actor, actress

```
# it's better to initialize client as global variable in initializer
$ddb ||= Aws::DynamoDB::Client.new({
  access_key_id: ENV["AWS_ACCESS_KEY_ID"],
  secret_access_key: ENV["AWS_SECRET_ACCESS_KEY"]
})

# create table API
$ddb.create_table({

  # array of attributes name and their type that describe schema for the Table and Indexes
  attribute_definitions: [
    {
      attribute_name: "id",
      attribute_type: "N"
    }
    {
      attribute_name: "name",
      attribute_type: "S",
    },
    {
      attribute_name: "year",
      attribute_type: "N",
    }
  ],

  # key_schema specifies the attributes that make up the primary key for a table
  # HASH - specifies Partition Key
  # RANGE - specifies Range Key
  # key_type can be either HASH or RANGE
  key_schema: [
    {
      attribute_name: "id",
      key_type: "HASH",
    },
    {
      attribute_name: "year",
      key_type: "RANGE",
    }
  ],

  # global_secondary_indexes array specifies one or more keys that makes up index, with name
of index and provisioned throughput for global secondary indexes
  global_secondary_indexes: [

    index_name: "name-year-index",
    key_schema: [
        {
            attribute_name: "name",
            key_type: "HASH"
        },
        {
            attribute_name: "year",
            key_type: "RANGE"
        }
    ],

    # Projection - Specifies attributes that are copied (projected) from the table into the
index.
    # Allowed values are - ALL, INCLUDE, KEYS_ONLY
    # KEYS_ONLY - only the index and primary keys are projected into the index.
    # ALL - All of the table attributes are projected into the index.
    # INCLUDE - Only the specified table attributes are projected into the index. The list of
projected attributes are then needs to be specified in non_key_attributes array
    projection: {
        projection_type: "ALL"
    },

    # Represents the provisioned throughput settings for specified index.
    provisioned_throughput: {
        read_capacity_units: 1,
        write_capacity_units: 1
    }
  ],

  # Represents the provisioned throughput settings for specified table.
  provisioned_throughput: {
    read_capacity_units: 1,
    write_capacity_units: 1,
  },
  table_name: "movies"
})

# wait till table is created
```

```
$ddb.wait_until(:table_exists, {table_name: "movies"})
```

Read How to create a DynamoDB Table online: https://riptutorial.com/amazon-dynamodb/topic/7686/how-to-create-a-dynamodb-table

Chapter 5: How to insert data into table using DynamoDb?

Examples

Import a CSV file into a DynamoDB table using boto (Python package)

The Python function `import_csv_to_dynamodb(table_name, csv_file_name, colunm_names, column_types)` below imports a CSV file into a DynamoDB table. Column names and column must be specified. It uses boto. Below is the function as well as a demo (`main()`) and the CSV file used.

```
import boto

MY_ACCESS_KEY_ID = 'copy your access key ID here'
MY_SECRET_ACCESS_KEY = 'copy your secrete access key here'


def do_batch_write(items, table_name, dynamodb_table, dynamodb_conn):
    '''
    From https://gist.github.com/griggheo/2698152#file-gistfile1-py-L31
    '''
    batch_list = dynamodb_conn.new_batch_write_list()
    batch_list.add_batch(dynamodb_table, puts=items)
    while True:
        response = dynamodb_conn.batch_write_item(batch_list)
        unprocessed = response.get('UnprocessedItems', None)
        if not unprocessed:
            break
        batch_list = dynamodb_conn.new_batch_write_list()
        unprocessed_list = unprocessed[table_name]
        items = []
        for u in unprocessed_list:
            item_attr = u['PutRequest']['Item']
            item = dynamodb_table.new_item(
                    attrs=item_attr
            )
            items.append(item)
        batch_list.add_batch(dynamodb_table, puts=items)


def import_csv_to_dynamodb(table_name, csv_file_name, colunm_names, column_types):
    '''
    Import a CSV file to a DynamoDB table
    '''
    dynamodb_conn = boto.connect_dynamodb(aws_access_key_id=MY_ACCESS_KEY_ID,
aws_secret_access_key=MY_SECRET_ACCESS_KEY)
    dynamodb_table = dynamodb_conn.get_table(table_name)
    BATCH_COUNT = 2 # 25 is the maximum batch size for Amazon DynamoDB

    items = []

    count = 0
    csv_file = open(csv_file_name, 'r')
    for cur_line in csv_file:
        count += 1
        cur_line = cur_line.strip().split(',')

        row = {}
        for colunm_number, colunm_name in enumerate(colunm_names):
            row[colunm_name] = column_types[colunm_number](cur_line[colunm_number])

        item = dynamodb_table.new_item(
                    attrs=row
            )
        items.append(item)

        if count % BATCH_COUNT == 0:
            print 'batch write start ... ',
            do_batch_write(items, table_name, dynamodb_table, dynamodb_conn)
            items = []
            print 'batch done! (row number: ' + str(count) + ')'

    # flush remaining items, if any
    if len(items) > 0:
        do_batch_write(items, table_name, dynamodb_table, dynamodb_conn)


    csv_file.close()


def main():
    '''
    Demonstration of the use of import_csv_to_dynamodb()
    We assume the existence of a table named `test_persons`, with
    - Last_name as primary hash key (type: string)
    - First_name as primary range key (type: string)
    '''
    colunm_names = 'Last_name First_name'.split()
    table_name = 'test_persons'
    csv_file_name = 'test.csv'
    column_types = [str, str]
    import_csv_to_dynamodb(table_name, csv_file_name, colunm_names, column_types)


if __name__ == "__main__":
    main()
    #cProfile.run('main()') # if you want to do some profiling
```

`test.csv`'s content (must be located in the same folder as the Python script):

```
John,Doe
Bob,Smith
Alice,Lee
Foo,Bar
a,b
c,d
e,f
g,h
i,j
j,l
```

Read How to insert data into table using DynamoDb? online: https://riptutorial.com/amazon-dynamodb/topic/6354/how-to-insert-data-into-table-using-dynamodb-

Chapter 6: Using AWS DynamoDb with the AWS .NET SDK

Remarks

Amazon DynamoDB is a fast NoSQL database service offered by Amazon Web Services (AWS). DynamoDB can be invoked from .NET applications by using the AWS SDK for .NET. The SDK provides three different models for communicating with DynamoDB. This topic is introduces the various APIs in each model.

The Models

The SDK provides three ways of communicating with DynamoDB. Each one offers tradeoffs between control and ease of use. See the AWS .NET SDK Reference for details on the APIs below.

- **Low-level**: `Amazon.DynamoDBv2` namespace — This is a thin wrapper over the DynamoDB service calls. It matches all the service features. You can reference the service documentation to learn more about each individual operation.
- **Document Model**: `Amazon.DynamoDBv2.DocumentModel` namespace — This is a model that provides a simpler interface for dealing with data. DynamoDB tables are represented by `Table` objects, while individual rows of data are represented by `Document` objects. Conversion of .NET objects to DynamoDB data is automatic for basic types.
- **Object Persistence Model**: `Amazon.DynamoDBv2.DataModel` namespace — This set of APIs allow you to store and load .NET objects in DynamoDB. Objects must be marked up to configure the target table and the hash/range keys. `DynamoDBContext` acts on marked up objects. It is used to store and load DynamoDB data, or to retrieve .NET objects from a query or scan operation. Basic data types are automatically converted to DynamoDB data and converters allow arbitrary types to be stored in DynamoDB.

The three models provide different approaches to working with the service. While the low-level approach requires more client-side code — the user must convert .NET types such as numbers and dates to DynamoDB-supported strings — it provides access to all service features. By comparison, the Object Persistence Model approach makes it easier to use the service—since the user is for the most part working with familiar .NET objects—but does not provide all the functionality. For example, it is not possible to make conditional Put calls with the Object Persistence Model.

Learn more about working AWS using the .NET SDK in the .NET SDK Developer Guide.

Note: This topic was adapted with permission from a blog post originally published on the AWS .NET SDK blog.

Examples

Low Level API Example

```
var client = new AmazonDynamoDBClient();

// Store item
client.PutItem(new PutItemRequest
{
    TableName = "Books",
    Item = new Dictionary<string, AttributeValue>
    {
        { "Title", new AttributeValue { S = "Cryptonomicon" } },
        { "Id", new AttributeValue { N = "42" } },
        { "Authors", new AttributeValue {
            SS = new List<string> { "Neal Stephenson" } } },
        { "Price", new AttributeValue { N = "12.95" } }
    }
});

// Get item
Dictionary<string, AttributeValue> book = client.GetItem(new GetItemRequest
{
    TableName = "Books",
    Key = new Dictionary<string, AttributeValue>
    {
        { "Id", new AttributeValue { N = "42" } }
    }
}).Item;

Console.WriteLine("Id = {0}", book["Id"].S);
Console.WriteLine("Title = {0}", book["Title"].S);
Console.WriteLine("Authors = {0}",
    string.Join(", ", book["Authors"].SS));
```

Document Model Example

```
var client = new AmazonDynamoDBClient();
Table booksTable = Table.LoadTable(client, "Books");

// Store item
Document book = new Document();
book["Title"] = "Cryptonomicon";
book["Id"] = 42;
book["Authors"] = new List<string> { "Neal Stephenson" };
book["Price"] = 12.95;
booksTable.PutItem(book);

// Get item
book = booksTable.GetItem(42);
Console.WriteLine("Id = {0}", book["Id"]);
Console.WriteLine("Title = {0}", book["Title"]);
Console.WriteLine("Authors = {0}",
    string.Join(", ", book["Authors"].AsListOfString()));
```

Object Persistence Model Example

This example consists of two parts: first, we must define our `Book` type; second, we use it with `DynamoDBContext`.

```
[DynamoDBTable("Books")]
class Book
{
    [DynamoDBHashKey]
    public int Id { get; set; }
    public string Title { get; set; }
    public List<string> Authors { get; set; }
    public double Price { get; set; }
}
```

Now, use it with `DynamoDBContext`.

```
var client = new AmazonDynamoDBClient();
DynamoDBContext context = new DynamoDBContext(client);

// Store item
Book book = new Book
{
    Title = "Cryptonomicon",
    Id = 42,
    Authors = new List<string> { "Neal Stephenson" },
    Price = 12.95
};
context.Save(book);

// Get item
book = context.Load<Book>(42);
Console.WriteLine("Id = {0}", book.Id);
Console.WriteLine("Title = {0}", book.Title);
Console.WriteLine("Authors = {0}", string.Join(", ", book.Authors));
```

Read Using AWS DynamoDb with the AWS .NET SDK online: https://riptutorial.com/amazon-dynamodb/topic/5275/using-aws-dynamodb-with-the-aws--net-sdk

Credits

S. No	Chapters	Contributors
1	Getting started with amazon-dynamodb	Community, Gustavo Tavares
2	Batch Operations: Things to know	SACHESH A C
3	Dynamodb delete data over time	Lior
4	How to create a DynamoDB Table	Mike Dinescu, Parth Modi
5	How to insert data into table using DynamoDb?	Franck Dernoncourt
6	Using AWS DynamoDb with the AWS .NET SDK	David Hale