**DynamoDB**

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed cloud database and supports both document and key-value store models.

**Changes to be done:**

Take a spring boot application with maven and make the following changes to connect that to DynamoDB.

**In pom.xml:**

Add the two maven dependencies

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.amazonaws</groupId>

<artifactId>aws-java-sdk-dynamodb</artifactId>

<version>1.11.408</version>

</dependency>

</dependencies>

**In application.yaml:**

amazon:

access:

key: A\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*J

secret-key: 0\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*F

region: ap-south-1

end-point:

url: dynamodb.ap-south-1.amazonaws.com

server:

port: 9001

**Create a DynamoDbConfig class:**

@Configuration

**public** **class** DynamoDbConfig {

@Value("${amazon.access.key}")

**private** String awsAccessKey;

@Value("${amazon.access.secret-key}")

**private** String awsSecretKey;

@Value("${amazon.region}")

**private** String awsRegion;

@Value("${amazon.end-point.url}")

**private** String awsDynamoDBEndPoint;

@Bean

**public** DynamoDBMapper mapper() {

**return** **new** DynamoDBMapper(amazonDynamoDBConfig());

}

**public** AmazonDynamoDB amazonDynamoDBConfig() {

**return** AmazonDynamoDBClientBuilder.*standard*()

.withEndpointConfiguration(**new** AwsClientBuilder.

EndpointConfiguration(awsDynamoDBEndPoint, awsRegion))

.withCredentials(**new** AWSStaticCredentialsProvider(**new** BasicAWSCredentials(awsAccessKey, awsSecretKey))).build();

}

}

**Model Class:**

Created two model classes Student and Address, to understand different annotations available in DynamoDB.

**Student class:**

@DynamoDBTable(tableName = "student")

**public** **class** Student **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String studentId;

**private** String firstName;

**private** String lastName;

**private** String age;

**private** Address address;

@DynamoDBHashKey(attributeName = "studentId")

@DynamoDBAutoGeneratedKey

**public** String getStudentId() {

**return** studentId;

}

**public** **void** setStudentId(String studentId) {

**this**.studentId = studentId;

}

@DynamoDBRangeKey

**public** String getLastName() {

**return** lastName;

}

**public** **void** setLastName(String lastName) {

**this**.lastName = lastName;

}

// .. getters & setters for remaining attributes

}

**Address class:**

@DynamoDBDocument

**public** **class** Address **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String addressLine1;

**private** String addressLine2;

**private** String state;

**private** String city;

**private** String zipCode;

@DynamoDBAttribute

**public** String getAddressLine1() {

**return** addressLine1;

}

**public** **void** setAddressLine1(String addressLine1) {

**this**.addressLine1 = addressLine1;

}

// .. getters & setters for remaining attributes

}

**In Controller:**

@RestController

@RequestMapping("/dynamoDb")

**public** **class** DynamoDbController {

@Autowired

**private** DynamoDbRepository repository;

@PostMapping

**public** String insertIntoDynamoDB(@RequestBody Student student) {

repository.insertIntoDynamoDB(student);

**return** "Successfully inserted into DynamoDB table";

}

//If required add code for other http methods as well.

//@GetMapping

//@PutMapping

//@DeleteMapping(value = "{studentId}/{lastName}")

}

**In Repository:**

@Repository

**public** **class** DynamoDbRepository {

**private** **static** **final** Logger ***LOGGER***= LoggerFactory.*getLogger*(DynamoDbRepository.**class**);

@Autowired

**private** DynamoDBMapper mapper;

**public** **void** insertIntoDynamoDB(Student student) {

mapper.save(student);

}

**public** Student getOneStudentDetails(String studentId, String lastName) {

**return** mapper.load(Student.**class**, studentId, lastName);

}

**public** **void** updateStudentDetails(Student student) {

**try** {

mapper.save(student, buildDynamoDBSaveExpression(student));

} **catch** (ConditionalCheckFailedException exception) {

***LOGGER***.error("invalid data - " + exception.getMessage());

}

}

**public** **void** deleteStudentDetails(Student student) {

mapper.delete(student); }

**public** DynamoDBSaveExpression buildDynamoDBSaveExpression(Student student){

DynamoDBSaveExpression saveExpression = **new**DynamoDBSaveExpression();

Map<String, ExpectedAttributeValue> expected = **new** HashMap<>();

expected.put("studentId", **new** ExpectedAttributeValue(**new** AttributeValue(student.getStudentId()))

.withComparisonOperator(ComparisonOperator.***EQ***));

saveExpression.setExpected(expected);

**return** saveExpression;

}

}

**Accessing DynamoDB:**

We can use DynbamoDB using local instance or using AWS DynamoDB console.

**Using Local instance:**

1. First download DynamoDB instance from Official website.

<https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBLocal.DownloadingAndRunning.html>

Default port is 8000, You can change if you want.

1. Select the specific region you want and remember the region code.

Example: For Mumbai aws region – code is **ap-south-1**

1. Download the archive, extract the contents and copy the extracted directory to a location of your choice.
2. Now open a command prompt window, navigate to the directory where you extracted DynamoDBLocal.jar, and enter the following command:

**java -Djava.library.path=./DynamoDBLocal\_lib -jar DynamoDBLocal.jar -sharedDb**

1. Download AWS CLI from official website to enable authorization to applications. If you have credentials, give them using following command:

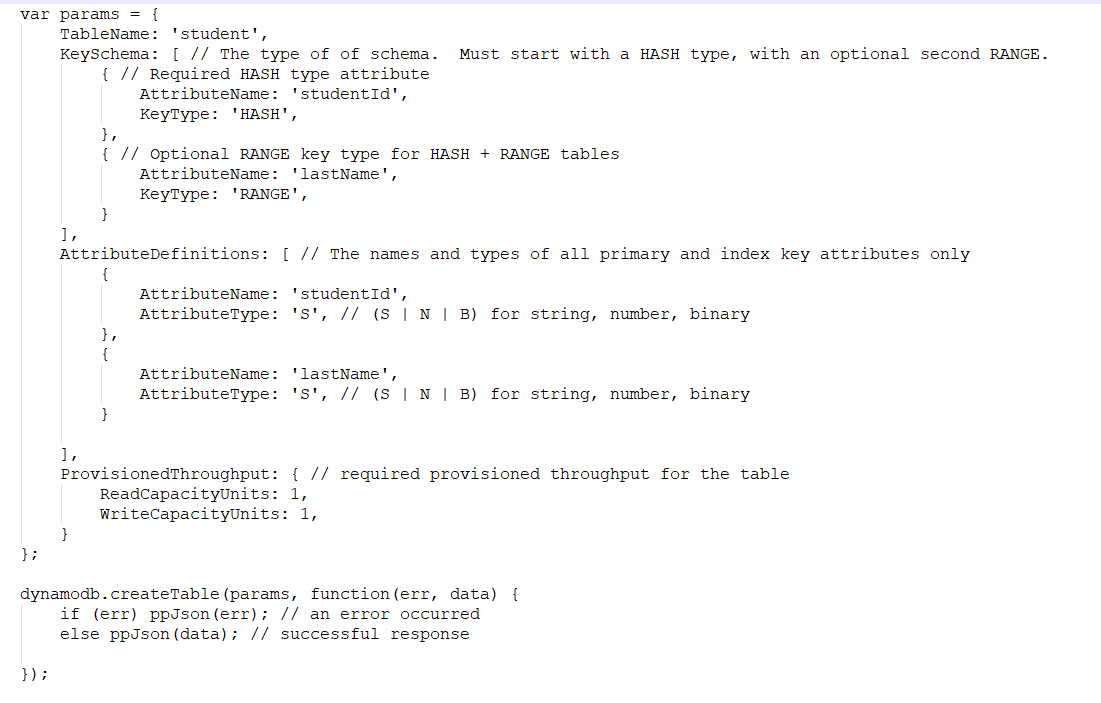
**aws configure**

If you don’t have credentials, you can give fake credentials for local usage of DynamoDB, but region-code must be specific.

1. After configuring, In AWS CLI to check tables in DynamoDB

**aws dynamodb list-tables --endpoint-url** [**http://localhost:8000**](http://localhost:8000)

1. Open <https://localhost:8000/shell/> to open DynamoDB Javasript Dashboard. Here you can create tables and use the DynamoDB using Javascript following the specific syntax.
2. Create table using JS as:



1. Now run the application and check the status of application using REST client by giving the required headers and Body for specified http methods.

**Using AWS DynamoDB Management Console:**

1. Take AWS subscription as Root user giving required data and do the required verifications as well.
2. Then login and check for DynamoDB service.
3. For admin access over all the required services, go to IAM service.
4. There create a group and select Administrator policy for that.
5. Then add a user to that group, you will get access key and secret key which are unique and confidential.
6. Then again go to DynamoDB service and create a table with specific name, partition key and sort key if required.
7. Add the Access key & secret key details in **application.yaml**
8. Now run the application and check the status of application using REST client by giving the required headers and Body for specified http methods.
9. Now check the data in table using console. You can find Item inserted into table.Item contains the inserted data.