Working with Tables: .NET

Topics

You can use the AWS SDK for .NET to create, update, and delete tables, list all the tables in your account, or get information about a specific table.

The following are the common steps for table operations using the AWS SDK for .NET.

1. Create an instance of the `AmazonDynamoDBClient` class (the client).

2. Provide the required and optional parameters for the operation by creating the corresponding request objects.

   For example, create a `CreateTableRequest` object to create a table and `UpdateTableRequest` object to update an existing table.

3. Execute the appropriate method provided by the client that you created in the preceding step.

   Note

   The examples in this section do not work with .NET core as it does not support synchronous methods. For more information, see AWS Asynchronous APIs for .NET (http://docs.aws.amazon.com/sdk-for-net/v3/developer-guide/sdk-net-async-api.html) .

Creating a Table

To create a table, you must provide the table name, its primary key, and the provisioned throughput values.

The following are the steps to create a table using the .NET low-level API.

1. Create an instance of the `AmazonDynamoDBClient` class.

2. Create an instance of the `CreateTableRequest` class to provide the request information.

   You must provide the table name, primary key, and the provisioned throughput values.

3. Execute the `AmazonDynamoDBClient.CreateTable` method by providing the request object as a parameter.

The following C# code snippet demonstrates the preceding steps. The sample creates a table (ProductCatalog) that uses Id as the primary key and set of provisioned throughput values. Depending on your application requirements, you can update the provisioned throughput values by using the `UpdateTable` API.

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ProductCatalog";

var request = new CreateTableRequest
{
  TableName = tableName,
  AttributeDefinitions = new List<AttributeDefinition>()
  {
    new AttributeDefinition
    {
      AttributeName = "Id",
      AttributeType = "N"
    }
  },
  KeySchema = new List<KeySchemaElement>()
  {
    new KeySchemaElement
    {
      AttributeName = "Id",
      KeyType = "HASH"  //Partition key
    }
  },
  ProvisionedThroughput = new ProvisionedThroughput
  {
    ReadCapacityUnits = 10,
    WriteCapacityUnits = 5
  }
};

var response = client.CreateTable(request);
```

You must wait until DynamoDB creates the table and sets the table status to `ACTIVE`. The `CreateTable` response includes the `TableDescription` property that provides the necessary table information.

Example

```
var result = response.CreateTableResult;
 var tableDescription = result.TableDescription;
```

```
Console.WriteLine("{1}: {0} \t ReadCapacityUnits: {2} \t WriteCapacityUnits: {3}",
                  tableDescription.TableStatus,
                  tableDescription.TableName,
                  tableDescription.ProvisionedThroughput.ReadCapacityUnits,
                  tableDescription.ProvisionedThroughput.WriteCapacityUnits);

string status = tableDescription.TableStatus;
Console.WriteLine(tableName + " - " + status);
```

You can also call the `DescribeTable` method of the client to get table information at anytime.

Example

```
var res = client.DescribeTable(new DescribeTableRequest{TableName =
"ProductCatalog"});
```

Updating a Table

You can update only the provisioned throughput values of an existing table. Depending on you application requirements, you might need to update these values.

> **Note**
>
> You can increase throughput capacity as often as needed, and decrease it up to nine times per table in a single UTC calendar day. For more information, see Limits in DynamoDB (Limits.html) .

The following are the steps to update a table using the .NET low-level API.

1. Create an instance of the `AmazonDynamoDBClient` class.
2. Create an instance of the `UpdateTableRequest` class to provide the request information.

   You must provide the table name and the new provisioned throughput values.
3. Execute the `AmazonDynamoDBClient.UpdateTable` method by providing the request object as a parameter.

The following C# code snippet demonstrates the preceding steps.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ExampleTable";

var request = new UpdateTableRequest()
 {
```

```
    TableName = tableName,
    ProvisionedThroughput = new ProvisionedThroughput()
    {
      // Provide new values.
      ReadCapacityUnits = 20,
      WriteCapacityUnits = 10
    }
  };
var response = client.UpdateTable(request);
```

Deleting a Table

The following are the steps to delete a table using the .NET low-level API.

1. Create an instance of the `AmazonDynamoDBClient` class.
2. Create an instance of the `DeleteTableRequest` class and provide the table name that you want to delete.
3. Execute the `AmazonDynamoDBClient.DeleteTable` method by providing the request object as a parameter.

The following C# code snippet demonstrates the preceding steps.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();
string tableName = "ExampleTable";

var request = new DeleteTableRequest{ TableName = tableName };
var response = client.DeleteTable(request);
```

Listing Tables

To list tables in your account using the AWS SDK for .NET low-level API, create an instance of the `AmazonDynamoDBClient` and execute the `ListTables` method. The ListTables (http://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_ListTables.html) operation requires no parameters. However, you can specify optional parameters. For example, you can set the `Limit` parameter if you want to use paging to limit the number of table names per page. This requires you to create a `ListTablesRequest` object and provide optional parameters as shown in the following C# code snippet. Along with the page size, the request sets the `ExclusiveStartTableName` parameter. Initially, `ExclusiveStartTableName` is null, however, after fetching the first page of result, to retrieve the next page of result, you must set this parameter value to the `LastEvaluatedTableName` property of the current result.

Example

```
AmazonDynamoDBClient client = new AmazonDynamoDBClient();

// Initial value for the first page of table names.
string lastEvaluatedTableName = null;
do
{
   // Create a request object to specify optional parameters.
   var request = new ListTablesRequest
   {
     Limit = 10, // Page size.
     ExclusiveStartTableName = lastEvaluatedTableName
   };

   var response = client.ListTables(request);
   ListTablesResult result = response.ListTablesResult;
   foreach (string name in result.TableNames)
     Console.WriteLine(name);

   lastEvaluatedTableName = result.LastEvaluatedTableName;

} while (lastEvaluatedTableName != null);
```