



lab title

# Programming AWS DynamoDB using the AWS NodeJS SDK V1.02



Course title

**AWS Certified Associate**



# Table of Contents

## *Contents*

Table of Contents.....	1
About the Lab .....	1
Creating a DynamoDB Table using the Console .....	1
Importing Items into DynamoDB using batchWriteItem .....	1
Querying DynamoDB Tables using the NodeJS SDK .....	1





## About the Lab

These lab notes are to support the instructional videos on Programming Amazon DynamoDB using the AWS NodeJS SDK in the BackSpace AWS Certified Developer course.

We will first create a DynamoDB table using the console and then add items to the table.

We will then:

- Connect to DynamoDB through our NodeJS EC2 instance.
- Upload a JSON file containing items using the SDK batchWriteItem method.
- Query the data using the SDK.

Please refer to the AWS JavaScript SDK documentation at:

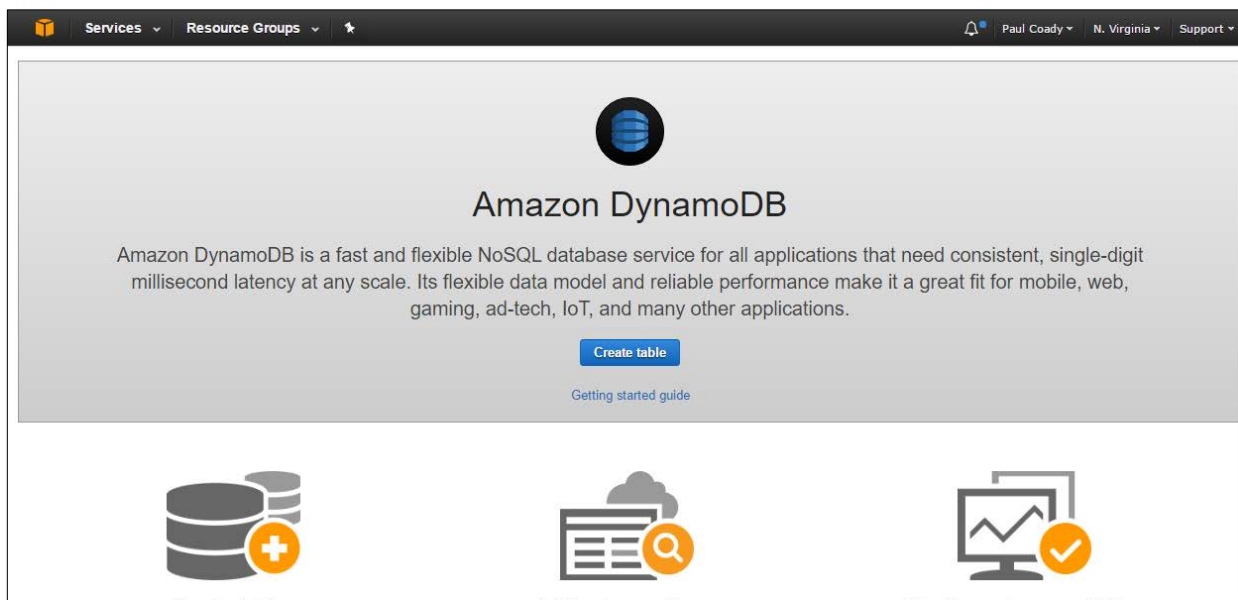
<http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html>

**Please note that AWS services change on a weekly basis and it is extremely important you check the version number on this document to ensure you have the latest version with any updates or corrections.**

# ▶ Creating a DynamoDB Table using the Console

In this section we will use the **DynamoDB console** to create a table and then add items individually using the console.

Select the DynamoDB Console



Click "Create Table"

Enter the following details (enter exactly with correct case)

**BE CAREFUL IF USING COPY/PASTE NOT TO INCLUDE ANY EXTRA SPACES ON THE END.**

Table Name: test-table

Primary Key Type: hash

Hash Attribute Type: Number

Hash Attribute Name: Id (case sensitive - make sure the first letter is capitalised)

Table name\*  ⓘ

Primary key\* Partition key

ⓘ

☐ Add sort key

Uncheck *Use Default Settings*

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

☐ Use default settings

Now create a global secondary index with hash key string ProductCategory and sort key number Price.

Use index name ProductCategory-Price-index

Click Add index to table.

Secondary indexes

Name	Type	Partition key	Sort key	Projected Attributes
+ Add index				

Enter index details

Add index

Primary key\* Partition key

ⓘ

☒ Add sort key

ⓘ

Index name\*  ⓘ

Projected attributes  ⓘ

☐ Create as Local Secondary Index ⓘ

Cancel Add index

Click *Add Index*

Continue using default settings.

Provisioned capacity

	Read capacity units	Write capacity units
Table	<input type="text" value="5"/>	<input type="text" value="5"/>
ProductCategory--Price-index	<input type="text" value="5"/>	<input type="text" value="5"/>

Estimated cost \$5.81 / month ([Capacity calculator](#))

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

Cancel Create

Click Create.

Press refresh until table status is listed as active.

Table details	
Table name	test-table
Primary partition key	Id (Number)
Primary sort key	-
Time to live attribute	DISABLED <a href="#">Manage TTL</a>
Table status	Active
Creation date	May 12, 2017 at 2:19:58 AM UTC+10
Provisioned read capacity units	5
Provisioned write capacity units	5
Last decrease time	-
Last increase time	-
Storage size (in bytes)	0 bytes
Item count	0
Region	US East (N. Virginia)
Amazon Resource Name (ARN)	arn:aws:dynamodb:us-east-1:802694931986:table/test-table

Storage size and item count are not updated in real-time. They are updated periodically, roughly every six hours.

Click on Items tab

Click on Create Item

**BE CAREFUL IF USING COPY/PASTE NOT TO INCLUDE ANY EXTRA SPACES ON THE END.**

Enter the Id as 101

ProductCategory- String: Book

Price - Number:-2

and then click on the action menus box on the left of the entry.

Select Append then String

Enter field *Title* and value *Book 101 Title*

Enter the rest of the details for the item. Make sure you select the right data type of string or number or boolean:



InPublication - Boolean:true

PageCount - Number:500

Dimensions - String: 8.5 x 11.0 x 0.5

Authors - String: Author 1

ISBN- String: 111-1111111111

```
▼ Item {9}
  Id Number : 101
  ProductCategory String : Book
  Price Number : 2
  Title String : Book 101 Title
  InPublication Boolean : true
  PageCount Number : 500
  Dimensions String : 8.5 x 11.0 x 0.5
  Author String : Author 1
  ISBN String : 111-1111111111
```

Click Save

test-table [Close](#)

Overview **Items** Metrics Alarms Capacity Indexes Triggers Access control Tags

Create item Actions ▼

Scan: [Table] test-table: Id ▲

Scan ▼ [Table] test-table: Id ▲

⊕ Add filter

Start search

	Id	Author	Dimensions	ISBN	InPublication	PageCount	Price	ProductCategory	Title
<input type="checkbox"/>	101	Author 1	8.5 x 11.0 x 0.5	111-1111111111	true	500	2	Book	Book 101 Title

# ▶ Importing Items into DynamoDB using batchWriteItem

In this section we will use our NodeJS EC2 instance to import items from a JSON file into a DynamoDB table.

Make sure you have set up your NodeJS development environment as detailed in the introduction lab.

Open Atom IDE.

Go to Packages – Remote Edit – Browse Hosts

Select your EC2 instance

Wait for the EC2 instance to connect then select the index.js file



Replace it with the following code:

```
// Load the AWS SDK for Node.js
var AWS = require('aws-sdk');

/**
 * Don't hard-code your credentials!
 * Create an IAM role for your EC2 instance instead.
 */

// Set your region
AWS.config.region = 'us-east-1';

var db = new AWS.DynamoDB();
db.listTables(function(err, data) {
  console.log(data.TableNames);
});
```

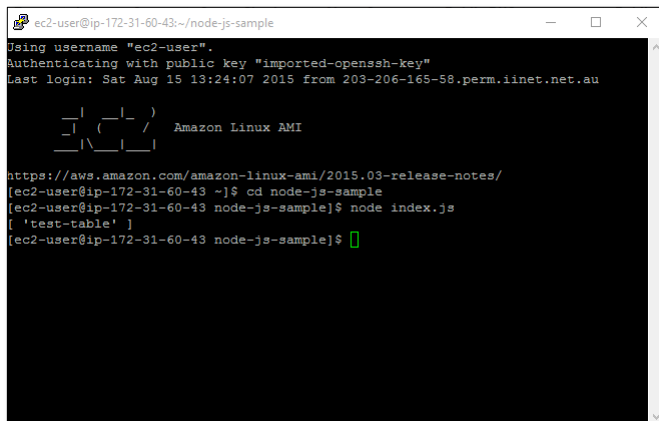
Click  +  to save to the EC2 instance.

Now open Putty and connect to your instance

Change to the node-js-sample directory

Run node index.js

You will see the results of the db.listTables method showing our test-table



```

ec2-user@ip-172-31-60-43: ~/node-js-sample
Using username "ec2-user".
Authenticating with public key "imported-openssh-key"
Last login: Sat Aug 15 13:24:07 2015 from 203-206-165-58.perm.iinet.net.au

 _ _ | _ _ |
 _ | ( _ _ | /   Amazon Linux AMI
 _ | \ _ _ | _ |

https://aws.amazon.com/amazon-linux-ami/2015.03-release-notes/
[ec2-user@ip-172-31-60-43 ~]$ cd node-js-sample
[ec2-user@ip-172-31-60-43 node-js-sample]$ node index.js
[ 'test-table' ]
[ec2-user@ip-172-31-60-43 node-js-sample]$

```

Download the following file:

<http://cdn.backspace.academy/public/classroom/aws-csa-a/test-table-items.json>

Go to the S3 console and create a folder in your bucket called lab-data

Upload the file to the lab-data folder

Now go back to Atom IDE

Change our listTables call and add a downloadData function call in our callback:

```

db.listTables(function(err, data) {
  console.log(data.TableNames);
  downloadData();
});

```

Add the downloadData function (remember to change YOUR\_BUCKET\_NAME):

```

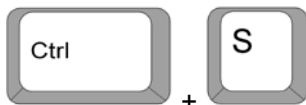
function downloadData(){
  // Get JSON file from S3
  var s3 = new AWS.S3();

```

```

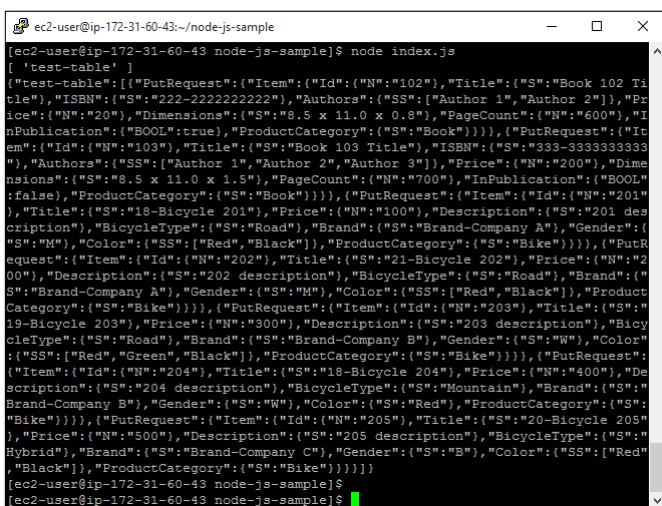
var params = {Bucket: 'YOUR_BUCKET_NAME', Key: 'lab-data/test-table-items.json'};
s3.getObject(params, function(error, data) {
  if (error) {
    console.log(error); // error is Response.error
  } else {
    var dataJSON = JSON.parse(data.Body);
    console.log(JSON.stringify(dataJSON));
  }
});
};

```



Click  +  to save to the EC2 instance.

Run your application again and you will see the contents of the JSON file output to the console.



```

ec2-user@ip-172-31-60-43:~/node-js-sample$ node index.js
[ 'test-table' ]
{"test-table":[{"PutRequest":{"Item":{"Id":{"N":"102"},"Title":{"S":"Book 102 Title"},"ISBN":{"S":"222-2222222222"},"Authors":{"SS":["Author 1","Author 2"]},"Price":{"N":"20"},"Dimensions":{"S":"8.5 x 11.0 x 0.8"},"PageCount":{"N":"600"},"InPublication":{"BOOL":true},"ProductCategory":{"S":"Book"}}},"PutRequest":{"Item":{"Id":{"N":"103"},"Title":{"S":"Book 103 Title"},"ISBN":{"S":"333-3333333333"},"Authors":{"SS":["Author 1","Author 2","Author 3"]},"Price":{"N":"200"},"Dimensions":{"S":"8.5 x 11.0 x 1.5"},"PageCount":{"N":"700"},"InPublication":{"BOOL":false},"ProductCategory":{"S":"Book"}}},"PutRequest":{"Item":{"Id":{"N":"201"},"Title":{"S":"18-Bicycle 201"},"Price":{"N":"100"},"Description":{"S":"201 description"},"BicycleType":{"S":"Road"},"Brand":{"S":"Brand-Company A"},"Gender":{"S":"M"},"Color":{"SS":["Red","Black"]},"ProductCategory":{"S":"Bike"}}},"PutRequest":{"Item":{"Id":{"N":"202"},"Title":{"S":"21-Bicycle 202"},"Price":{"N":"200"},"Description":{"S":"202 description"},"BicycleType":{"S":"Road"},"Brand":{"S":"Brand-Company A"},"Gender":{"S":"M"},"Color":{"SS":["Red","Black"]},"ProductCategory":{"S":"Bike"}}},"PutRequest":{"Item":{"Id":{"N":"203"},"Title":{"S":"19-Bicycle 203"},"Price":{"N":"300"},"Description":{"S":"203 description"},"BicycleType":{"S":"Road"},"Brand":{"S":"Brand-Company B"},"Gender":{"S":"W"},"Color":{"SS":["Red","Green","Black"]},"ProductCategory":{"S":"Bike"}}},"PutRequest":{"Item":{"Id":{"N":"204"},"Title":{"S":"18-Bicycle 204"},"Price":{"N":"400"},"Description":{"S":"204 description"},"BicycleType":{"S":"Mountain"},"Brand":{"S":"Brand-Company B"},"Gender":{"S":"W"},"Color":{"S":"Red"},"ProductCategory":{"S":"Bike"}}},"PutRequest":{"Item":{"Id":{"N":"205"},"Title":{"S":"20-Bicycle 205"},"Price":{"N":"500"},"Description":{"S":"205 description"},"BicycleType":{"S":"Hybrid"},"Brand":{"S":"Brand-Company C"},"Gender":{"S":"B"},"Color":{"SS":["Red","Black"]},"ProductCategory":{"S":"Bike"}}}]}]
ec2-user@ip-172-31-60-43 node-js-sample$

```

Now we will add these items to our DynamoDB database.

Change our downloadData function and add a writeDynamoDB function call in our callback:

```

function downloadData(){
  // Get JSON file from S3
  var s3 = new AWS.S3();
  var params = {Bucket: 'YOUR_BUCKET_NAME', Key: 'lab-data/test-table-items.json'};
  s3.getObject(params, function(error, data) {
    if (error) {
      console.log(error); // error is Response.error
    } else {
      var dataJSON = JSON.parse(data.Body);

```

```

        console.log(JSON.stringify(dataJSON));
        writeDynamoDB(dataJSON);
    }
});
}

```

Now add the writeDynamoDB function:

```

function writeDynamoDB(dataJSON){
    // Write items from object to DynamoDB
    var params = { RequestItems: dataJSON };
    db.batchWriteItem(params, function(err, data) {
        if (err) console.log(err, err.stack); // an error occurred
        else     console.log(data);           // successful response
    });
}

```



Click  +  to save to the EC2 instance.

Now run the app again.

You should get the output “UnprocessedItems: {}” meaning no problems.

```

scription":{"S":"204 description"},"BicycleType":{"S":"Mountain"},"Brand":{"S":"
Brand-Company B"},"Gender":{"S":"W"},"Color":{"S":"Red"},"ProductCategory":{"S":
"Bike"}}},{ "PutRequest":{"Item":{"Id":{"N":"205"},"Title":{"S":"20-Bicycle 205"
},"Price":{"N":"500"},"Description":{"S":"205 description"},"BicycleType":{"S":
Hybrid"},"Brand":{"S":"Brand-Company C"},"Gender":{"S":"B"},"Color":{"SS":["Red
","Black"]},"ProductCategory":{"S":"Bike"}}}}]}
{ UnprocessedItems: {} }
[ec2-user@ip-172-31-60-43 node-js-sample]$

```

NOTE: If you get an error “*ValidationException: The provided key element does not match the schema*” this means there is a mismatch between the keys in the JSON file and the actual keys created in the database. Check the keys in the database are spelled correctly and the case is correct.

Now go to the DynamoDB console and view the added items:

Id	Author	Dimensions	ISBN	InPublication	PageCount	Price	ProductCategory	Title	BicycleType	Brand	Color	Description
205	Author 1	8.5 x 11.0 x 0.5	111-1111111111	true	500	500	Bike	20-Bicycle 205	Hybrid	Brand-Compa...	{ "Black", "Re...	205 descripti
203						300	Bike	19-Bicycle 203	Road	Brand-Compa...	{ "Black", "Gre...	203 descripti
202						200	Bike	21-Bicycle 202	Road	Brand-Compa...	{ "Black", "Re...	202 descripti
201						100	Bike	18-Bicycle 201	Road	Brand-Compa...	{ "Black", "Re...	201 descripti
204						400	Bike	18-Bicycle 204	Mountain	Brand-Compa...	Red	204 descripti
102	Author 1	8.5 x 11.0 x 0.8	222-2222222...	true	600	20	Book	Book 102 Title				
103		8.5 x 11.0 x 1.5	333-3333333...	false	700	200	Book	Book 103 Title				
101		8.5 x 11.0 x 0.5	111-1111111111	true	500	2	Book	Book 101 Title				

# ▶ Querying DynamoDB Tables using the NodeJS SDK

In this section we will use NodeJS SDK to query items in a DynamoDB table.

We will now use our Global Secondary Index to find all bikes \$300 or less.

Change writeDynamoDB in index.js to add queryDynamoDB() to the callback:

```
function writeDynamoDB(dataJSON){
  // Write items from object to DynamoDB
  console.log(JSON.stringify(dataJSON));
  var params = { RequestItems: dataJSON };
  db.batchWriteItem(params, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else{
      console.log(data);                // successful response
      queryDynamoDB();
    }
  });
}
```

Now add the queryDynamoDB () function:

```
function queryDynamoDB(){
  // Query DynamoDB table using JSON data
  var params = {
    TableName: 'test-table', /* required */
    IndexName: 'ProductCategory-Price-index',
    KeyConditions: {
      "ProductCategory": {
        "AttributeValueList": [{ "S": "Bike" }],
        "ComparisonOperator": "EQ"
      },
      "Price": {
        "AttributeValueList": [{ "N": "300" }],
        "ComparisonOperator": "LE"
      }
    }
  }
}
```

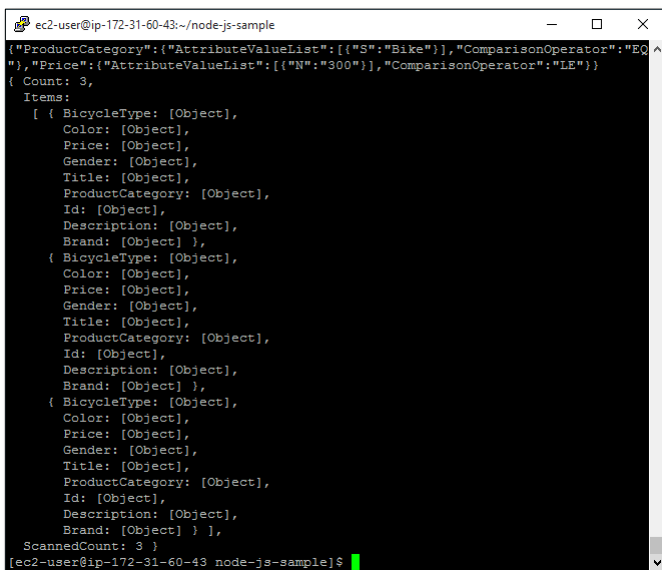
```

}
db.query(params, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else     console.log(data.Items);      // successful response
});
}

```

Run your application.

You will have the three bikes \$300 or less output to the console.



```

ec2-user@ip-172-31-60-43:~/node-js-sample
{"ProductCategory":{"AttributeValueList":[{"S":"Bike"}],"ComparisonOperator":"EQ"},
{"Price":{"AttributeValueList":[{"N":"300"}],"ComparisonOperator":"LE"}}
{ Count: 3,
  Items:
    [ { BicycleType: [Object],
        Color: [Object],
        Price: [Object],
        Gender: [Object],
        Title: [Object],
        ProductCategory: [Object],
        Id: [Object],
        Description: [Object],
        Brand: [Object] },
      { BicycleType: [Object],
        Color: [Object],
        Price: [Object],
        Gender: [Object],
        Title: [Object],
        ProductCategory: [Object],
        Id: [Object],
        Description: [Object],
        Brand: [Object] },
      { BicycleType: [Object],
        Color: [Object],
        Price: [Object],
        Gender: [Object],
        Title: [Object],
        ProductCategory: [Object],
        Id: [Object],
        Description: [Object],
        Brand: [Object] } ],
  ScannedCount: 3 }
[ec2-user@ip-172-31-60-43 ~]$

```

NOTE: If you get an error message such as *'Query condition missed key schema element: ProductCategory'* it means you have misspelled the index when creating the table. In this case a space is on the end of ProductCategory which caused an error.

We will now use another method, KeyConditionExpression to achieve the same thing.

Change queryDynamoDB to the following code:

```

function queryDynamoDB(){
    // Query DynamoDB table using JSON data
    var params = {
        TableName: 'test-table', /* required */
        IndexName: 'ProductCategory-Price-index',
        KeyConditionExpression: "ProductCategory = :prod_cat AND Price <= :price",
        ExpressionAttributeValues: {

```



```
        ":prod_cat": {"S": "Bike"},
        ":price": {"N": "300"}
    }
}
db.query(params, function(err, data) {
    if (err) console.log(err, err.stack); // an error occurred
    else     console.log(data.Items);      // successful response
});
}
```

Run your application.

You will have the three bikes \$300 or less output to the console.