



lab



lab title

# Programming Amazon DynamoDB using the AWS SDK

V1.05



Course title

AWS Certified Associate



# Table of Contents

## *Contents*

Table of Contents.....	1
About the Lab.....	2
Creating a DynamoDB Table using the Console.....	3
Importing Items into DynamoDB using NodeJS.....	9
Creating the Cloud 9 Environment .....	9
Querying DynamoDB Tables using the NodeJS SDK.....	18
Clean Up.....	20
Importing Items into DynamoDB using Python.....	23
Creating the Cloud 9 Environment .....	23
Creating the Python Code.....	24
Querying DynamoDB Tables using the Python SDK.....	32
Clean Up.....	34

## 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:

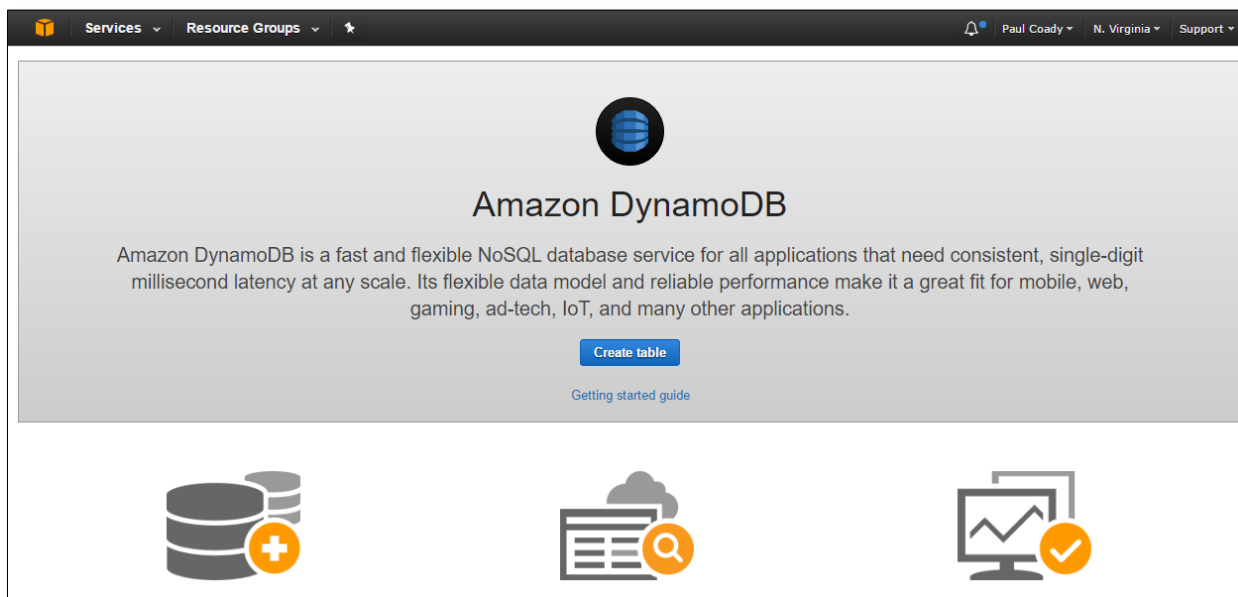
<http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html>

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

Primary Key Type: Number

### Create DynamoDB table

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

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 and sort key number Price.

Click *Add index to table*.

#### Secondary indexes

Name	Type	Partition key	Sort key	Projected Attributes
+ Add index				

Enter index details:

Partition key: ProductCategory

Type: String

Click *Add sort key*

Sort key: Price

Type: Number

Leave *Index name* as ProductCategory-Price-index

Click *Add index*

**Add index**

Primary key\* Partition key  
ProductCategory String

☒ Add sort key  
Price Number

Index name\* ProductCategory-Price-index

Projected attributes All

☐ Create as Local Secondary Index

Cancel Add index

Continue using default settings.

Provisioned capacity

	Read capacity units	Write capacity units
Table	5	5
ProductCategory-Price-index	5	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

Disable auto scaling

Auto Scaling

☐ Read capacity ☐ Write capacity

Leave encryption *DEFAULT*

Click Create.

### Encryption At Rest

Select Server-side encryption settings for your DynamoDB table to help protect data at rest. [Learn more](#)

☒ **DEFAULT**

The key is owned by Amazon DynamoDB. You are not charged any fee for using these CMKs.

☐ **KMS - Customer managed CMK**

The key is stored in your account that you create, own, and manage. AWS Key Management Service (KMS) charges apply. [Learn more](#)

☐ **KMS - AWS managed CMK**

The key is stored in your account and is managed by AWS Key Management Service (KMS). AWS KMS charges apply.

[+ Add tags](#) **NEW!**

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](#)

Press refresh until table status is listed as active.

test-table [Close](#)

[Overview](#)
[Items](#)
[Metrics](#)
[Alarms](#)
[Capacity](#)
[Indexes](#)
[Global Tables](#)
[Backups](#)
[Contributor Insights](#)
[More](#)

Recent alerts

No CloudWatch alarms have been triggered for this table.

Stream details

Stream enabled	No
View type	-
Latest stream ARN	-

[Manage Stream](#)

Table details [Refresh](#)

Table name	test-table
Primary partition key	Id (Number)
Primary sort key	-
Point-in-time recovery	DISABLED <a href="#">Enable</a>
Encryption Type	DEFAULT <a href="#">Manage Encryption</a>
KMS Master Key ARN	Not Applicable
Encryption Status	-
CloudWatch Contributor Insights	DISABLED <a href="#">Manage Contributor Insights</a> <b>NEW</b>
Time to live attribute	DISABLED <a href="#">Manage TTL</a>
Table status	Active
Creation date	July 24, 2020 at 4:10:18 AM UTC+10
Read/write capacity mode	Provisioned
Last change to on-demand mode	-
Provisioned read capacity units	5 (Auto Scaling Disabled)
Provisioned write capacity units	5 (Auto Scaling Disabled)
Last decrease time	-
Last increase time	-
Storage size (in bytes)	0 bytes

Click on Items tab




Click on Create Item

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

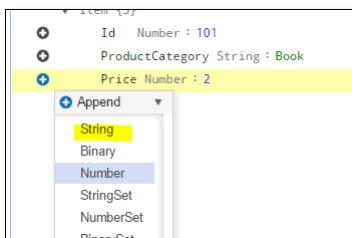
Enter the Id as 101

ProductCategory- String: Book

Price - Number: 2

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

Select Append then String to add another attribute



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, string set 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](#)

OverviewItemsMetricsAlarmsCapacityIndexesTriggersAccess controlTags

Create itemActions ▼

Scan: [Table] test-table: Id ▲

Scan ▼ [Table] test-table: Id ▼ ▲

⊕ Add filter

Start search

<input type="checkbox"/>	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 NodeJS

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

## Creating the Cloud 9 Environment

Go to Services - Cloud9 from the console

Click 'Create environment'

Give your environment a name

Click 'Next step'

Step 1  
**Name environment**

Step 2  
Configure settings

Step 3  
Review

### Name environment

**Environment name and description**

**Name**  
The name needs to be unique per user. You can update it at any time in your environment settings.

BackSpace DynamoDB lab

Limit: 60 characters

**Description - Optional**  
This will appear on your environment's card in your dashboard. You can update it at any time in your environment settings.

Write a short description for your environment

Limit: 200 characters

Cancel **Next step**

Leave default settings

Click 'Next step'

☒ **Create a new instance for environment (EC2)**  
Launch a new instance in this region to run your new environment.

☐ **Connect and run in remote server (SSH)**  
Display instructions to connect remotely over SSH and run your new environment.

**Instance type**

☒ **t2.micro (1 GiB RAM + 1 vCPU)**  
Free-tier eligible. Ideal for educational users and exploration.

☐ **t2.small (2 GiB RAM + 1 vCPU)**  
Recommended for small-sized web projects.

☐ **m4.large (8 GiB RAM + 2 vCPU)**  
Recommended for production and general-purpose development.

☐ **Other instance type**  
Select an instance type.  
t2.nano

**Cost-saving setting**  
Choose a predetermined amount of time to auto-hibernate your environment and prevent unnecessary charges. We recommend a hibernation settings of half an hour of no activity to maximize savings.  
After 30 minutes (default)

**IAM role**  
AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. [Learn more](#)

AWSServiceRoleForAWScloud9

► **Network settings (advanced)**

Cancel Previous step **Next step**

Click 'Create environment'

Turn on **AWS CloudTrail** in your AWS account to track activity in your environment. [Learn more](#)

- Only share your environment with **trusted users**. Sharing your environment may put your AWS access credentials at risk. [Learn more](#)

Cancel Previous step **Create environment**

Creating the NodeJS Code

When your environment is ready clone the code for the lab:

```
git clone https://github.com/backspace-academy/aws-dynamodb-nodejs
```

Open *index.js*

```
// 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.
 * For development an IAM role is not required for Cloud9
 */

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

var db = new AWS.DynamoDB();
db.listTables(function(err, data) {
  if (err) {
    console.log(err, err.stack); // an error occurred
  }
  else {
    console.log(data.TableNames); // successful response
  }
});
```

From the Bash console at the bottom of the screen enter:

```
cd aws-dynamodb-nodejs
npm install
```

```

npm - "ip-172-31-92- x"
pcoady:~/environment $ git clone https://github.com/backspace-academy/aws-dynamodb-nodejs
Cloning into 'aws-dynamodb-nodejs'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (9/9), done.
pcoady:~/environment $ cd aws-dynamodb-nodejs
pcoady:~/environment/aws-dynamodb-nodejs (master) $ npm install
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN aws-dynamodb-nodejs@1.0.0 No description
npm WARN aws-dynamodb-nodejs@1.0.0 No repository field.

added 14 packages from 66 contributors and audited 14 packages in 2.117s
found 0 vulnerabilities

pcoady:~/environment/aws-dynamodb-nodejs (master) $

```

Now that you have installed the AWS SDK you can run the app

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

```

npm - "ip-172-31-92- x" [New] - Idle x aws-dynamodb-nodejs x
Run Run Config Name Command: aws-dynamodb-nodejs/index.js
Debugger listening on ws://127.0.0.1:15454/18ca23c7-de40-415c-9862-7986fd427ece
For help, see: https://nodejs.org/en/docs/inspector
Debugger attached.
[ 'test-table' ]
Waiting for the debugger to disconnect...

Process exited with code: 0

```

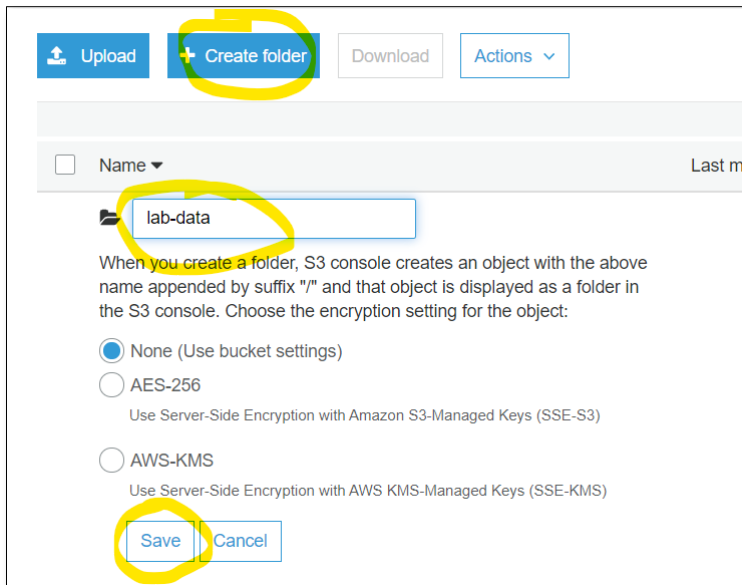
Download the following file:

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

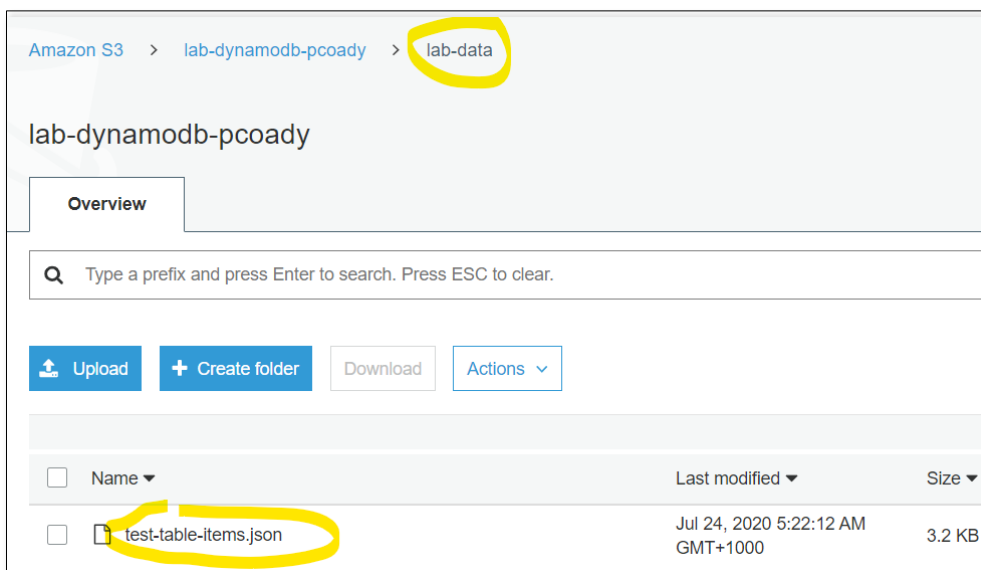
Go to the S3 console

Create a bucket. This bucket and folder can be private if using Cloud 9. If using an IDE remotely connected to EC2 will require an EC2 role for S3 access.

Create a folder in your bucket called */lab-data*



Upload the file to the lab-data folder



Now go back to Cloud 9 IDE

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

```

db.listTables(function(err, data) {
  if (err) {
    console.log(err, err.stack);    // an error occurred
  }
  else {
    console.log(data.TableNames);    // successful response
    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 the file.

Run your application again

You will see the contents of the JSON file output to the console.

Copyright 2017 all rights reserved - [BackSpace.Academy](#)



```
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)); // successful response
            writeDynamoDB(dataJSON);
        }
    });
};
```

Now add the *writeDynamoDB* function to import the data to DynamoDB:

```
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
        }
    });
}
```



Click  +  to save to the EC2 instance.

Now run the app again.

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

```

npm - "ip-172-31-92: x [New] - Idle x aws-dynamodb-node x
Run Run Config Name Command: aws-dynamodb-nodejs/index.js Runner: Node.js CWD ENV
":{"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"}}}}
{ UnprocessedItems: {} }
Waiting for the debugger to disconnect...

Process exited with code: 0

```

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 refresh the screen to view the added items:

<input type="checkbox"/>	Id	Authors	Dimensions	ISBN	InPublication	PageCount	Price
<input type="checkbox"/>	101	Author 1	8.5 x 11.0 x 0.5	111-1111111111	true	500	2
<input type="checkbox"/>	102	{ "Author 1", "Author 2" }	8.5 x 11.0 x 0.8	222-2222222222	true	600	20
<input type="checkbox"/>	103	{ "Author 1", "Author 2", "Author 3" }	8.5 x 11.0 x 1.5	333-3333333333	false	700	200
<input type="checkbox"/>	201						100
<input type="checkbox"/>	202						200
<input type="checkbox"/>	203						300
<input type="checkbox"/>	204						400
<input type="checkbox"/>	205						500

# ▶ 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 successful response:

```
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',
    KeyConditionExpression: 'ProductCategory = :c AND Price <= :p',
    ExpressionAttributeValues: {
      ':c': { "S": "Bike" },
      ':p': { "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.

```

npm - "ip-172-31-92- x" [New] - Idle x
Run Run Config Name Co
}, "Gender": {"S": "W"}, "Color": {"S": "Red"}, "Prod
}, "BicycleType": {"S": "Hybrid"}, "Brand": {"S": "t
{ UnprocessedItems: {} }
[ { Title: { S: '18-Bicycle 201' },
  Gender: { S: 'M' },
  Price: { N: '100' },
  Brand: { S: 'Brand-Company A' },
  ProductCategory: { S: 'Bike' },
  Description: { S: '201 description' },
  Color: { SS: [Array] },
  Id: { N: '201' },
  BicycleType: { S: 'Road' } },
{ Title: { S: '21-Bicycle 202' },
  Gender: { S: 'M' },
  Price: { N: '200' },
  Brand: { S: 'Brand-Company A' },
  ProductCategory: { S: 'Bike' },
  Description: { S: '202 description' },
  Color: { SS: [Array] },
  Id: { N: '202' },
  BicycleType: { S: 'Road' } },
{ Title: { S: '19-Bicycle 203' },
  Gender: { S: 'W' },
  Price: { N: '300' },
  Brand: { S: 'Brand-Company B' },
  ProductCategory: { S: 'Bike' },
  Description: { S: '203 description' },
  Color: { SS: [Array] },
  Id: { N: '203' },
  BicycleType: { S: 'Road' } } ]
Waiting for the debugger to disconnect...

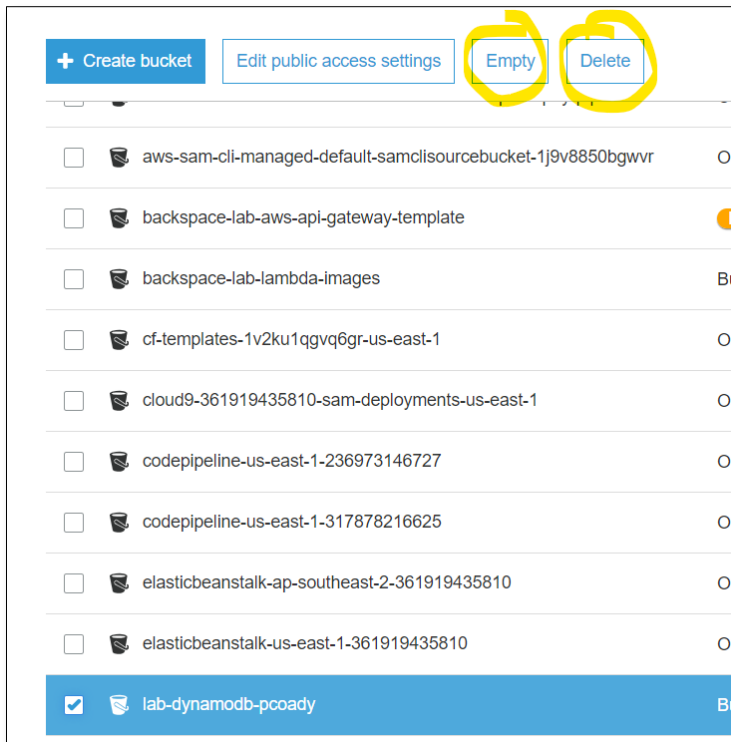
Process exited with code: 0

```

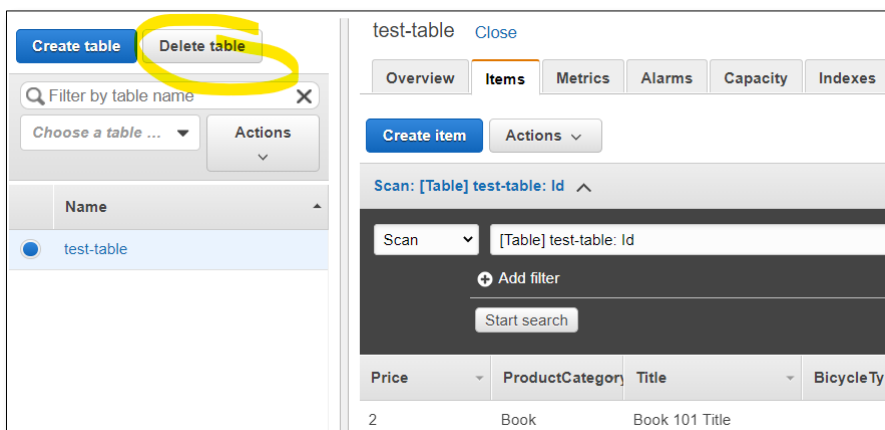
NOTE: If you get an error message such as *'Query condition missed key schema element: ProductCategory'* it means you have misspelled the index when creating the table. In this case a space is on the end of ProductCategory which caused an error.

## Clean Up

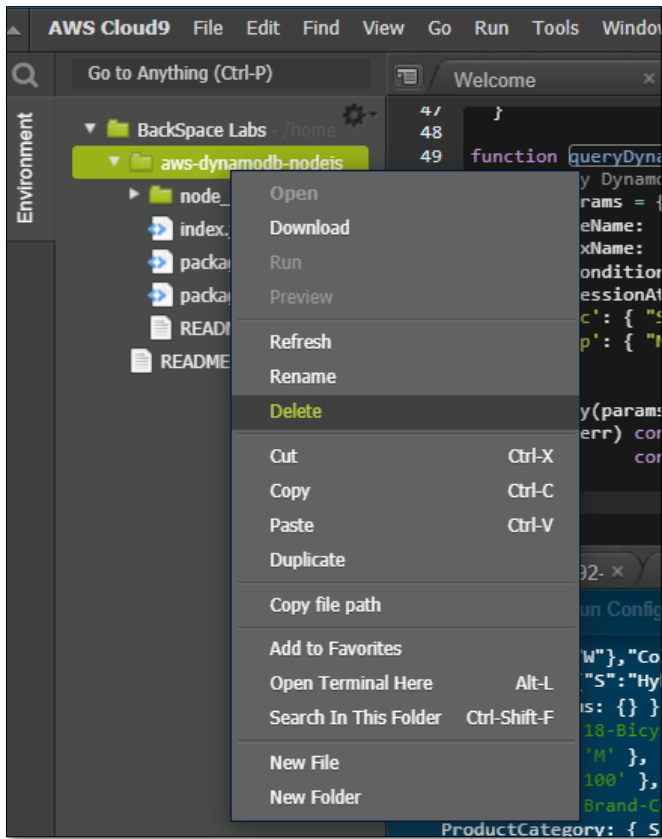
Go to the S3 console and empty the bucket, then delete it.



Go to the DynamoDB Console and delete the table



Go to the Cloud9 IDE and delete the repository



# ▶ Importing Items into DynamoDB using Python

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

## Creating the Cloud 9 Environment

Go to Services - Cloud9 from the console

Click 'Create environment'

Give your environment a name

Click 'Next step'

Step 1  
**Name environment**

Step 2  
Configure settings

Step 3  
Review

### Name environment

**Environment name and description**

**Name**  
The name needs to be unique per user. You can update it at any time in your environment settings.

BackSpace DynamoDB lab

Limit: 60 characters

**Description - Optional**  
This will appear on your environment's card in your dashboard. You can update it at any time in your environment settings.

Write a short description for your environment

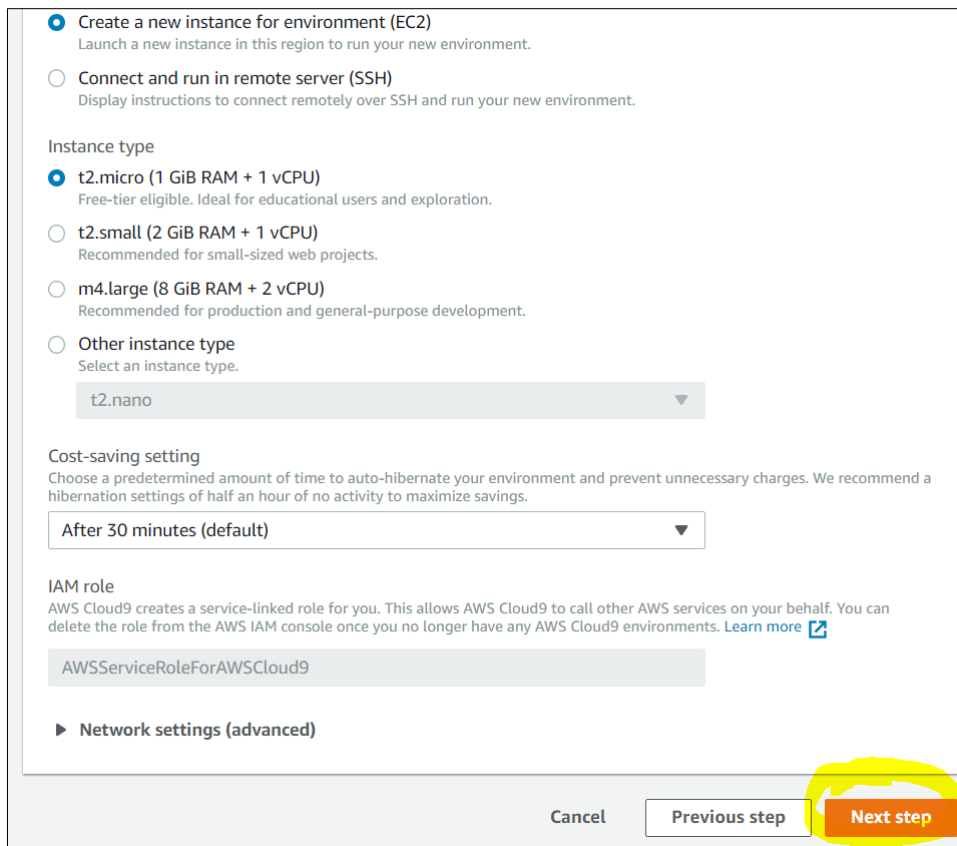
Limit: 200 characters

Cancel **Next step**

Leave default settings

Click 'Next step'





☒ **Create a new instance for environment (EC2)**  
Launch a new instance in this region to run your new environment.

☐ **Connect and run in remote server (SSH)**  
Display instructions to connect remotely over SSH and run your new environment.

**Instance type**

☒ **t2.micro (1 GiB RAM + 1 vCPU)**  
Free-tier eligible. Ideal for educational users and exploration.

☐ **t2.small (2 GiB RAM + 1 vCPU)**  
Recommended for small-sized web projects.

☐ **m4.large (8 GiB RAM + 2 vCPU)**  
Recommended for production and general-purpose development.

☐ **Other instance type**  
Select an instance type.  
t2.nano

**Cost-saving setting**  
Choose a predetermined amount of time to auto-hibernate your environment and prevent unnecessary charges. We recommend a hibernation settings of half an hour of no activity to maximize savings.  
After 30 minutes (default)

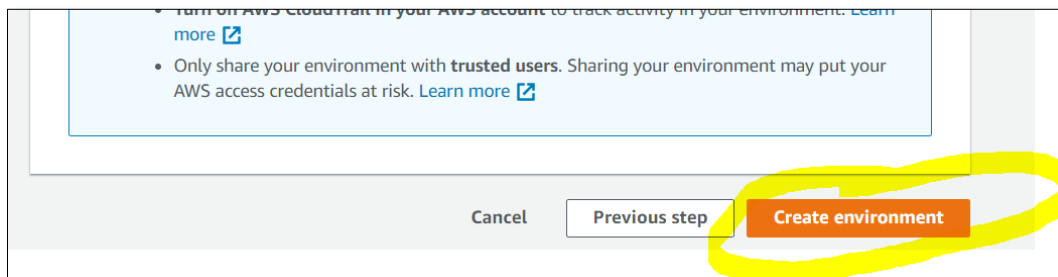
**IAM role**  
AWS Cloud9 creates a service-linked role for you. This allows AWS Cloud9 to call other AWS services on your behalf. You can delete the role from the AWS IAM console once you no longer have any AWS Cloud9 environments. [Learn more](#)

AWSServiceRoleForAWScloud9

► **Network settings (advanced)**

Cancel Previous step **Next step**

Click 'Create environment'



• Turn on **AWS CloudTrail** in your AWS account to track activity in your environment. [Learn more](#)

• Only share your environment with **trusted users**. Sharing your environment may put your AWS access credentials at risk. [Learn more](#)

Cancel Previous step **Create environment**

## Creating the Python Code

When your environment is ready clone the code for the lab:

```
git clone https://github.com/backspace-academy/aws-dynamodb-python
```

Open *lab.py*

```

# Load the AWS SDK for Python
import boto3

# Load the exceptions for error handling
from botocore.exceptions import ClientError, ParamValidationError

# JSON handling
import json

# Create AWS service client and set region
db = boto3.client('dynamodb', region_name='us-east-1')

# Get a list of tables in region
def get_tables():
    try:
        data = db.list_tables()
        return data['TableNames']
    # An error occurred
    except ParamValidationError as e:
        print("Parameter validation error: %s" % e)
    except ClientError as e:
        print("Client error: %s" % e)

# Main program
def main():
    table_names = get_tables()
    if (len(table_names)) == 0:
        print('No tables in region.')
    else:
        for x in table_names:
            print('Table name: ' + x )

if __name__ == '__main__':
    main()

```

Make sure Boto3 is installed:

```
pip show boto3
```

```

bash - "ip-172-31-92" x [New] - Idle x +
pcoady:~/environment $ git clone https://github.com/backspace-academy/aws-sqs-python
Cloning into 'aws-sqs-python'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 25 (delta 4), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (25/25), done.
pcoady:~/environment $ pip show boto3
Name: boto3
Version: 1.12.14
Summary: The AWS SDK for Python
Home-page: https://github.com/boto/boto3
Author: Amazon Web Services
Author-email: UNKNOWN
License: Apache License 2.0
Location: /usr/local/lib/python3.6/site-packages
Requires: s3transfer, jmespath, botocore
You are using pip version 9.0.3, however version 20.1.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
pcoady:~/environment $

```

Run the app

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

```

bash - "ip-172-31-92" x [New] - Idle x aws-sqs-python/sqs_ x +
Run Run Config Name Command: aws-sqs-python/sqs_example.py
Table name: test-table
Process exited with code: 0

```

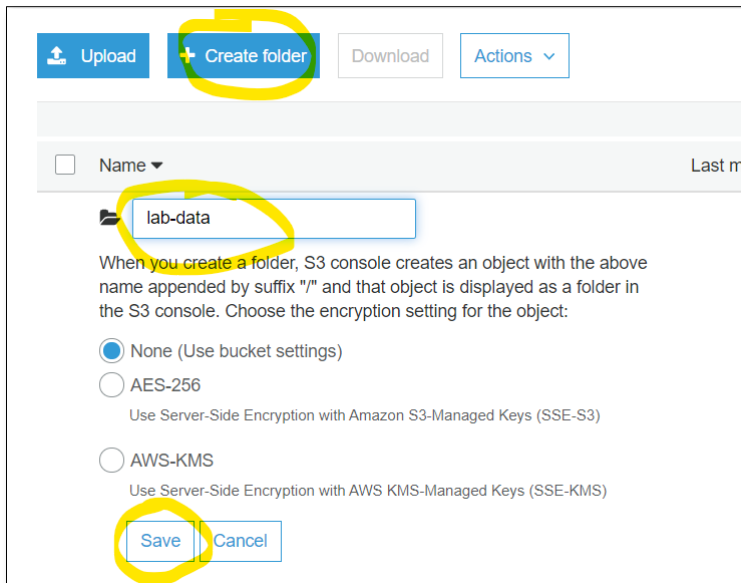
Download the following file:

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

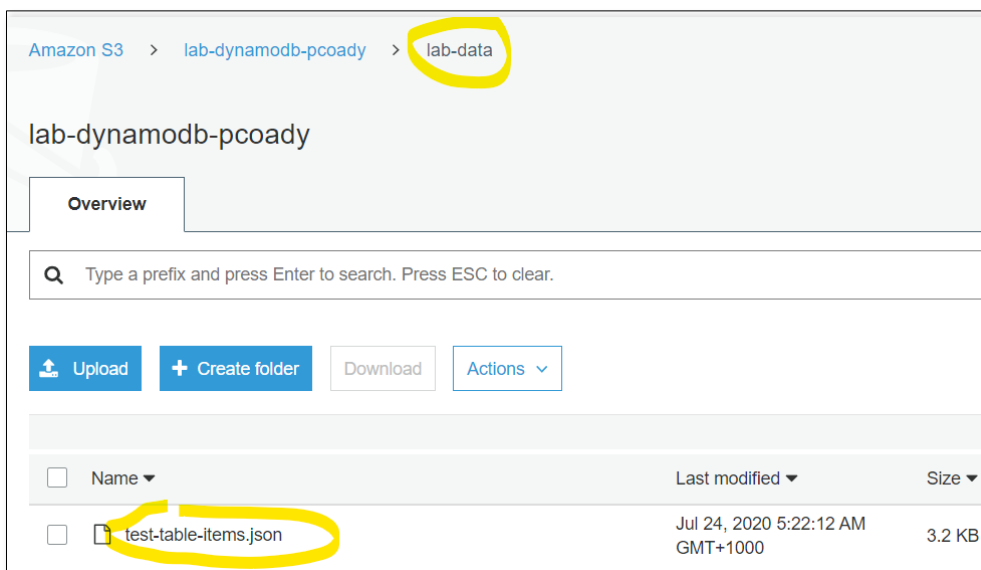
Go to the S3 console

Create a bucket. This bucket and folder can be private if using Cloud 9. If using an IDE remotely connected to EC2 will require an EC2 role for S3 access.

Create a folder in your bucket called *lab-data*



Upload the file to the lab-data folder



Now go back to Cloud 9 IDE

Create a call to a `download_data` function in our main program:

```
# Main program
def main():
    table_names = get_tables()
    if (len(table_names)) == 0:
        print('No tables in region.')
    else:
        for x in table_names:
            print('Table name: ' + x )
            y = download_data()

if __name__ == '__main__':
    main()
```

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

```
# Download JSON data from S3
s3 = boto3.client('s3', region_name='us-east-1')
def download_data():
    try:
        data_object = s3.get_object(
            Bucket='YOUR_BUCKET_NAME',
            Key='lab-data/test-table-items.json'
        )
        data_string = data_object['Body'].read().decode('utf-8')
        print('Downloaded from S3:')
        print(data_string)
        data = json.loads(data_string)
        return data
    # An error occurred
    except ParamValidationError as e:
        print("Parameter validation error: %s" % e)
    except ClientError as e:
        print("Client error: %s" % e)
```



Click  +  to save the file.

Run your application again

You will see the contents of the JSON file output to the console.

```

Command: aws-sqs-python/sqs_example.py

{
  "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"}
    }
  }
}
}

Process exited with code: 0
Pane is dead

```

Now we will add these items to our DynamoDB database.

Create a call to a `write_dynamo_db` function in our main program passing variable `y`:

```

# Main program
def main():
    table_names = get_tables()
    if (len(table_names)) == 0:
        print('No tables in region.')
    else:
        for x in table_names:
            print('Table name: ' + x )
        y = download_data()
        write_dynamo_db(y)
if __name__ == '__main__':
    main()

```

Now add the `write_dynamo_db` function to import the data to DynamoDB:

```
def write_dynamo_db(json_data):
    try:
        data = db.batch_write_item(
            RequestItems = json_data
        )
        print('UnprocessedItems: ')
        print(data['UnprocessedItems'])
        return data
    # An error occurred
    except ParamValidationError as e:
        print("Parameter validation error: %s" % e)
    except ClientError as e:
        print("Client error: %s" % e)
```



Click  +  to save to the EC2 instance.

Now run the app again.

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

```
bash - "ip-172-31-92" x [New] - Idle x aws-sqs-python/sqs_ x +
Run Run Config Name Command: aws-sqs-python/sqs_example.py

    "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:
{}

Process exited with code: 0
Pane is dead
```

If you get the following error:

ValidationException: The provided key element does not match the schema

You have entered the incorrect values when creating the table. Make sure no spaces when copying and pasting and, the correct data type (string, number etc.). You will have to delete and re-create the table.

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

<input type="checkbox"/>	Id ⓘ	Authors	Dimensions	ISBN	InPublication	PageCount	Price
<input type="checkbox"/>	101	Author 1	8.5 x 11.0 x 0.5	111-1111111111	true	500	2
<input type="checkbox"/>	102	{ "Author 1", "Author 2" }	8.5 x 11.0 x 0.8	222-2222222222	true	600	20
<input type="checkbox"/>	103	{ "Author 1", "Author 2", "Author 3" }	8.5 x 11.0 x 1.5	333-3333333333	false	700	200
<input type="checkbox"/>	201						100
<input type="checkbox"/>	202						200
<input type="checkbox"/>	203						300
<input type="checkbox"/>	204						400
<input type="checkbox"/>	205						500



# ▶ Querying DynamoDB Tables using the Python SDK

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

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

Create a call to a `query_dynamo_db` function in our main program:

```
# Main program
def main():
    table_names = get_tables()
    if (len(table_names)) == 0:
        print('No tables in region.')
    else:
        for x in table_names:
            print('Table name: ' + x )
            y = download_data()
            write_dynamo_db(y)
            query_dynamo_db()
if __name__ == '__main__':
    main()
```

Now add the `query_dynamo_db` function to import the data to DynamoDB:

Now add the `query_dynamo_db` function:

```
def query_dynamo_db():
    try:
        data = db.query(
            TableName='test-table',
            IndexName='ProductCategory-Price-index',
            KeyConditionExpression='ProductCategory = :c AND Price <= :p',
            ExpressionAttributeValues={
                ':c': { 'S': 'Bike' },
                ':p': { 'N': '300' }
            }
        )
        print('Matching Items:')
        for x in data['Items']:
            print(x)
    # An error occurred
    except ParamValidationError as e:
        print('Parameter validation error: %s' % e)
    except ClientError as e:
        print('Client error: %s' % e)
```

Run your application.

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

```
bash - lp-172-31-92 x [New] - Idle x aws-sqs-python/sqs x
Run Run Config Name Command: aws-sqs-python/sqs_example.py

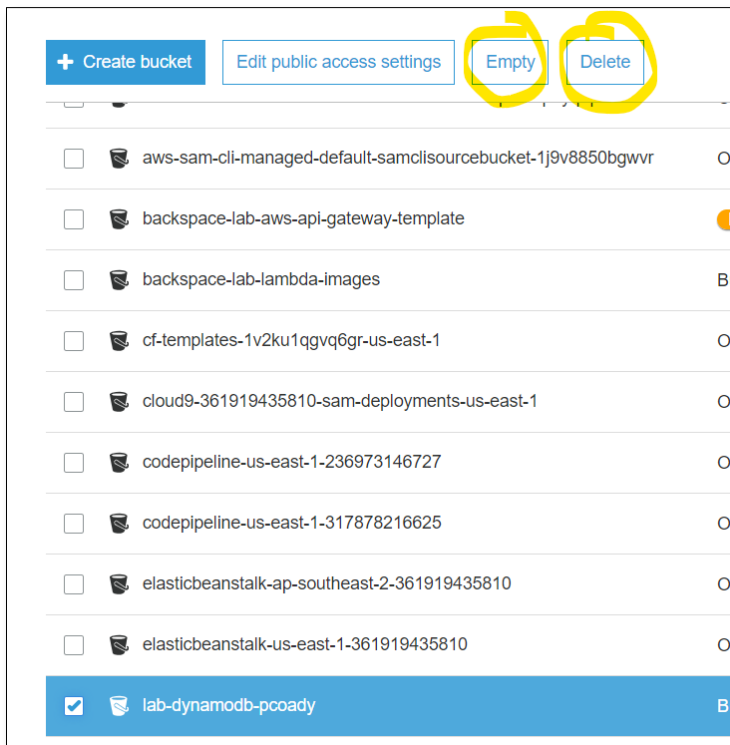
    }
    }
}
]
}
UnprocessedItems:
{}
Matching Items:
{'Title': {'S': '18-Bicycle 201'}, 'Gender': {'S': 'M'}, 'Price': {'N': '100'}, 'Brand': {'S': 'Brand-Company A'}, 'ProductCategory': {'S': 'Bike'}, 'Description': {'S': 'Black', 'Red'}, 'Id': {'N': '201'}, 'BicycleType': {'S': 'Road'}}
{'Title': {'S': '21-Bicycle 202'}, 'Gender': {'S': 'M'}, 'Price': {'N': '200'}, 'Brand': {'S': 'Brand-Company A'}, 'ProductCategory': {'S': 'Bike'}, 'Description': {'S': 'Black', 'Red'}, 'Id': {'N': '202'}, 'BicycleType': {'S': 'Road'}}
{'Title': {'S': '19-Bicycle 203'}, 'Gender': {'S': 'M'}, 'Price': {'N': '300'}, 'Brand': {'S': 'Brand-Company B'}, 'ProductCategory': {'S': 'Bike'}, 'Description': {'S': 'Black', 'Green', 'Red'}, 'Id': {'N': '203'}, 'BicycleType': {'S': 'Road'}}

Process exited with code: 0
Pane is dead
```

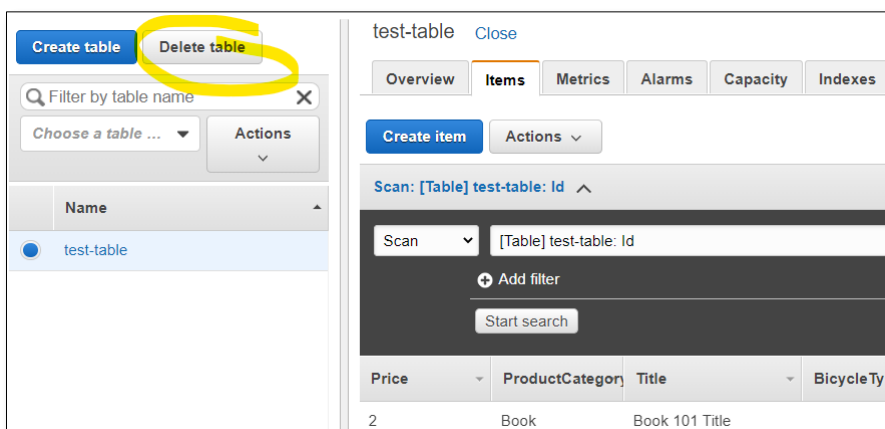
NOTE: If you get an error message such as *'Query condition missed key schema element: ProductCategory'* it means you have misspelled the index when creating the table. In this case a space is on the end of ProductCategory which caused an error.

## Clean Up

Go to the S3 console and empty the bucket, then delete it.



Go to the DynamoDB Console and delete the table



Go to the Cloud9 IDE and delete the repository

