aws CERTIFIED Developer Associate

aws CERTIFIED Solutions Architect Associate

aws CERTIFIED SysOps Administrator Associate

lab

lab title

Programming Amazon DynamoDB using the AWS SDK V1.05

Course title

AWS Certified Associate

BackSpace

Table of Contents

Contents

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:

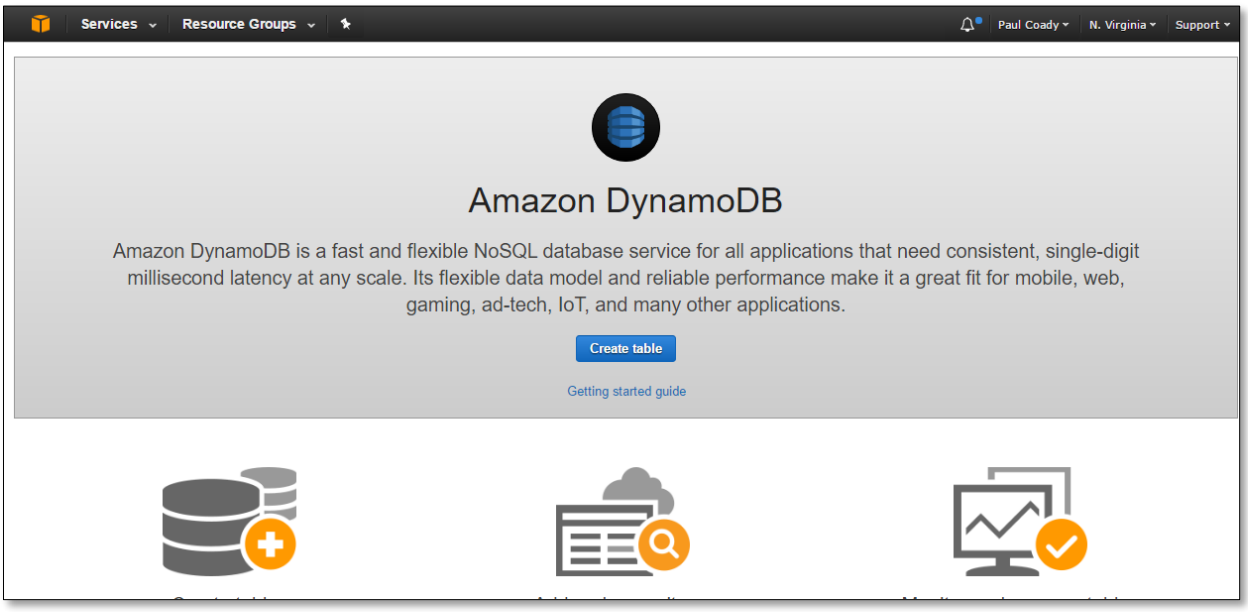
http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/DynamoDB.html

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 lastest 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"

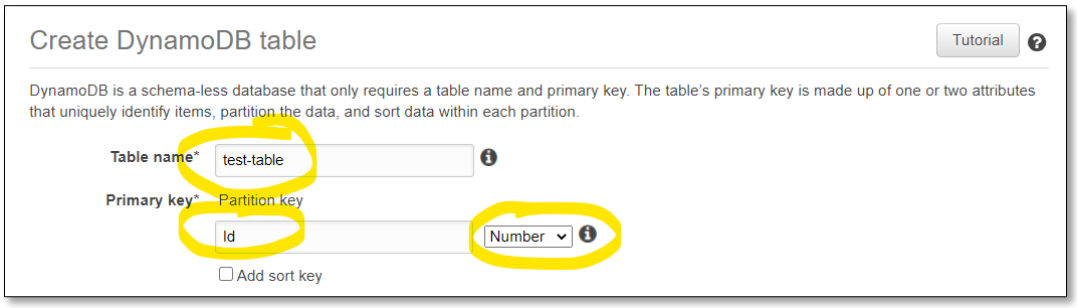
Enter the following details (enter exactly with correct case)

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

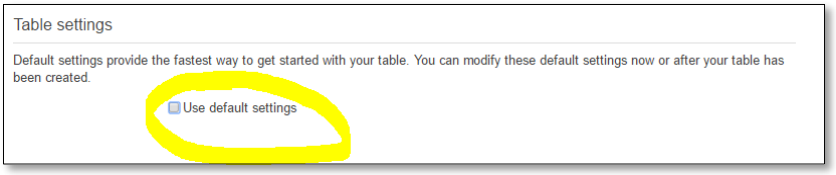
Table Name: test-table

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

Primary Key Type: Number

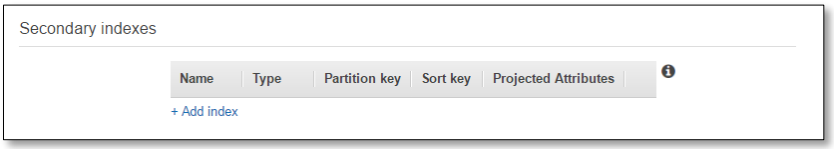


Uncheck *Use Default Settings*



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

Click Add index to table.



Enter index details:

Partition key: ProductCategory

Type: String

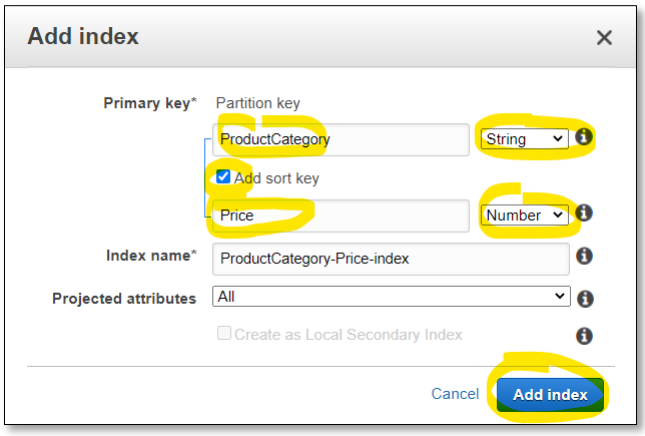
Click *Add sort key*

Sort key: Price

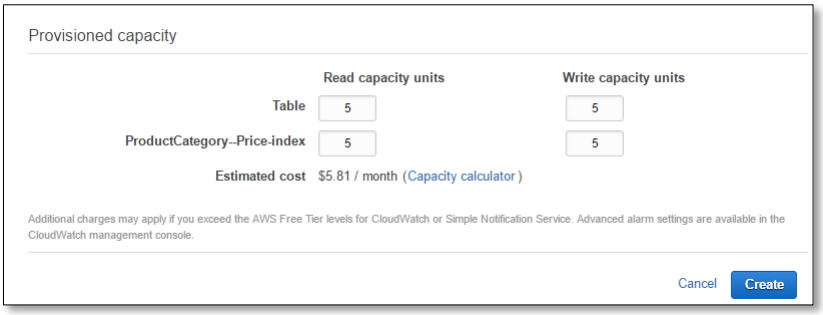
Type: Number

Leave *Index name* as ProductCategory-Price-index

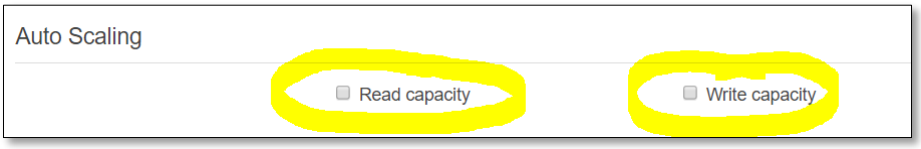
Click *Add index*



Continue using default settings.

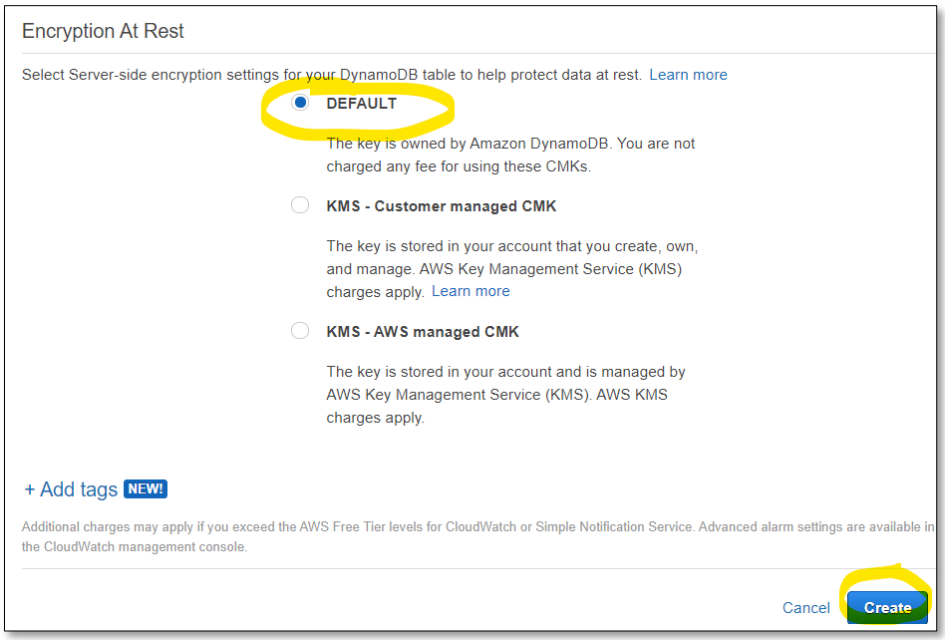


Disable auto scaling

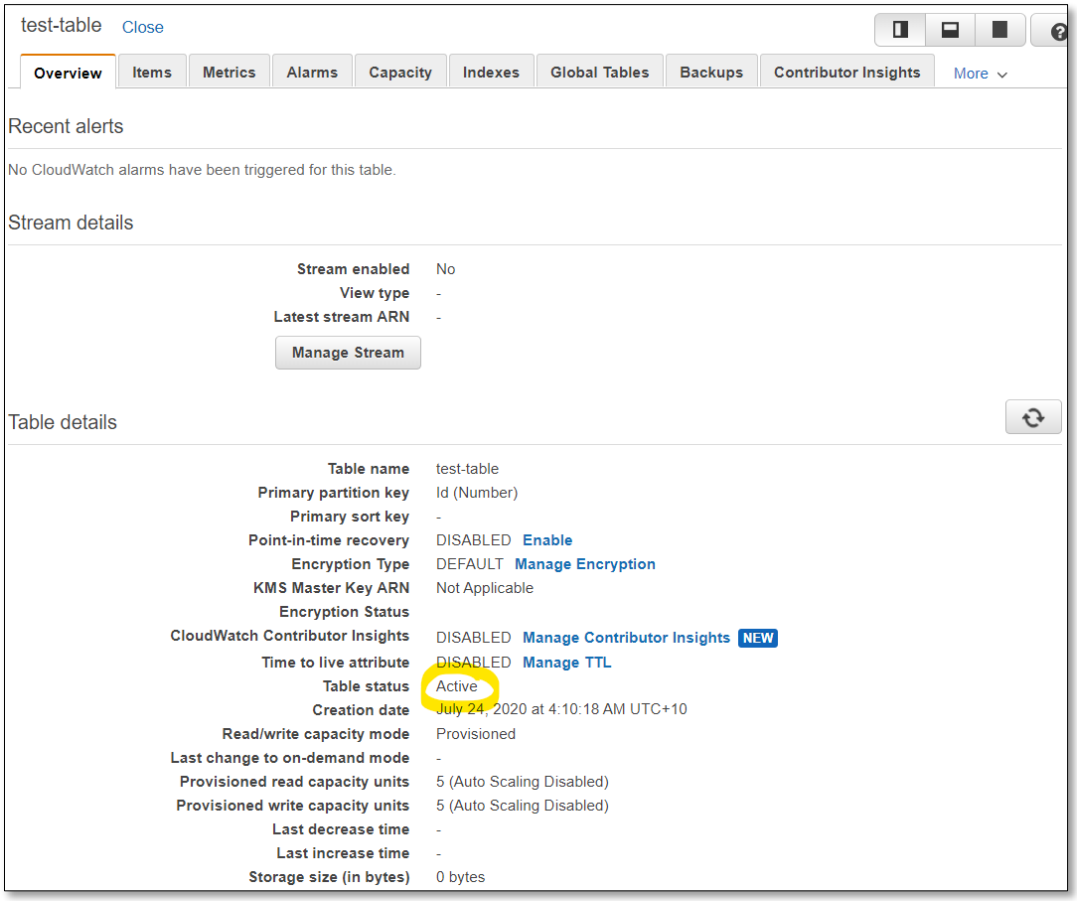


Leave encryption *DEAULT*

Click Create.



Press refresh until table status is listed as active.



Click on Items tab

Click on Create Item

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

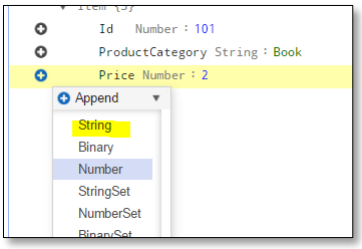
Enter the Id as 101

ProductCategory- String: Book

Price - Number: 2

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

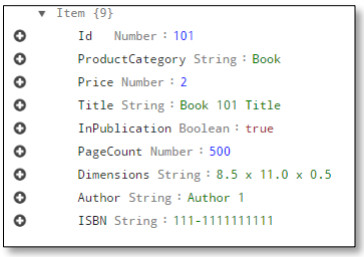
Select Append then String to add another attribute



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



Click Save

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.

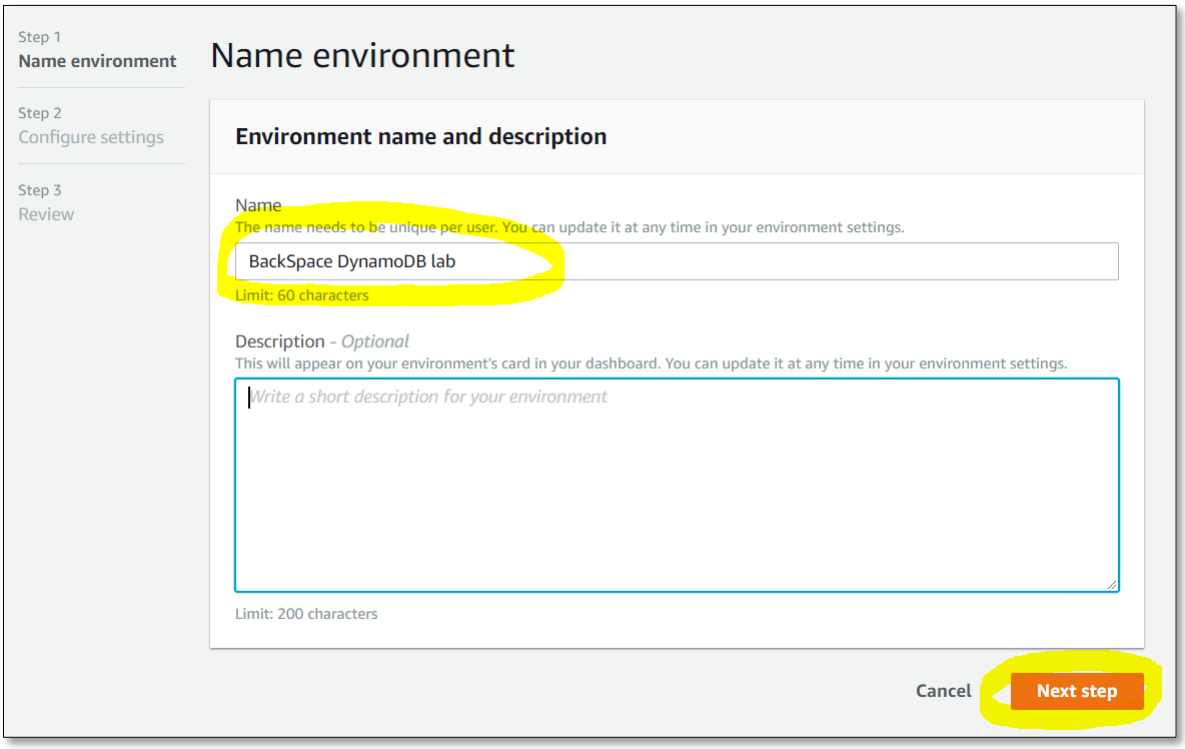
Creating the Cloud 9 Environment

Go to Services - Cloud9 from the console

Click 'Create environment"

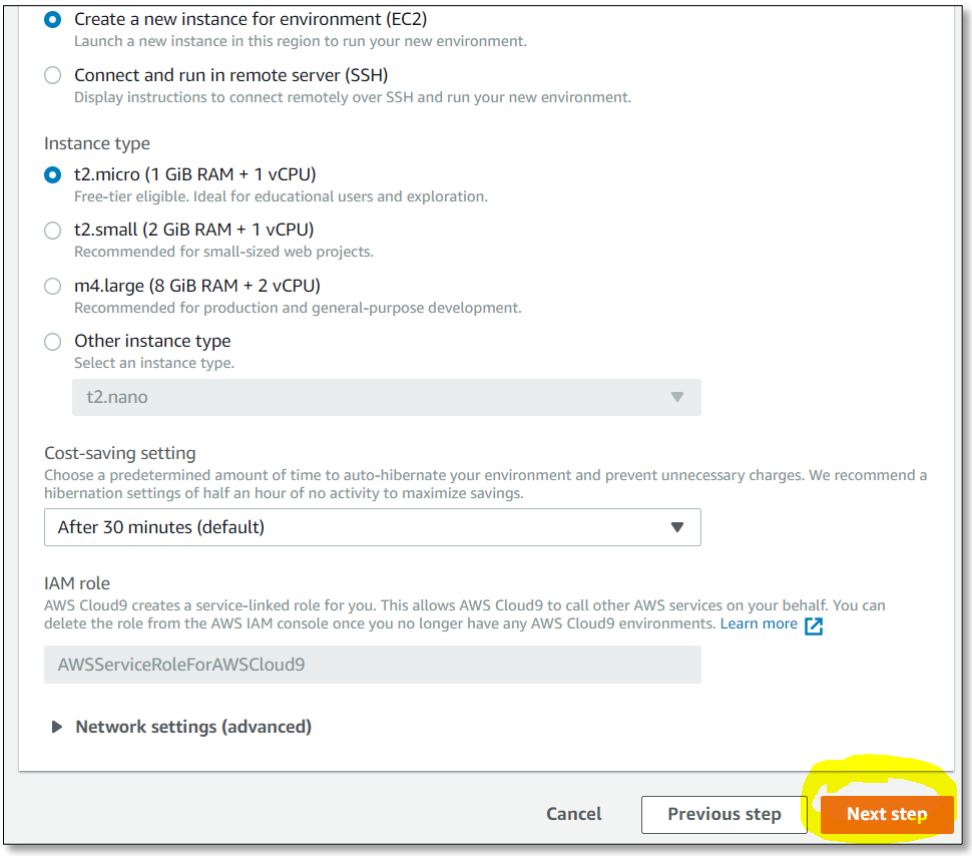
Give your environment a name

Click 'Next step"

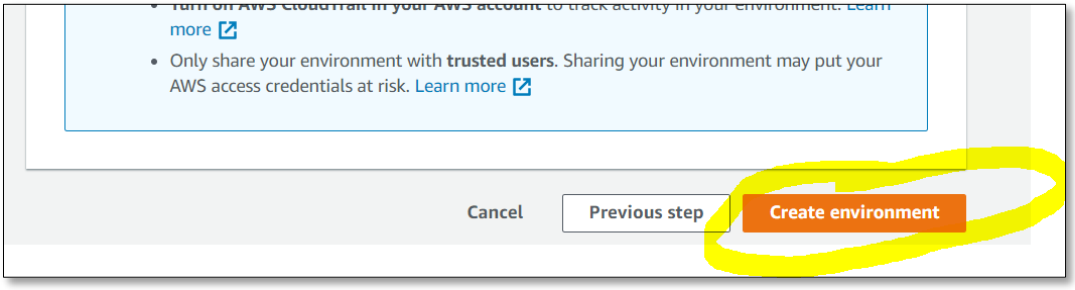


Leave default settings

Click 'Next step"



Click '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-
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-   [New] - Idle   aws-dynamodb-node
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
```

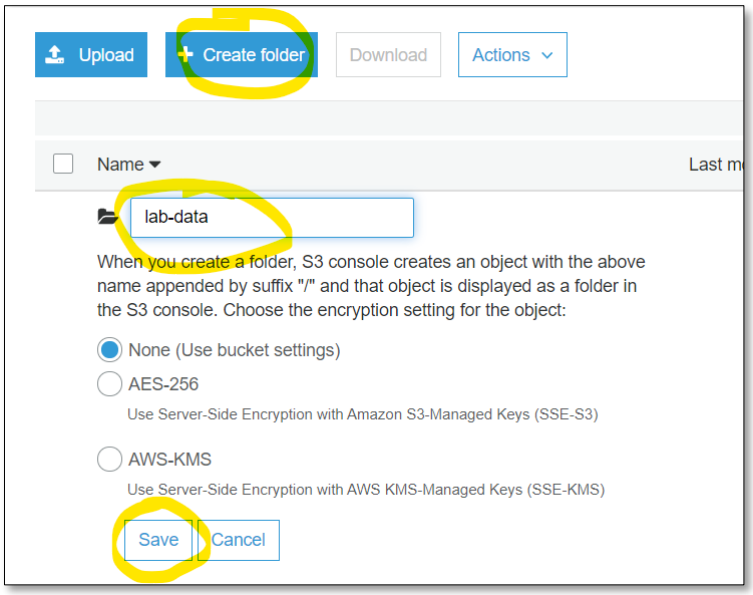
Download the following file:

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

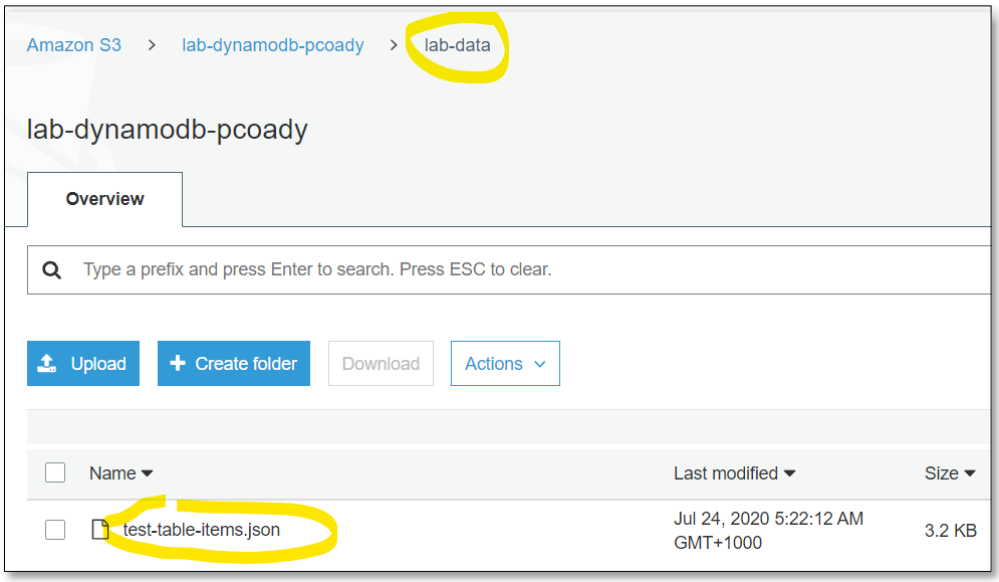
Go to the S3 console

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.

```
17      }
18      else {
19         console.log(data.TableNames);        // successful response
20         downloadData();
21      }
22   });
23
24   function downloadData(){
25       // Get JSON file from S3
26       var s3 = new AWS.S3();
27       var params = {Bucket: 'lab-dynamodb-pcoady', Key: 'lab-data/test-table-items.json'};
28       s3.getObject(params, function(error, data) {
29         if (error) {
30           console.log(error); // error is Response.error
31         } else {
32           var dataJSON = JSON.parse(data.Body);
33           console.log(JSON.stringify(dataJSON));
34         }
35       });
36     };
37
```

```
36:5   JavaScript   Spaces: 3
npm - "ip-172-31-92-   [New] - Idle   aws-dynamodb-node
Run   Run Config Name   Command:   aws-dynamodb-nodejs/index.js   Runner: Node.js   CWD   ENV
":{"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"},"Descri
ption":{"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":"R
oad"},"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"},"G
ender":{"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"]},"ProductCateg
ory":{"S":"Bike"}}}}]}
Waiting for the debugger to disconnect...

Process exited with code: 0
```

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 we will add these items to our DynamoDB database.

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

```
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
      }
    });
  }
```

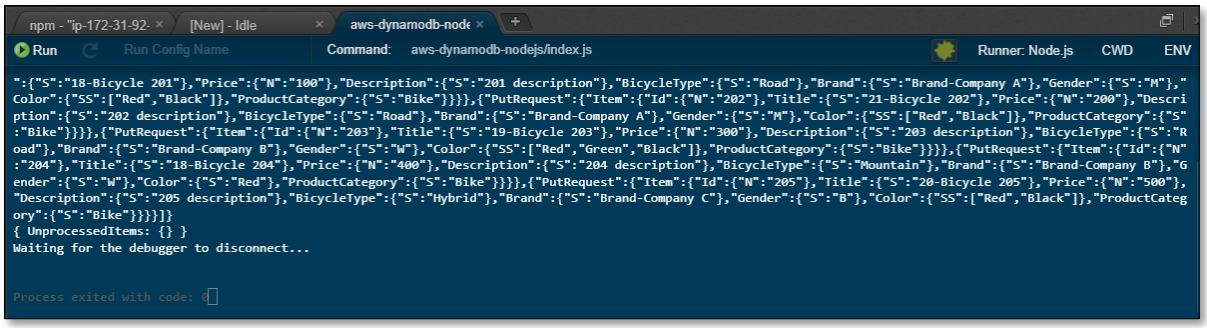
Click 

+ 

to save to the EC2 instance.

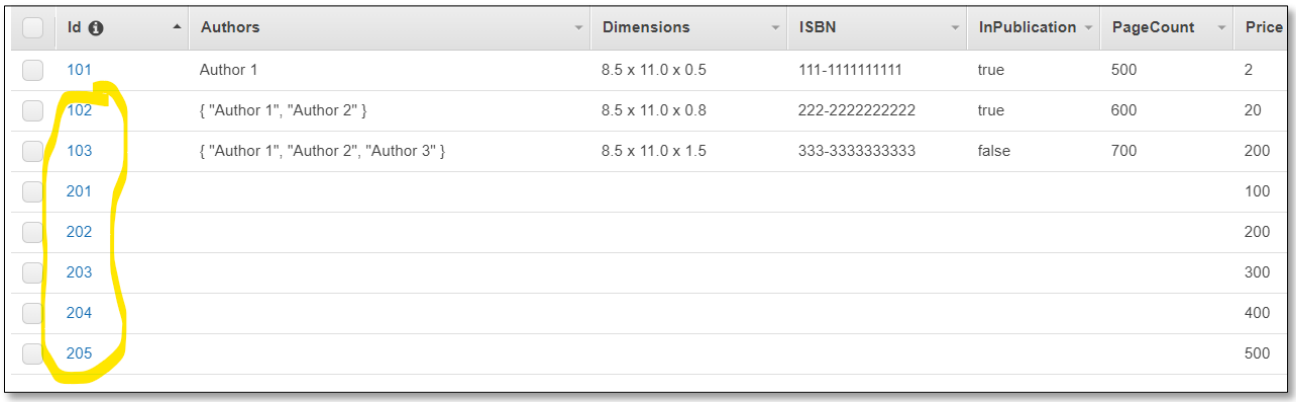
Now run the app again.

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



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:



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

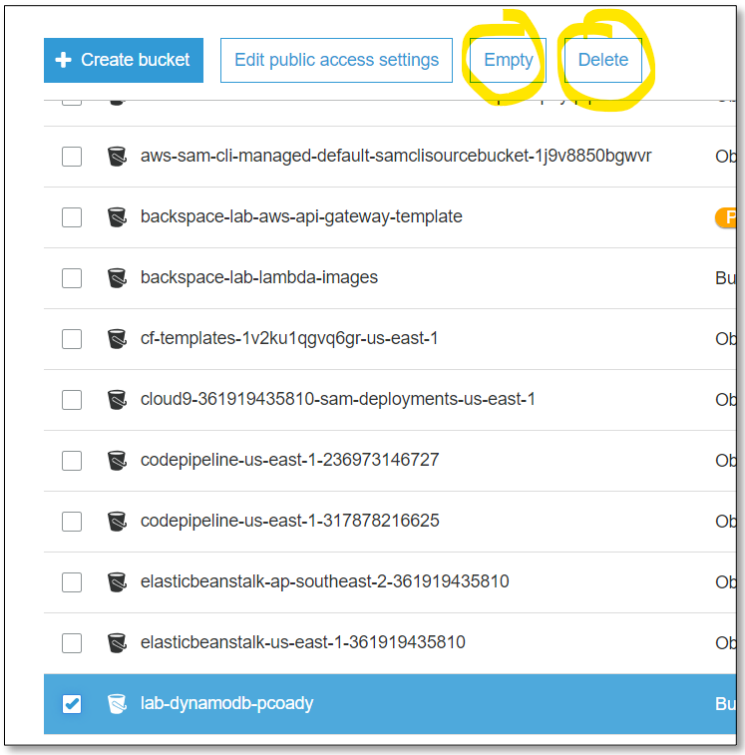
```
npm - "ip-172-31-92- ×   [New] - Idle
Run   Run Config Name   Co
},"Gender":{"S":"W"},"Color":{"S":"Red"},"Proc
},"BicycleType":{"S":"Hybrid"},"Brand":{"S":"E
{ UnprocessedItems: {} }
[ { Title: { S: '18-Bicycle 201' },
    Gender: { S: 'M' },
    Price: { N: '100' },
    Brand: { S: 'Brand-Company 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
```

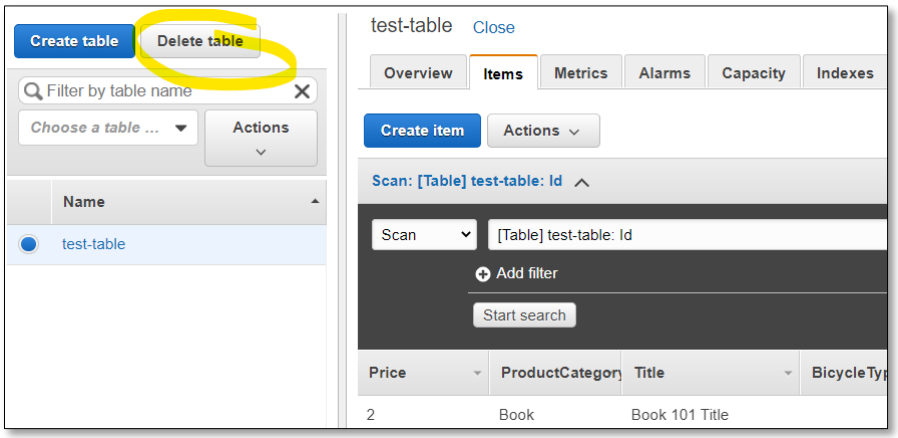
NOTE: If you get an error message such as *'Query condition missed key schema element: ProductCategory'* it means you have misspelled the index when creating the table. In this case a space is on the end of ProductCategory which caused an error.

Clean Up

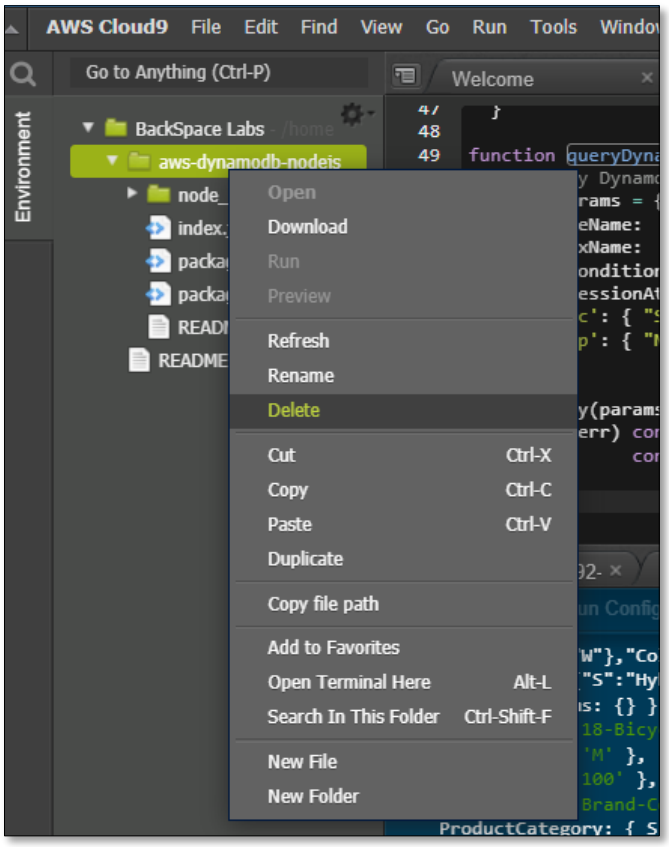
Go to the S3 console and empty the bucket, then delete it.



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.

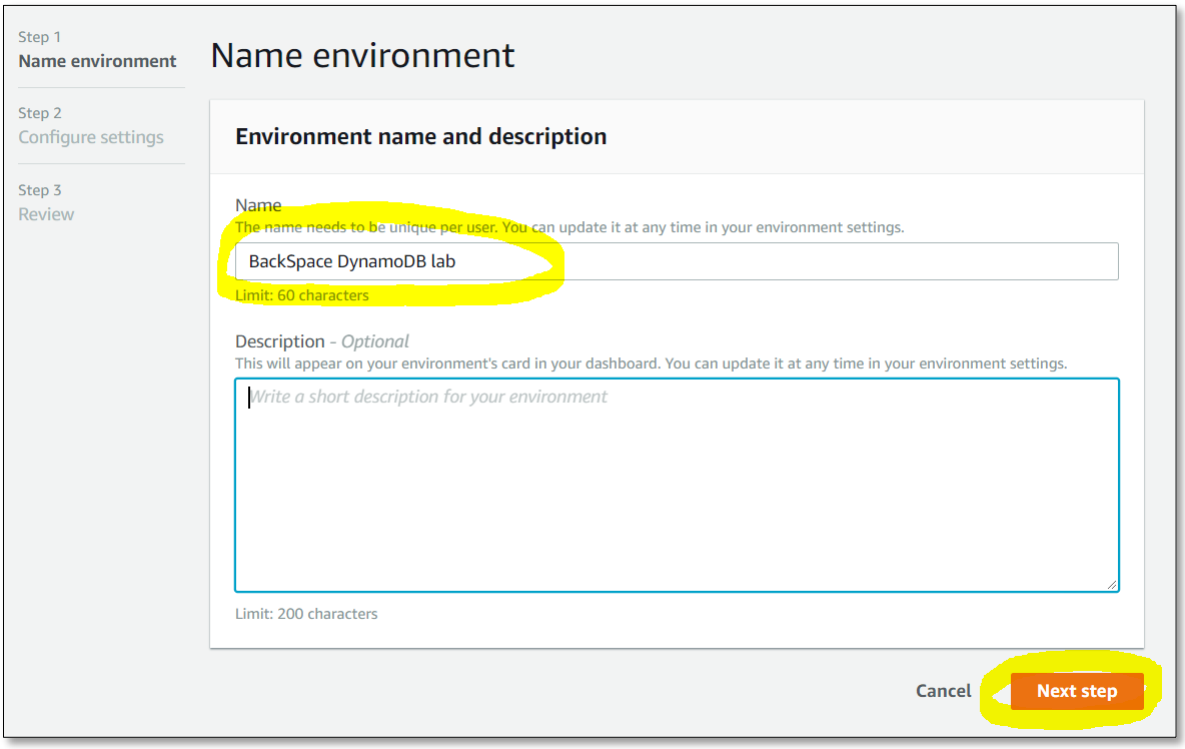
Creating the Cloud 9 Environment

Go to Services - Cloud9 from the console

Click 'Create environment"

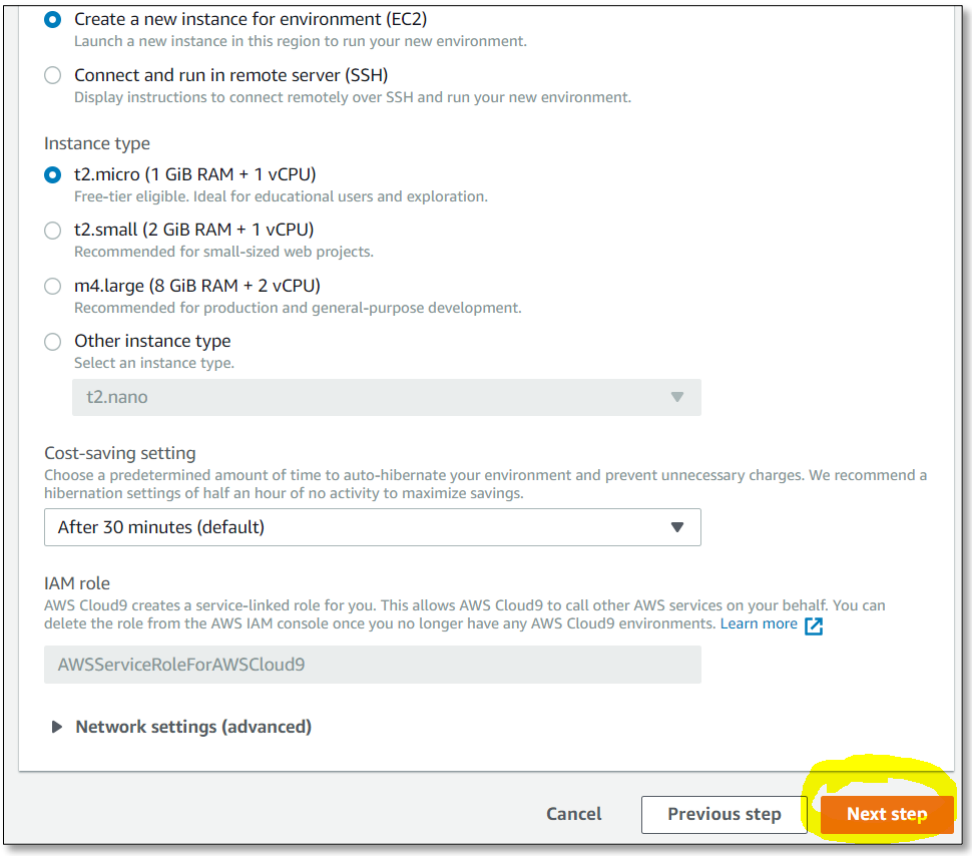
Give your environment a name

Click 'Next step"

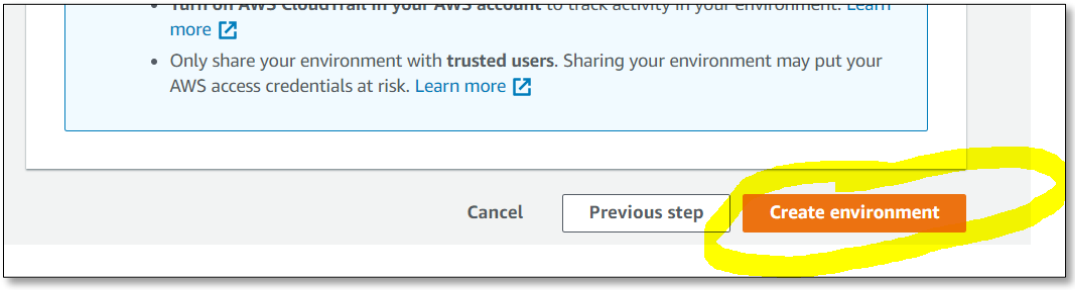


Leave default settings

Click 'Next step"



Click '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*

```python
# 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 ×   [New] - Idle ×
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 ×   [New] - Idle ×   aws-sqs-python/sqs_ ×
Run   Run Config Name   Command:   aws-sqs-python/sqs_example.py
Table name: test-table

Process exited with code: 0
```

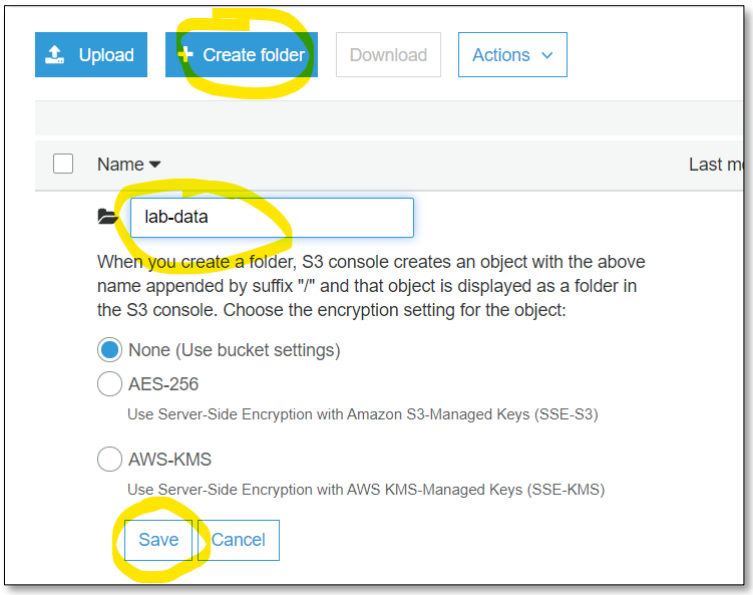
Download the following file:

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

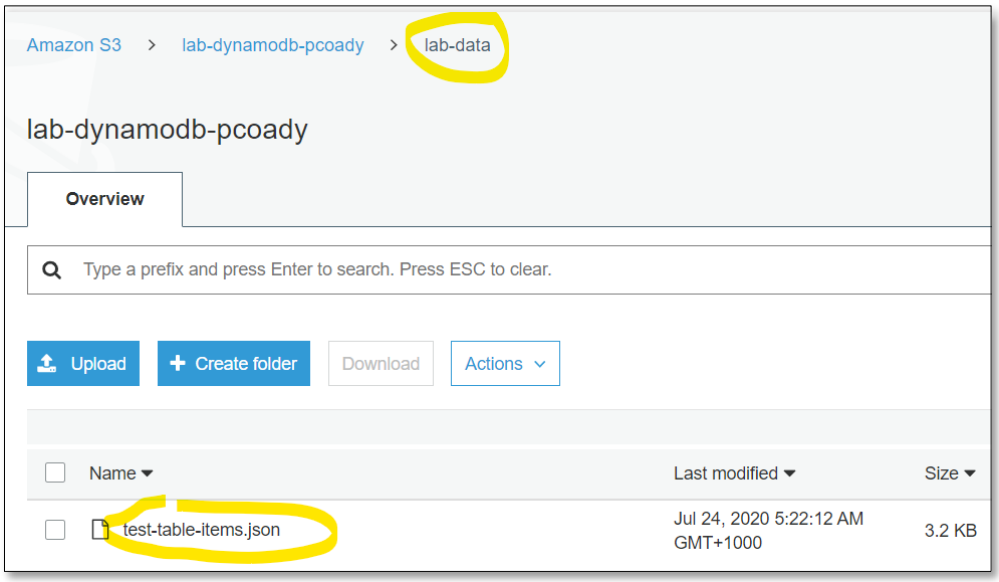
Go to the S3 console

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.

```
bash - "ip-172-31-92 ×   [New] - Idle ×   aws-sqs-python/sqs_ ×
Run   Run Config Name   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)
```

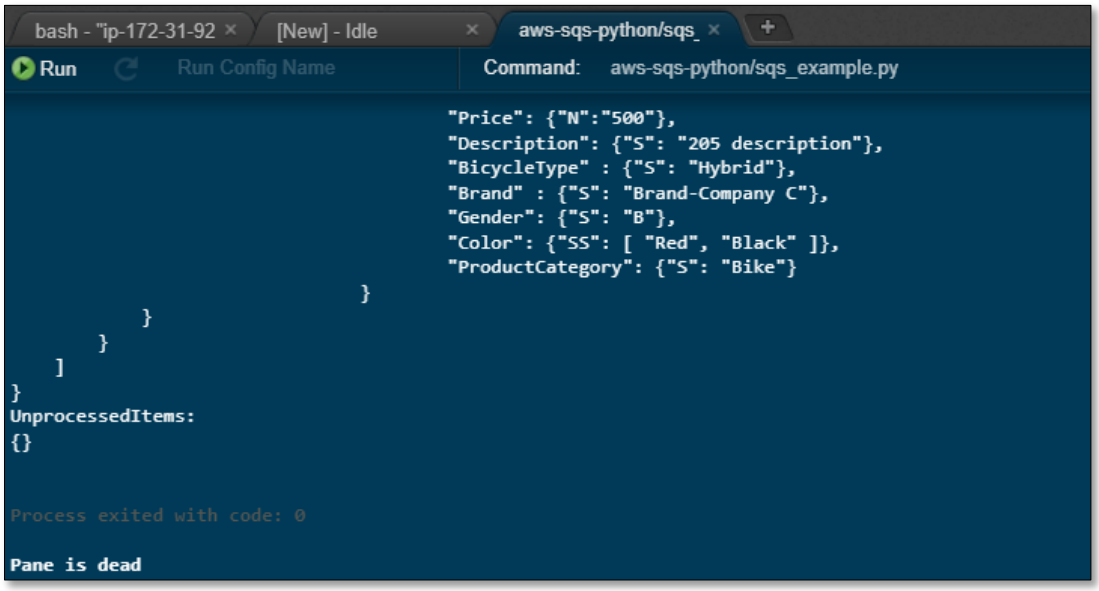
Click 

+ 

to save to the EC2 instance.

Now run the app again.

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



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:

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

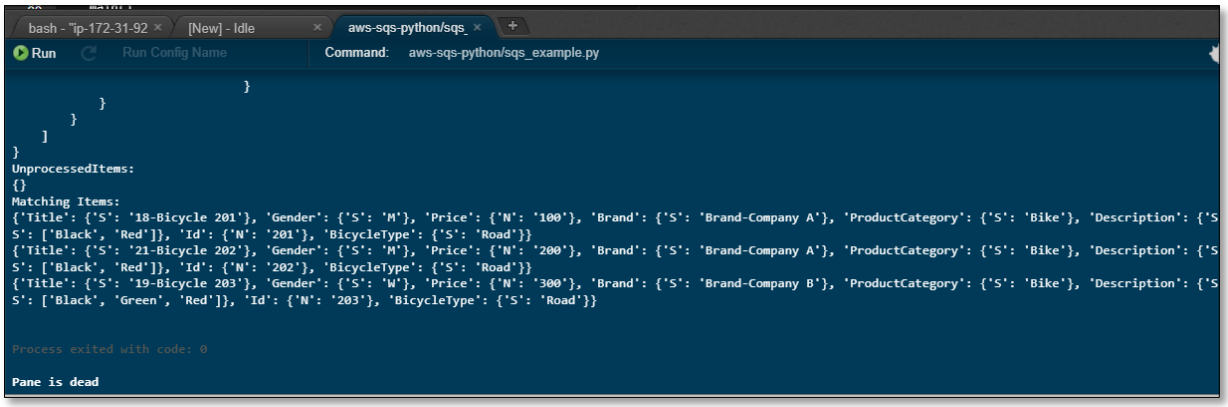
Now add the *query_dynamo_db* function to import the data to DynamoDB:

Now add the *query_dynamo_db* function:

```python
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.

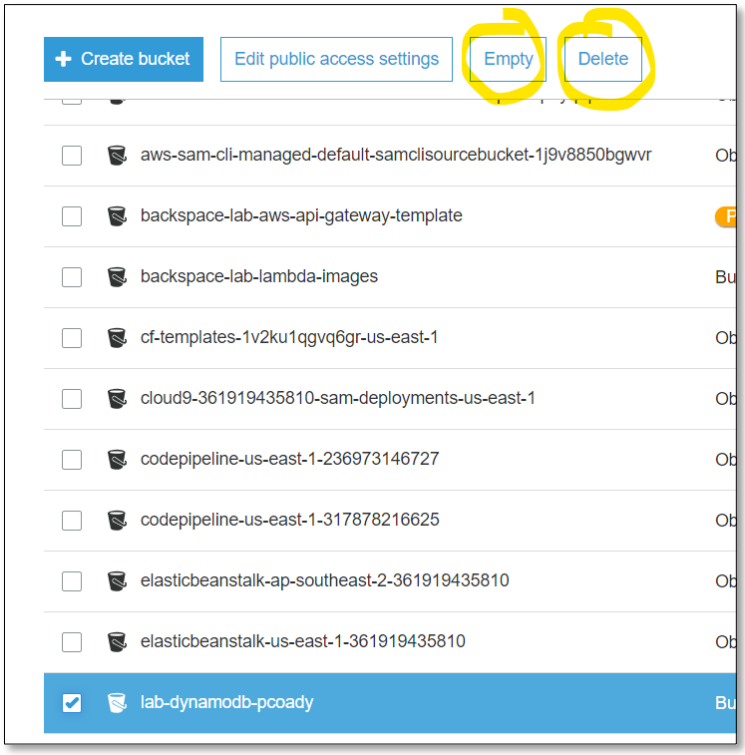
You will have the three bikes $300 or less output to the console.



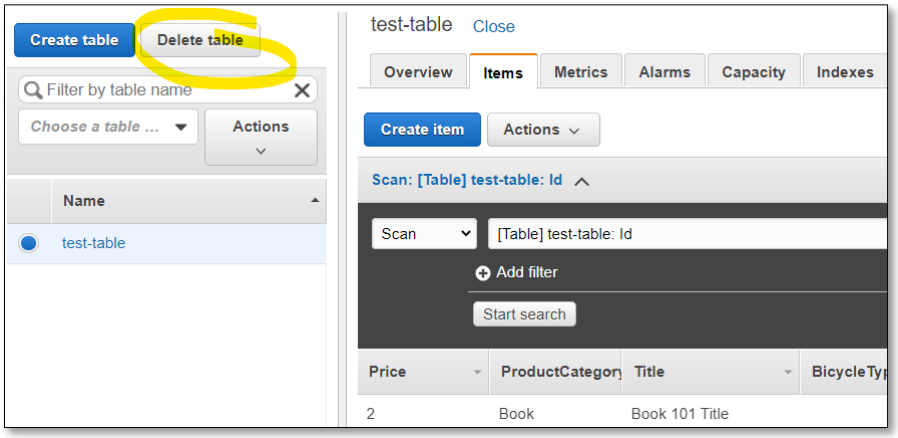
NOTE: If you get an error message such as *'Query condition missed key schema element: ProductCategory'* it means you have misspelled the index when creating the table. In this case a space is on the end of ProductCategory which caused an error.

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