Homework 2

AWS CLI and DynamoDB

Brian Prost

Dr. Erroll Waithe

SDEV 400 7380

16 November 2021

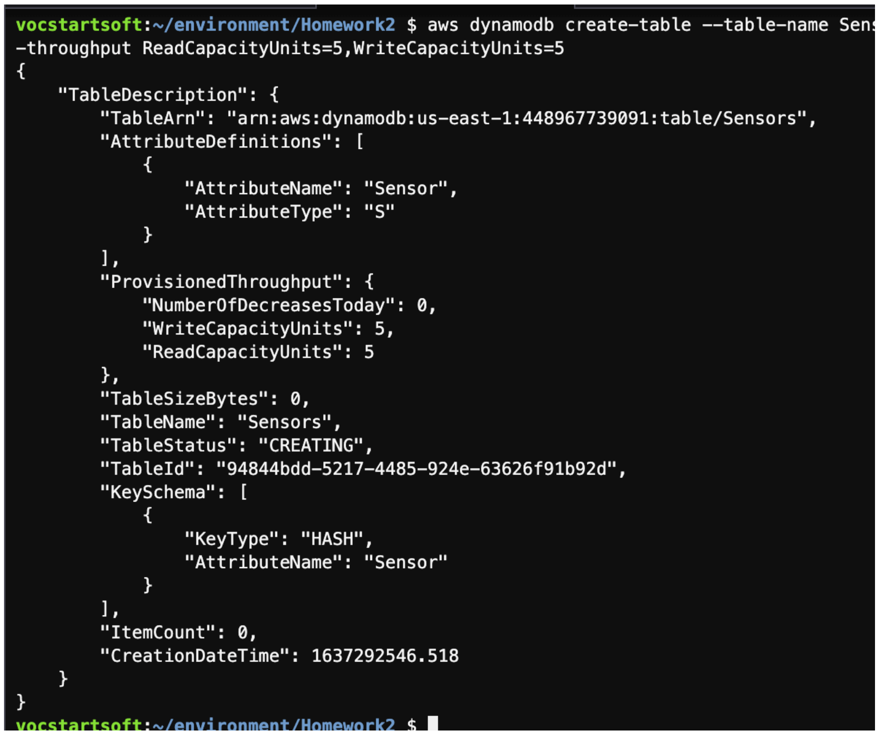
Part One

Create Table

AWS CLI Command

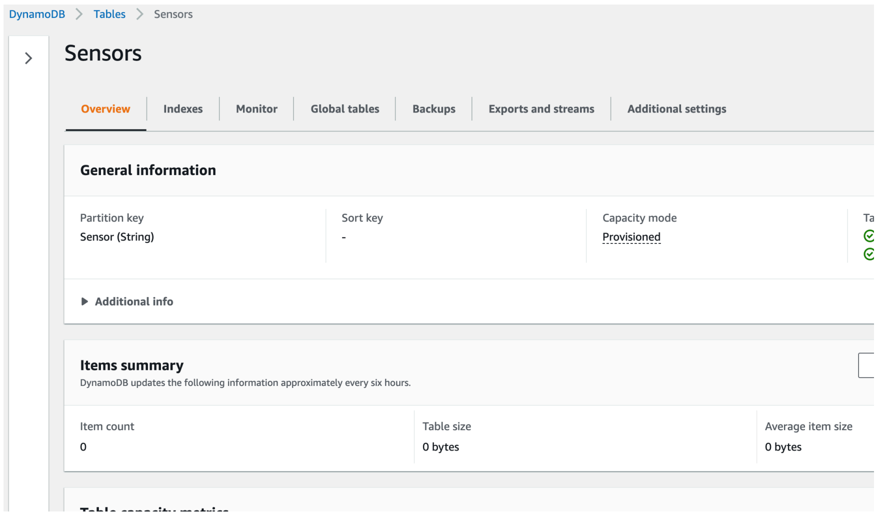
```
aws dynamodb create-table --table-name Sensors --attribute-definitions
    AttributeName=Sensor,AttributeType=S --key-schema
    AttributeName=Sensor,KeyType=HASH --provisioned-throughput
    ReadCapacityUnits=5,WriteCapacityUnits=5
```

Did it work? I don’t believe you. Pics or it didn’t happen



```
vocstartsoft:~/environment/Homework2 $ aws dynamodb create-table --table-name Sens
-throughput ReadCapacityUnits=5,WriteCapacityUnits=5
{
    "TableDescription": {
        "TableArn": "arn:aws:dynamodb:us-east-1:448967739091:table/Sensors",
        "AttributeDefinitions": [
            {
                "AttributeName": "Sensor",
                "AttributeType": "S"
            }
        ],
        "ProvisionedThroughput": {
            "NumberOfDecreasesToday": 0,
            "WriteCapacityUnits": 5,
            "ReadCapacityUnits": 5
        },
        "TableSizeBytes": 0,
        "TableName": "Sensors",
        "TableStatus": "CREATING",
        "TableId": "94844bdd-5217-4485-924e-63626f91b92d",
        "KeySchema": [
            {
                "KeyType": "HASH",
                "AttributeName": "Sensor"
            }
        ],
        "ItemCount": 0,
        "CreationDateTime": 1637292546.518
    }
}
vocstartsoft:~/environment/Homework2 $
```

Fig. 1: Creating a table



DynamoDB > Tables > Sensors

Sensors

General information

Partition key: Sensor (String)

Sort key: -

Capacity mode: Provisioned

Ta

Items summary

DynamoDB updates the following information approximately every six hours.

Item count: 0

Table size: 0 bytes

Average item size: 0 bytes

Fig. 2: AWS DynamoDB console page verifying table was created

Write Data

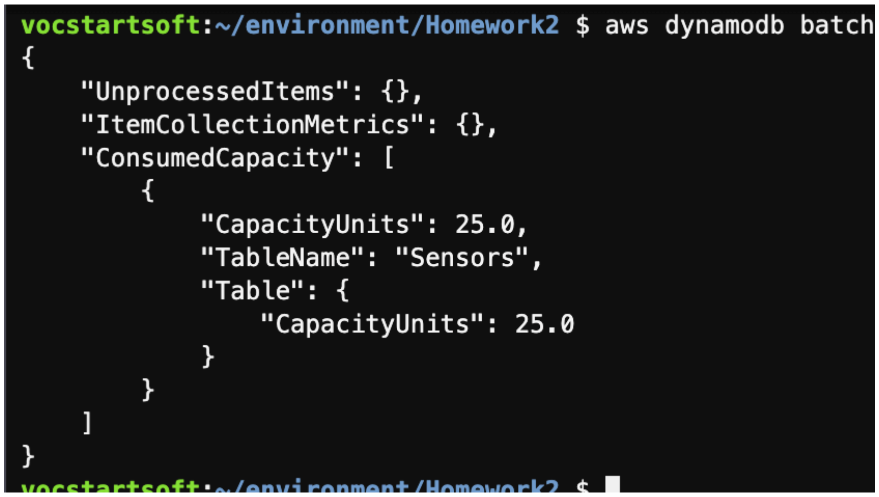
JSON Format & Data

```
{
    "Sensors": [
        {
            "PutRequest": {
                "Item": {
                    "Sensor": {"S": "sensor_Y96CDEQF"},
                    "SensorDescription": {"S": "ice core sensor"},
                    "ImageFile": {"S": "/sensors/image/sensor_Y96CDEQF.png"},
                    "SampleRate": {"N": "60"},
                    "Locations": {"SS":[
                        "Seoul, South Korea",
                        "Vijayawada, India"
                    ]}
                }
            }
        },
        ...(*24)
    ]
}
```

AWS CLI Command for Writing the data

```
aws dynamodb batch-write-item --request-items file://sensors.json --return-
    consumed-capacity INDEXES --return-item-collection-metrics SIZE
```

Seeing is believing still…show me that it did in fact work



```
vocstartsoft:~/environment/Homework2 $ aws dynamodb batch
{
    "UnprocessedItems": {},
    "ItemCollectionMetrics": {},
    "ConsumedCapacity": [
        {
            "CapacityUnits": 25.0,
            "TableName": "Sensors",
            "Table": {
                "CapacityUnits": 25.0
            }
        }
    ]
}
vocstartsoft:~/environment/Homework2 $
```

Fig. 3 Loading of table data

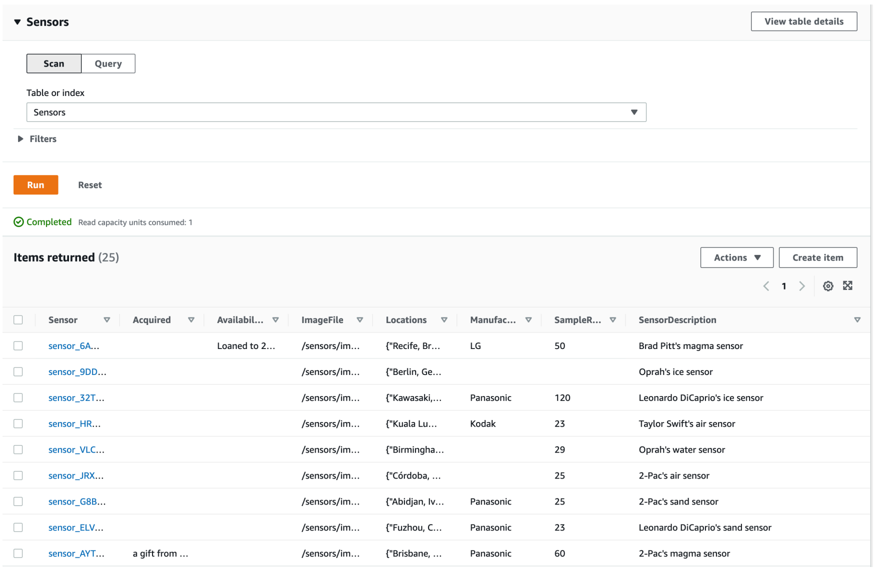


Table or index: Sensors

Filters

Completed Read capacity units consumed: 1

Items returned (25)

Sensor	Acquired	Availabil…	ImageFile	Locations	Manufac…	SampleR…	SensorDescription
sensor_6A…		Loaned to 2…	/sensors/im…	{"Recife, Br…	LG	50	Brad Pitt's magma sensor
sensor_9DD…			/sensors/im…	{"Berlin, Ge…			Oprah's ice sensor
sensor_32T…			/sensors/im…	{"Kawasaki,…	Panasonic	120	Leonardo DiCaprio's ice sensor
sensor_HR…			/sensors/im…	{"Kuala Lu…	Kodak	23	Taylor Swift's air sensor
sensor_VLC…			/sensors/im…	{"Birmingha…		29	Oprah's water sensor
sensor_JRX…			/sensors/im…	{"Córdoba, …		25	2-Pac's air sensor
sensor_G8B…			/sensors/im…	{"Abidjan, Iv…	Panasonic	25	2-Pac's sand sensor
sensor_ELV…			/sensors/im…	{"Fuzhou, C…	Panasonic	23	Leonardo DiCaprio's sand sensor
sensor_AYT…	a gift from …		/sensors/im…	{"Brisbane, …	Panasonic	60	2-Pac's magma sensor

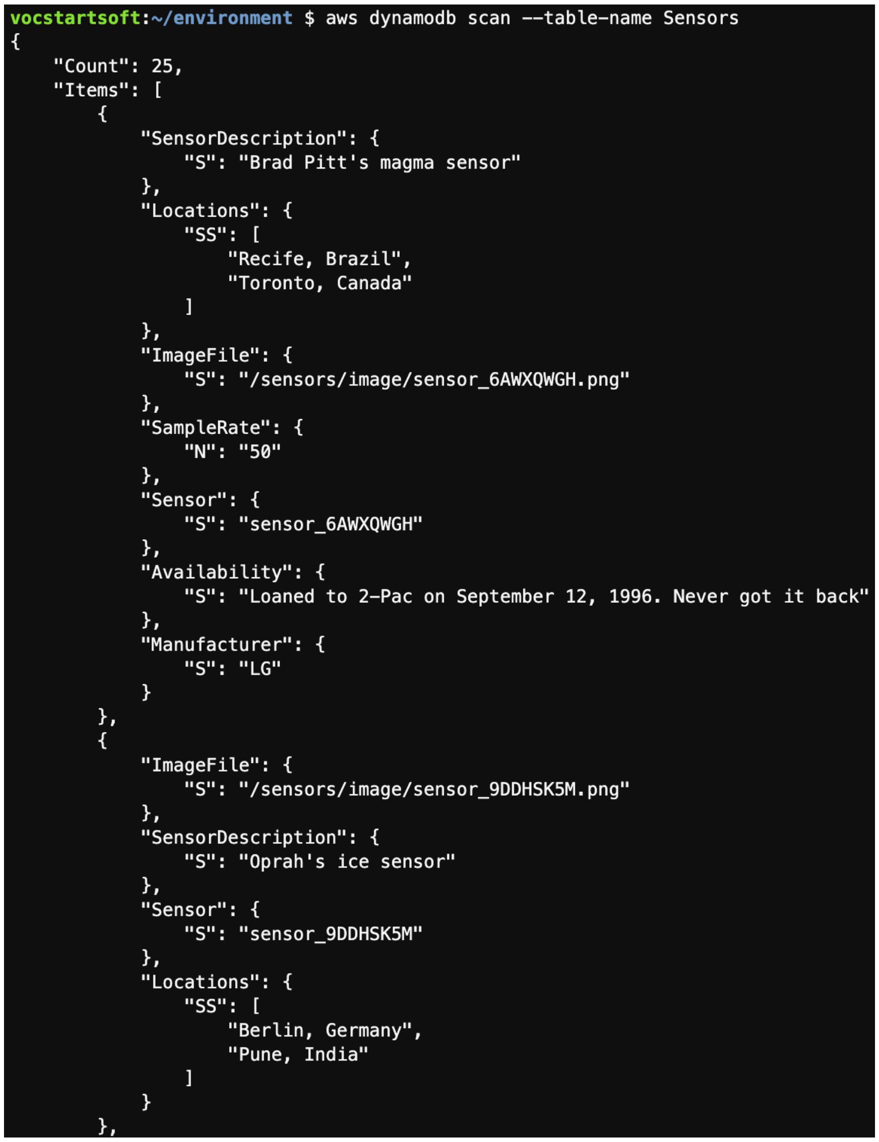
Fig. 4: AWS DynamoDB console verifying the batch write of all 25 of our entries

Printing the contents of the table

AWS CLI Command

```
aws dynamodb scan --table-name Sensors
```

No corny statement here. Just a screenshot of part...or kernel 🌽...of the scan



```
vocstartsoft:~/environment $ aws dynamodb scan --table-name Sensors
{
    "Count": 25,
    "Items": [
        {
            "SensorDescription": {
                "S": "Brad Pitt's magma sensor"
            },
            "Locations": {
                "SS": [
                    "Recife, Brazil",
                    "Toronto, Canada"
                ]
            },
            "ImageFile": {
                "S": "/sensors/image/sensor_6AWXQWGH.png"
            },
            "SampleRate": {
                "N": "50"
            },
            "Sensor": {
                "S": "sensor_6AWXQWGH"
            },
            "Availability": {
                "S": "Loaned to 2-Pac on September 12, 1996. Never got it back"
            },
            "Manufacturer": {
                "S": "LG"
            }
        },
        {
            "ImageFile": {
                "S": "/sensors/image/sensor_9DDHSK5M.png"
            },
            "SensorDescription": {
                "S": "Oprah's ice sensor"
            },
            "Sensor": {
                "S": "sensor_9DDHSK5M"
            },
            "Locations": {
                "SS": [
                    "Berlin, Germany",
                    "Pune, India"
                ]
            }
        },
```

Fig. 5: printing all items in the "Sensors" table

Part Two

Create Table

We're creating a table called CourseCatalog, with a KeySchema of CourseID that is a key of the type HASH, and an attribute of the type Number.

```
dynamodb = boto3.resource("dynamodb")
table = dynamodb.create_table(
    TableName="CourseCatalog",
    KeySchema=[
        {
            "AttributeName": "CourseID",
            "KeyType": "HASH"
        }
    ],
    AttributeDefinitions=[
        {
            "AttributeName": "CourseID",
            "AttributeType": "N"
        },
    ],
    ProvisionedThroughput={
        "ReadCapacityUnits": 5,
        "WriteCapacityUnits": 5
    }
)
```

Add Data to Table

Soooo, this part really messed with me. I went through probably 30-40 revisions of a JSON document and numerous different implementations of a batch upload of our table data, in an effort to reduce network requests.

But even before we upload the data, we need to make sure that the table exists. This part also got me pretty bad as the program was executing faster than the completion/availability of the table. So we invoke the wait_until _exists() method to delay any further execution, which is, in this case, loading the table data into it.

```
# wait for table to be available
print("Creating table. Please wait...")
table.wait_until_exists()
print("Table created! Adding data now...")

# add data to table
with open('course_catalog_for_batch.json') as json_data:
    request_items = json.loads(json_data.read())
    for item in request_items:
        table.put_item(Item=item)
```

Interface for User Interaction

Simple CLI here. get _user _input is a simple method that just handles error handling without having to repeat code for try/except statements.

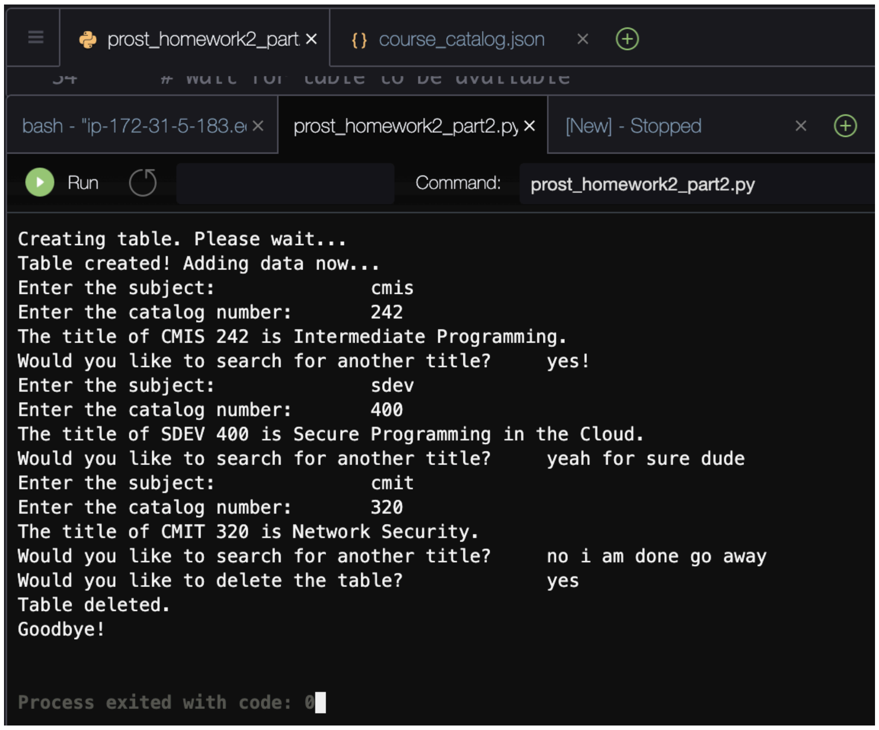
```
while(True):
        course_subject = get_user_input("Enter the subject:\t\t").upper()
        catalog_number = int(get_user_input("Enter the catalog number:\t"))

        course_title = get_course_title(table, course_subject,
    catalog_number)
        if not course_title:
            print("No course found. Please try again")
            continue

        print(
            f"The title of {course_subject} {catalog_number} is
    {course_title}.")
```

Getting Course Title

For grabbing the course course title, we have to download the entire table unfortunately so that we can search by attribute. Once we've done that, we can use FilterExpression to find the course item we're looking for, and then return the "CourseTitle" attribute of that item once we find a match.

```
def get_course_title(table, course_subject, catalog_number):
    response = table.scan(
        FilterExpression=Attr("Subject").eq(course_subject)
        & Attr("CatalogNumber").eq(catalog_number)
    )
    items = response["Items"]
    if len(items) > 0:
        the_item = items[0]
        return the_item["CourseTitle"]
    else:
        return False
```

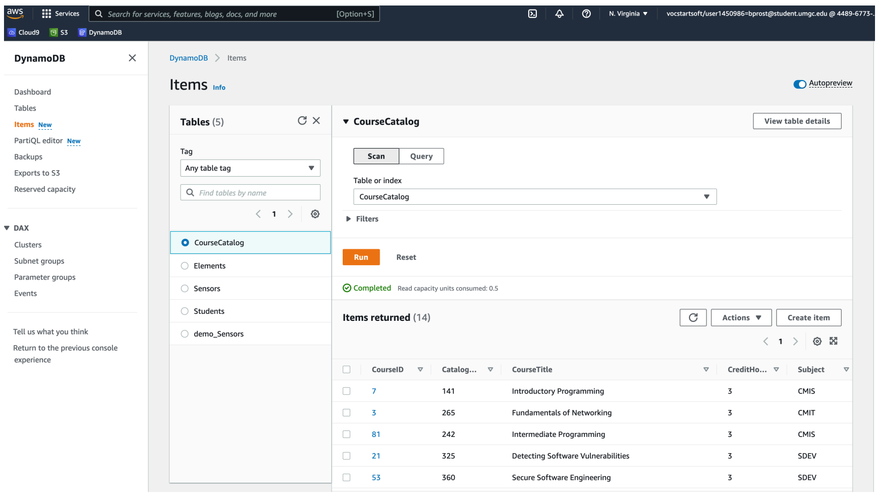
Sample Run



```
Creating table. Please wait...
Table created! Adding data now...
Enter the subject:              cmis
Enter the catalog number:       242
The title of CMIS 242 is Intermediate Programming.
Would you like to search for another title?     yes!
Enter the subject:              sdev
Enter the catalog number:       400
The title of SDEV 400 is Secure Programming in the Cloud.
Would you like to search for another title?     yeah for sure dude
Enter the subject:              cmit
Enter the catalog number:       320
The title of CMIT 320 is Network Security.
Would you like to search for another title?     no i am done go away
Would you like to delete the table?             yes
Table deleted.
Goodbye!


Process exited with code: 0
```

Fig. 6: Sample run of our program that retreives, from DynamoDB, the course title of a class from a user-entered subject and catalog number

Table Status



Completed Read capacity units consumed: 0.5

Items returned (14)

CourseID	Catalog...	CourseTitle	CreditHo...	Subject
7	141	Introductory Programming	3	CMIS
3	265	Fundamentals of Networking	3	CMIT
81	242	Intermediate Programming	3	CMIS
21	325	Detecting Software Vulnerabilities	3	SDEV
53	360	Secure Software Engineering	3	SDEV

Fig. 7: Verification inside of the DynamoDB console that our table was in fact created from the Python program

Ending the program and deleting table

To end, we ask if the user would like to continue. If they would, we ask if they would also like to delete the table on the output. While it seems like a silly thing to include, I believe that it is important because the program is the thing that creates the table in the first place. So if we have data, and we upload it to a service to use the features of that service, there will often be a use case that we don't want or need to leave that data in that service.

```
keep_going = get_user_input(
            "Would you like to search for another title?\t")
        if keep_going[0].lower() == 'n':

            delete_table_confirmation = get_user_input(
                "Would you like to delete the table?\t\t")

            if delete_table_confirmation[0].lower() == 'y':
                table.delete()
                print("Table deleted.")

            print("Goodbye!")
            break
```

Alternatively, to delete the table via the AWS CLI, we could run:

```
aws dynamodb delete-table --table-name CourseCatalog
```

And if we also wanted to delete the Sensors table, we would use the AWS CLI like this:

```
aws dynamodb delete-table --table-name Sensors
```