

Name: Pradeep Medagiri

Homework 10: Lambda

You are building an information system for a startup financial company in Northern Virginia. It needs a database to store companies' financial information, such as their stock price, industry they are in, and some other information (some companies have more information than others). This database is intended for the customers so that they can **query** information quickly using company's **ticker**. You decide to use the NoSQL and fast AWS **DynamoDB** for the database.

As part of their work process, a staff will research the financial information of a company, create a json file to record the data, and upload the json file in S3. One company per json file.

The company wants to implement Robotic Process Automation (RPA). In other words, they want to automate repetitive activities with software. So instead of populating the database manually, they want the system to automatically read financial data every time a json file is uploaded to the S3 bucket, and to populate the data into the database accordingly.

You need to create a lambda function (similar to the demo) to automatically read uploaded json file and populate the dynamodb database.

You have data of 4 companies and you need to create a json file for each of them for testing purposes (all numbers are integers to simplify things).

Ticker: AAPL

Stock_price: 389

Industry: Computer

Name: Apple

Volume: 20279276

Ticker: IBM

Stock_price: 129

Name: International Business Machines

Ticker: HPE

Stock_price: 10

Industry: Computer

Name: Hewlett Packard Enterprise

Ticker: TSLA

Stock_price: 1592

Industry: Automobile

Name: Tesla

Volume: 14161080

State: California

City: San Francisco

Your tasks:

1. Create a dynamodb table – name it yourlastname-hw10-dynamodb.
2. Create an S3 bucket with private access to store the json files.
3. Create a json file for each of the companies listed above. Name the each of the json file using the ticker. For example, the json file for the apple is AAPL.json
4. Create a lambda function that will read json file uploaded to the S3 and automatically populate the dynamodb table.
5. Upload all four json files. All four companies must be in the dynamodb database.

What to submit:

1. The screenshot of the IAM Role you created that shows all its policies. (16%)

The screenshot shows the AWS IAM console for the role 'medagiri-hw10'. The role's ARN is 'arn:aws:iam::431429019981:role/medagiri-hw10'. It allows Lambda functions to call AWS services on behalf of the user. The role has three policies attached: AmazonS3FullAccess, AmazonDynamoDBFullAccess, and CloudWatchLogsFullAccess. The role was created on 2020-12-02 at 17:10 CST and last activity was on 2020-12-02 at 17:47 CST (Today). The maximum session duration is 1 hour.

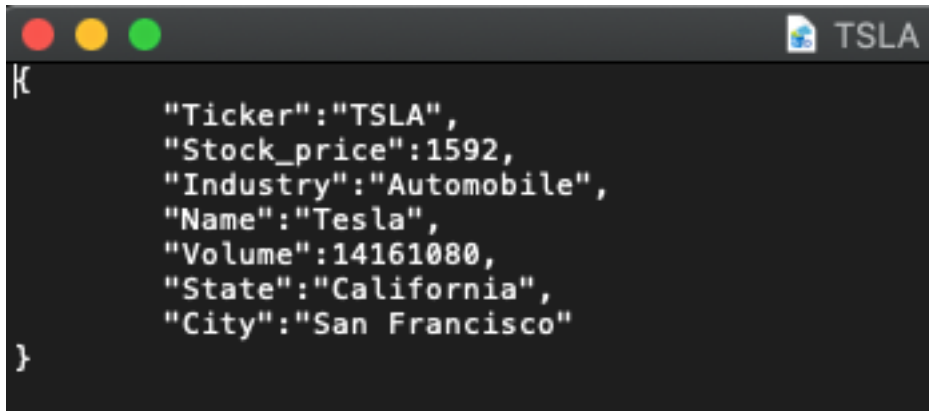
Policy name	Policy type
AmazonS3FullAccess	AWS managed policy
AmazonDynamoDBFullAccess	AWS managed policy
CloudWatchLogsFullAccess	AWS managed policy

2. The screenshot of the final working lambda function. (20%)

The screenshot shows the AWS Lambda console for the function 'medagiri-hw10'. The function code is displayed in a text editor. The code uses boto3 to interact with S3 and DynamoDB. It reads a file from S3, loads it as JSON, and puts it into a DynamoDB table.

```
1 import boto3
2 import json
3 s3_client=boto3.client('s3')
4 dynamodb=boto3.resource('dynamodb')
5 def lambda_handler(event,context):
6     bucket=event['Records'][0]['s3']['bucket']['name']
7     json_file_name =event['Records'][0]['s3']['object']['key']
8     json_object =s3_client.get_object(Bucket=bucket,Key=json_file_name)
9     jsonFileReader =json_object['Body'].read()
10    jsonDict=json.loads(jsonFileReader)
11    table=dynamodb.Table('medagiri-hw10-dynamodb')
12    table.put_item(Item=jsonDict)
13    return'completed'
14
```

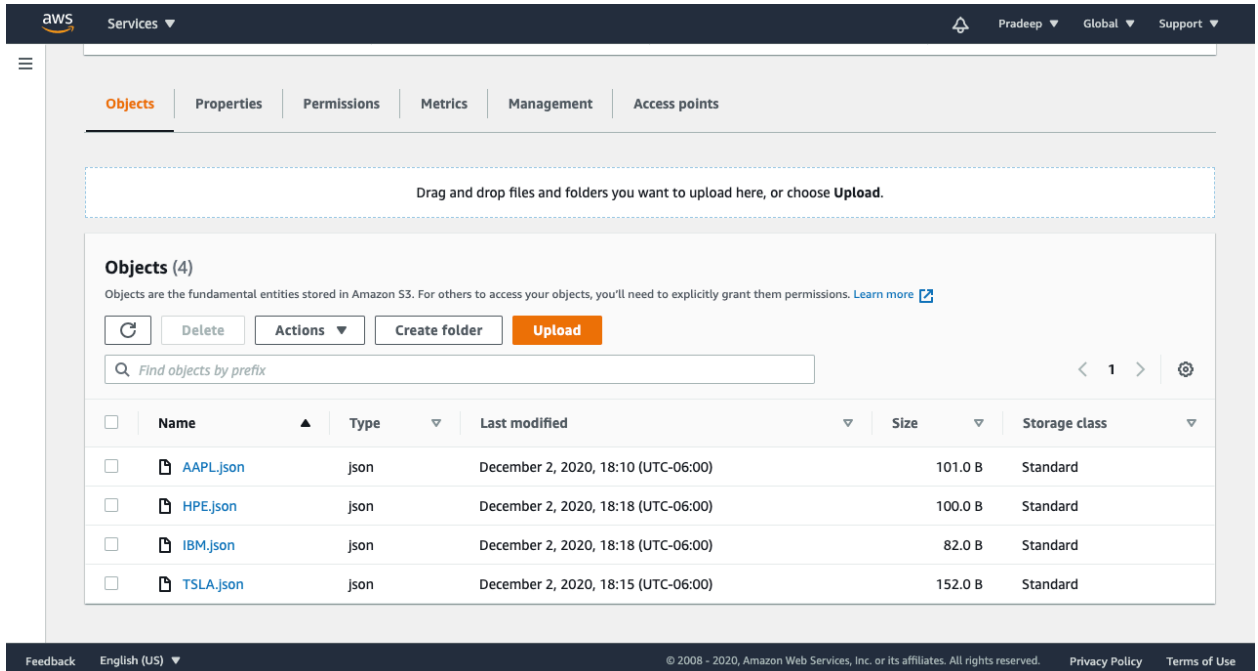
3. The screenshot of Tesla json file. (16%)



A screenshot of a code editor window titled "TSLA". The editor displays a JSON object with the following fields: "Ticker": "TSLA", "Stock_price": 1592, "Industry": "Automobile", "Name": "Tesla", "Volume": 14161080, "State": "California", and "City": "San Francisco". The code is formatted with syntax highlighting and line numbers.

```
{
  "Ticker": "TSLA",
  "Stock_price": 1592,
  "Industry": "Automobile",
  "Name": "Tesla",
  "Volume": 14161080,
  "State": "California",
  "City": "San Francisco"
}
```

4. The screenshot of the S3 bucket with all the json files uploaded. (16%)



A screenshot of the AWS S3 console interface. The top navigation bar shows the AWS logo, "Services", and user information. The main content area has tabs for "Objects", "Properties", "Permissions", "Metrics", "Management", and "Access points". A message prompts the user to "Drag and drop files and folders you want to upload here, or choose Upload." Below this, the "Objects (4)" section shows a list of four JSON files: AAPL.json, HPE.json, IBM.json, and TSLA.json. Each file entry includes a checkbox, a document icon, the filename, type (json), last modified date, size, and storage class (Standard). At the bottom, there is a footer with "Feedback", "English (US)", copyright information, "Privacy Policy", and "Terms of Use".

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	AAPL.json	json	December 2, 2020, 18:10 (UTC-06:00)	101.0 B	Standard
<input type="checkbox"/>	HPE.json	json	December 2, 2020, 18:18 (UTC-06:00)	100.0 B	Standard
<input type="checkbox"/>	IBM.json	json	December 2, 2020, 18:18 (UTC-06:00)	82.0 B	Standard
<input type="checkbox"/>	TSLA.json	json	December 2, 2020, 18:15 (UTC-06:00)	152.0 B	Standard

5. The screenshot CloudWatch log groups for this lambda function that shows no errors (after all 4 json files have been uploaded). (16%)

The image displays two screenshots of the AWS CloudWatch console, showing log events for a Lambda function. The top screenshot shows a successful request and a report. The bottom screenshot shows a successful request and a report, with a 'Copy' button visible.

Top Screenshot:

- Log events:** View as text, Actions, Create Metric Filter
- Filter events:** Clear, 1m, 30m, 1h, 12h, Custom
- Log events table:**

Timestamp	Message
No older events at this moment. Retry	
2020-12-02T18:15:54.309-06:00	START RequestId: ee9956aa-e651-47bb-a821-330533a87d86 Version: \$LATEST
2020-12-02T18:15:54.542-06:00	[ERROR] JSONDecodeError: Expecting property name enclosed in double quotes: line 5 colu...
2020-12-02T18:15:54.563-06:00	END RequestId: ee9956aa-e651-47bb-a821-330533a87d86
2020-12-02T18:15:54.563-06:00	REPORT RequestId: ee9956aa-e651-47bb-a821-330533a87d86 Duration: 253.91 ms Billed Durat...
2020-12-02T18:18:15.732-06:00	START RequestId: ff2bbd23-9c07-4d0b-9b48-bdda42234abb Version: \$LATEST
START RequestId: ff2bbd23-9c07-4d0b-9b48-bdda42234abb Version: \$LATEST	
2020-12-02T18:18:16.103-06:00	END RequestId: ff2bbd23-9c07-4d0b-9b48-bdda42234abb
END RequestId: ff2bbd23-9c07-4d0b-9b48-bdda42234abb	
2020-12-02T18:18:16.103-06:00	REPORT RequestId: ff2bbd23-9c07-4d0b-9b48-bdda42234abb Duration: 368.60 ms Billed Durat...
REPORT RequestId: ff2bbd23-9c07-4d0b-9b48-bdda42234abb Duration: 368.60 ms Billed Duration: 369 ms Memory Size: 128 MB Max Memory Used: 81 MB	
No newer events at this moment. Auto retry paused. Resume	

Bottom Screenshot:

- Log events:** View as text, Actions, Create Metric Filter
- Filter events:** Clear, 1m, 30m, 1h, 12h, Custom
- Log events table:**

Timestamp	Message
2020-12-02T18:16:53.047-06:00	REPORT RequestId: 348ffae2-4dd4-417c-8915-51c820666885 Duration: 25.02 ms Billed Durat...
2020-12-02T18:18:15.711-06:00	START RequestId: 0c00fb88-91d9-4bbb-8e61-e5ccc3d140ff Version: \$LATEST
START RequestId: 0c00fb88-91d9-4bbb-8e61-e5ccc3d140ff Version: \$LATEST	
2020-12-02T18:18:16.148-06:00	END RequestId: 0c00fb88-91d9-4bbb-8e61-e5ccc3d140ff
END RequestId: 0c00fb88-91d9-4bbb-8e61-e5ccc3d140ff	
2020-12-02T18:18:16.148-06:00	REPORT RequestId: 0c00fb88-91d9-4bbb-8e61-e5ccc3d140ff Duration: 433.42 ms Billed Durat...
2020-12-02T18:18:37.047-06:00	START RequestId: ee9956aa-e651-47bb-a821-330533a87d86 Version: \$LATEST
START RequestId: ee9956aa-e651-47bb-a821-330533a87d86 Version: \$LATEST	
2020-12-02T18:18:37.288-06:00	END RequestId: ee9956aa-e651-47bb-a821-330533a87d86
END RequestId: ee9956aa-e651-47bb-a821-330533a87d86	
2020-12-02T18:18:37.288-06:00	REPORT RequestId: ee9956aa-e651-47bb-a821-330533a87d86 Duration: 238.79 ms Billed Durat...
2020-12-02T18:18:43.229-06:00	START RequestId: 348ffae2-4dd4-417c-8915-51c820666885 Version: \$LATEST
START RequestId: 348ffae2-4dd4-417c-8915-51c820666885 Version: \$LATEST	
2020-12-02T18:18:43.448-06:00	END RequestId: 348ffae2-4dd4-417c-8915-51c820666885
END RequestId: 348ffae2-4dd4-417c-8915-51c820666885	
2020-12-02T18:18:43.448-06:00	REPORT RequestId: 348ffae2-4dd4-417c-8915-51c820666885 Duration: 216.36 ms Billed Durat...
No newer events at this moment. Auto retry paused. Resume	

6. The screenshot of the items in the **dynamodb** table (after all 4 json files have been uploaded). (16%)

The screenshot shows the AWS Management Console interface for the 'medagiri-hw10-dynamodb' table. The left sidebar contains navigation links for DynamoDB, Tables, Backups, Reserved capacity, Preferences, DAX, Clusters, Subnet groups, Parameter groups, and Events. The main content area displays the 'Items' tab for the table. A search bar is present at the top of the items list. Below the search bar, there is a 'Scan' section with a dropdown menu set to '[Table] medagiri-hw10-dynamodb: Ticker' and a 'Start search' button. The items are displayed in a table with columns: Ticker, Name, Stock_price, Industry, and Volume. The items are: AAPL (Apple, 389, Computer, 20279276), HPE (Hewlett Packard Enterprise, 10, Computer), IBM (International Business Machines, 129), and TSLA (Tesla, 1592, Automobile, 14161080).

Ticker	Name	Stock_price	Industry	Volume
AAPL	Apple	389	Computer	20279276
HPE	Hewlett Packard Enterprise	10	Computer	
IBM	International Business Machines	129		
TSLA	Tesla	1592	Automobile	14161080