

20CS3234AA – ADC

Name: B. Naveen

Reg.no: 2000031509

Experiments

Create one AWS Lambda function which can print the success message in CloudWatch after inserting data into DynamoDB and that lambda function need to be triggered by the Restful API Gateway.

The API Gateway need to be authorized by the token value of Amazon Cognito Login Page

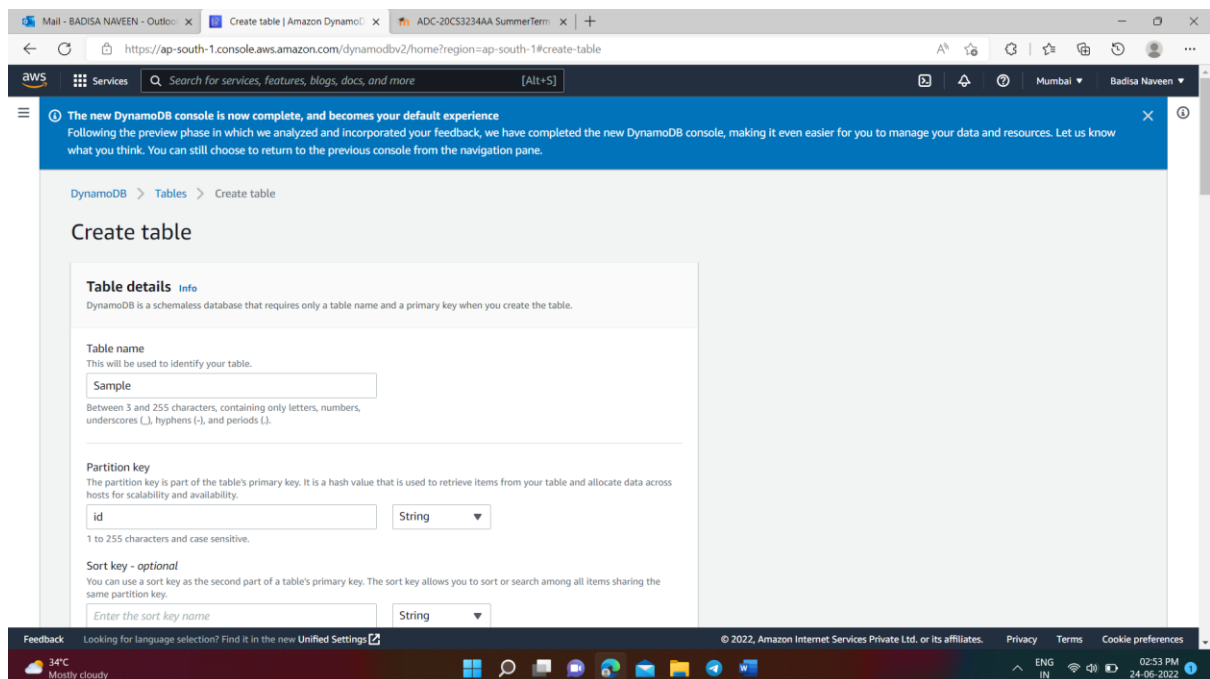
If request header has the proper token vale, then allow the lambda to display the success message and insert the data into DynamoDB. The success message need to be printed in Postman App and as well as Amazon CloudWatch.

If the request header is failed to provide the token value then display unauthorized

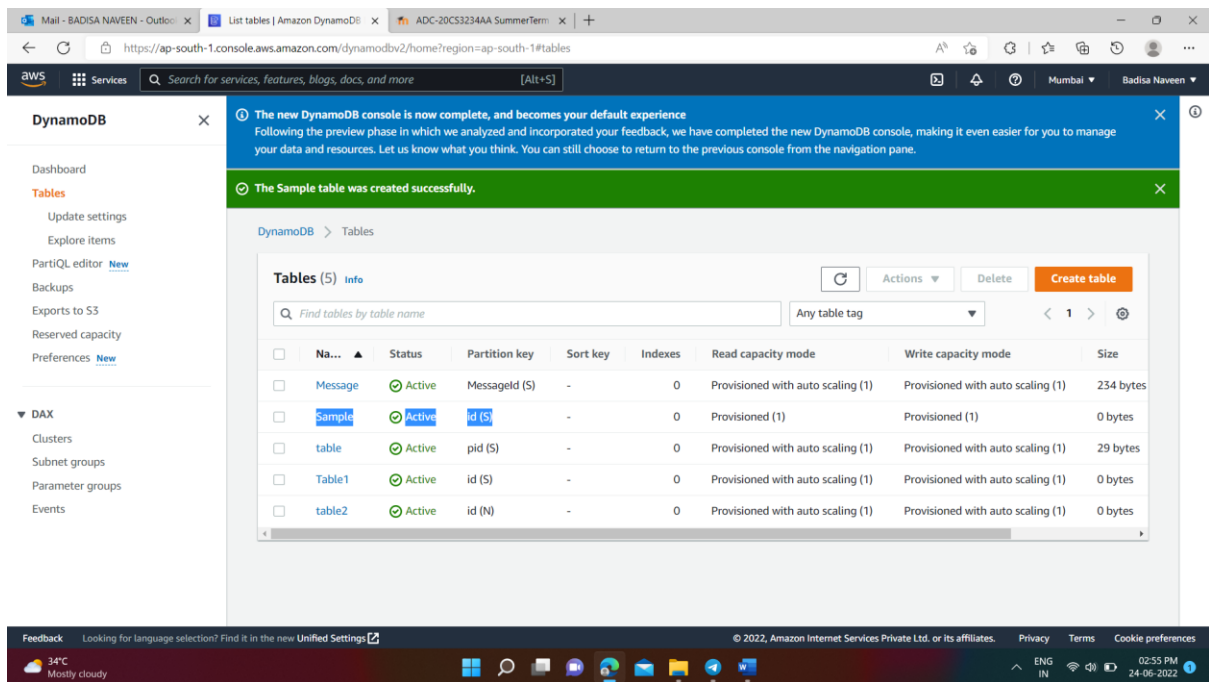
You can use Postman app (to set the header token), to display the output instead of the browser

A.

Step1: Create a table in Dynamodb



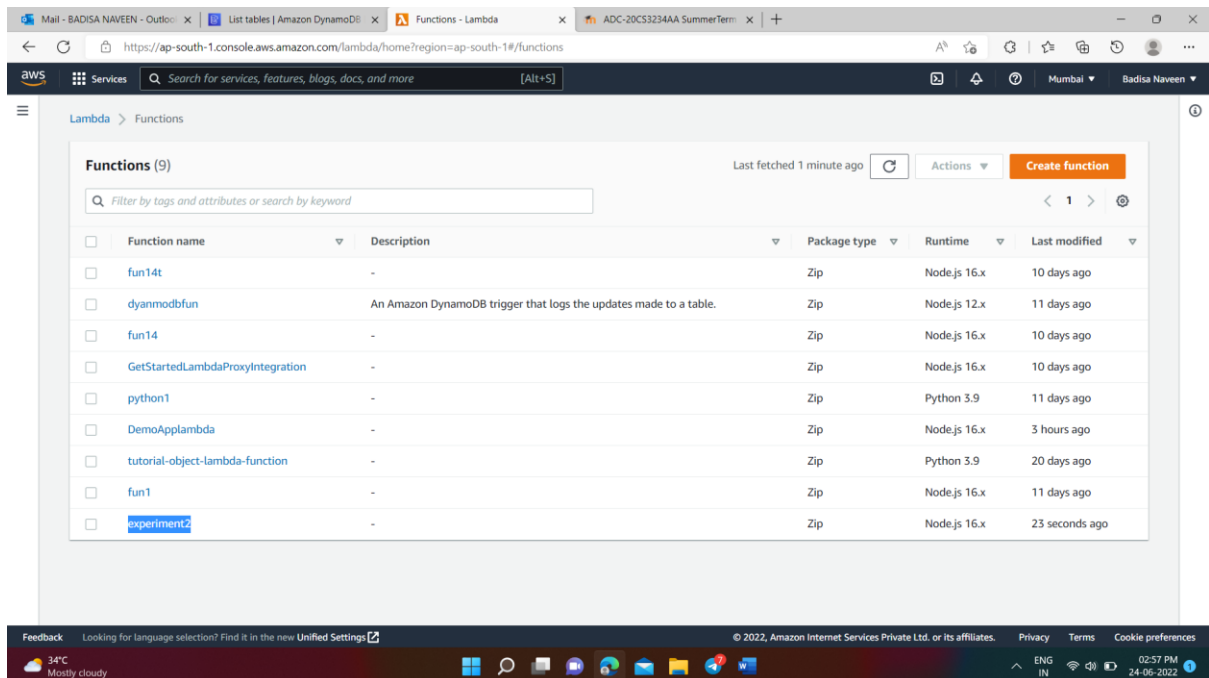
Step2: Dynamodb Table created



The screenshot shows the AWS DynamoDB console interface. At the top, there are two notification banners: a blue one stating 'The new DynamoDB console is now complete, and becomes your default experience' and a green one stating 'The Sample table was created successfully.' The left sidebar contains navigation links for Dashboard, Tables, Update settings, Explore items, PartiQL editor, Backups, Exports to S3, Reserved capacity, and Preferences. The main content area displays a table titled 'Tables (5)' with columns: Name, Status, Partition key, Sort key, Indexes, Read capacity mode, Write capacity mode, and Size. The table lists five tables: 'Message', 'Sample', 'table', 'Table1', and 'table2'. The 'Sample' table is highlighted. The bottom of the screen shows a Windows taskbar with the date and time as 02:55 PM on 24-06-2022.

	Name	Status	Partition key	Sort key	Indexes	Read capacity mode	Write capacity mode	Size
<input type="checkbox"/>	Message	Active	MessageId (S)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	234 bytes
<input type="checkbox"/>	Sample	Active	id (S)	-	0	Provisioned (1)	Provisioned (1)	0 bytes
<input type="checkbox"/>	table	Active	pid (S)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	29 bytes
<input type="checkbox"/>	Table1	Active	id (S)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	0 bytes
<input type="checkbox"/>	table2	Active	id (N)	-	0	Provisioned with auto scaling (1)	Provisioned with auto scaling (1)	0 bytes

Step3: Created a lambda function



The screenshot shows the AWS Lambda console interface. The top navigation bar includes a search bar and a 'Create function' button. The main content area displays a table titled 'Functions (9)' with columns: Function name, Description, Package type, Runtime, and Last modified. The table lists nine functions: 'fun14t', 'dyanmodbfun', 'fun14', 'GetStartedLambdaProxyIntegration', 'python1', 'DemoApplambda', 'tutorial-object-lambda-function', 'fun1', and 'experiment2'. The 'experiment2' function is highlighted. The bottom of the screen shows a Windows taskbar with the date and time as 02:57 PM on 24-06-2022.

	Function name	Description	Package type	Runtime	Last modified
<input type="checkbox"/>	fun14t	-	Zip	Node.js 16.x	10 days ago
<input type="checkbox"/>	dyanmodbfun	An Amazon DynamoDB trigger that logs the updates made to a table.	Zip	Node.js 12.x	11 days ago
<input type="checkbox"/>	fun14	-	Zip	Node.js 16.x	10 days ago
<input type="checkbox"/>	GetStartedLambdaProxyIntegration	-	Zip	Node.js 16.x	10 days ago
<input type="checkbox"/>	python1	-	Zip	Python 3.9	11 days ago
<input type="checkbox"/>	DemoApplambda	-	Zip	Node.js 16.x	3 hours ago
<input type="checkbox"/>	tutorial-object-lambda-function	-	Zip	Python 3.9	20 days ago
<input type="checkbox"/>	fun1	-	Zip	Node.js 16.x	11 days ago
<input type="checkbox"/>	experiment2	-	Zip	Node.js 16.x	23 seconds ago

Step4: write the lambda code to write in dynamodb table

```
const AWS=require('aws-sdk');

const ddb=new AWS.DynamoDB.DocumentClient({region:'ap-south-1'});

exports.handler=async(event,context,callback)=>{

    const requestId=context.awsRequestId;

    await createMessage(requestId).then(()=>{

        callback(null,{

            statusCode:201,

            body:"",

            headers:{

                'Access-Control-Allow-Origin': '*'

            }

        })

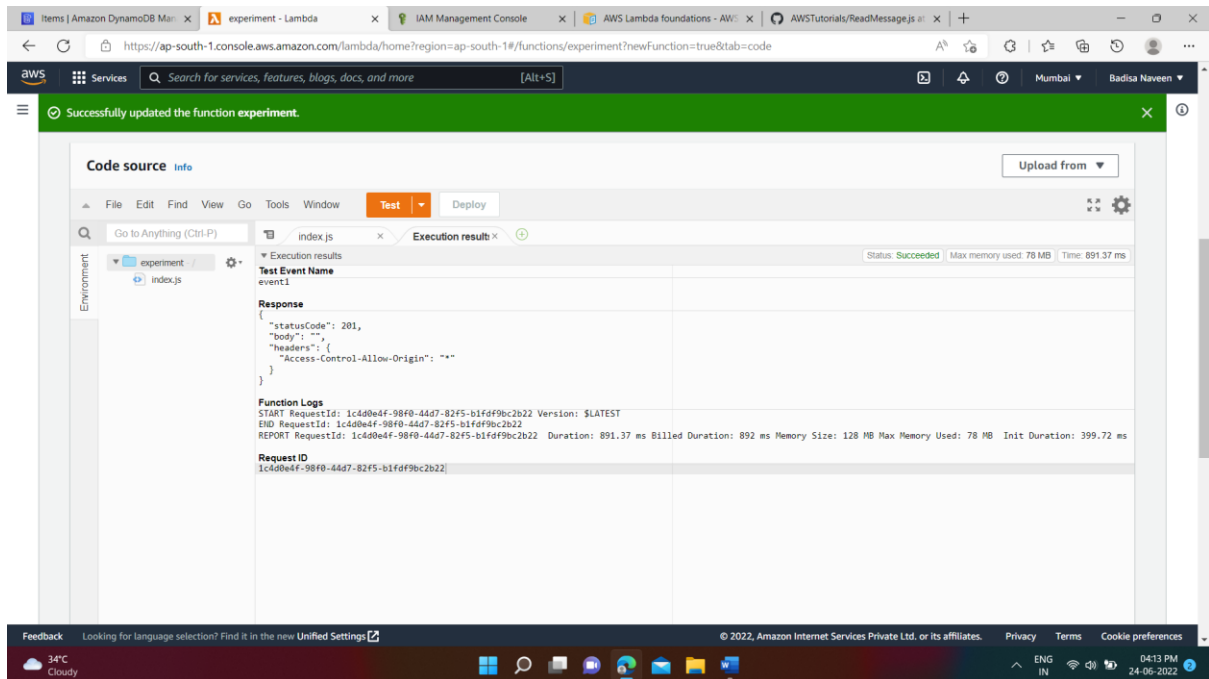
    }).catch((err)=>{

        console.error(err)

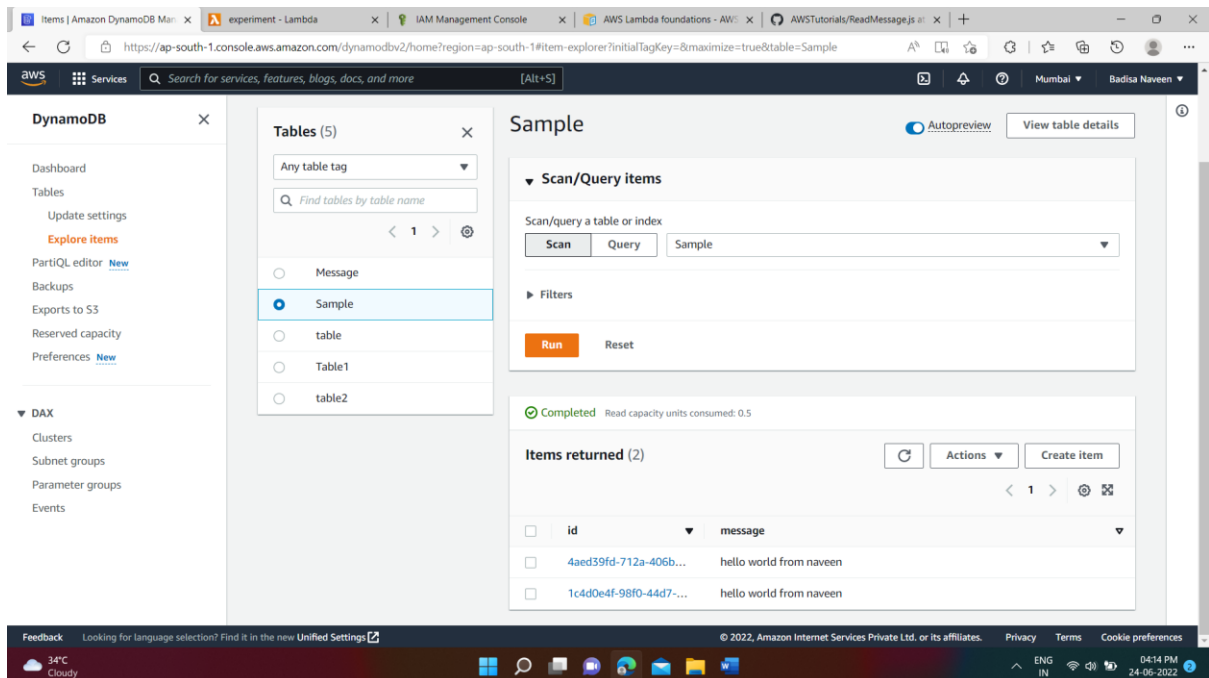
    });

};
```

Step5: click on test



Step6: open dynamodb table and see the data inserted



Step8: Read Message code

```
// Loads in the AWS SDK
```

```
const AWS = require('aws-sdk');
```

```
// Creates the document client specifying the region
```

```
// The tutorial's table is 'in us-east-1'
```

```
const ddb = new AWS.DynamoDB.DocumentClient({region: 'ap-south-1'});
```

```
exports.handler = async (event, context, callback) => {
```

```
    // Handle promise fulfilled/rejected states
```

```
    await readMessage().then(data => {
```

```
        data.Items.forEach(function(item) {
```

```
            console.log(item.message)
```

```
        });
```

```
        callback(null, {
```

```
            // If success return 200, and items
```

```
            statusCode: 200,
```

```
            body: data.Items,
```

```
            headers: {
```

```
                'Access-Control-Allow-Origin': '*',
```

```
            },
```

```
        })
```

```
    }).catch((err) => {
```

```
        // If an error occurs write to the console
```

```
        console.error(err);
```

```

    })

};

// Function readMessage

// Reads 10 messages from the DynamoDb table Message

// Returns promise

function readMessage() {

    const params = {

        TableName: 'Sample',

        Limit: 10

    }

    return ddb.scan(params).promise();

}

```

Output:

The screenshot displays the AWS Lambda console interface. At the top, a green banner indicates "Successfully updated the function experiment." Below this, the "Execution results" tab is selected, showing a successful execution status. The "Test Event Name" is "event1". The "Response" section shows a JSON object with a status code of 200 and a body containing two messages: "hello world from naveen". The "Function Logs" section shows the execution details, including the start and end times, request ID, and memory usage. The "Code properties" section is also visible at the bottom.

Execution results

Status: **Succeeded** | Max memory used: 79 MB | Time: 946.70 ms

A function update is still in progress so the invocation went to the previously deployed code and configuration.

Test Event Name
event1

Response

```
{
  "statusCode": 200,
  "body": [
    {
      "id": "4aed39fd-712a-406b-8e3d-e9b022db6aed",
      "message": "hello world from naveen"
    },
    {
      "id": "1c4d0e4f-98f0-44d7-82f5-b1fd9bc2b22",
      "message": "hello world from naveen"
    }
  ],
  "headers": {
    "Access-Control-Allow-Origin": ""
  }
}
```

Function Logs

```
START RequestId: 2788d0e7-1a0a-44ee-9a5c-02c312858982 Version: $LATEST
2022-06-24T10:48:13.907Z 2788d0e7-1a0a-44ee-9a5c-02c312858982 INFO hello world from naveen
2022-06-24T10:48:13.943Z 2788d0e7-1a0a-44ee-9a5c-02c312858982 INFO hello world from naveen
END RequestId: 2788d0e7-1a0a-44ee-9a5c-02c312858982
REPORT RequestId: 2788d0e7-1a0a-44ee-9a5c-02c312858982 Duration: 946.70 ms Billed Duration: 947 ms Memory Size: 128 MB Max Memory Used: 79 MB Init Duration: 453.72 ms
```

Request ID
2788d0e7-1a0a-44ee-9a5c-02c312858982

Code properties

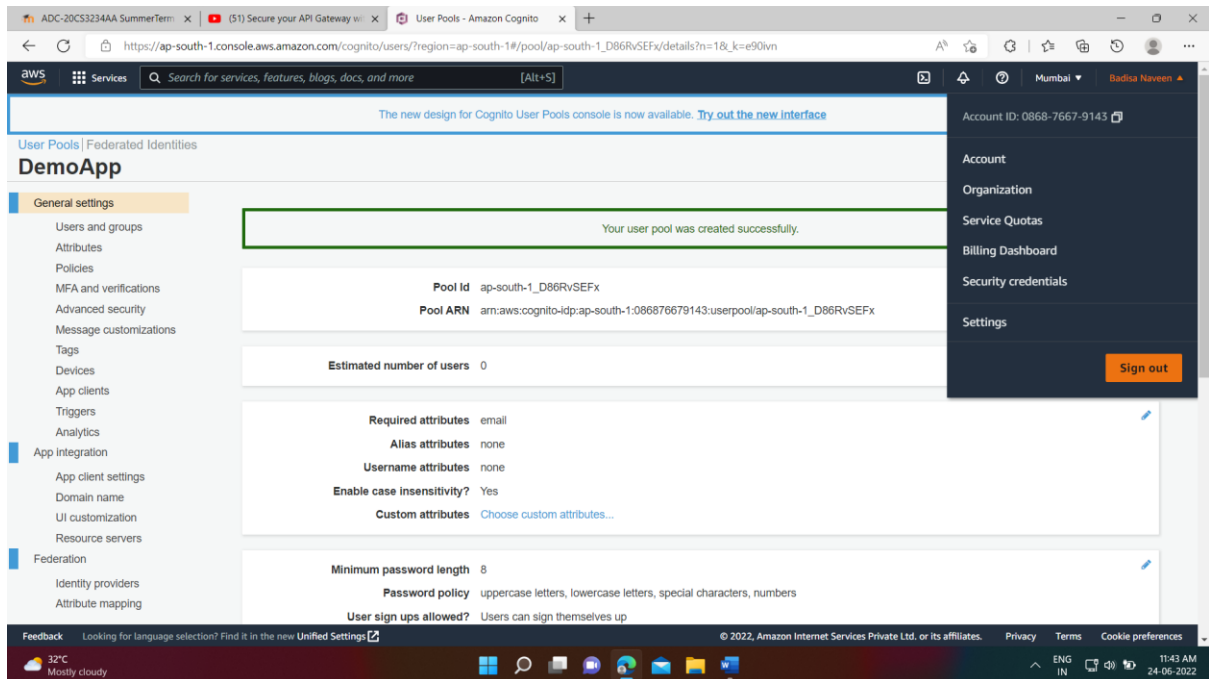
Feedback Looking for language selection? Find it in the new Unified Settings

© 2022, Amazon Internet Services Private Ltd. or its affiliates. Privacy Terms Cookie preferences

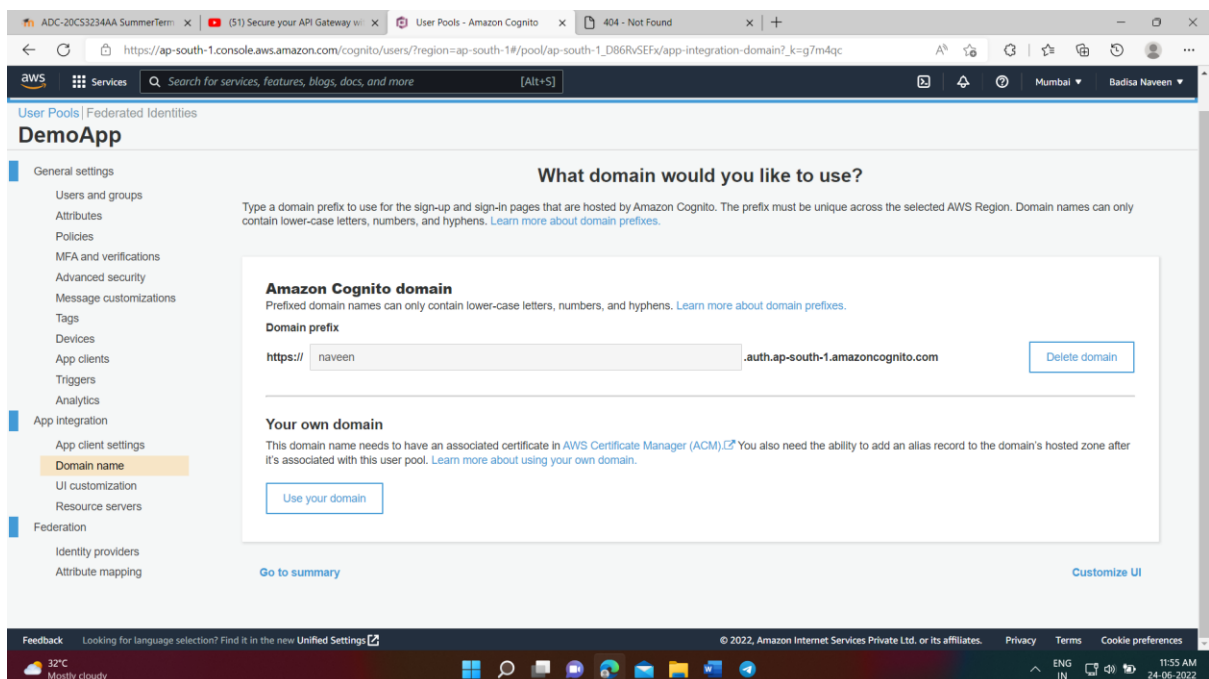
34°C Cloudy 04:18 PM 24-06-2022

AWS COGNITO

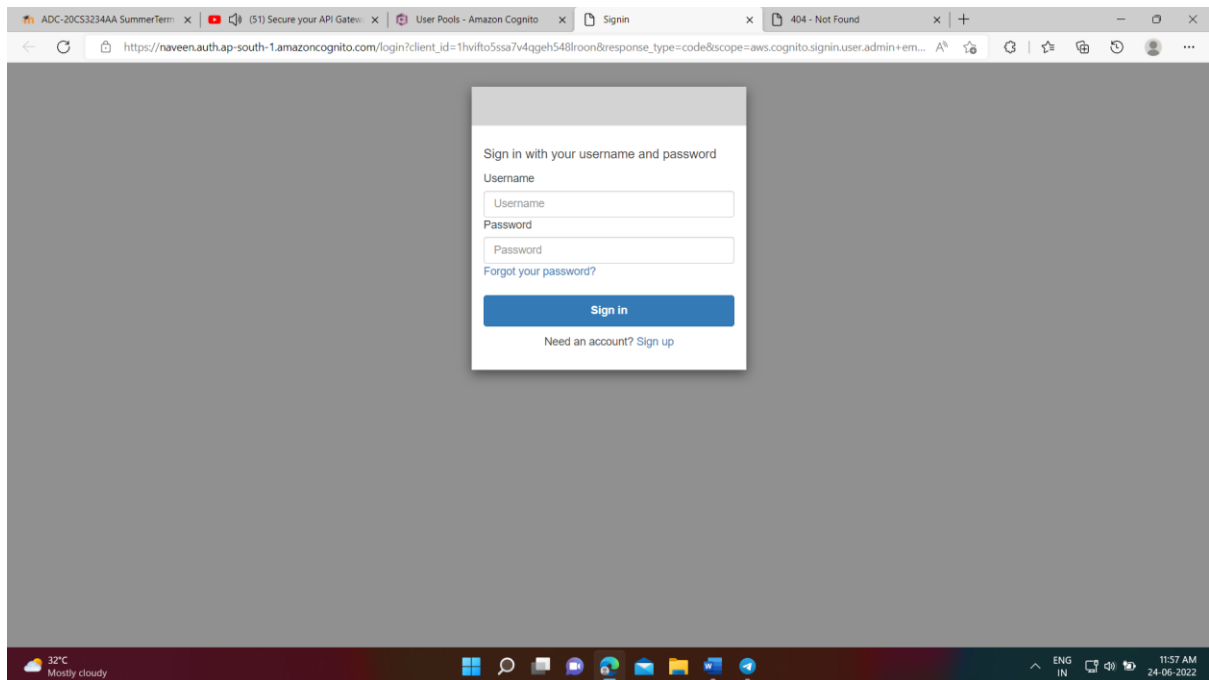
Step1: Open and create the Amazon cognito by creating user pool



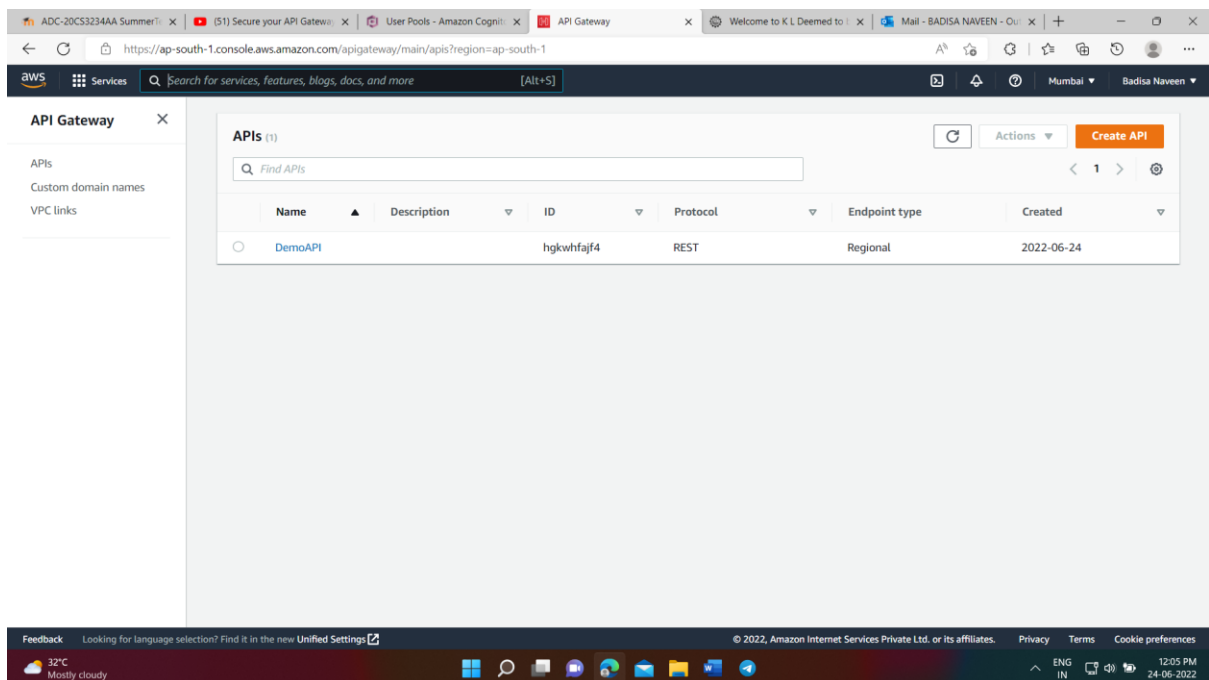
Step2: create a Domain Name



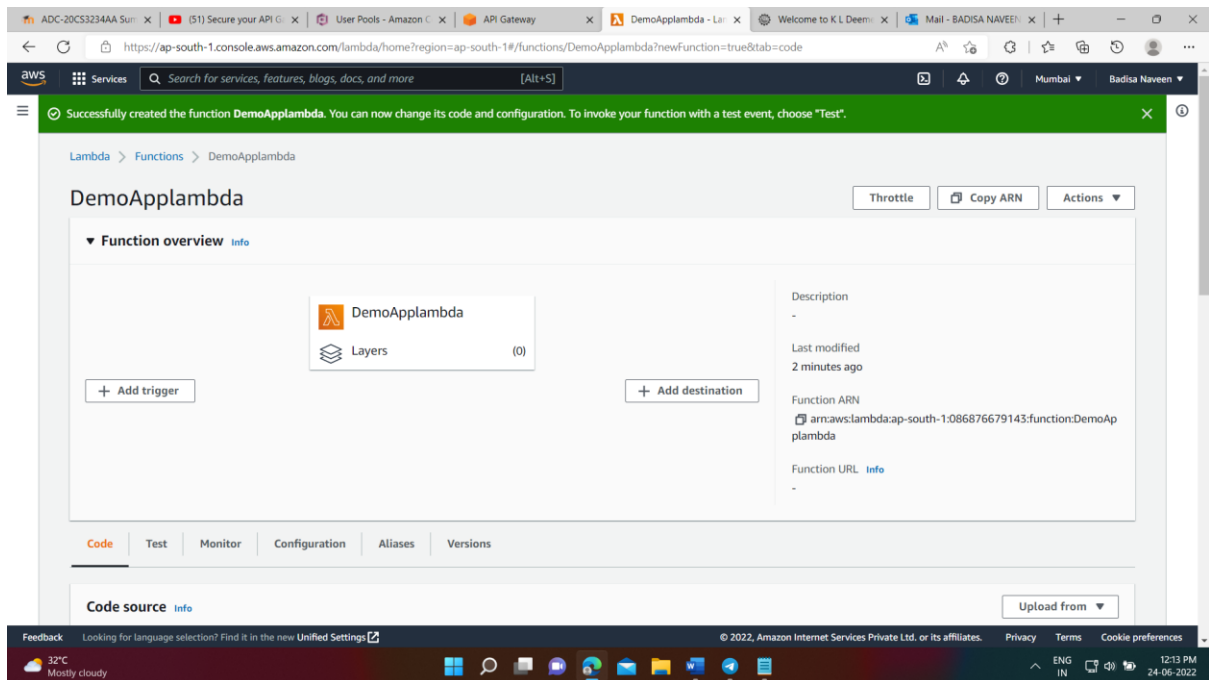
Step3: Signup and login



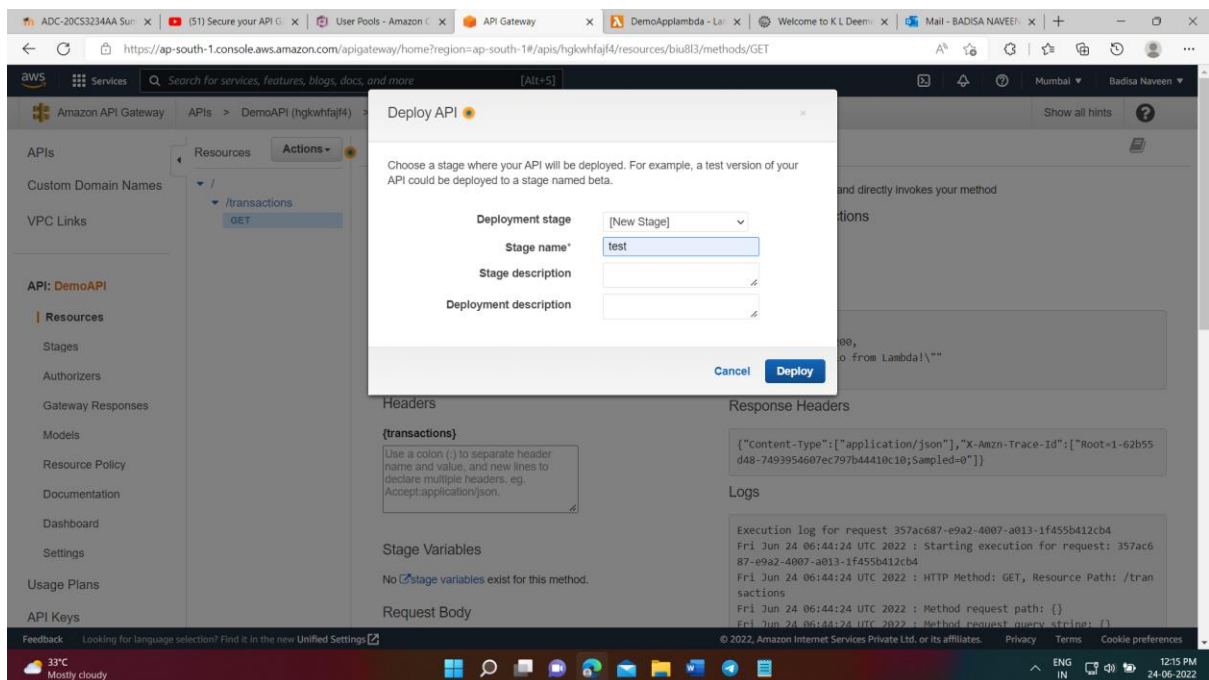
Step4: create API Gateway by choosing REST API



Step5: create Lambda function

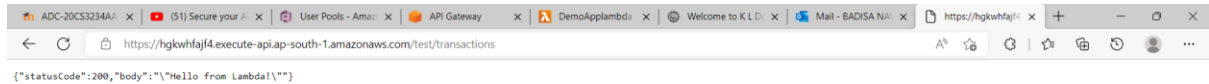


Step6: Deploy API

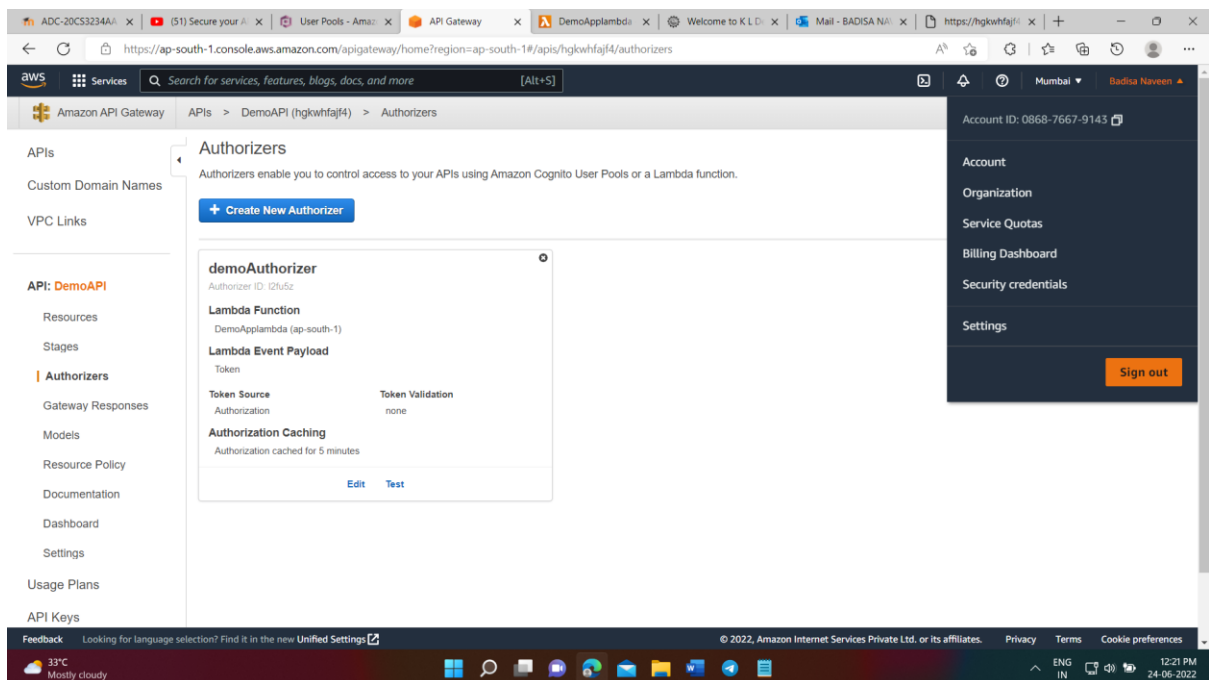


Step7: Open invoke url

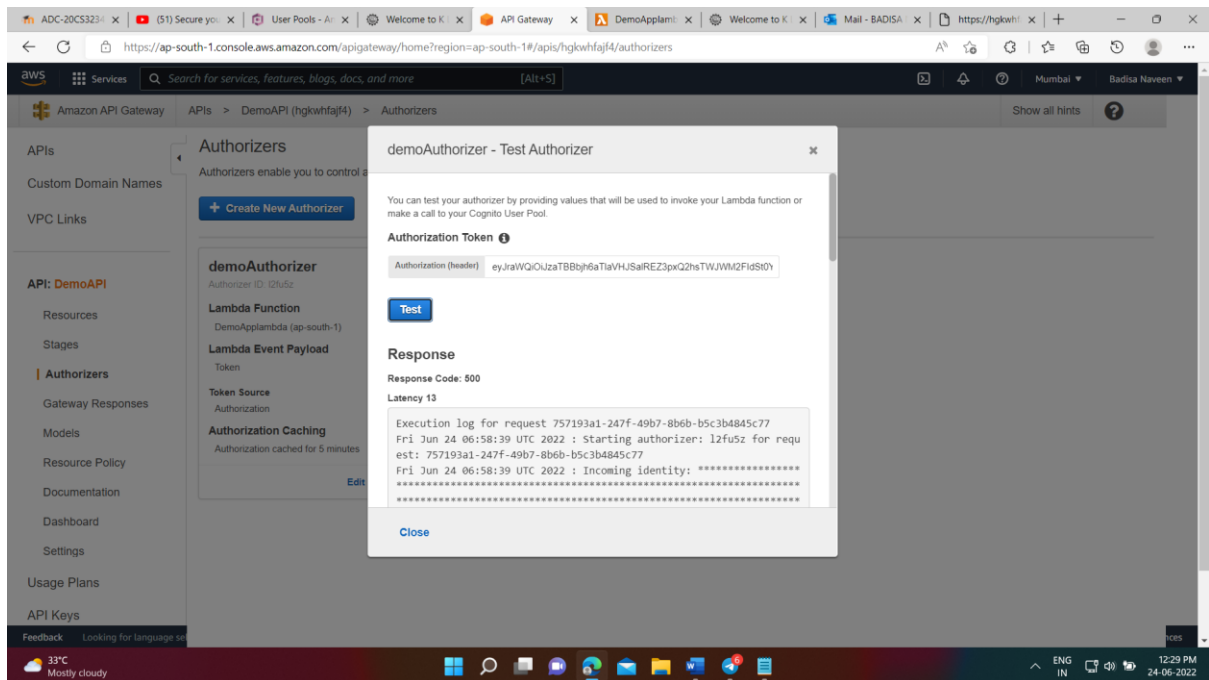
<https://hgkwhfajf4.execute-api.ap-south-1.amazonaws.com/test/transactions>



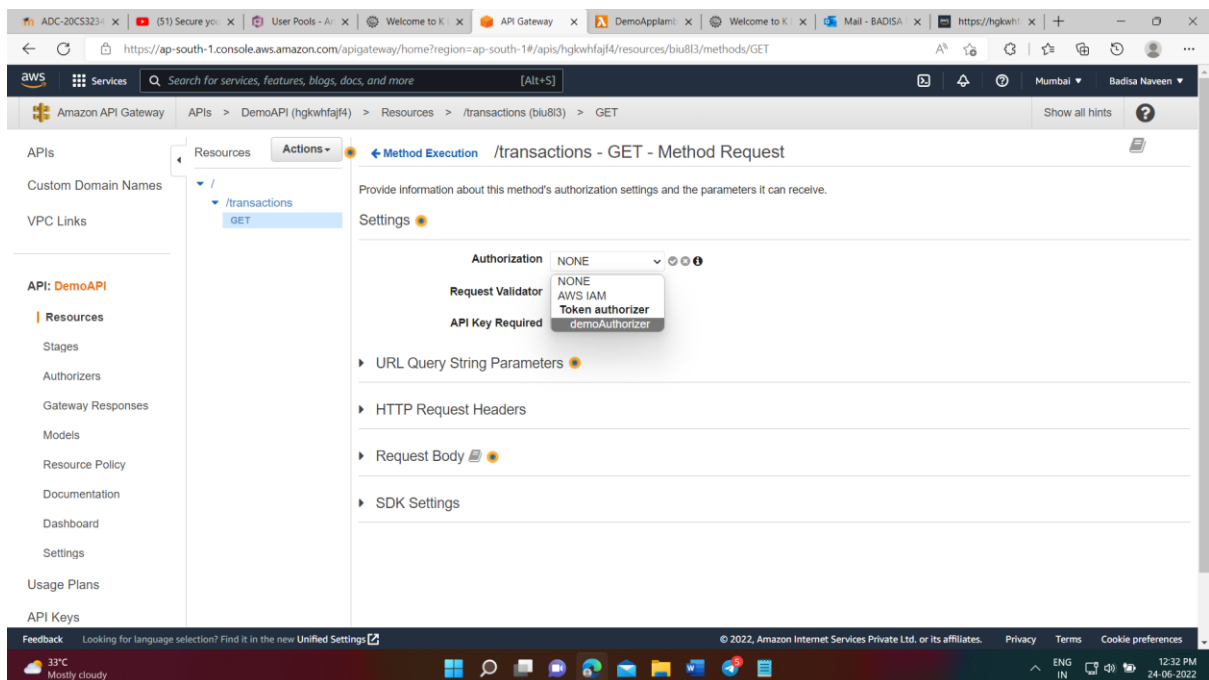
Step8: create Authorization



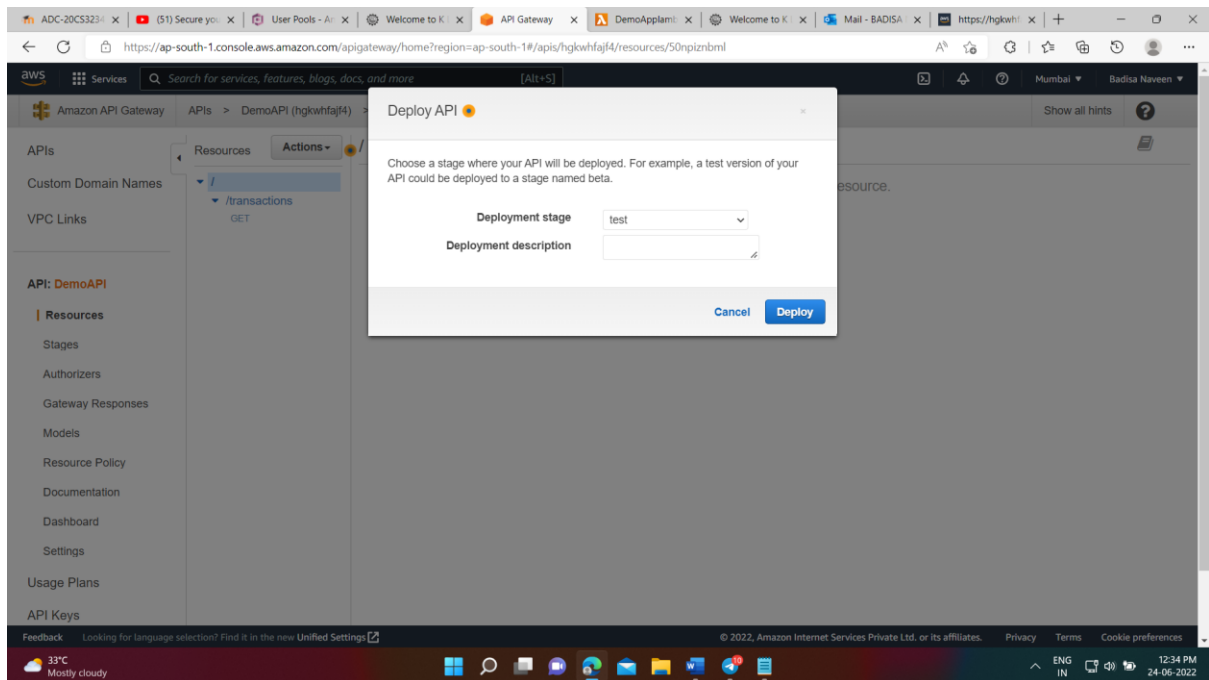
Step9: Test the Authorize token



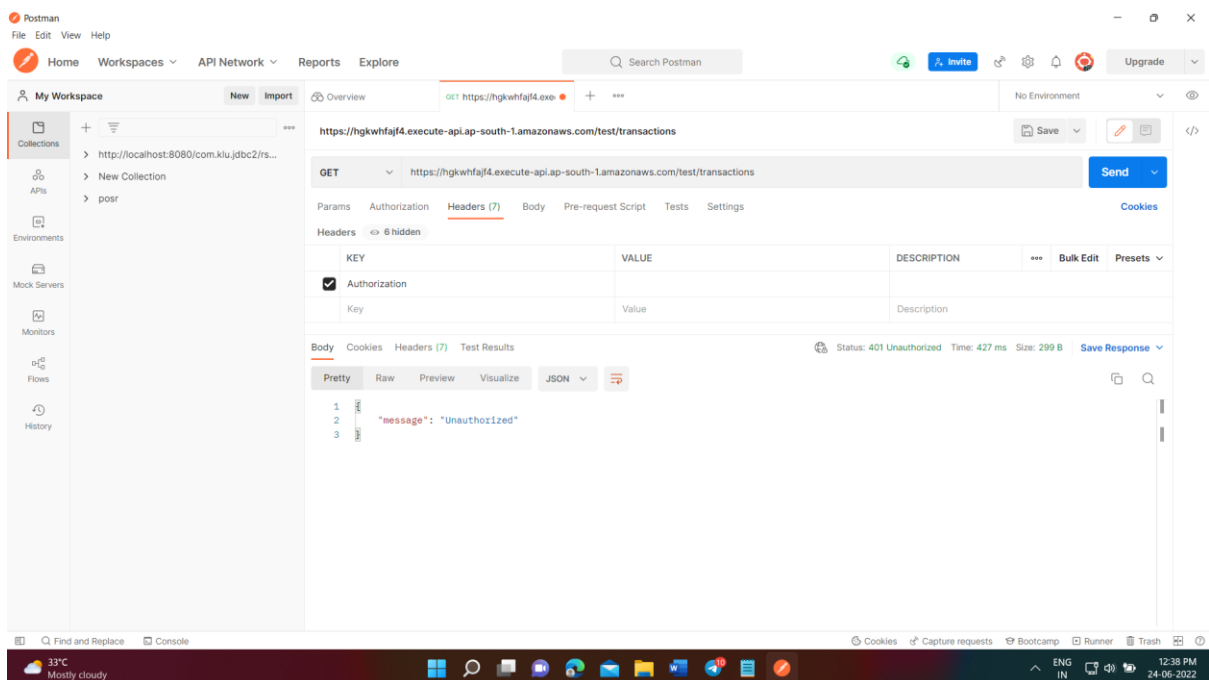
Step10: change the Authorization



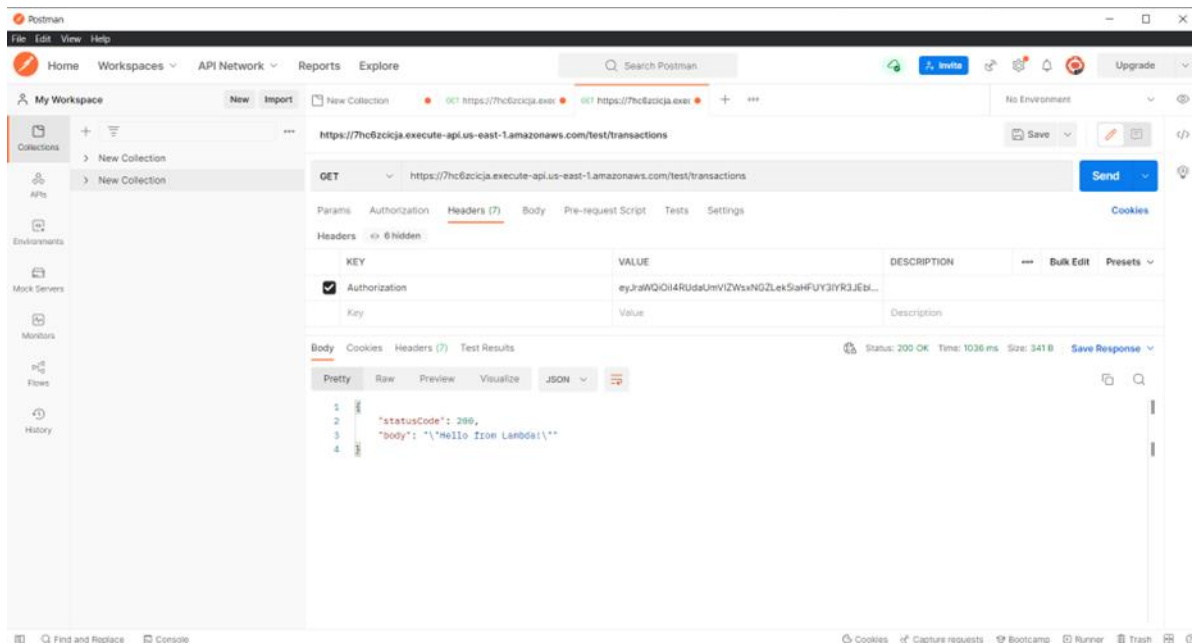
Step11: Deploy API



Step12: Open postman and paste the url



Step13: change the Authorization and provide key



Conclusion:

1. We can write and read data from Dynamodb using AWS lambda
2. AWS Cognito provides Authentication for WebApplications
3. I used Postman for setting header token and to display the output instead of the browser