Building a serverless - web application on Amazon web services to GET details from Dynamo DB

This article delineates the detailed process of building a serverless web application that retrieves data from DynamoDB tables using AWS API gateway, Amazon S3, DynamoDB, and the reliable serverless service lambda.

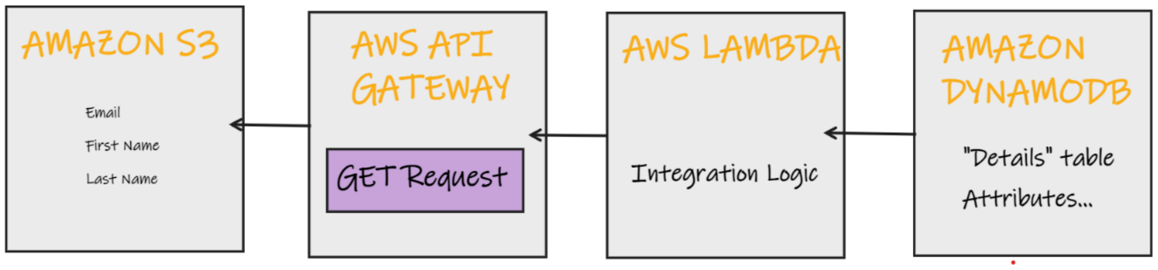
ARCHITECTURE OF THE PROJECT is as follows.

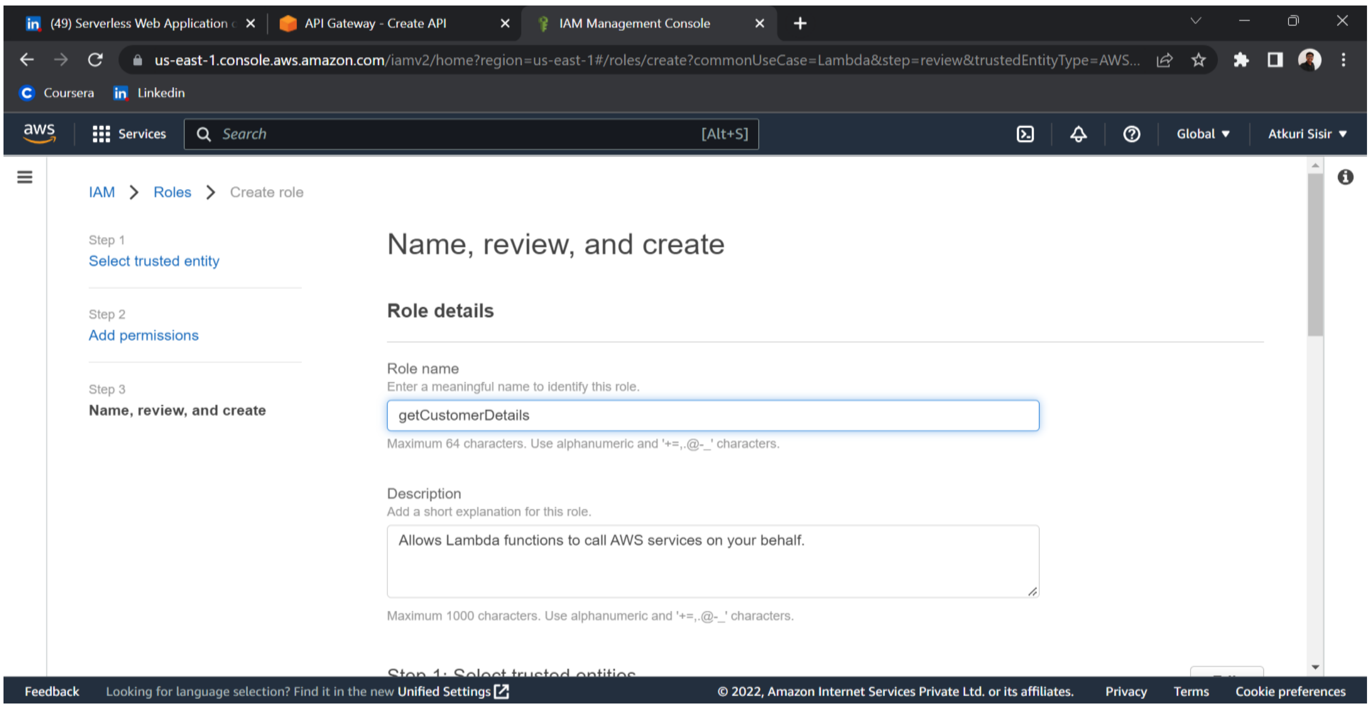
NOTE: - You need a basic free-tier root-use account to begin the project.

Open AWS Console.

1. IAM ROLE

Create a with the required name and description and attach two policies namely AmazonDynamoDBReadOnlyAccess and AWSLambda_FullAccess.



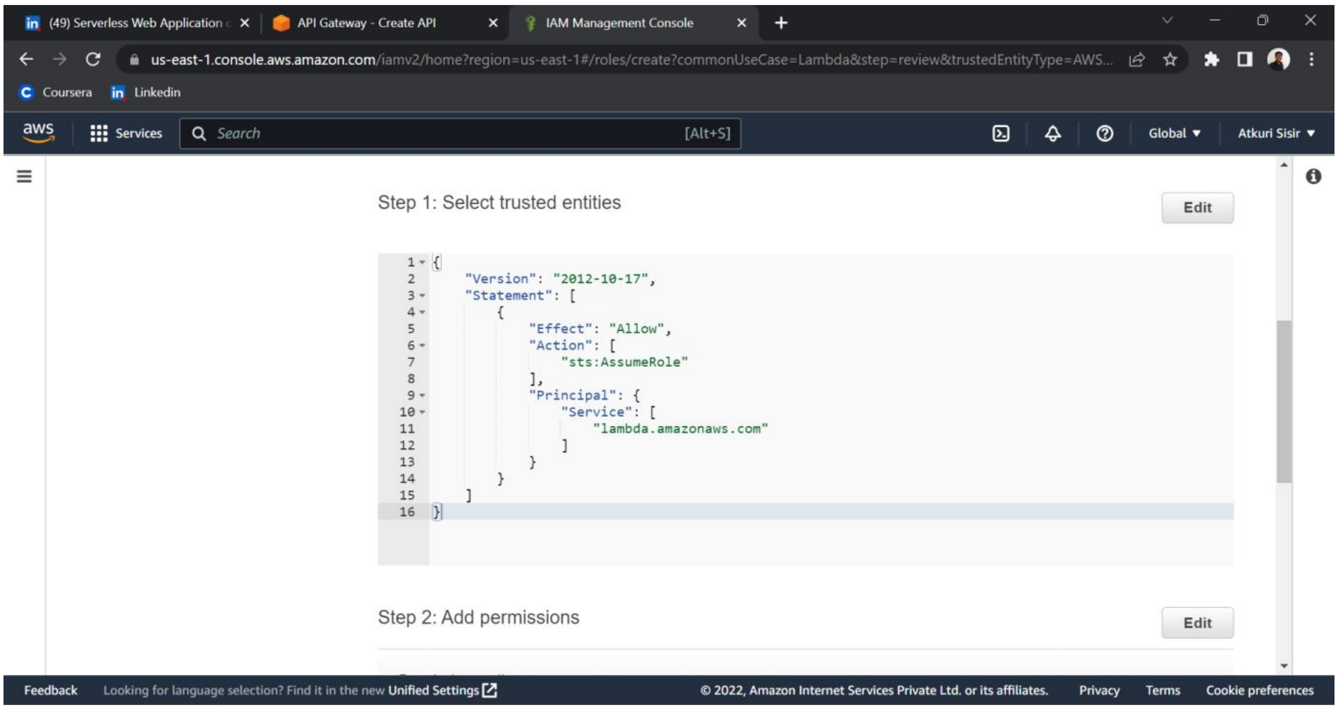

(49) Serverless Web Application
API Gateway - Create API
IAM Management Console
us-east-1.console.aws.amazon.com/iamv2/home?region=us-east-1#/roles/create?commonUseCase=Lambda&step=review&trustedEntityType=AWS...
Coursera
Linkedin
Services
Search
[Alt+S]
Global
Atkuri Sisir
IAM
Roles
Create role
Step 1
Select trusted entity
Step 2
Add permissions
Step 3
Name, review, and create
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

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.
getCustomerDetails
Maximum 64 characters. Use alphanumeric and '+=,.@-_' characters.

Description
Add a short explanation for this role.
Allows Lambda functions to call AWS services on your behalf.
Maximum 1000 characters. Use alphanumeric and '+=,.@-_' characters.


Step 1: Select trusted entities

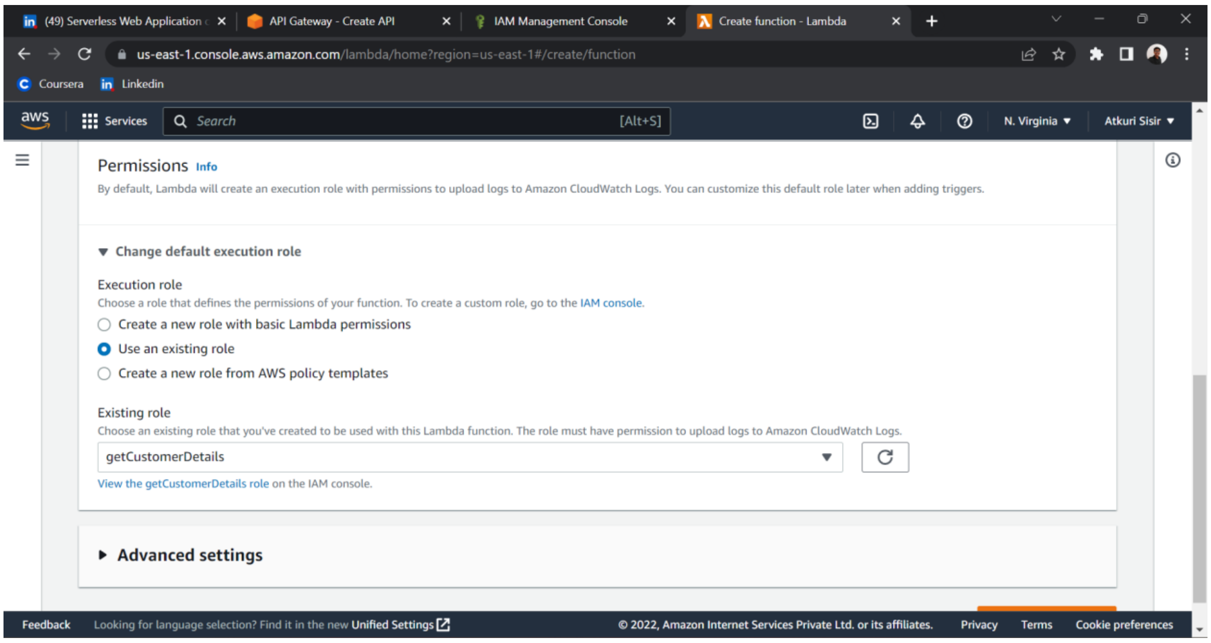
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "lambda.amazonaws.com"
                ]
            }
        }
    ]
}
```

Step 2: Add permissions

2. AWS LAMBDA

Create a lambda function with the required name and add the below code in the source code section in order to integrate the lambda function with the DynamoDB table that is going to be created in the coming step.

In the Permissions section, use the existing role that was created just now.


Permissions Info

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

○ Create a new role with basic Lambda permissions

● Use an existing role

○ Create a new role from AWS policy templates

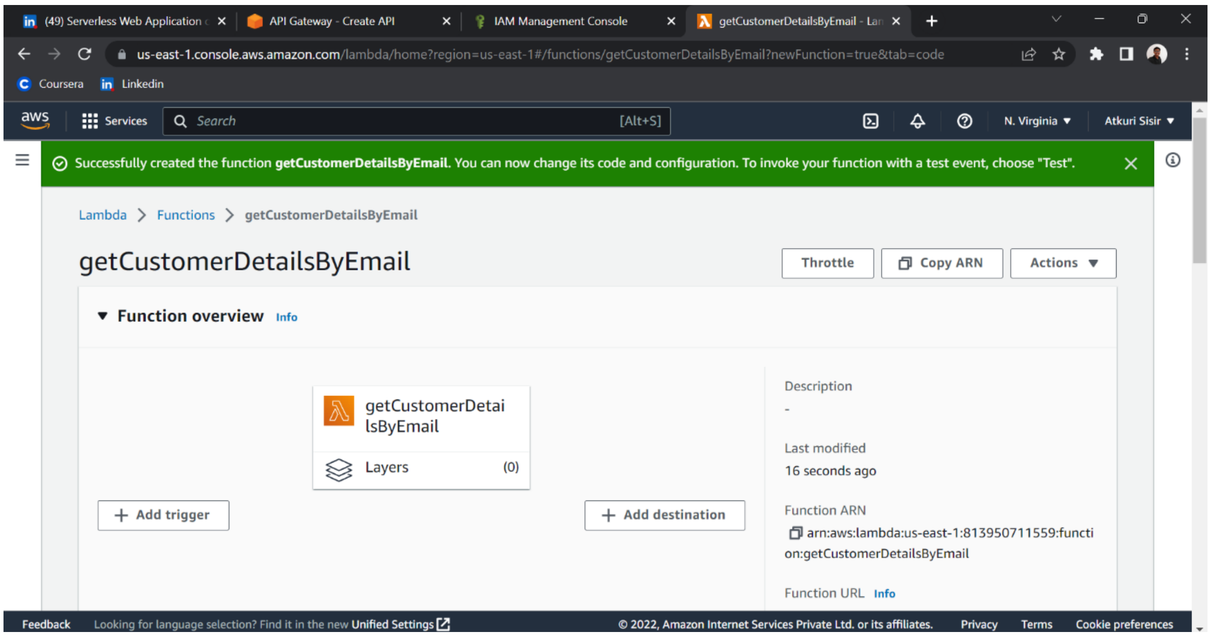
Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

getCustomerDetails

View the getCustomerDetails role on the IAM console.

▶ Advanced settings


Successfully created the function **getCustomerDetailsByEmail**. You can now change its code and configuration. To invoke your function with a test event, choose "Test".

Lambda > Functions > getCustomerDetailsByEmail

getCustomerDetailsByEmail

Function overview

Description
-

Last modified
16 seconds ago

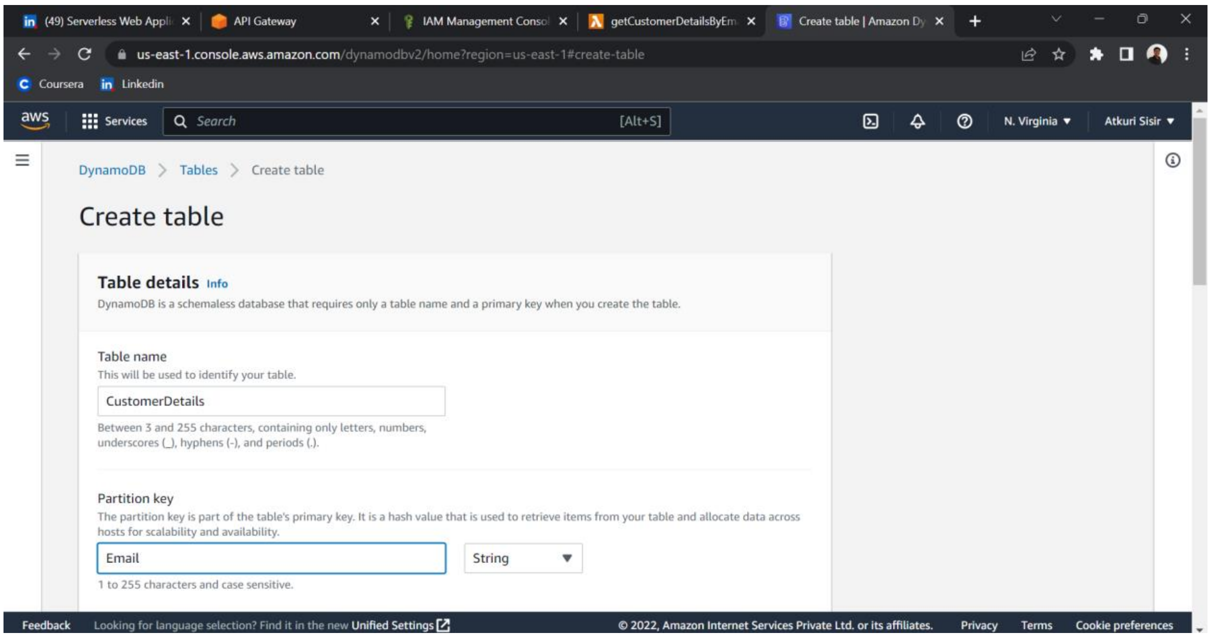
Function ARN
arn:aws:lambda:us-east-1:813950711559:function:getCustomerDetailsByEmail

Function URL

3. DynamoDB

Create a DynamoDB table with the required name and add a partition key with your email as a value.

Create one or more items with attributes like email, first name, and last name. All attributes are considered string parameters.


DynamoDB > Tables > Create table

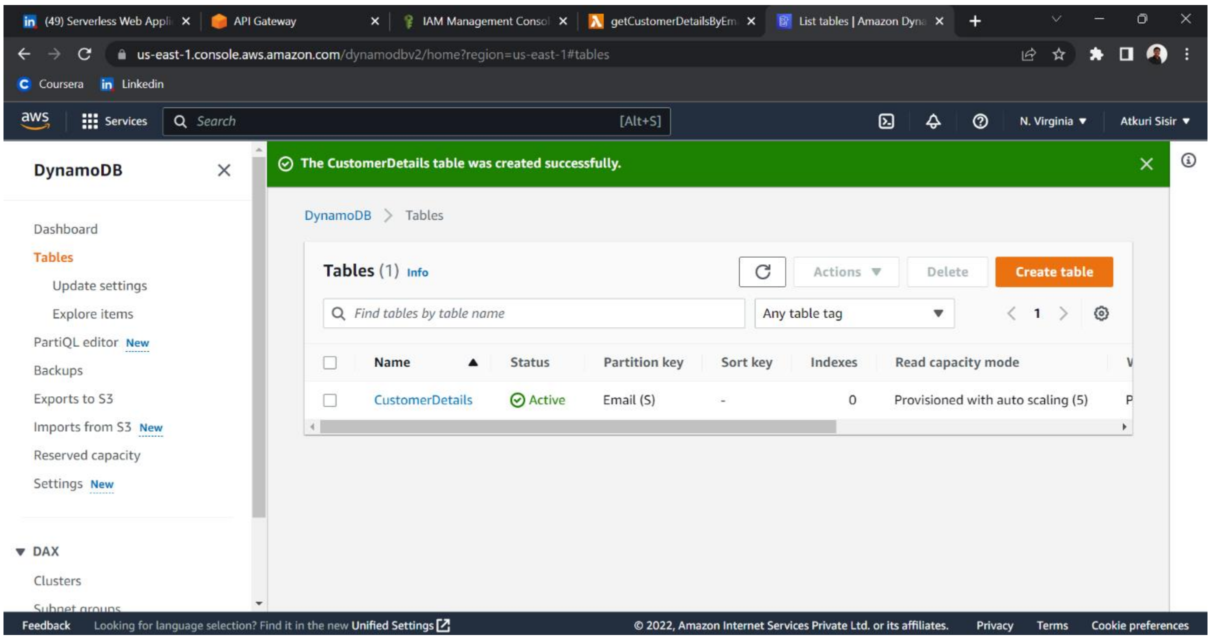
Create table

Table details

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name
This will be used to identify your table.
CustomerDetails
Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key
The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.
Email | String
1 to 255 characters and case sensitive.


The CustomerDetails table was created successfully.

DynamoDB > Tables

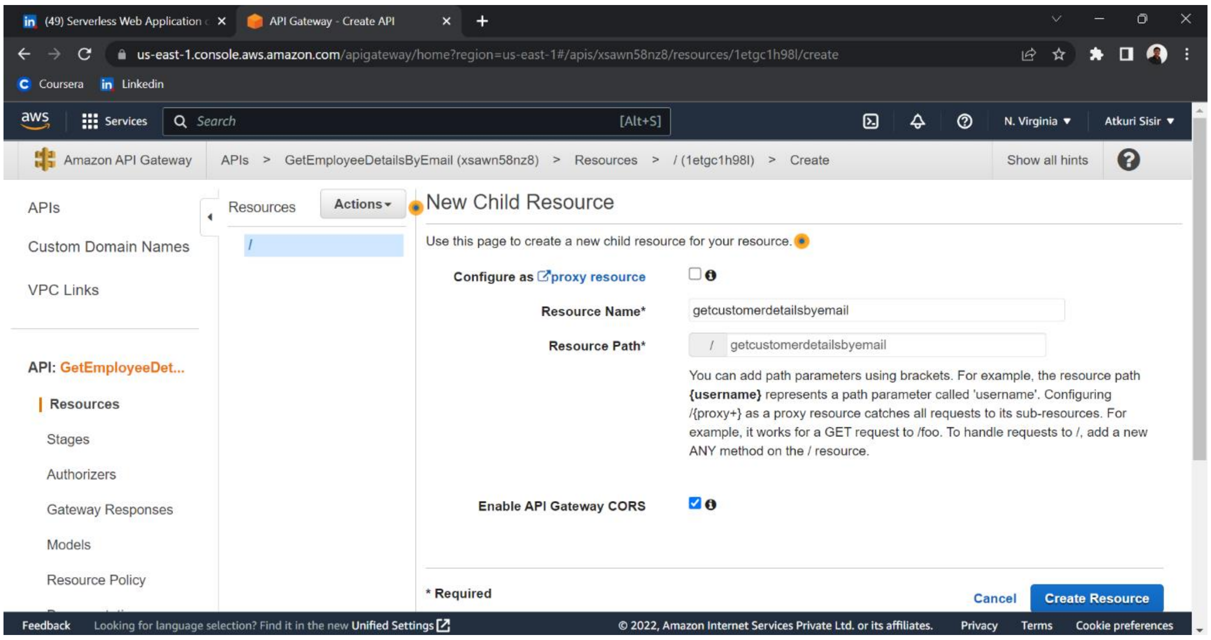
Tables (1)

Name	Status	Partition key	Sort key	Indexes	Read capacity mode
CustomerDetails	Active	Email (S)	-	0	Provisioned with auto scaling (5)

4. API Gateway

Create a REST API with the required name and create a resource with a specific name by clicking the Actions tab. Also, Enable CORS.

Create a GET method by changing the Integration type to Lambda function from mock(default). Select the name of the created lambda function.


APIs > GetEmployeeDetailsByEmail (xsawn58nz8) > Resources > / (1etgc1h98l) > Create

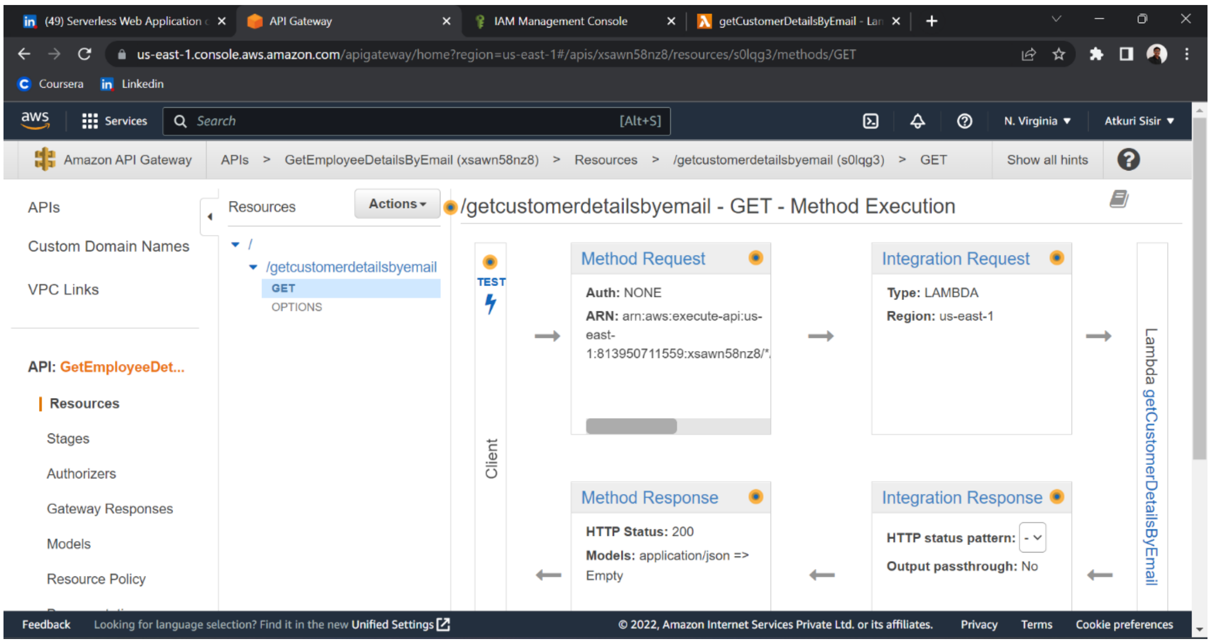
New Child Resource

Use this page to create a new child resource for your resource.

Configure as proxy resource	☐
Resource Name*	getcustomerdetailsbyemail
Resource Path*	/ getcustomerdetailsbyemail
Enable API Gateway CORS	☑

You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

* Required


/getcustomerdetailsbyemail - GET - Method Execution
Method Request
Integration Request
Method Response
Integration Response
Client
Lambda getCustomerDetailsByEmail
TEST

us-east-1.console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/xsawn58nz8/resources/s0lqg3/methods/GET

Auth: NONE
ARN: arn:aws:execute-api:us-east-1:813950711559:xsawn58nz8/*

Type: LAMBDA
Region: us-east-1

HTTP Status: 200
Models: application/json => Empty

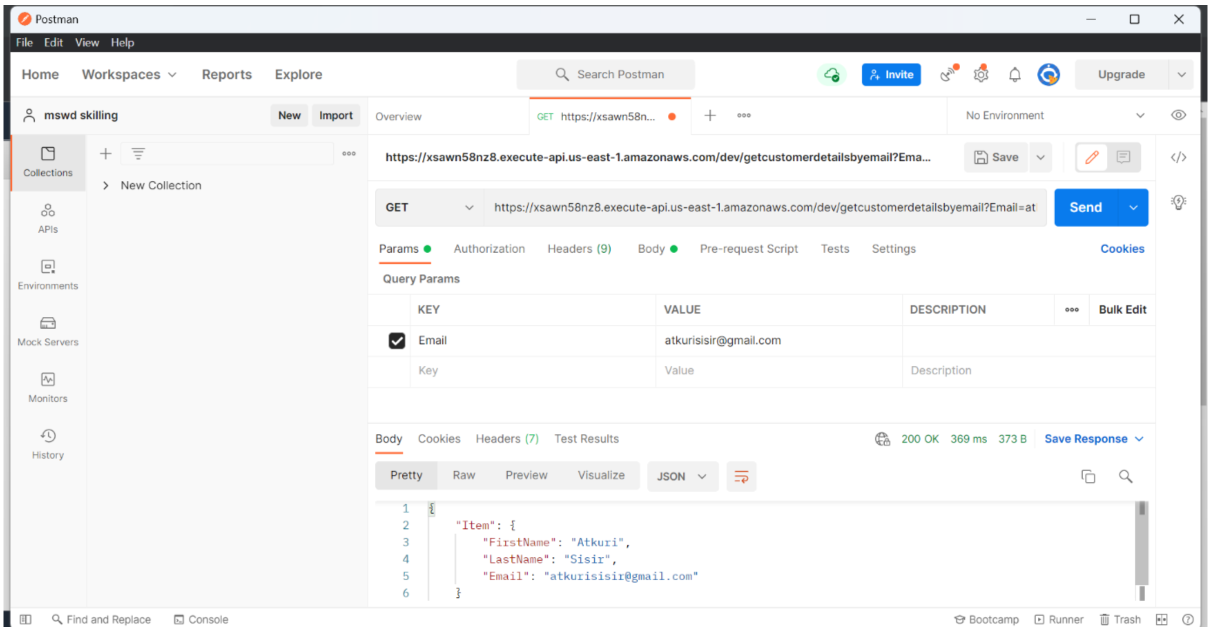
HTTP status pattern: -
Output passthrough: No

Add a URL query string parameter for the created API with content type Json. Make and confirm the integration request with the created lambda function. Last but not least, Deploy the API with a suitable name.

NOTE: - For API to work without hassles, Enable and update the CORS as needed.

Succeeding steps: -

- After integrating the lambda function with the REST API, open POSTMAN, an application to test various API requests.
- As part of the project, we are testing the GET request status.
- Provide the key and value along with the invoke URL of the deployed API.



GET https://xsawn58nz8.execute-api.us-east-1.amazonaws.com/dev/getcustomerdetailsbyemail?Email=at

KEY	VALUE	DESCRIPTION
Email	atkurisisir@gmail.com	

200 OK 369 ms 373 B

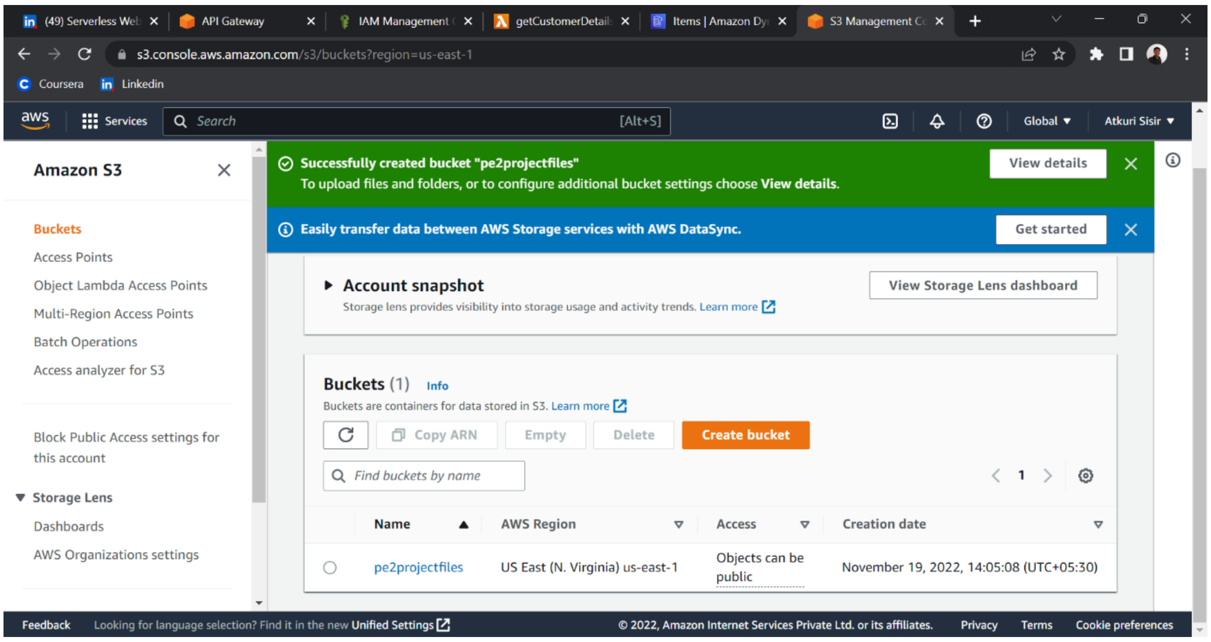
```
{
    "Item": {
        "FirstName": "Atkuri",
        "LastName": "Sisir",
        "Email": "atkurisisir@gmail.com"
    }
```

5. Amazon S3

NOTE: - we need specific static website resources to perform this step.

Create an HTML file with the required input to redirect the page to retrieve information stored in the DynamoDB table.

Host the static website by uploading the files into an Amazon S3 bucket. Enable static website hosting to get a website endpoint for the web application. Do not forget to provide public access to the files and write a bucket policy to display the index file while invoking the endpoint.



s3.console.aws.amazon.com/s3/buckets?region=us-east-1

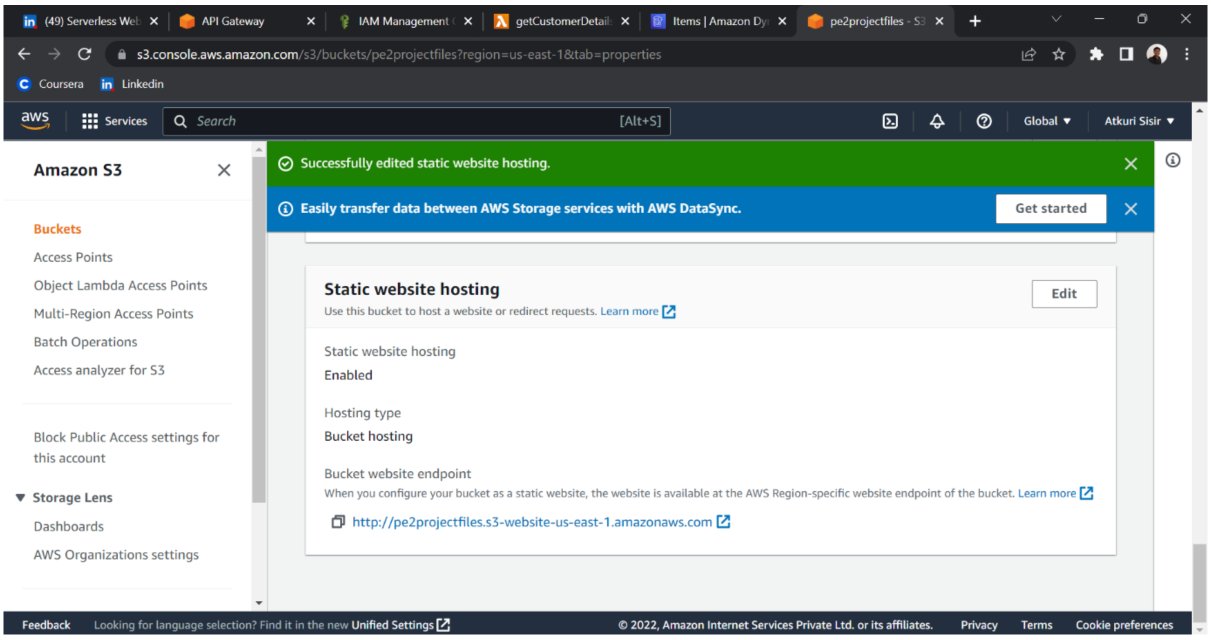
Successfully created bucket "pe2projectfiles"
To upload files and folders, or to configure additional bucket settings choose **View details**.

Easily transfer data between AWS Storage services with AWS DataSync.

Account snapshot
Storage lens provides visibility into storage usage and activity trends. Learn more

Buckets (1) Info
Buckets are containers for data stored in S3. Learn more

Name	AWS Region	Access	Creation date
pe2projectfiles	US East (N. Virginia) us-east-1	Objects can be public	November 19, 2022, 14:05:08 (UTC+05:30)



s3.console.aws.amazon.com/s3/buckets/pe2projectfiles?region=us-east-1&tab=properties

Successfully edited static website hosting.

Easily transfer data between AWS Storage services with AWS DataSync.

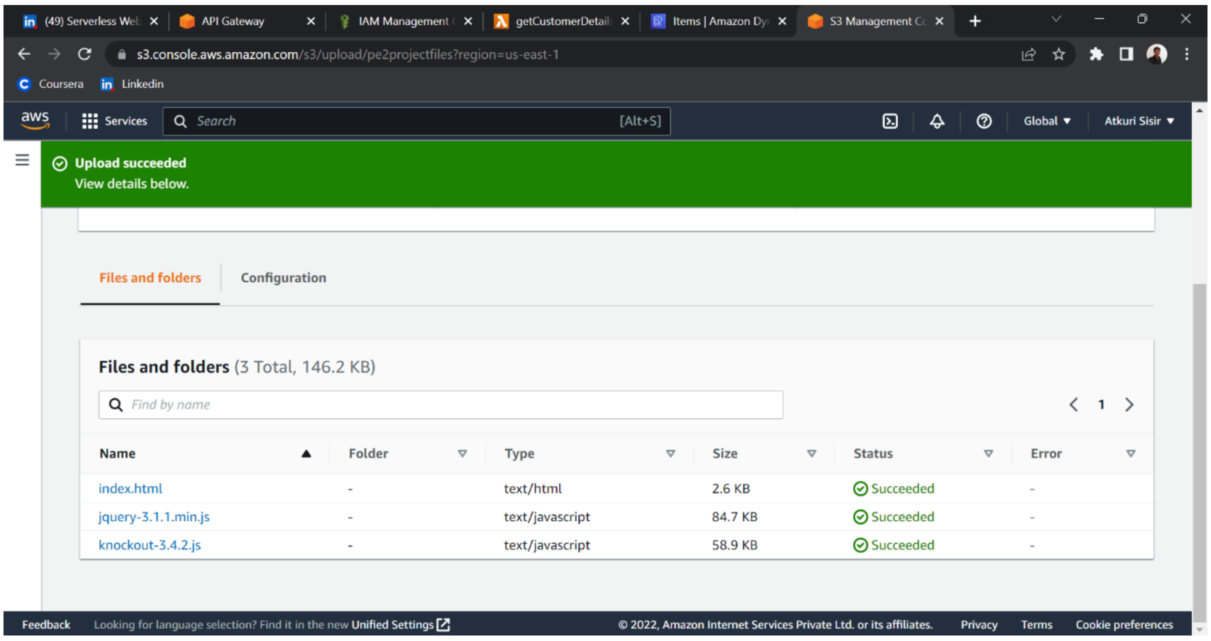
Static website hosting
Use this bucket to host a website or redirect requests. Learn more

Static website hosting
Enabled

Hosting type
Bucket hosting

Bucket website endpoint
When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. Learn more
http://pe2projectfiles.s3-website-us-east-1.amazonaws.com

Amazon S3
Buckets
Access Points
Object Lambda Access Points
Multi-Region Access Points
Batch Operations
Access analyzer for S3
Block Public Access settings for this account
Storage Lens
Dashboards
AWS Organizations settings
Get started
Edit

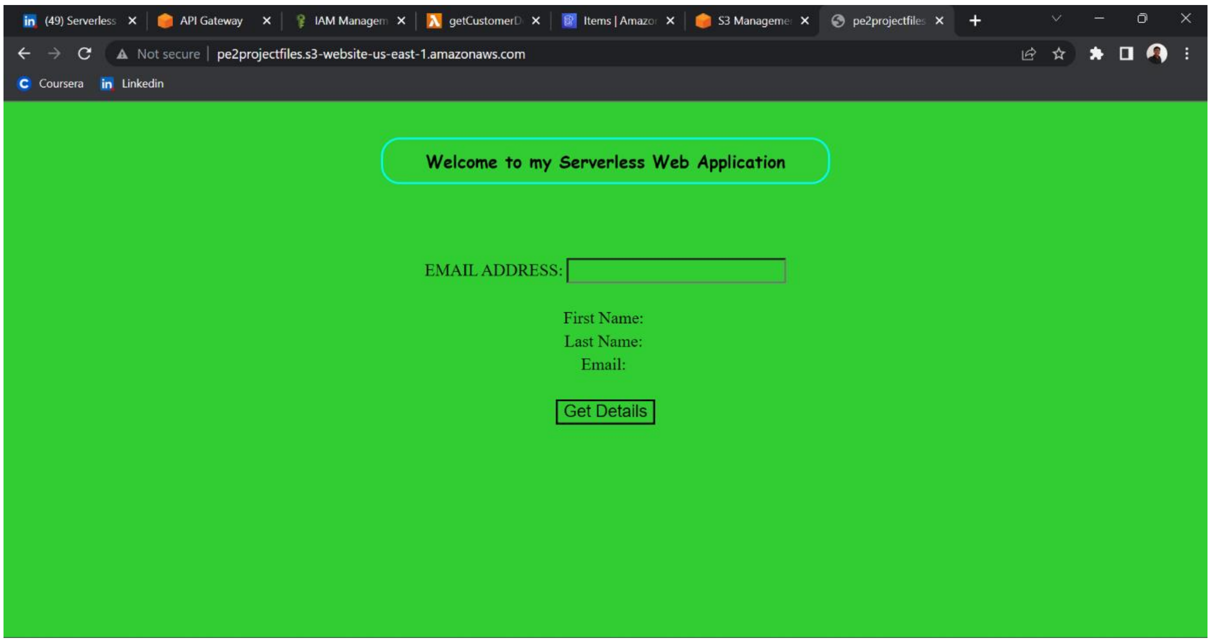


Upload succeeded
View details below.

Files and folders (3 Total, 146.2 KB)

Name	Folder	Type	Size	Status	Error
index.html	-	text/html	2.6 KB	Succeeded	-
jquery-3.1.1.min.js	-	text/javascript	84.7 KB	Succeeded	-
knockout-3.4.2.js	-	text/javascript	58.9 KB	Succeeded	-

Test the GET request status by invoking the serverless web application using the website endpoint. Type your email in the Email input section to display the information related to that partition key in your DynamoDB table.

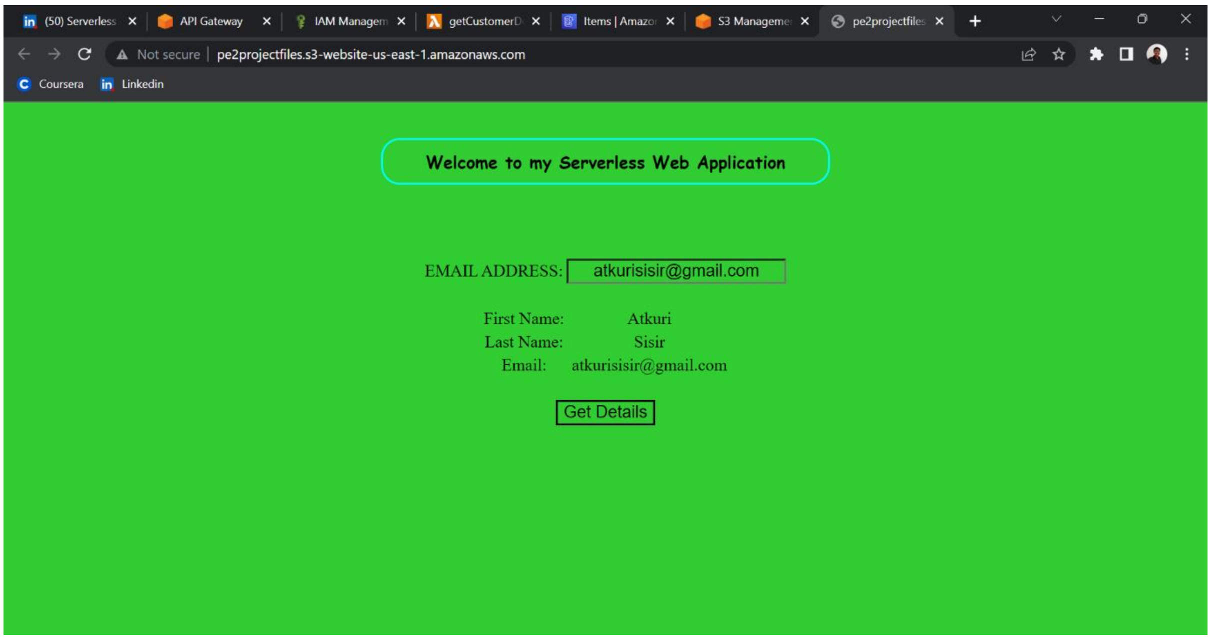


Welcome to my Serverless Web Application

EMAIL ADDRESS:

First Name:
Last Name:
Email:

Get Details



Welcome to my Serverless Web Application

EMAIL ADDRESS: atkurisisir@gmail.com

First Name: Atkuri
Last Name: Sisir
Email: atkurisisir@gmail.com

Get Details

RESULT:

The Cloud Internship experience from August 2022 to January 2023, focused on developing a serverless web application to retrieve details from Amazon DynamoDB using various AWS services, including Amazon S3, AWS API Gateway, AWS Lambda, and Amazon DynamoDB. In my role as an AWS Developer, I played a crucial part in the project's success.

Throughout this internship, we successfully designed and implemented a serverless web application that leveraged the power of AWS services to efficiently retrieve and display data from DynamoDB. We utilized Amazon S3 for storage, AWS API Gateway for managing the API requests, AWS Lambda for serverless computing, and Amazon DynamoDB as our NoSQL database.

One of the key highlights of the project was the ability to test the GET request status by invoking the serverless web application using the website endpoint. Users could input an email in the designated section, which would then fetch and display relevant information from the DynamoDB table, demonstrating the practicality and real-world application of the solution.

In conclusion, this internship provided invaluable hands-on experience in cloud computing and AWS services. I gained a deep understanding of serverless architecture, API management, and database integration, along with the skills necessary to develop and deploy a functional web application in a cloud environment. This experience not only broadened my technical knowledge but also reinforced the significance of cloud technologies in modern software development. I am grateful for the opportunity to work on this project, and I am excited to apply the skills and knowledge I have acquired to future endeavors in the world of cloud computing and web application development.