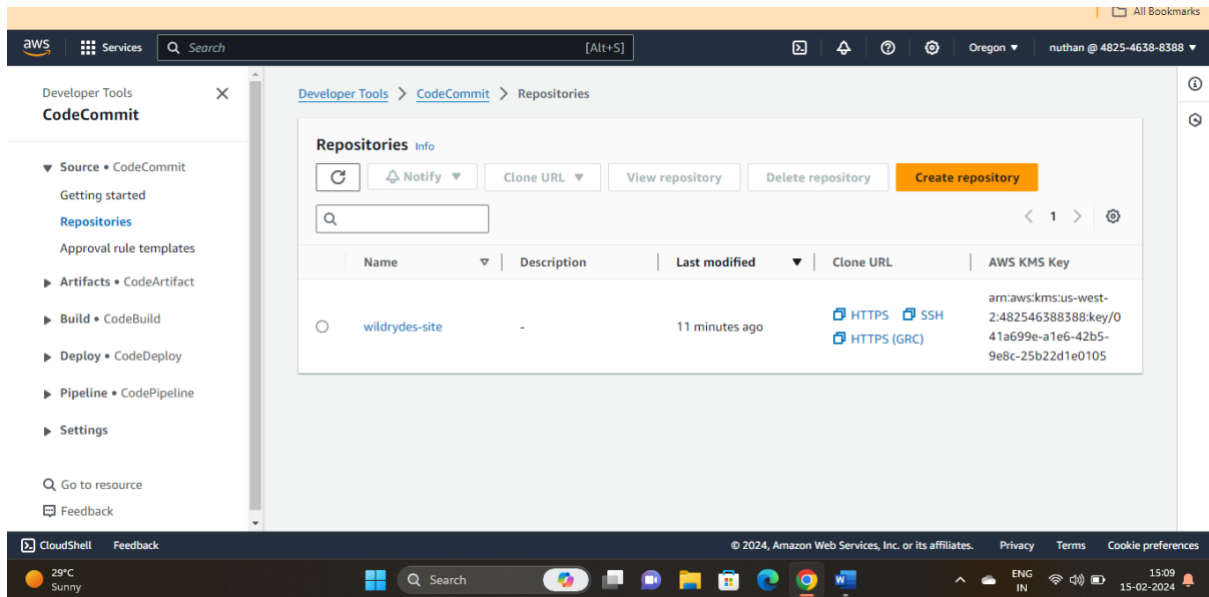


Building a Serverless Web Application

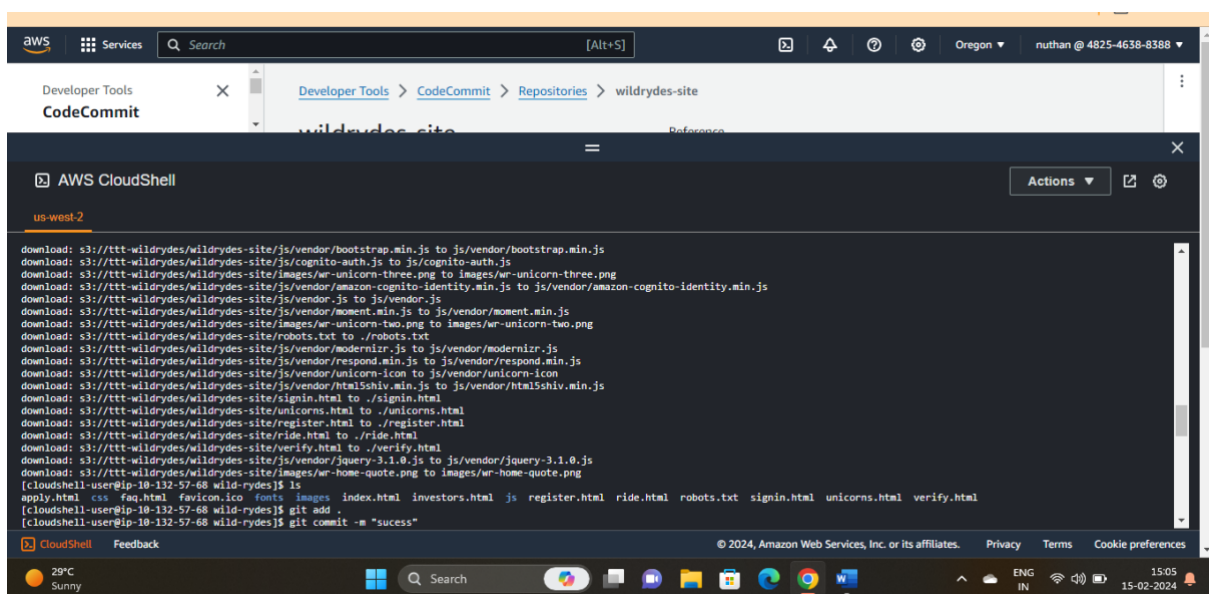
with AWS Lambda, Amazon API Gateway, AWS Amplify, Amazon DynamoDB, and Amazon Cognito

STEP-1 :

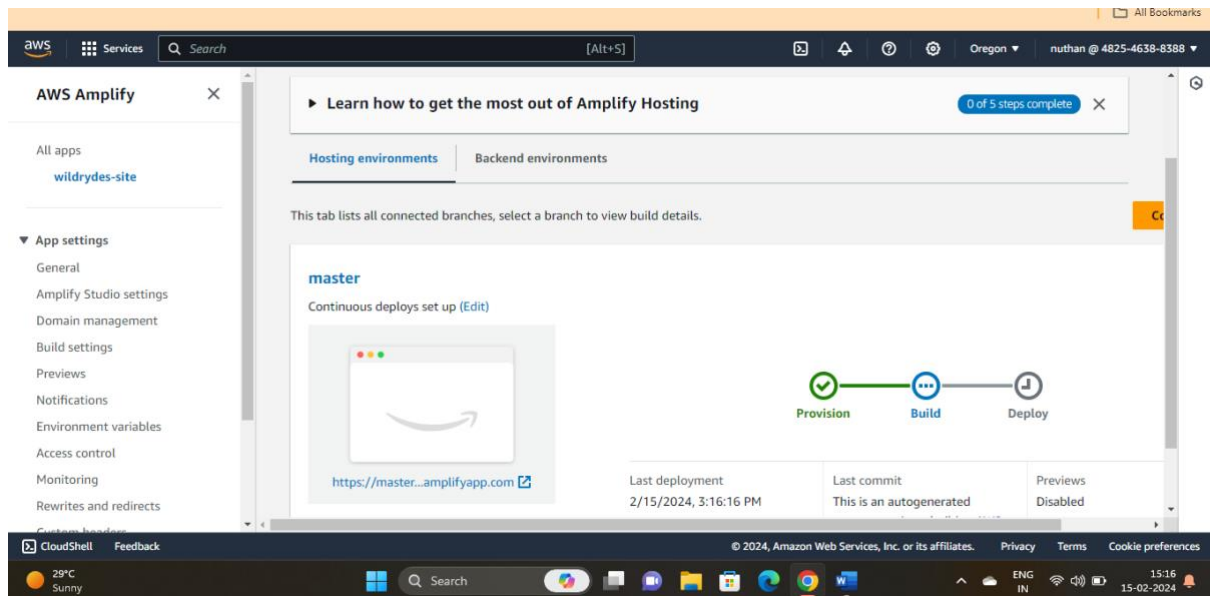
Create a repository in code commit , repo name is “ **wildrydes-site** ”



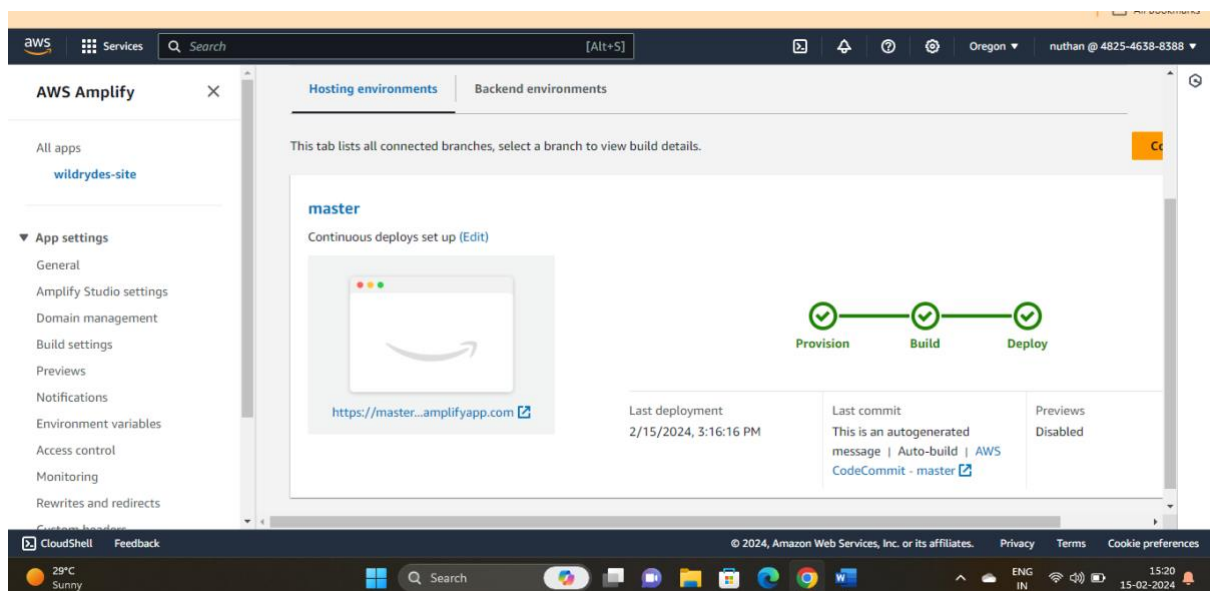
Add website files to repository from a public aws s3 bucket with command “ **aws s3 cp s3://ttt-wildrydes/wildrydes-site ./ --recursive** ”



Open **aws amplify** and create a web hosting application and deploy source code from created repository **"wildrydes-site"** of codecommit



After few minutes the source code is completely provisioned, build and deployed successfully .



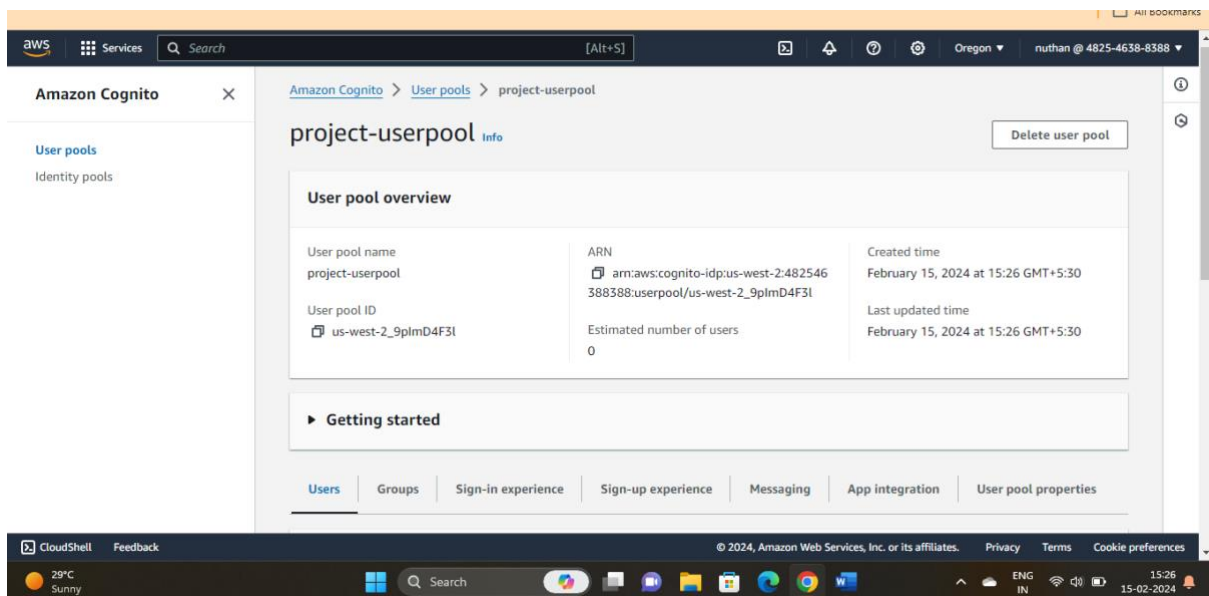
After successful deployment , the website is ready . we can see it by opening the link beside of status bar.



Upto now first step is completed.

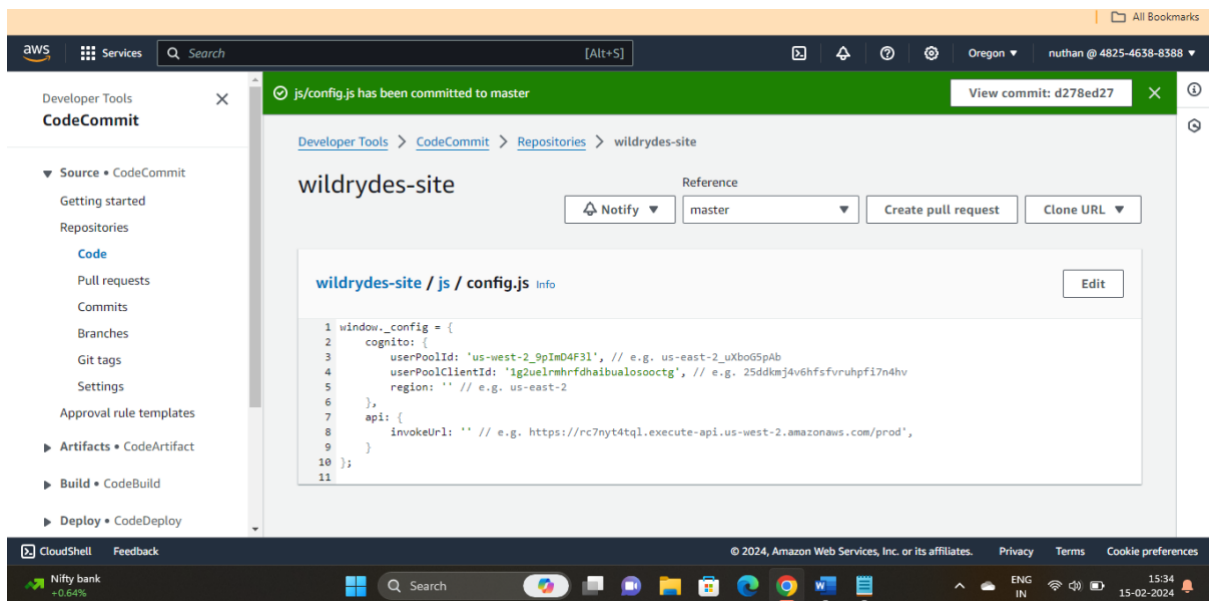
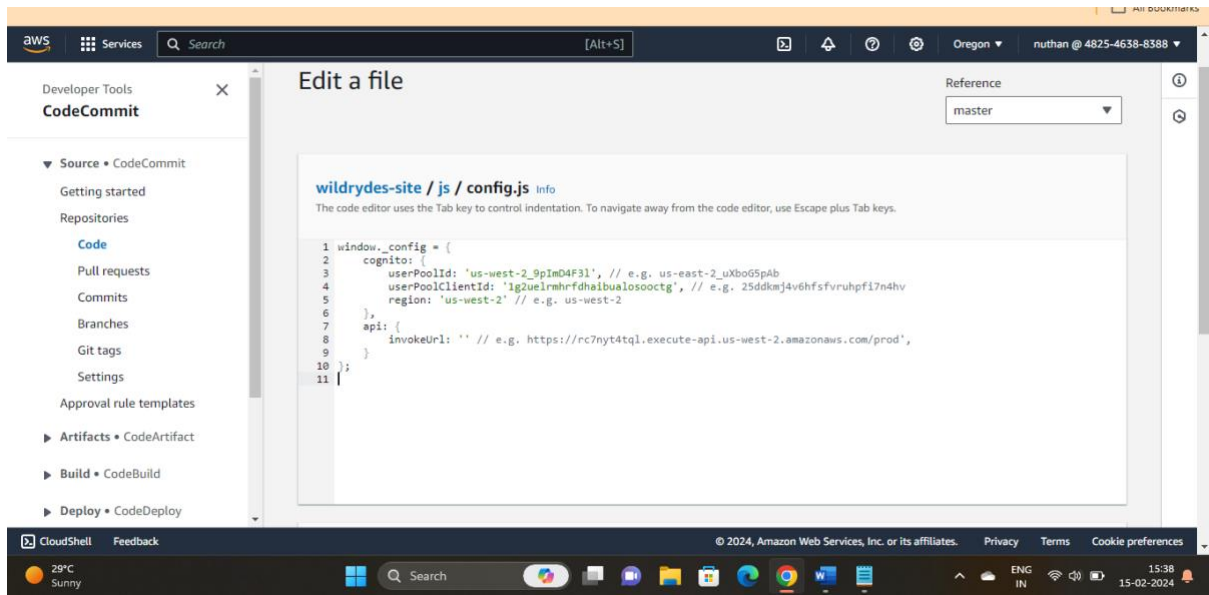
STEP-2 :

Goto **aws cognito** and create a user pool with name “ project-userpool” with **no-MFA** and **No-attributes**



Copy userpool-ID and client ID of userpool

Goto **codecommit** -> **wildrydes-site** repo -> **js** -> **config.js** file and edit file by adding userpool-id , client ID and region.



After committing code , automatically , the code in repo is rebuit and deployed into aws amplify

aws Services Search [Alt+S] Oregon nuthan @ 4825-4638-8388

AWS Amplify

All apps
wildrydes-site

App settings
General
Amplify Studio settings
Domain management
Build settings
Previews
Notifications
Environment variables
Access control
Monitoring
Rewrites and redirects

environments Backend environments

Connect branch

Previous deploys set up (Edit)

Provision Build Deploy

Last deployment: 2/15/2024, 3:38:40 PM

Last commit: Please visit AWS CodeCommit Co... | 81899be | AWS CodeCommit - master

Previews: Disabled

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

29°C Sunny 15:40 15-02-2024

aws Services Search [Alt+S] Oregon nuthan @ 4825-4638-8388

AWS Amplify

All apps
wildrydes-site

App settings
General
Amplify Studio settings
Domain management
Build settings
Previews
Notifications
Environment variables
Access control
Monitoring
Rewrites and redirects

environments Backend environments

Connect branch

Previous deploys set up (Edit)

Provision Build Deploy

Last deployment: 2/15/2024, 3:38:40 PM

Last commit: Please visit AWS CodeCommit Co... | 81899be | AWS CodeCommit - master

Previews: Disabled

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

29°C Sunny 15:40 15-02-2024

master.denn99iizt8jh.amplifyapp.com/register.html

WildRYDES

REGISTER

nuthannavuluri321@gmail.com

.....

.....

LET'S RYDE

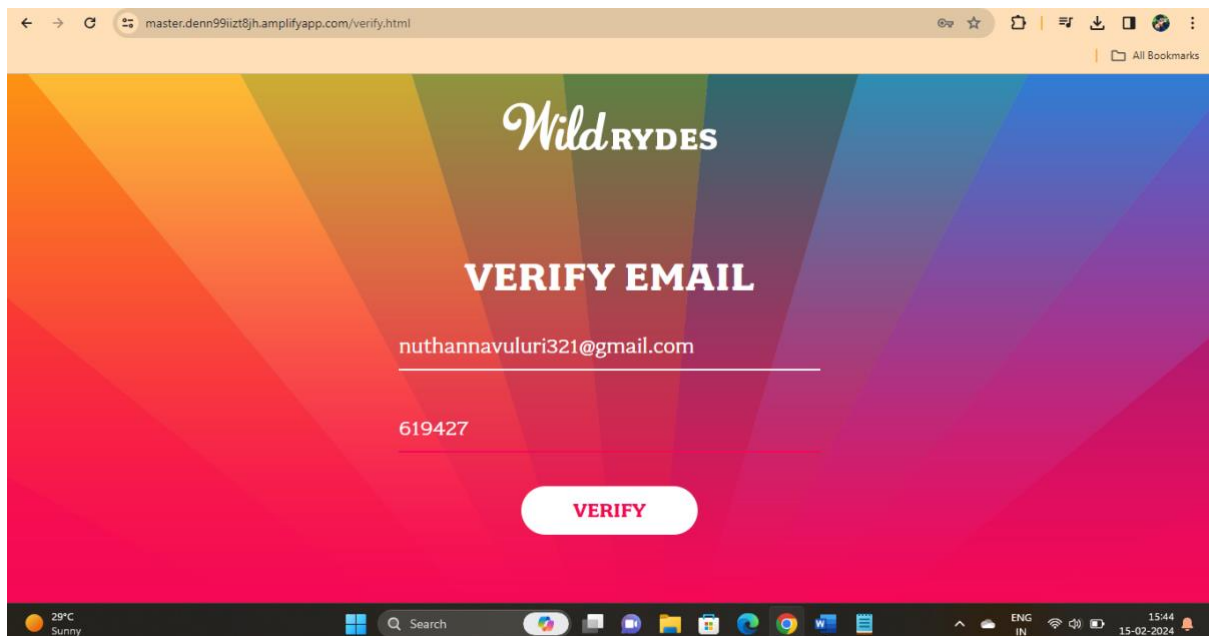
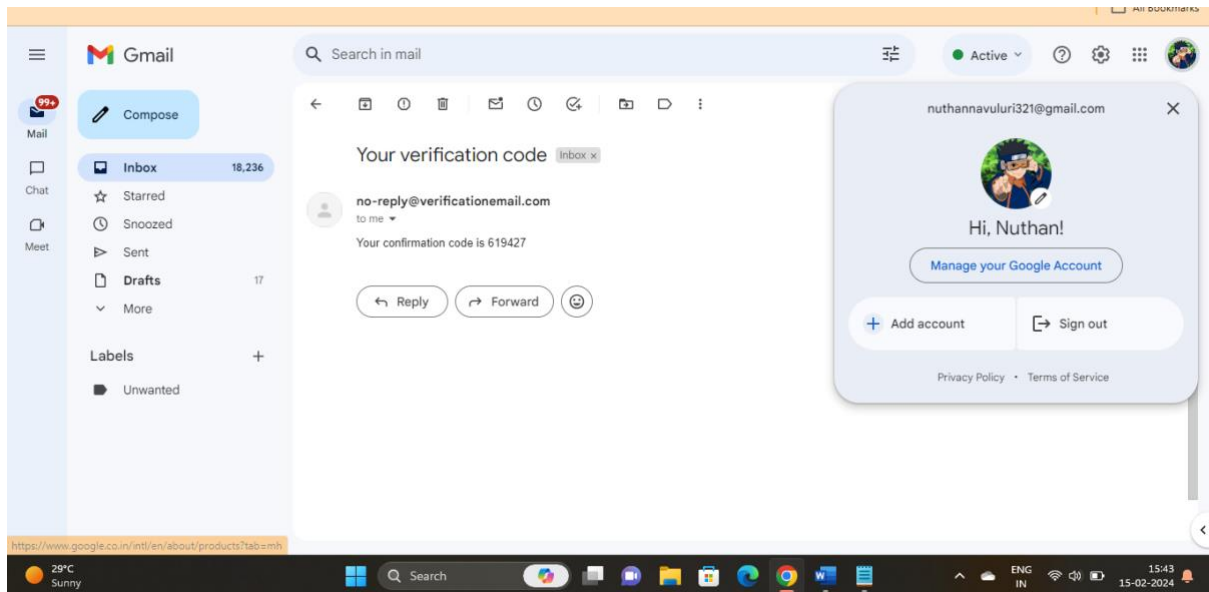
Nuthan Navuluri
nuthannavuluri321@gmail.com

Sync is on
Manage your Google Account

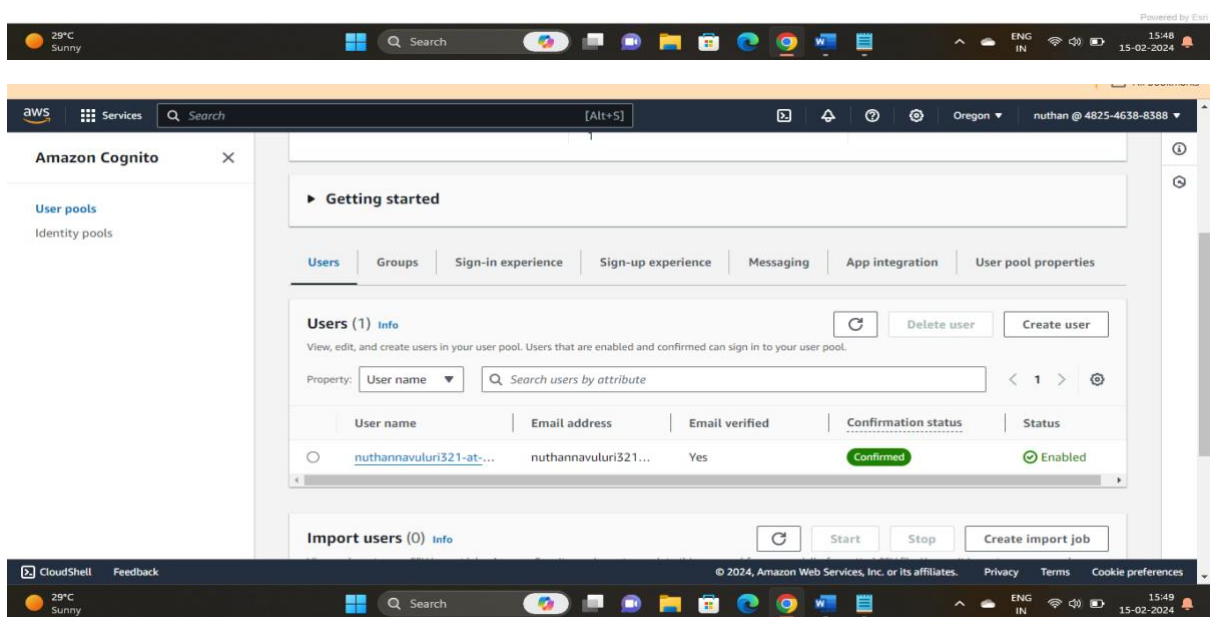
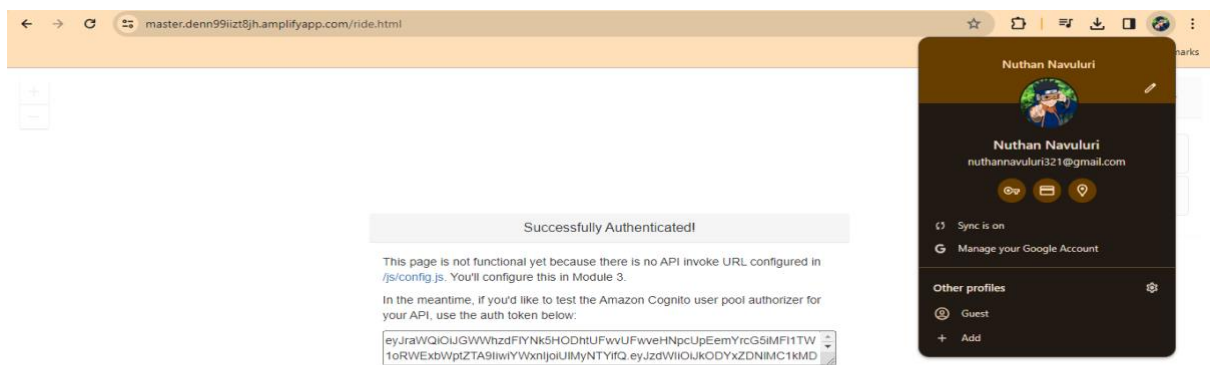
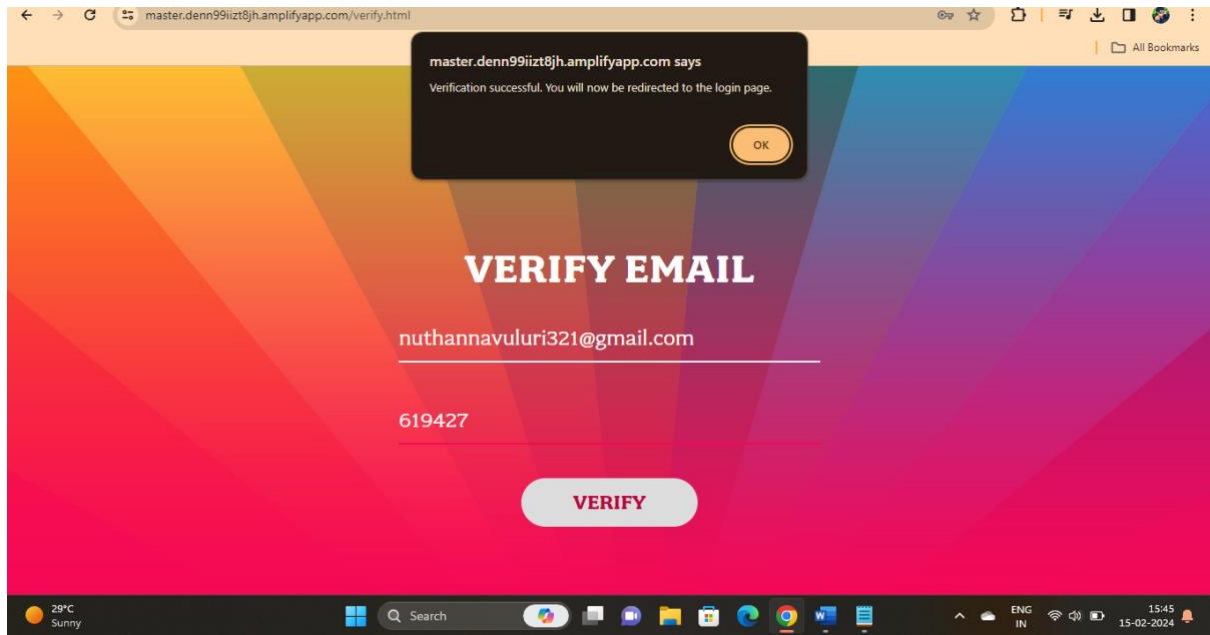
Other profiles
Guest
Add

29°C Sunny 15:41 15-02-2024

On clicking **lets ride** -> otp will sent to email for verifying email address.



verification of mail is successful , we can login into website

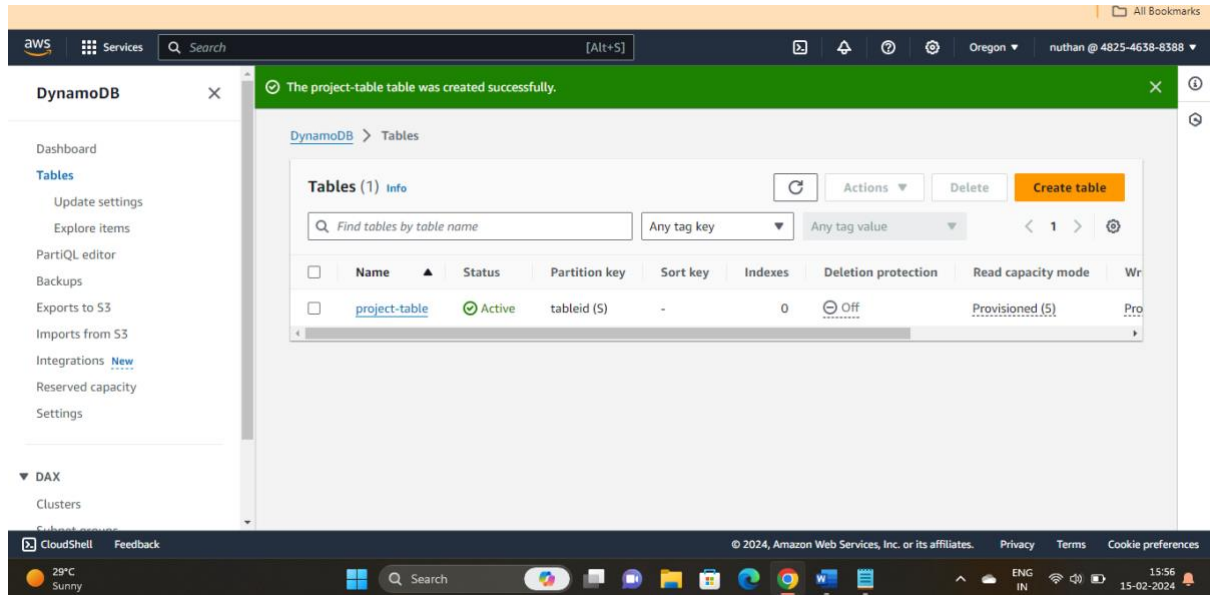


Hence , userpool is successfully added to website . now we can register and login into the website.

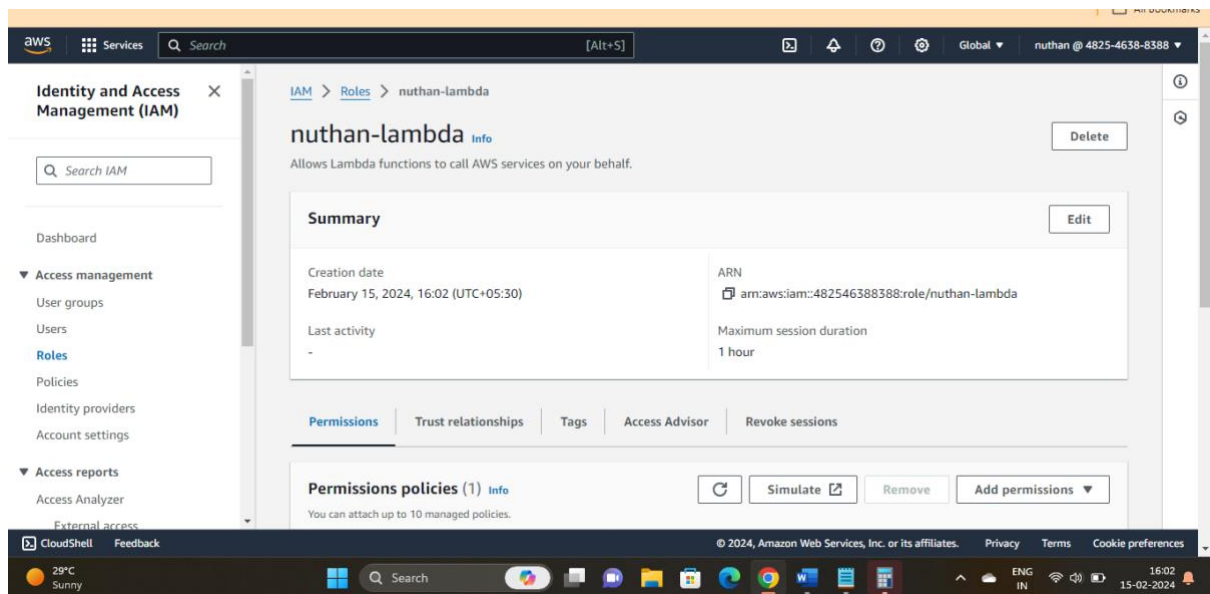
STEP-3:

Creating serverless backend , we need a database and lambda function.

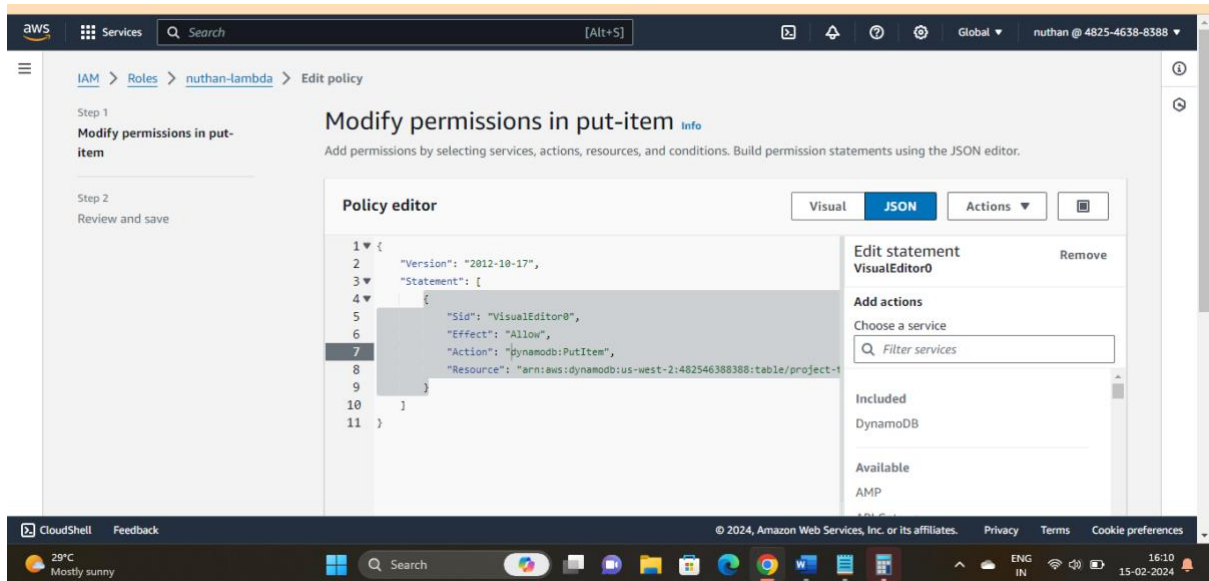
➔ open Dynamo DB . In that , create a table and copy ARN of it



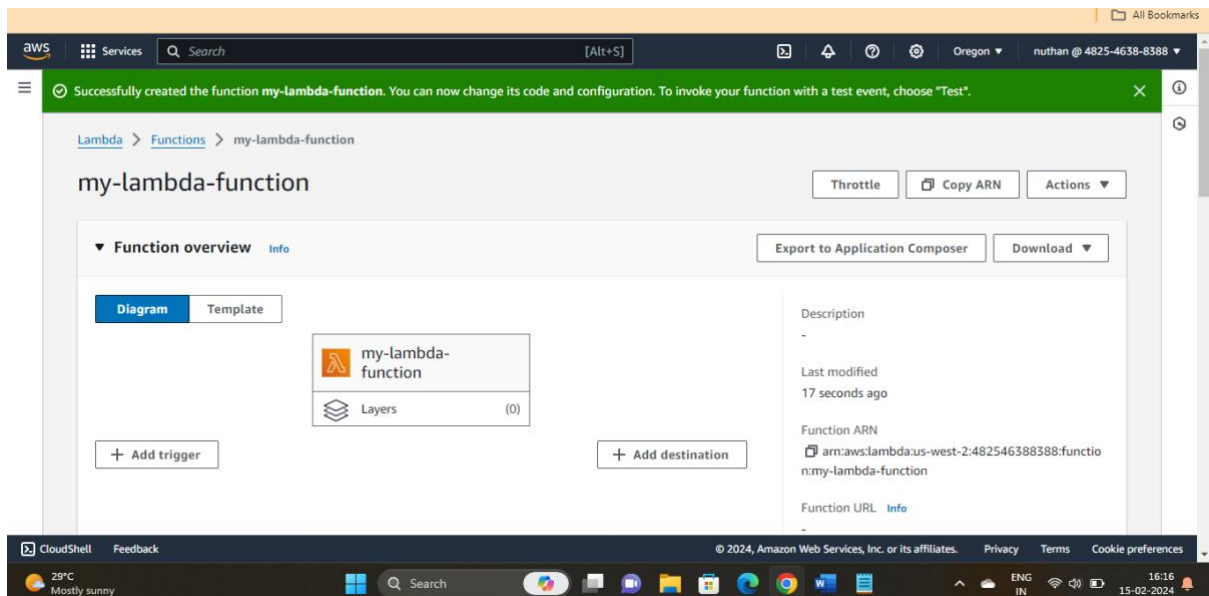
Create a **lambda** service role



Modify service role so that it can write table of Dynamo Db by **creating inline policy**. I named it as **“put-item”** policy.



Now , create a Lambda Function to create serverless application with Node.js.16.x environment and created service role **“ Nuthan-lambda”** .



Now add below code into lambda function which is given by aws .

```
const randomBytes = require('crypto').randomBytes;
const AWS = require('aws-sdk');
const ddb = new AWS.DynamoDB.DocumentClient();

const fleet = [
  {
    Name: 'Angel',
    Color: 'White',
    Gender: 'Female',
  },
  {
    Name: 'Gil',
    Color: 'White',
    Gender: 'Male',
  },
  {
    Name: 'Rocinante',
    Color: 'Yellow',
    Gender: 'Female',
  },
];

exports.handler = (event, context, callback) => {
  if (!event.requestContext.authorizer) {
    errorResponse('Authorization not configured', context.awsRequestId, callback);
    return;
  }

  const ridId = toUrlString(randomBytes(16));
  console.log('Received event (', ridId, '): ', event);

  // Because we're using a Cognito User Pools authorizer, all of the claims
  // included in the authentication token are provided in the request context.
  // This includes the username as well as other attributes.
  const username = event.requestContext.authorizer.claims['cognito:username'];

  // The body field of the event in a proxy integration is a raw string.
  // In order to extract meaningful values, we need to first parse this string
  // into an object. A more robust implementation might inspect the Content-Type
  // header first and use a different parsing strategy based on that value.
  const requestBody = JSON.parse(event.body);

  const pickupLocation = requestBody.PickupLocation;

  const unicorn = findUnicorn(pickupLocation);

  recordRide(ridId, username, unicorn).then(() => {
    // You can use the callback function to provide a return value from your Node.js
    // Lambda functions. The first parameter is used for failed invocations. The
    // second parameter specifies the result data of the invocation.

    // Because this Lambda function is called by an API Gateway proxy integration
    // the result object must use the following structure.
    callback(null, {
      statusCode: 201,
      body: JSON.stringify({
        RidId: ridId,
        Unicorn: unicorn,
        Eta: '30 seconds',
        Rider: username,
      }),
      headers: {
        'Access-Control-Allow-Origin': '*',
      },
    });
  }).catch((err) => {
    console.error(err);

    // If there is an error during processing, catch it and return
    // from the Lambda function successfully. Specify a 500 HTTP status
    // code and provide an error message in the body. This will provide a
    // more meaningful error response to the end client.
    errorResponse(err.message, context.awsRequestId, callback)
  });
};

// This is where you would implement logic to find the optimal unicorn for
// this ride (possibly invoking another Lambda function as a microservice.)
```

```

// For simplicity, we'll just pick a unicorn at random.
function findUnicorn(pickupLocation) {
  console.log('Finding unicorn for ', pickupLocation.Latitude, ', ', pickupLocation.Longitude);
  return fleet[Math.floor(Math.random() * fleet.length)];
}

function recordRide(rideId, username, unicorn) {
  return db.put({
    TableName: 'Rides',
    Item: {
      RideId: rideId,
      User: username,
      Unicorn: unicorn,
      RequestTime: new Date().toISOString(),
    },
  }).promise();
}

function toUrlString(buffer) {
  return buffer.toString('base64')
    .replace(/\+/g, '-')
    .replace(/\//g, '_')
    .replace(/=/g, '');
}

function ErrorResponse(errorMessage, awsRequestId, callback) {
  callback(null, {
    statusCode: 500,
    body: JSON.stringify({
      Error: errorMessage,
      Reference: awsRequestId,
    }),
    headers: {
      'Access-Control-Allow-Origin': '*',
    },
  });
}

```

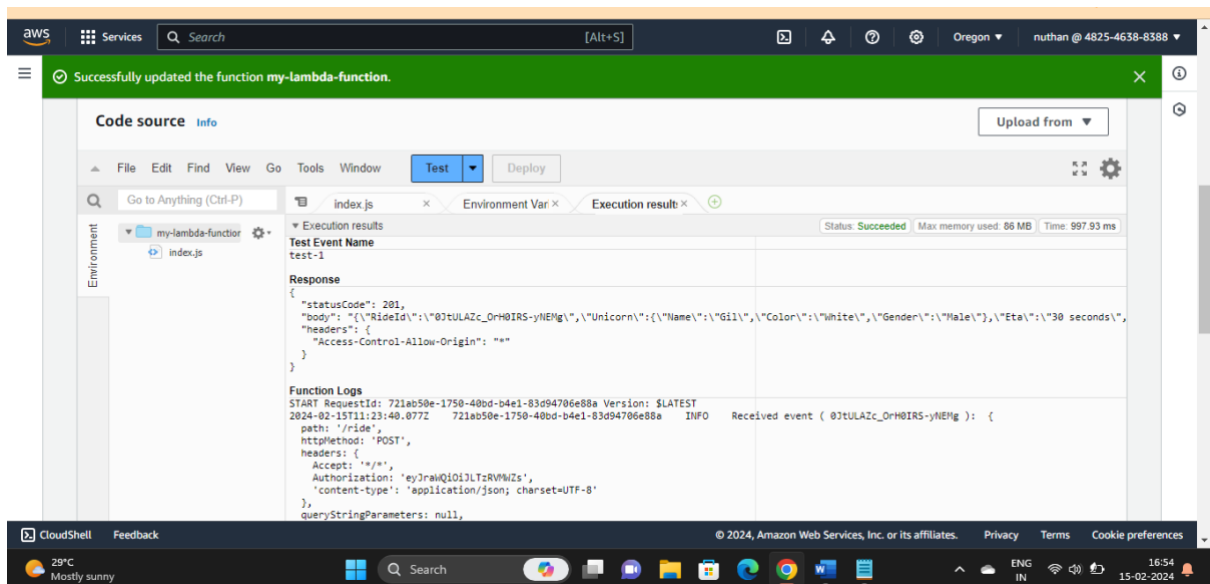
TEST EVENT FOR LAMBDA FUNCTION

```

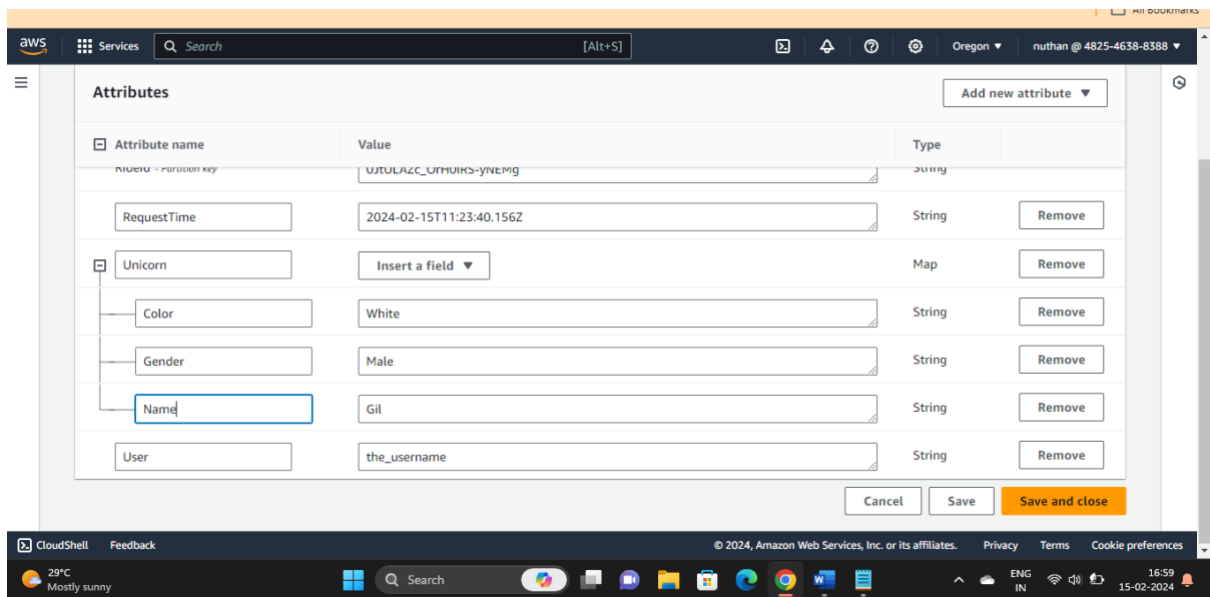
{
  "path": "/ride",
  "httpMethod": "POST",
  "headers": {
    "Accept": "*/*",
    "Authorization": "eyJraWQiOiJLTzRVMWZs",
    "content-type": "application/json; charset=UTF-8"
  },
  "queryStringParameters": null,
  "pathParameters": null,
  "requestContext": {
    "authorizer": {
      "claims": {
        "cognito:username": "the_username"
      }
    }
  },
  "body": "{\"PickupLocation\":{\"Latitude\":47.6174755835663,\"Longitude\":-122.28837066650185}}"
}

```

After modifying function code (enter dynao DB table instead of default table name), on testing we get.



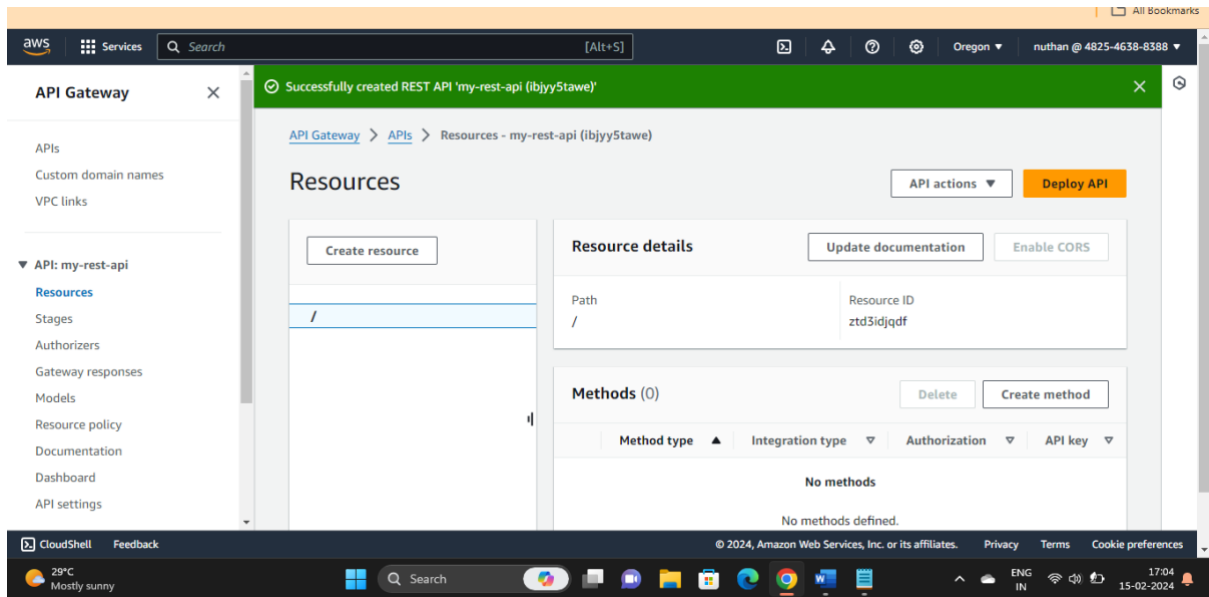
Successfully values from lambda function is write into table.



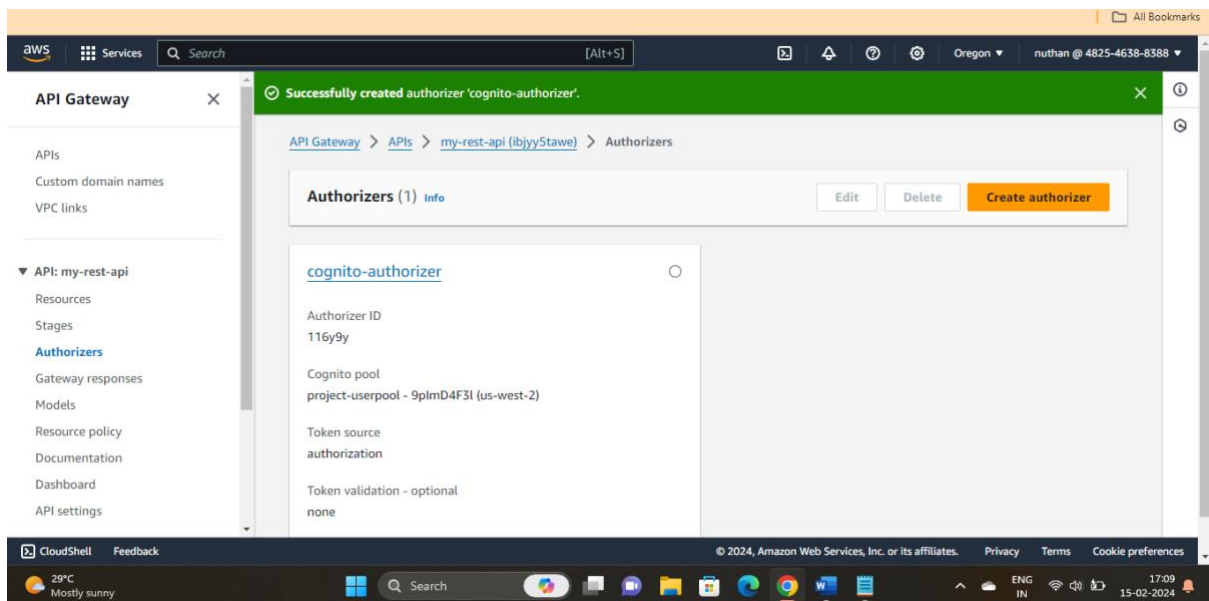
STEP-4 :

Deploy a RESTFUL API

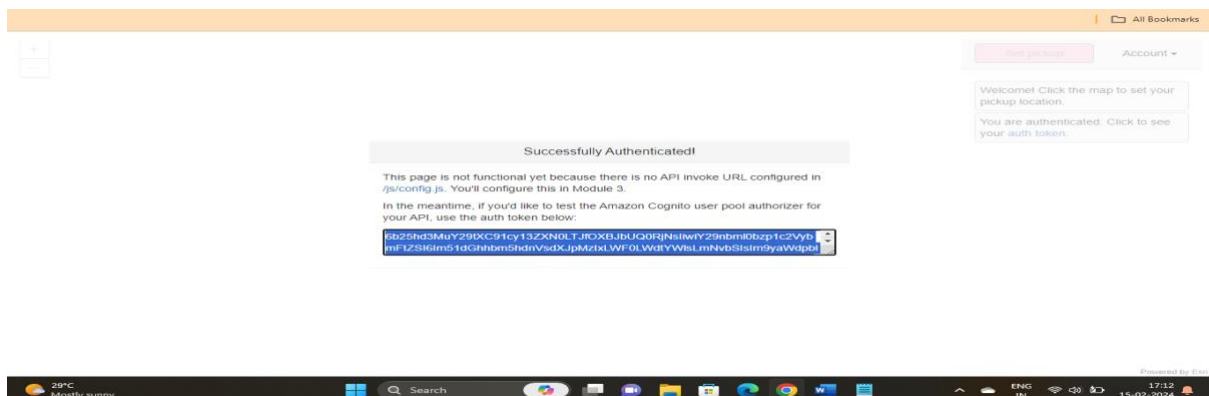
➔ open AWS API Gateway -> create a new REST API



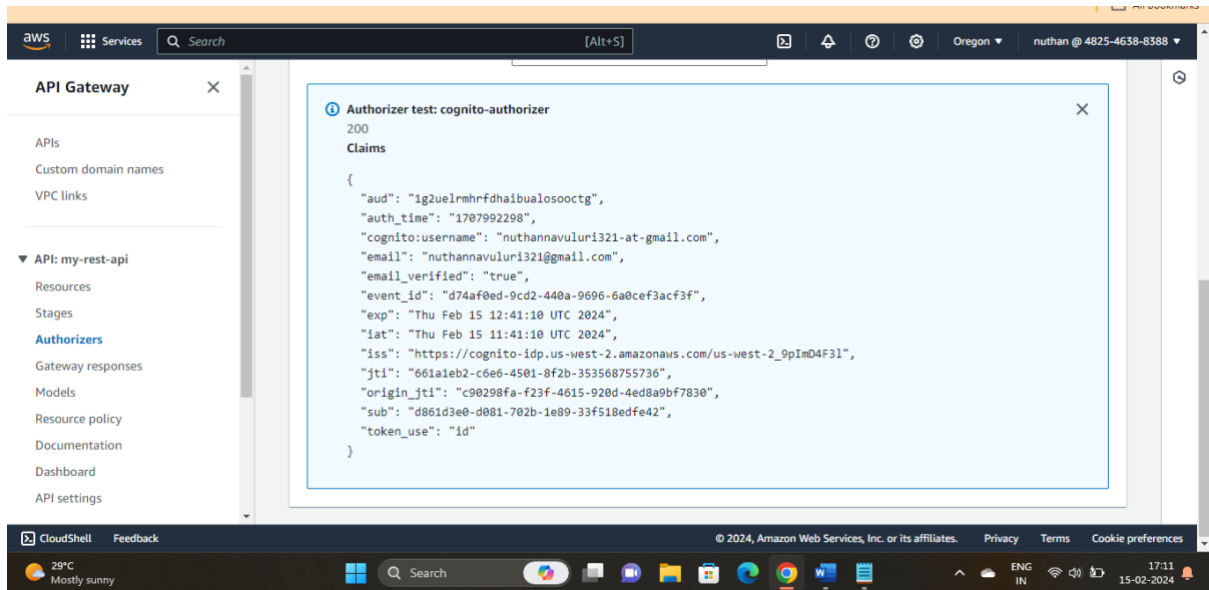
We are using cognito pool , for authenticating we need authenticator . so create a authorizer in REST API.



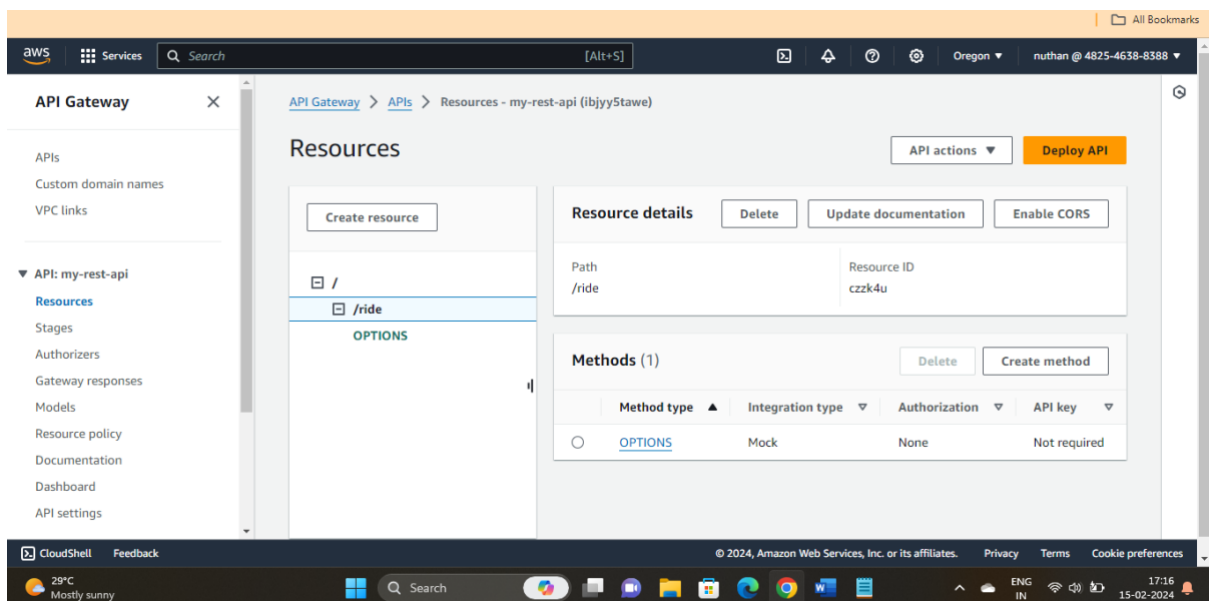
Copy authorisation token from website.



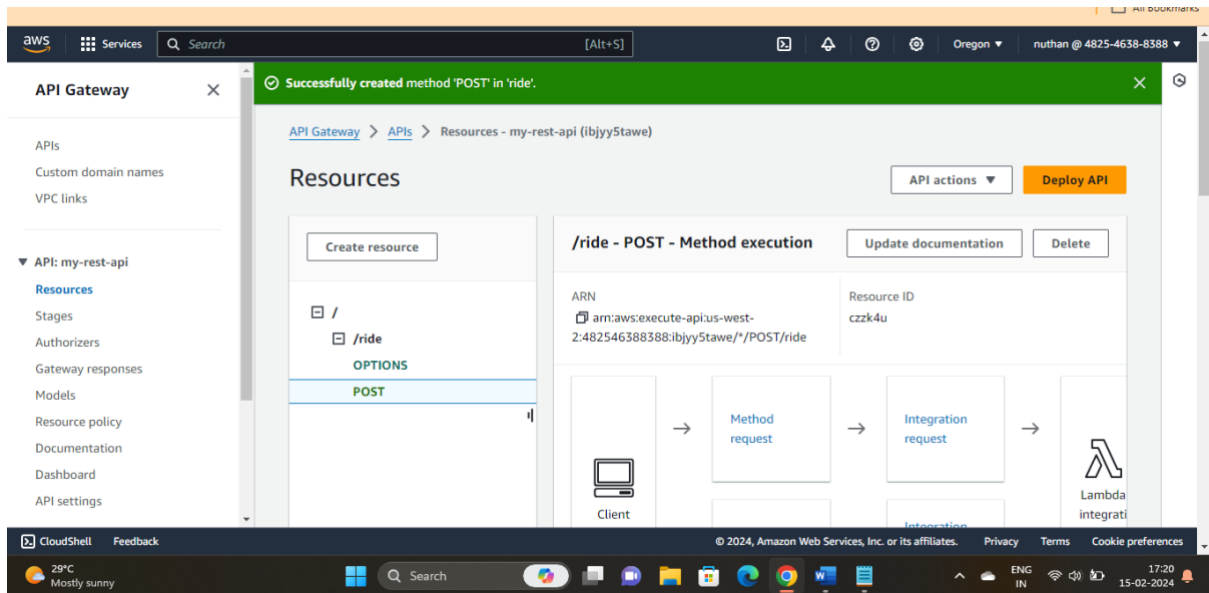
Check authorisation by checkin in authorizer . which shows success code -200.



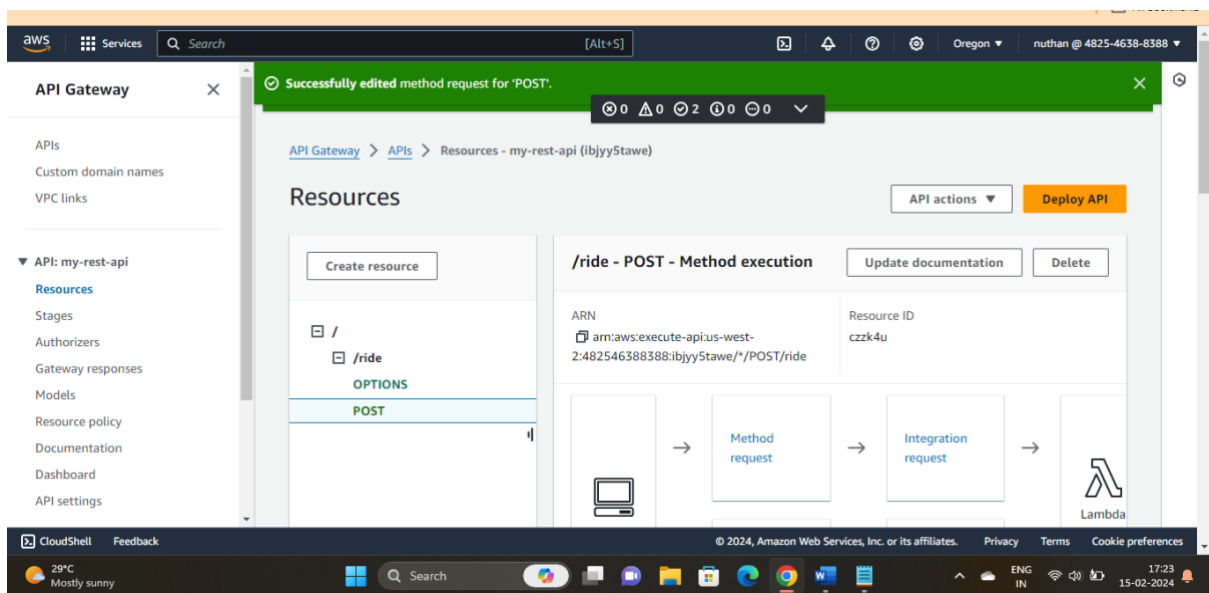
Now create a resource in REST API by enabling CORS(Cross origin resource sharing)



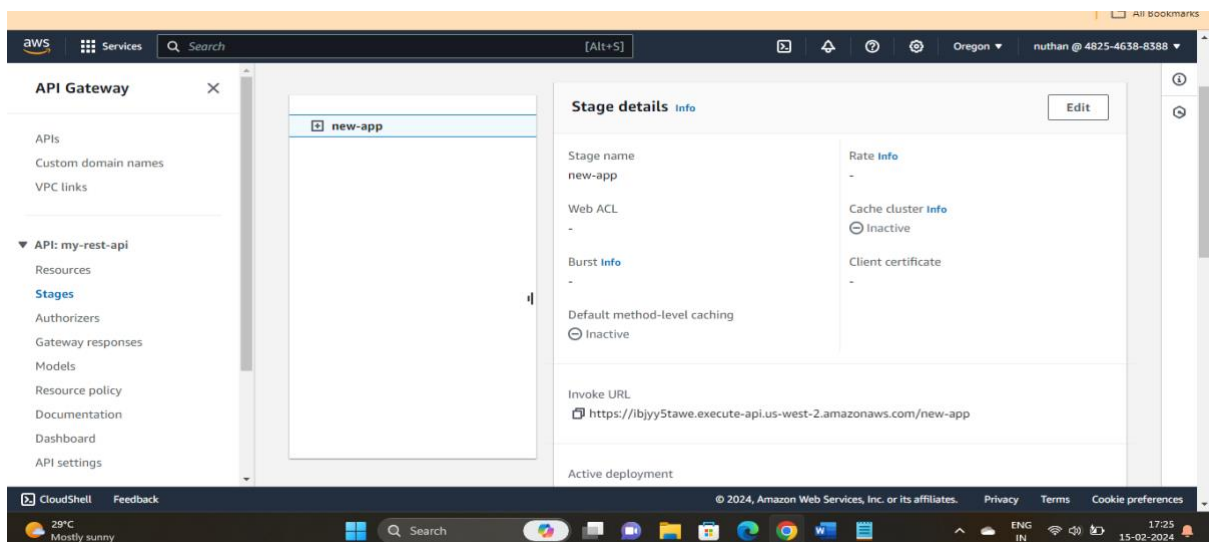
Now create POST method in resource by integrating with lambda function we created earlier.



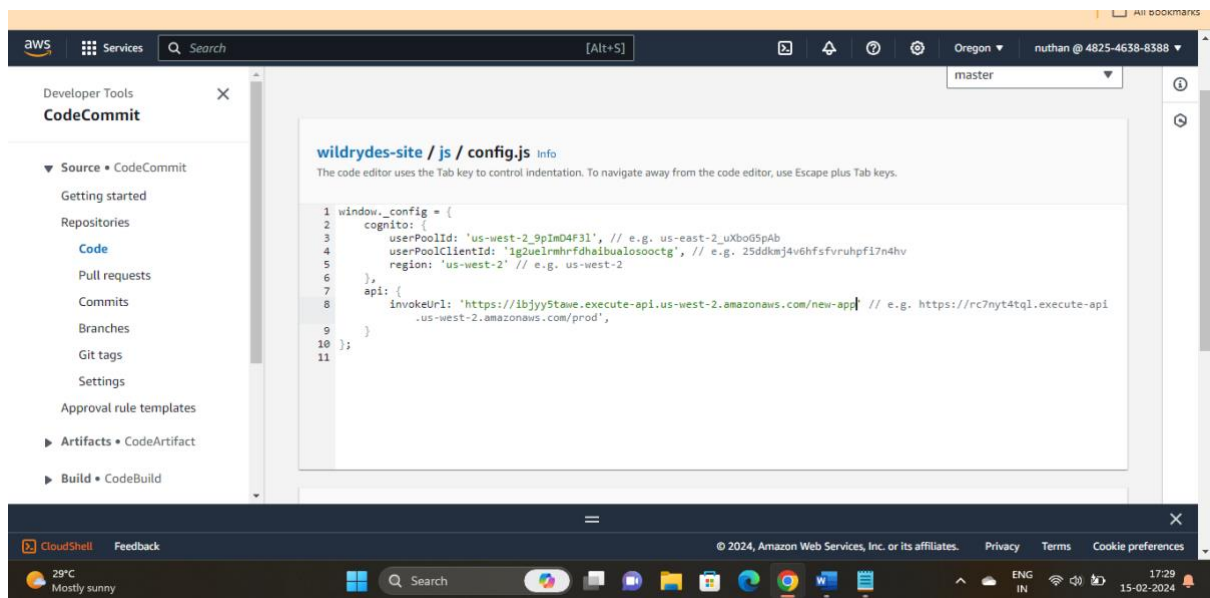
Now -> click on Method request -> edit -> add authorizer which we created earlier.



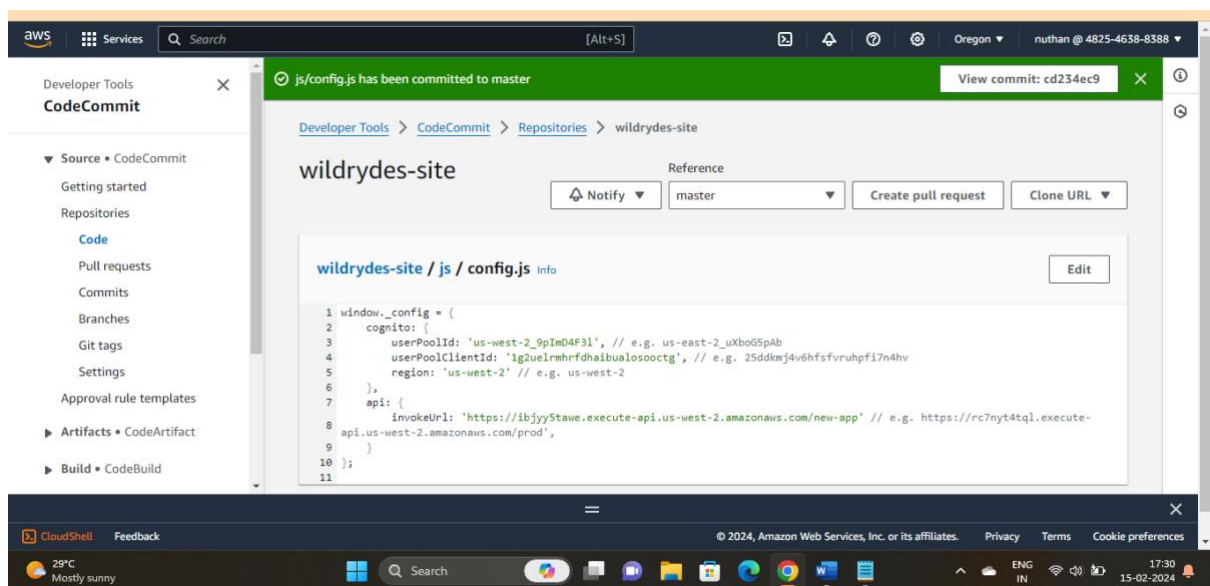
Now deploy API. After deploying , copy invoke URL



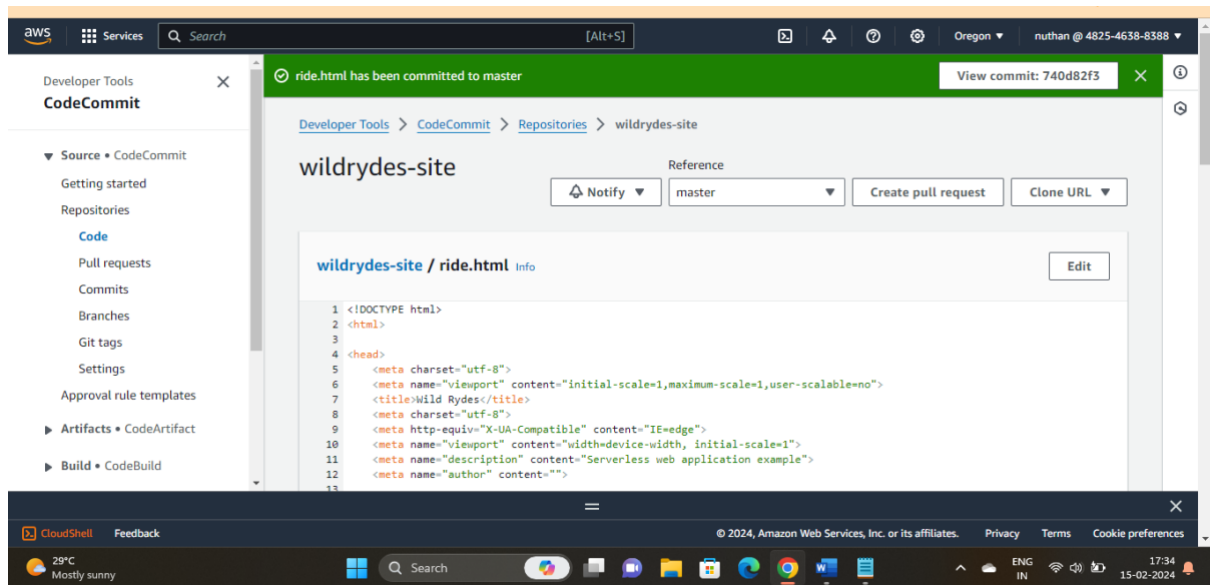
Goto -> code commit -> repo "wildrydes-site" -> js -> config.js file and edit file. Add invoke url to the file.



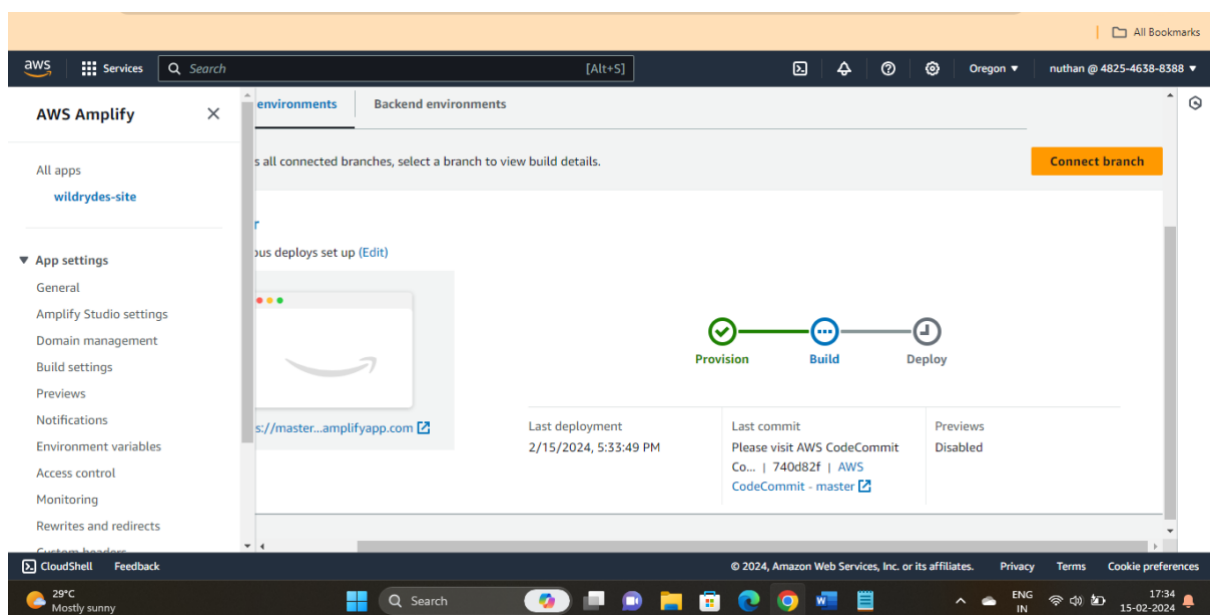
Commit changes

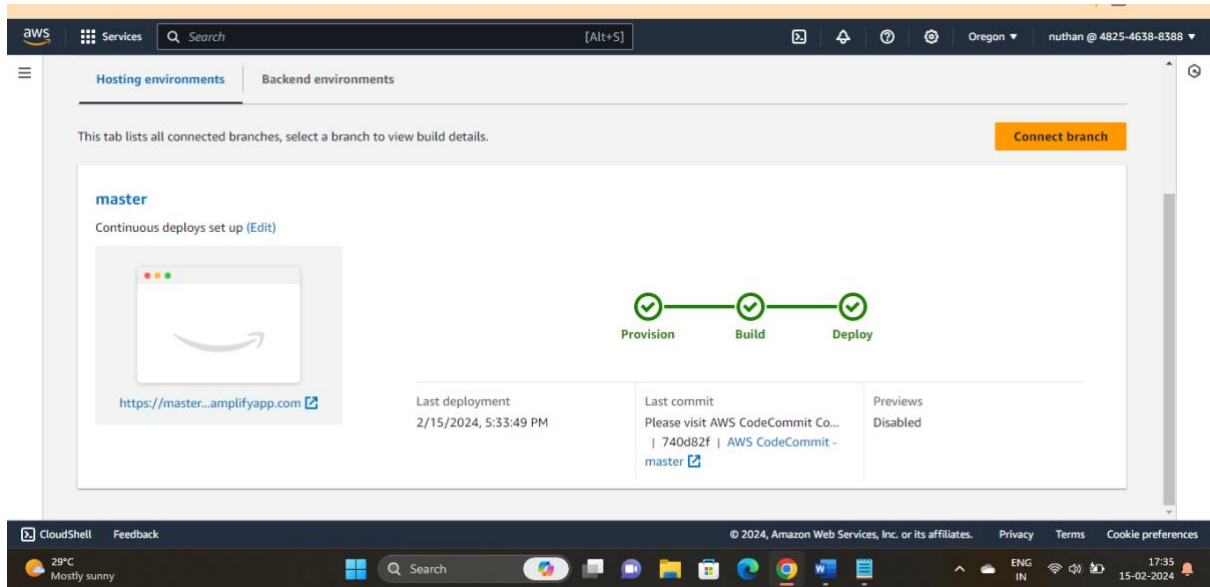


Goto -> code commit -> repo "wildrydes-site" -> ride.html file and edit js version 4.3 to 4.6.

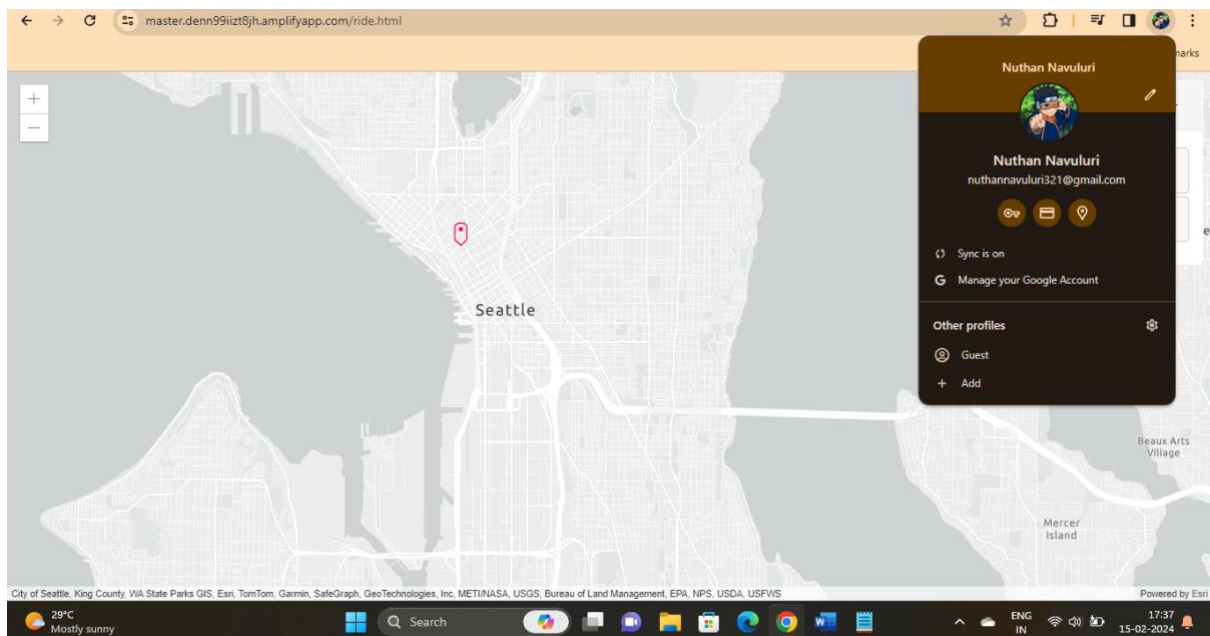


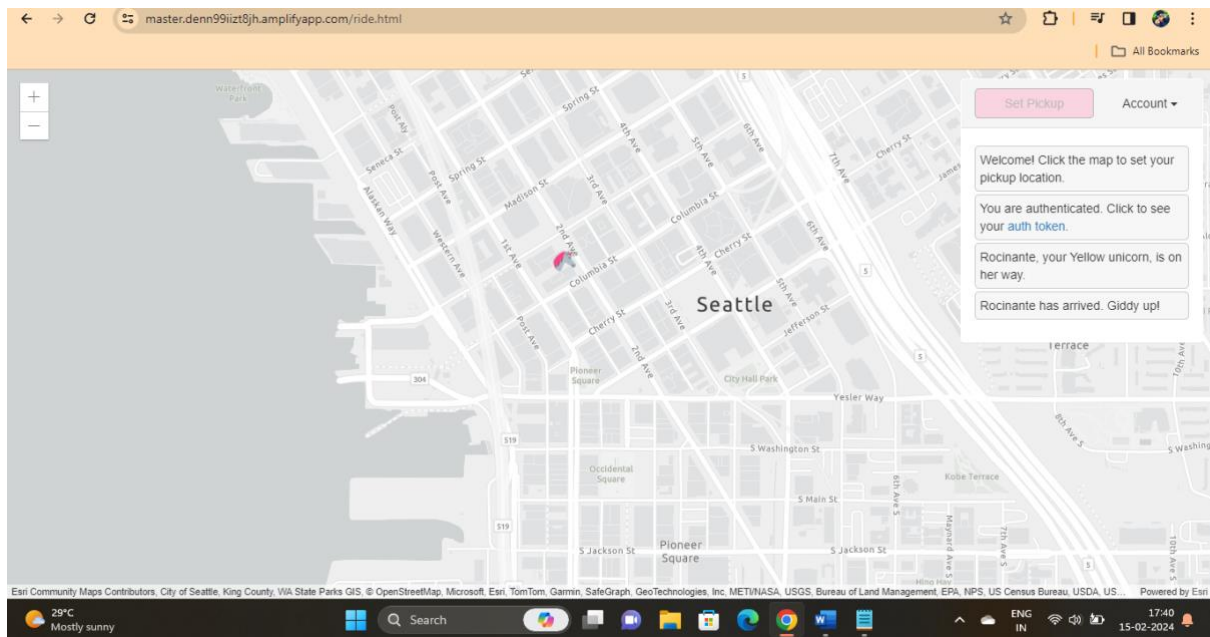
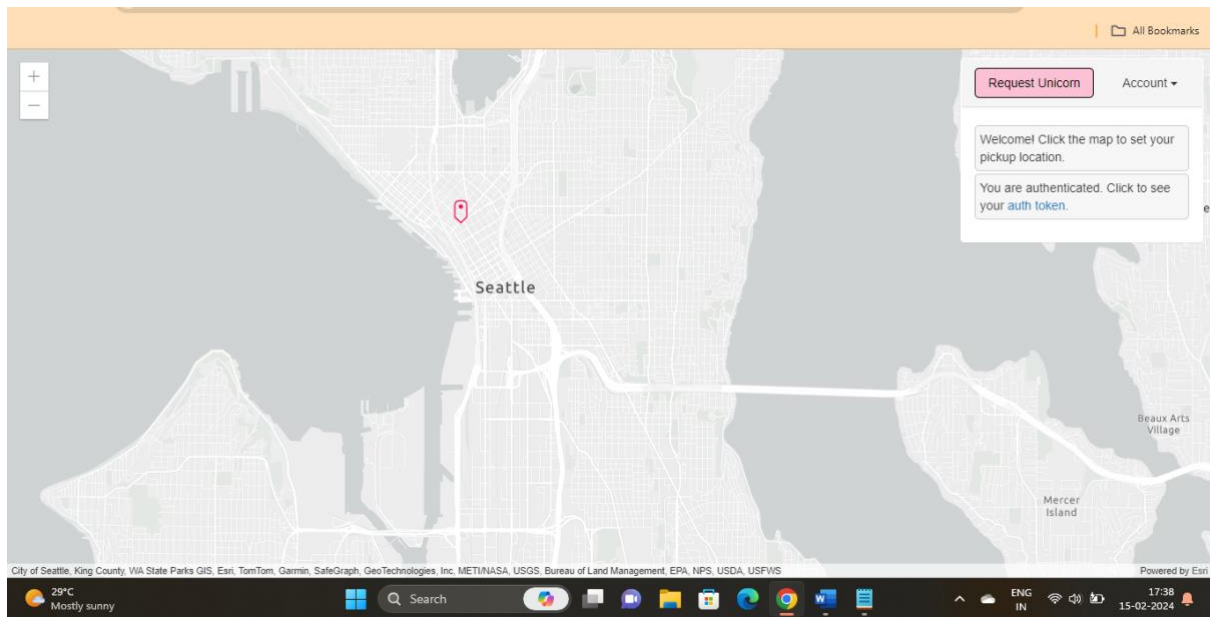
After committing amplify deploys application once again .





Successfully code is deployed Check with amplify RL . we get below desired output showing map.





STEP -5 :

Delete all resources..

Project is successfully completed.....