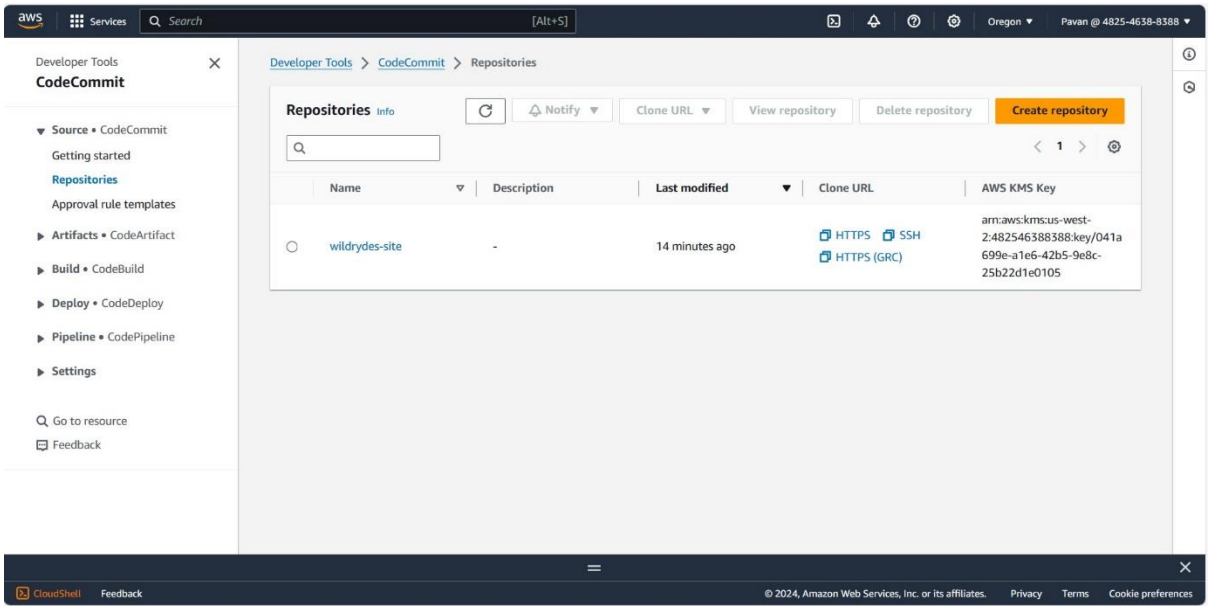
SURE TRUST MAJOR PROJECT

SEVERLESS Web Application

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

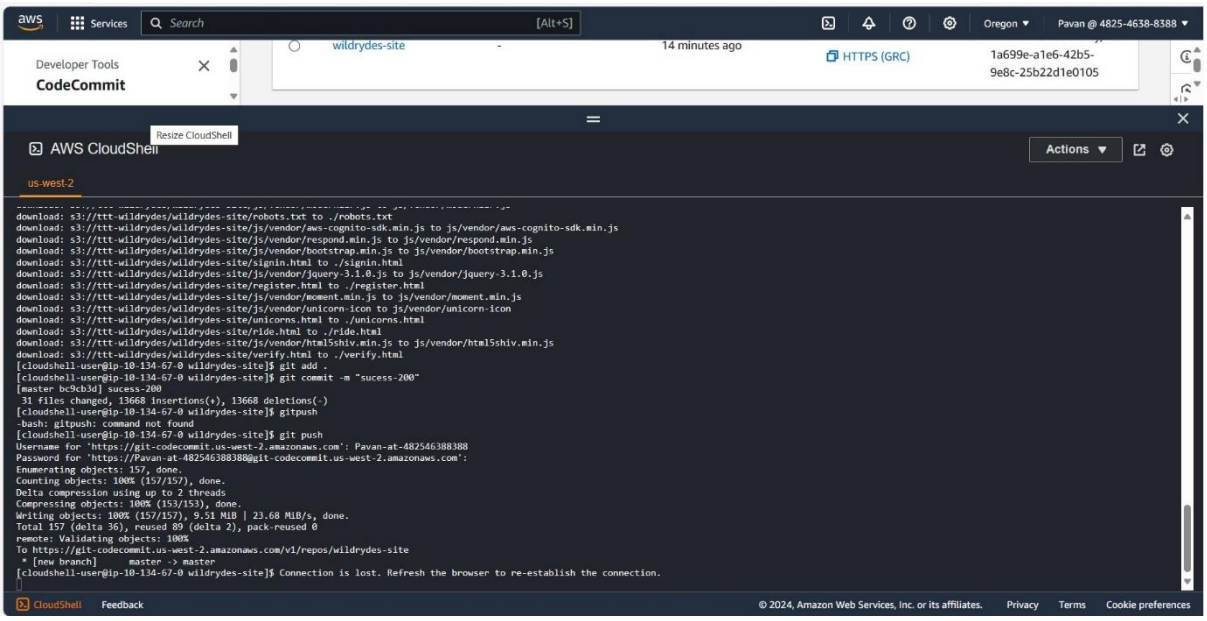
STEP-1 :



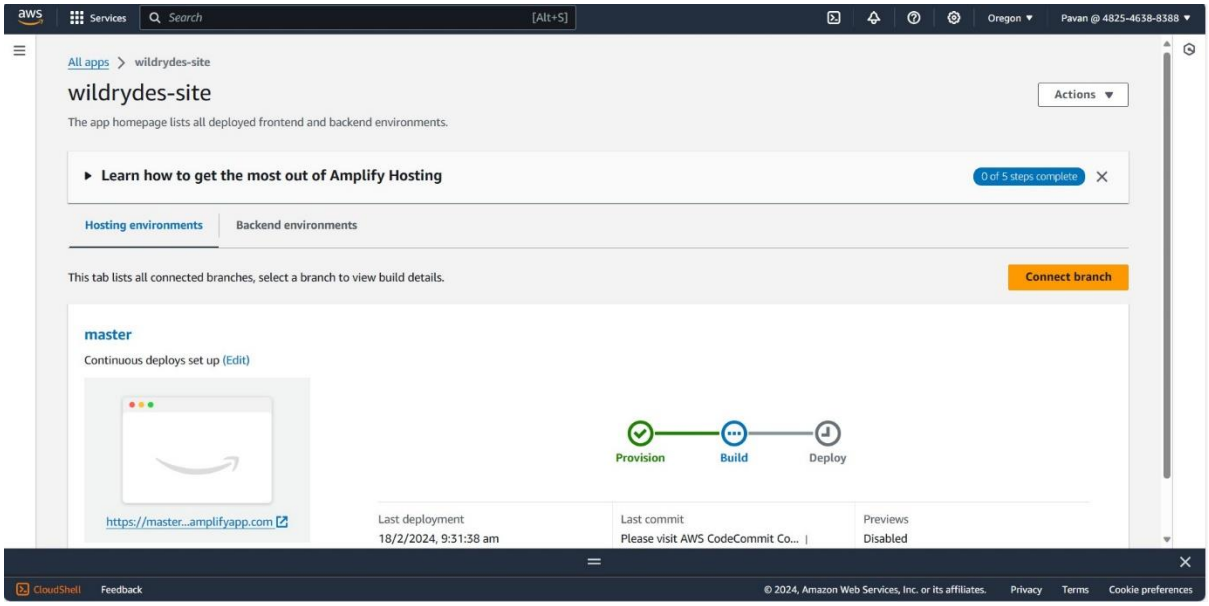
Create a repository in code commit , repo name is **“ wildrydes-site ”**

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

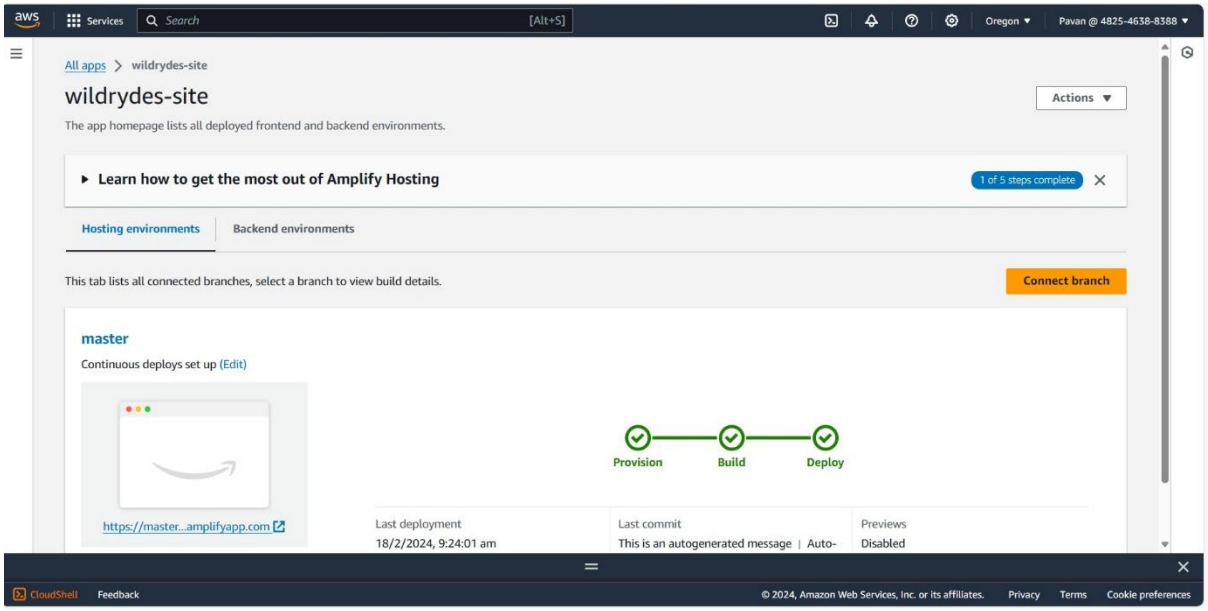
CLONE THE CODECOMMIT REPOSITORY URL USING CLOUD SHELL,COPY CODE FROM AN S3 BUCKET TO CLOUDSHELL AND PUSH THE CHANGES USING GIT COMMANDS GIT ADD,GIT COMMIT AND GIT PUSH.



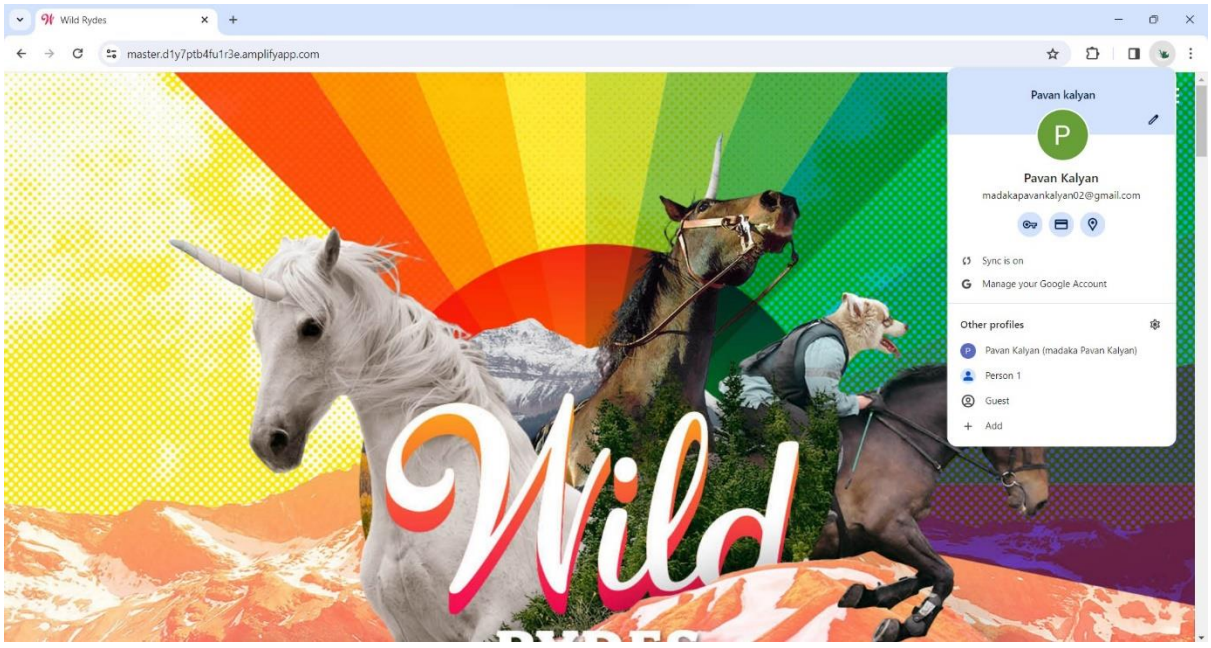
Open **aws amplify** and create a web hosting application and deploy source code from created repository **“wildrydes-site”** of codecommit



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

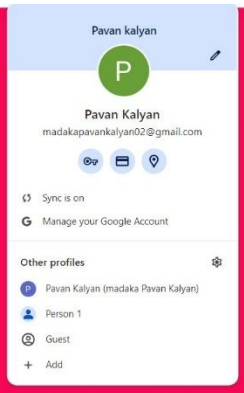


After successful deployment , the website is ready . we can see it by opening the link beside of status bar.



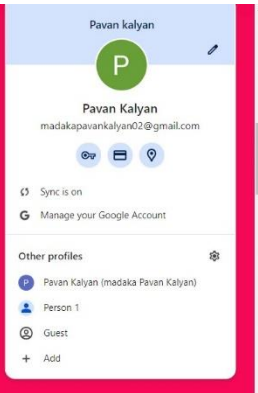
HOW DOES THIS WORK?

In today's fast paced world, you've got places you need to be but not enough time in your jam packed schedule. Wouldn't it be nice if there were a transportation service that changed the way you get around daily? Introducing Wild Rydes, an innovative transportation service that helps people get to their destination faster and hassle-free. Getting started is as easy as tapping a button in our app.



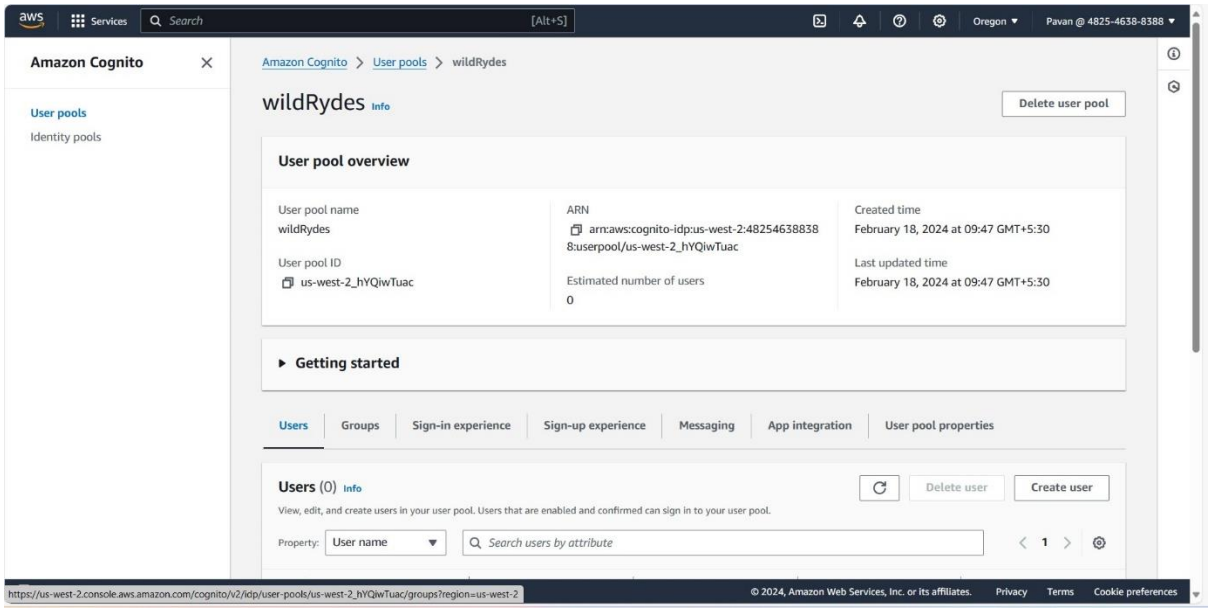
HOW DOES THIS THING WORK?

In today's fast paced world, you've got places you need to be but not enough time in your jam packed schedule. Wouldn't it be nice if there were a transportation service that changed the way you get around daily? Introducing Wild Rydes, an innovative transportation service that helps people get to their destination faster and hassle-free. Getting started is as easy as tapping a button in our app.



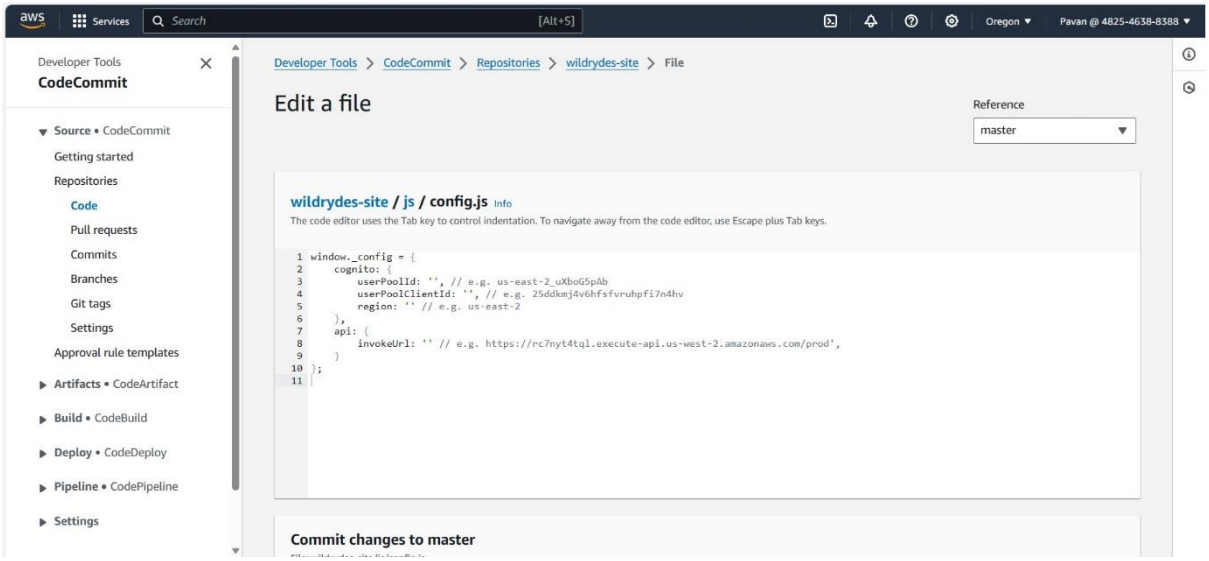
STEP-2 :

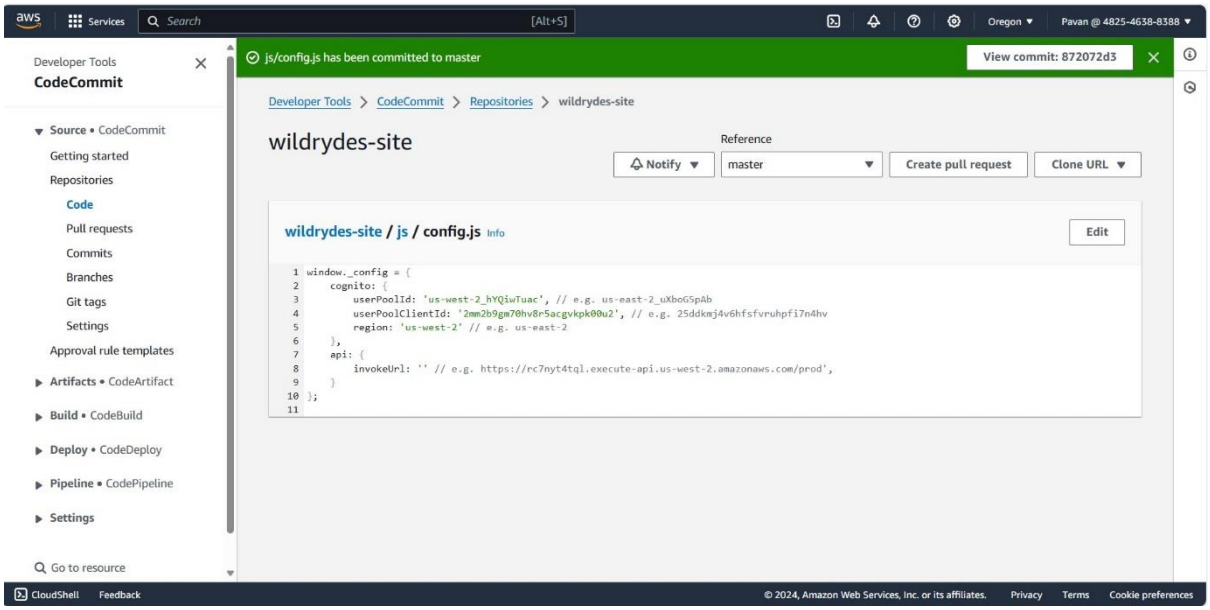
Goto **aws cognito** and create a user pool with name " project-userpool" with **no-MFA and No-attributes**



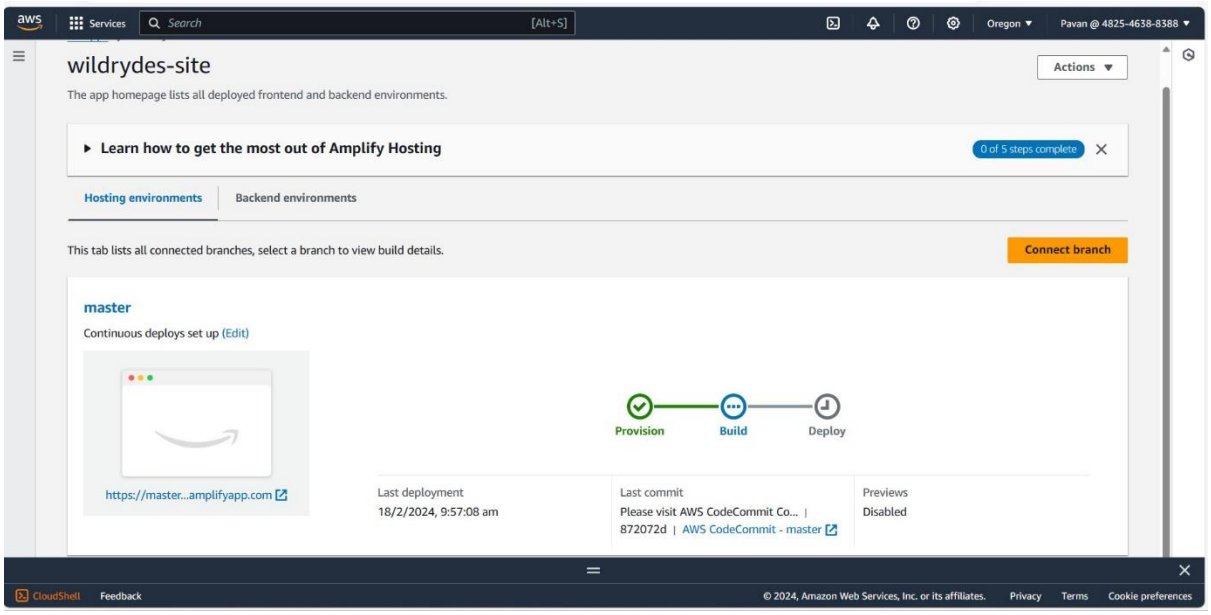
Copy userpool-ID and client ID of userpool

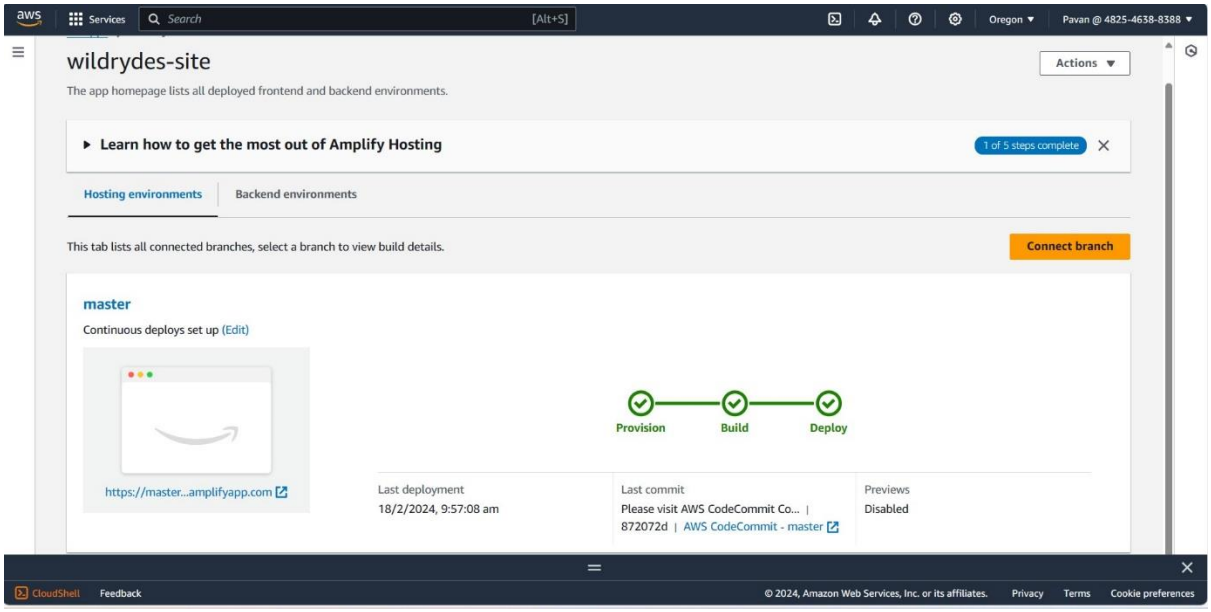
Goto **codecommi**t -> **wildrydes-site** repo -> **js** -> **config.js** file and edit file by adding userpool-id , client ID and region.

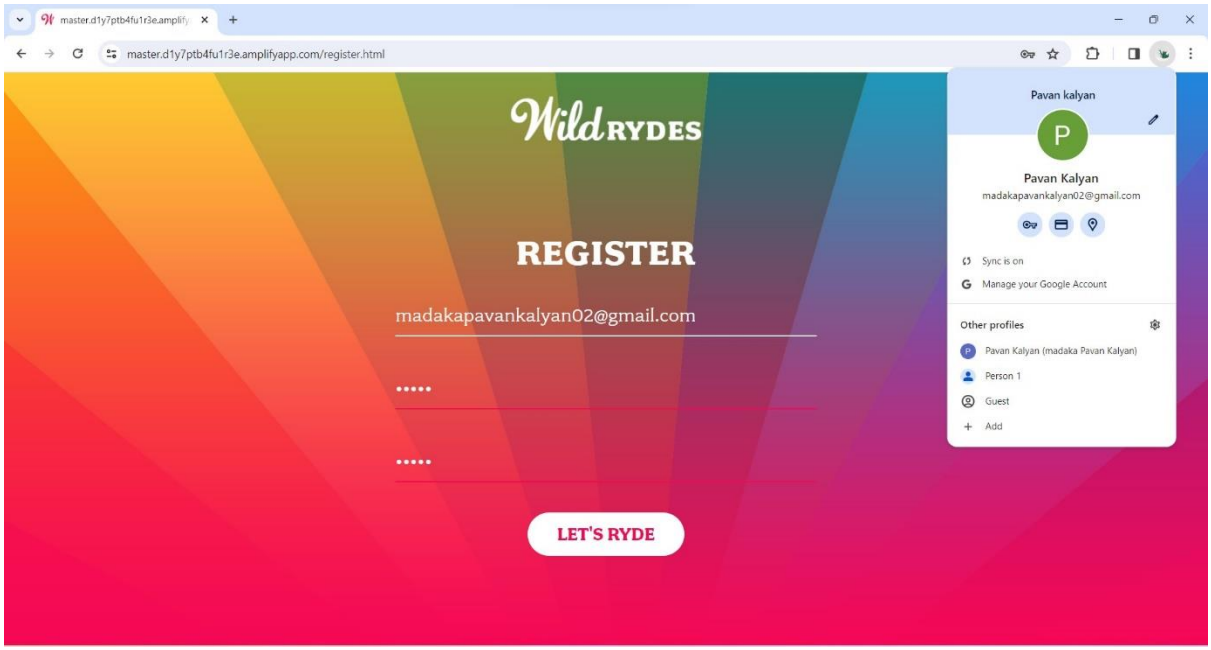




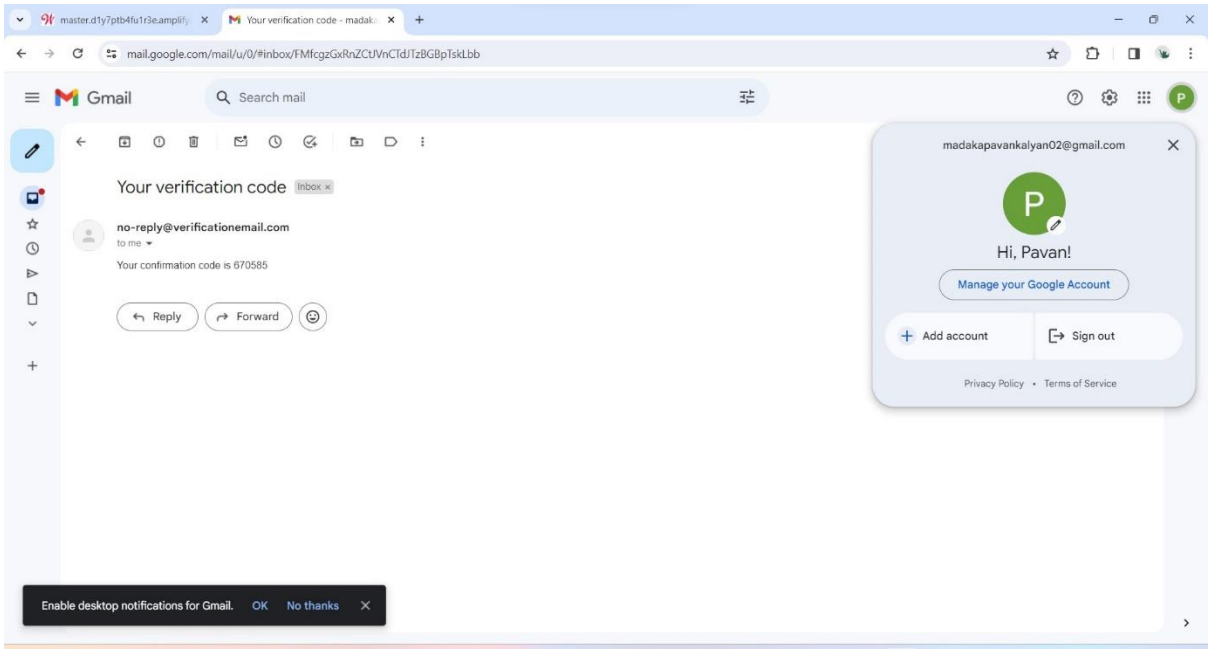
After commiting code , automatically , the code in repo is rebuit and deployed into aws amplify

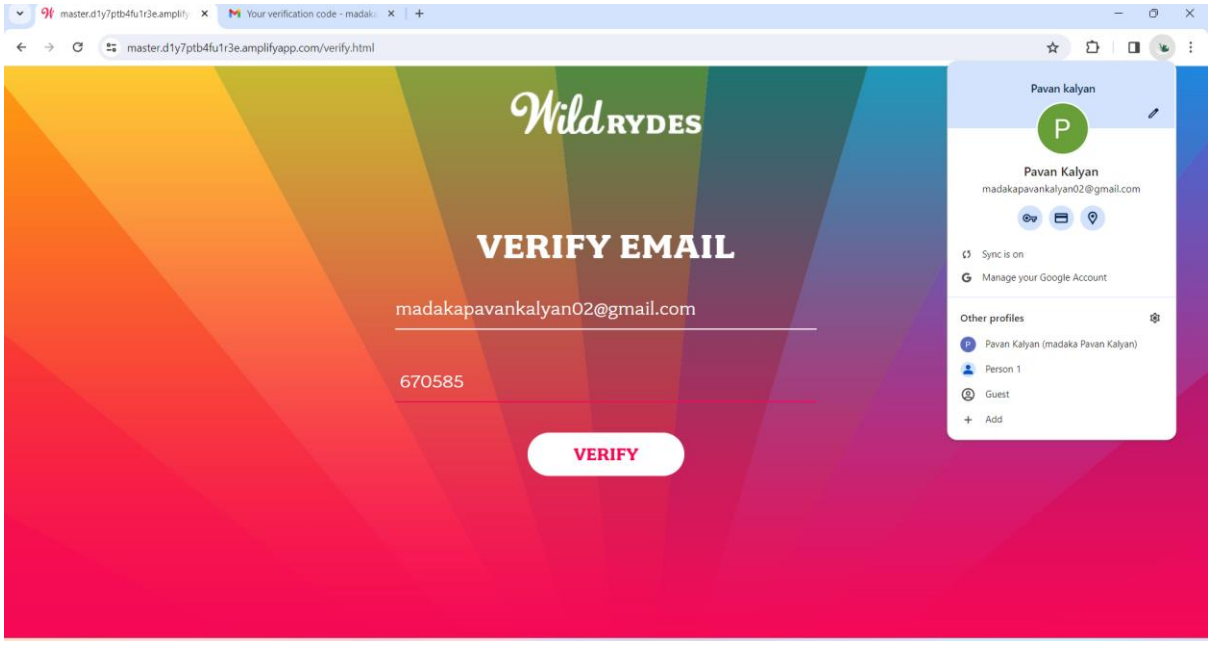




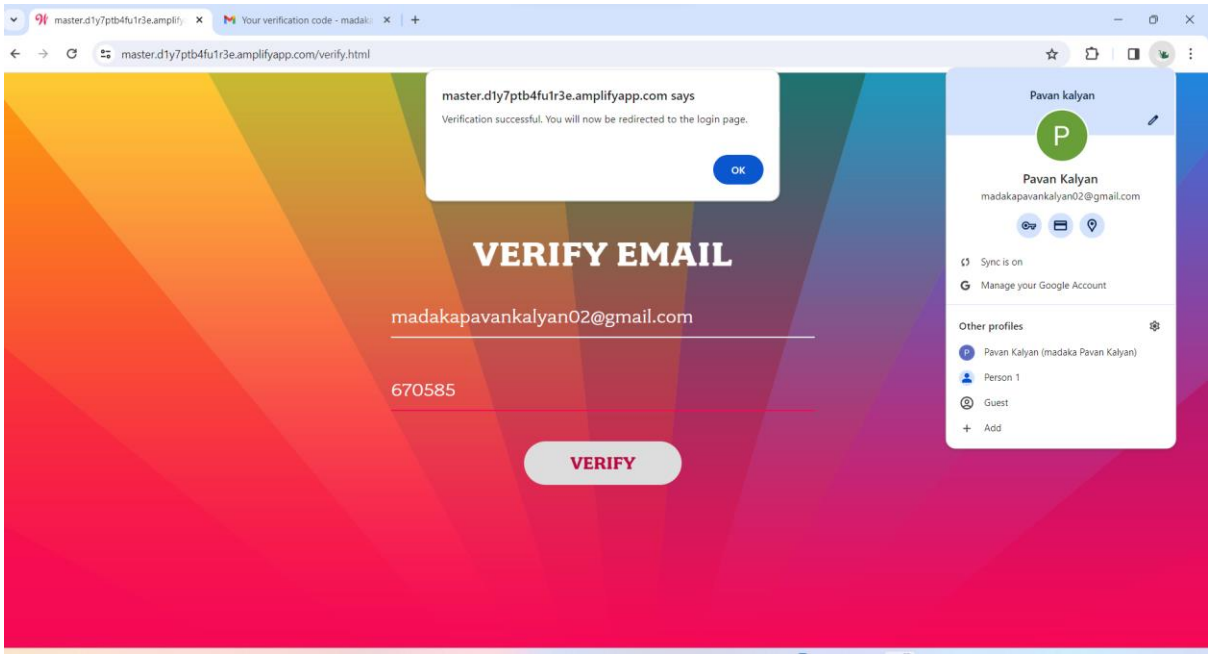


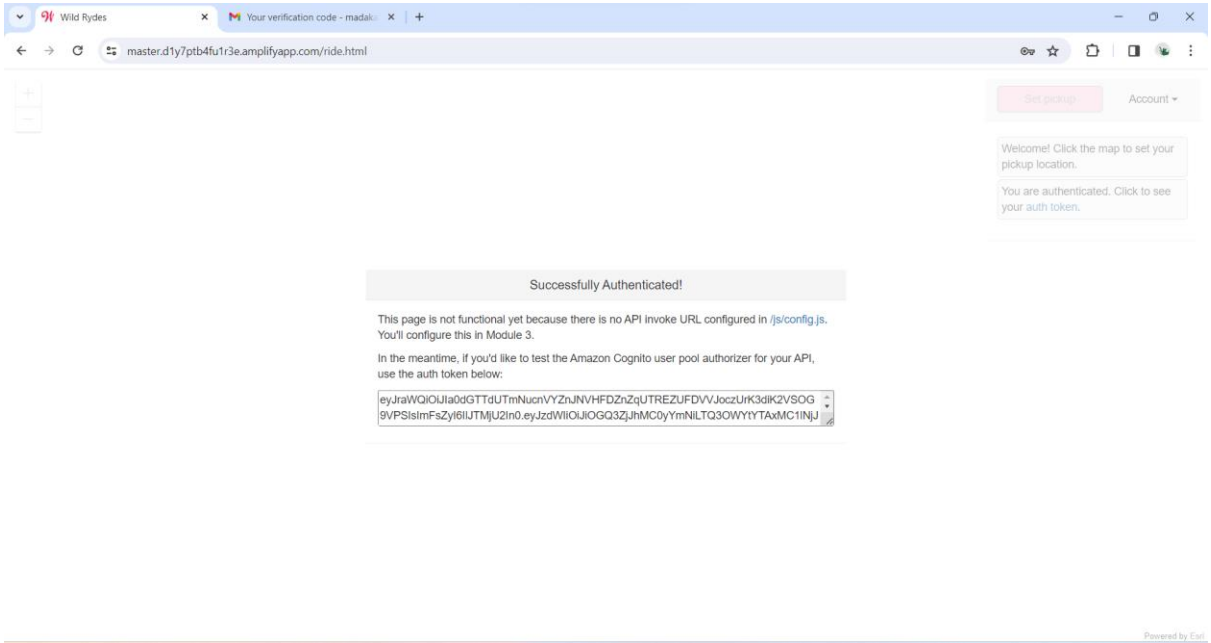
On clicking **lets ride** -> otp will sent to email for verifying email address.

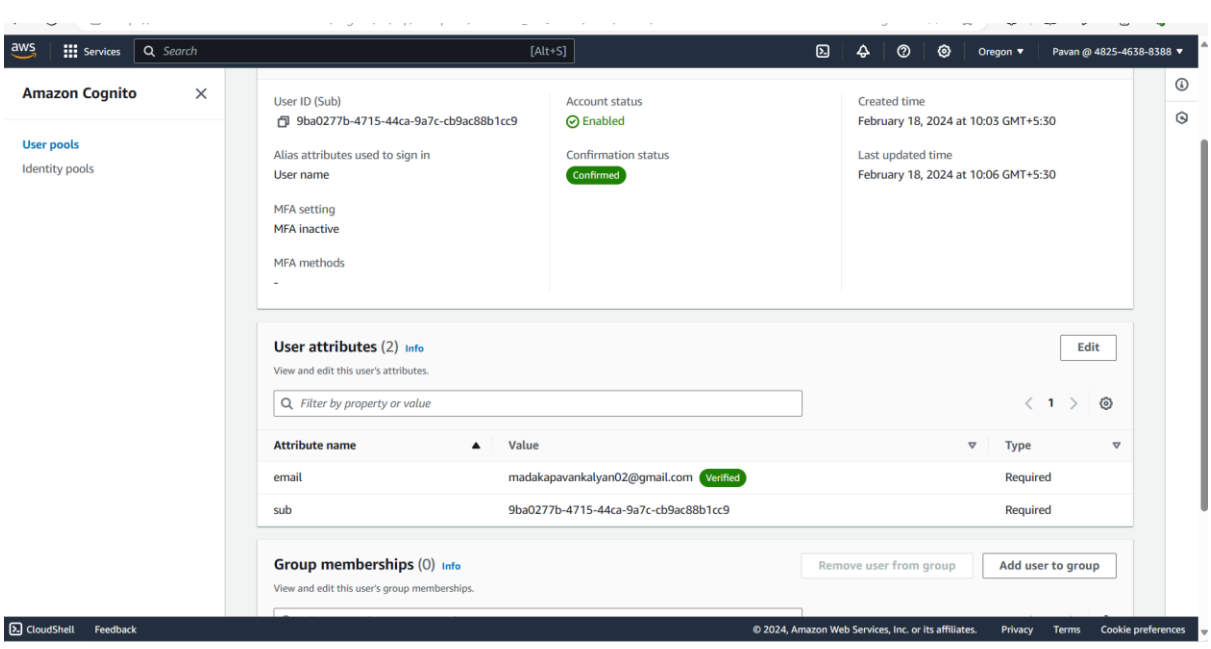




verification of mail is successful , we can login into website


master.d1y7ptb4fu1r3e.amplifyapp.com/verify.html
master.d1y7ptb4fu1r3e.amplifyapp.com says
Verification successful. You will now be redirected to the login page.
OK
VERIFY EMAIL
madakapavankalyan02@gmail.com
670585
VERIFY
Pavan kalyan
Pavan Kalyan
madakapavankalyan02@gmail.com
Sync is on
Manage your Google Account
Other profiles
Pavan Kalyan (madaka Pavan Kalyan)
Person 1
Guest
Add


master.d1y7ptb4fu1r3e.amplifyapp.com/ride.html
Account
Welcome! Click the map to set your pickup location.
You are authenticated. Click to see your auth token.
Successfully Authenticated!
This page is not functional yet because there is no API invoke URL configured in /js/config.js. You'll configure this in Module 3.
In the meantime, if you'd like to test the Amazon Cognito user pool authorizer for your API, use the auth token below:
eyJraWQiOiJla0dGTTdUTmNucnVYZnJNVHFDZnZqUTREZUFDVVJoczUrK3dlK2VSOG
9VPSIsImFsZyI6IlJTMjU2In0.eyJzdWIiOiJlOGQ3ZjJhMC0yYmNiLTQ3OWYtYTAxMC1lNjJ

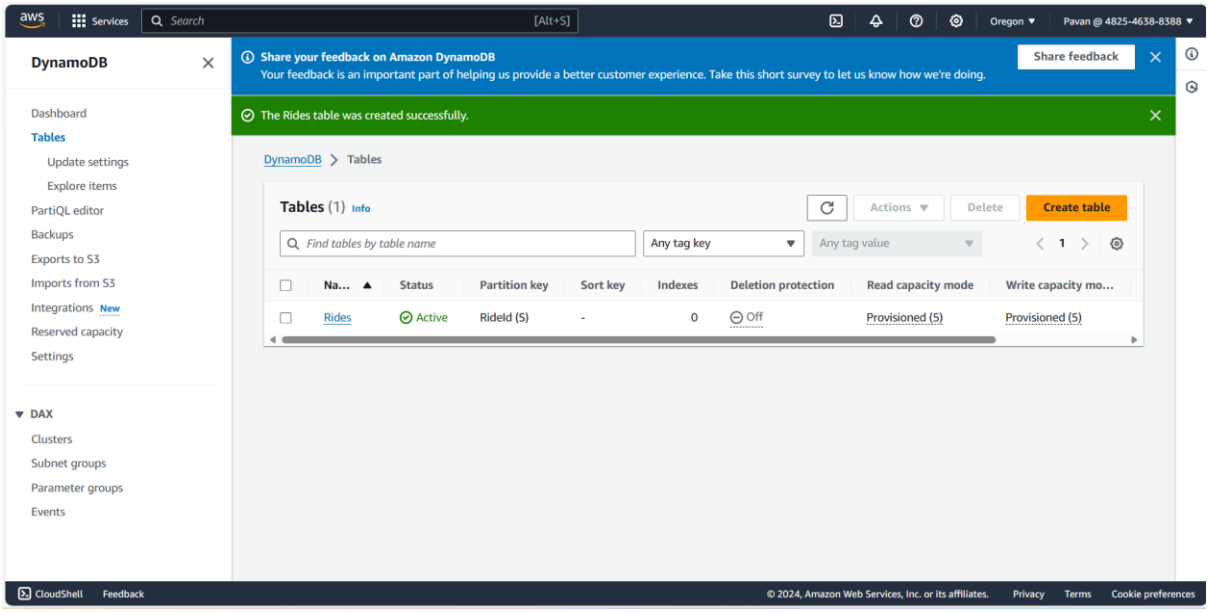

aws
Services
Search
[Alt+S]
Oregon
Pavan @ 4825-4638-8388
Amazon Cognito
User pools
Identity pools
User ID (Sub)
9ba0277b-4715-44ca-9a7c-cb9ac88b1cc9
Account status
Enabled
Created time
February 18, 2024 at 10:03 GMT+5:30
Alias attributes used to sign in
User name
Confirmation status
Confirmed
Last updated time
February 18, 2024 at 10:06 GMT+5:30
MFA setting
MFA inactive
MFA methods
-
User attributes (2) Info
View and edit this user's attributes.
Edit
Filter by property or value
Attribute name
Value
Type
email
madakapavankalyan02@gmail.com
Verified
Required
sub
9ba0277b-4715-44ca-9a7c-cb9ac88b1cc9
Required
Group memberships (0) Info
View and edit this user's group memberships.
Remove user from group
Add user to group
CloudShell
Feedback
© 2024, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

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

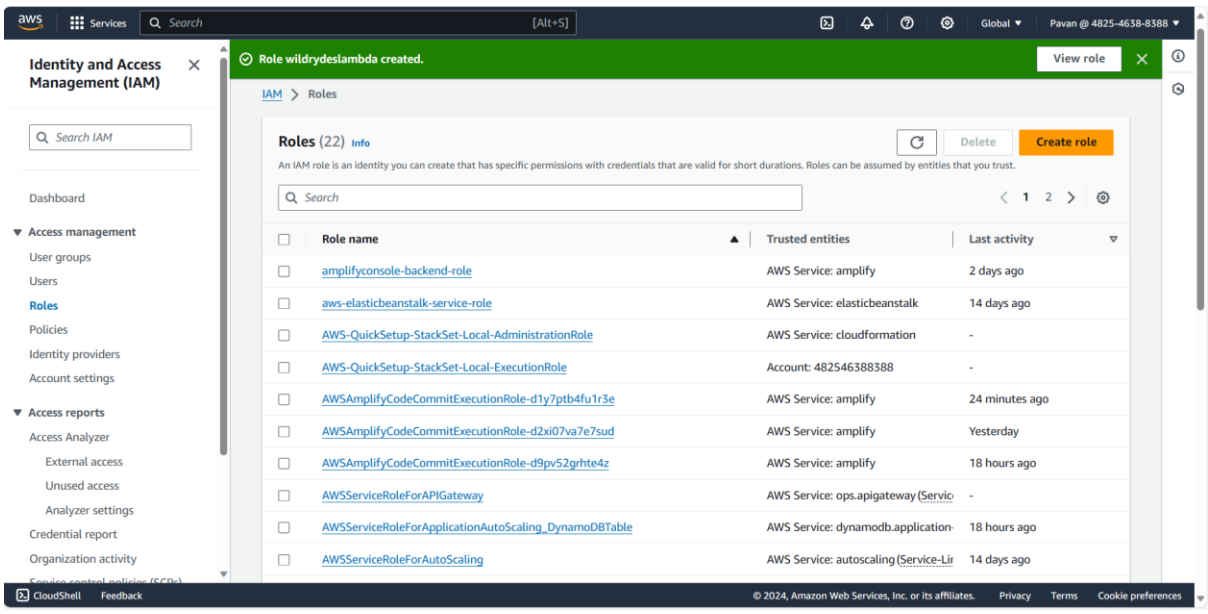
STEP-3:

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

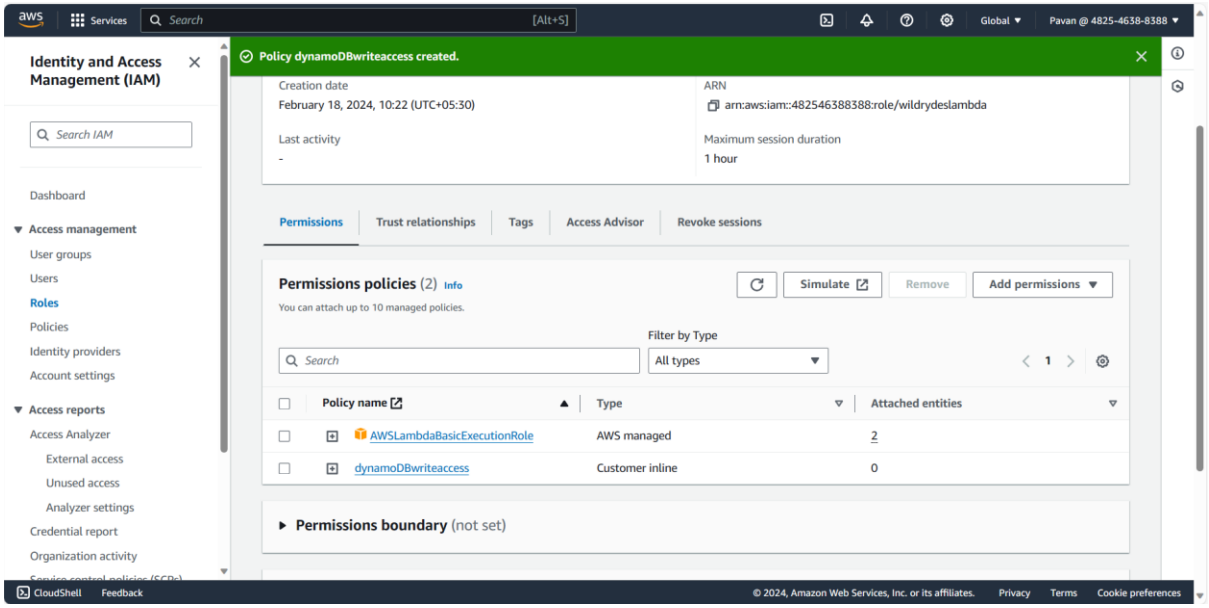
- ➔ open Dynamo DB . In that , create a table and copy ARN of it
- ➔ IMPLEMENTING RIDE SHARING RIDE FUNCTIONALITY WITH LAMBDA AND DYNAMODB. CREATE A NEW DYNAMO DB TABLE CREATING AN IAM ROLE TO BE USED FOR A LAMBDA EXECUTION ROLE ALLOWING PUTIEM IN DYNAMODB TABLE CREATING A NEW LAMBDA FUNCTION TO CHOOSE A UNICORN AND WRITE THE RIDE SHARING INFO TO DYNAMODB DEPLOYING LAMBDA CODE AND EXECUTING TEST EVENT



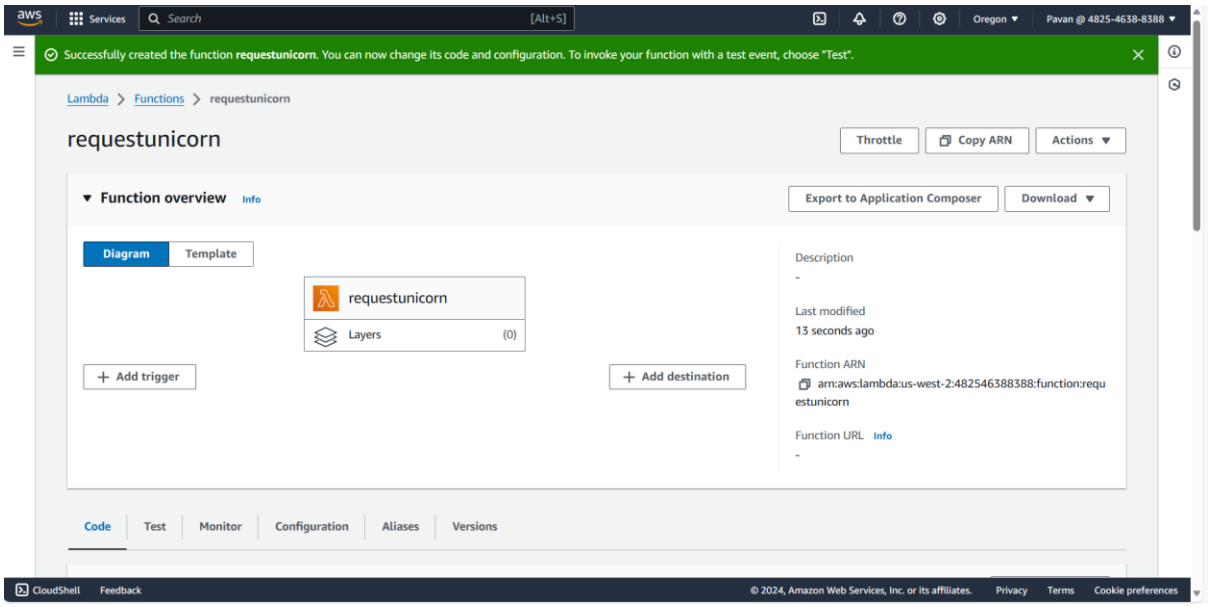
Create a **lambda** service role



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



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



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

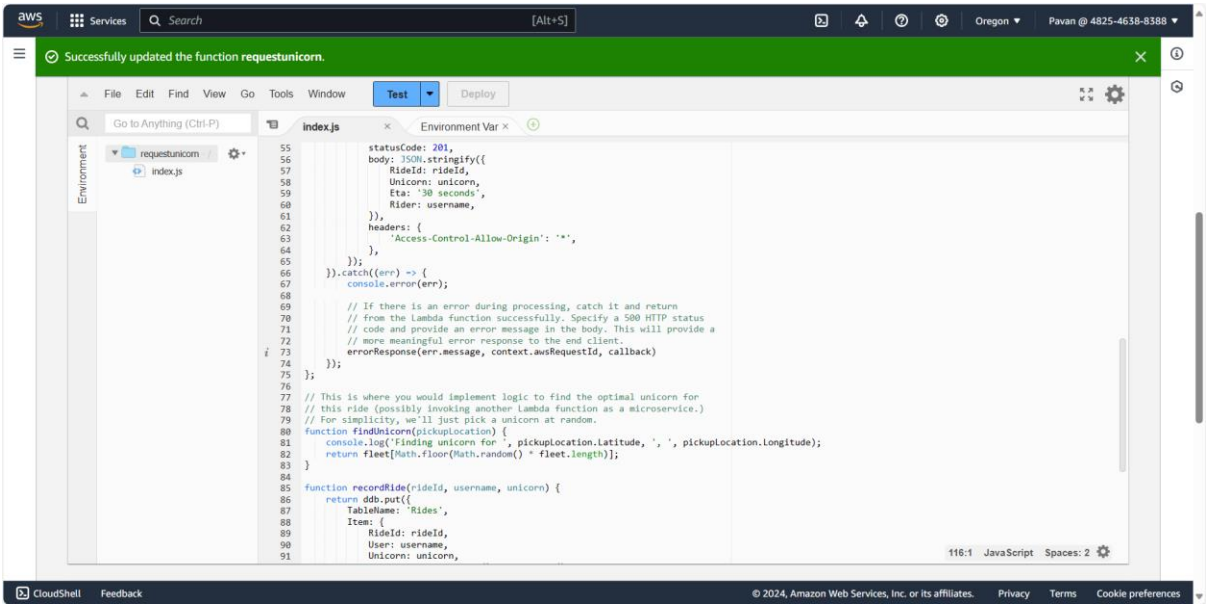
```
const randomBytes = require('crypto').randomBytes;
const AWS = require('aws-sdk');
const ddb = new AWS.DynamoDB.DocumentClient();

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 rideId = toUrlString(randomBytes(16));
    console.log('Received event (', rideId, '): ', 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(rideId, 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({
                RideId: rideId,
                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 ddb.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': '*',
    },
  });
}
```

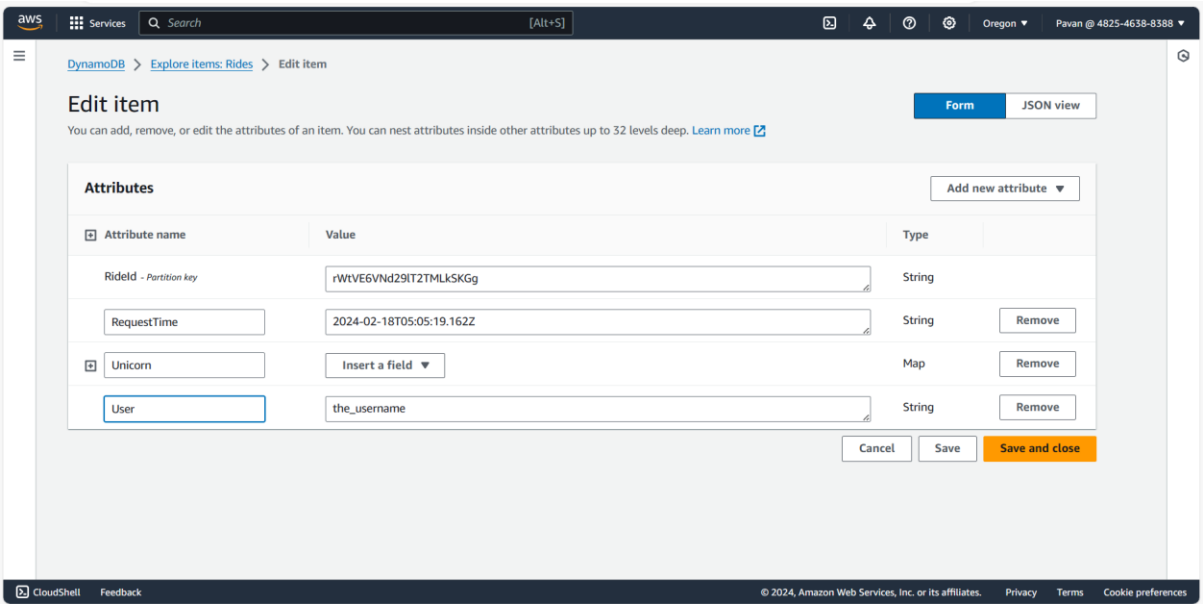
TEST EVENT FOR LAMBDA FUNCTION

```
{
    "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.



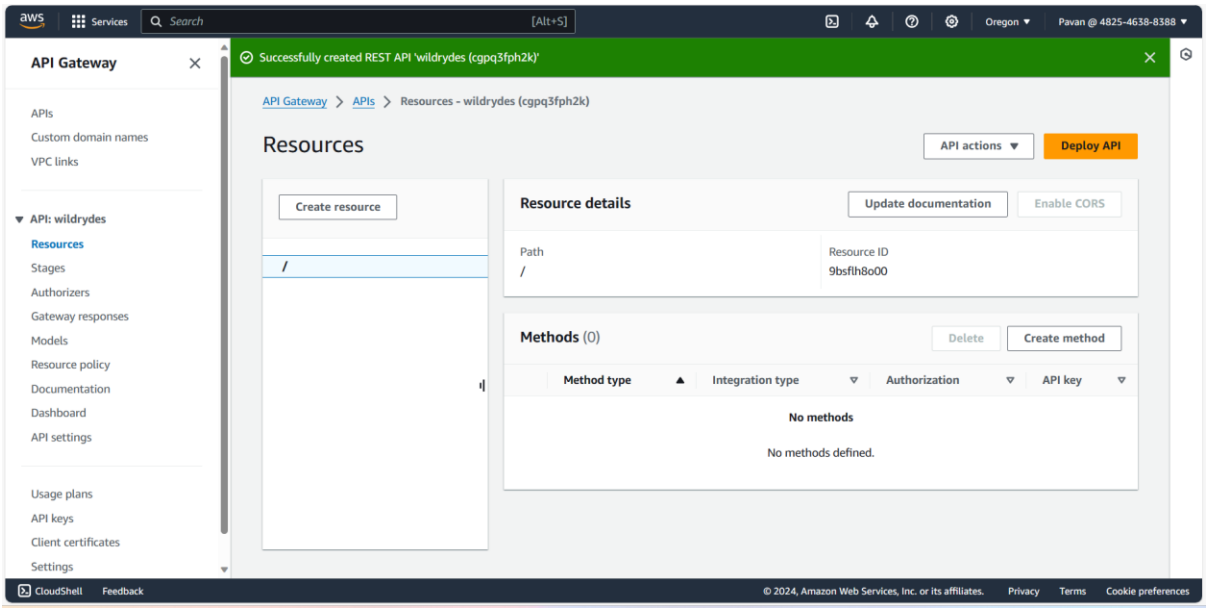
Successfully values from lambda function is write into table.



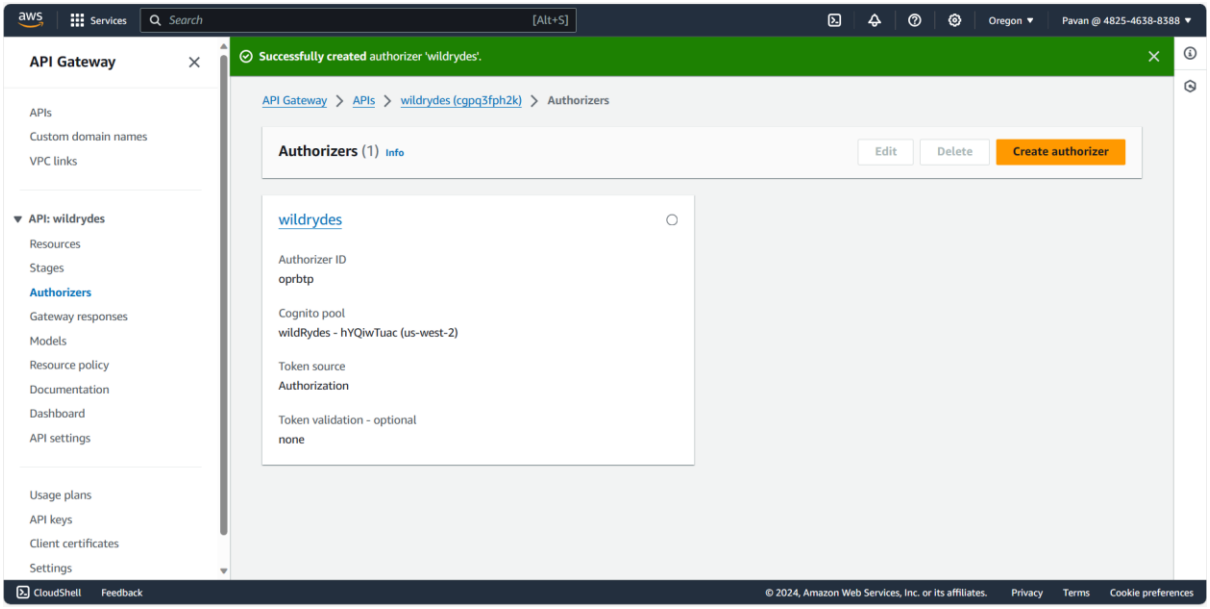
STEP-4 :

Deploy a RESTFUL API

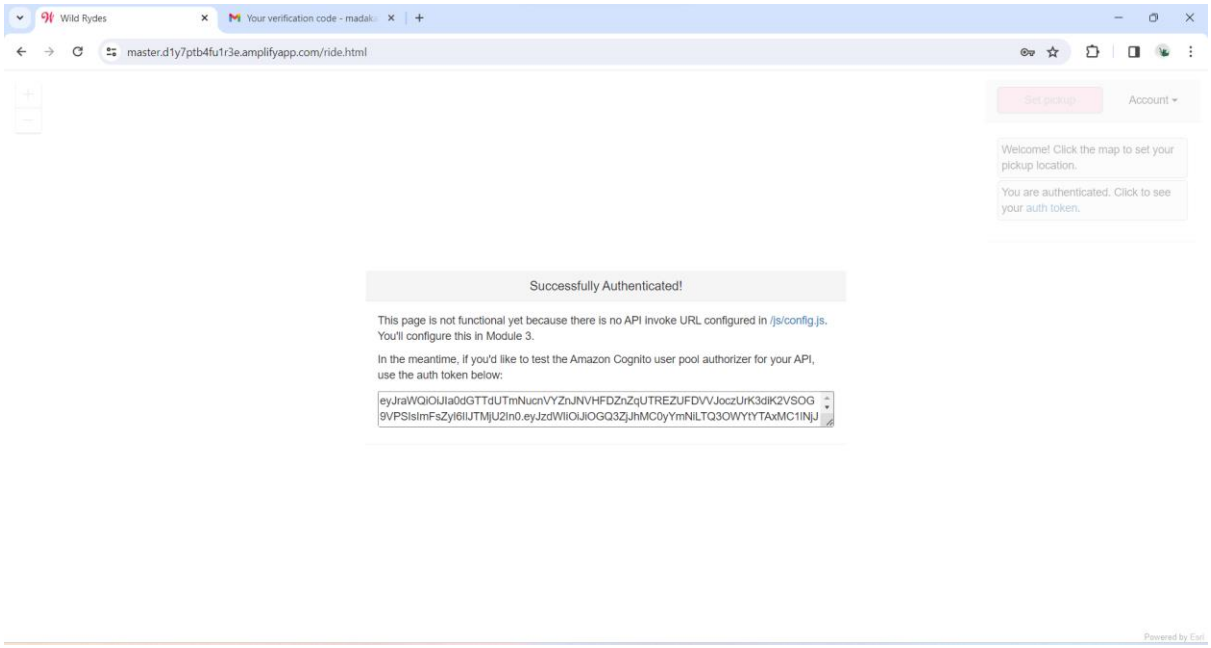
➔ **open AWS API Gateway -> create a new REST API**



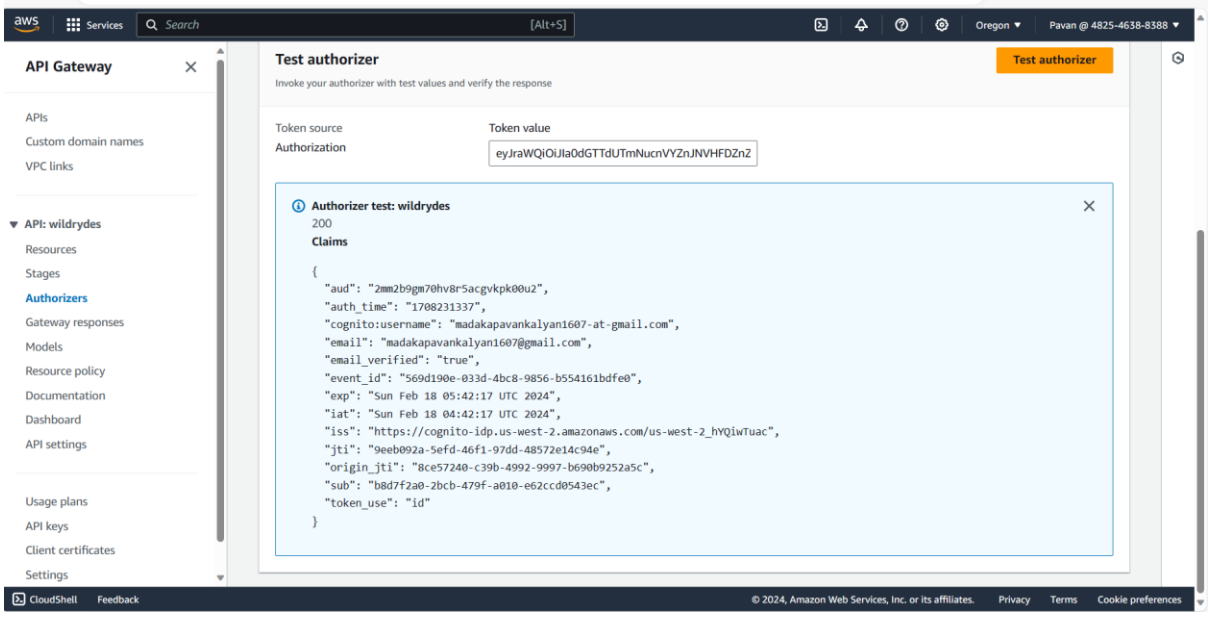
We are using cognito pool , for authencticating we need authenticator . so create a authorizer in REST API.



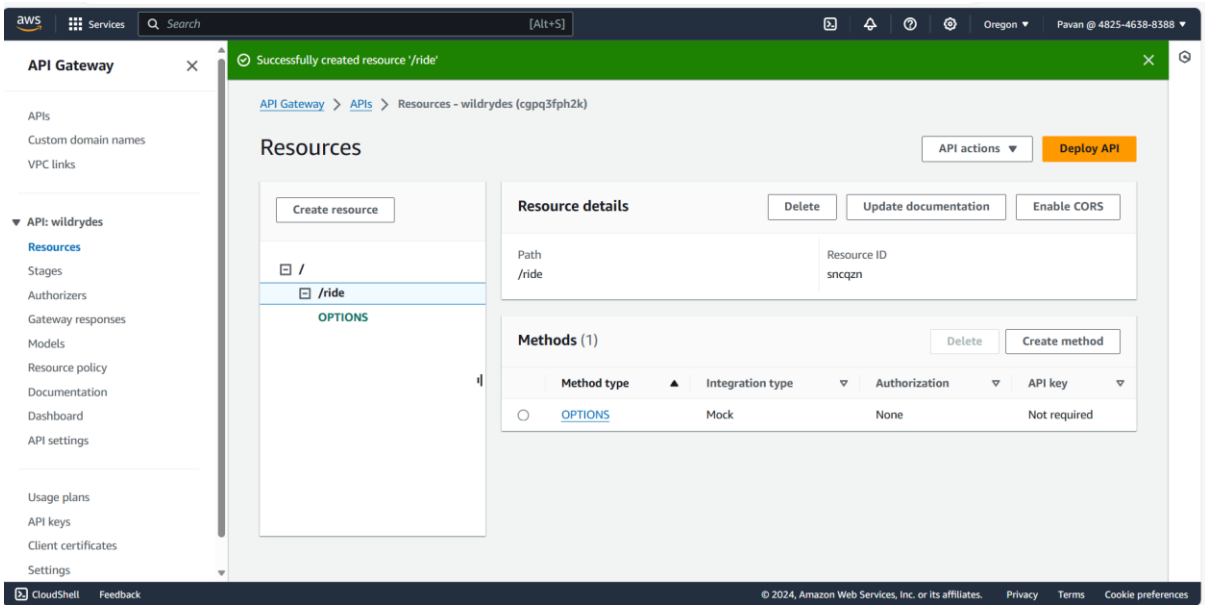
Copy authorisation token from website.



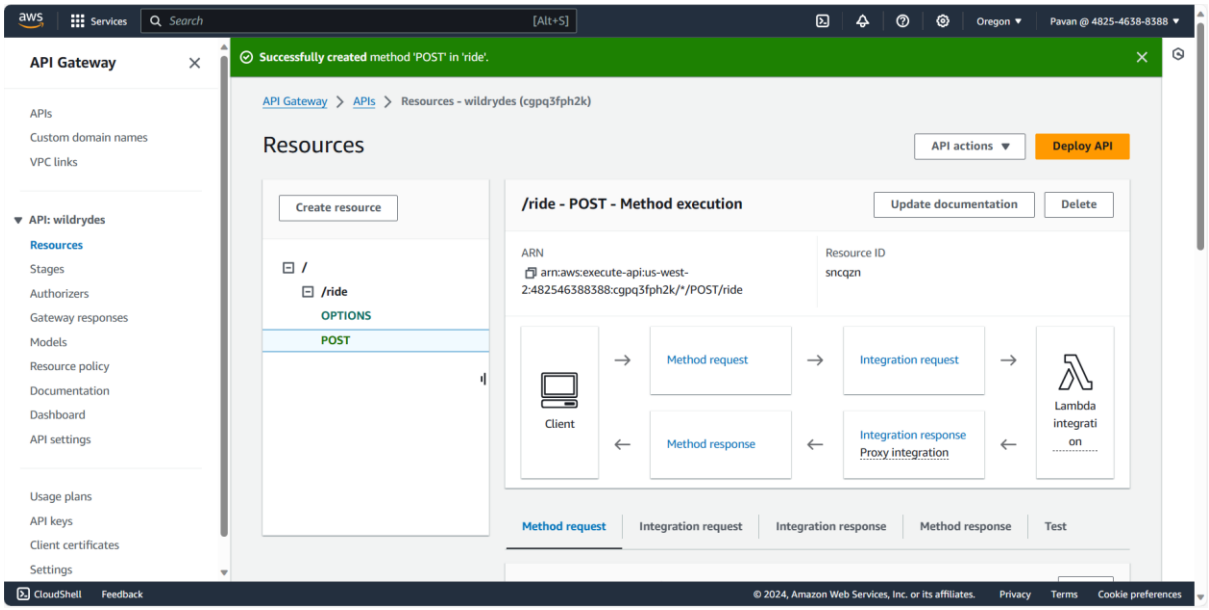
Check authorisation by checkinh in authorizer . which shows success code -200.



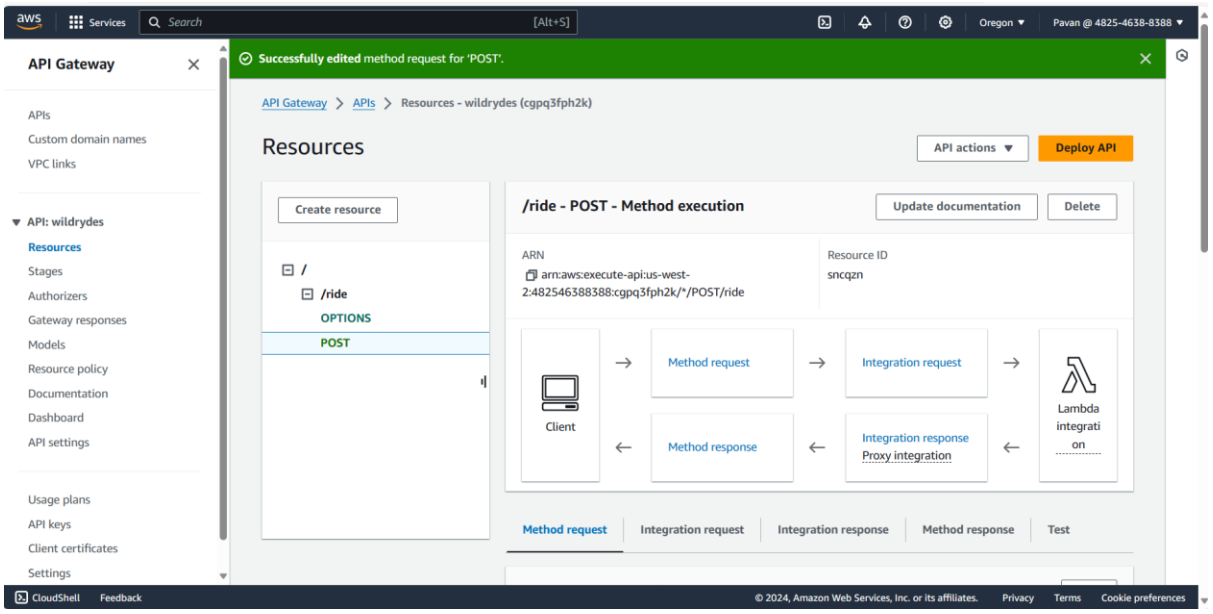
Now create a resource in REST API by enabling CORS(Cross origin resource sharing)



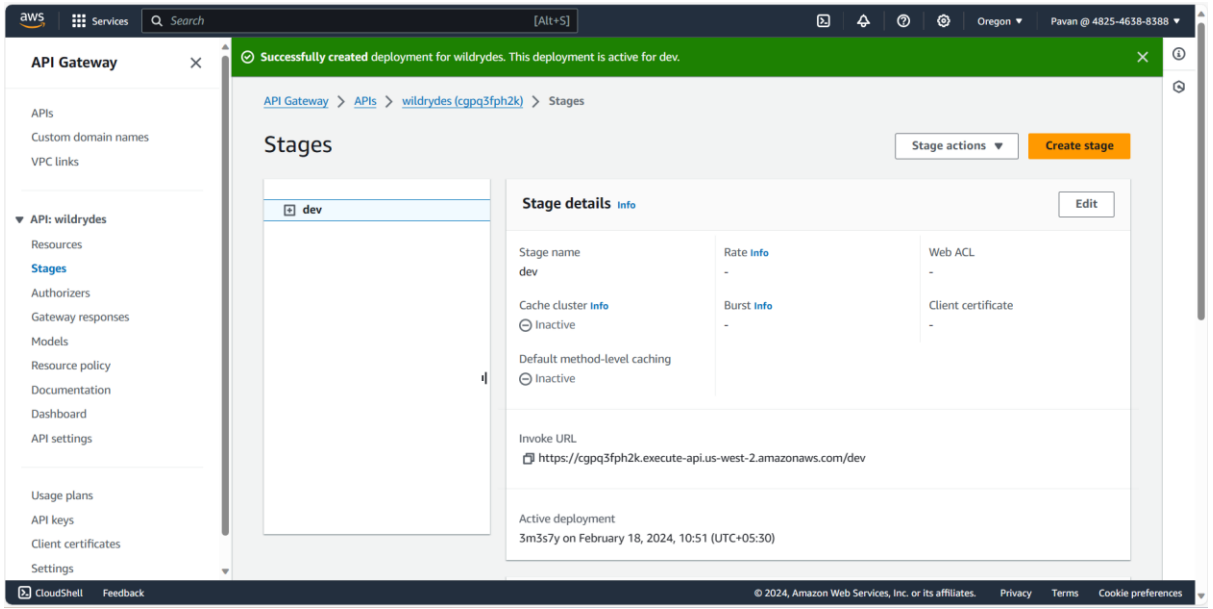
Now create POST method in resource by integrating with lambda function we created earlier.



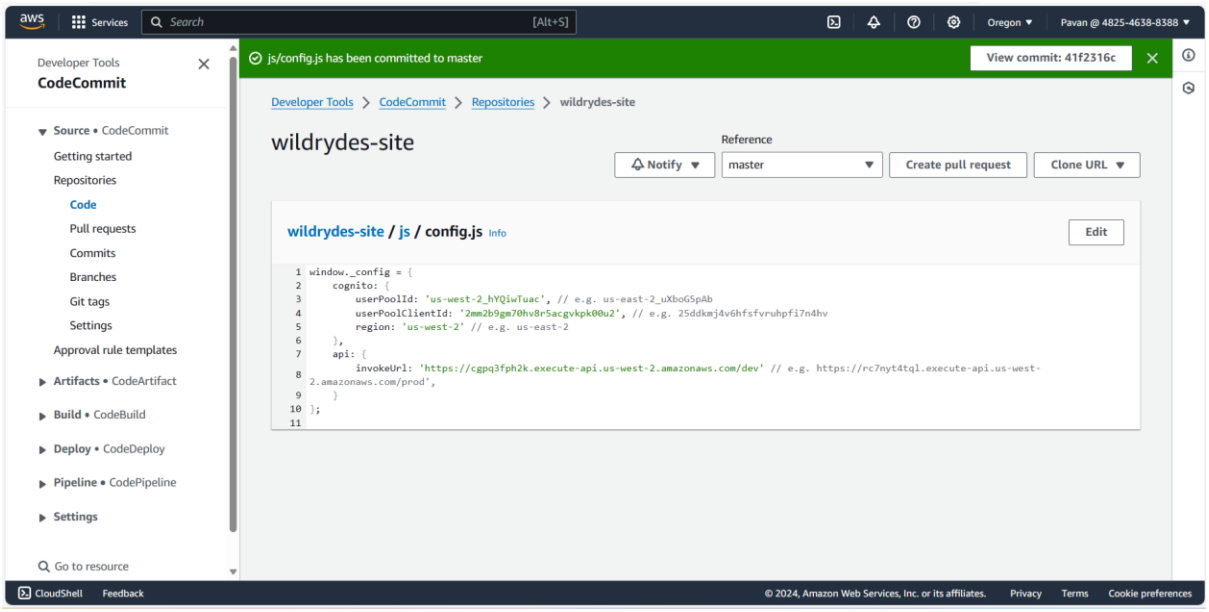
Now -> click on Method request -> edit -> add authorizerl which we created earlier.



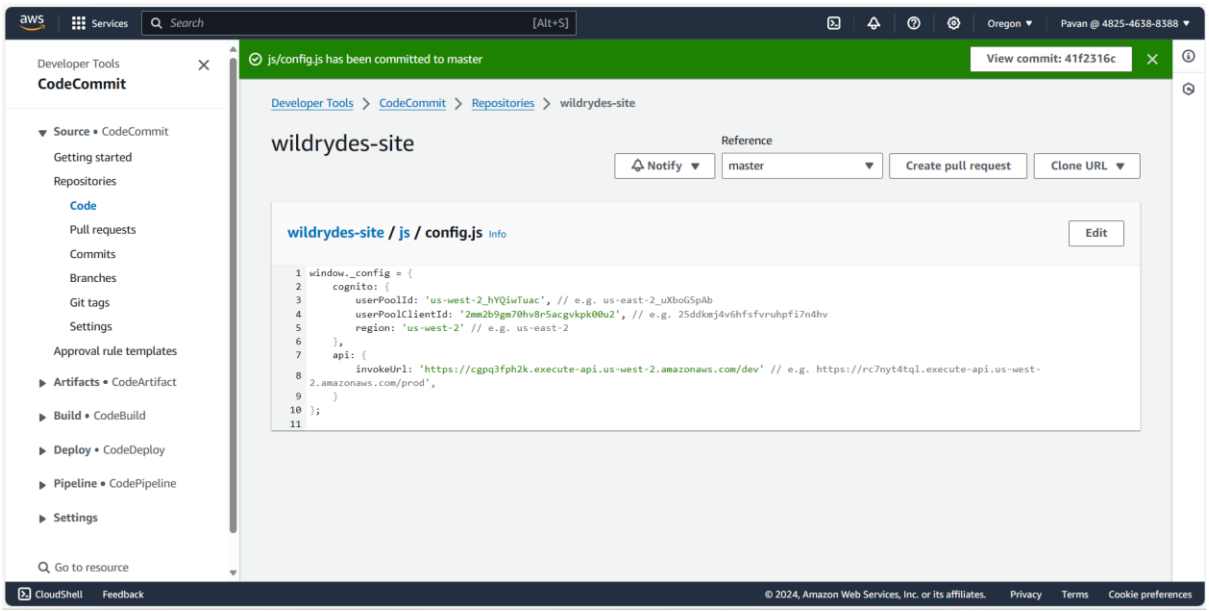
Now deploy API. After deploying , copy invoke URL



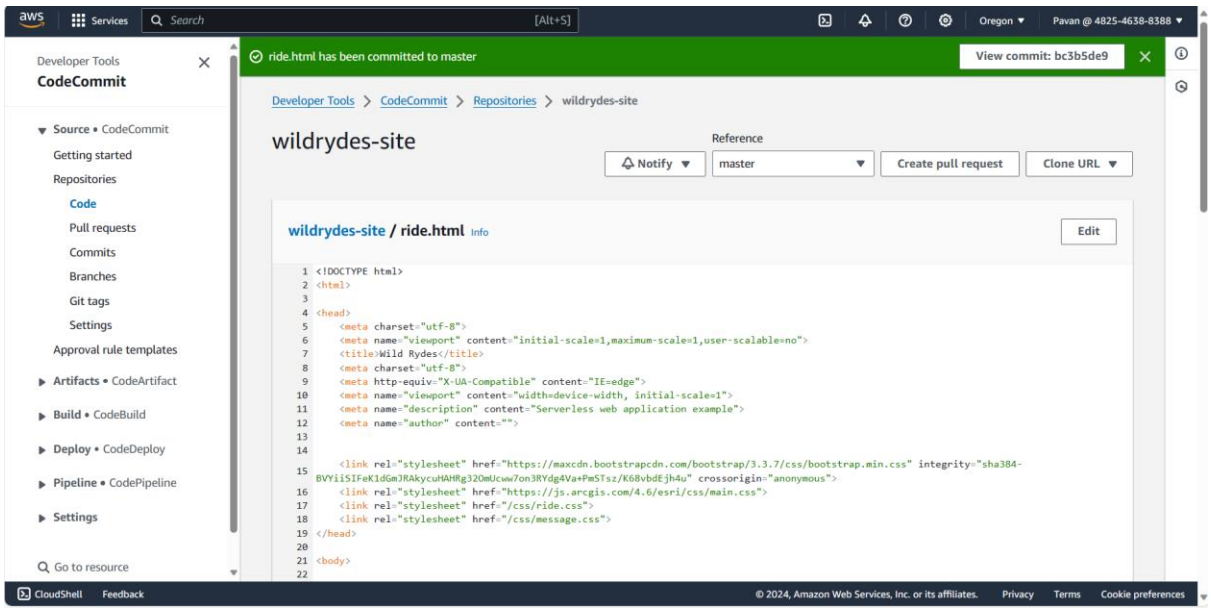
Goto -> code commit -> repo “ wildrydes-site” -> js -> config.js file and edit file. Add invoke url to the file.



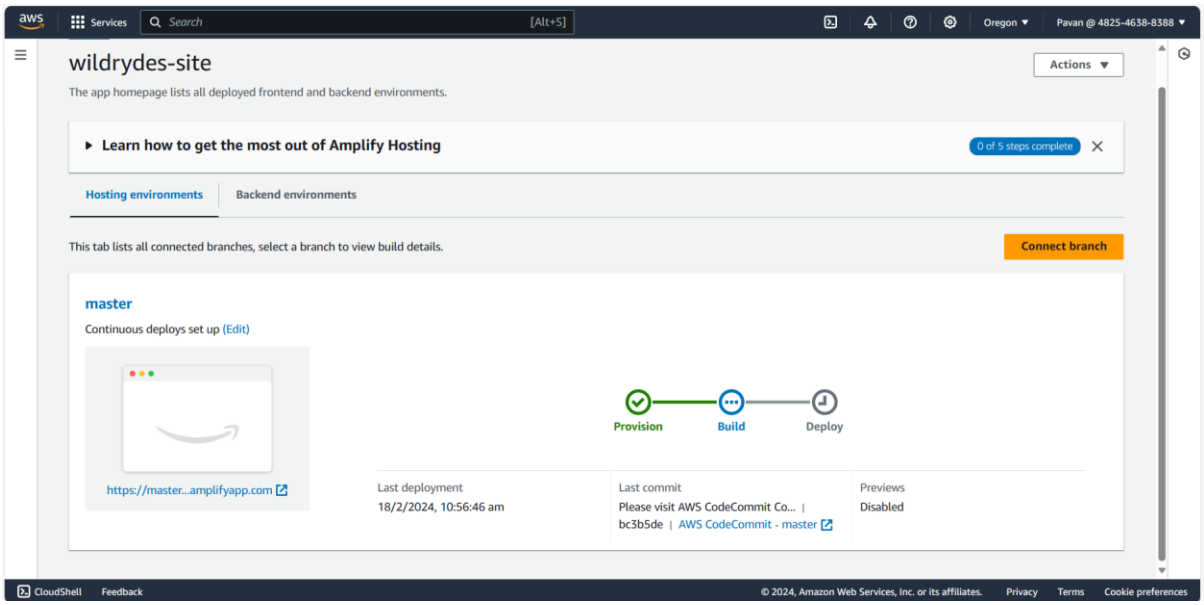
Commit changes

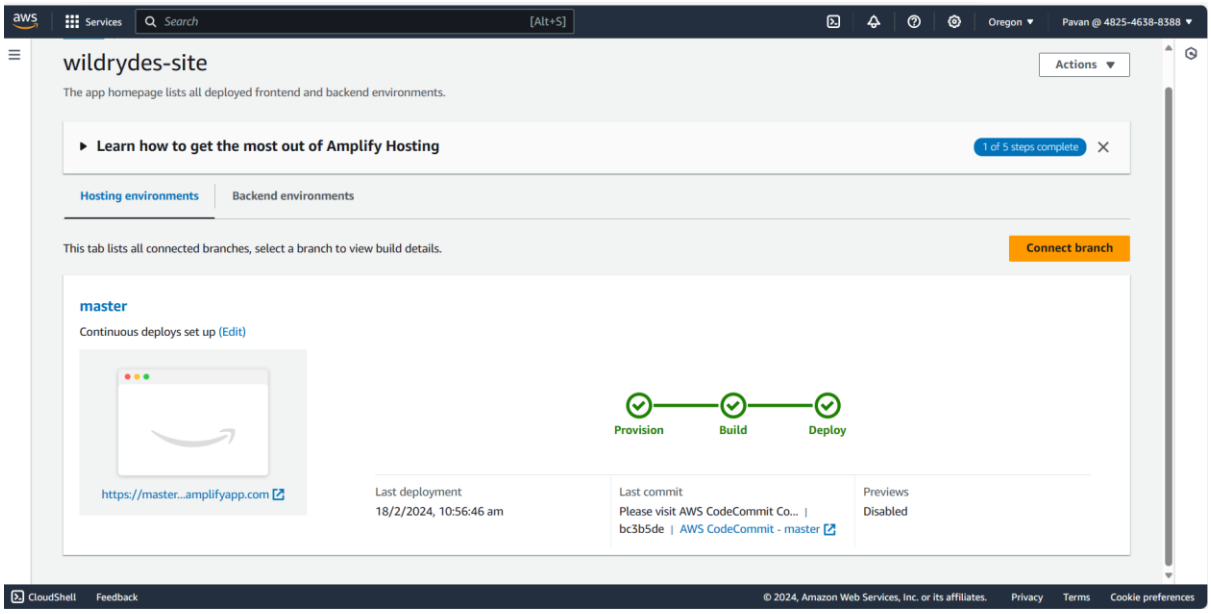


Goto -> code commit -> repo “wildrydes-site ” -> ride.html file and edit js version 4.3 to 4.6.

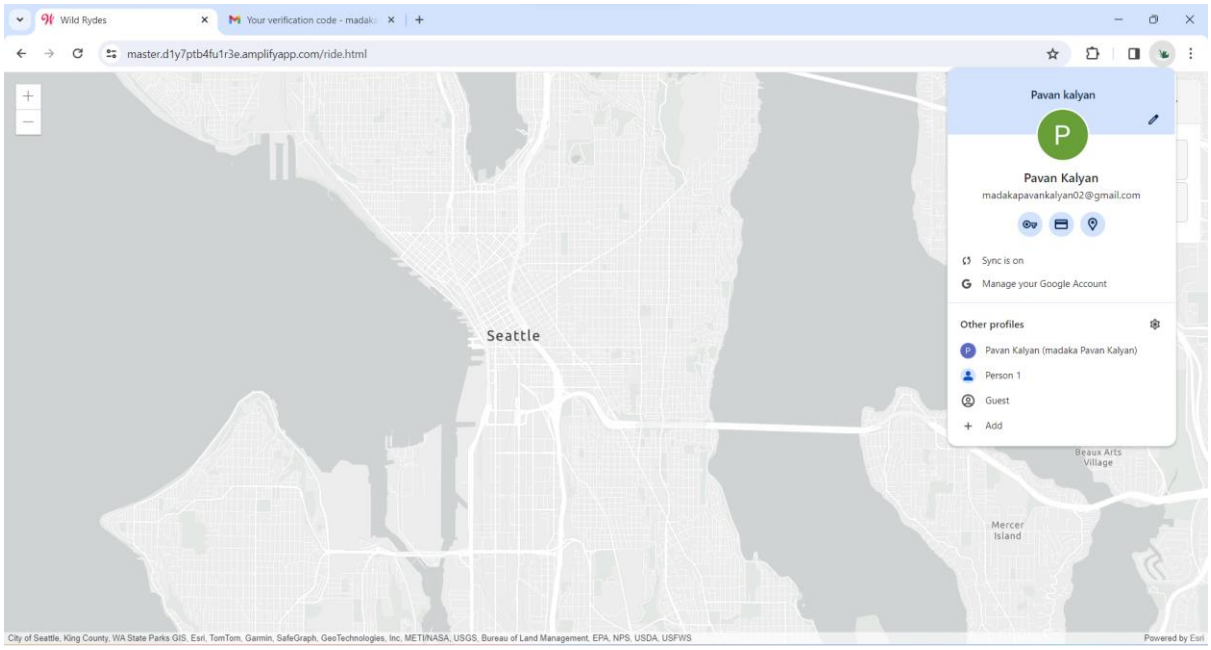


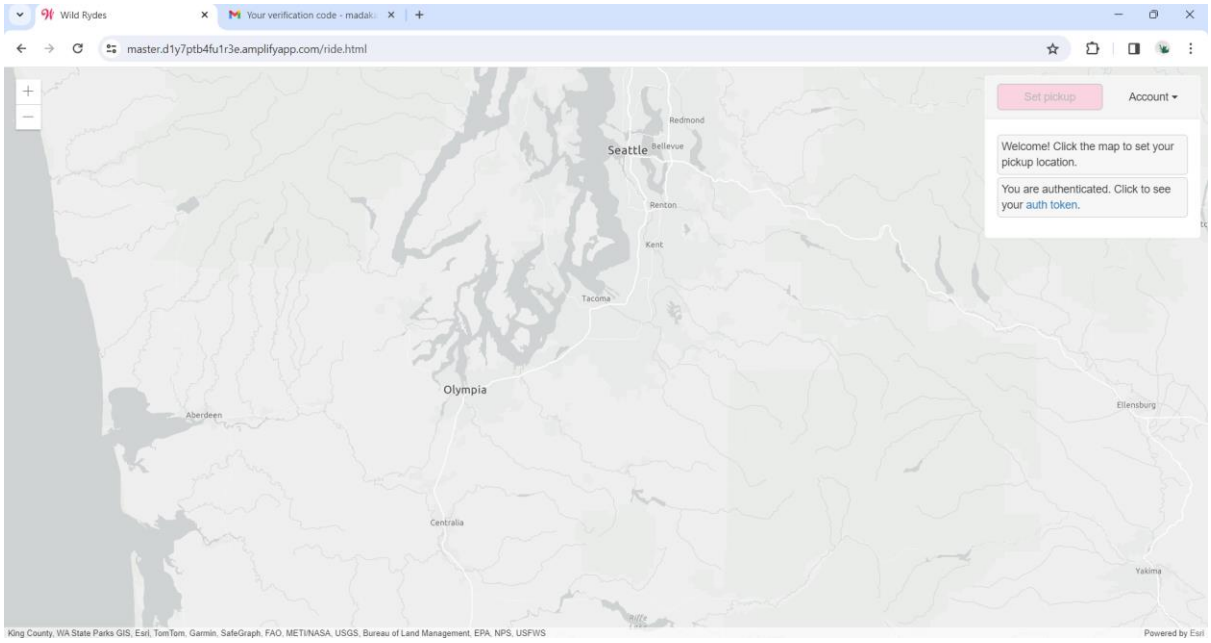
After commiting amplify deploys application once again .

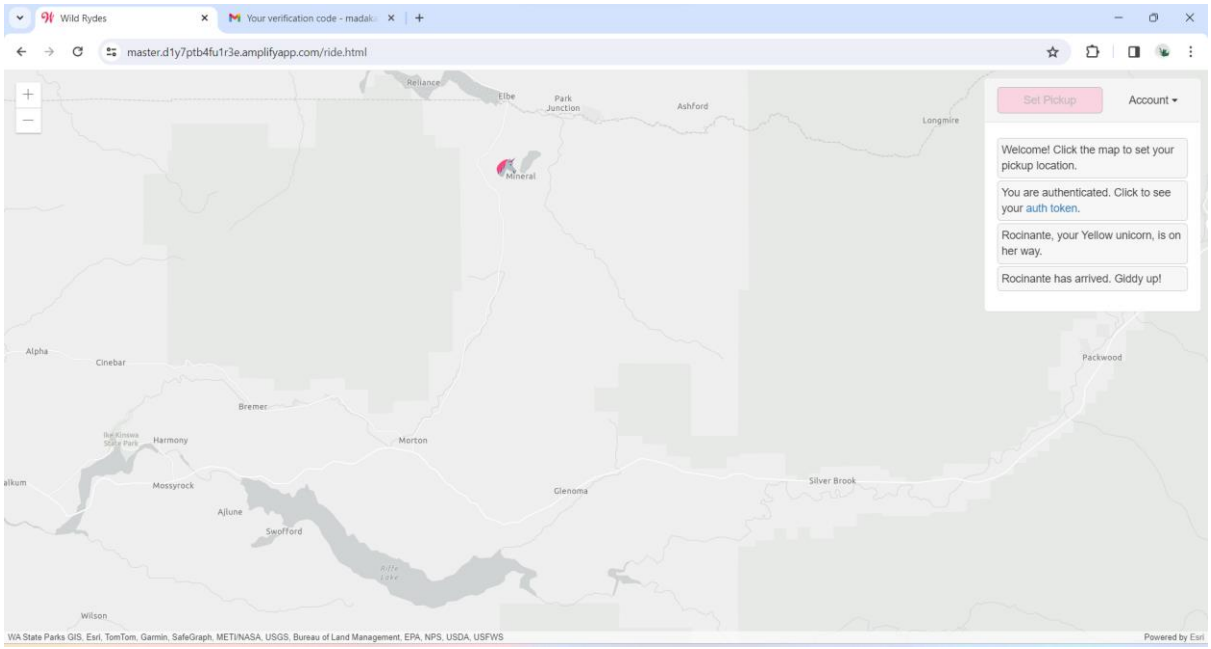




Successfully code is deployed …. Check with amplify RL . we get below desired output showing map.







STEP -5 :

Delete all resources..