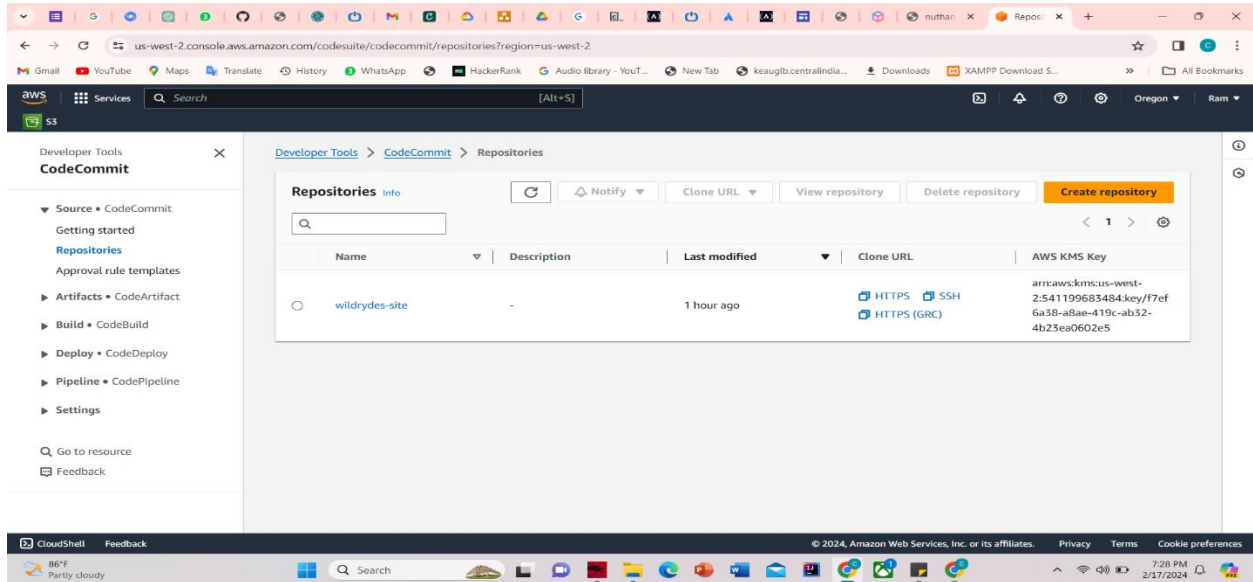


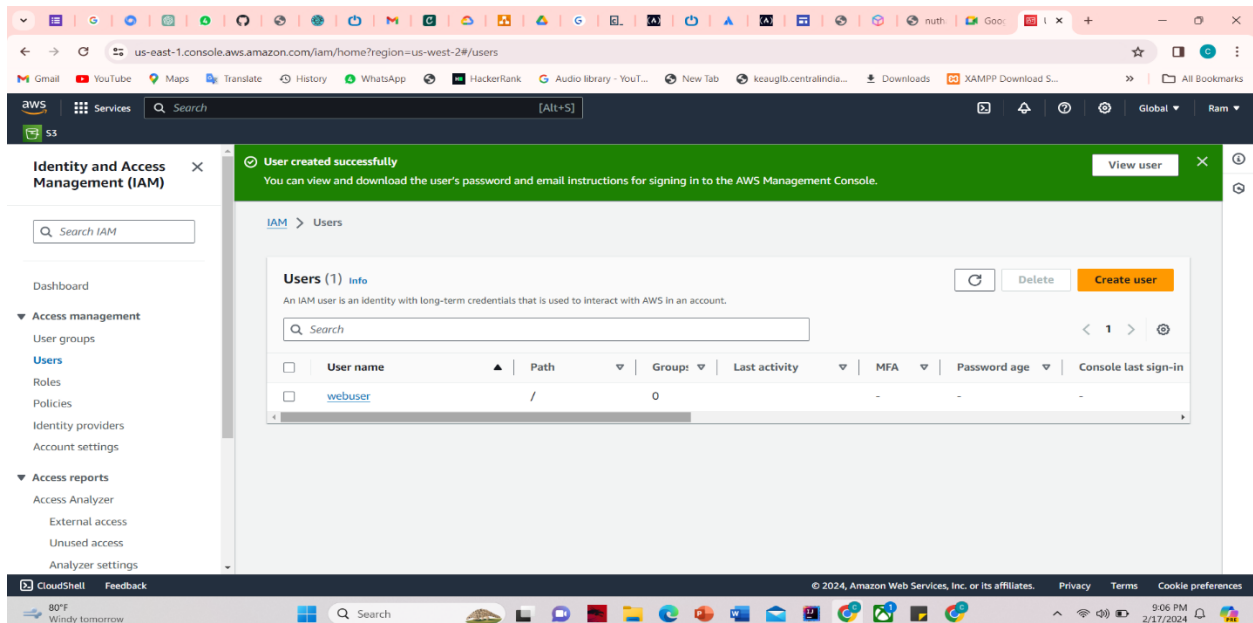
Serverless Web Application

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

1. In AWS Console (user ID “Ram”) Region (us-west-2) go to Code Commit and Create a Repo with name “wildryes-site”

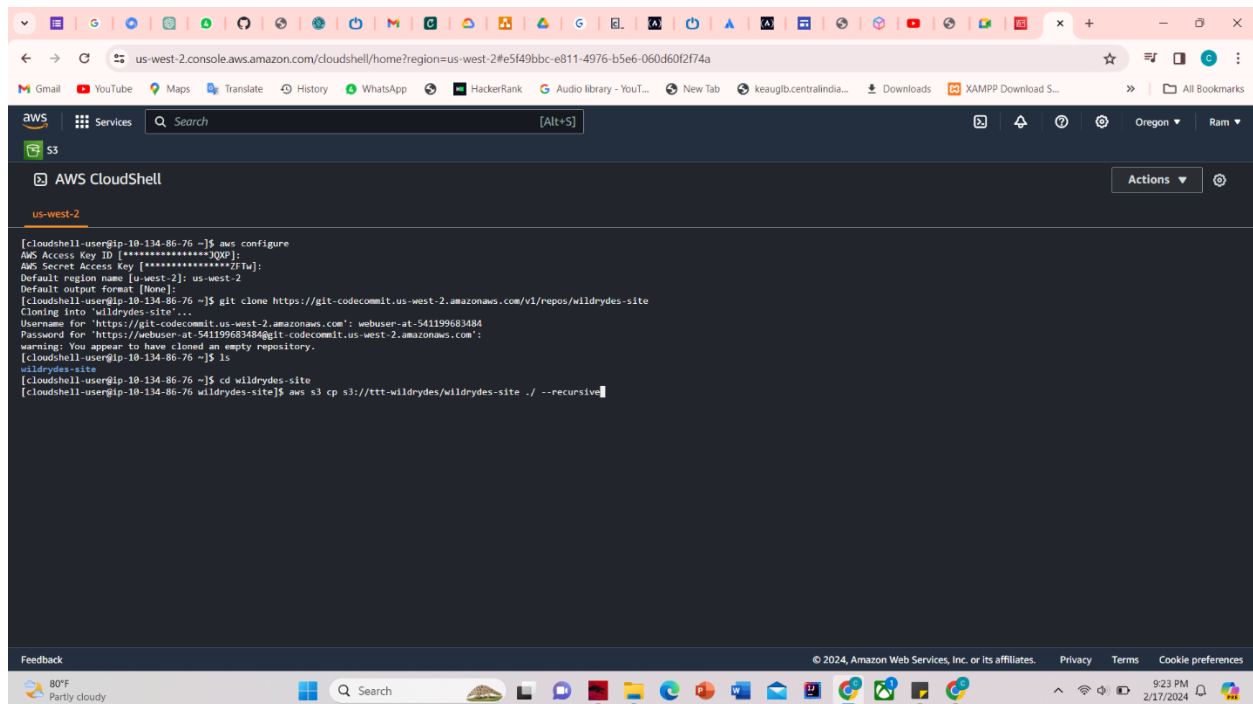


2. Create IAM User (webuser) with Access key and HTTPS Git Credentials



3. Configure Access Key in CLI with Access key With Region and clone the Repo (git clone (HTTP URL copies in Code Commit) and in Repo Add website files to repo from s3 bucket (command “aws s3 cp s3://ttt-wildrydes/wildrydes-site/ --recursive”) and commit and push it

- git add .
- git commit -m “message”
- git push



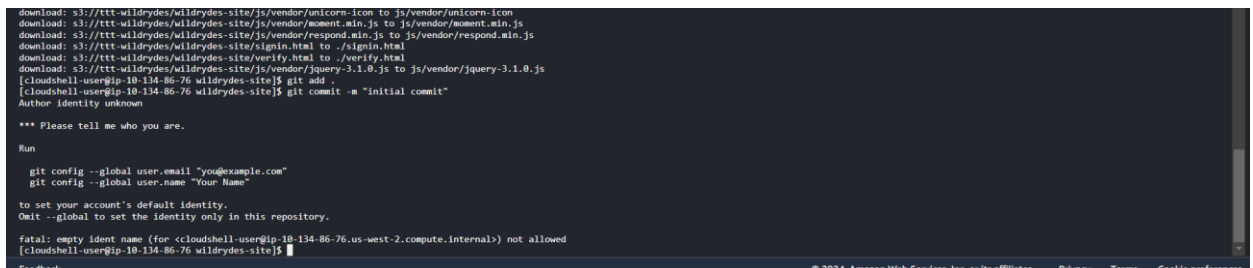
```
us-west-2.console.aws.amazon.com/cloudshell/home?region=us-west-2#e5f49bbc-e811-4976-b5e6-060d60f2f74a

AWS CloudShell

us-west-2

[cloudshell-user@ip-10-134-86-76 ~]$ aws configure
AWS Access Key ID [*****]: JQXP
AWS Secret Access Key [*****]: 2Ftu
Default region name [us-west-2]: us-west-2
Default output format [None]:

[cloudshell-user@ip-10-134-86-76 ~]$ git clone https://git-codecommit.us-west-2.amazonaws.com/v1/repos/wildrydes-site
Cloning into 'wildrydes-site'...
Username for 'https://git-codecommit.us-west-2.amazonaws.com': webuser-at-541199683484
Password for 'https://webuser-at-541199683484@git-codecommit.us-west-2.amazonaws.com':
warning: You appear to have cloned an empty repository.
[cloudshell-user@ip-10-134-86-76 ~]$ ls
wildrydes-site
[cloudshell-user@ip-10-134-86-76 ~]$ cd wildrydes-site
[cloudshell-user@ip-10-134-86-76 wildrydes-site]$ aws s3 cp s3://ttt-wildrydes/wildrydes-site ./ --recursive
```



```
download: s3://ttt-wildrydes/wildrydes-site/js/vendor/unicorn-icon to js/vendor/unicorn-icon
download: s3://ttt-wildrydes/wildrydes-site/js/vendor/moment.min.js to js/vendor/moment.min.js
download: s3://ttt-wildrydes/wildrydes-site/js/vendor/respond.min.js to js/vendor/respond.min.js
download: s3://ttt-wildrydes/wildrydes-site/signin.html to ./signin.html
download: s3://ttt-wildrydes/wildrydes-site/verify.html to ./verify.html
download: s3://ttt-wildrydes/wildrydes-site/js/vendor/jquery-3.1.0.js to js/vendor/jquery-3.1.0.js
[cloudshell-user@ip-10-134-86-76 wildrydes-site]$ git add
[cloudshell-user@ip-10-134-86-76 wildrydes-site]$ git commit -m "initial commit"
Author identity unknown

*** Please tell me who you are.

Run

git config --global user.email "you@example.com"
git config --global user.name "Your Name"

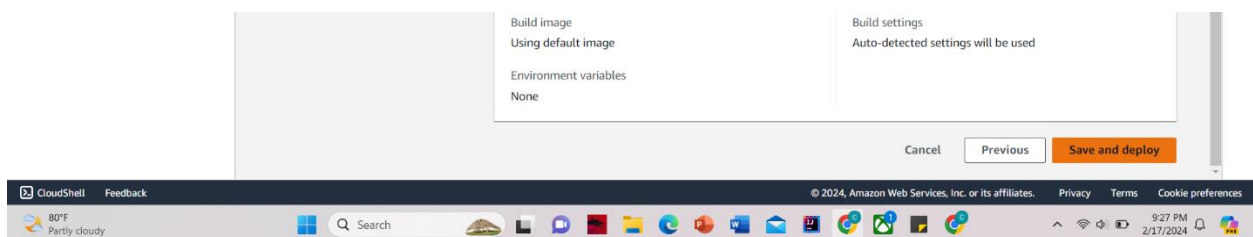
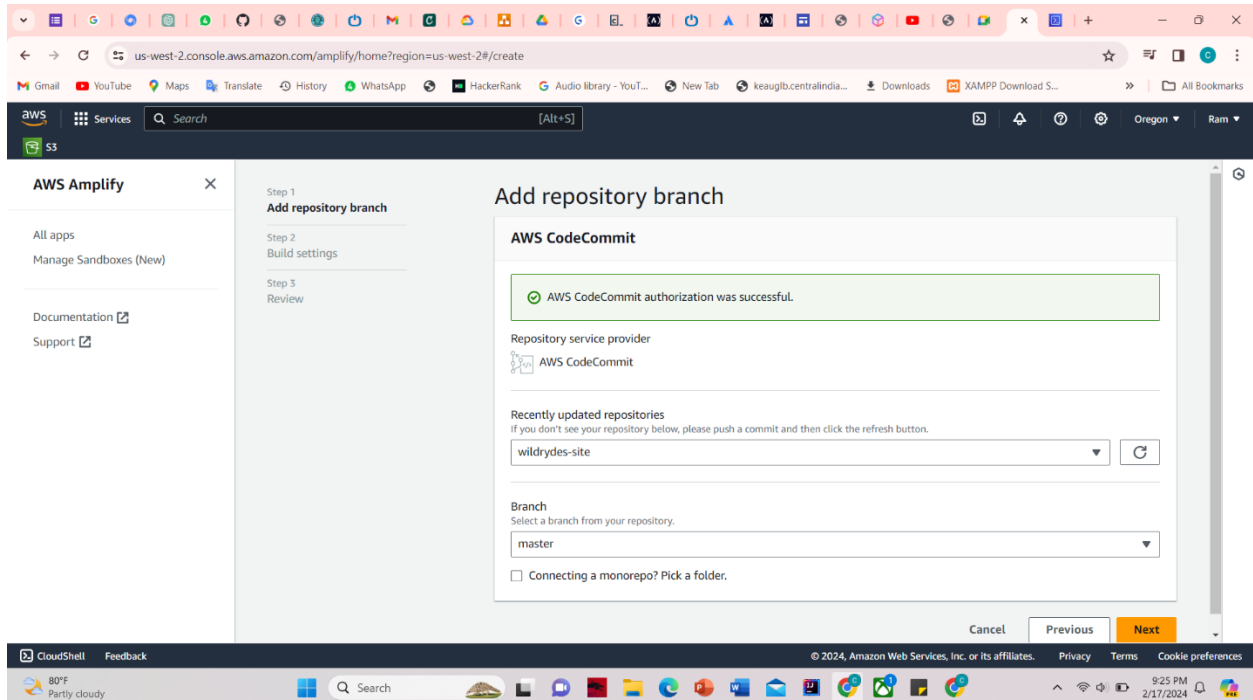
to set your account's default identity.
Omit --global to set the identity only in this repository.

fatal: empty ident name (for <cloudshell-user@ip-10-134-86-76.us-west-2.compute.internal>) not allowed
[cloudshell-user@ip-10-134-86-76 wildrydes-site]$
```

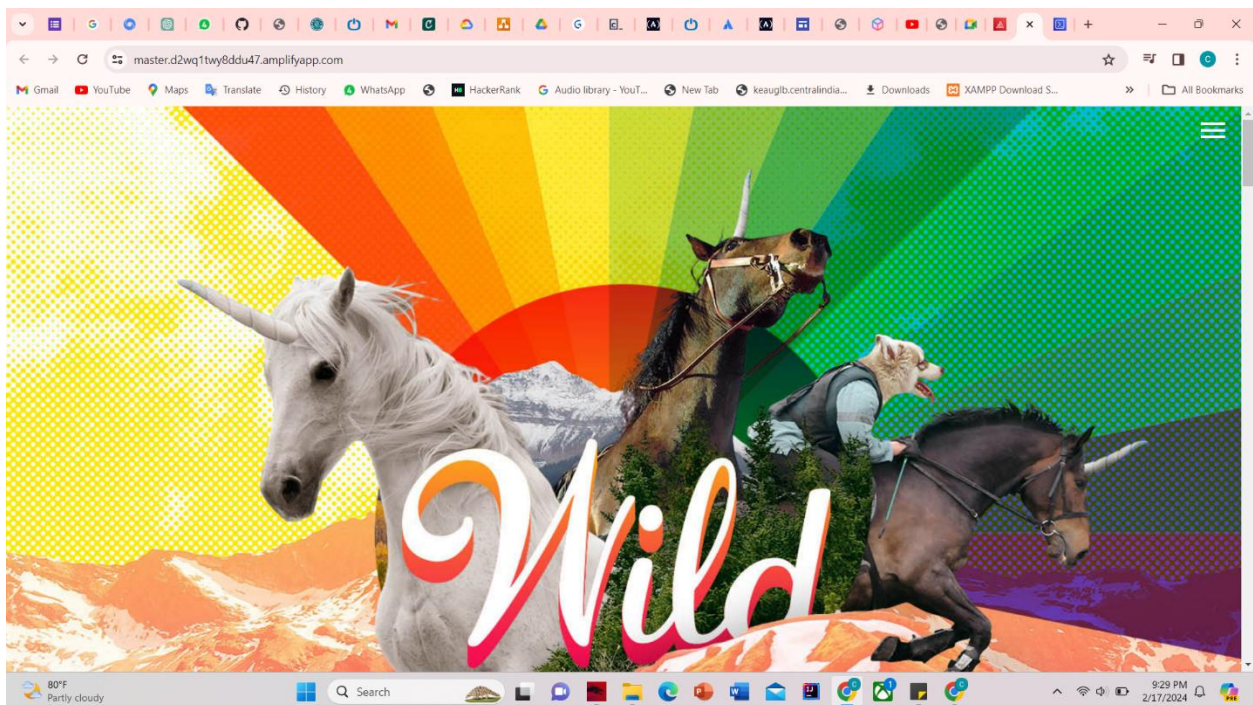
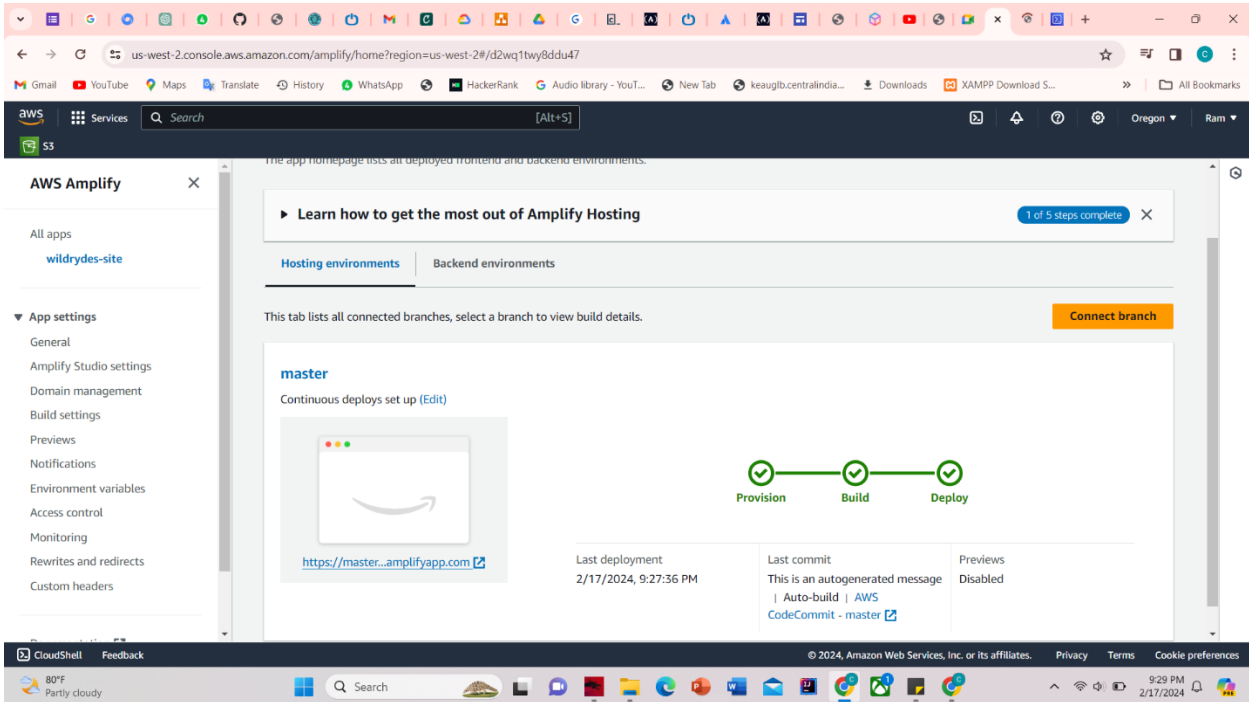
3. In Amplify Create application and deploy it:

- Choose AWS CodeCommit
- Select wildrydes-site Repo from drop down list
- In Branch select master

Click on Next and Review and Deploy

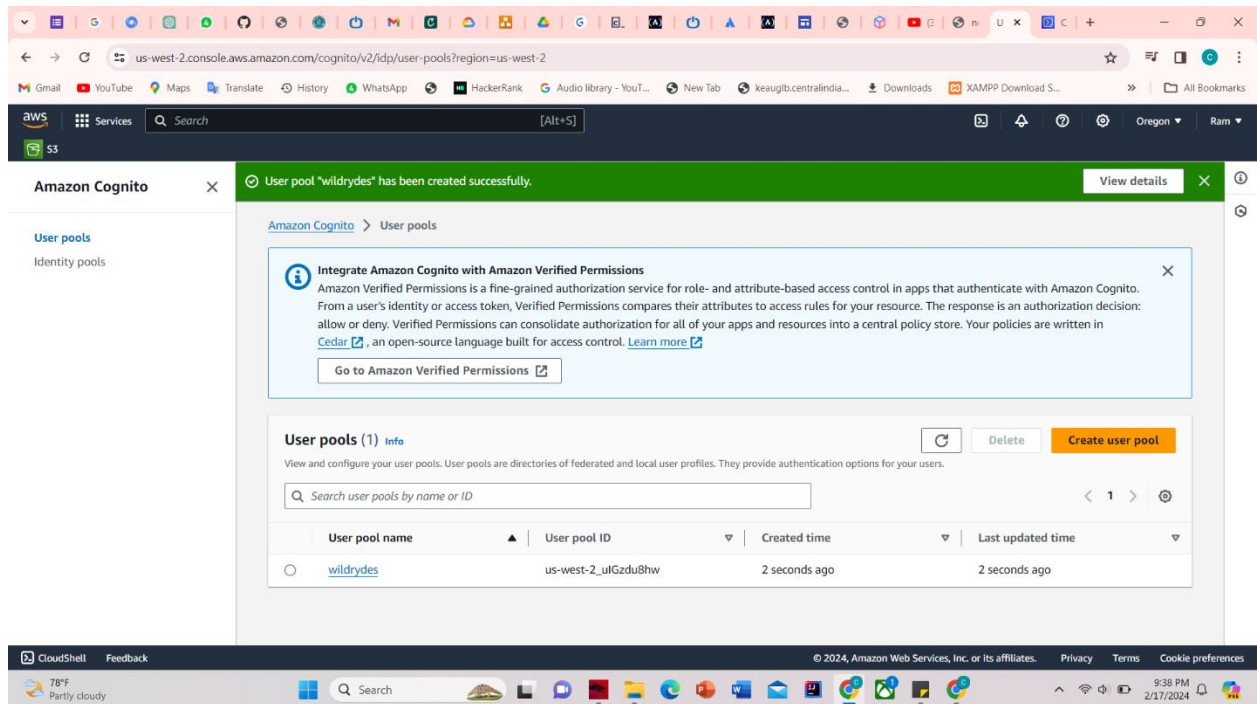


- After few minutes it will completely Deploy the code and click on the the link to see the website



4. In Console AWS Cognito create User pool with name “wildrydes”

- No MFA and Attributes
- In Configure message Delivery (Send email with Cognito)
- Got for Next and last Create user pool
- Copy User pool ID and Client ID (in App Client list)



5. Update the config.js file in CodeCommit->js->config.js

- Update the user Pool ID and Client ID
- Commit the changes by entering Author name and Email Address



- Then the code will redeploy and then validate the implementation

WildRYDES

REGISTER

ravanram267@gmail.com

.....

.....

LET'S RYDE

- Enter the details and click on let's ride you receive an OTP validate it

master.d2wq1twy8ddu47.amplifyapp.com says
Verification successful. You will now be redirected to the login page.

OK

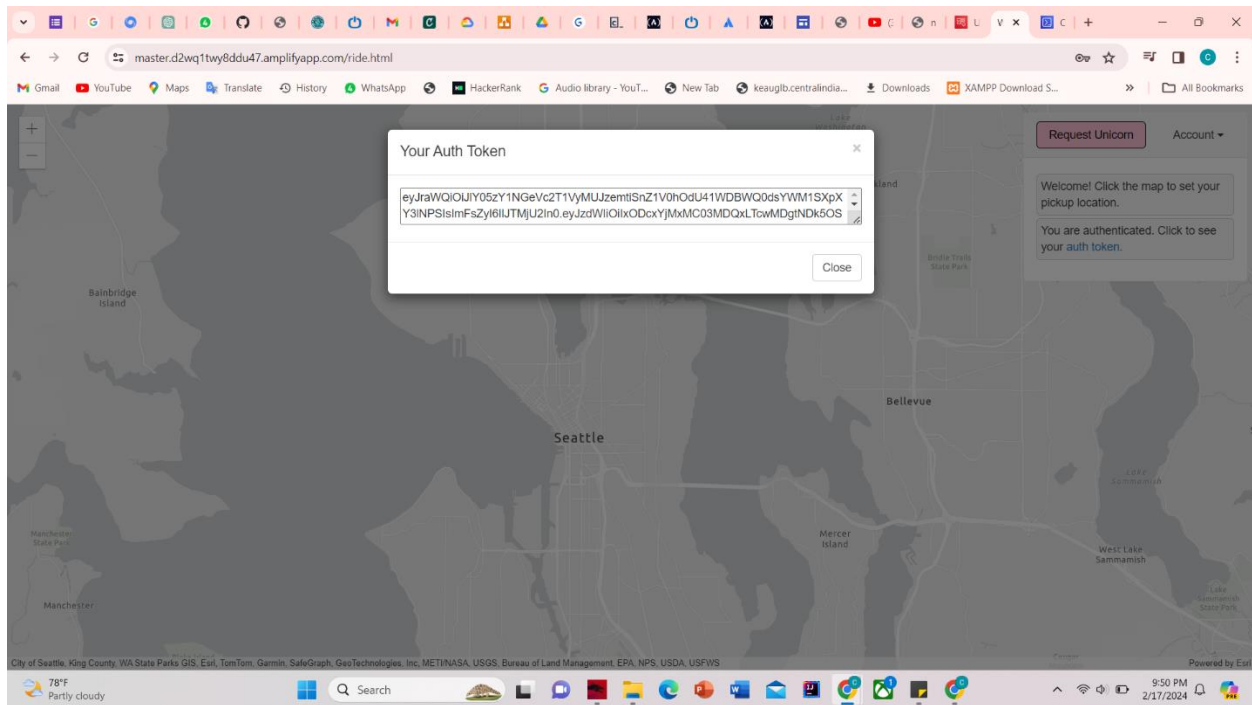
VERIFY EMAIL

ravanram267@gmail.com

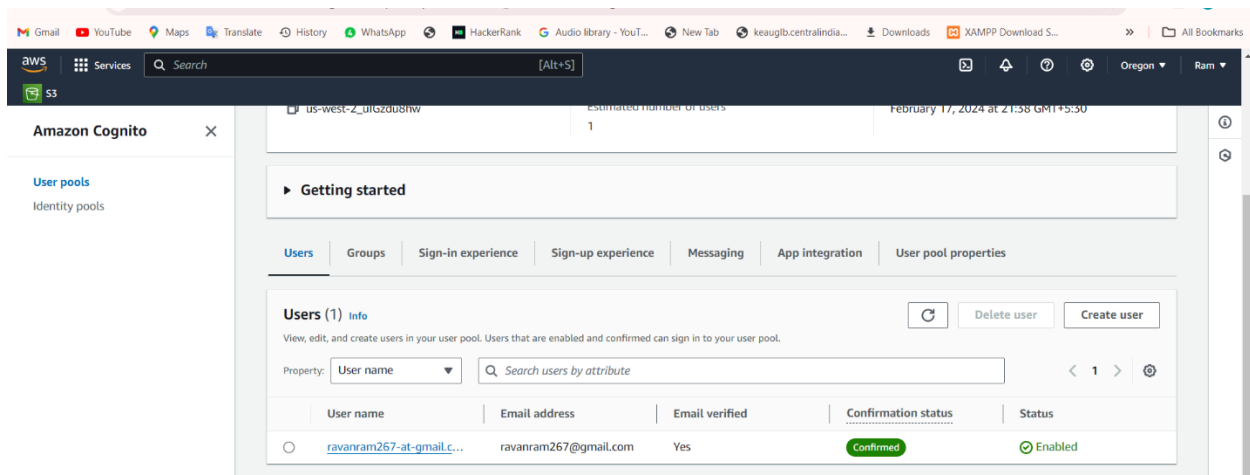
762127

VERIFY

➤ Now we can login since it is successfully registered

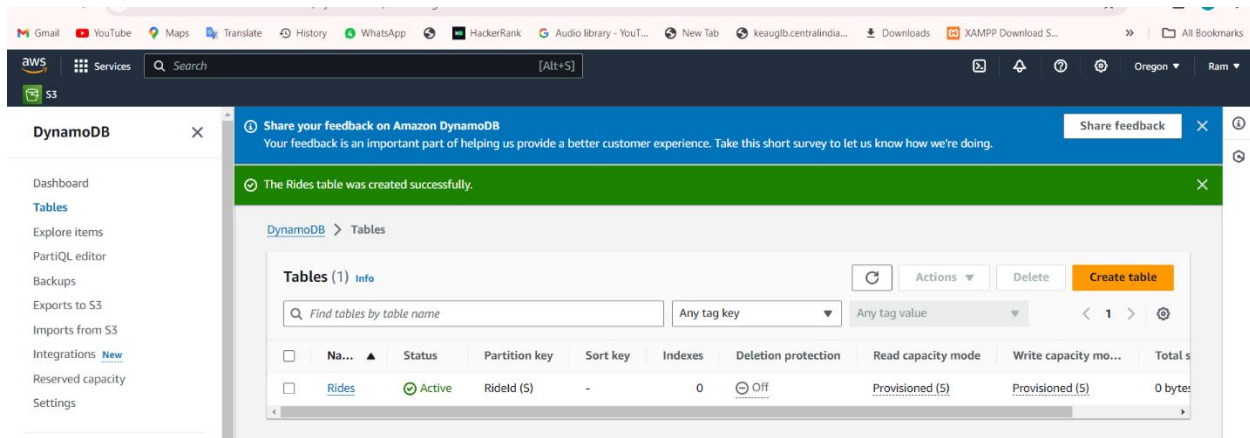


➤ you can also see the users in user pool in cognito



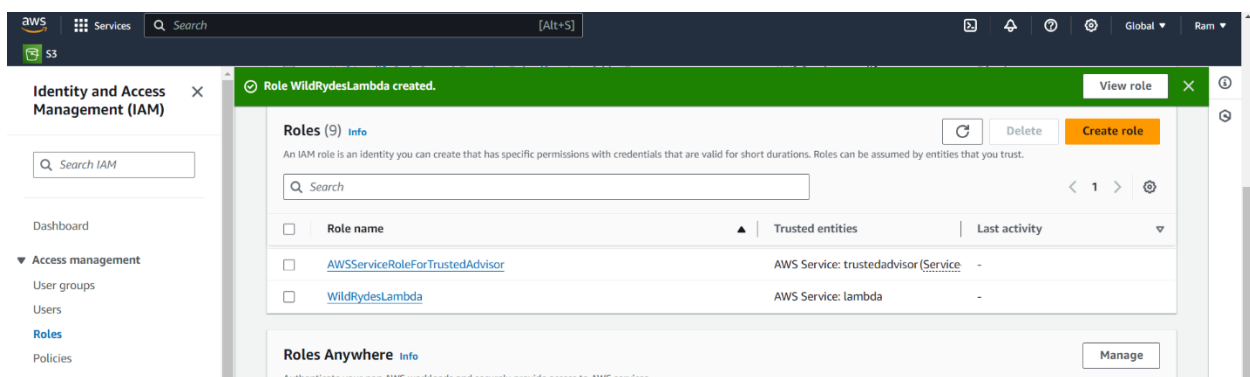
6. Now we create Data base (Dynamo DB) and lambda for backend

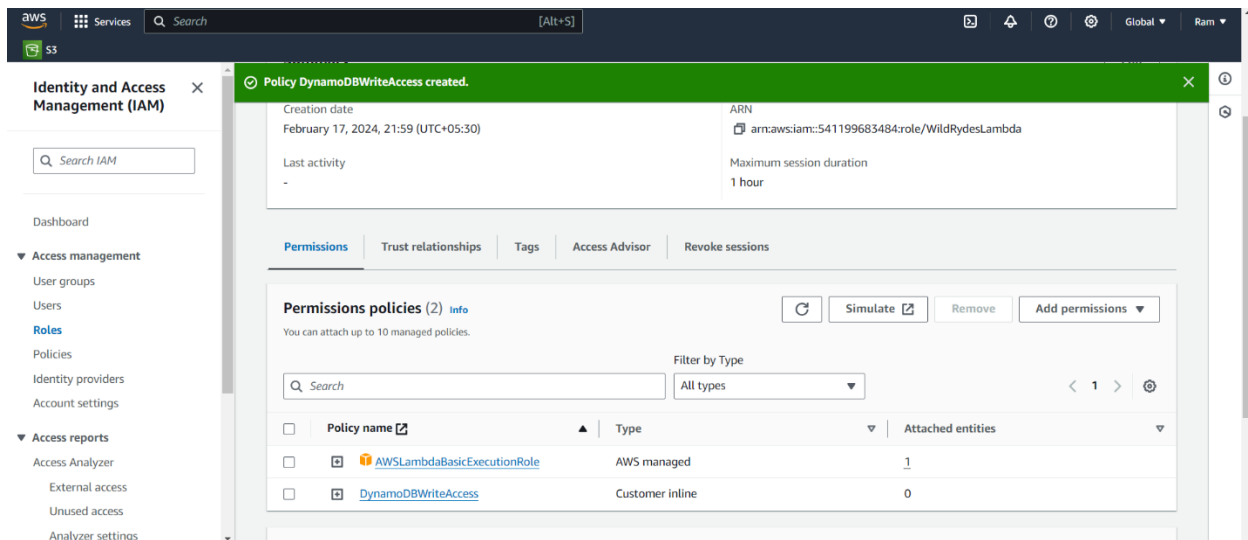
- Create a Table with name Rides in Dynamo DB and copy the ARN (Overview -> General information -> additional information)



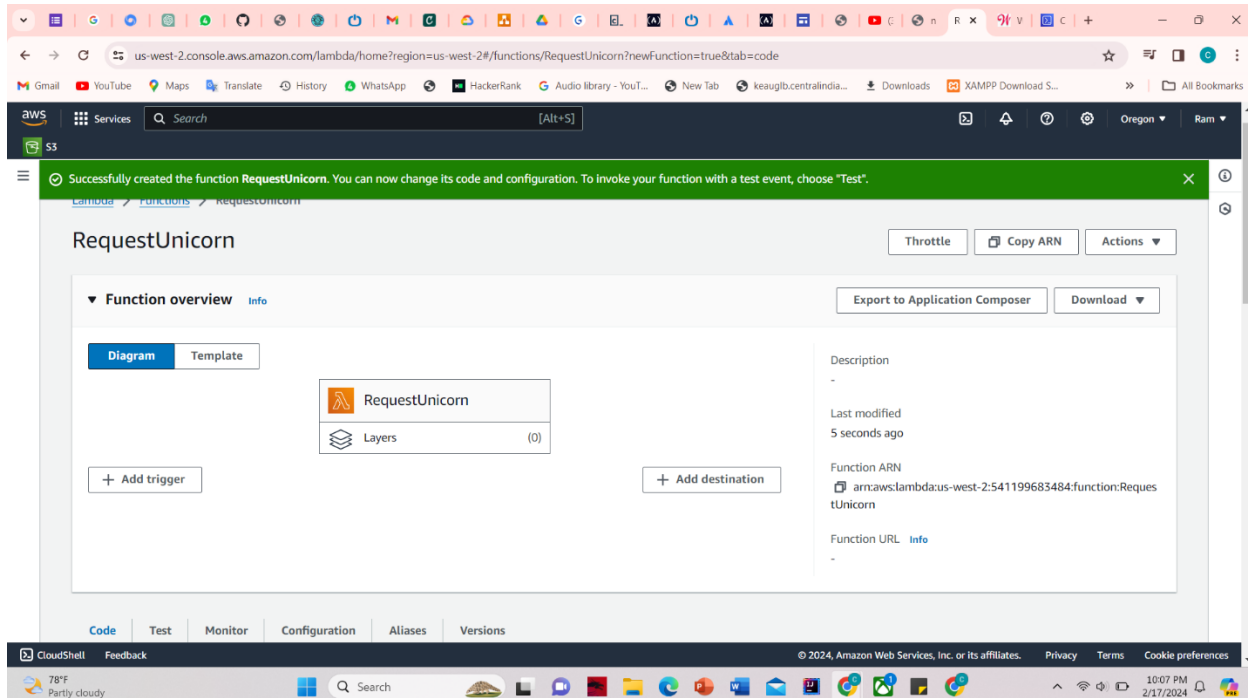
7. Create IAM role (WildRydesLambda) for lambda function

- Attach AWSLambdaBasicExecutionRole and create an inline Policy select service Dynamo DB and Action allowed putitem and select Add ARN link in Test add the ARN link of table and give a name (DynamoDBWriteAccess) for policy and create policy





- Create lambda function with name RequestUnicorn
- Select Node.js 16.x for runtime
- Select existing role (WildRydeslambda)
- Create the function



- Down in code source replace node.js with this code (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(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': '*',
    },
  });
}
```

- Choose Deploy
- Choose Test with event name TestrequestEvent and copy and paste this code in Event JSON

TEST EVENT

```
{
  "path": "/ride",
  "httpMethod": "POST",
  "headers": {
    "Accept": "*/*",
    "Authorization": "eyJraWQjOiJLTzRVMWZs",
    "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}}

```

➤ Save it and choose Test (result (Succeeded))

The screenshot displays the AWS Lambda console interface. At the top, a green notification bar states: "The test event TestRequestEvent was successfully saved." Below this, the "Code source" tab is active, showing the "index.js" file. The "Test" button is highlighted. The "Execution results" tab is selected, showing the test event "TestRequestEvent" with a status of "Succeeded". The response is displayed as a JSON object:

```

{
  "statusCode": 201,
  "body": "{\"PickupLocation\":{\"Latitude\":47.6174755835663,\"Longitude\":-122.28837066650185}}",
  "headers": {
    "Access-Control-Allow-Origin": "*"
  }
}

```

The function logs show the request event and the unicorn finding process:

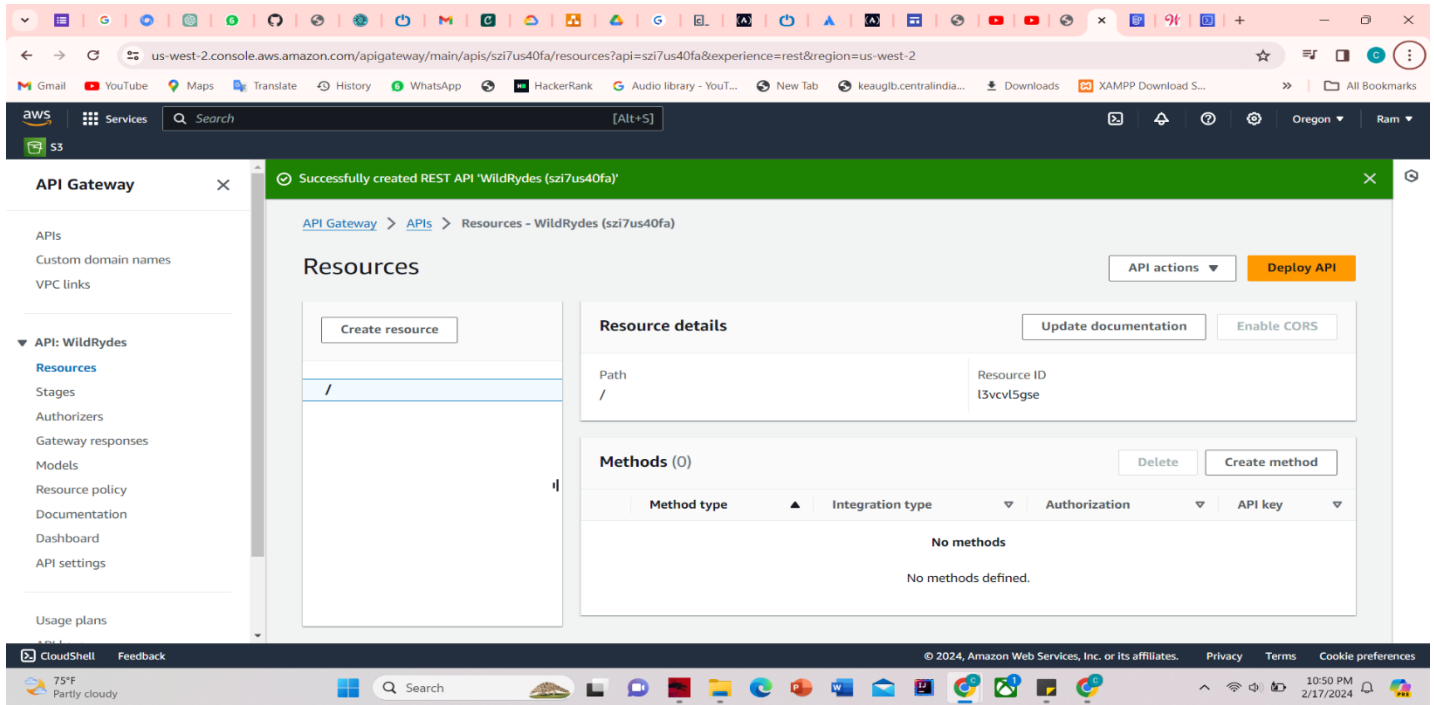
```

START RequestId: 63d58771-34b6-452c-beb1-23b30812f985 Version: $LATEST
2024-02-17T17:18:07.423Z 63d58771-34b6-452c-beb1-23b30812f985 INFO Received event { 1dJ-L1mtXXbFyzPoD_C-2g ): {
  path: "/ride",
  httpMethod: "POST",
  headers: {
    Accept: "*/*",
    Authorization: "eyJraWQ1O13LT:RVWZs",
    "content-type": "application/json; charset=UTF-8"
  },
  queryStringParameters: null,
  pathParameters: null,
  requestContext: { authorizer: { claims: [Object] } },
  body: '{"PickupLocation":{"Latitude":47.6174755835663,"Longitude":-122.28837066650185}}'
}
2024-02-17T17:18:07.482Z 63d58771-34b6-452c-beb1-23b30812f985 INFO Finding unicorn for 47.6174755835663 , -122.28837066650185
END RequestId: 63d58771-34b6-452c-beb1-23b30812f985

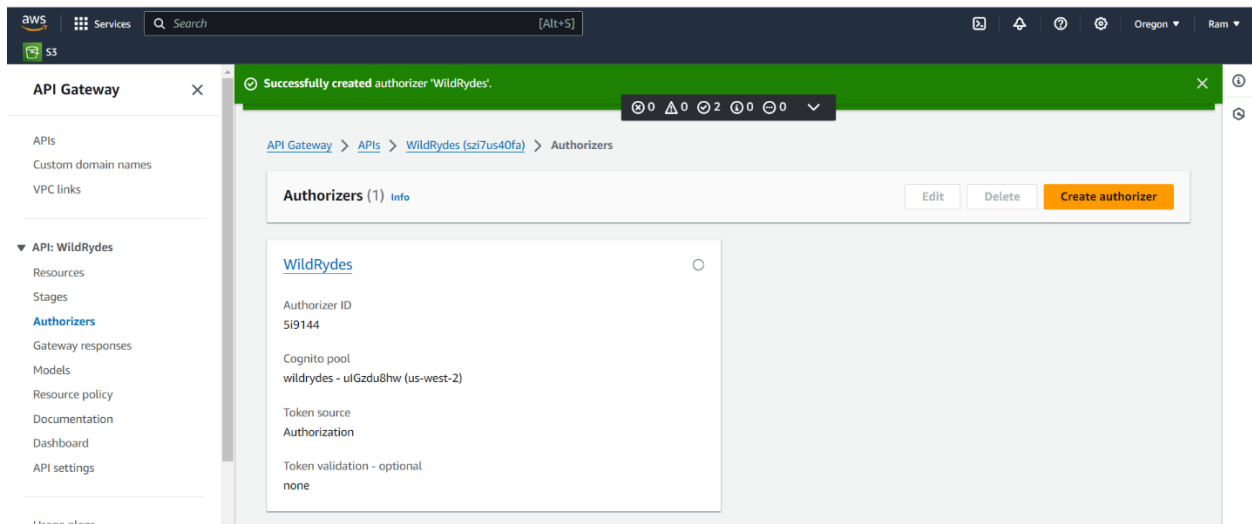
```


8. Deploy Restful API

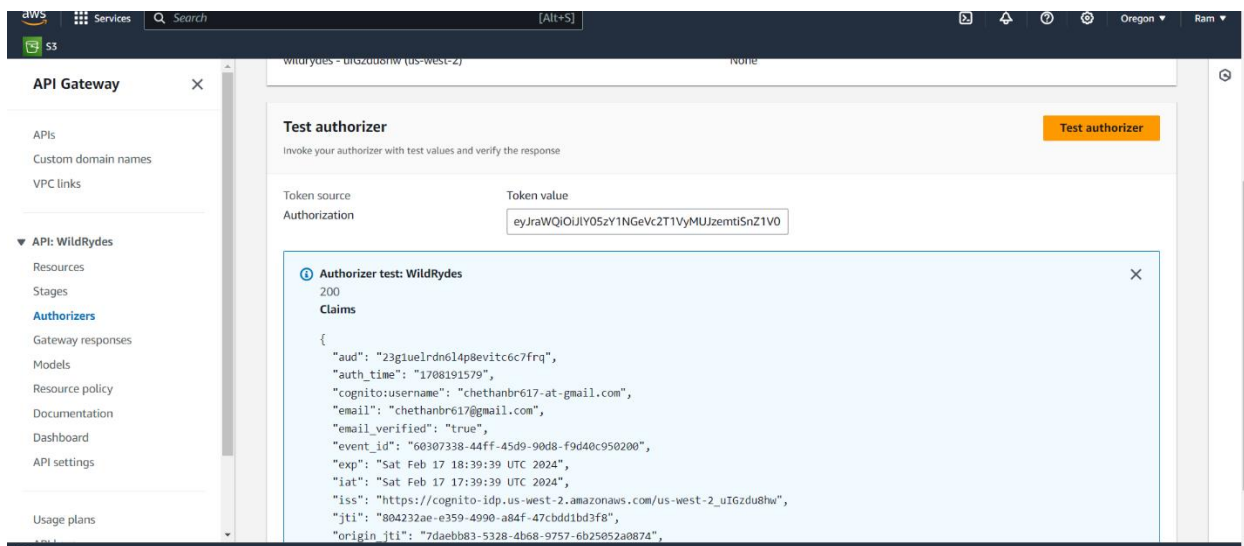
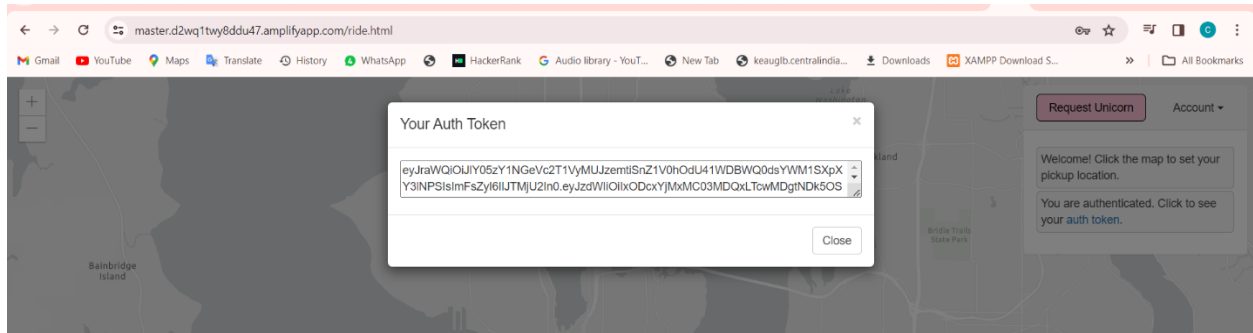
- In AWS API gateway create a Rest API



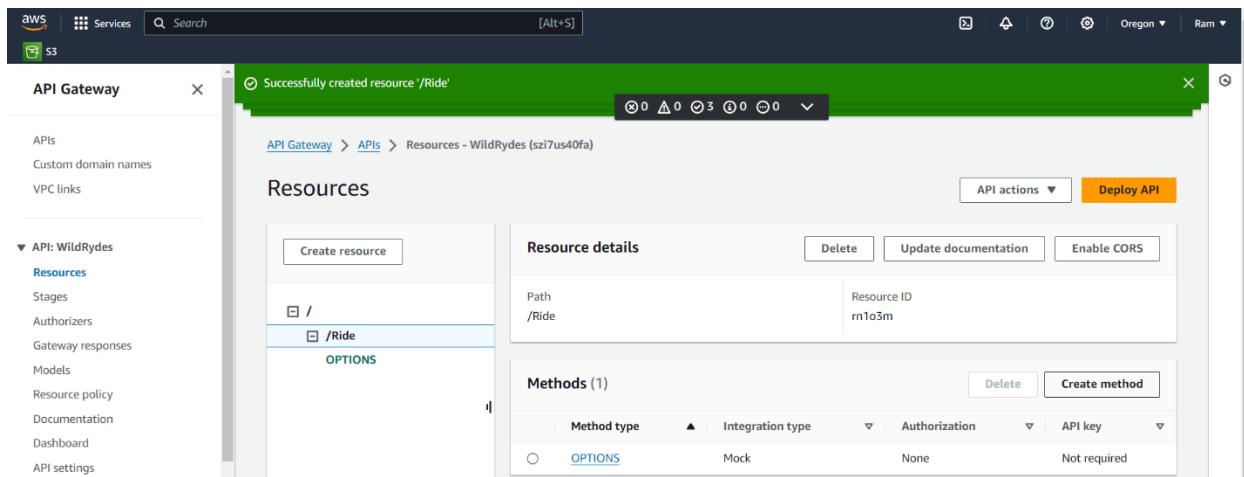
- Create a Authorizer under Rest API because API uses JSON Web tokens which are returned by Cognito User Pool for Authenticate API calls
- Authorizer name WildRydes and Cognito as type and Authorization for Token Source and create it



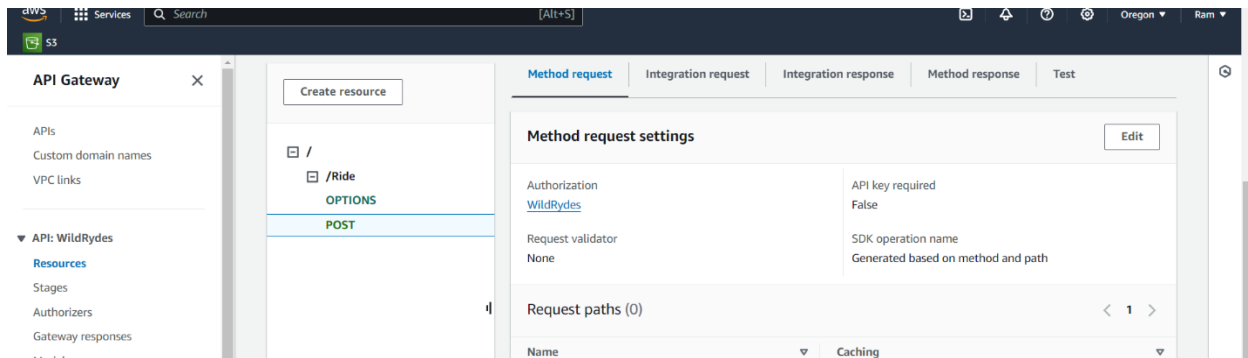
- To verify Authorizer configuration copy and paste the Authorization Token copied from web page and respond code 200



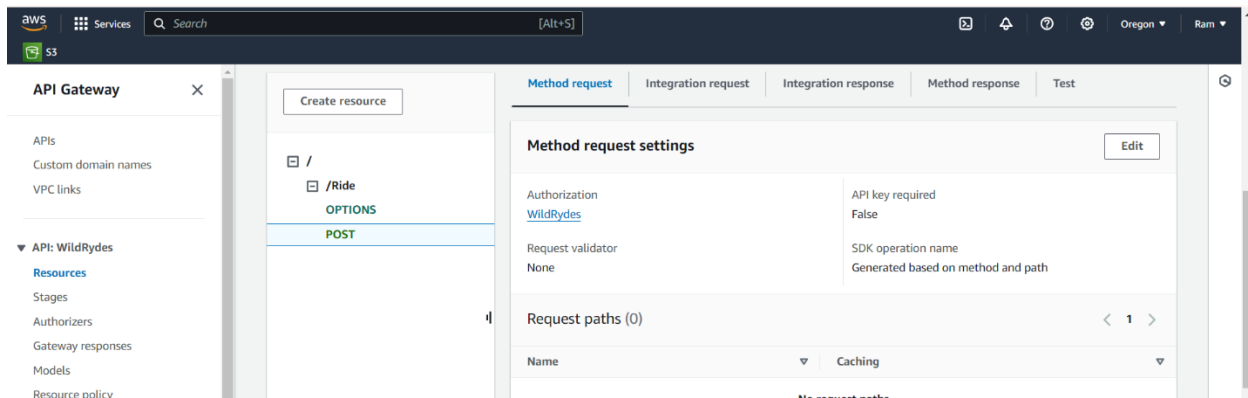
- now create Resource with name ride and Enable API Gateway CORS



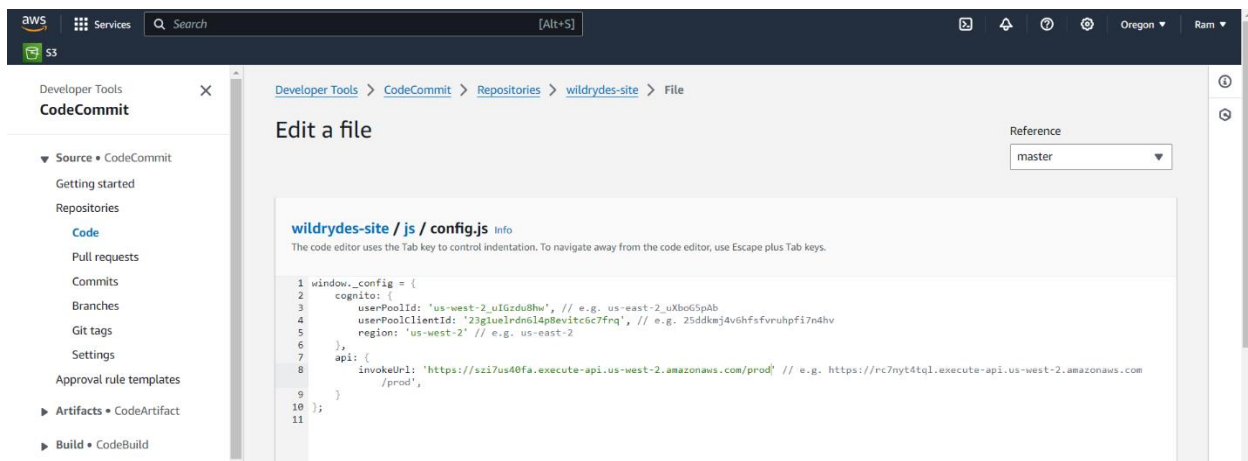
- Create POST Method with ride resource selected
- Select same region and lambda function



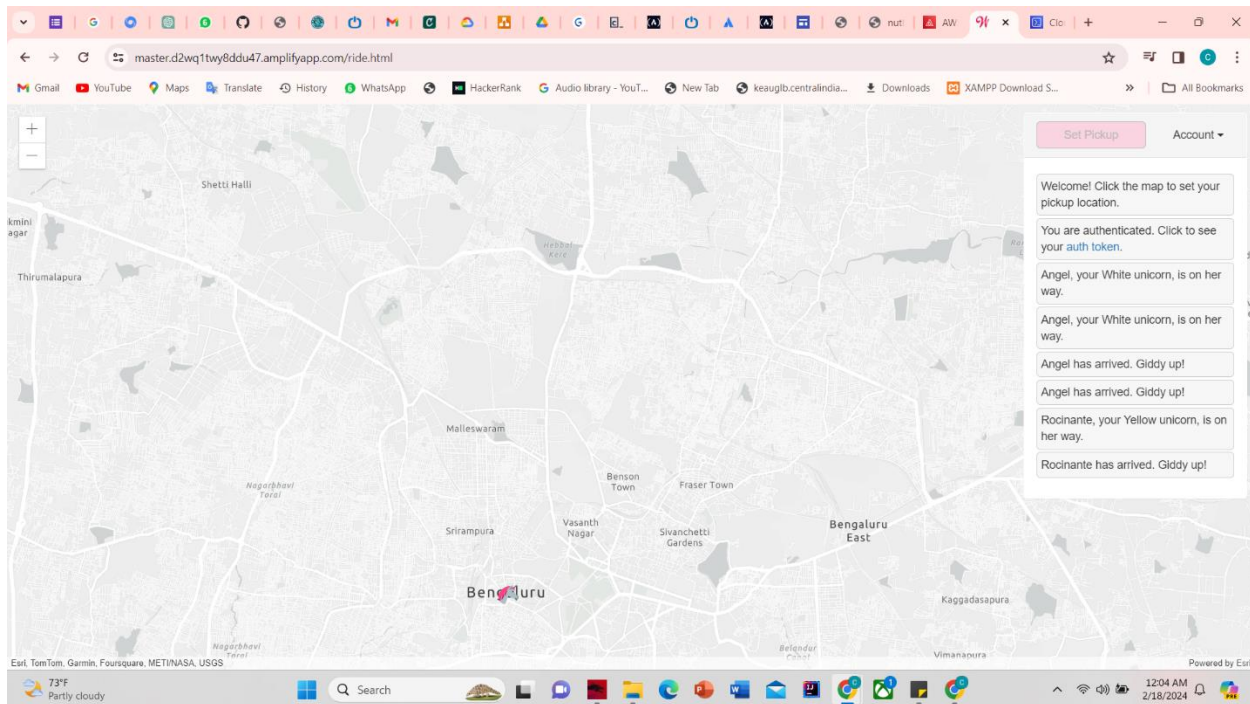
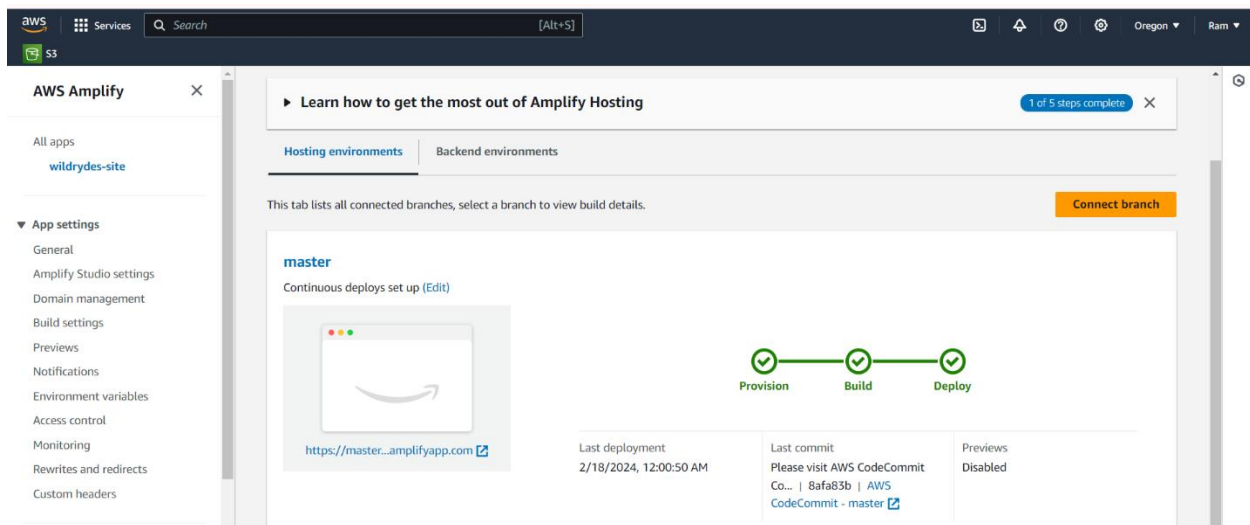
- Click on method request edit select WildRydes Cognito User Pool Authorizer we created earlier
- Deploy API and copy invoke URL



- update the config.js file with URL (Codecommit -> repo->js->config.js)



- After commit the Amplify redeploys application and check the URL
- We get output mark on map and click on request unicorn



Hence, we have Created and Deployed a Serverless Web application Successfully

At the End Cleanup The Resource