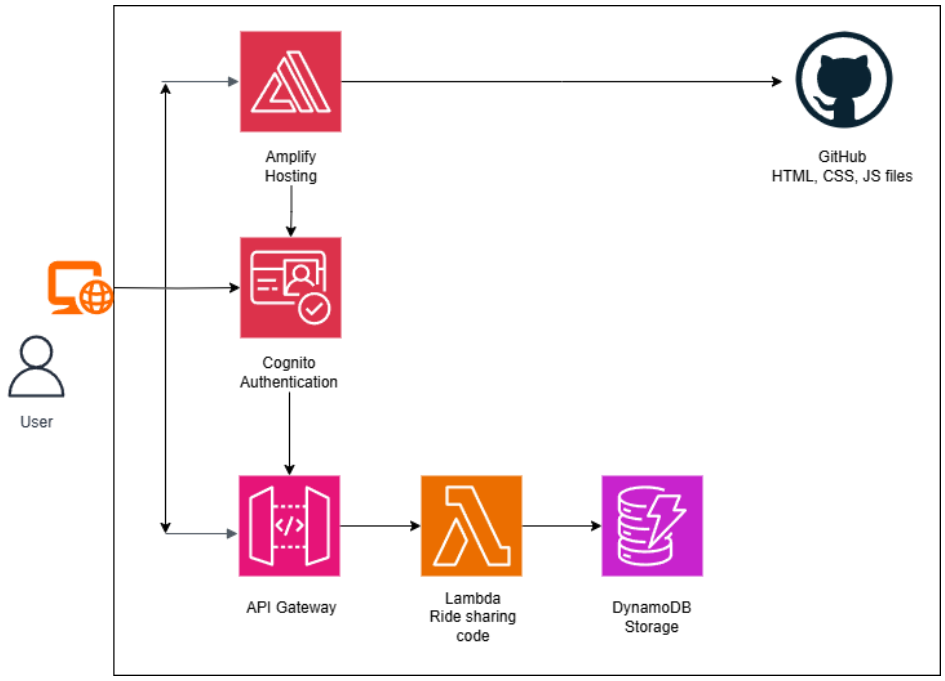
AWS Project: End-to-End AWS Web Application

Architecture Diagram



Project Description and Resources Used

For this project I'll be building an end-to-end application where I can request a unicorn to give me a ride (from the original Amazon Workshop). The website will take me through the process of creating an account, logging in, and accessing the map (powered by ArcGIS) to ask for the ride.

Link to the tutorial followed: Tiny Technical Tutorials

AWS Services and Integration

GitHub - Used as a source control system to create the wildryde-sites repo where the code will live.

AWS Amplify - Amplify will allow me to build and host the static website. It will be hooked up to the wildrydes-sites GitHub repo for retrieving the code, and help trigger the CICD pipeline automatically.

Amazon Cognito - Used for authentication and registration. The user pool will be set up manually and then updated on the code to hook things up.

AWS Lambda - Used to create/trigger a function when the end user requests a ride.
Amazon DynamoDB - After the Lambda function is invoked, the function will select a unicorn for them and record the request in the DynamoDB table and respond to the front end with details about the unicorn that will come and pick them up.
AWS IAM - Used to create an execution role to grant Lambda write permissions to DynamoDB.
API Gateway - Used to build HTTP, REST and WebSocket APIs. It will be in charge of invoking the Lambda function.

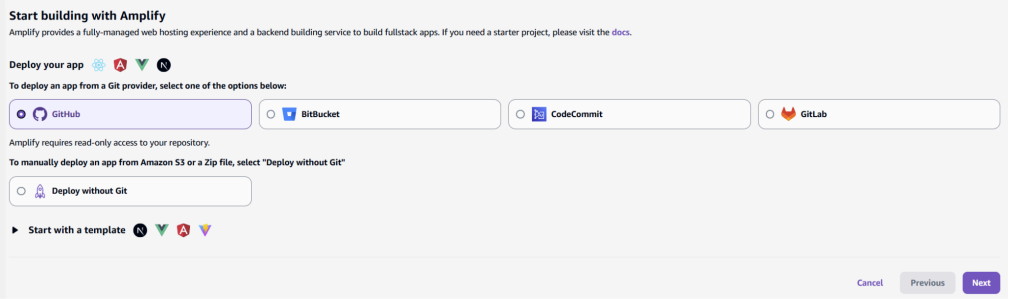
WildRydes Website

- Link to the static website: https://main.d2qzfmq4oht1md.amplifyapp.com/
- Link to the sign-in page: https://main.d2qzfmq4oht1md.amplifyapp.com/signin.html
- Link to the rides page: https://main.d2qzfmq4oht1md.amplifyapp.com/ride.html

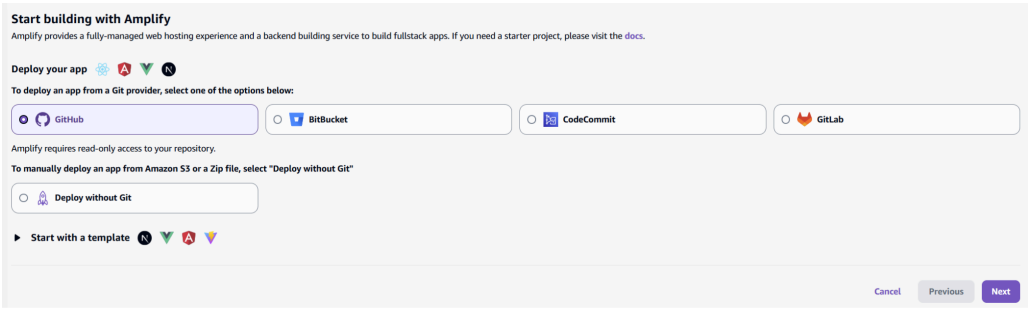
Step-by-Step Guide

AWS Amplify

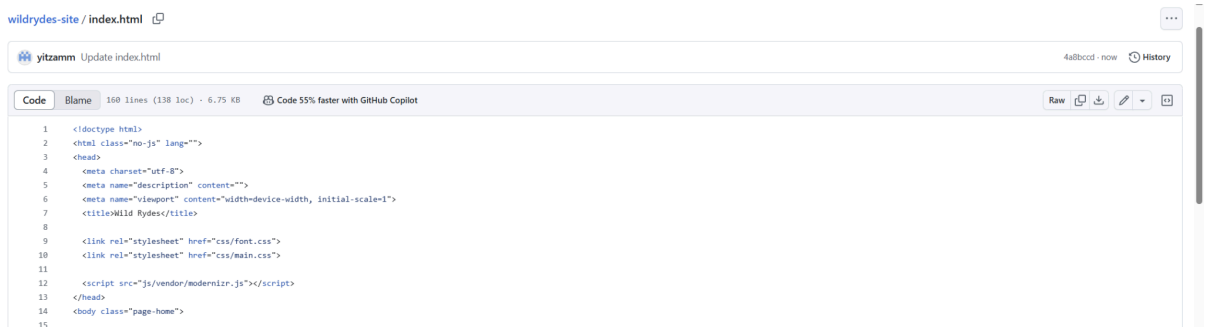
Navigate to AWS Amplify in the console. Select GitHub to deploy the app from a Git provider and grant the required Access permissions.

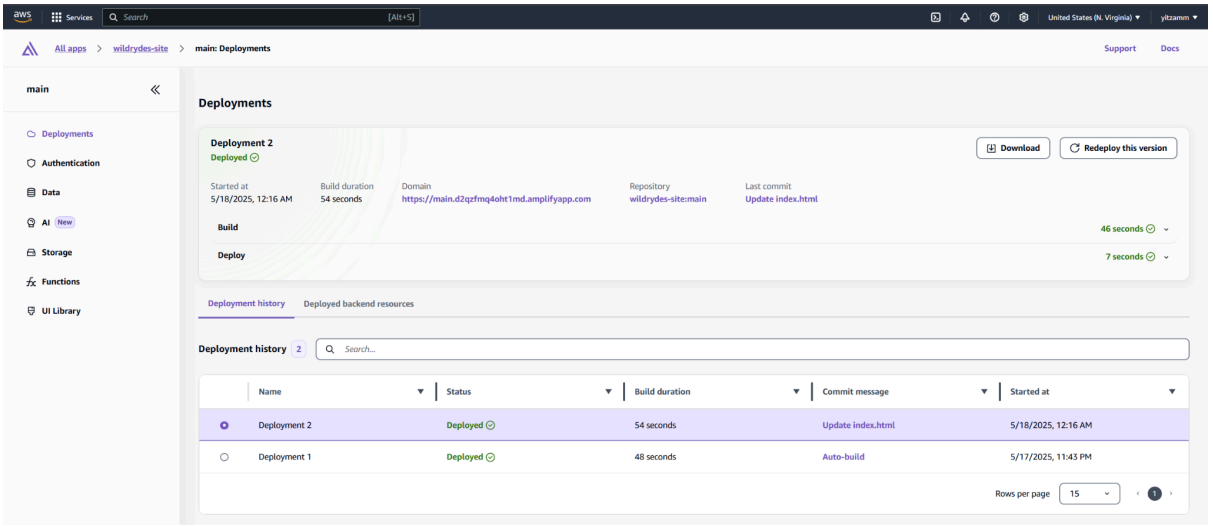


Choose the repo you'll be working with from the drop-down menu. Feel free to go back to the GitHub permissions section if your repo doesn't show up. Confirm the changes and deploy.



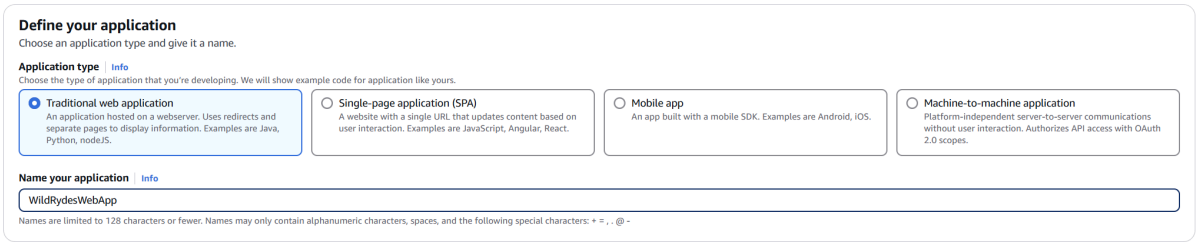
Wait for the deployment to finish running. Back to Github, find the index.html file, click on edit and modify any parts of the text, then commit the changes. Amplify will redeploy the site to reflect the changes made.

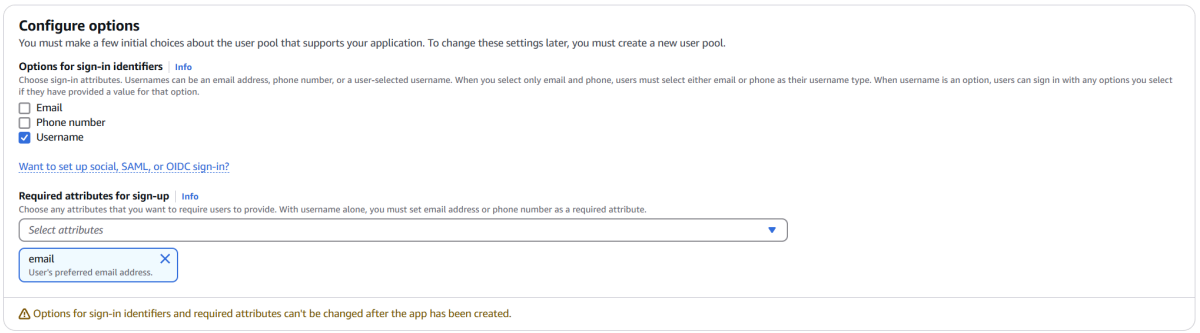


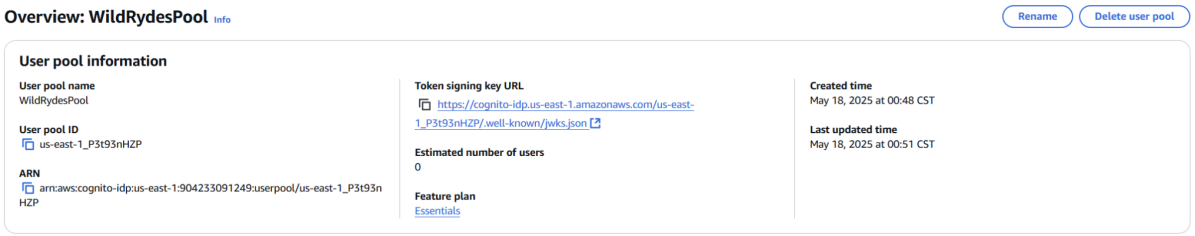


AWS Cognito

Navigate to AWS Cognito to create a new user pool. Start by defining your application and configuration options. For the application name I've picked **WildRydesWebApp** and for the user pool **WildRydesPool**.





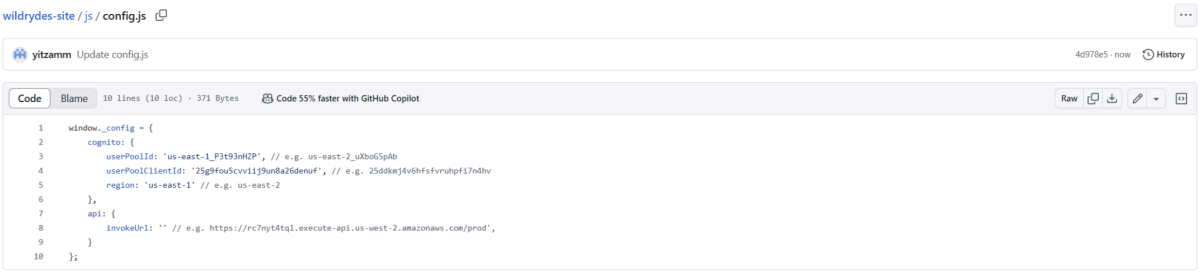


Gather the *User Pool ID* and then the *Client ID* from the Apps Clients tab. Back into Github look for the [config.js]() file inside the js folder, click on edit and enter both IDs along with the working region. Commit the changes once done.

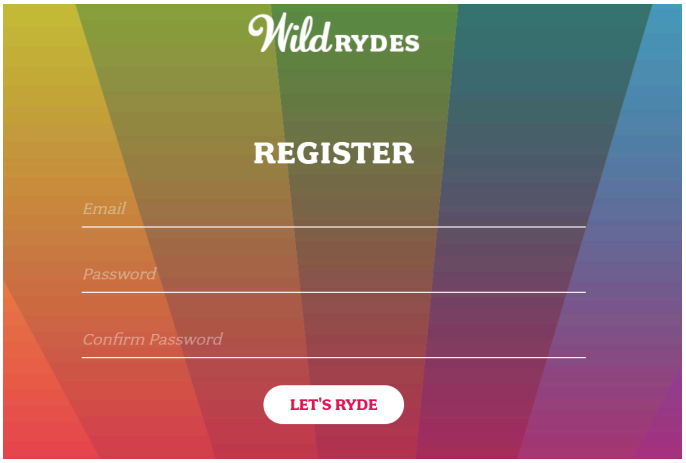
User Pool ID: us-east-1_P3t93nHZP

Client ID: 2gf8pj5rvrr531k2hiir5niplj

Region: us-east-1

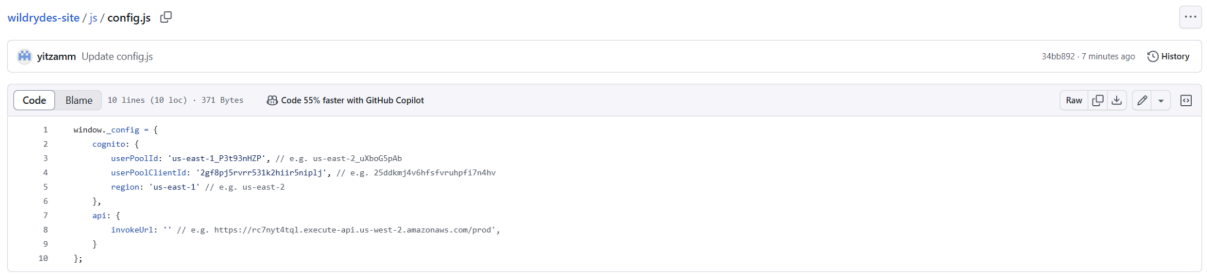


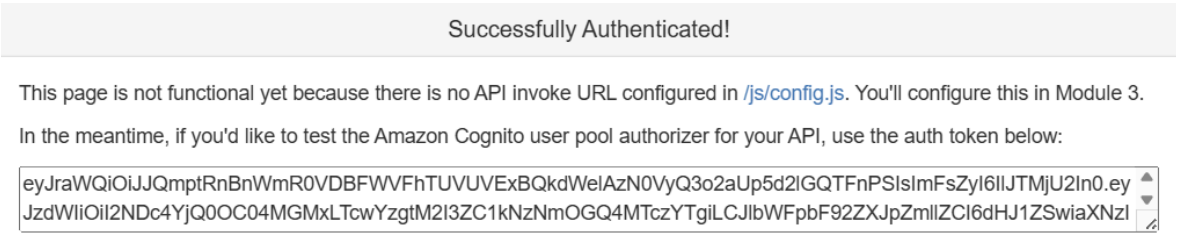
It's time to register to WydesRide, enter a valid email address for this step, so you can receive the verification code.



NotAuthorizedException ERROR

Turns out I got a ***NotAuthorizedException:*** *Client 25g9fou5cvviij9un8a26denuf is configured with secret but SECRET_HASH was not received* error when I attempted to complete the registration. The traditional web application type created a private client app which cannot be edited to disable the secret hash. To correct it, I went back to create a second client app named WildRydes, but this time I made it public using the Single-page application type. Then I updated the GitHub config file with the new client ID. This time the authentication was successful and I was able to log in.

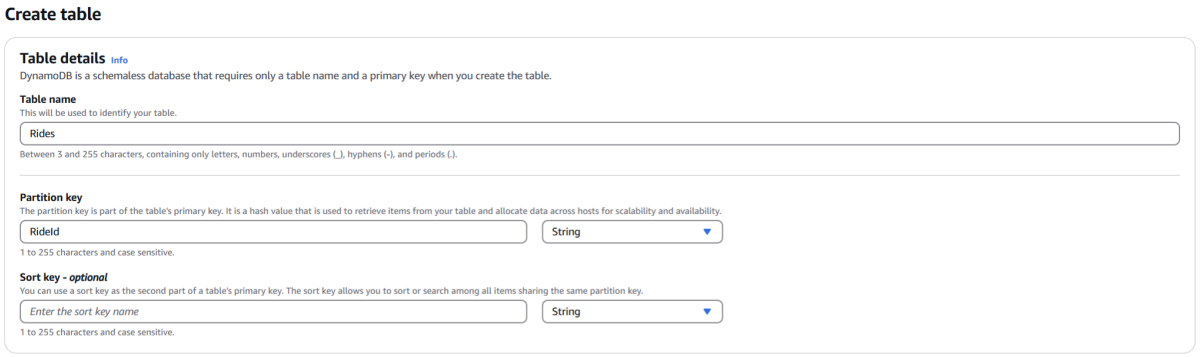




Copy the token on screen for the upcoming step.

AWS DynamoDB

Navigate to DynamoDB to create a new table. I will define the name of the table as **Rides** and the partition key as **RideId**. The rest of the settings can remain as default.

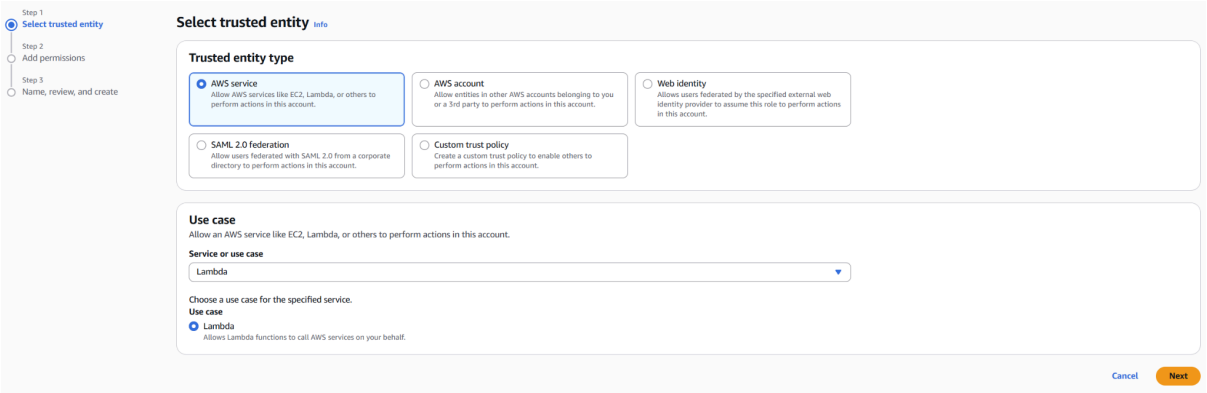


Gather the ARN for the table before exiting out. **ARN:** arn:aws:dynamodb:us-east-1:904233091249:table/Rides

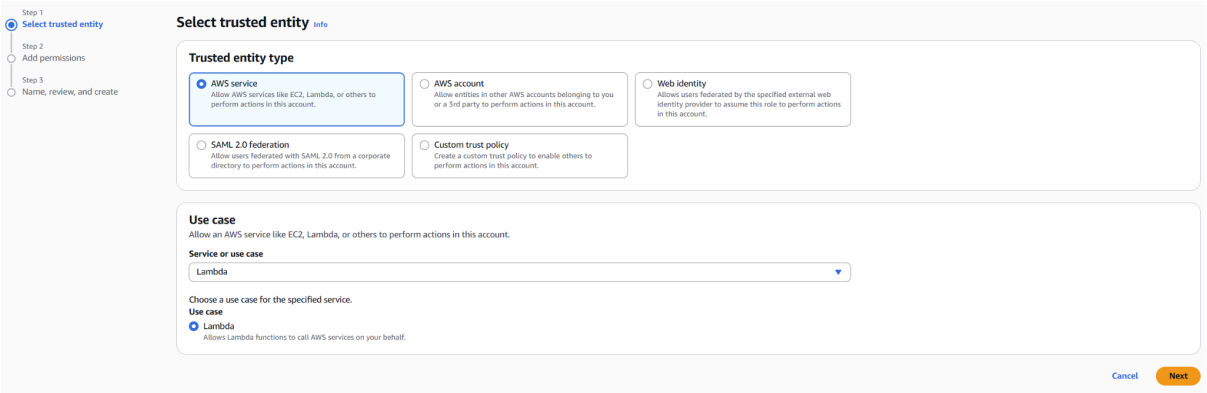
Lambda is going to need permissions to write to the above DynamoDB table. So, it's time to create an IAM role.

IAM

Navigate to the Identity Access Management console and create a new role. For trusted entity types will go with **AWS service** and for use case **Lambda**.



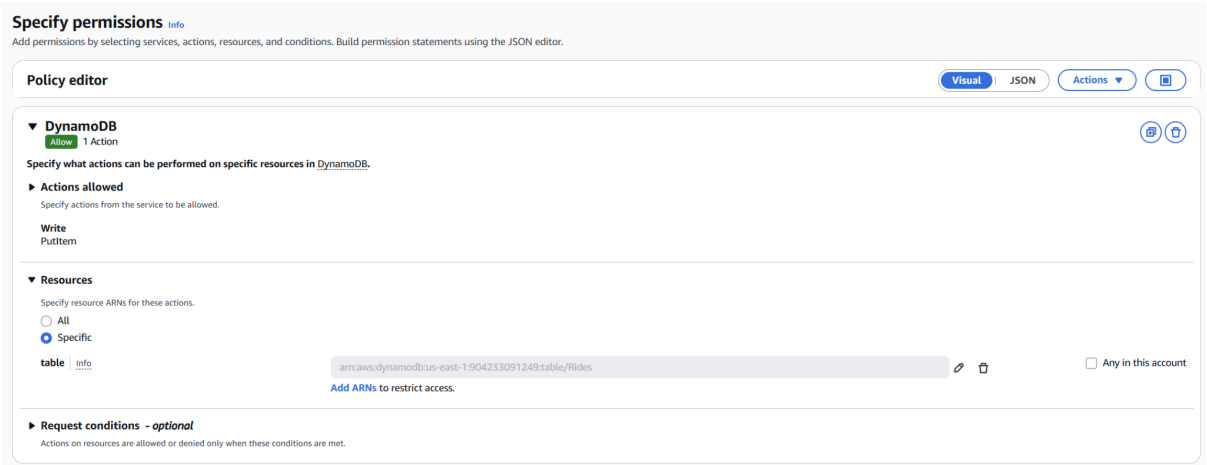
For the policy search for the existing **AWSLambdaBasicExecutionRole**.



Finally pick a name for the new IAM role. I'm naming the role **WildRydesLambda**. Now hit on *Create Role*.

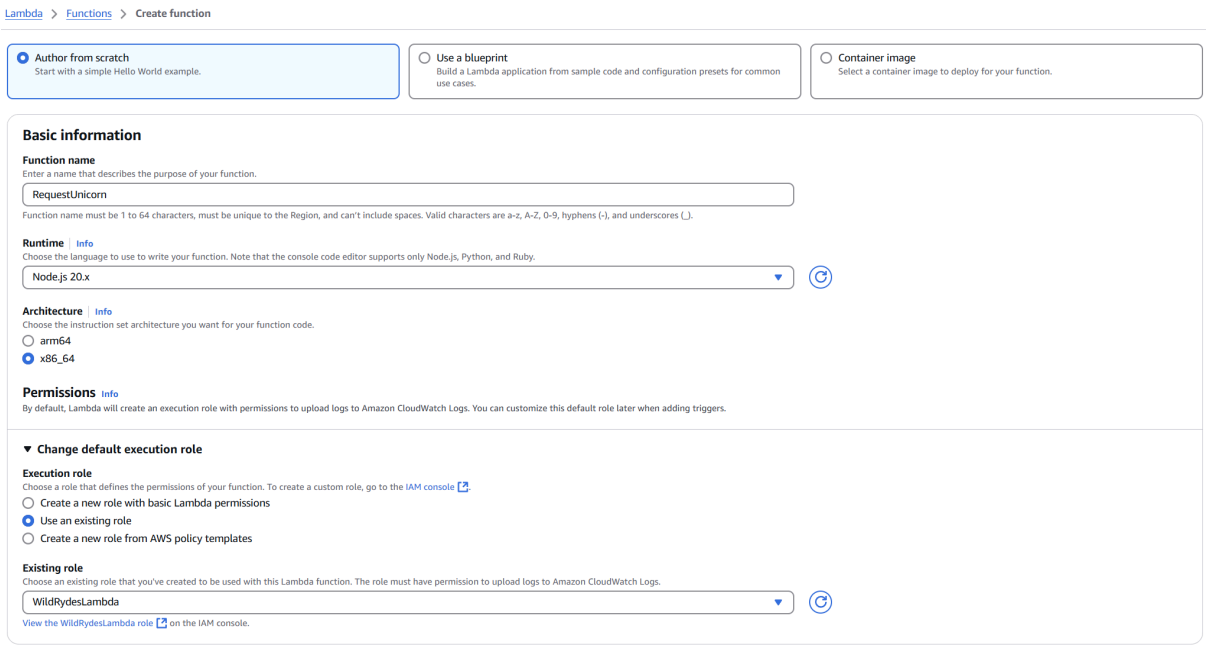
The role has been created. Click on the role and go to permissions. Under *Add Permissions* select *Create Inline Policy*.

Here make sure to specify the DynamoDB permission, so Lambda can insert items into the DynamoDB table. With that said, service will be **DynamoDB**, the permission type **putitem**, and the resource the **ARN** copied earlier. The policy name will be **DynamoDBWriteAccess**.



AWS Lambda

Navigate to Lambda to create a new function. It will be authored from scratch, so the name will be **RequestUnicorn**, the runtime Node.js 20.x and the execution role will be the one previously created **WildRydesLambda**.



Lambda Code

I've read and interpreted the Lambda code attached, and commented on the most relevant sections. I also decided to add some new horses to the fleet to have a few more options available. The rest of it has been kept the same.

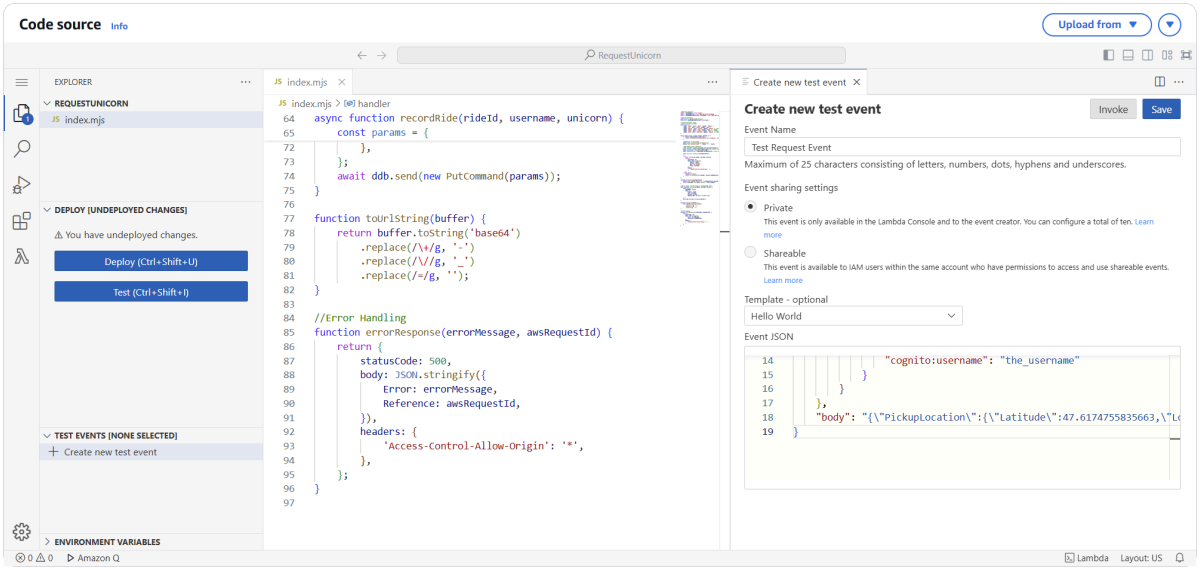
```
//Importing Libraries
import { randomBytes } from 'crypto';
import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
import { DynamoDBDocumentClient, PutCommand } from
'@aws-sdk/lib-dynamodb';

//Constant Variables
const client = new DynamoDBClient({});
const ddb = DynamoDBDocumentClient.from(client);

//Constant Variable 'fleet'
///List of horses available
const fleet = [
    { Name: 'Angel', Color: 'White', Gender: 'Female' },
    { Name: 'Gil', Color: 'Black', Gender: 'Male' },
    { Name: 'Spirit', Color: 'Yellow', Gender: 'Female' }, //Edited
    { Name: 'Comet', Color: 'Gray', Gender: 'Male' }, //Added
    { Name: 'Dust', Color: 'Brown', Gender: 'Male' }, //Added
    { Name: 'Echo', Color: 'Dark Blue', Gender: 'Female' }, //Added
];
```

```
export const handler = async (event, context) => {
    if (!event.requestContext.authorizer) {
                return  errorResponse('Authorization  not  configured',
context.awsRequestId);
    }

    //Randomizer for the RideID
    const rideId = toUrlString(randomBytes(16));
    console.log('Received event (', rideId, '): ', event);

    //AWS Cognito saved credentials
    ///The username is being retrieved from the requestContext
                                        const          username          =
event.requestContext.authorizer.claims['cognito:username'];
    const requestBody = JSON.parse(event.body);
     const pickupLocation = requestBody.PickupLocation; //This will be
the location clicked on the map

    const unicorn = findUnicorn(pickupLocation);

    try {
        await recordRide(rideId, username, unicorn);
        return {
            statusCode: 201,
            body: JSON.stringify({
                RideId: rideId,
                Unicorn: unicorn,
                Eta: '30 seconds',
                Rider: username,
            }),
            headers: {
                'Access-Control-Allow-Origin': '*',
            },
        };
    } catch (err) {
        console.error(err);
        return errorResponse(err.message, context.awsRequestId);
    }
};

function findUnicorn(pickupLocation) {
```

```
     console.log('Finding unicorn for ', pickupLocation.Latitude, ', ',
pickupLocation.Longitude);
    return fleet[Math.floor(Math.random() * fleet.length)];
}

//Here's where we write things to the DynamoDB table
async function recordRide(rideId, username, unicorn) {
    const params = {
        TableName: 'Rides',
        Item: {
            RideId: rideId,
            User: username,
            Unicorn: unicorn,
            RequestTime: new Date().toISOString(),
        },
    };
    await ddb.send(new PutCommand(params));
}

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

//Error Handling
function errorResponse(errorMessage, awsRequestId) {
    return {
        statusCode: 500,
        body: JSON.stringify({
            Error: errorMessage,
            Reference: awsRequestId,
        }),
        headers: {
            'Access-Control-Allow-Origin': '*',
        },
    };
}
```

Test Event

Will now create a test event to make sure the code works. For the test event name I'm using **Test Request Event**. Once saved click on Test.



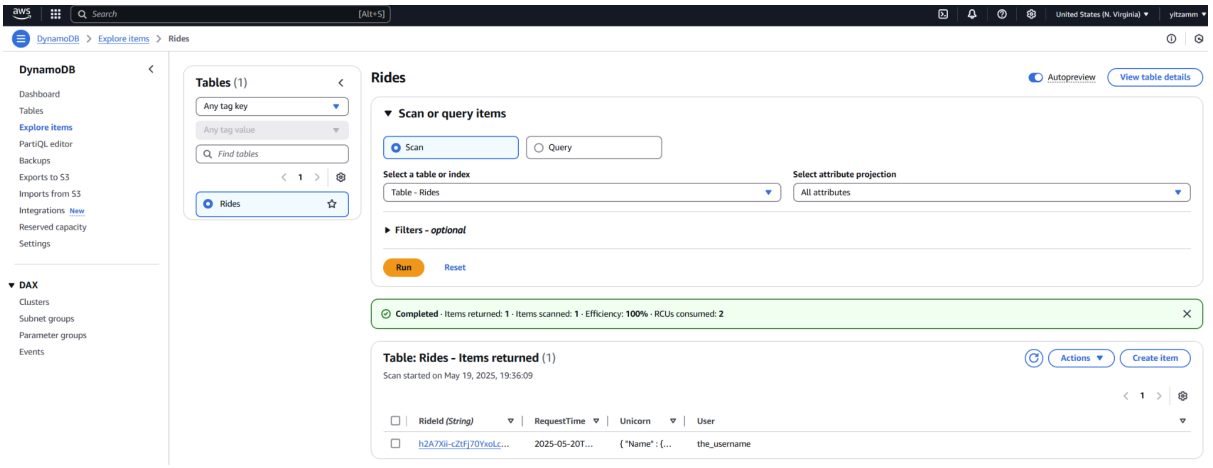
JSON Formatted Code


```
{
   "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"
         }
      }
   },
```

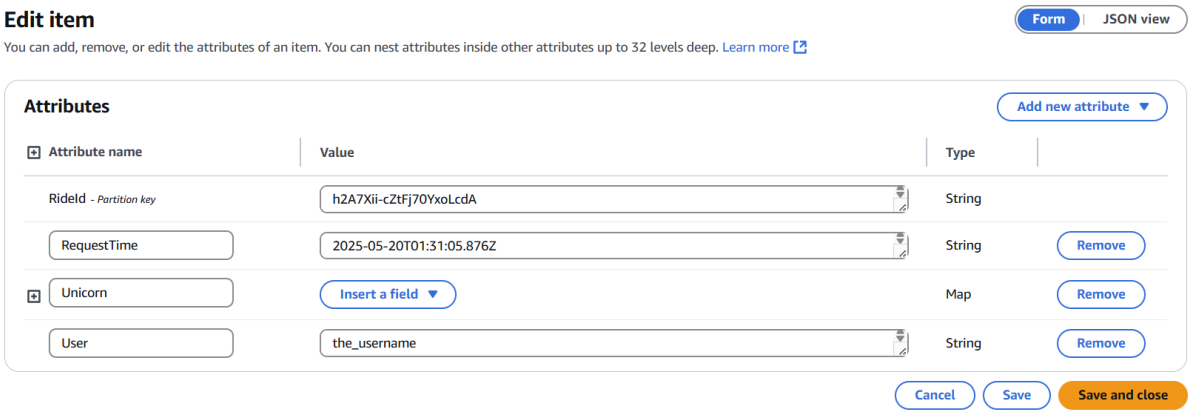
"body": "{\"PickupLocation\":{\"Latitude\":47.6174755835663,\"Longitude\":-122.28837066650185}}"
}

Output Returned

Back to the DynamoDB table under explore table items a new entry should come up.

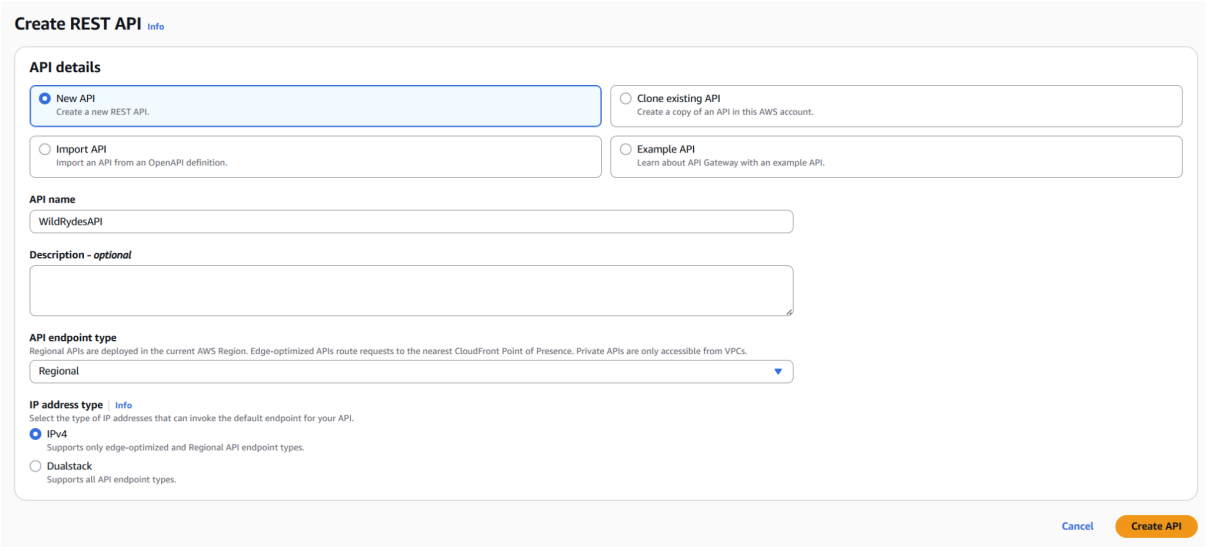


Some additional details such as the attribute names and their respective values will display on the screen by clicking on the entry.

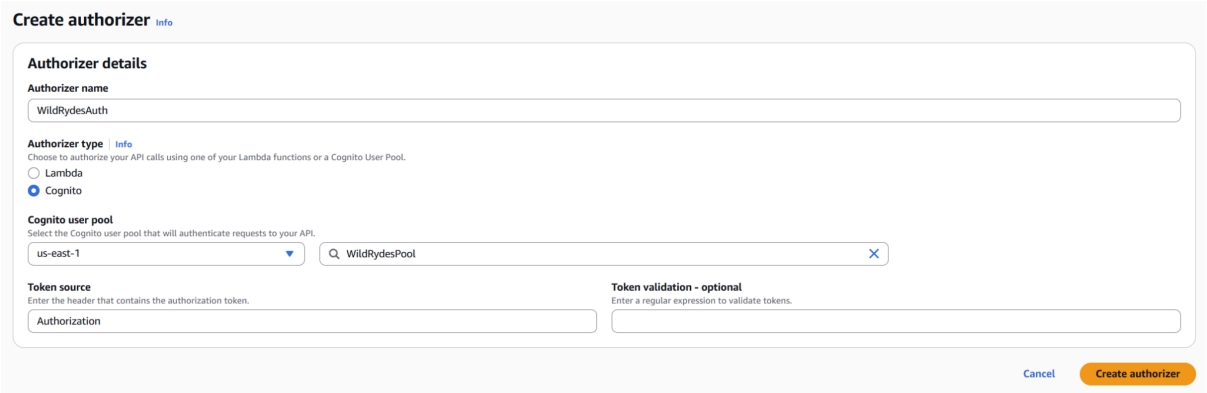


API Gateway

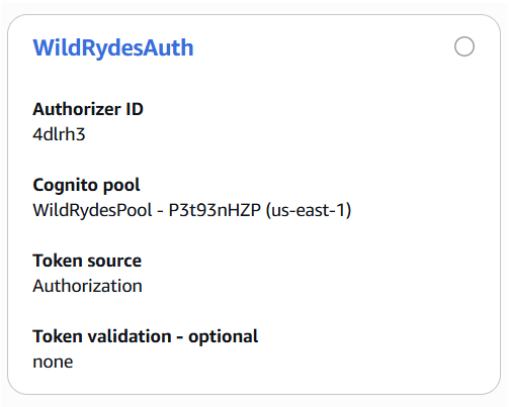
Navigate to the API Gateway service, hit on create API and then build REST API. For the API name I'll do WildRydesAPI.



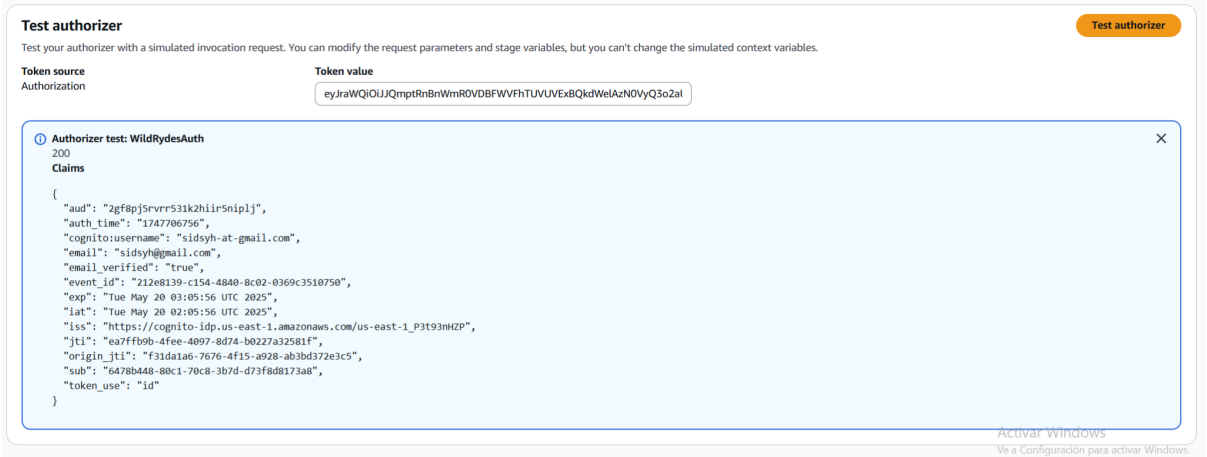
Because I'm using a Cognito user pool I need to create an authorizer in API Gateway. To authenticate calls API Gateway is going to use JSON web tokens or JWTs that are returned by Cognito. To do this, go to the Authorizer tab on the right and then create an authorizer. Will name it **WildRydesAuth**, the authorizer type is **Cognito**, the user pool previously created **WildRydesPool** should appear listed under the user-east-1 region, and finally the token source must be 'Authorization'.



Now click on the WildRydesAuth and copy the token value previously saved and hit on test.

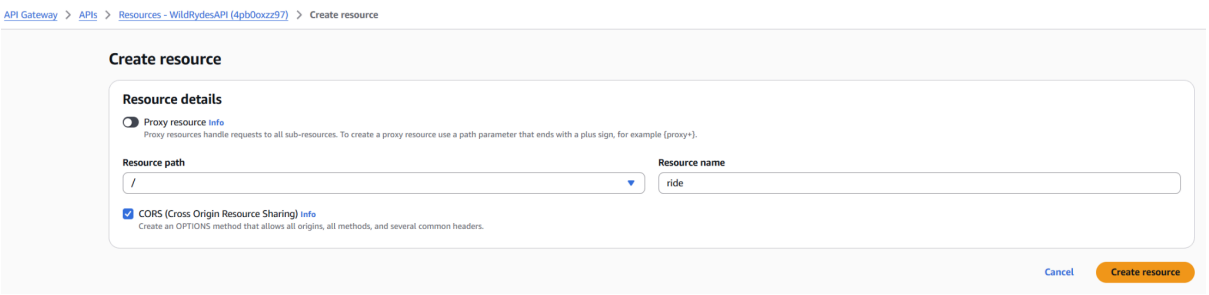


I realized that the token did expire after a while, so I got a ***401 Log Unauthorized request:*** *63da589c-4a19-4150-a068-58e19a0cee70* error when I went to use it for the authorization test. I had to log back in for a new token to be generated.

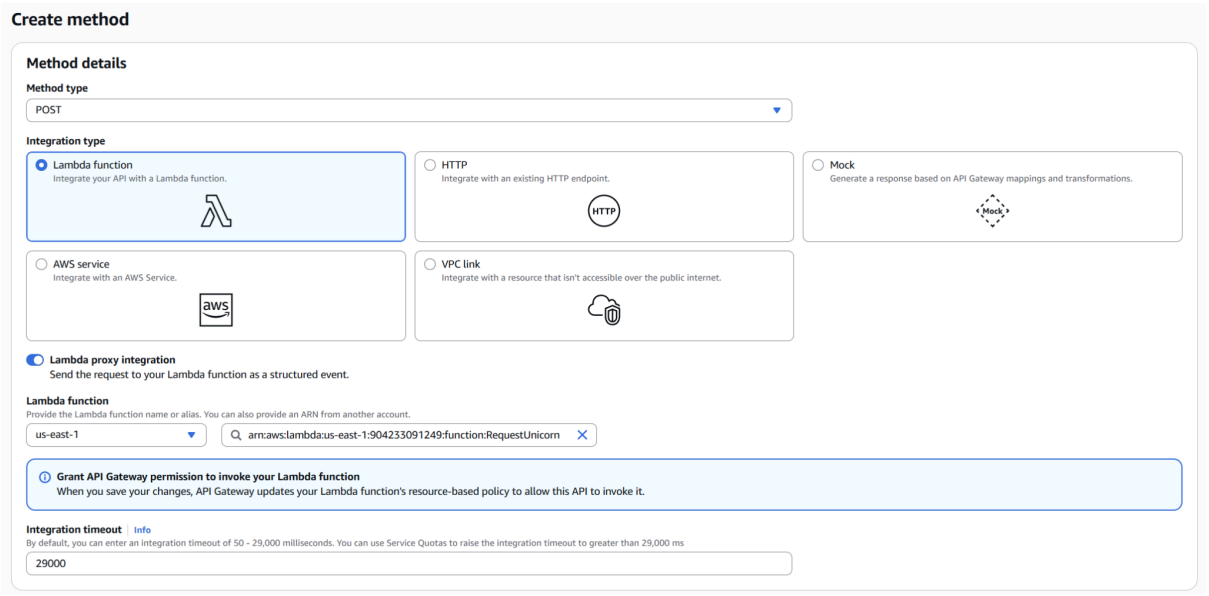


Go to the resources tab now to hook up the Lambda function. I'll do '**ride'** for the resource name and enable **CORS**. CORS must be enabled for the invocation to be successful.

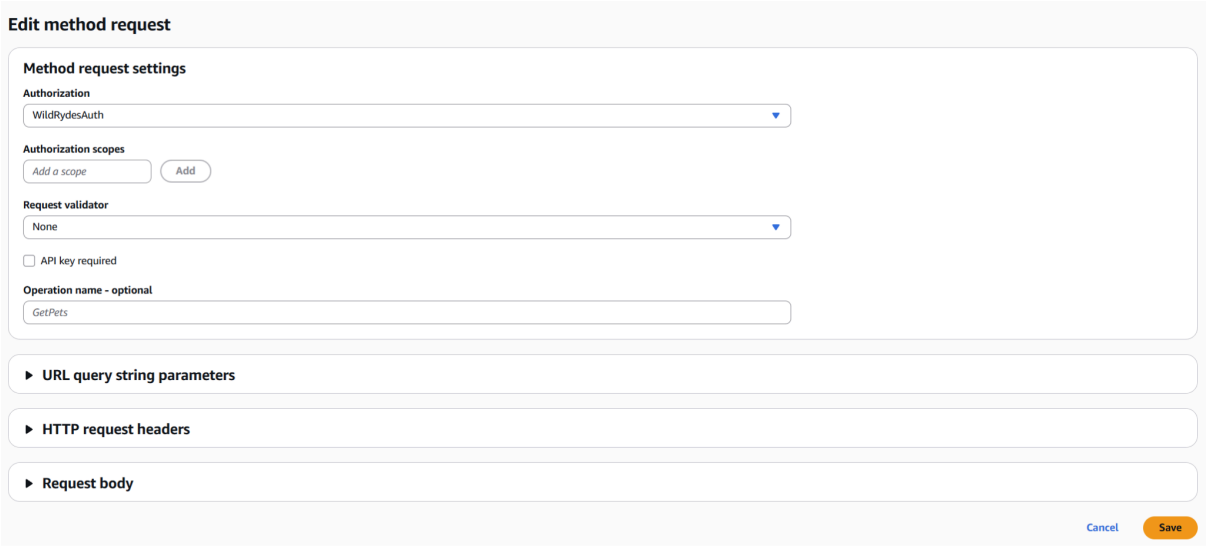
That's because the domain name of the site on Amplify will be different from the domain name of the API gateway, so it's going cross origin.



With /ride selected go to create method. The method type will be **POST** and integration type **Lambda**. Toggle on **Lambda proxy integration** and then select the Lambda function 'RequestUnicorn'.

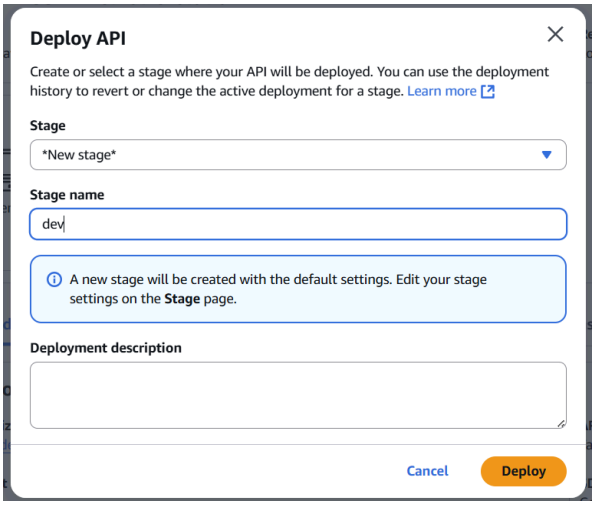


Once created click on edit to edit the method request settings. Select **WildRydesAuth** for the Authorization and hit save.

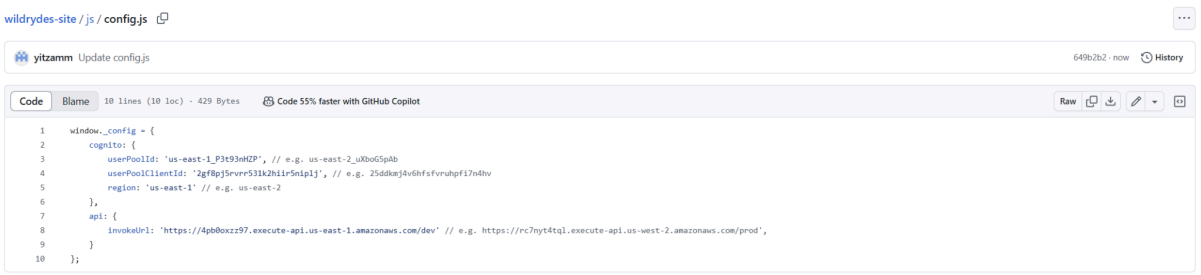


We're ready to deploy the API. Click on deploy API, stage ***New stage*** and name **dev**. Copy the Invoke URL displayed before exiting out.

Invoke URL: https://4pb0oxzz97.execute-api.us-east-1.amazonaws.com/dev

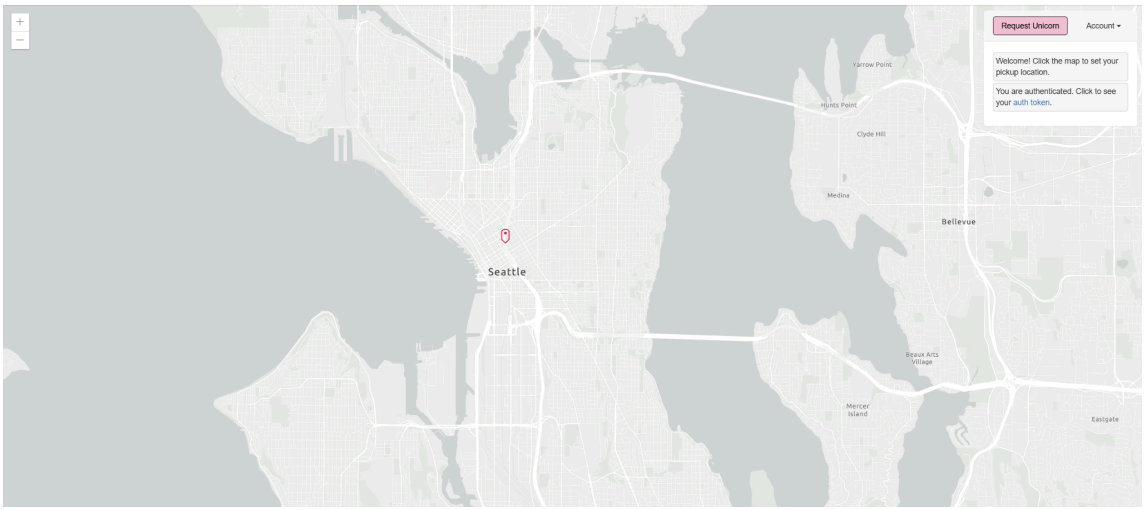


Back to GitHub one more time, open the config file to add the invoke URL copied and commit the changes.

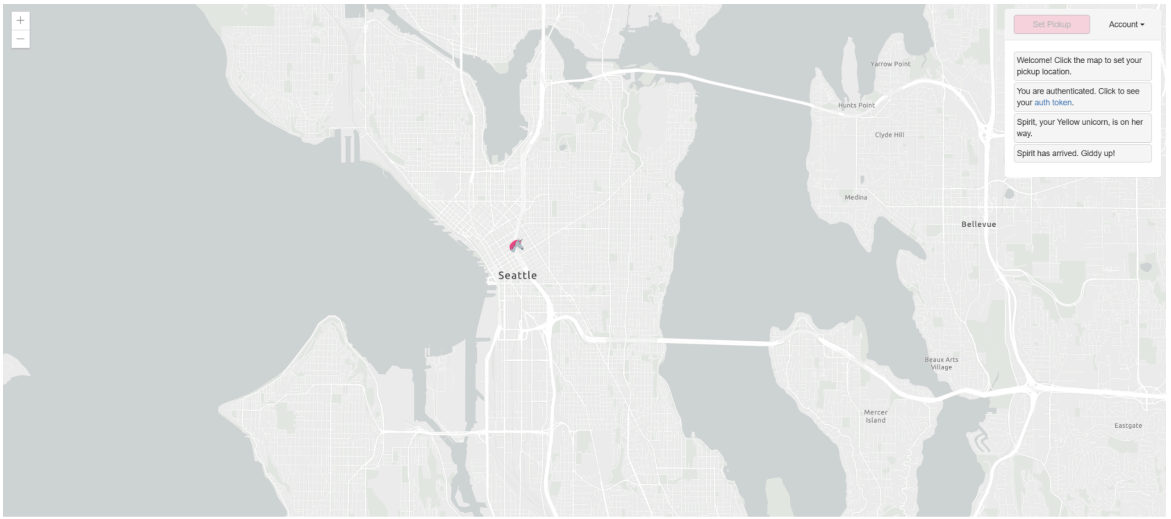


Final Testing

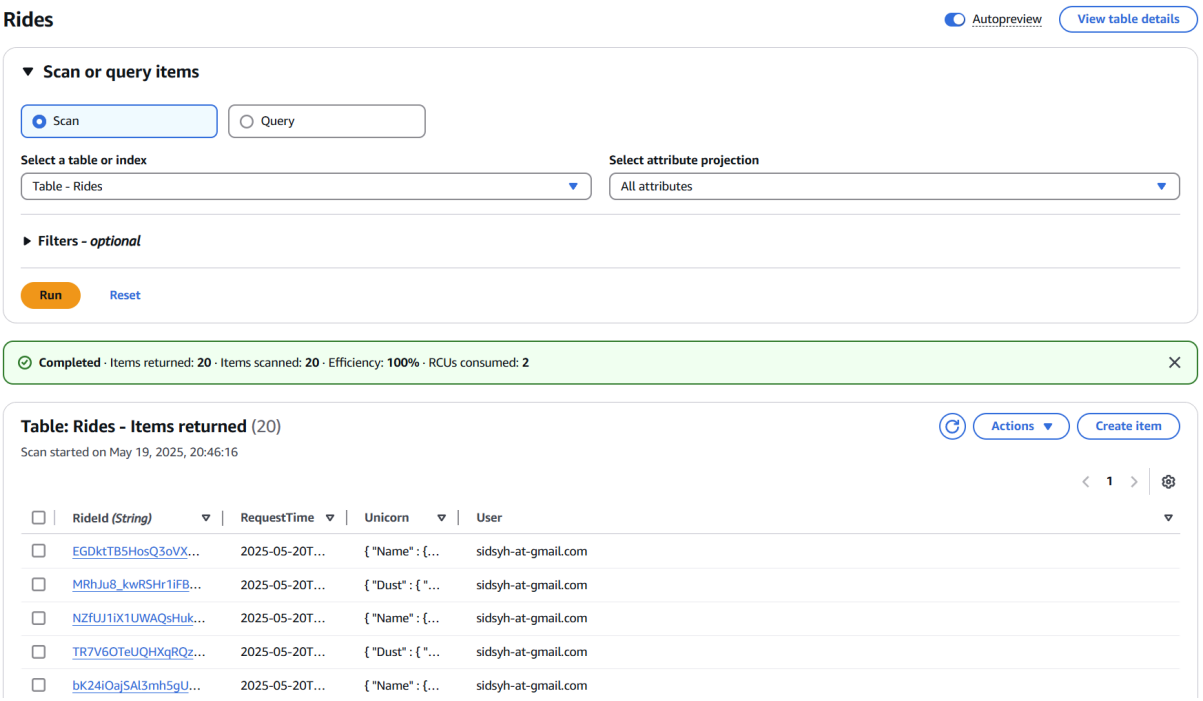
Go to https://main.d2qzfmq4oht1md.amplifyapp.com/ride.html and click on the desired pick up location on the map, then hit on the Request Unicorn button.



A unicorn from the predefined fleet should arrive, and the horse details should display on screen.



If we refresh the DynamoDB, we can see the new entries made have been properly registered to the table.



Learning Experience

From the project I was able to connect GitHub, Amplify , Cognito, Lambda, DynamoDB, IAM and API gateway and learn how these can interact with one another to get WildRydes to work. I encountered a couple of challenges along the way like the ***NotAuthorizedException*** error during the registration process due to the secret hash and then ***401 Log Unauthorized request*** error while testing the authorizer due to the expired token. It was a fun experience overall. I found that there's no sign in option on the website, and although the registration page is able to recognize you're already registered it wouldn't let you sign in unless you type it in manually on the browser. The sign-up option at the bottom of the site is non-functional, so you have to sign-up from the Giddy up button instead. The website can be polished to be more interactive and more functionalities can be added to the rides page so it looks more realistic, but for the purpose of this project it certainly fulfilled its purpose.