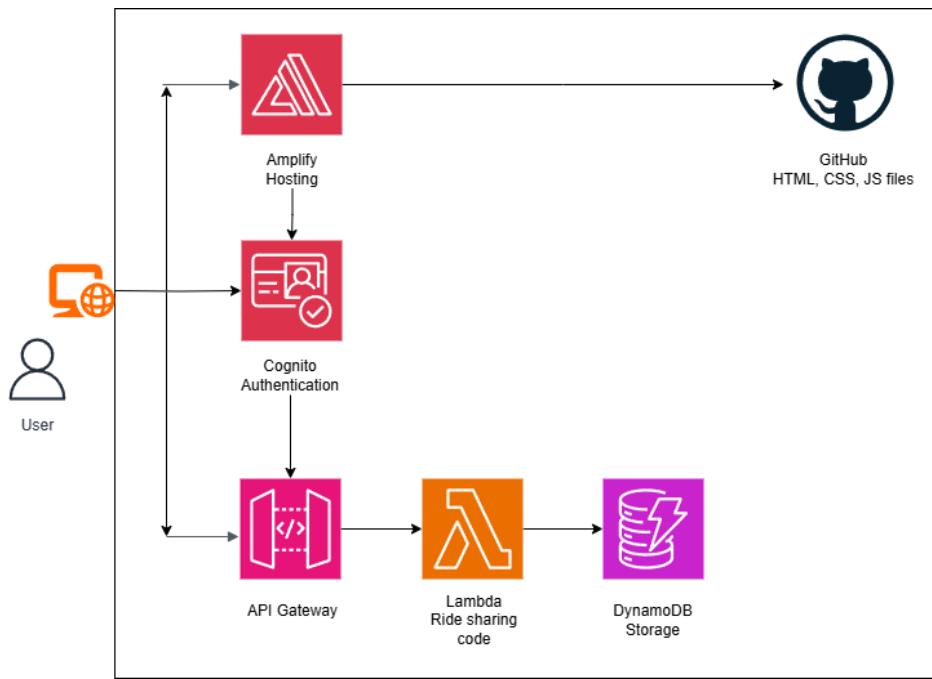


AWS Project: End-to-End AWS Web Application

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.

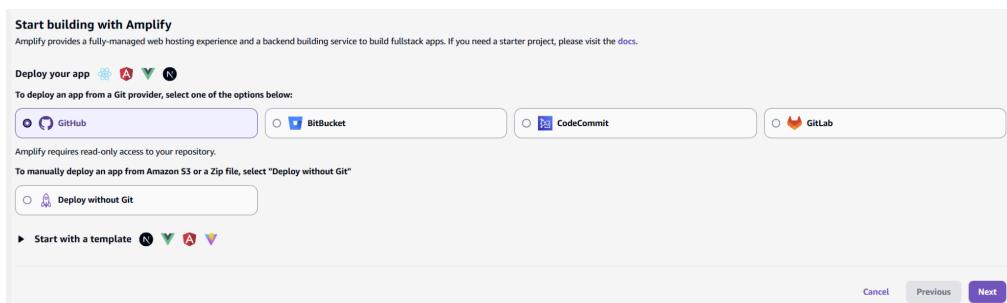
WildRydes Website

- 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>

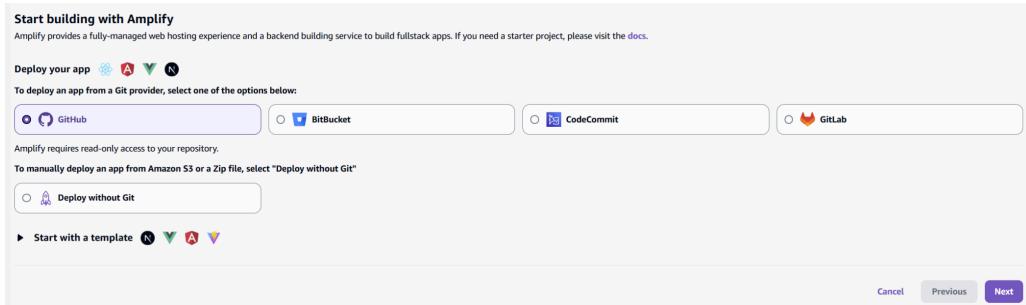
Step-by-Step Guide

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.

Name	Status	Build duration	Commit message	Started at
Deployment 2	Deployed	54 seconds	Update index.html	5/18/2025, 12:16 AM
Deployment 1	Deployed	48 seconds	Auto-build	5/17/2025, 11:43 PM

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**.

Define your application

Choose an application type and give it a name.

Application type: [Info](#)

Choose the type of application that you're developing. We will show example code for application like yours.

- Traditional web application**
An application hosted on a webserver. Uses redirects and separate pages to display information. Examples are Java, Python, node.js.
- Single-page application (SPA)**
A website with a single URL that updates content based on user interaction. Examples are JavaScript, Angular, React.
- Mobile app**
An app built with a mobile SDK. Examples are Android, iOS.
- Machine-to-machine application**
Platform-independent server-to-server communications without user interaction. Authorizes API access with OAuth 2.0 scopes.

Name your application: [Info](#)

Names are limited to 128 characters or fewer. Names may only contain alphanumeric characters, spaces, and the following special characters: + = . @ -

Configure options

You must make a few initial choices about the user pool that supports your application. To change these settings later, you must create a new user pool.

Options for sign-in identifiers: [Info](#)

Choose sign-in attributes. Usernames can be an email address, phone number, or a user-selected username. When you select only email and phone, users must select either email or phone as their username type. When username is an option, users can sign in with any options you select if they have provided a value for that option.

- Email
- Phone number
- Username

[Want to set up social, SAML, or OIDC sign-in?](#)

Required attributes for sign-up: [Info](#)

Choose any attributes that you want to require users to provide. With username alone, you must set email address or phone number as a required attribute.

Select attributes ▾

[X](#)

User's preferred email address.

⚠️ Options for sign-in identifiers and required attributes can't be changed after the app has been created.

Overview: WildRydesPool [Info](#)

[Rename](#) [Delete user pool](#)

User pool information		Created time	Last updated time
User pool name	WildRydesPool	May 18, 2025 at 00:48 CST	May 18, 2025 at 00:51 CST
User pool ID	us-east-1_P3t93nHZP		
ARN	arn:aws:cognito-idp:us-east-1:904233091249:userpool/us-east-1_P3t93nHZP		
Token signing key URL	https://cognito-idp.us-east-1.amazonaws.com/us-east-1_P3t93nHZP/well-known/jwks.json		
Estimated number of users	0		
Feature plan	Essentials		

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

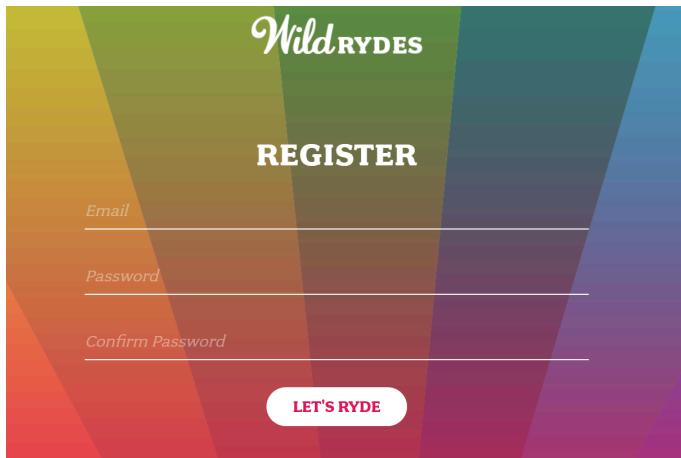
wildrydes-site / js / config.js [...](#)

[yitzamm](#) Update config.js 4d97be5 · now [History](#)

Code Blame 10 lines (10 loc) · 371 Bytes [Code 55% faster with GitHub Copilot](#)

```
1 window._config = {
2   cognito: {
3     userPoolId: 'us-east-1_P3t93nHZP', // e.g. us-east-2_uXboG5pAb
4     userPoolClientId: '2gf8pj5rvrr531k2hiir5niplj',
5     region: 'us-east-1' // e.g. us-east-2
6   },
7   api: {
8     invokeUrl: '' // e.g. https://rc7nyt4tq1.execute-api.us-west-2.amazonaws.com/prod',
9   }
10};
```

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 25g9fou5cvvijj9un8a26denuf 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.

wildrydes-site / js / config.js

yitzamm · Update config.js · 34bb892 · 7 minutes ago · History

Code Blame 10 lines (10 loc) · 371 Bytes · Code 55% faster with GitHub Copilot

```

1 window._config = {
2   cognito: {
3     userPoolId: 'us-east-1_P3t93mQP', // e.g. us-east-2_uXboG5pAb
4     userPoolClientId: '2gf8p5rvr53khmir5n1plj', // e.g. 25d0kej4v6hfsfvruhpfi7n4hv
5     region: 'us-east-1' // e.g. us-east-
6   },
7   api: {
8     invokeUrl: '' // e.g. https://rc7nyt4tql.execute-api.us-west-2.amazonaws.com/prod',
9   }
10 };

```

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:

```
eyJraWQiOijJQmpRnBnWmR0VDBFWVFhTUVUVExBQkdWeIAzN0VyQ3o2aUp5d2lGQTFnPStslmFsZyl6IIJTMyU2In0.eyJzdWIiOiI2NDc4YjQ0OC04MGMxLTcwYzgtM2I3ZC1kNzNmOGQ4MTczYTgiLCJlbWFpbF92ZXJpZmlIZCI6dHJ1ZSwiaXNzI
```

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.

Create table

Table details Info
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.
 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.
 1 to 255 characters and case sensitive.

Sort key - optional
You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.
 1 to 255 characters and case sensitive.

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**.

Step 1 **Select trusted entity** Info
 Step 2
 Add permissions
 Step 3
 Name, review, and create

Select trusted entity Info

Trusted entity type

AWS service Allow AWS services like EC2, Lambda, or others to perform actions in this account.

AWS account Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

Web identity Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

SAML 2.0 federation Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

Custom trust policy Create a custom trust policy to enable others to perform actions in this account.

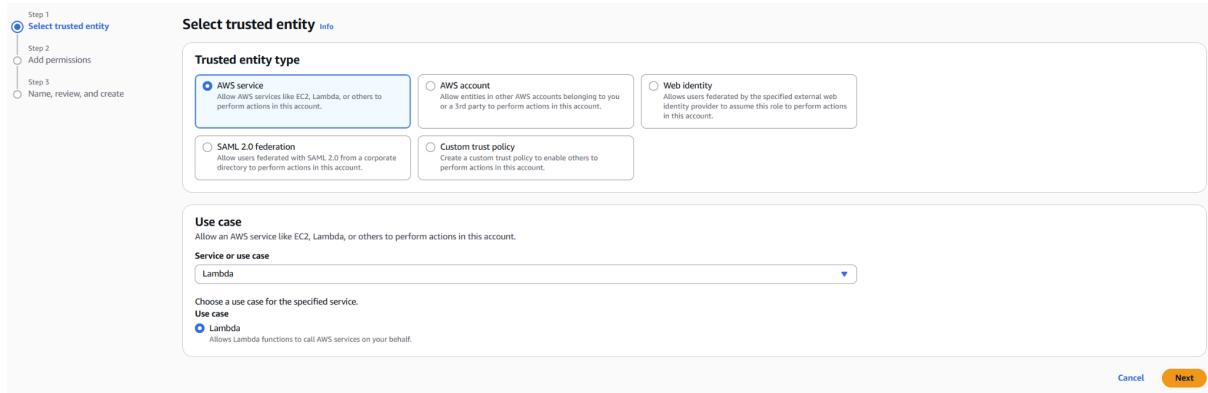
Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case
 Choose a use case for the specified service.

Use case
 Lambda Allows Lambda functions to call AWS services on your behalf.

Cancel

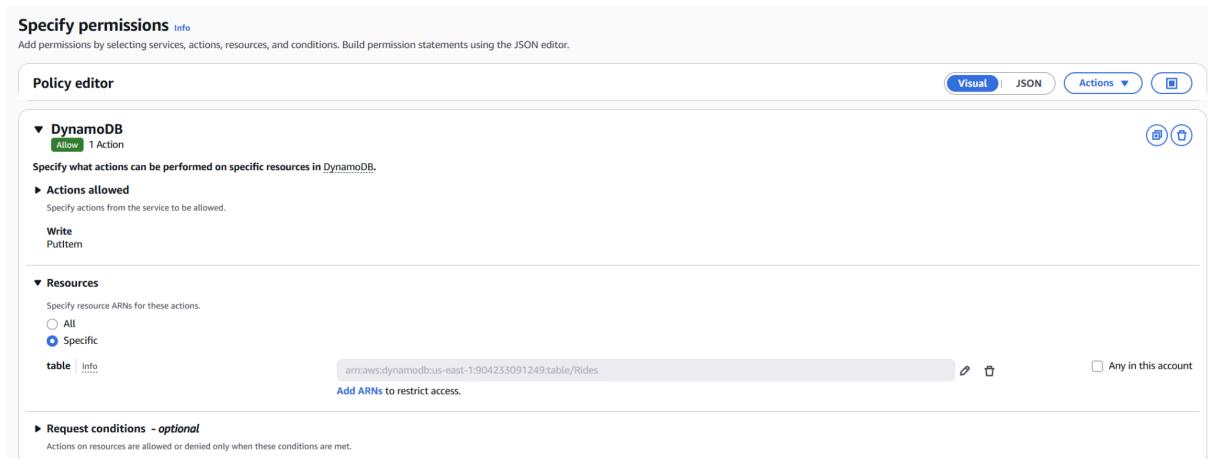
For the policy search for the existing **AWSLambdaBasicExecutionRole**.



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

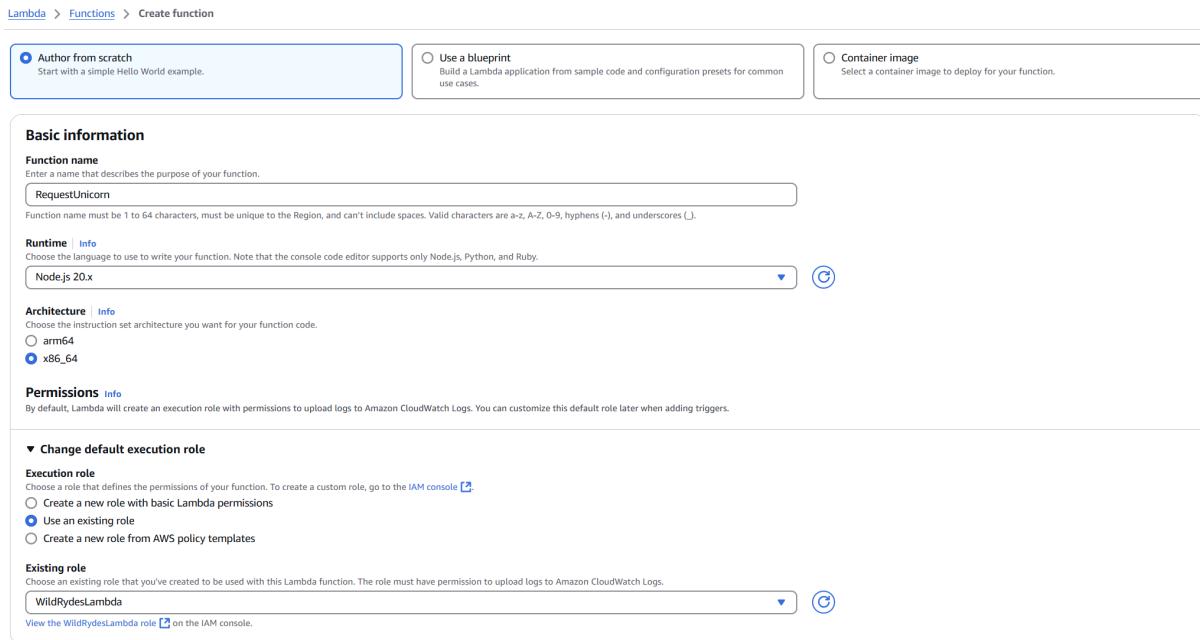
The role has been created. Click on the role and go to permissions. Under *Add Permissions* select *Create Inline Policy*.

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));
}

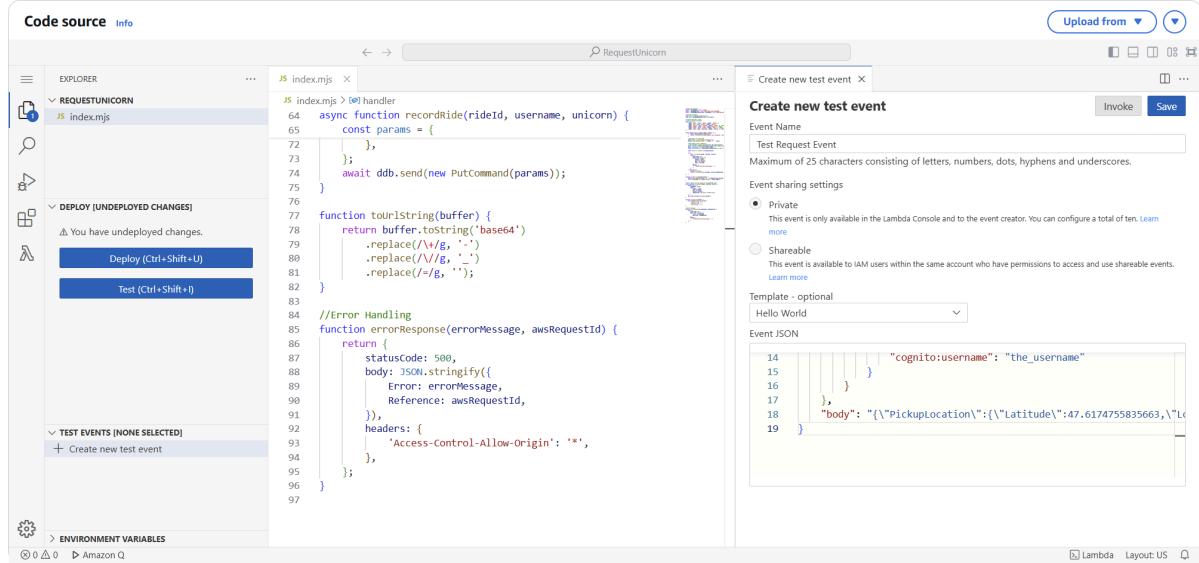
function toUrlString(buffer) {
    return buffer.toString('base64')
        .replace(/\+/g, '-')
        .replace(/\//g, '_')
        .replace(/=/g, '');
}

//Error Handling
function errorMessage(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

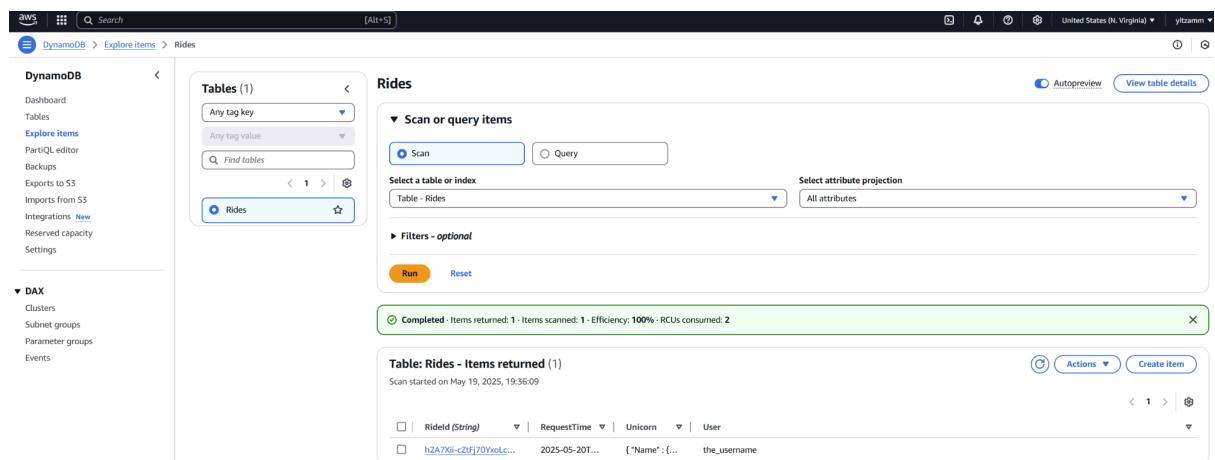
```
{
  "path": "/ride",
  "httpMethod": "POST",
  "headers": {
    "Accept": "*/*",
    "Authorization": "eyJraWQiOiJLTzRVMWZs",
    "content-type": "application/json; charset=UTF-8"
  },
  "queryStringParameters": null,
  "pathParameters": null,
  "requestContext": {
    "authorizer": {
      "claims":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

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

Output Returned

```
PROBLEMS    OUTPUT    CODE REFERENCE LOG    TERMINAL  
  
Status: Succeeded  
Test Event Name: TestRequestEvent  
  
Response:  
{  
  "statusCode": 201,
```

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



The screenshot shows the AWS DynamoDB console interface. On the left, there's a navigation sidebar with sections for 'DynamoDB' (Dashboard, Tables, Explore items, etc.) and 'DAX' (Clusters, Subnet groups, Parameter groups, Events). The main area is titled 'Rides' and shows a table with one item. The table has columns: Ridelid (String), RequestTime (String), Unicorn (String), and User (String). The single item listed is: Ridelid: h2A7X0i-cZfJ70YwL..., RequestTime: 2025-05-20T..., Unicorn: {"Name": "..."}, and User: the_username. A message at the bottom indicates a successful scan operation: 'Completed - Items returned: 1 - Items scanned: 1 - Efficiency: 100% - RCU consumed: 2'.

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

Edit item

Form | JSON view

You can add, remove, or edit the attributes of an item. You can nest attributes inside other attributes up to 32 levels deep. [Learn more](#)

Attributes		Type	Add new attribute ▾
<input type="checkbox"/> Attribute name	Value		
Rideld - Partition key	h2A7Xii-cZtFj70YxoLcdA	String	<input type="button" value="Remove"/>
<input type="checkbox"/> RequestTime	2025-05-20T01:31:05.876Z	String	<input type="button" value="Remove"/>
<input type="checkbox"/> Unicorn	<input type="button" value="Insert a field ▾"/>	Map	<input type="button" value="Remove"/>
<input type="checkbox"/> User	the_username	String	<input type="button" value="Remove"/>

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.

Create REST API [Info](#)

API details

New API
Create a new REST API.

Clone existing API
Create a copy of an API in this AWS account.

Import API
Import an API from an OpenAPI definition.

Example API
Learn about API Gateway with an example API.

API name

Description - optional

API endpoint type
Regional APIs are deployed in the current AWS Region. Edge-optimized APIs route requests to the nearest CloudFront Point of Presence. Private APIs are only accessible from VPCs.
 Regional

IP address type [Info](#)
Select the type of IP addresses that can invoke the default endpoint for your API.

IPv4
Supports only edge-optimized and Regional API endpoint types.

Dualstack
Supports all API endpoint types.

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'.

Create authorizer [Info](#)

Authorizer details

Authorizer name
WildRydesAuth

Authorizer type [Info](#)
Choose to authorize your API calls using one of your Lambda functions or a Cognito User Pool.
 Lambda
 Cognito

Cognito user pool
Select the Cognito user pool that will authenticate requests to your API.
 us-east-1 Q WildRydesPool

Token source
Enter the header that contains the authorization token.
Authorization

Token validation - optional
Enter a regular expression to validate tokens.

[Cancel](#) [Create authorizer](#)

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

WildRydesAuth

Authorizer ID
4dlrh3

Cognito pool
WildRydesPool - P3t93nHZP (us-east-1)

Token source
Authorization

Token validation - optional
none

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.

Test authorizer

Test your authorizer with a simulated invocation request. You can modify the request parameters and stage variables, but you can't change the simulated context variables.

Token source
Authorization

Token value
eyJraWQiOiJJQmpfRnBnWmR0VDBFWVFhTUUVUExBQkdWeAzN0VYQ3o2al

Authorizer test: WildRydesAuth

200

Claims

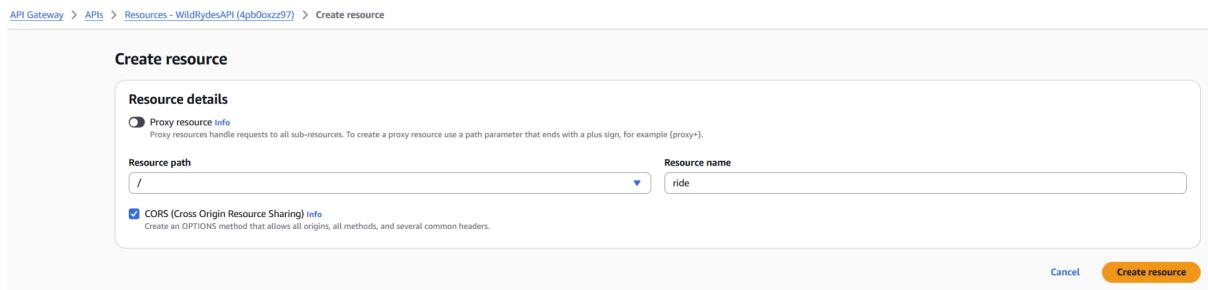
```
{
  "aud": "2gf8pj5rvrr51k2h1r5nplj",
  "auth_time": "1747706756",
  "cognito:username": "sidsyh-at-gmail.com",
  "email": "sidsyh@gmail.com",
  "email_verified": "true",
  "event_id": "21268139-c154-4840-8c02-0369c3510750",
  "exp": "Tue May 20 03:05:54 UTC 2025",
  "iat": "Tue May 20 02:05:54 UTC 2025",
  "iss": "https://cognito-idp.us-east-1.amazonaws.com/us-east-1_P3t93nHZP",
  "jti": "ea7ff9b-4fee-4097-8d74-b0227a32581f",
  "origin_jti": "f31da1a6-7676-4f15-a928-ab3bd372e3c5",
  "sub": "6478b448-80c1-70c8-3b7d-d73f8d8173a8",
  "token_use": "id"
}
```

[Test authorizer](#)

Activar Windows
Ve a Configuración para activar Windows.

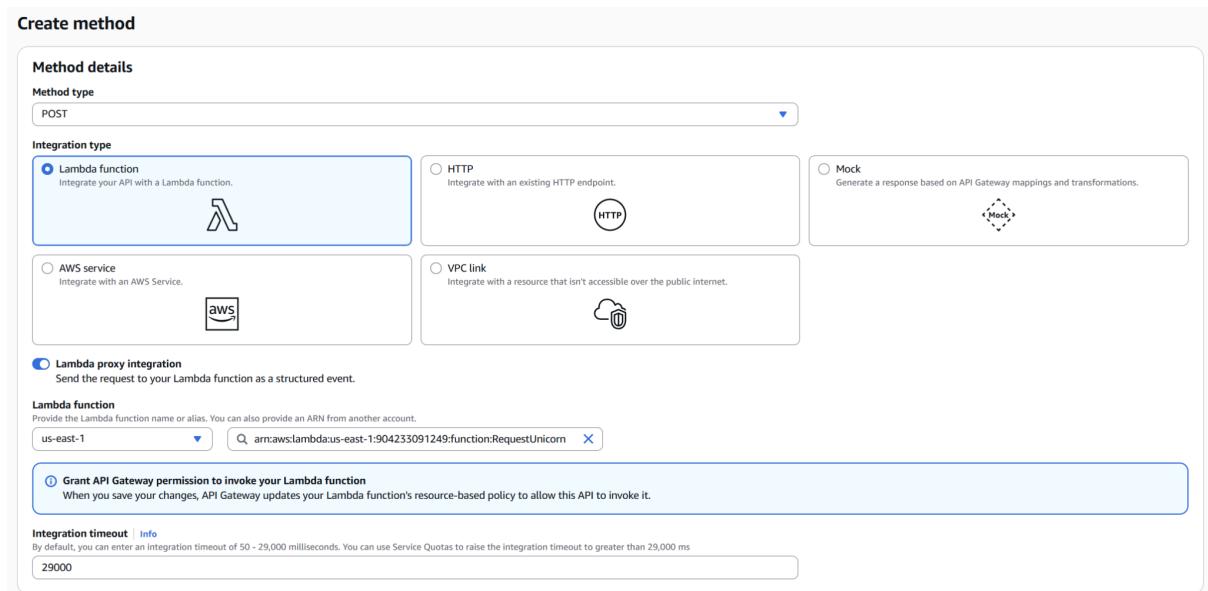
Go to the resources tab now to hook up the Lambda function. I'll do '**ride**' for the resource name and enable **CORS**. CORS must be enabled for the invocation to be successful.

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.



The screenshot shows the 'Create resource' interface in the AWS API Gateway. The 'Resource path' is set to '/' and the 'Resource name' is 'ride'. The 'Proxy resource info' checkbox is checked. There is a note about CORS. At the bottom right are 'Cancel' and 'Create resource' buttons.

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'.



The screenshot shows the 'Create method' interface in the AWS API Gateway. The 'Method type' is 'POST'. Under 'Integration type', 'Lambda function' is selected. Other options like 'HTTP', 'Mock', 'AWS service', and 'VPC link' are shown. Under 'Lambda function', 'us-east-1' is selected with ARN 'Q_amaws:lambda:us-east-1:904233091249:function:RequestUnicorn'. A note about granting API Gateway permission to invoke the Lambda function is present. At the bottom is an 'Integration timeout' field set to 29000 ms.

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

Edit method request

Method request settings

Authorization: WildRidesAuth

Authorization scopes: Add a scope, Add

Request validator: None

API key required

Operation name - optional: GetPets

URL query string parameters

HTTP request headers

Request body

[Cancel](#) [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>

Deploy API

Create or select a stage where your API will be deployed. You can use the deployment history to revert or change the active deployment for a stage. [Learn more](#)

Stage: *New stage*

Stage name: dev

ⓘ A new stage will be created with the default settings. Edit your stage settings on the [Stage page](#).

Deployment description

[Cancel](#) [Deploy](#)

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

wildrides-site / js / config.js

yitzamm Update config.js

Code Blame 10 lines (10 loc) - 429 Bytes [Code 55% faster with GitHub Copilot](#)

```

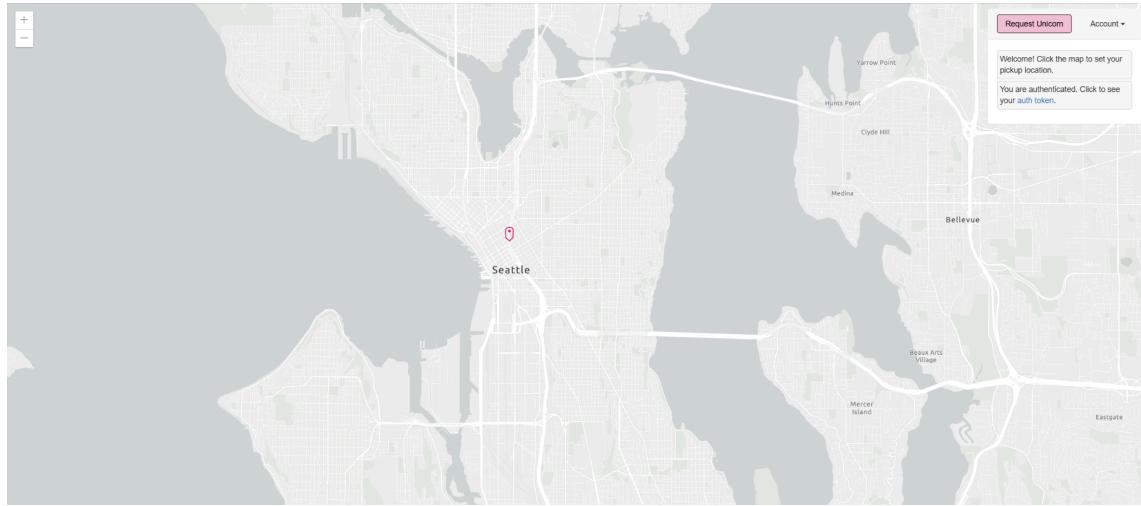
1 window._config = {
2   cognito: {
3     userPoolId: 'us-east-1_P3t93nHZP', // e.g. us-east-2_0xb065pAb
4     userPoolClientId: '2g8pj5vr531k2h1r5n1plj', // e.g. 25ddkmj4v6hfsfvruhpfi7n4hv
5     region: 'us-east-1' // e.g. us-east-2
6   },
7   api: {
8     invokeUrl: 'https://4pb0oxzz97.execute-api.us-east-1.amazonaws.com/dev' // e.g. https://rc7nyt4tql.execute-api.us-west-2.amazonaws.com/prod',
9   }
10 };

```

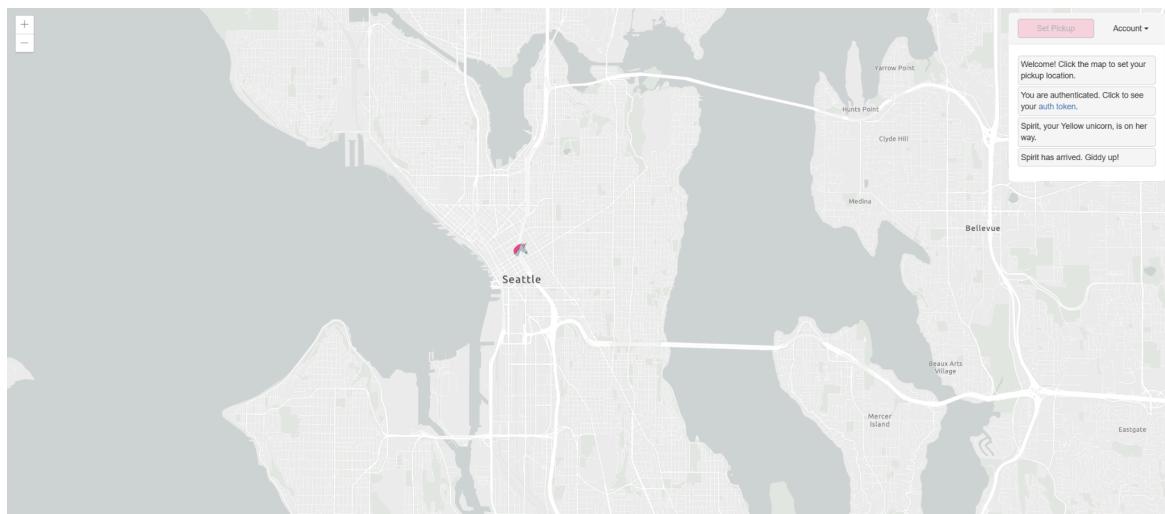
Raw [Copy](#) [Edit](#) [Diff](#) [History](#)

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.

Rides

Scan Query

Select a table or index
Table - Rides

Select attribute projection
All attributes

▶ Filters - optional

Run [Reset](#)

Completed · Items returned: 20 · Items scanned: 20 · Efficiency: 100% · RCUs consumed: 2 [X](#)

Table: Rides - Items returned (20)

Scan started on May 19, 2025, 20:46:16

<input type="checkbox"/>	RideId (String)	RequestTime	Unicorn	User
<input type="checkbox"/>	EGDktTB5HosQ3oVX...	2025-05-20T...	{"Name": "...", "Dust": {...}}	sidsyh-at-gmail.com
<input type="checkbox"/>	MRhJu8_kwRSh1iFB...	2025-05-20T...	{"Name": "...", "Dust": {...}}	sidsyh-at-gmail.com
<input type="checkbox"/>	NzfUJ1iX1UWAQsHuk...	2025-05-20T...	{"Name": "...", "Dust": {...}}	sidsyh-at-gmail.com
<input type="checkbox"/>	TR7V60TeUQHxqRQz...	2025-05-20T...	{"Name": "...", "Dust": {...}}	sidsyh-at-gmail.com
<input type="checkbox"/>	bK24iOajSAI3mh5gU...	2025-05-20T...	{"Name": "...", "Dust": {...}}	sidsyh-at-gmail.com

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.