

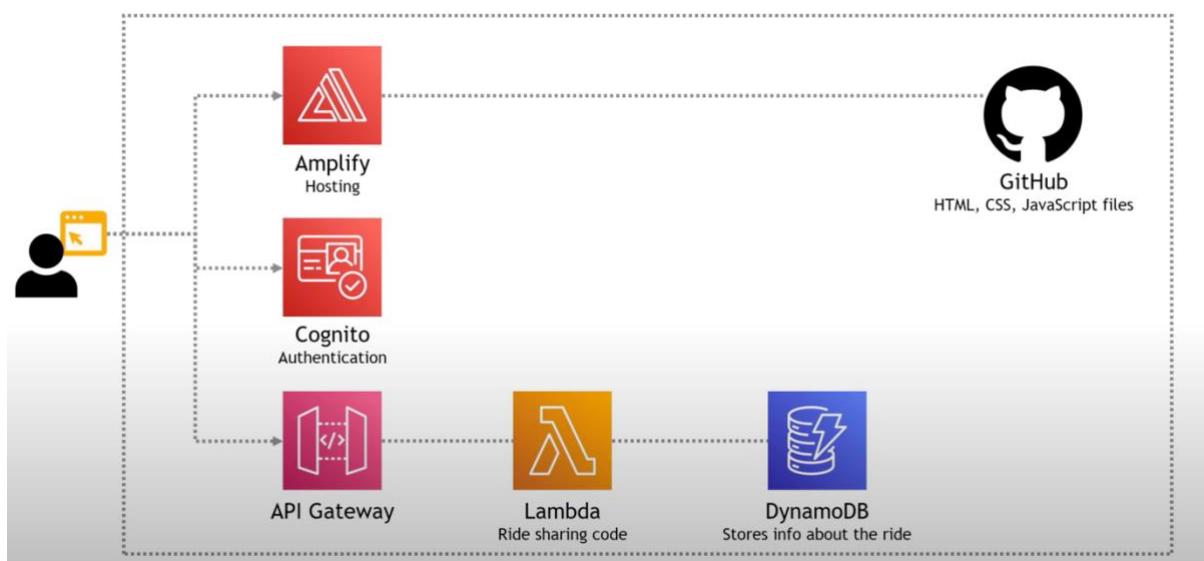
Wild Rydes Application (an AWS sample project)

In this project, we will be building **Wild Rydes**, a fun take on an Uber-like ride service, but with unicorns! The application will allow users to register, log in, and request a ride by simply clicking on a map. Along the way, you will gain hands-on experience with several key AWS services that power modern, serverless applications.

We will use the following seven AWS services to build this application: GitHub, Amplify, Cognito, Lambda, IAM, API Gateway and DynamoDB.

Throughout this project, you will not only interact with these services but also understand how they work together to provide a scalable, secure, and responsive application. By the end of this lab, you will have a working Wild Rydes application where users can interact with a unicorn ride-sharing system through a simple map interface.

Below is the architecture we are going to build.



AWS Amplify:

A development platform that simplifies building and deploying full-stack web and mobile applications. It helps set up and connect backend services like authentication, APIs, and storage while offering hosting for frontend applications.

Amazon DynamoDB:

A fully managed NoSQL database service that delivers high performance, scalability, and low latency. Ideal for applications needing fast and flexible key-value or document data storage.

AWS Lambda:

A serverless compute service that lets you run code without provisioning or managing servers. Code executes in response to triggers such as API calls, database changes, or events from other AWS services.

Amazon API Gateway:

A fully managed service for creating, publishing, and managing APIs. It acts as a bridge between your frontend application and backend services like Lambda, enabling secure and scalable API interactions.

Amazon Cognito:

A service for adding user authentication and authorization to your applications. It supports sign-up, sign-in, and access control with built-in integrations for identity providers like Google and Facebook.

GitHub:

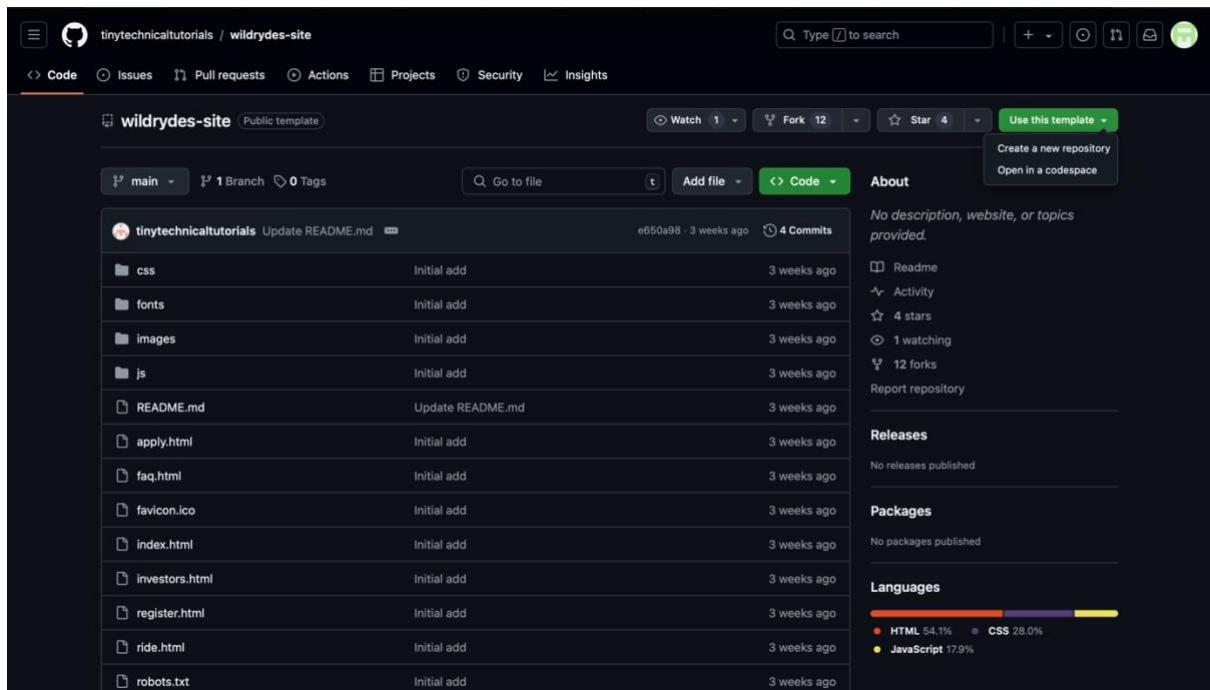
A version control and collaboration platform for managing code repositories. It allows teams to track changes, manage branches, and collaborate on projects efficiently.

Note: If you don't already have a GitHub account, go to <https://github.com/> and create an account.

For this project, you will need the code files, and they are available in the repository linked below.

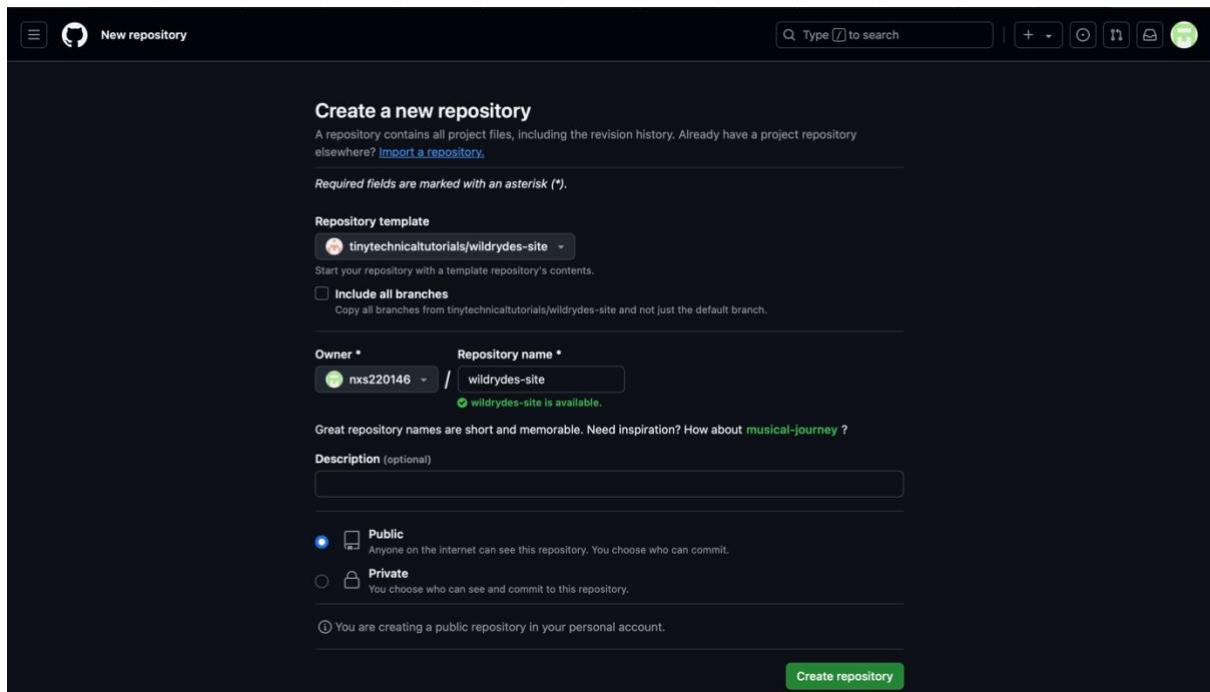
To bring the code to your GitHub account, follow these steps:

1. Go to the following link: <https://github.com/tinytechnicaltutorials/wildrydes-site>.
2. Click on the **Use this template** button at the top-right of the repository page.
3. Select **Create a new repository**.

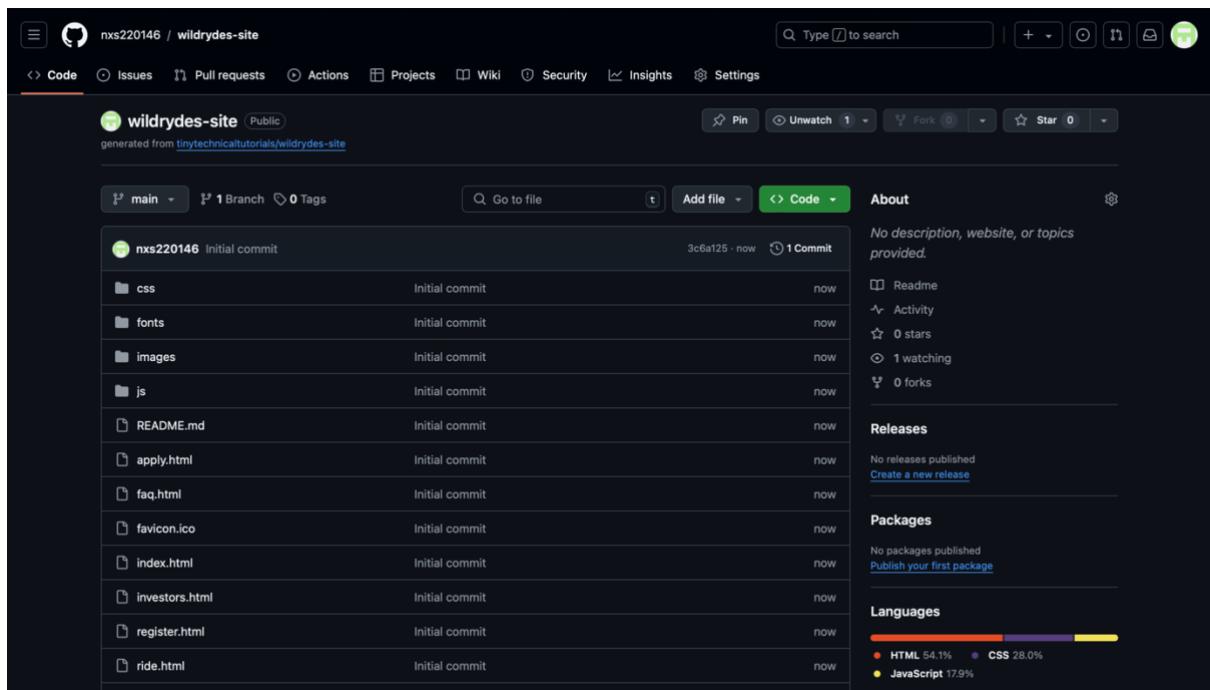


Fill in the **Repository name** as **wildrydes-site** .

Click **Create repository** to generate your own version of the repository.

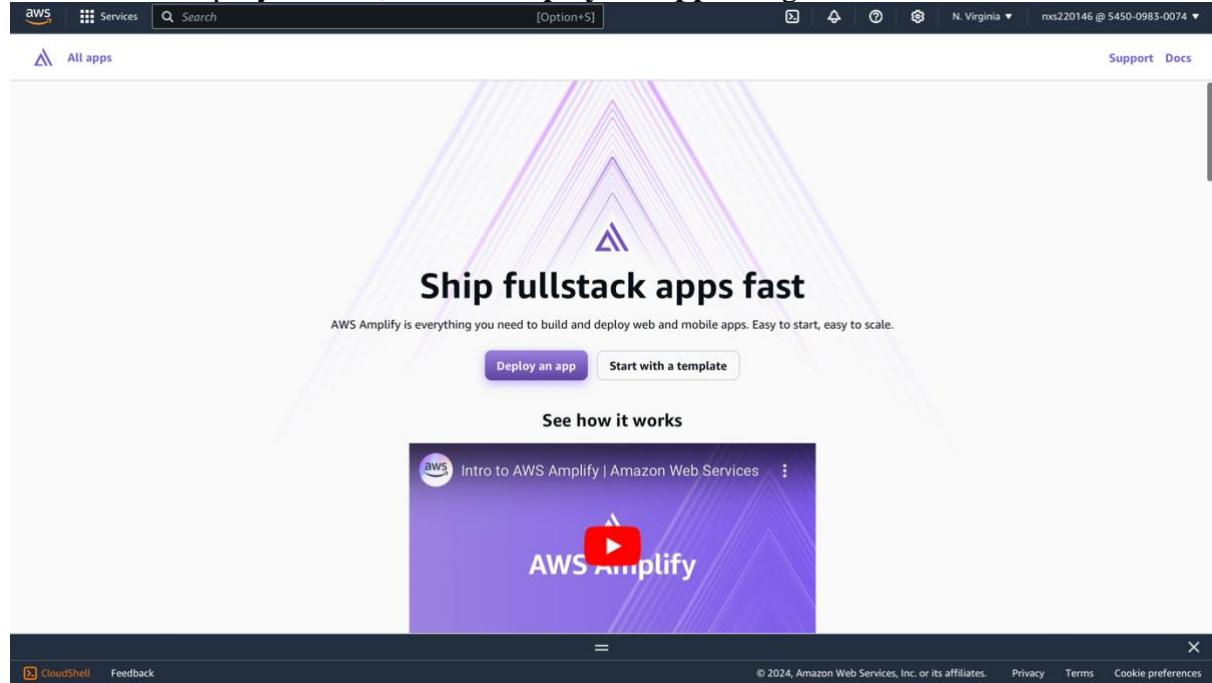


You should see something like the screenshot below. This screenshot shows the repository with all the code files now available in your GitHub account.

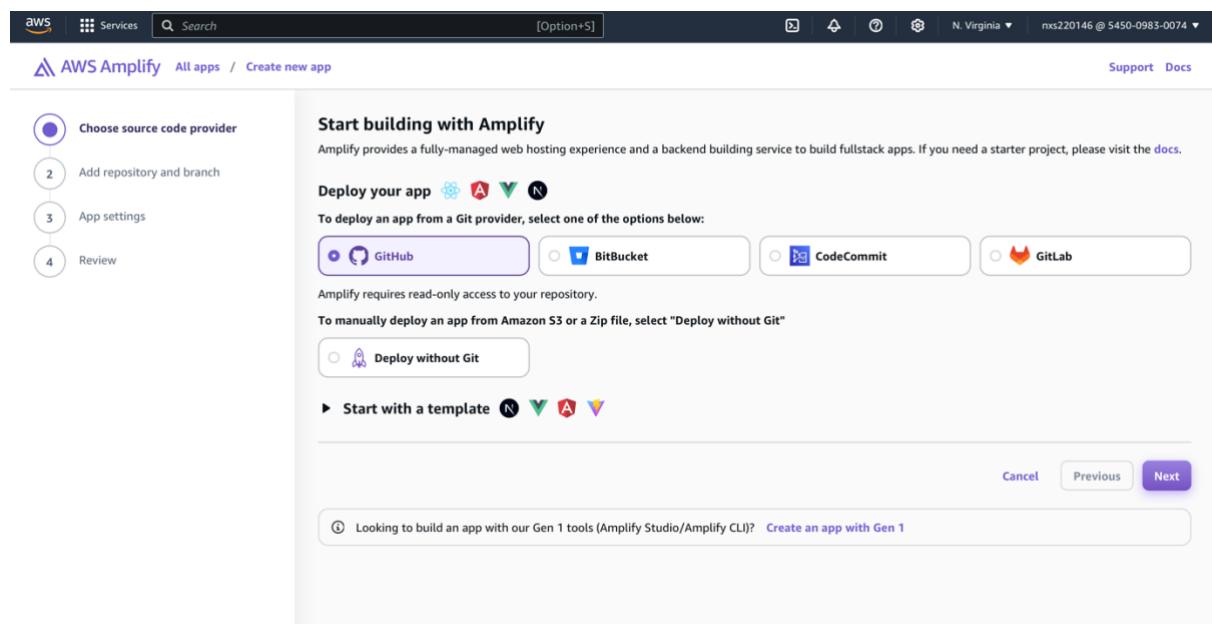


i. Creating a new app for hosting in AWS Amplify

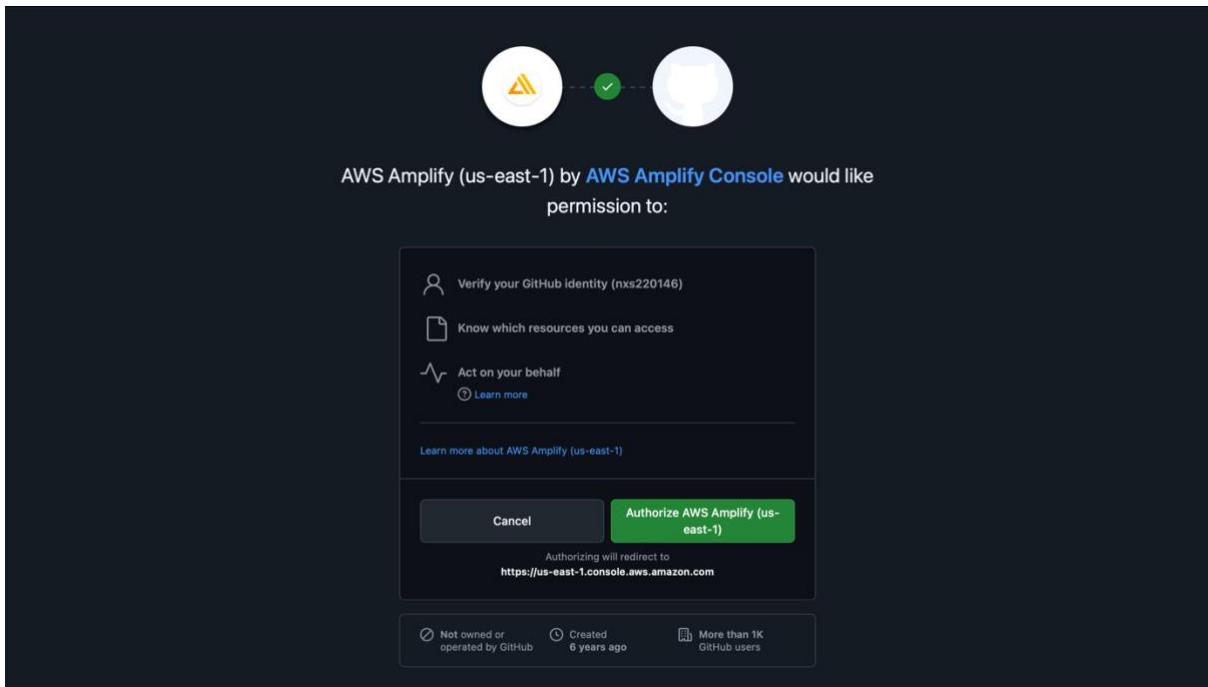
Click **Search** in the AWS Management Console, type **Amplify**, and select it from the results. Once in the Amplify console, click on **Deploy an App** to begin.



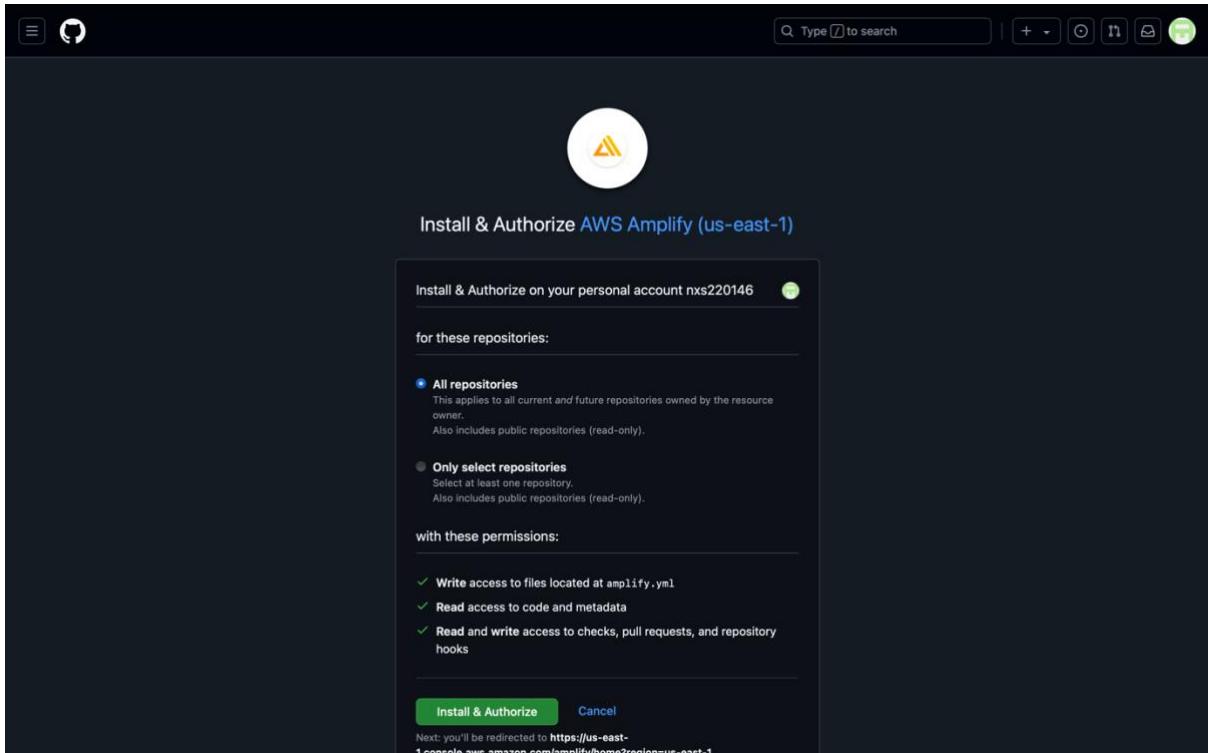
Choose **GitHub** as your repository source and click **Next** to proceed.



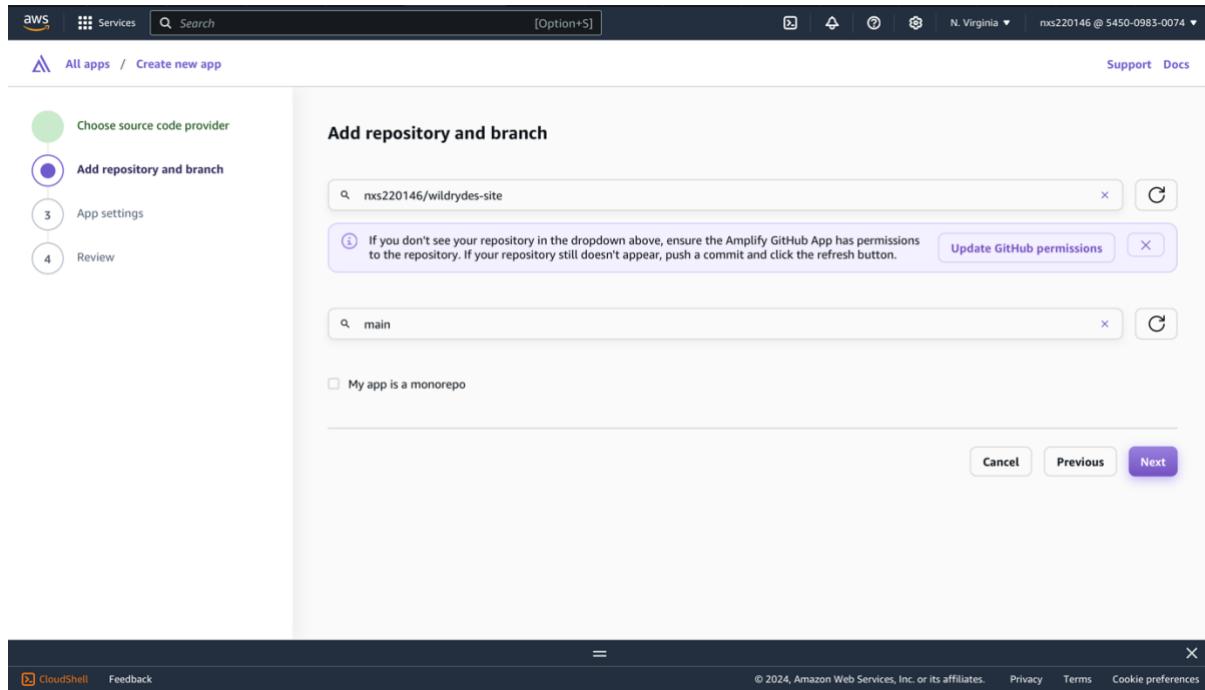
A prompt will appear asking you to authorize **AWS Amplify** to access your GitHub account. Follow the instructions to complete the authorization.



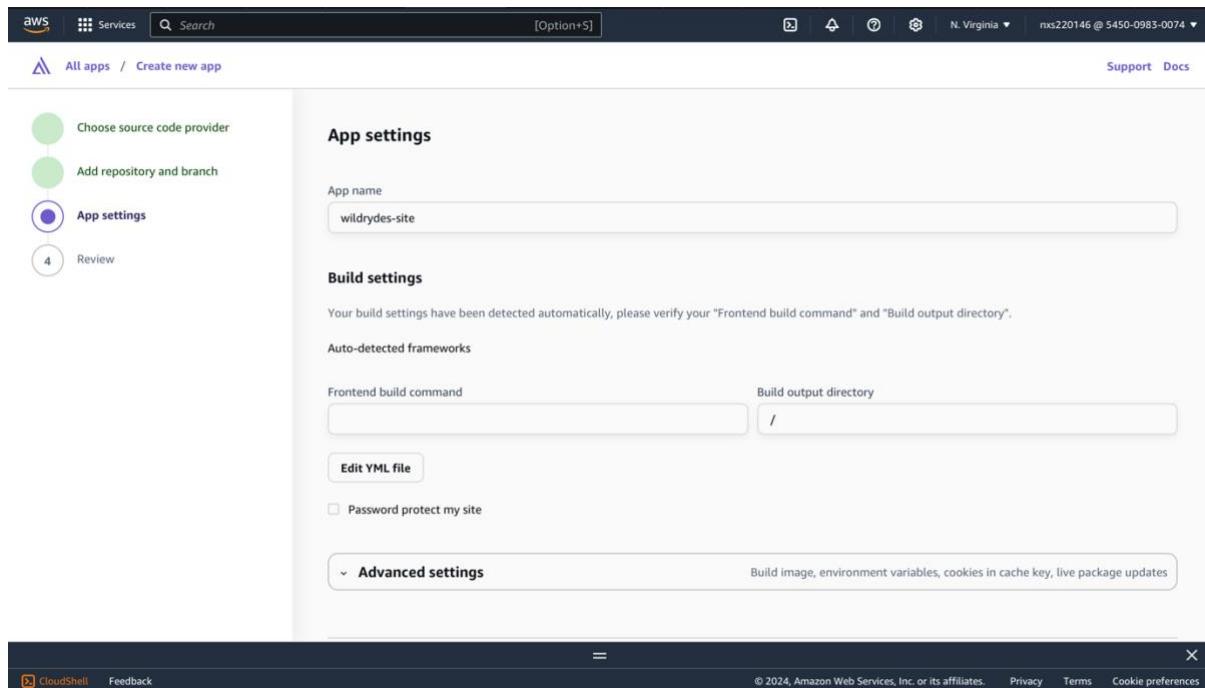
Click on the prompt. You can choose to authorize **AWS Amplify** for a specific repository or all repositories. In this case, I selected **All Repositories**, then click **Install and Authorize** to proceed.



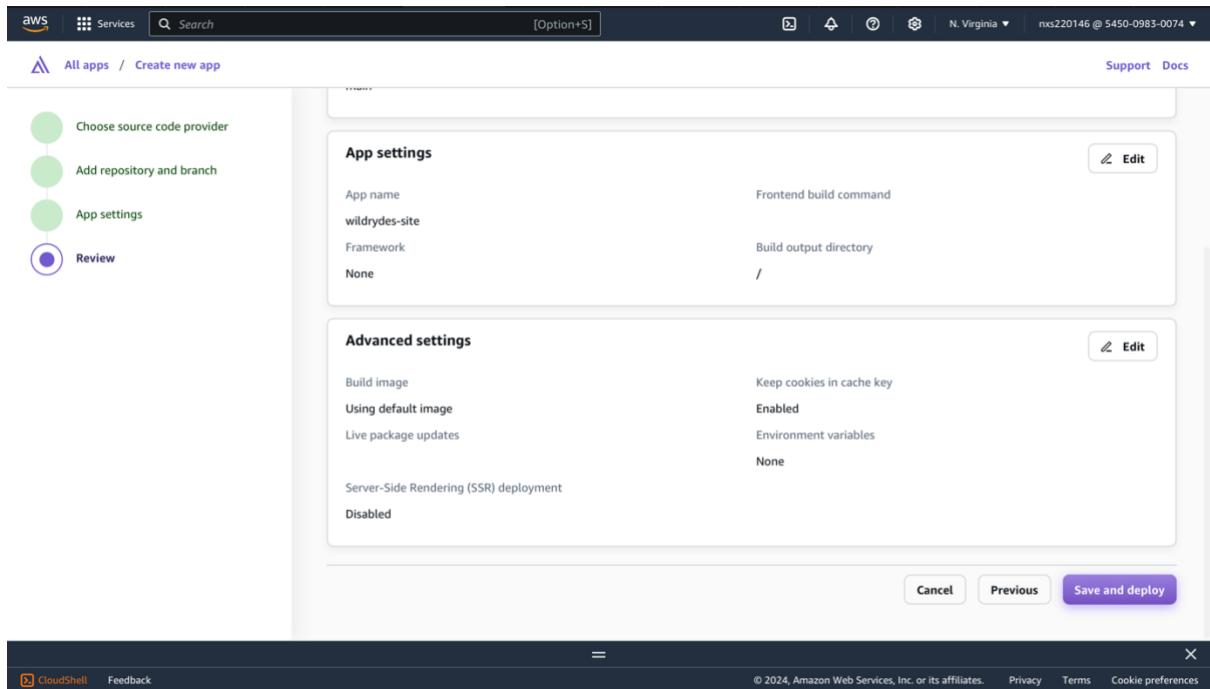
In the dropdown menu, select the desired repository. If the repository is not visible, click **Refresh** or **Update GitHub Permissions** to reload the list. Once selected, click **Next** to continue.



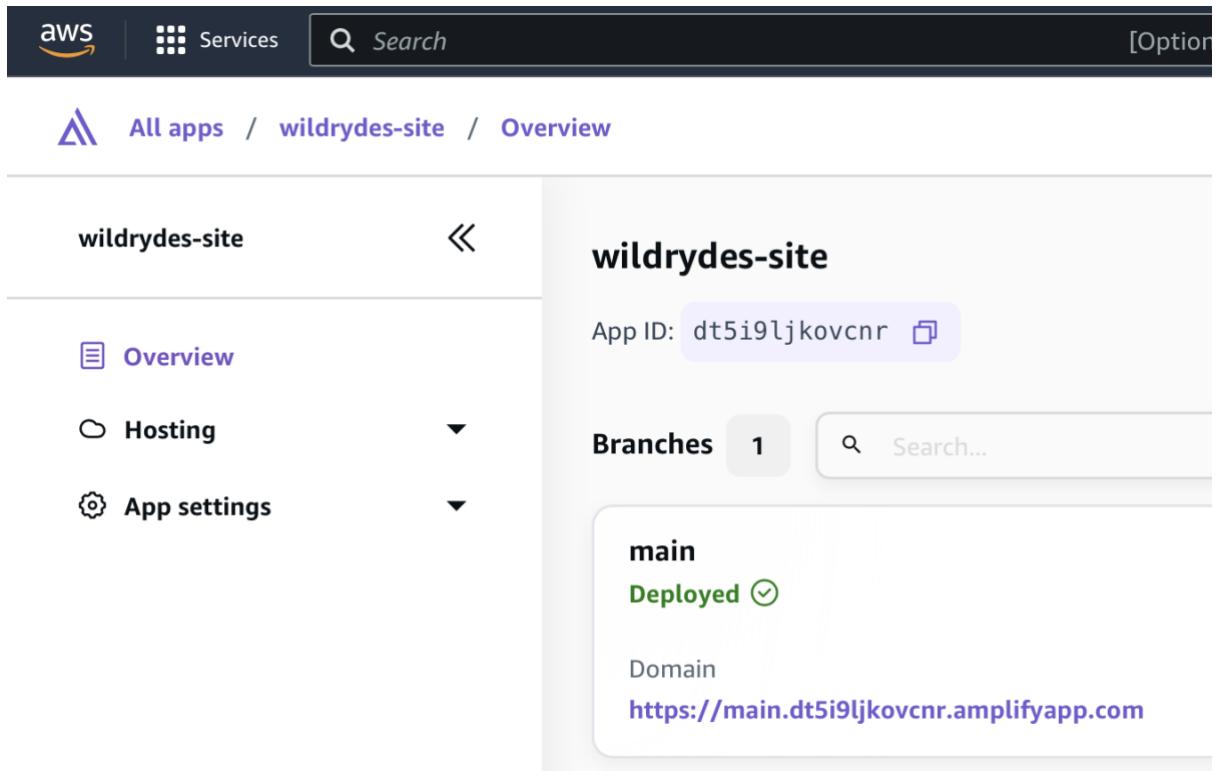
Enter **wildrydes-site** as the app name and click **Next** to proceed.



Click **Save and Deploy** to finalize the setup and start deploying your application.



After the deployment is complete, click on the **domain link** provided to view your application.



Note: The above screenshot is a deliverable. (It should capture the entire screen, with your **NETID** visible in the top-right corner.)

You will see the **WildRydes** website. Notice how we didn't need to manage any underlying infrastructure—AWS Amplify handled everything, allowing us to seamlessly deploy the application with minimal effort.



GIDDY UP!

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.

Go to your **GitHub repository** and click on **index.html**. Click on the **Edit this file** button to modify the file directly in GitHub.

The screenshot shows a GitHub repository page for 'nxs220146'. The repository has 1 branch and 0 tags. It contains several files: css, fonts, images, js, README.md, apply.html, faq.html, favicon.ico, index.html, investors.html, register.html, ride.html, robots.txt, signin.html, unicorns.html, and verify.html. All files were committed 13 minutes ago. The repository has no description, website, or topics provided. It has 0 stars, 1 watching, and 0 forks. There are no releases published, and no packages are published. The languages used are HTML (54.1%), CSS (28.0%), and JavaScript (17.9%). Suggested workflows include Grunt, which is described as building a NodeJS project with npm and grunt.

In the **index.html** file, locate **line 33**.

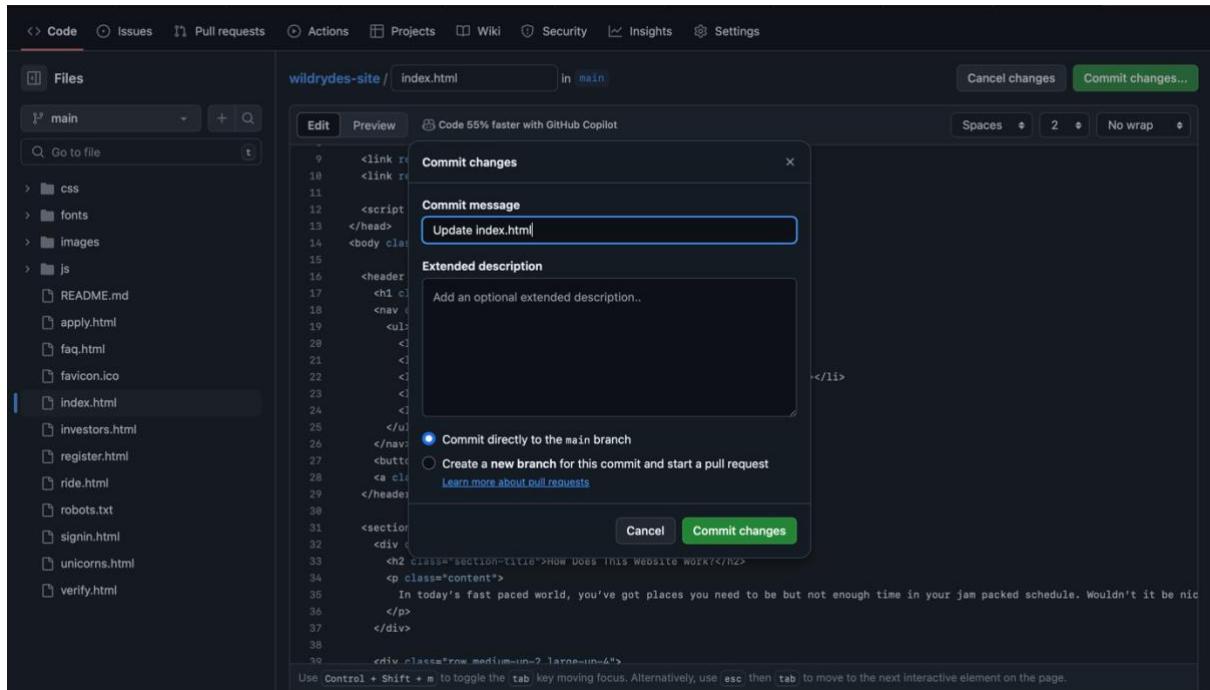
Replace the text "**How does this work**" with "**How does this Website work**".

The screenshot shows the GitHub Copilot interface for editing the 'index.html' file in the 'wildrydes-site' repository. The code editor displays the following HTML:

```
9   <link rel="stylesheet" href="css/font.css">
10  <link rel="stylesheet" href="css/main.css">
11
12  <script src="js/vendor/modernizr.js"></script>
13 </head>
14 <body class="page-home">
15
16  <header class="site-header">
17    <h1 class="title">Wild Rydes</h1>
18    <nav class="site-nav">
19      <ul>
20        <li><a href="#index.html">Home</a></li>
21        <li><a href="#unicorns.html">Meet the Unicorns</a></li>
22        <li><a href="#investors.html">Investors & Board of Directors</a></li>
23        <li><a href="#faq.html">FAQ</a></li>
24        <li><a href="#apply.html">Apply</a></li>
25      </ul>
26    </nav>
27    <button type="button" class="btn-menu"><span>Menu</span></button>
28    <a class="home-button" href="/register.html">Giddy Up!</a>
29  </header>
30
31  <section class="home-about">
32    <div class="row column large-9 xlarge-6">
33      <h2 class="section-title">How Does This Website Work?</h2>
34      <p class="content">
35        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 was a way to make things easier?
36      </p>
37    </div>
38
39    <div class="row medium-up-2 large-up-4">
40      <div class="column">
41        <div class="home-about-block">
```

The code editor includes GitHub Copilot features like code completion and navigation. The file list on the left shows all files in the repository, including css, fonts, images, js, README.md, apply.html, faq.html, favicon.ico, index.html, investors.html, register.html, ride.html, robots.txt, signin.html, unicorns.html, and verify.html.

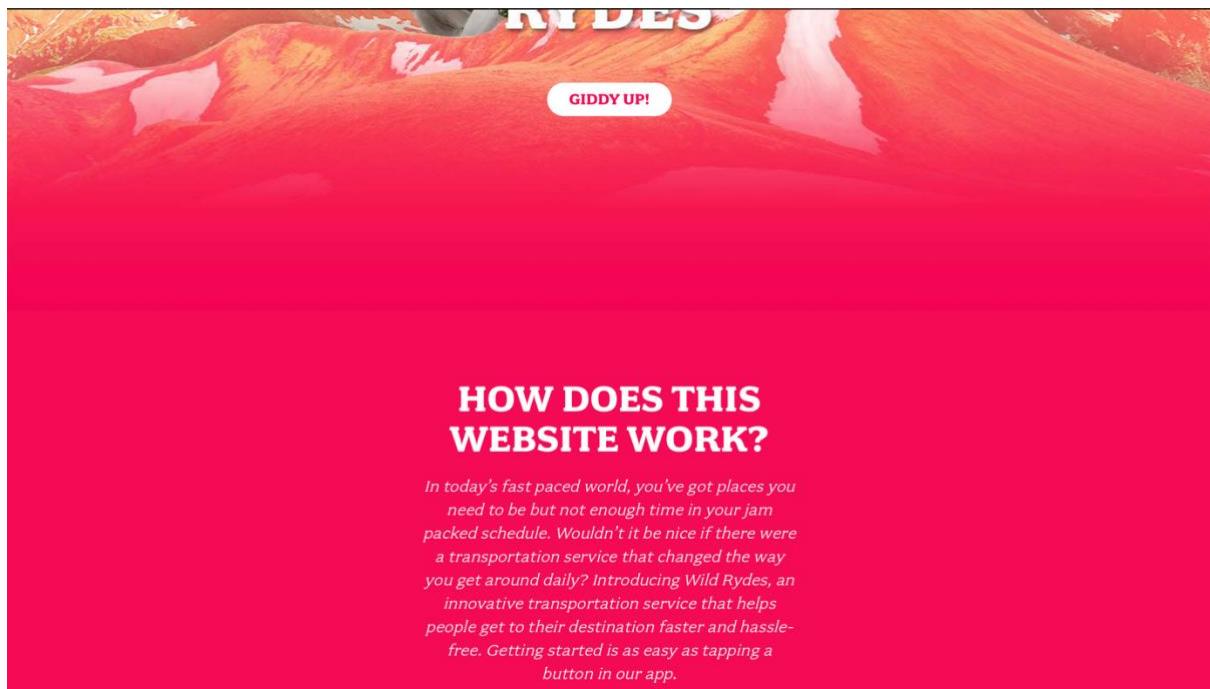
Save your changes by clicking **Commit changes**.



Switch back to your **AWS Amplify portal**, and you will see that the app is in the **Deploying** state, indicating that AWS Amplify is automatically processing your changes.

A screenshot of the AWS Amplify portal. At the top, there's a navigation bar with 'AWS', 'Services', 'Search', and user information. The main content area shows an app named 'wildrydes-site'. On the left, there's a sidebar with 'All apps / wildrydes-site / Overview' and sections for 'Overview', 'Hosting', and 'App settings'. The main panel shows the app ID 'dt5i9ljkovcnr' and a 'Branches' section with one branch named 'main'. The 'main' branch is shown with the status 'Deploying'. Below this, there's information about the domain 'https://main.dt5i9ljkovcnr.amplifyapp.com', 'Last deployment 0 minutes ago', and 'Last commit Update index.html / wildrydes-site:main'. At the bottom, there are links for 'CloudShell', 'Feedback', and copyright information.

Once the deployment is complete, refresh your website or click on the **domain link** again to view the updated changes.



HOW DOES THIS WEBSITE 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.

ii. Setting up Amazon Cognito for user authentication

In the AWS Management Console, use the search bar to type **Cognito** and select it from the results. Once in the Cognito dashboard, click on **Create User Pool** to start setting up a new user pool.

A screenshot of the AWS Cognito landing page. At the top, there is a navigation bar with the AWS logo, a services dropdown, a search bar, and account information. The main heading is "Amazon Cognito" with the subtitle "Secure identity and access management for apps". Below this, there is a brief description of what Cognito user pools do. To the right, there is a "Start from your business case" section with a "Create user pool" button, a "Pricing" section with a note about free usage, and a "Get started" button. At the bottom left, there is a "Benefits and features" section with two boxes: "Secure and scalable user directory" and "Social and enterprise identity federation".

Security, Identity & Compliance

Amazon Cognito

Secure identity and access management for apps

Amazon Cognito user pools let you add registration and sign-in to your apps. With Amazon Cognito identity pools, you can provide AWS credentials for access to your cloud resources.

Start from your business case

Add user directories to your app

Amazon Cognito user pools are a managed service that lets you add secure authentication and authorization to your apps, and can scale to support millions of users.

Create user pool

Pricing

Cognito identity pools are free of charge. With Cognito user pools, you pay only for what you use.

Learn more about pricing

Get started

Benefits and features

| | |
|--|--|
| Secure and scalable user directory | Social and enterprise identity federation |
| Amazon Cognito user pools are secure user directories that scale to millions of users. User pools bring capacity to your identity environment without having to build infrastructure up-front. | With Amazon Cognito, your users can sign in through social identity providers like Google, Facebook, and Amazon. They can also sign in through enterprise providers like ADFS and Okta with SAML 2.0 and |

For the sign-in options, select **Username** as the preferred method. Proceed with the default settings provided by Cognito. Then, click **Next** to continue.

The screenshot shows the 'Configure sign-in experience' step of the Cognito user pool creation wizard. On the left, a sidebar lists steps 1 through 6. Step 1 is expanded, showing 'Configure sign-in experience'. The main content area is titled 'Configure sign-in experience' with an 'Info' link. It explains that users can sign in with a user name and password or a third-party provider. The 'Authentication providers' section has 'Cognito user pool' selected. The 'Provider types' section has 'Federated identity providers' selected. The 'Cognito user pool sign-in options' section has 'User name' selected. Other options like 'Email' and 'Phone number' are available but not selected.

For the password policy, select **No MFA required**. While Cognito offers various customizable options here, we will stick with the default settings for simplicity. Click **Next** to proceed.

The screenshot shows the 'Configure sign-up experience' step of the Cognito user pool creation wizard. The 'MFA enforcement' section has 'No MFA' selected. The 'User account recovery' section has 'Enable self-service account recovery - Recommended' selected. The 'Delivery method for user account recovery messages' section has 'Email only' selected. Other delivery methods like 'SMS only', 'Email if available, otherwise SMS', 'SMS if available, otherwise email', and 'SMS if available, otherwise email, and allow a user to reset their password via SMS if they are also using it for MFA' are available but not selected.

Under **Configure Sign-Up Experience**, enable the option for **Self Registration** to allow users to sign up for your application. Then, proceed to the next step.

The screenshot shows the 'Configure sign-up experience' step in the 'Create user pool' wizard. Under 'Self-service sign-up', the 'Enable self-registration' checkbox is checked. A note below it states: 'If you activate user sign-up in your user pool, anyone on the internet can sign up for an account and sign in to your apps. Don't enable self-registration in your user pool until you want to open your app to public sign-up.' Below this, there's a section for 'Attribute verification and user account confirmation'.

The screenshot shows the 'Active attribute values when an update is pending' section, where 'Email address' is selected. It also shows the 'Required attributes' section, which lists 'email' as a required attribute. A note says: 'Required attributes can't be changed once this user pool has been created.' At the bottom, there are 'Cancel', 'Previous', and 'Next' buttons, with 'Next' highlighted.

Select the option to send emails using **Cognito** instead of the **Simple Email Service (SES)**. Toggle this setting to enable it, and then proceed to the next step.

The screenshot shows the 'Create user pool' wizard at Step 4: 'Configure message delivery'. On the left sidebar, other steps like 'Configure sign-in experience' and 'Integrate your app' are visible. The main area is titled 'Configure message delivery' and contains an 'Email' section. It shows two options for email provider: 'Send email with Amazon SES - Recommended' (disabled) and 'Send email with Cognito' (selected). A note states: 'You must have configured a verified sender with [Amazon SES](#) to use the SES feature. [Learn more](#)'.

For the **User Pool Name**, enter **wildrydes**.

The screenshot shows the 'Create user pool' wizard at Step 5: 'Integrate your app'. The sidebar shows previous steps. The main area is titled 'Integrate your app' and contains a 'User pool name' section where 'wildrydes' is entered. A note says: 'User pool names are limited to 128 characters or less. Names may only contain alphanumeric characters, spaces, and the following special characters: + = _ @ -'. Below this is a warning: '⚠ Your user pool name can't be changed once this user pool is created.' Further down are sections for 'Hosted authentication pages' (with a checkbox for 'Use the Cognito Hosted UI') and 'Initial app client'.

Provide a name for your initial app client. For the **Client Name**, enter **wildrydeswebapp**.

Initial app client

Configure an app client. App clients are single-app platforms in your user pool that have permissions to call unauthenticated API operations. A user pool can have multiple app clients.

App type [Info](#)

Select an app type and we will automatically populate common default settings. You can add additional app clients after the user pool is created.

Public client
A native, browser or mobile-device app. Cognito API requests are made from user systems that are not trusted with a client secret.

Confidential client
A server-side application that can securely store a client secret. Cognito API requests are made from a central server.

Other
A custom app. Choose your own grant, auth flow, and client-secret settings.

App client name [Info](#)
Enter a friendly name for your app client.

wildrydeswebapp

App client names are limited to 128 characters or less. Names may only contain alphanumeric characters, spaces, and the following special characters: +, -, @ -

Client secret [Info](#)
Choose whether your app client will have a client secret. Client secrets are used by the server-side component of an app to authorize API requests. Using a client secret can prevent a third party from impersonating your client.

Generate a client secret
 Don't generate a client secret

⚠️ You cannot change or remove a client secret after you allow Amazon Cognito to generate it for your app client.

[Advanced app client settings](#)

Click on next and create user pool.

| Attribute | Read | Write |
|-----------------------|-------------------------------------|-------------------------------------|
| middle_name | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| name | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| nickname | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| phone_number | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| phone_number_verified | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| picture | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| preferred_username | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| profile | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| updated_at | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| website | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |
| zoneinfo | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> |

Tags (0)

| Key | Value |
|---------------|-------|
| No tags found | |

Cancel [Previous](#) [Create user pool](#)

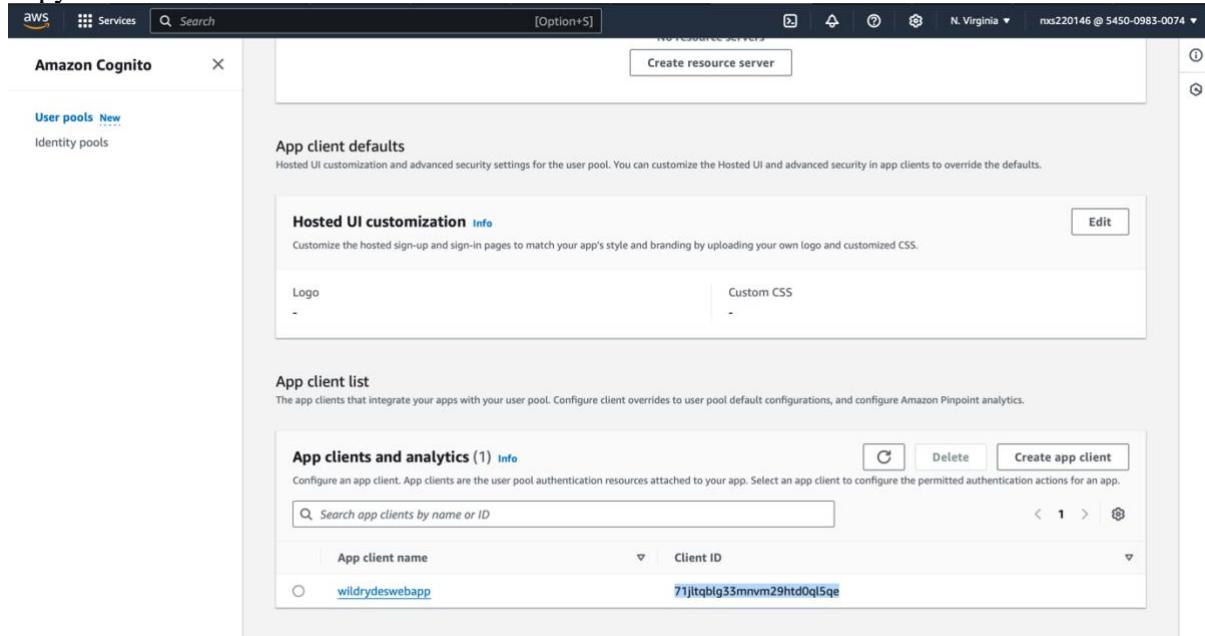
The screenshot shows the AWS Cognito User Pools dashboard. At the top, a green banner displays the message "User pool 'wildrydes' has been created successfully." Below this, a blue header bar contains the text "New advanced security features" and a link to "Amazon Verified Permissions". The main content area shows a table titled "User pools (1) Info" with one entry: "wildrydes" (User pool ID: us-east-1_TR665pOCD). A red arrow points from the text "Note: The above screenshot is a deliverable" to this table.

Note: The above screenshot is a deliverable

Open your newly created **User Pool** from the Cognito dashboard. Locate the **User Pool ID** and copy it to a safe place, such as a notepad or a secure document, for future reference.

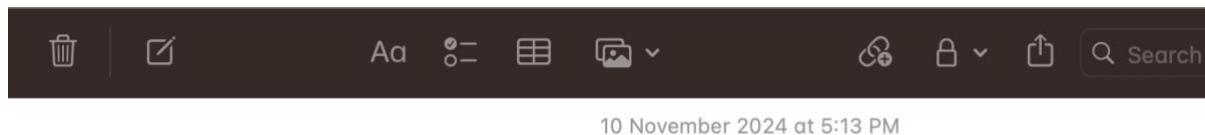
The screenshot shows the "wildrydes" User Pool details page. It includes sections for "User pool overview" (with fields like User pool name, User pool ID, ARN, Token signing key URL, Created time, Last updated time, Estimated number of users, and Advanced security status), "Getting started", and "Configuration for all app clients". A red arrow points from the text "Note: The above screenshot is a deliverable" to the User pool ID field in the overview section.

In your **User Pool**, go to the **App Integration** section. Scroll down to find the **Client ID**, and copy it to a secure location for future use.



The screenshot shows the AWS Cognito User Pools console. In the left sidebar, 'User pools' is selected. The main area displays the 'App client defaults' section, which includes 'Hosted UI customization' and 'Custom CSS' options. Below this is the 'App client list' section, which shows a table with one row. The table columns are 'App client name' and 'Client ID'. The single entry is 'wildrydeswebapp' with the Client ID '71jltqblg33mnvm29htd0ql5qe'. There are 'Edit', 'Delete', and 'Create app client' buttons at the top of the list table.

I have saved the **User Pool ID** and **Client ID** in my Notes app for easy access.



The screenshot shows a dark-themed notes application interface. At the top, there are various icons for file operations like delete, edit, and search. The main content area contains the following text:

10 November 2024 at 5:13 PM

COGNITO USER POOL ID: us-east-1_TR665pOCD

CLIENT ID: 71jltqblg33mnvm29htd0ql5qe

Updating the app configuration file to use the Amazon Cognito user pool

Go to your **GitHub repository** and open the **config.js** file. Click on the **Edit this file** button to make changes directly in the file.

```

window._config = {
  cognito: {
    userPoolId: '', // e.g. us-east-2_uXboG5pAb
    userPoolClientId: '', // e.g. 25ddkmj4v6hfsfvruhpf17n4hv
    region: '' // e.g. us-east-2
  },
  api: {
    invokeUrl: '' // e.g. https://rc7nyt4tql.execute-api.us-west-2.amazonaws.com/prod',
  }
};

```

In the **config.js** file, paste your **User Pool ID** and **Client ID** between the quotes. Additionally, check which AWS region you are currently working in, and enter the corresponding region code in the appropriate field.

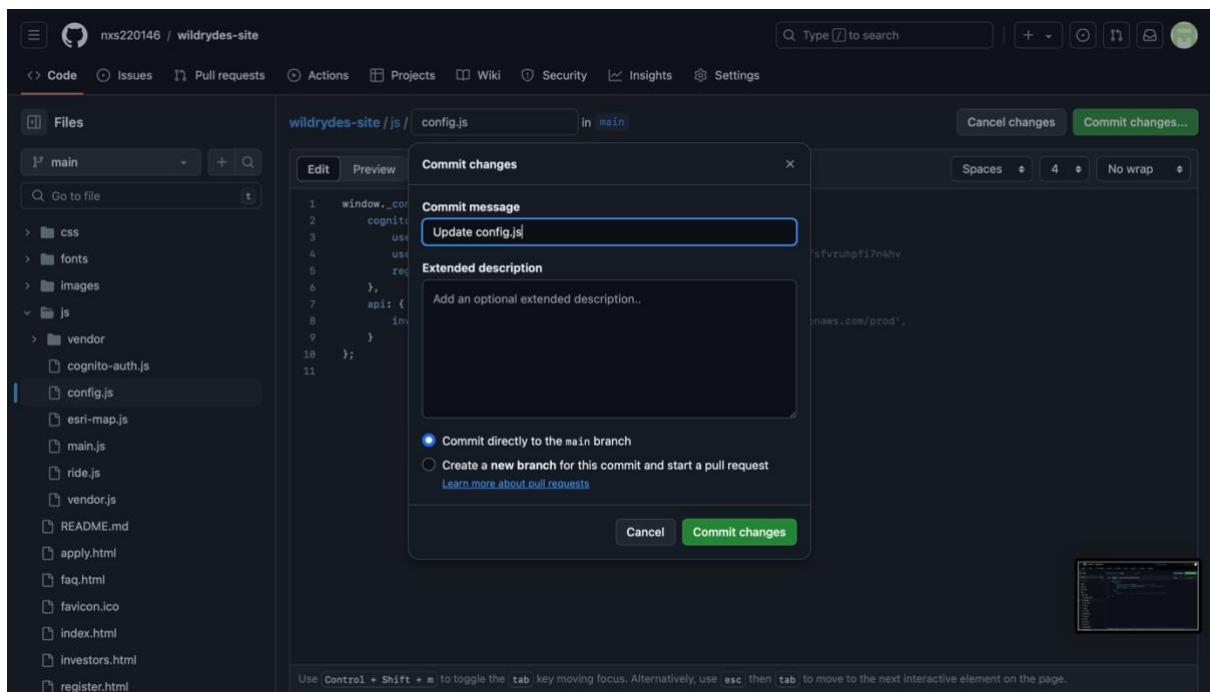
You can find your AWS region by going to the AWS Management Console. Look at the top-right corner and click the down arrow next to the region name. It will display your current region. For example, if you are in **North Virginia**, the region code will be **us-east-1**. Use this code when entering your region in the **config.js** file.

| Region | Code |
|---------------------------|----------------|
| US East (N. Virginia) | us-east-1 |
| US East (Ohio) | us-east-2 |
| US West (N. California) | us-west-1 |
| US West (Oregon) | us-west-2 |
| Asia Pacific (Mumbai) | ap-south-1 |
| Asia Pacific (Osaka) | ap-northeast-3 |
| Asia Pacific (Seoul) | ap-northeast-2 |
| Asia Pacific (Singapore) | ap-southeast-1 |
| Asia Pacific (Sydney) | ap-southeast-2 |
| Asia Pacific (Tokyo) | ap-northeast-1 |
| Canada (Central) | ca-central-1 |
| Europe (Frankfurt) | eu-central-1 |
| Europe (Ireland) | eu-west-1 |
| Europe (London) | eu-west-2 |
| Europe (Paris) | eu-west-3 |
| Europe (Stockholm) | eu-north-1 |
| South America (São Paulo) | sa-east-1 |

A screenshot of the GitHub code editor interface. The repository is 'nxs220146 / wildrydes-site'. The file 'config.js' is open in the main editor area. The code contains configuration for AWS services like Cognito and Lambda. The GitHub Copilot icon is visible in the top right of the editor. The left sidebar shows the project structure with files like 'main.js', 'js', and 'index.html'. The top navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings.

```
1 window._config = {
2   cognito: {
3     userPoolId: 'us-east-1_TR665p0CD', // e.g. us-east-2_uXboG5pAb
4     userPoolClientId: '71j1tqb1g3mnm29htd0q15qe', // e.g. 25ddkmj4v6hfsfvruhpfi7n4hv
5     region: 'us-east-1' // e.g. us-east-2
6   },
7   api: {
8     invokeUrl: '' // e.g. https://rc7nyt4tql.execute-api.us-west-2.amazonaws.com/prod',
9   }
10 };
11
```

Click on commit changes.

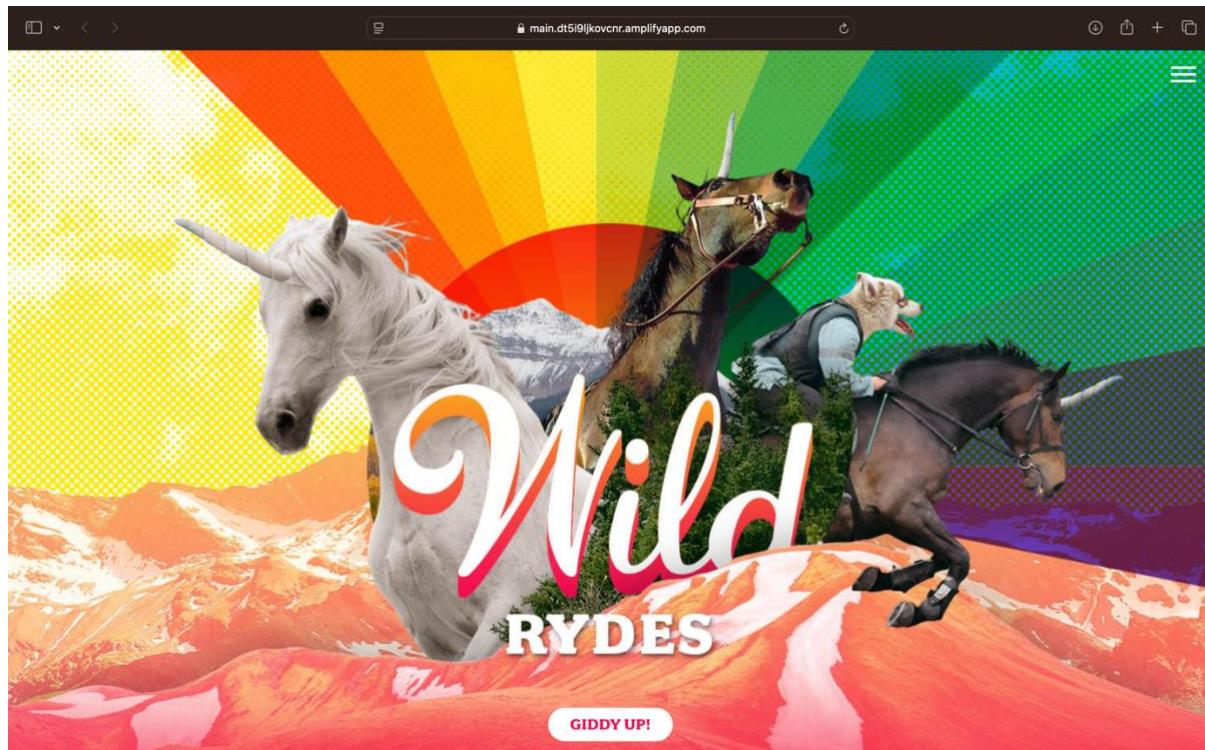


As you've seen before, **AWS Amplify** automatically detects changes in your code. Once you commit the changes, Amplify will redeploy your application with the updated configurations.

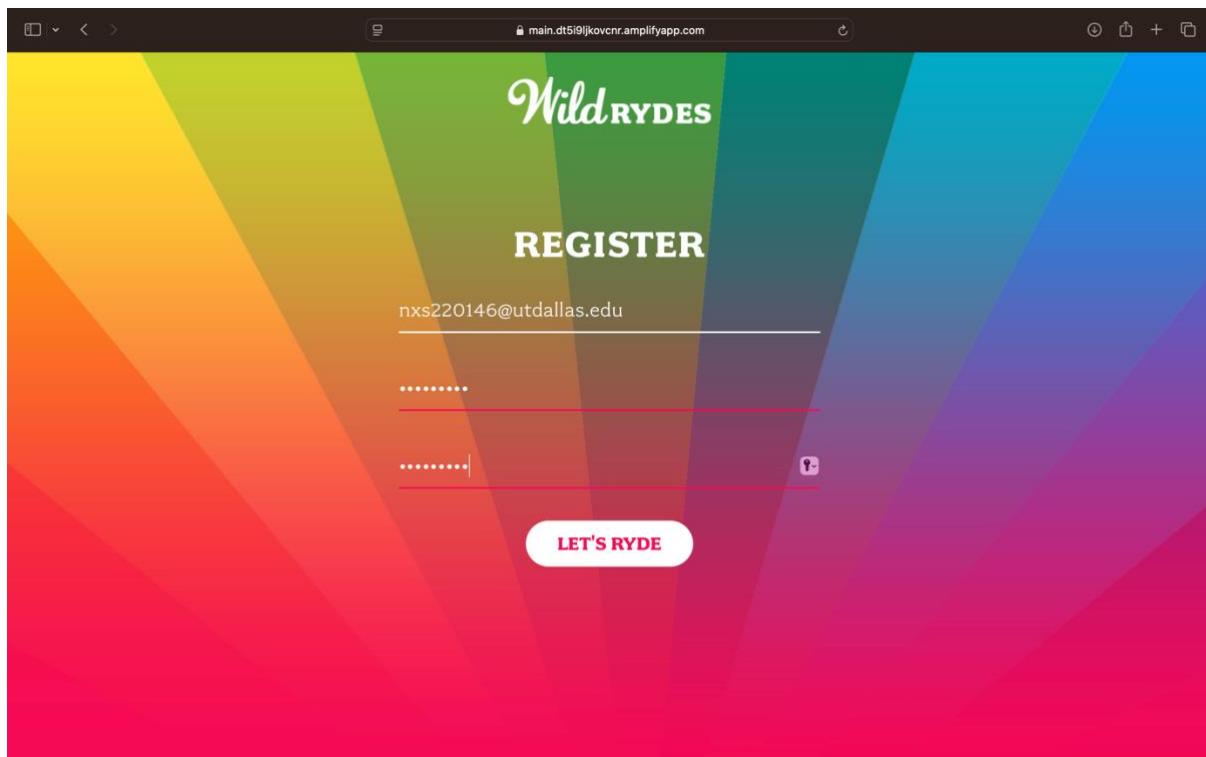
The screenshot shows the AWS Amplify console interface. On the left, there's a sidebar with navigation links: Deployments, Authentication, Data, Storage, Functions, and UI Library. The main area is titled 'Deployments' and shows 'Deployment 3' which is 'Deployed'. It provides details such as 'Started at 11/10/2024, 5:21 PM', 'Build duration 47 seconds', 'Domain https://main.dt5ijlkovcnr.amplifyapp.com', 'Repository wildrydes-site:main', and 'Last commit Update config.js'. Below this, there are 'Build' and 'Deploy' buttons with timing information: '39 seconds' and '7 seconds'. At the bottom, there are tabs for 'Deployment history' (which shows three entries) and 'Deployed backend resources'. The deployment history table has columns for Name, Status, Build duration, Commit message, and Started at. The first entry is 'Deployment 3' with status 'Deployed', build duration '47 seconds', commit message 'Update config.js', and started at '11/10/2024, 5:21 PM'. The second entry is 'Deployment 2' with status 'Deployed', build duration '50 seconds', commit message 'Update index.html', and started at '11/10/2024, 5:06 PM'. The third entry is 'Deployment 1' with status 'Deployed', build duration '50 seconds', commit message 'Auto-build', and started at '11/10/2024, 5:02 PM'. The footer includes links for CloudShell, Feedback, and various AWS services, along with copyright information and privacy terms.

Click on the **domain URL** or refresh your website to see the updated changes reflected in your application.

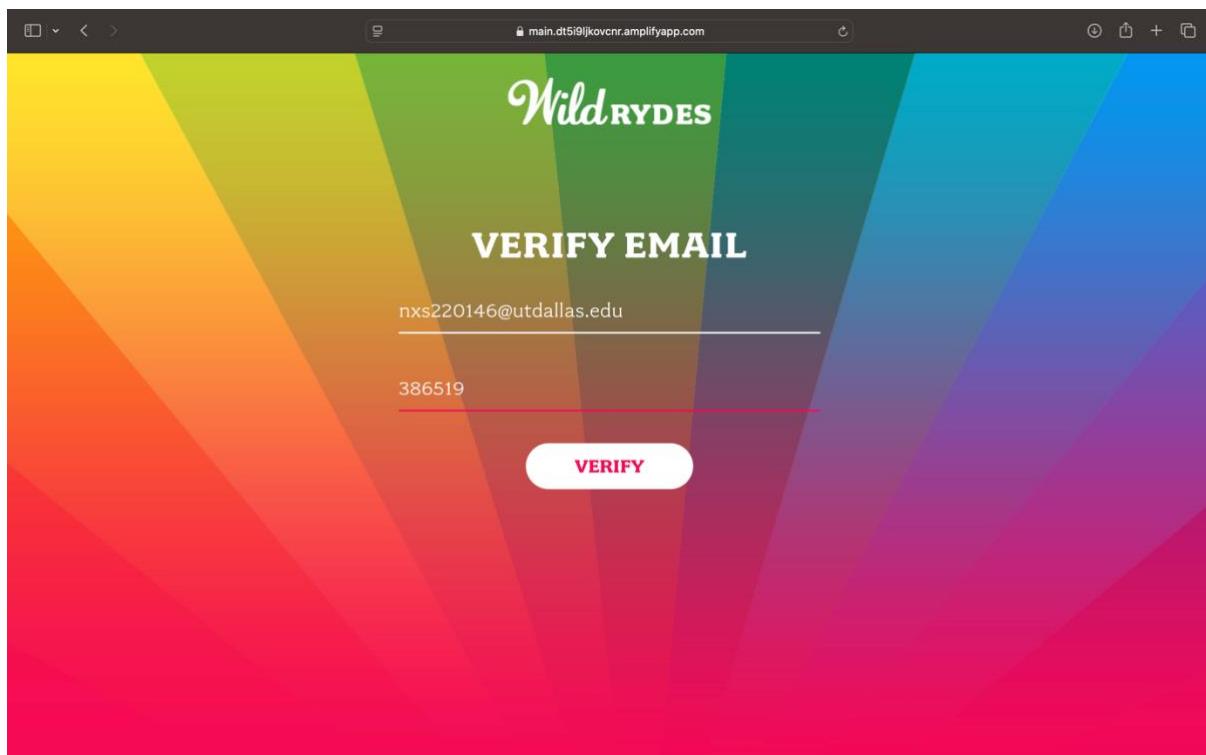
Testing Cognito integration by doing user registration and login



Click on **Giddy Up!** to register for an account. Make sure to use a valid email address, as a verification code will be sent to this email for confirmation.



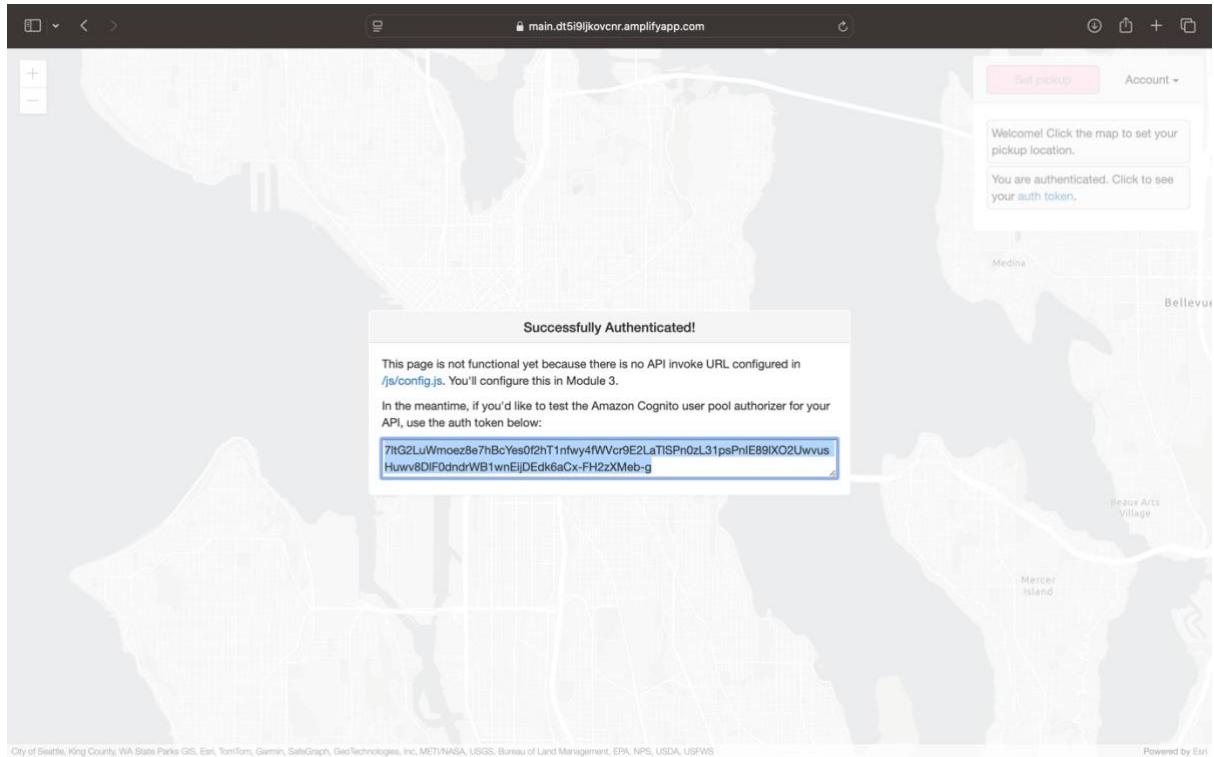
After completing the registration, you'll be prompted to verify your account. Check your email for the verification code sent by AWS Cognito, and enter the code to complete the verification process.



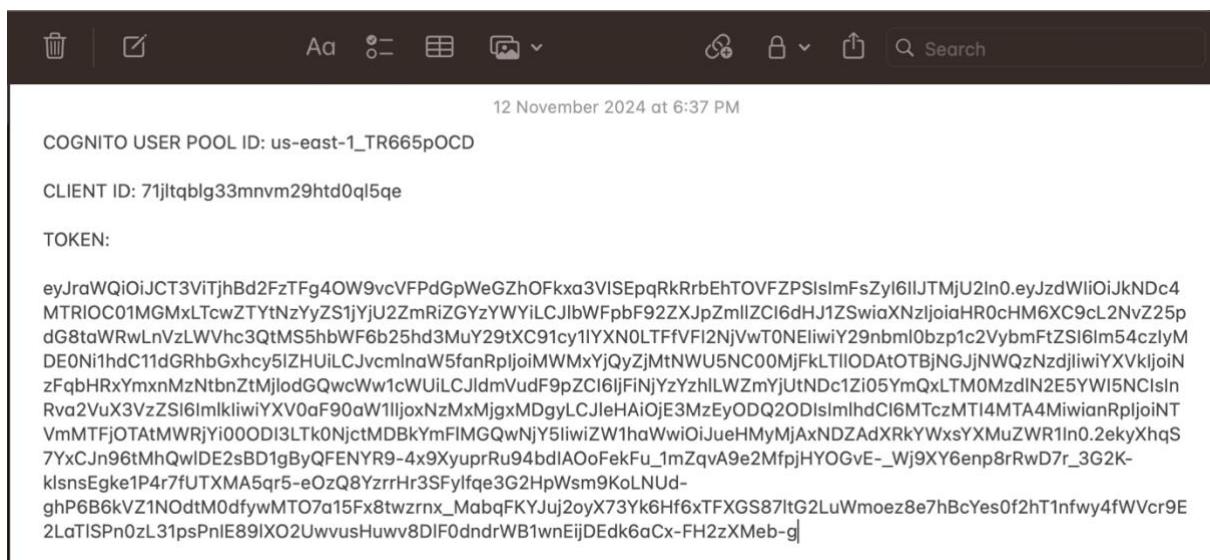
After verifying your account, proceed to **sign in** using your registered email address and password.

After successfully signing in, you might see a page you weren't expecting. This page will display a **token**. You'll need this token for testing your API later.

Select the entire token, copy it, and save it securely—for example, paste it into your **Notes** app or a text editor like Notepad. Make sure you don't lose it, as it will be essential for the next steps.



It would look something like this. Make sure to copy the entire token, as it may be long and span multiple lines.



iii. Creating a new DynamoDB table

In the AWS Management Console, search for **DynamoDB** and select it. Once you're in the DynamoDB dashboard, click on **Create Table** to start setting up a new table.

The screenshot shows the AWS DynamoDB service dashboard. On the left, there's a sidebar with links like Dashboard, Tables (which is selected and highlighted in blue), Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Integrations, Reserved capacity, and Settings. Below that is a section for DAX with Clusters, Subnet groups, Parameter groups, and Events. The main content area is titled "Tables (0) Info" and shows a table header with columns: Name, Status, Partition key, Sort key, Indexes, Replication Regions, Deletion protection, Favorite, and Read. A message below the table says "You have no tables in this account in this AWS Region." At the bottom of the table area is a large orange "Create table" button. The top right corner of the main content area has a small circular icon with a question mark. The bottom of the screen includes standard AWS navigation links for CloudShell, Feedback, and legal notices like © 2024, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

In the **Create Table** screen:

1. Set the **Table Name** to **Rides**.
2. For the **Partition Key**, enter **RideId**

Ensure the capitalization matches exactly - Capital R, small ide, Capital I, small d.

3. Leave all other settings as default.
4. Click **Create Table** to finalize the setup.

S | Services | Search [Option+S] | N. Virginia | nxs220146 @ 5450-0983-0074 | ⓘ | ⓘ

DynamoDB > Tables > Create table

Create table

Table details ⓘ

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.

| | |
|--------|--------|
| Rideld | String |
|--------|--------|

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.

| | |
|-------------------------|--------|
| Enter the sort key name | String |
|-------------------------|--------|

1 to 255 characters and case sensitive.

Table settings

Default settings The fastest way to create a table. You can modify these settings later.

Customize settings Use these advanced features to create DynamoDB tables.

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

S | Services | Search [Option+S] | N. Virginia | nxs220146 @ 5450-0983-0074 | ⓘ | ⓘ

DynamoDB

The Rides table was created successfully.

DynamoDB > Tables

Tables (1) Info

| Name | Status | Partition key | Sort key | Indexes | Replication Regions | Deletion protection | Favorite | Re |
|-------|--------|---------------|----------|---------|---------------------|---------------------|----------|----|
| Rides | Active | Rideld (\$) | - | 0 | 0 | Off | ☆ | Pr |

Find tables Any tag key Any tag value 1 match ⌂ Actions Delete Create table

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

Open the **Rides** table in the DynamoDB console. Locate the **Amazon Resource Name (ARN)** for the table, copy it, and paste it into your **Notes** or a secure text editor for future reference.

The screenshot shows the AWS DynamoDB console with the 'Rides' table selected. The 'General information' section displays the following details:

- Partition key:** RidId (String)
- Sort key:** -
- Capacity mode:** Provisioned
- Table status:** Active
- Alarms:** No active alarms
- Point-in-time recovery (PITR):** Off
- Resource-based policy:** Not active
- Table class:** DynamoDB Standard
- Indexes:** 0 globals, 0 locals
- DynamoDB stream:** Off
- Time to Live (TTL):** Off
- Replication Regions:** 0 Regions
- Encryption:** Owned by Amazon
- Date created:** November 10, 2024, 17:45:56 (UTC-06:00)
- Deletion protection:** Off
- Integrations:** None

A green message box at the bottom left indicates that the ARN has been copied: **ARN copied**. Below it, the full ARN is shown: `arn:aws:dynamodb:us-east-1:545009830074:table/Rides`.

Note: The above screenshot is a deliverable

Creating an IAM Role for Lambda Execution with DynamoDB PutItem Permissions

In the AWS Management Console, navigate to **IAM** and click on **Create Role** to start setting up a new role.

The screenshot shows the AWS IAM console with the 'Create role' wizard open. The first step, 'Select trusted entity', is displayed. The 'Trusted entity type' section contains the following options:

- AWS service** (selected): Allows AWS services like EC2, Lambda, or others to perform actions in this account.
- AWS account**: Allows 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**: Allows 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.

Below the entity type selection, there is a 'Use case' section with the following text: "Allow an AWS service like EC2, Lambda, or others to perform actions in this account." A dropdown menu labeled "Service or use case" is present, with the placeholder "Choose a service or use case".

For the **use case**, select **Lambda** as the trusted entity and click **Next** to proceed.

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Select trusted entity

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

Lambda

Choose a use case for the specified service.

Use case

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

Cancel Next

In the **Permissions policies** section, search for **AWSLambdaBasicExecutionRole**. Select it, and then click **Next** to continue.

IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions

Permissions policies (1/965) Info

Choose one or more policies to attach to your new role.

Filter by Type

lambdabasic

| Policy name | Type | Description |
|---|------------------|---|
| <input checked="" type="checkbox"/> AWSLambdaBasicExecutionRole | AWS managed | Provides write permissions to CloudW... |
| <input type="checkbox"/> AWSLambdaBasicExecutionRole-4d... | Customer managed | - |
| <input type="checkbox"/> AWSLambdaBasicExecutionRole-7f... | Customer managed | - |

Set permissions boundary - optional

Cancel Previous Next

In the **Role Name** field, enter **WildRydesLambda**, and then click **Create Role** to finalize the setup.

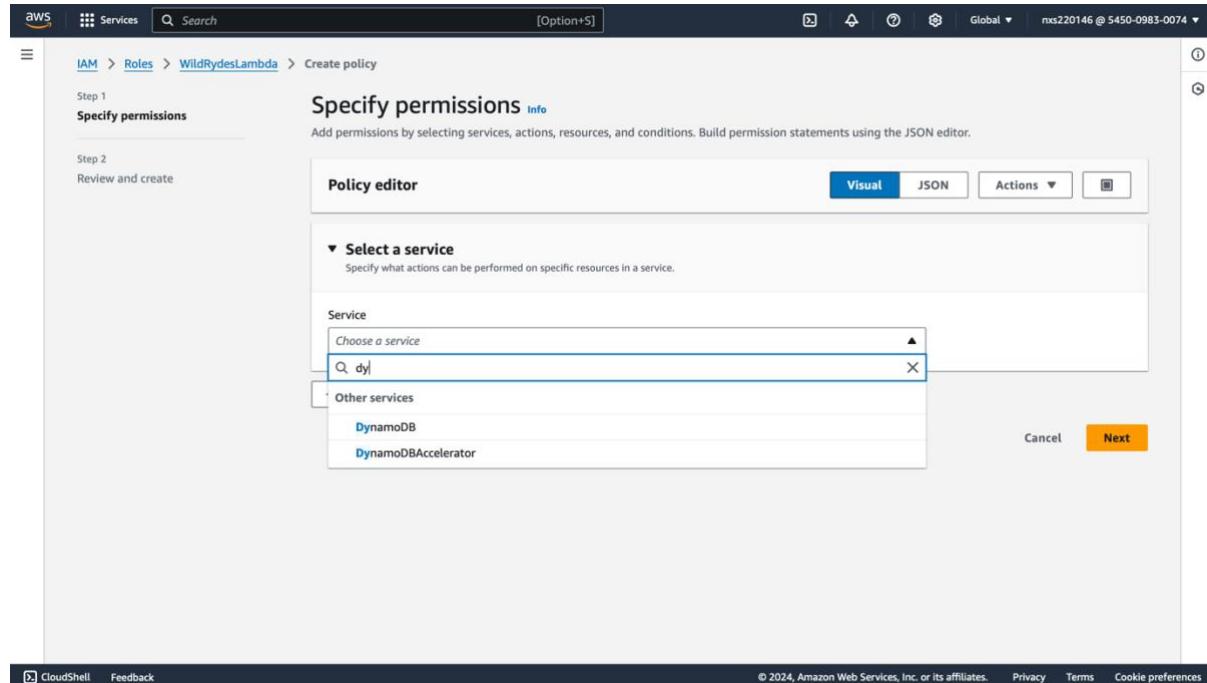
The screenshot shows the 'Name, review, and create' step of the IAM role creation wizard. In the 'Role details' section, the 'Role name' is set to 'WildRydesLambda'. The 'Description' field contains the text 'Allows Lambda functions to call AWS services on your behalf.' Below this, there is a code editor showing the JSON trust policy:

```
1 - {  
2 -   "Version": "2012-10-17",  
3 -   "Statement": [  
4 -     {  
5 -       "Effect": "Allow",  
6 -       "Action": [  
7 -         "sts:AssumeRole"  
8 -       ],  
9 -       "Principal": "  
10 -      
```

Open the newly created **WildRydesLambda** role. Under the **Permissions** tab, click on **Add Permissions** and then select **Create Inline Policy** to define a custom policy for the role.

The screenshot shows the 'WildRydesLambda' role page in the IAM console. The 'Permissions' tab is selected. In the top right corner of the main content area, the 'Create inline policy' button is highlighted with a blue border.

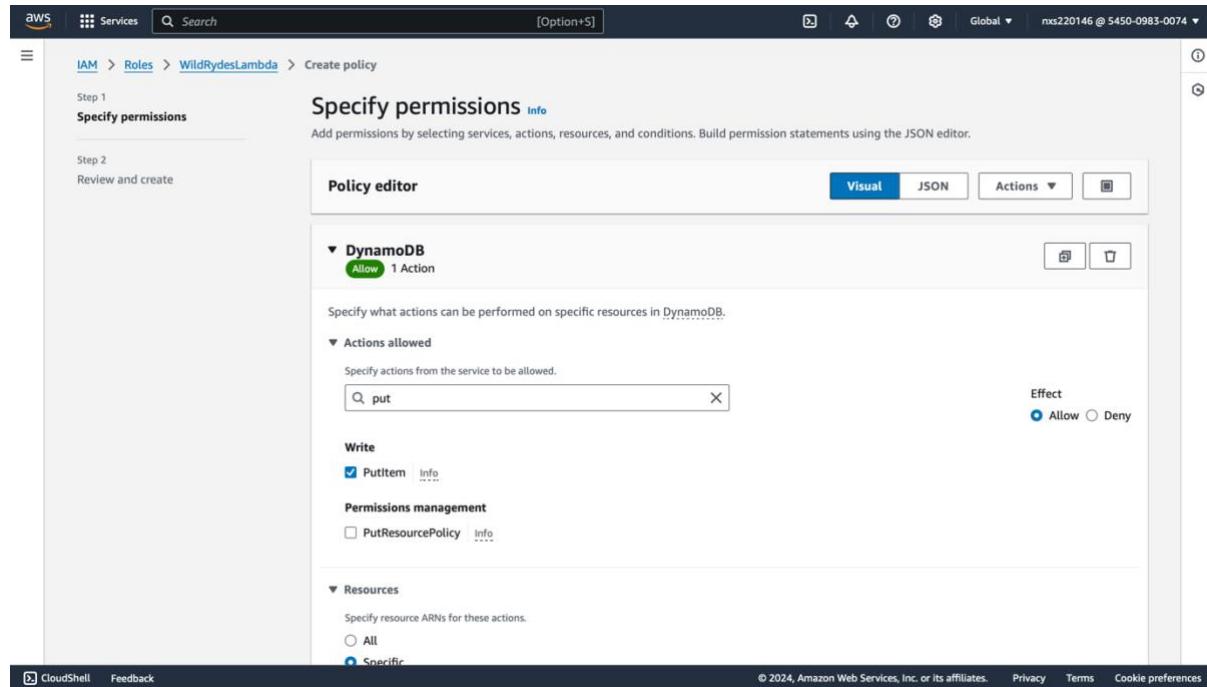
In the **Create Policy** screen, under the **Service** section, search for and select **DynamoDB** as the service.



The screenshot shows the AWS IAM 'Create policy' interface. The top navigation bar includes the AWS logo, 'Services' button, and a search bar. Below the navigation is a breadcrumb trail: 'IAM > Roles > WildRydesLambda > Create policy'. On the left, there are two tabs: 'Step 1 Specify permissions' (selected) and 'Step 2 Review and create'. The main area is titled 'Specify permissions' with a 'Info' link. It contains a note: 'Add permissions by selecting services, actions, resources, and conditions. Build permission statements using the JSON editor.' A 'Policy editor' panel has three tabs: 'Visual' (selected), 'JSON', and 'Actions'. Below the tabs is a section titled '▼ Select a service' with the sub-instruction: 'Specify what actions can be performed on specific resources in a service.' A 'Service' dropdown menu is open, showing 'Choose a service' at the top, followed by a search input field containing 'Q. dyl'. Below the search field, 'DynamoDB' is listed under 'Other services'. At the bottom right of the editor panel are 'Cancel' and 'Next' buttons. The bottom of the page includes standard AWS footer links: CloudShell, Feedback, © 2024, Amazon Web Services, Inc. or its affiliates., Privacy, Terms, and Cookie preferences.

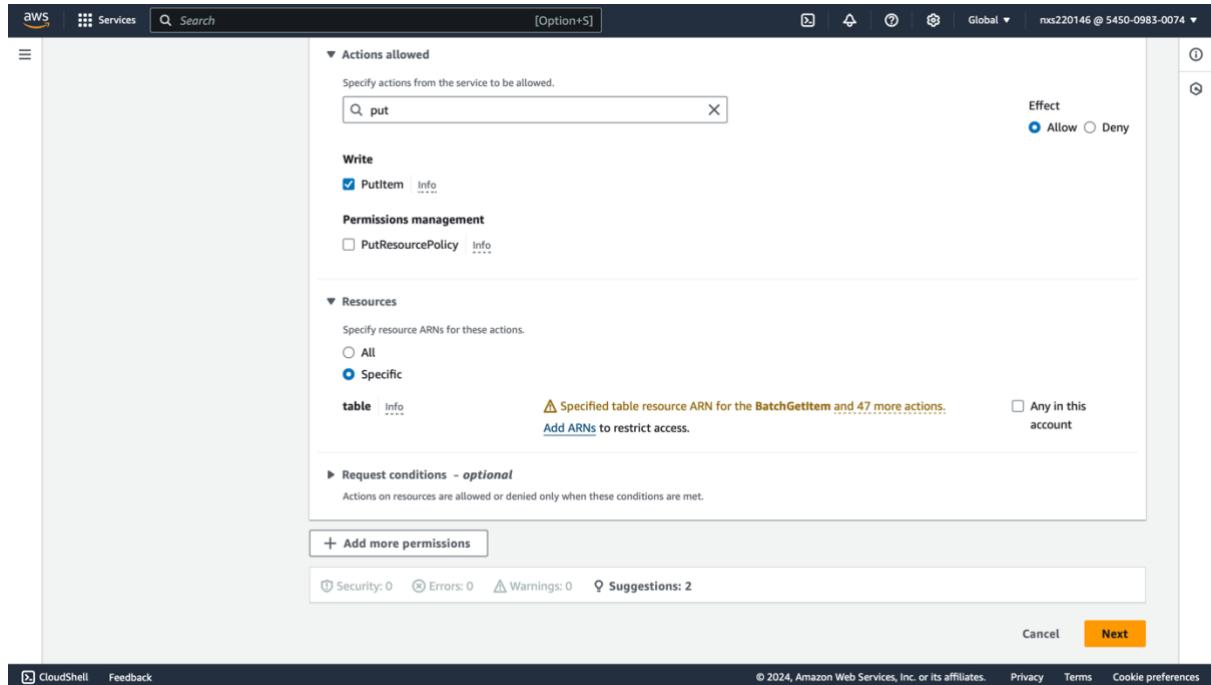
In the **Actions** section, search for **PutItem** and select it.

Under the **Resources** section, choose **Specific** to specify the DynamoDB table this policy will apply to.

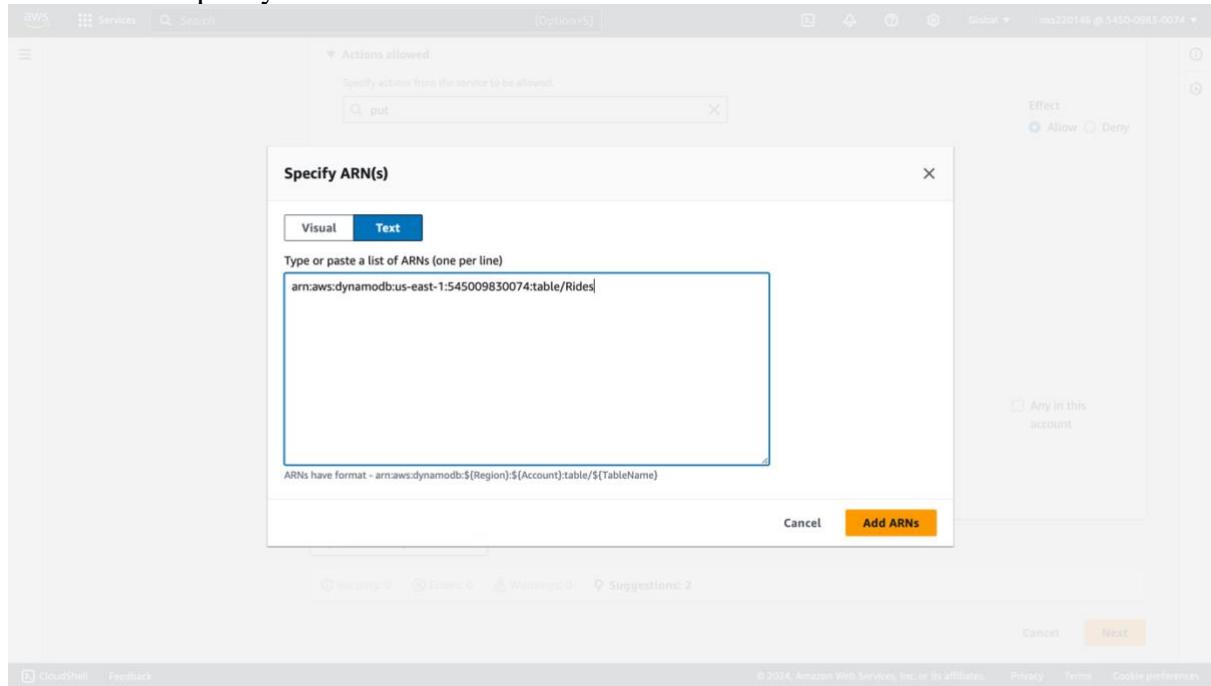


The screenshot shows the AWS IAM 'Create policy' interface, identical to the previous one but with different configuration details. The 'Service' dropdown now shows 'DynamoDB' selected. In the 'Actions allowed' section, 'Put' is selected under 'Write', and 'PutItem' is checked. In the 'Resources' section, 'Specific' is selected. The rest of the interface, including the 'Policy editor' panel and footer, remains the same.

Click on **Add ARN** under the **Resources** section. This ensures the Lambda function will have permission to perform the **PutItem** operation only on the specific DynamoDB table.



Switch to the **Text** view, and paste the ARN you previously copied from the DynamoDB **Rides** table into the specified field. Once done, click **Add ARN** to associate the table with this policy.



After adding the ARN, click **Next** to proceed to the review screen.

The screenshot shows the AWS IAM Policy Editor interface. In the 'Actions allowed' section, 'PutItem' is selected under 'Write'. In the 'Resources' section, the ARN 'arn:aws:dynamodb:us-east-1:545009830074:table/Rides' is specified. The 'Effect' is set to 'Allow'. At the bottom right, there are 'Cancel' and 'Next' buttons, with 'Next' being highlighted.

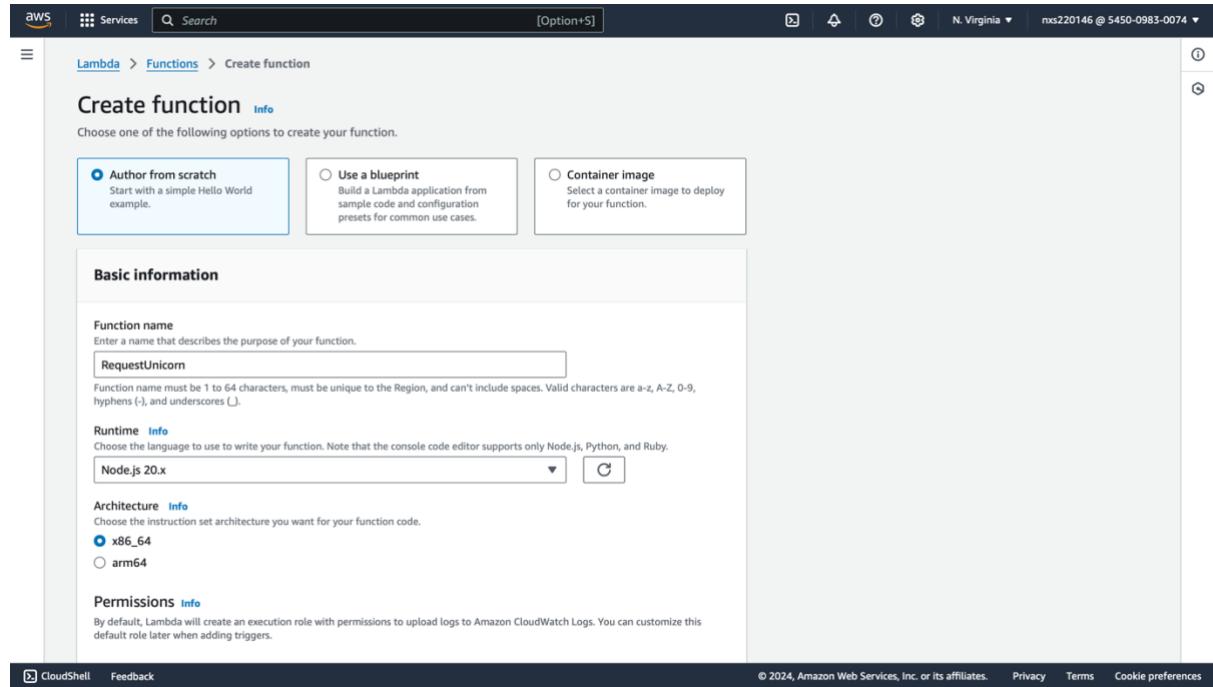
On the review screen, enter **DynamoDBWriteAccess** as the policy name. Then, click **Create Policy** to finalize and attach the policy to the role.

The screenshot shows the 'Review and create' step of the IAM policy creation wizard. The 'Policy details' section shows the policy name 'DynamoDBWriteAccess' entered. The 'Permissions defined in this policy' section lists 'Allow (1 of 427 services)' for 'DynamoDB' with the 'Limited: Write' access level and the resource 'region| string like |us-east-1, TableName| string like |Rides'. At the bottom right, there are 'Cancel', 'Previous', and 'Create policy' buttons, with 'Create policy' being highlighted.

iv. Creating a New Lambda Function for Unicorn Selection and DynamoDB Integration

In the AWS Management Console, search for **Lambda** and select it. Click **Create Function** and follow these steps:

1. Choose **Author from scratch**.
2. Enter the **Function Name** as **RequestUnicorn**.
3. Select **Runtime** as **Node.js 20.x**.



Expand the **Change default execution role** section.

1. Choose the **Use an existing role** option.
2. From the dropdown, select the **WildRydesLambda** IAM role you created earlier.
3. Click **Create Function** to finalize the setup.

The screenshot shows the AWS Lambda 'Create Function' configuration page. In the 'Change default execution role' section, the 'Use an existing role' option is selected, and the 'WildRydesLambda' role is chosen from the dropdown. Other options like 'Create a new role with basic Lambda permissions' and 'Create a new role from AWS policy templates' are also listed. The 'Create function' button is at the bottom right.

The screenshot shows the AWS Lambda function details page for 'RequestUnicorn'. A green success message at the top states: 'Successfully created the function RequestUnicorn. You can now change its code and configuration. To invoke your function with a test event, choose "Test".' The 'Function overview' section shows the function name 'RequestUnicorn', a diagram icon, and a 'Layers (0)' button. The 'Code source' section has an 'Upload from' button. The bottom navigation bar includes 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions' tabs.

To add the Lambda function code:

Go to your **GitHub repository** and scroll down to the **README** section.

The screenshot shows a GitHub repository page for 'wildrydes-site'. The repository has 1 branch, 0 tags, and 3 commits. The README file contains the following content:

```
AWS Project - Build a Full End-to-End Web Application  
with 7 Services | Step-by-Step Tutorial  
  
This repo contains the code files used in this YouTube video.  
  
TL;DR  
We're creating a web application for a unicorn ride-sharing service called Wild Rydes (from the original Amazon workshop). The app uses IAM, Amplify, Cognito, Lambda, API Gateway and DynamoDB, with code stored in GitHub and incorporated into a CI/CD pipeline with Amplify.  
The app will let you create an account and log in, then request a ride by clicking on a map (powered by ArcGIS). The code can also be extended to build out more functionality.  
  
Cost  
All services used are eligible for the AWS Free Tier. Outside of the Free Tier, there may be small charges associated with building the app (less than $1 USD), but charges will continue to incur if you leave the app running. Please see the end of the YouTube video for instructions on how to delete all resources used in the video.  
  
The Application Code  
The application code is here in this repository.  
  
The Lambda Function Code  
Here is the code for the Lambda function, originally taken from the AWS workshop, and updated for Node 20.x:
```

On the READ me, you will find the lambda function code.

Locate the Lambda function code provided in the README.

Click on the **Copy** button next to the code to copy it to your clipboard.

README

The Application Code

The application code is here in this repository.

The Lambda Function Code

Here is the code for the Lambda function, originally taken from the [AWS workshop](#), and updated for Node 20.x:

```

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

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

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

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

  const rideId = toUrlString(randomBytes(16));
  console.log(`Received event ${rideId}, ${event}`);
}

const username = event.requestContext.authorizer.claims['cognito:username'];
const requestBody = JSON.parse(event.body);
const pickupLocation = requestBody.PickupLocation;

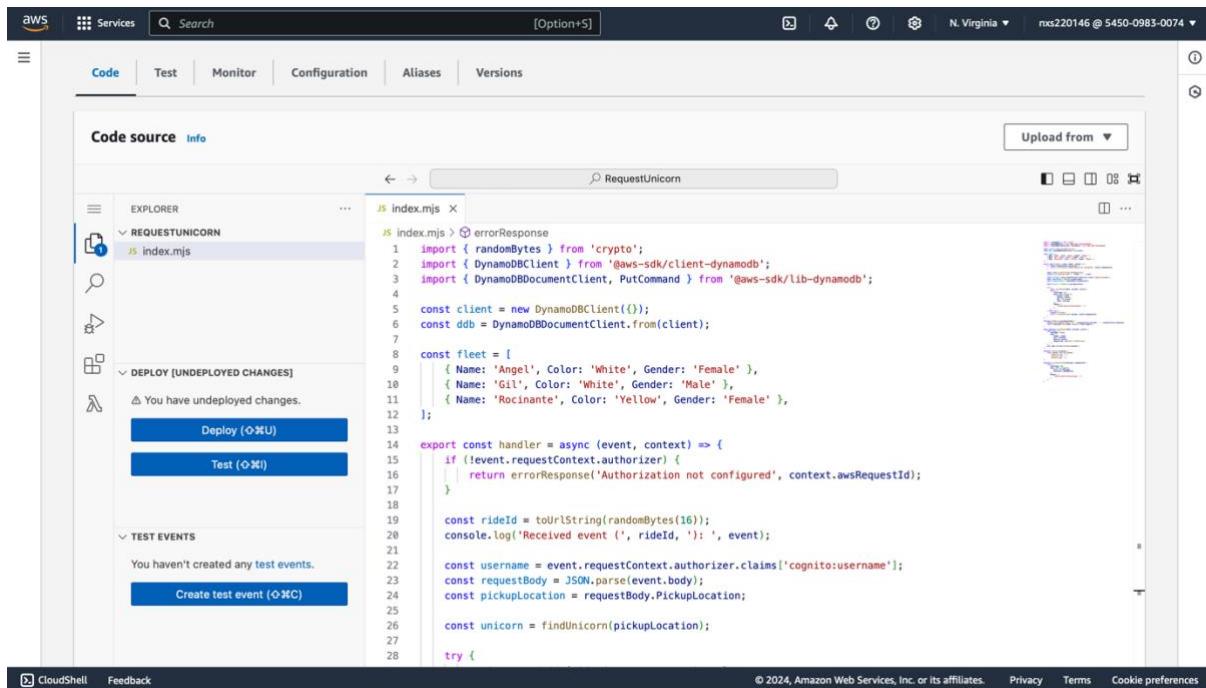
const unicorn = findUnicorn(pickupLocation);

try {
  await recordRide(rideId, username, unicorn);
}

```

In the AWS Management Console, navigate to your **RequestUnicorn** Lambda function:

1. Open the **Code** tab.
2. Select the existing code and delete it.
3. Paste the new code “**The Lambda Function Code**” you copied from GitHub into the code editor.



Note: In the Lambda code, locate **line 55** where the DynamoDB table name is defined. If you named your table something other than **Rides**, update the table name in the code to match your table's name exactly.

Ensure the name is accurate and case-sensitive, as a mismatch will prevent the function from working correctly.

The screenshot shows the AWS Lambda code editor interface. The function name is 'RequestUnicorn'. The code editor displays the 'index.mjs' file with the following content:

```
index.mjs
14 export const handler = async (event, context) => {
15   const params = {
16     TableName: 'Rides',
17     Item: {
18       RideId: rideId,
19       User: username,
20       Unicorn: unicorn,
21       RequestTime: new Date().toISOString(),
22     },
23   };
24   await ddb.send(new PutCommand(params));
25 }
26
27 function toUrlString(buffer) {
28   return buffer.toString('base64');
29 }
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
```

The sidebar on the left shows the project structure under 'REQUESTUNICORN' with 'index.mjs' selected. Under 'DEPLOY [UNDEPLOYED CHANGES]', there is a 'Deploy (0%U)' button. The status bar at the bottom indicates 'AWS Lambda code editor based on Code-OSS (VS Code open source)'.

Note: The above screenshot is a deliverable

Click Deploy to save and apply the changes.

The screenshot shows the AWS Lambda code editor interface after deployment. A green banner at the top indicates 'Successfully updated the function RequestUnicorn.' The code editor displays the same 'index.mjs' file as before, but now includes a 'Deployment successful' message in the bottom right corner of the code area.

The sidebar on the left shows the project structure under 'REQUESTUNICORN' with 'index.mjs' selected. Under 'DEPLOY', there is a 'Deploy (0%U)' button. The status bar at the bottom indicates 'AWS Lambda code editor based on Code-OSS (VS Code open source)'.

To create a test event and verify the Lambda function:

1. In the **RequestUnicorn** Lambda function console, click on **Create Test Event**.
2. Enter the name of the test event as **TestRequestEvent**.
3. Go to your **GitHub repository** and scroll down to the **README** section.
4. Locate the test event JSON “**The Lambda Function Test Function**” for the Lambda function.
5. Click the **Copy** button to copy the test event code.
6. Replace the existing JSON code in the test event editor with the copied code.

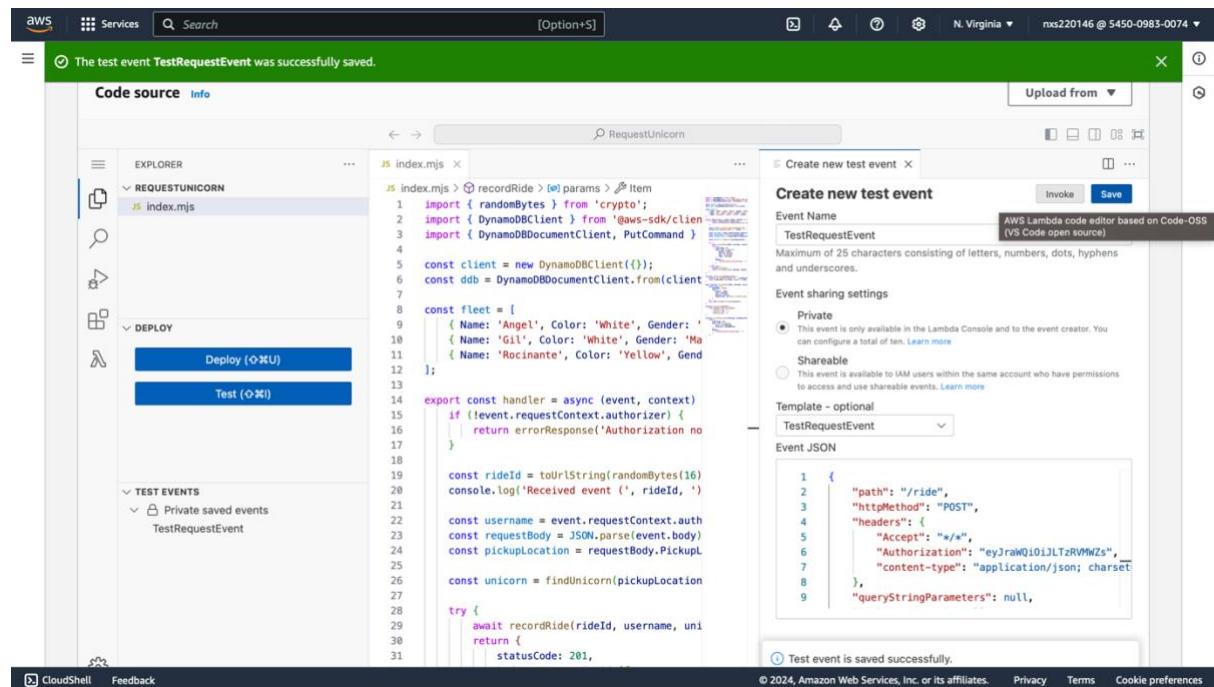
The screenshot shows the AWS Lambda console interface. The left sidebar includes 'EXPLORER', 'DEPLOY' (with 'Deploy' and 'Test' buttons), and 'TEST EVENTS' (with a 'Create test event' button). The main area shows the 'index.mjs' file content. To the right, a modal window titled 'Create new test event' is displayed, with the 'Event Name' field set to 'TestRequestEvent'. The 'Event JSON' field contains the following JSON code:

```
1  "path": "/ride",
2  "httpMethod": "POST",
3  "headers": {
4    "Accept": "*/*",
5    "Authorization": "eyJraWQiOiJLTzRVMWZs",
6    "Content-Type": "application/json; charset=UTF-8"
7  },
8  "queryStringParameters": null,
9  "pathParameters": null,
10 "requestContext": {
11   "authorizer": {
12     "claims": {
13       "cognito:username": "the_username"
14     }
15   }
16 },
17 "body": "{\"PickupLocation\":{\"Latitude\":47.6174755835663,\"Longitude\":-122.288378666}
```

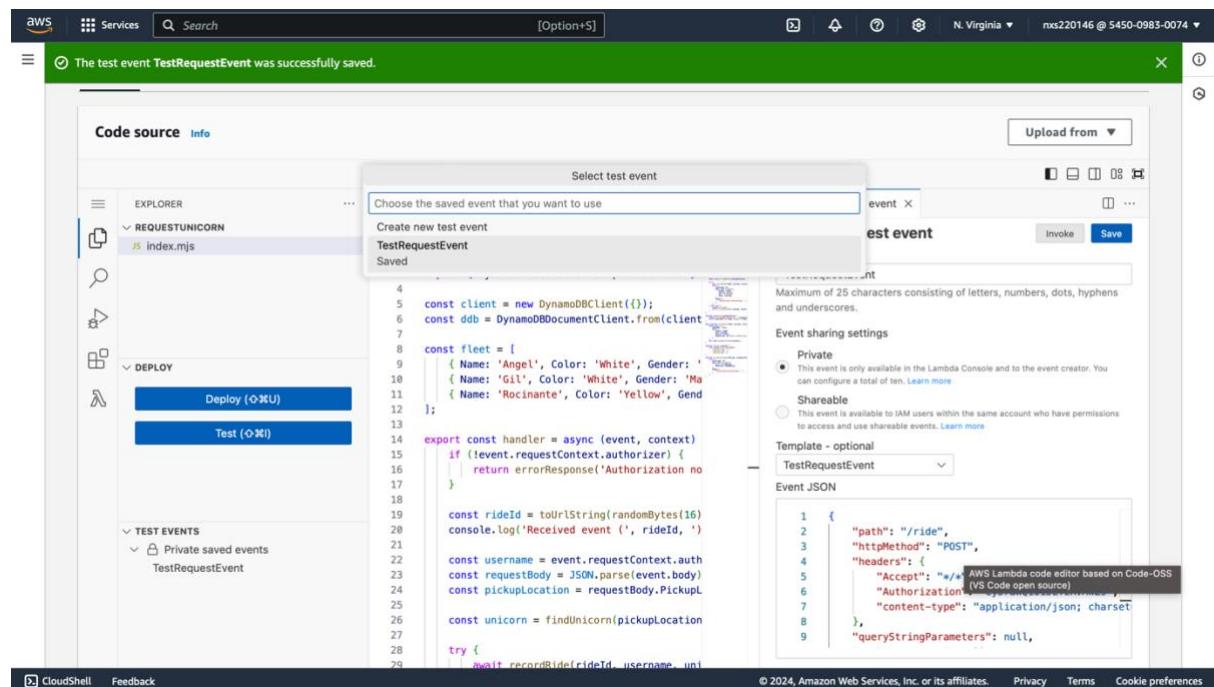
The screenshot shows the GitHub repository's README page. The 'The Lambda Function Test Function' section contains the following JSON code:

```
{
  "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\":-122.288378666}
```

Click **Save**. Then click on **Test**.



Select **TestRequestEvent**.



After running the test event, the **Execution results** section should display a **Succeeded** status. This confirms that your Lambda function executed successfully.

The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with 'EXPLORER', 'DEPLOY', and 'TEST EVENTS' sections. Under 'TEST EVENTS', there's a 'Private saved events' section containing 'TestRequestEvent'. The main area shows the code editor for 'index.mjs' with the following content:

```

1 import { randomBytes } from 'crypto';
2 import { DynamoDBClient } from '@aws-sdk/client-dynamodb';
3 import { DynamoDBDocumentClient, PutCommand } from '@aws-sdk/lib-dynamodb';
4
5 const client = new DynamoDBClient({});
6 const db = DynamoDBDocumentClient.from(client);
7
8 const fleet = [
9   { Name: 'Angel', Color: 'White', Gender: 'Female' },
10  { Name: 'G11', Color: 'White', Gender: 'Male' },
11  { Name: 'Rocinante', Color: 'Yellow', Gender: 'Male' };
12];
13
14 export const handler = async (event, context)
15   if (!event.requestContext.authorizer) {

```

Below the code editor, the 'PROBLEMS' tab shows 'Status: Succeeded'. The 'OUTPUT' tab displays the response to the test event:

```

{
  "statusCode": 201,
  "body": "{\"RideId\":\"QvcOSpEFM6gv1NgKsabRQ\"},\"Unicorn\":{\"Name\":\"Angel\", \"Color\":\"White\", \"Gender\":\"Female\"},\"Eta\":30 seconds\", \"Rider\":\"the_username\"}",
  "headers": {
    "Access-Control-Allow-Origin": "*"
  }
}

```

The 'FUNCTION LOGS' section shows the log entry:

```

START RequestId: ee51b7b6-5cb6-4c2b-ad2d-2e28008db741 Version: $LATEST

```

To verify the Lambda function's output:

1. Go to your **DynamoDB** console.
2. Open the **Rides** table (or the table you created).
3. Check the **Items** tab to see if a new record has been inserted.

This record confirms that the Lambda function successfully wrote data to the DynamoDB table

The screenshot shows the AWS DynamoDB console. On the left, there's a sidebar with 'Dashboard', 'Tables', 'Explore items', 'PartiQL editor', 'Backups', 'Exports to S3', 'Imports from S3', 'Integrations', 'Reserved capacity', and 'Settings'. Under 'Tables', there's a 'Tables (1)' section with a search bar and a 'Find tables' button. The main area shows the 'Rides' table details. The 'Scan or query items' section has 'Scan' selected. The 'Items returned (1)' table shows the following data:

| | RideId (String) | RequestTime | Unicorn | User |
|--|---------------------|----------------|-----------------|--------------|
| | QvcOSpEFM6gv1NgK... | 2024-11-11T... | { "Name": (...} | the_username |

The screenshot shows the AWS DynamoDB 'Edit item' interface. At the top, there are tabs for 'Form' and 'JSON view'. Below the tabs is a table titled 'Attributes' with columns for 'Attribute name', 'Value', and 'Type'. The table contains four rows:

| Attribute name | Value | Type |
|------------------------|--------------------------|--------|
| RideId - Partition key | QvcOSpEFM6gv1NgKsashRQ | String |
| RequestTime | 2024-11-11T00:07:45.223Z | String |
| Unicorn | Insert a field ▾ | Map |

Below the table are three buttons: 'Cancel', 'Save', and 'Save and close'. At the bottom of the page, there are links for 'CloudShell', 'Feedback', and copyright information: '© 2024, Amazon Web Services, Inc. or its affiliates.' followed by 'Privacy', 'Terms', and 'Cookie preferences'.

v. Setting Up API Gateway to Invoke the Ride-Sharing Functionality.

Create a new REST API in **API Gateway** to enable the invocation of your Lambda function for the ride-sharing feature. This will allow the application to trigger the **RequestUnicorn** Lambda function through a secure and scalable API endpoint.

In the AWS Management Console:

1. Search for **API Gateway** and open it.
2. Click on **Create an API**.
3. Select **REST API** from the options to create a RESTful interface for your Lambda function.

The screenshot shows the AWS API Gateway landing page. On the left, there's a sidebar with navigation links like 'APIs', 'Usage plans', 'API keys', 'Client certificates', and 'Settings'. The main area has three sections: 1) 'WebSocket API' which says 'Build a WebSocket API using persistent connections for real-time use cases such as chat applications or dashboards.' It includes a note about working with Lambda, HTTP, and AWS Services, and a yellow 'Build' button. 2) 'REST API' which says 'Develop a REST API where you gain complete control over the request and response along with API management capabilities.' It includes a note about working with Lambda, HTTP, and AWS Services, and 'Import' and 'Build' buttons. 3) 'REST API Private' which says 'Create a REST API that is only accessible from within a VPC.' It includes a note about working with Lambda, HTTP, and AWS Services, and 'Import' and 'Build' buttons.

In the **Create API** screen: Enter **WildRydes** as the **API Name**. Click **Create API** to proceed.

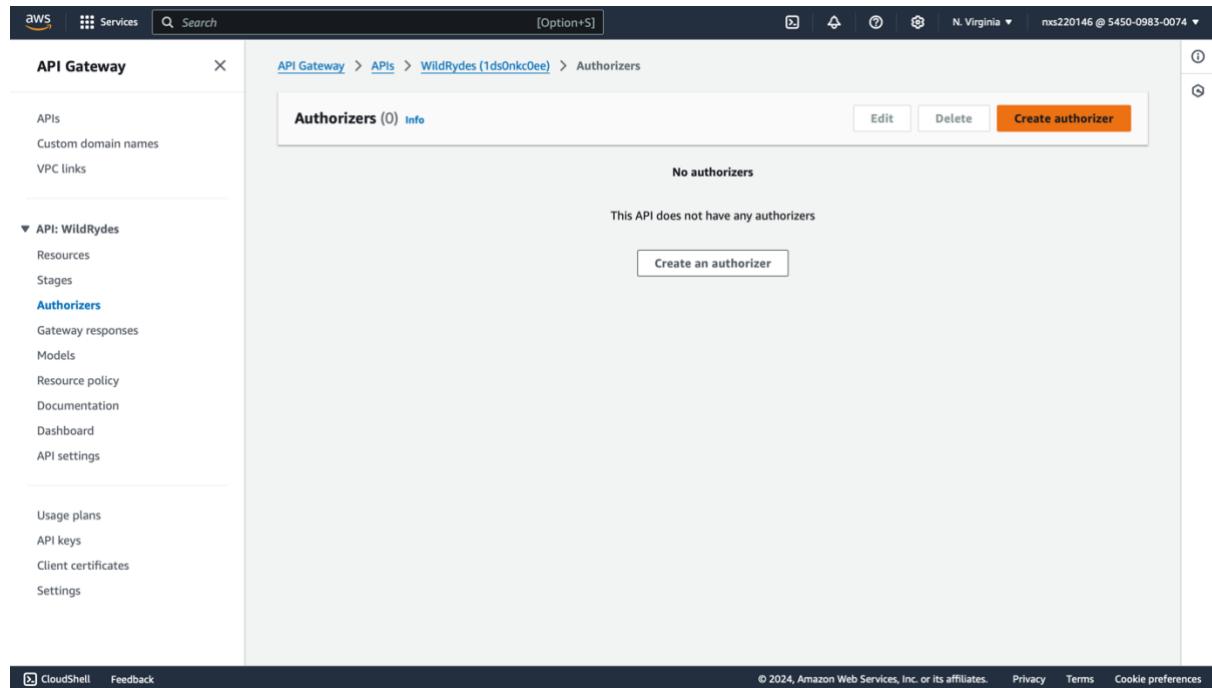
The screenshot shows the 'Create REST API' form. At the top, it says 'API details' with four options: 'New API' (selected), 'Clone existing API', 'Import API', and 'Example API'. Below that, the 'API name' field contains 'WildRydes'. Under 'Description - optional', there's a large empty text area. In the 'API endpoint type' section, it says '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.' A dropdown menu is set to 'Regional'. At the bottom right, there are 'Cancel' and 'Create API' buttons.

Creating an Authorizer to Enable API Gateway Integration with Cognito

Since we're using a **Cognito User Pool**, we need to create an authorizer in **API Gateway** to authenticate API calls. API Gateway will use **JSON Web Tokens (JWTs)** returned by

Cognito to verify and authorize requests. This step connects API Gateway with the Cognito User Pool for secure access.

1. In the **API Gateway** console, navigate to your API.
2. On the left pane, click on **Authorizers**.
3. Select **Create Authorizer** to set up the connection.



In the **Create Authorizer** screen:

1. Set the **Name** to **WildRydes**.
2. For the **Authorizer Type**, select **Cognito**.
3. Choose your **Cognito User Pool** from the dropdown menu.
4. In the **Token Source** field, enter **Authorization**.
5. Click **Create Authorizer** to finalize the setup.

The screenshot shows the 'Create authorizer' page in the AWS API Gateway console. The 'Authorizer name' is set to 'WildRydes'. The 'Authorizer type' is selected as 'Cognito'. The 'Cognito user pool' is set to 'us-east-1' and the 'Token source' is 'Authorization'. The 'Create authorizer' button is highlighted.

Open the **WildRydes** authorizer you just created.

In the **Token Value** field, paste the **token** you previously saved.

Note: Ensure there are no extra spaces at the beginning or end of the token.

The screenshot shows the 'Authorizers' list page in the AWS API Gateway console. A green banner at the top indicates that the 'WildRydes' authorizer was successfully created. The 'WildRydes' authorizer is listed with the following details: Authorizer ID 'eghsqq', Cognito pool 'wildrydes - TR665pOCD (us-east-1)', Token source 'Authorization', and Token validation - optional 'none'. The 'Create authorizer' button is highlighted.



COGNITO USER POOL ID: us-east-1_TR665pOCD

CLIENT ID: 71jltqblg33mnvm29htd0ql5qe

```
eyJraWQiOiJCT3ViTjhBd2FzTFg4OW9vcVFPdGpWeGZhOFkxa3ViSEpqRkRrbEhTOVFZPSIsImFsZyl6IiJTMjU2In0.eyJzdWliOiJkNDc4MTRIOC01MGmxLTcwZTYtNzYyZS1jYjU2ZmRiZGYZwYiLCJlbWFpbF92ZXJpZmlIZCI6dHJ1ZSwiaXNzljoiaHR0cHM6XC9cL2NvZ25pdG8taWRwLnVzLWVhc3QtMS5hbWF6b25hd3MuY29tXC91cy1lYN0LTffVFI2NjVwT0NEliwiY29nbmli0bzp1c2VybmrZSI6lm54czlyMDE0Ni1hdC11dGRhbGxhcy5lZHUiLCJvcmlnaW5fanRpIjoimWMxYjQyZjMtNWU5NC00MjFkLTIIODAtOTBjNGJjNWQzNzdjliwiYXVkljoiNzFqbHRxYmxnMzNtbmZtMjlodGQwcWw1cWUiLCJldmVudF9pZC16ljFinjYzYzhILWZmYjUtNDc1Zi05YmQxLTm0Mzdln2E5YWl5NClslnRva2VuX3VzZSI6lmkliwiYXV0aF90aW1llojoxNzMxMjgxMDgyLCJleHAiOjE3MzEyODQ2ODIsmlhdC16MTczMTI4MTA4MiwanRpIjoINTVmMTFjOTATMWRjYi000OD13LTk0NjctMDBkYmfIMGQwNjY5liwiZWhaWwiOjJueHMyMjAxNDZAdXRkYWxsYXMuZWR1In0.2ekyXhqS7YxCJn96tMhQwIDE2sBD1gByQFENYR9-4x9yuprRu94bdIAOoFekFu_1mZqvA9e2MfpjHYOGvE-_Wj9XY6enp8RwD7r_3G2K-klsnsEgke1P4r7fUTXMA5qr5-eOzQ8YzrrHr3SFylfqe3G2HpWsm9KoLNUs-ghP6B6kVZ1NOdtM0dfywMTO7a15Fx8twzrxn_MabqFKYJuj2oyX73Yk6Hf6xTFXGS87ItG2LuWmoez8e7hBcYes0f2hT1nfwy4fWvcr9E2LaTISpnozL31psPnIE89jXO2UvvusHuuvv8Df0dnrdWB1wnEijDEdk6aCx-FH2zXMeb-g
```

arn:aws:dynamodb:us-east-1:545009830074:table/Rides

Click on **Test Authorizer** to validate the setup.

If everything is configured correctly, you should receive a **200 status code**, indicating the authorizer is working as expected.

API Gateway > APIs > WildRydes (1dsOnkc0ee) > Authorizers > WildRydes

WildRydes

Authorizer details

| | | | |
|---------------|-----------------------------------|-----------------------------|---------------|
| Authorizer ID | eghsqq | Token source | Authorization |
| Cognito pool | wildrydes - TR665pOCD (us-east-1) | Token validation - optional | None |

Test authorizer

Test your authorizer with a simulated invocation request. Enter an identity token that's provisioned from your Cognito user pool.

Token source: Authorization: eyJraWQiOiJCT3ViTjhBd2FzTFg4OW9vcVFPdGp

Authorizer test: WildRydes

200
Claims

```
{"aud": "71jltqblg33mnvm29htd0ql5qe", "auth_time": "1731281082", "cognito:username": "nxs220146-at-utdallas.edu", "email": "nxs220146@utdallas.edu"},
```

Creating a Resource and POST Method in API Gateway for Lambda Integration

1. In the **API Gateway** console, select your **WildRydes** API.
2. On the left pane, click **Resources**.
3. Click **Create Resource** to define a new endpoint where you'll connect your Lambda function.

The screenshot shows the AWS API Gateway Resources page. The left sidebar is collapsed, and the main area displays a single resource entry:

- Create resource** button (disabled)
- Resource details** section:
 - Path**: /
 - Resource ID**: jijffqhiij
- Methods (0)** section:
 - Method type**, **Integration type**, **Authorization**, and **API key** dropdowns
 - No methods defined.

In the **Create Resource** screen: Enter **ride** as the **Resource Name**.

Check the box to enable CORS for the resource.

Enabling CORS (Cross Origin Resource Sharing) for the API Gateway

We do need to enable CORS, if you don't do that nothing's going to work and that's because the domain of the site on amplify is going to be different from the domain of API Gateway here so it's going cross origin.

Click on **create resource**.

Create resource

Resource details

Proxy resource info
Proxy resources handle requests to all sub-resources. To create a proxy resource use a path parameter that ends with a plus sign, for example {proxy+}.

Resource path: / Resource name: ride

CORS (Cross Origin Resource Sharing) Info
Create an OPTIONS method that allows all origins, all methods, and several common headers.

Cancel **Create resource**

Choose Create Method.

API Gateway

Resources

Methods (1)

| Method type | Integration type | Authorization | API key |
|-------------|------------------|---------------|--------------|
| OPTIONS | Mock | None | Not required |

API actions Deploy API

To configure the method:

1. Select **POST** as the **Method Type** and click the checkmark to confirm.
2. For the **Integration Type**, choose **Lambda Function**.
3. Check the box for **Enable Lambda Proxy Integration** to allow API Gateway to pass requests directly to Lambda.
4. In the **Lambda Function** field, select the **RequestUnicorn** function you created earlier.

This will connect your API Gateway endpoint to the Lambda function.

The screenshot shows the 'Create method' configuration page for AWS Lambda. The 'Method type' is set to 'POST'. Under 'Integration type', 'Lambda function' is selected, showing the Lambda icon. Other options like 'HTTP', 'Mock', 'AWS service', and 'VPC link' are shown with their respective icons. Below this, 'Lambda proxy integration' is selected, with the note 'Send the request to your Lambda function as a structured event.' A dropdown for 'Lambda function' shows 'us-east-1' and a search bar containing 'ibda:us-east-1:545009830074:function:RequestUnicorn'. The bottom navigation bar includes CloudShell, Feedback, and links to AWS terms and cookie preferences.

Click **Create Method** to complete the setup.

This will connect your API Gateway endpoint to the Lambda function.

The screenshot shows the 'Create method' configuration page for AWS Lambda, with the 'Lambda proxy integration' option selected. The Lambda function ARN 'arn:aws:lambda:us-east-1:545009830074:function:RequestUnicorn' is entered in the search bar. A note below it says 'Grant API Gateway permission to invoke your Lambda function's resource policy yourself, or provide an invoke role that API Gateway uses to invoke your function.' The 'Integration timeout' is set to '29000'. On the right, there are sections for 'Method request settings', 'URL query string parameters', 'HTTP request headers', and 'Request body'. At the bottom, there are 'Cancel' and 'Create method' buttons, with the latter being highlighted in orange.

In the **POST** method configuration for your **ride** resource:

1. Go to the **Method Request** section.
2. Click **Edit** to modify the method settings as needed.

The screenshot shows the AWS API Gateway interface. On the left, the navigation pane is visible with sections like APIs, Stages, Authorizers, and API: WildRydes. The main area displays the flow of requests: Client → Method request → Integration request → Lambda integration → Integration response → Method response. Below this, the 'Method request' tab is selected in the navigation bar. The 'Method request settings' panel is open, showing the following configuration:

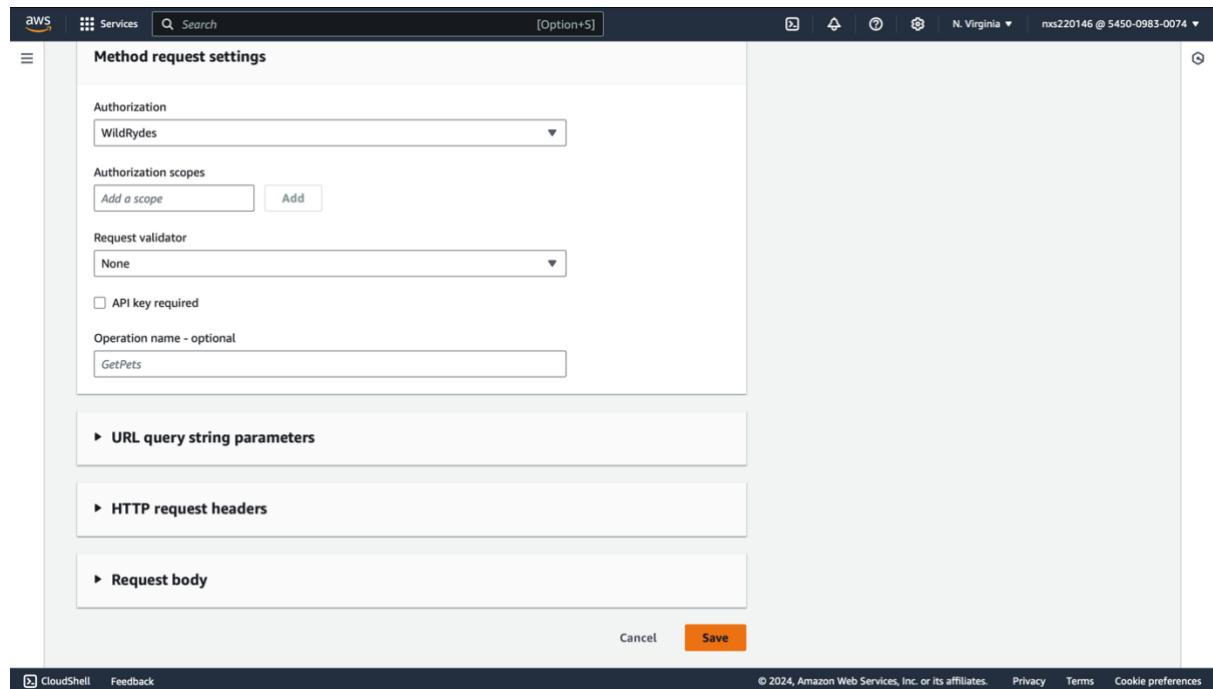
| Method request settings | |
|-------------------------|------------------------------------|
| Authorization | API key required |
| NONE | False |
| Request validator | SDK operation name |
| None | Generated based on method and path |

In the **Method Request** settings:

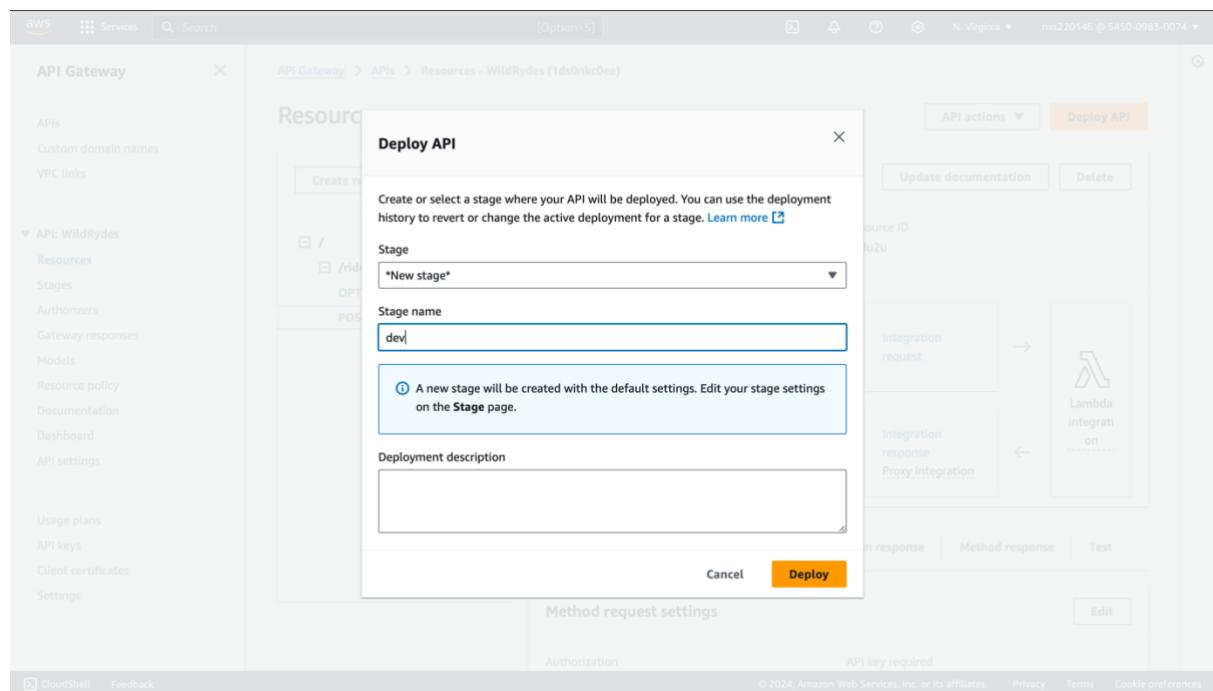
1. Locate the **Authorization** field and click **Edit**.
2. From the dropdown, select the **Cognito User Pool** authorizer you created earlier (e.g., **WildRydes**).

The screenshot shows the 'Edit method request' dialog box. The 'Method request settings' tab is selected. In the 'Authorization' section, 'WildRydes' is selected from the dropdown. The 'Cognito user pool authorizers' dropdown also lists 'WildRydes'. The 'API key required' checkbox is unchecked. The 'Operation name - optional' field contains 'GetPets'. Below the main settings, there are sections for 'URL query string parameters' and 'HTTP request headers'.

Leave the rest to default and click on **Save**.



Click on **Deploy API**, select **New Stage**, enter the stage name as **dev** and **deploy**.



Copy the **invoke URL**, and paste it in your notepad.

The screenshot shows the AWS API Gateway interface. On the left, a sidebar for the 'API: WildRydes' shows sections like 'Stages'. The main area displays the 'Stages' for the 'WildRydes (1ds0nkc0ee)' API. The 'dev' stage is selected. The 'Stage details' section shows the stage name 'dev', cache cluster 'Inactive', and default method-level caching 'Inactive'. Below this, a tooltip with a green checkmark and the word 'Copied' is overlaid on the invoke URL field, which contains the URL `https://1ds0nkc0ee.execute-api.us-east-1.amazonaws.com/dev`. At the bottom of the stage details, it says 'Active deployment j4l3dg on November 10, 2024, 18:33 (UTC-06:00)'. The footer includes links for CloudShell, Feedback, and copyright information.

Note: The above screenshot is a deliverable

Updating the config file for the new Invoke URL from API Gateway

Open Config.js in github. Click on edit this file.

The screenshot shows the GitHub repository 'nxs220146 / wildrydes-site'. The 'Code' tab is selected, and the file 'config.js' is open. The code editor shows the following content:

```
window._config = {
  cognito: {
    userPoolId: 'us-east-1_TR665p0CD', // e.g. us-east-2_uXboG5pAb
    userPoolClientId: '7ijltqb1g3mnmv29htd0ql5qe', // e.g. 25ddkmj4v6hfsfvrupf17n4hv
    region: 'us-east-1' // e.g. us-east-2
  },
  api: {
    invokeUrl: '' // e.g. https://rc7nyt4tql.execute-api.us-west-2.amazonaws.com/prod',
  }
};
```

Paste the invoke url between the quotes ‘’ on line 8.

The screenshot shows the GitHub code editor interface for a repository named 'wildrydes-site'. The left sidebar displays the project structure with files like 'main.css', 'fonts', 'images', 'js' (containing 'cognito-auth.js', 'config.js', 'esri-map.js', 'main.js', 'ride.js', 'vendor.js'), 'README.md', and various HTML files ('apply.html', 'faq.html', 'favicon.ico', 'index.html', 'investors.html', 'register.html'). The main editor area shows the 'config.js' file with the following content:

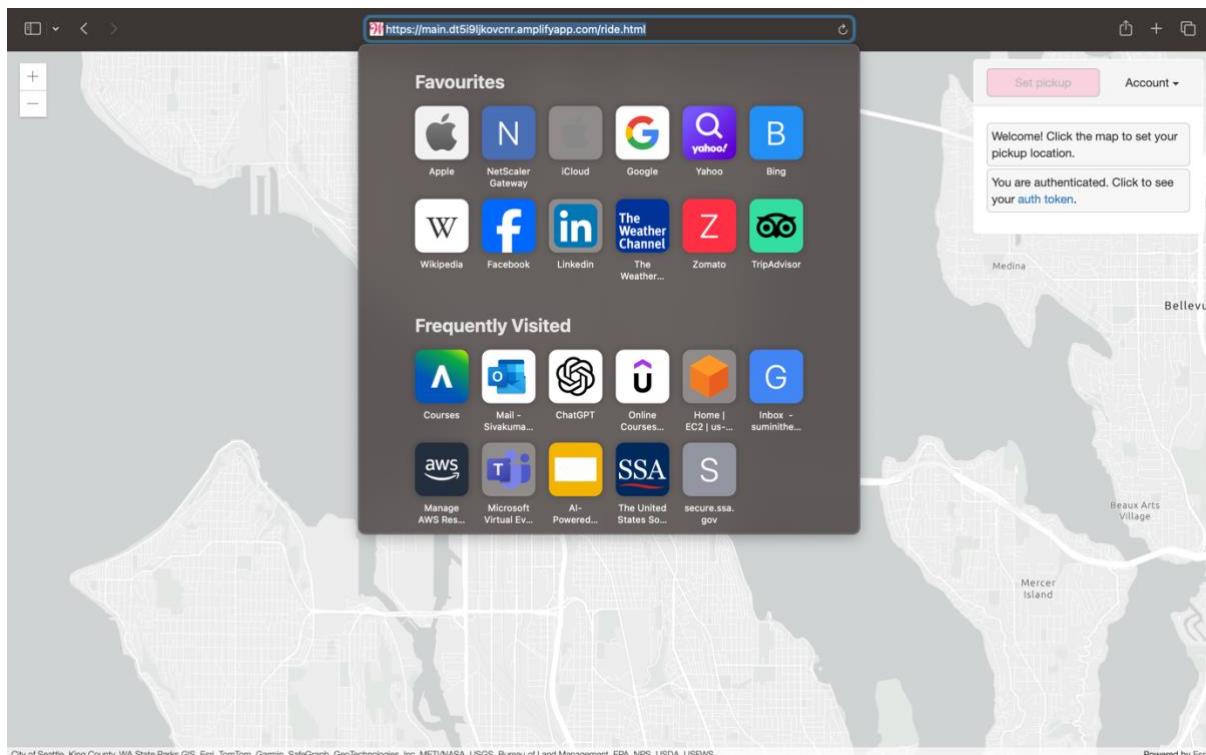
```
1 window._config = {
2   cognito: {
3     userPoolId: 'us-east-1_TR665pOCD', // e.g. us-east-2_uXboG5pAb
4     userPoolClientId: '71j1tqb1g3mnv29htd0q15qe', // e.g. 25ddkmj4v6hfsfvruhpfi7n4hv
5     region: 'us-east-1' // e.g. us-east-2
6   },
7   api: {
8     invokeUrl: 'https://ds0nk0ee.execute-api.us-east-1.amazonaws.com/dev' // e.g. https://rc7nyt4tql.execute-api.us-west-2.amazonaws.com/dev
9   }
10 };
11
```

At the top right, there are buttons for 'Cancel changes' and 'Commit changes...', and settings for 'Spaces', '4', and 'No wrap'. A status bar at the bottom indicates keyboard navigation instructions.

Click on Commit changes.

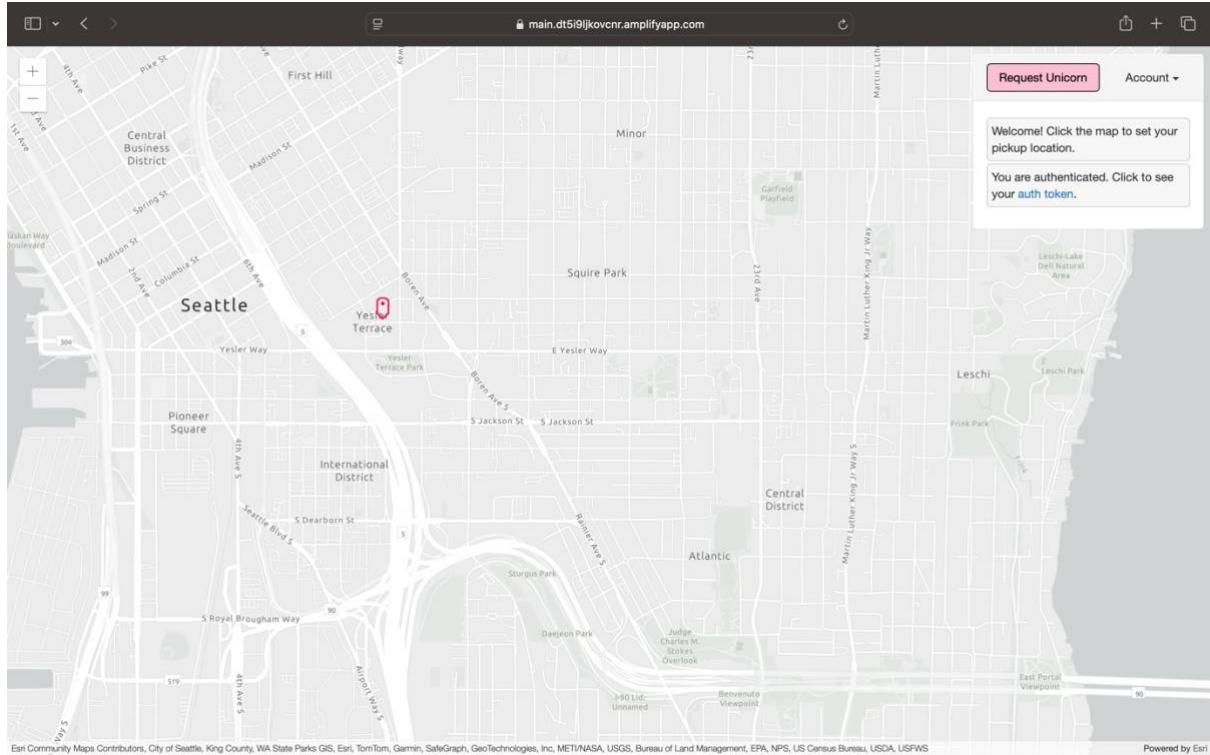
Testing our final application

Go to your website, and refresh once the new changes are deployed on Amplify. Navigate to **/ride.html** to access the ride functionality and verify that everything is working as expected.

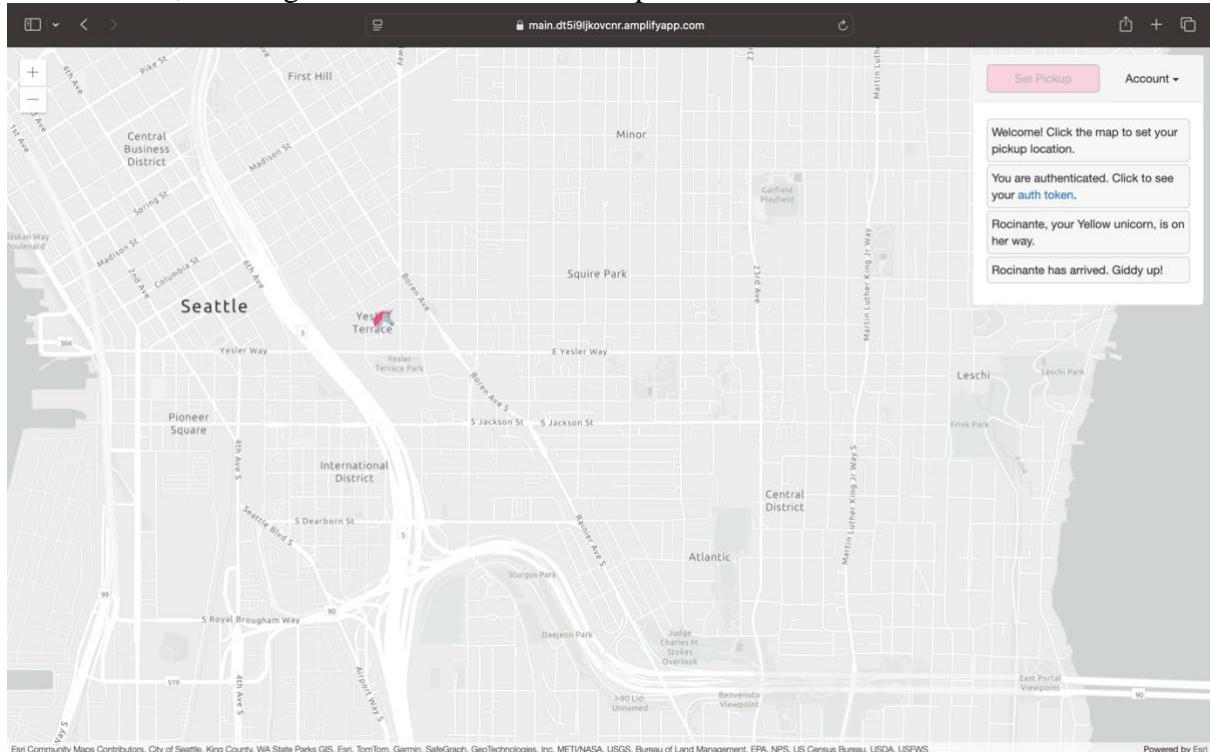


Click anywhere on the map to select a location.

Click on **Request Unicorn** to trigger the functionality and see if the ride-sharing process works as expected.



After clicking **Request Unicorn**, the unicorn should appear on the map, confirming that the ride request was successfully processed. You should see the updated information displayed on the screen, showing the unicorn's arrival as expected.



Note: The above screenshot is a deliverable (The URL must be visible)

Open your **DynamoDB** console and navigate to the **Rides** table.

Check the **Items** tab to see if a new record has been added, confirming that the ride-sharing information was successfully written to the table by the Lambda function.

The screenshot shows the AWS DynamoDB console with the 'Rides' table selected. The 'Scan or query items' section is active, showing a successful scan operation with 0.5 read capacity units consumed. The 'Items returned' section displays two items:

| RideId | RequestTime | Unicorn | User |
|------------------------|--------------------------|------------------|---------------------------|
| hfbRp2Tf2z5UWgyjElmd7w | 2024-11-11T00:52:12.086Z | { "Name": "... } | nxs220146-at-utdallas.edu |
| QvcOSpEFM6gv1NgK... | 2024-11-11T... | { "Name": "... } | the_username |

The screenshot shows the 'Edit item' form for the 'Rides' table. The 'Form' tab is selected. The 'Attributes' section contains four fields:

| Attribute name | Value | Type | Action |
|------------------------|---------------------------|--------|--------|
| RideId - Partition key | hfbRp2Tf2z5UWgyjElmd7w | String | Remove |
| RequestTime | 2024-11-11T00:52:12.086Z | String | Remove |
| Unicorn | Insert a field ▾ | Map | Remove |
| User | nxs220146-at-utdallas.edu | String | Remove |

At the bottom are 'Cancel', 'Save', and 'Save and close' buttons.

To clean up after the lab and avoid any unnecessary charges, make sure to delete the following resources:

1. **Lambda Function:** Go to the Lambda console, select your RequestUnicorn function, and click Delete.
2. **API Gateway:** In the API Gateway console, delete the WildRydes API.
3. **DynamoDB Table:** Go to the DynamoDB console, select your Rides table, and click Delete Table.
4. **Cognito User Pool:** In the Cognito console, select the WildRydes user pool and delete it.
5. **IAM Role:** In the IAM console, delete the WildRydesLambda role.
6. **Amplify App:** Go to the Amplify console, select your app, and click Delete App.

Make sure to delete any other resources you've created during the lab to ensure you're not charged for unused services.