



# AWS – CAPSTONE 3

Build a serverless web application using Amplify, Cognito, API Gateway, Lambda & DynamoDB

Rajeswari Subramanian | December,2022

## Contents

Build a serverless web application using Amplify, Cognito, API Gateway, Lambda & DynamoDB.....	1
Project Overview:.....	4
Modules .....	4
Architectural Diagram .....	4
Module 1: Static Web Hosting .....	5
Architectural Overview .....	5
Step 1: Select Region.....	6
Step 2: Create a Git Repository.....	6
Step 3: Populate the Git Repository.....	9
Step 4: Enable Web Hosting with the Amplify Console .....	10
Step 5: Modify site .....	11
Module 2: User Management.....	13
Architectural Overview .....	14
Step 1: Create an Amazon Cognito User Pool.....	14
Step 2: Add an App to User Pool.....	15
Step 3: Update the website config.....	16
Step 4: Validate the Implementation.....	17
Module 3: Serverless Backend .....	26
Architectural Overview .....	26
Step 1: Create an Amazon DynamoDB Table .....	26
Step 2: Create an IAM Role for the Lambda function .....	27
Step 3: Create a Lambda Function for Handling Requests .....	30
Step 4: Validate the Implementation.....	33
Module 4: RESTful API.....	35
Architectural Overview .....	36
Step 1: Create a New REST API.....	36
Step 2: Create a Cognito User Pool Authorizer .....	37
Step 3: Create a new resource and method .....	41
Step 4: Deploy our API .....	43
Step 5: Update the Website Config.....	45
Step 6: Validate the implementation.....	46
Cost Analysis .....	48
Lessons & Observations .....	48



## Project Overview:

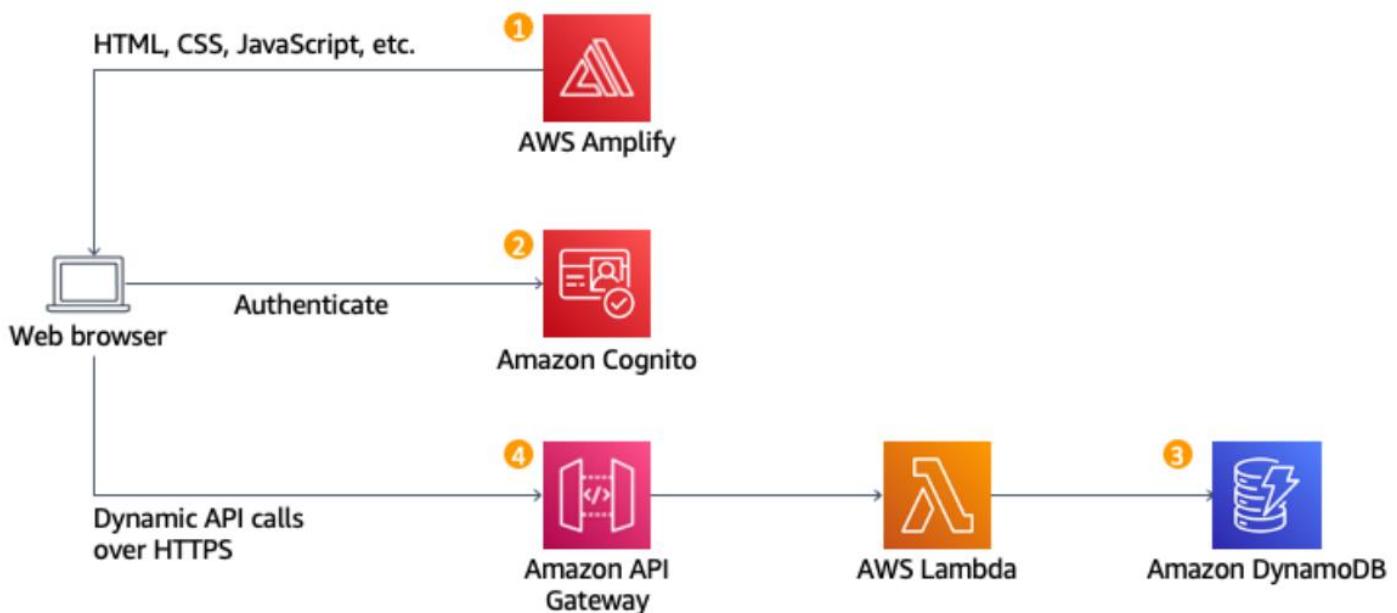
In this project. We are creating simple serverless web application that enables users to request unicorn rides from the Wild Rydes fleet. The application will present users with an HTML based user interface for indicating the location where they would like to be picked up and will interface on the backend with a RESTful web service to submit the request and dispatch a nearby unicorn. The application will also provide facilities for users to register with the service and log in before requesting rides.

## Modules

This capstone project is divided into four modules. Each module describes a scenario of what we're going to build and step-by-step directions to help to implement the architecture.

1. Host a Static Website: Configure AWS Amplify to host the static resources for the web application with continuous deployment built in
2. Manage Users: Create an Amazon Cognito user pool to manage users' accounts
3. Build a Serverless Backend: Build a backend process for handling requests our web application
4. Deploy a RESTful API: Use Amazon API Gateway to expose the Lambda function, that built in the previous module as a RESTful API

## Architectural Diagram



## 1. Static Web Hosting:

AWS Amplify hosts static web resources including HTML, CSS, JavaScript, and image files which are loaded in the user's browser.

## 2. User Management:

Amazon Cognito provides user management and authentication functions to secure the backend API.

## 3. Serverless Backend:

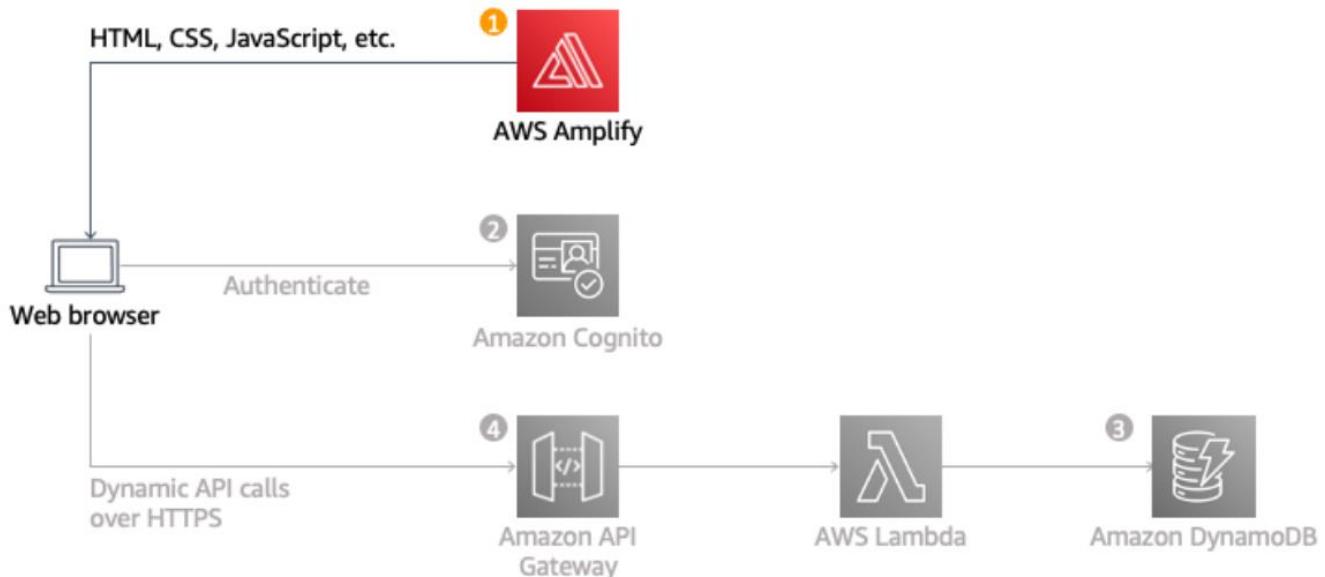
Amazon DynamoDB provides a persistence layer where data can be stored by the API's Lambda function.

## 4. RESTful API:

JavaScript executed in the browser sends and receives data from a public backend API built using Lambda and API Gateway.

# Module 1: Static Web Hosting

## Architectural Overview

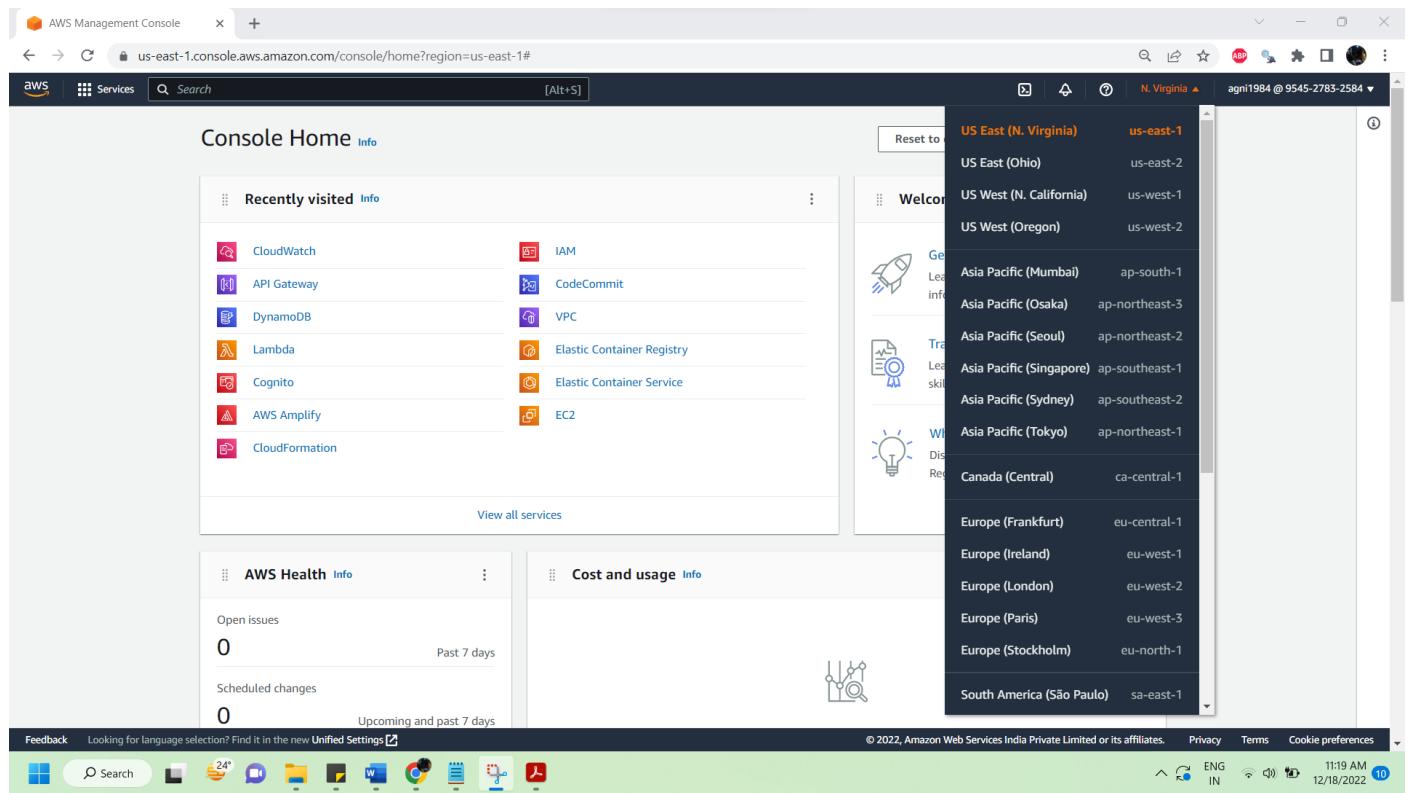


The architecture for this module is straightforward. All of our static web content including HTML, CSS, JavaScript, images, and other files will be managed by AWS Amplify Console. End users will then access the site using the public website URL exposed by AWS Amplify Console. We don't need to run any web servers or use other services to make our site available.

## Step 1: Select Region

This web application can be deployed in any AWS Region that supports all the services used in this application, which include AWS Amplify, AWS CodeCommit, Amazon Cognito, AWS Lambda, Amazon API Gateway, and Amazon DynamoDB.

Select “**US East (N. Virginia)**” Region from the dropdown in the upper right corner of the AWS Management Console.



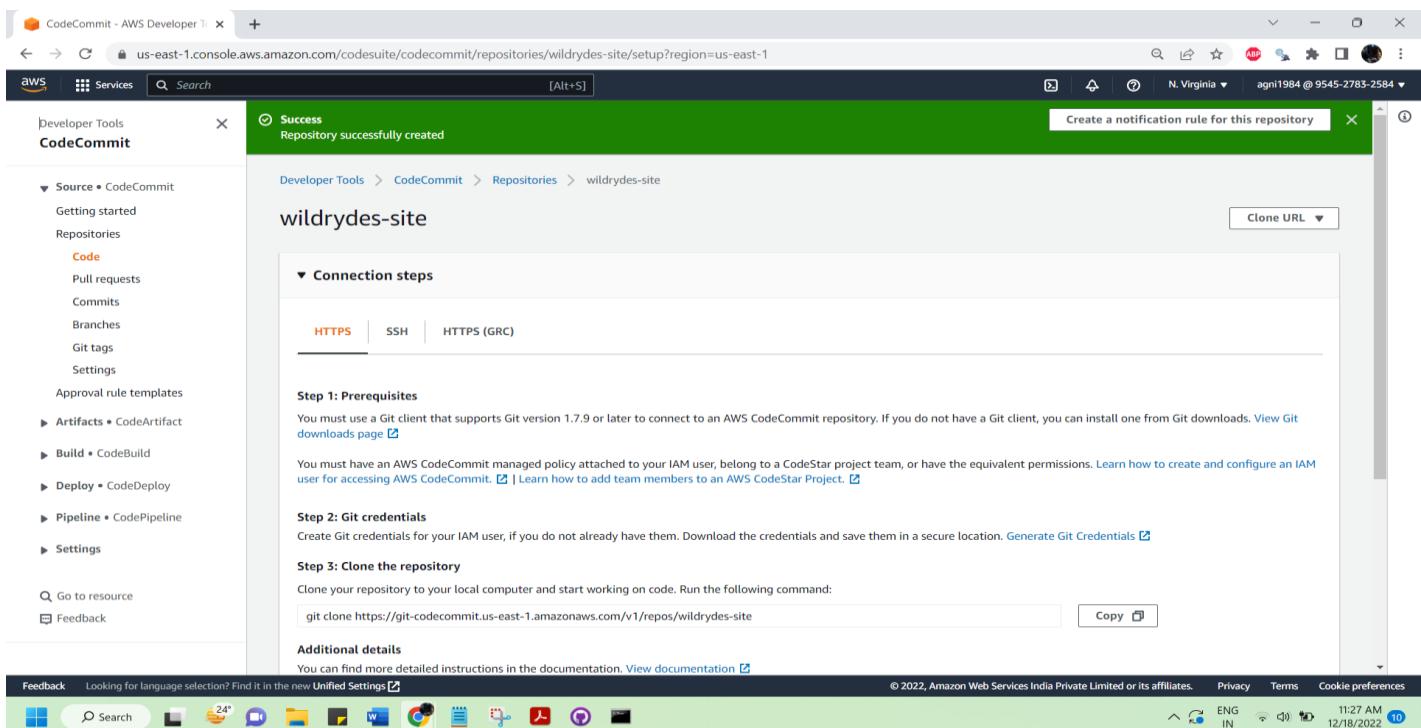
The screenshot shows the AWS Management Console homepage. On the right side, there is a dropdown menu for selecting a region. The "US East (N. Virginia)" region is highlighted in orange, indicating it is selected. A list of available regions is shown below:

Region	Region ID
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

## Step 2: Create a Git Repository

'CodeCommit' used to store & manage our application code. Follow the below steps to create 'Codecommit' repository.

- Open the AWS CodeCommit console
- Select Create Repository
- Set the Repository name\* to "**wildrydes-site**"
- Select Create
- Now that the repository is created.



The screenshot shows the AWS CodeCommit console in a browser window. The URL is [us-east-1.console.aws.amazon.com/codesuite/codecommit/repositories/wildrydes-site/setup?region=us-east-1](https://us-east-1.console.aws.amazon.com/codesuite/codecommit/repositories/wildrydes-site/setup?region=us-east-1). The page displays a green success message: "Success: Repository successfully created". The navigation pane on the left shows "Source" selected under "CodeCommit". The main content area is titled "wildrydes-site" and contains "Connection steps" for HTTPS, SSH, and HTTPS (GRC). It provides instructions for Step 1: Prerequisites, Step 2: Git credentials, and Step 3: Clone the repository. A copy button is available for the clone URL. At the bottom, there's an "Additional details" section and a feedback link. The browser status bar shows the user is from N. Virginia and has a session ID.

- Generate Git credentials in the IAM user, in IAM console following these instructions.
  - In the IAM console, in the navigation pane, choose **Users**, and from the list of users, choose IAM user.
  - On the user details page, choose the **Security Credentials** tab, and in **HTTPS Git credentials for AWS CodeCommit**, choose **Generate**.
  - Copy the user name and password that IAM generated, either by showing, copying, and then pasting this information into a secure file on the local computer, or by choosing **Download credentials** to download this information as a .CSV file. We need this information to connect to CodeCommit.

Screenshot of the AWS IAM Management Console showing the 'Users' section. A red box highlights the 'HTTPS Git credentials for AWS CodeCommit' section.

**Identity and Access Management (IAM)**

- Dashboard
- Access management
  - User groups
  - Users**
  - Roles
  - Policies
  - Identity providers
  - Account settings
- Access reports
  - Access analyzer
  - Archive rules
  - Analyzers
  - Settings
- Credential report
- Organization activity
- Service control policies (SCPs)

**Search IAM**

AWS account ID: **AGNI1984@954527832584**

Access key ID	Created	Last used	Status
AKIA54PSK3EMMMZB05Q	2022-08-25 20:00 UTC+0530	2022-08-25 21:36 UTC+0530 with ec2 in us-east-1	Inactive   Make active
AKIA54PSK3IEOU4VPS72	2022-12-12 23:17 UTC+0530	2022-12-15 11:46 UTC+0530 with s3 in us-east-1	Active   Make inactive

**SSH keys for AWS CodeCommit**  
Use SSH public keys to authenticate access to AWS CodeCommit repositories. [Learn more](#)

**Upload SSH public key**

SSH key ID	Uploaded	Status
No results		

**HTTPS Git credentials for AWS CodeCommit**  
Generate a user name and password you can use to authenticate HTTPS connections to AWS CodeCommit repositories. You can generate and store up to 2 sets of credentials. [Learn more](#)

**Generate credentials** Actions ▾

User name	Status	Created
agni1984-at-954527832584	Active	2022-12-15 11:40 UTC+0530

**Credentials for Amazon Keyspaces (for Apache Cassandra)**  
Generate a user name and password you can use to authenticate to Amazon Keyspaces. You can generate and store up to 2 sets of credentials for Amazon Keyspaces. [Learn more](#)

**Generate credentials**

No credentials have been generated.

Feedback Looking for language selection? Find it in the new [Unified Settings](#) © 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

➤ In the CodeCommit console, From the Clone URL drop down, select Clone HTTPS

Screenshot of the AWS CodeCommit console showing the 'wildrydes-site' repository setup page. A red box highlights the 'Clone URL' dropdown menu.

**Developer Tools > CodeCommit > Repositories > wildrydes-site**

**wildrydes-site**

**Connection steps**

**Step 1: Prerequisites**  
You must use a Git client that supports Git version 1.7.9 or later to connect to an AWS CodeCommit repository. If you do not have a Git client, you can install one from Git downloads page.

**Step 2: Git credentials**  
Create Git credentials for your IAM user, if you do not already have them. Download the credentials and save them in a secure location. [Generate Git Credentials](#)

**Step 3: Clone the repository**  
Clone your repository to your local computer and start working on code. Run the following command:

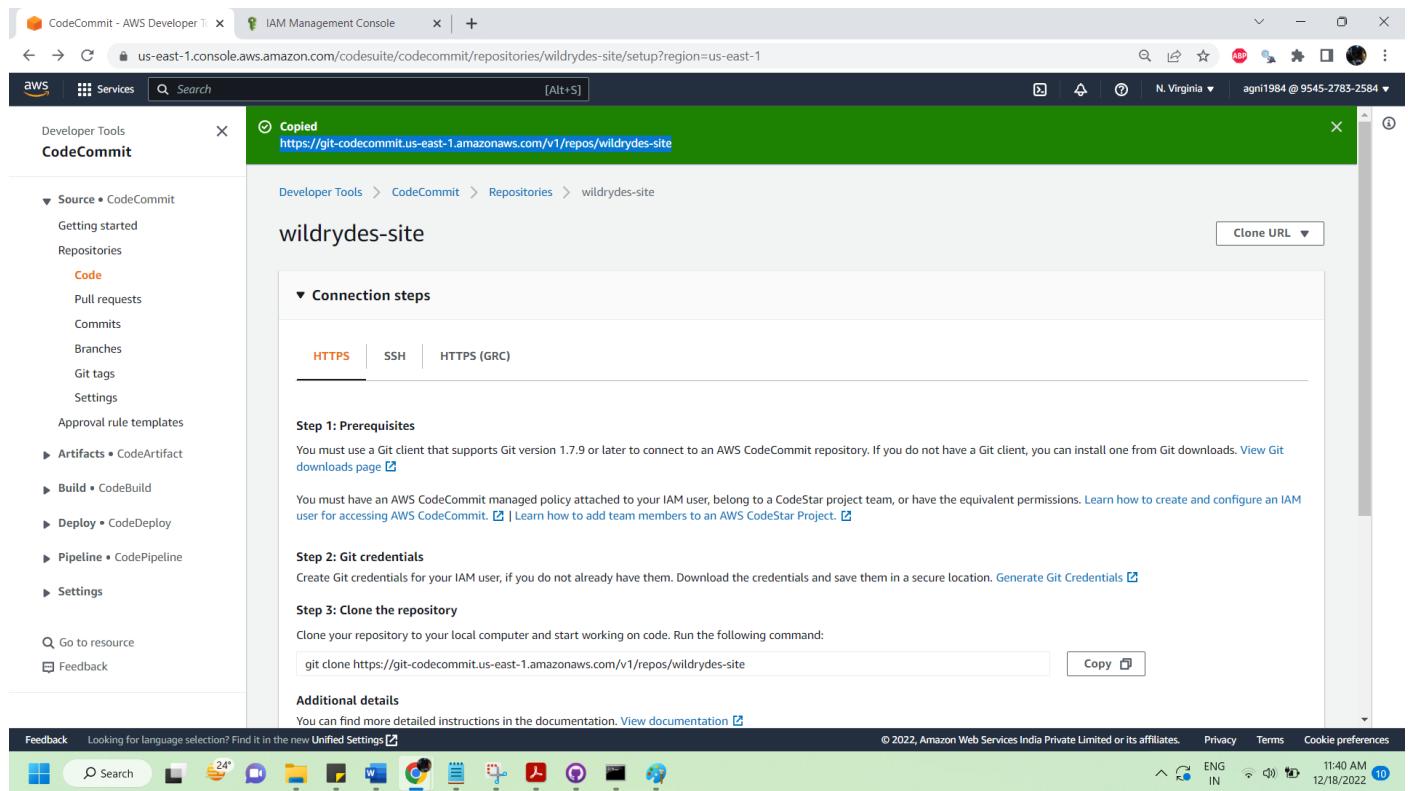
```
git clone https://git-codecommit.us-east-1.amazonaws.com/v1/repos/wildrydes-site
```

**Copy**

**Additional details**  
You can find more detailed instructions in the documentation. [View documentation](#)

Feedback Looking for language selection? Find it in the new [Unified Settings](#) © 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

ENG IN 11:37 AM 12/18/2022 10



The screenshot shows the AWS CodeCommit console. A message at the top indicates that the URL <https://git-codecommit.us-east-1.amazonaws.com/v1/repos/wildrydes-site> has been copied. The main page displays the repository details, including connection steps for HTTPS, SSH, and HTTPS (GRC). A command line interface (CLI) section provides the HTTPS URL for cloning the repository.

- From a terminal window run git clone and the HTTPS URL of the repository:

```
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3>git clone https://git-codecommit.us-east-1.amazonaws.com/v1/repos/wildrydes-site
Cloning into 'wildrydes-site'...
warning: You appear to have cloned an empty repository.
```

## Step 3: Populate the Git Repository

Copy the web site content from an existing publicly accessible S3 bucket's content to the repository.

- Change directory into our repository and copy the static files from S3:

```
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3>cd wildrydes-site
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>aws s3 cp
s3://wildrydes-us-east-1/WebApplication/1_StaticWebHosting/website ./ --recursive
```

```
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3>cd wildrydes-site  
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>aws s3 cp s3://wildrydes-us-east-1/WebApplication/1_StaticWebHosting/website ./ --recursive
```

### b. Commit the files to our Git service

```
$ git add .  
$ git commit -m 'new'  
$ git push
```

```
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>git push  
Enumerating objects: 95, done.  
Counting objects: 100% (95/95), done.  
Delta compression using up to 8 threads  
Compressing objects: 100% (94/94), done.  
Writing objects: 100% (95/95), 9.44 MiB | 1.93 MiB/s, done.  
Total 95 (delta 2), reused 0 (delta 0), pack-reused 0  
remote: Validating objects: 100%  
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/wildrydes-site  
 * [new branch] master -> master
```

## Step 4: Enable Web Hosting with the Amplify Console

AWS Amplify takes care of the work of setting up a place to store static web application code and provides a number of helpful capabilities to simplify both the lifecycle of that application as well as enable best practices. Follow the below steps to deploy code using AWS Amplify,

- Launch the Amplify Console page
- Click Get Started under Deploy with Amplify Console
- Go to New App on the top right and choose Host Web App
- Select CodeCommit under Get started with Amplify Hosting
- Select the Repository service provider used today and select Next
- From the dropdown select the Repository and Branch just create
- On the "Configure build settings" page leave all the defaults and select Next.
- On the "Review" page select Save and deploy
- Amplify Console to create the necessary resources and to deploy our code.
- Once completed, click on the site image to launch Wild Rydes site.

Screenshot of the AWS Amplify console showing the 'wildrydes-site' application. The left sidebar shows 'App settings' for 'wildrydes-site'. The main panel displays the homepage with a banner 'Learn how to get the most out of Amplify Hosting' and tabs for 'Hosting environments' and 'Backend environments'. Below is a table showing connected branches:

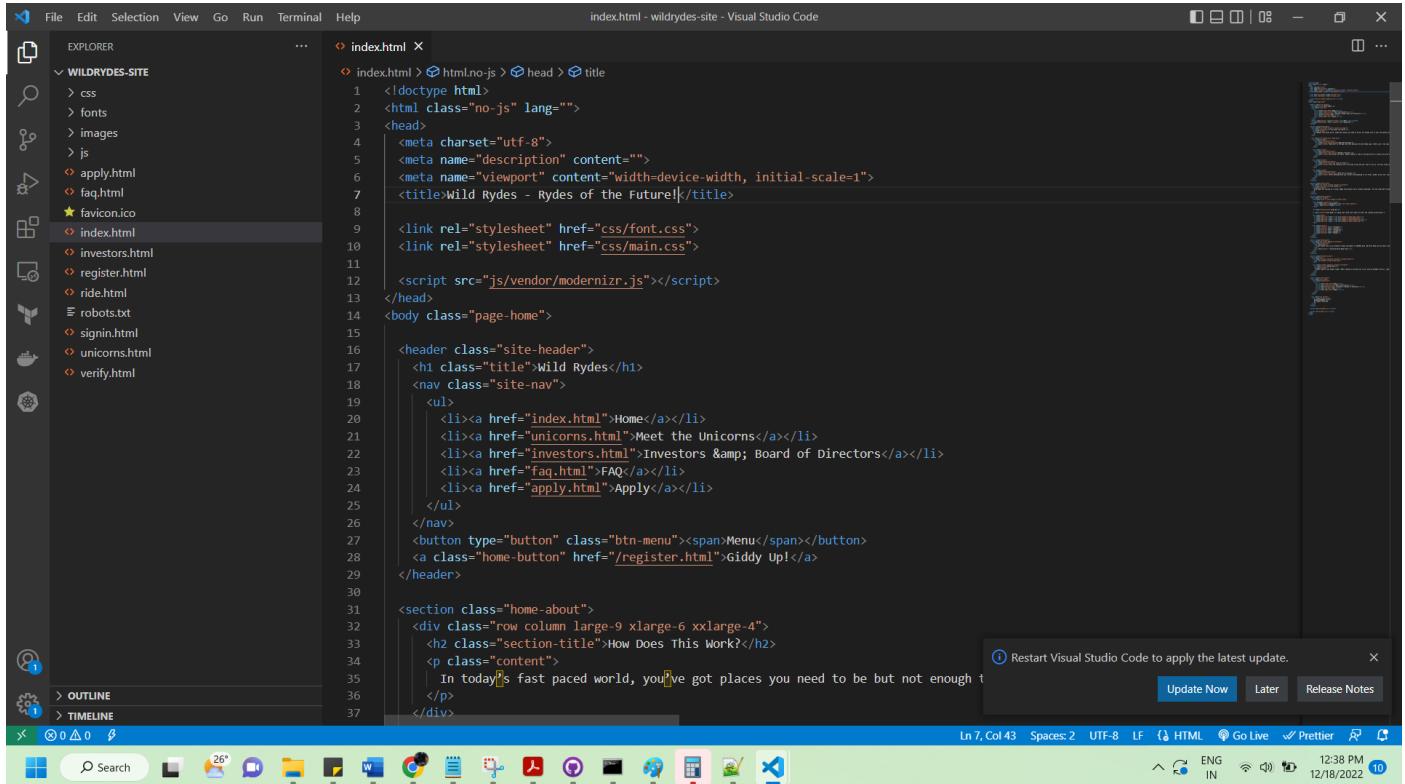
master	Continuous deploys set up (Edit)	Last deployment 12/18/2022, 12:32:31 PM	Last commit This is an autogenerated message   Auto-build   AWS CodeCommit - master	Previews Disabled
		<a href="https://master.d1mcs3d524z6sr.amplifyapp.com">https://master.d1mcs3d524z6sr.amplifyapp.com</a>		

Screenshot of a web browser showing the deployed 'wildrydes-site' at the URL <https://master.d1mcs3d524z6sr.amplifyapp.com>. The page features a vibrant, colorful collage of animals (a unicorn, a horse, a dog) against a background of mountains and a rainbow. The word 'Wild' is prominently displayed in large, stylized letters.

## Step 5: Modify site

The AWS Amplify Console will rebuild and redeploy the app when it detects changes to the connected repository. Make a change to the main page to test out this process.

- From local machine, open `wildryde-site/index.html` in visual studio and modify the title line so that it says: **<title>Wild Rydes - Rydes of the Future!</title>**



```

File Edit Selection View Go Run Terminal Help
index.html - wildrydes-site - Visual Studio Code
EXPLORE index.html
WILDRYDES-SITE
> css
> fonts
> images
> js
apply.html
faq.html
★ favicon.ico
index.html
investors.html
register.html
ride.html
robots.txt
signin.html
unicorns.html
verify.html
<title>Wild Rydes - Rydes of the Future!</title>
<link rel="stylesheet" href="css/font.css">
<link rel="stylesheet" href="css/main.css">
<script src="js/vendor/modernizr.js"></script>
</head>
<body class="page-home">
<header class="site-header">
<h1 class="title">Wild Rydes</h1>
<nav class="site-nav">
<ul>
<li><a href="index.html">Home</a></li>
<li><a href="unicorns.html">Meet the Unicorns</a></li>
<li><a href="investors.html">Investors & Board of Directors</a></li>
<li><a href="faq.html">FAQ</a></li>
<li><a href="apply.html">Apply</a></li>
</ul>
</nav>
<button type="button" class="btn-menu"><span>Menu</span></button>
<a class="home-button" href="/register.html">Giddy Up!</a>
</header>
<section class="home-about">
<div class="row column large-9 xlarge-6 xxlarge-4">
<h2 class="section-title">How Does This Work?</h2>
<p class="content">
In today's fast-paced world, you've got places you need to be but not enough t
</p>
</div>

```

Restart Visual Studio Code to apply the latest update.

Update Now Later Release Notes

Ln 7, Col 43 Spaces: 2 UTF-8 LF ⚡ Go Live ✅ Prettier ⌂ 12:38 PM 10 12/18/2022

- Save the file and commit to git repository again. Amplify Console will begin to build the site again soon after it notices the update to the repository.

```
$ git add index.html
$ git commit -m "updated title"
```

```
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>git add index.html
warning: in the working copy of 'index.html', LF will be replaced by CRLF the next time Git touches it
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>git commit -m "updated title"
[master 5438018] updated title
1 file changed, 1 insertion(+), 1 deletion(-)

E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 308 bytes | 308.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/wildrydes-site
  053dc44..5438018 master -> master
```

- b. Once completed, re-open the Wild Rydes site and notice the title change.



CodeCommit - AWS Developer Tools

User Pools - Amazon Cognito

IAM Management Console

AWS Amplify console

us-east-1.console.aws.amazon.com/amplify/home?region=us-east-1#/d1mcs3d524z6sr

aws Services Search [Alt+S] N. Virginia agni1984 @ 9545-2783-2584

## AWS Amplify

All apps wildrydes-site

### wildrydes-site

The app homepage lists all deployed frontend and backend environments.

▶ Learn how to get the most out of Amplify Hosting 0 of 5 steps complete

App settings

- General
- Amplify Studio settings
- Domain management
- Build settings
- Previews
- Notifications
- Environment variables
- Access control
- Monitoring
- Rewrites and redirects
- Custom headers

Documentation Support

Actions

Hosting environments Backend environments

This tab lists all connected branches, select a branch to view build details. Connect branch

**master**  
Continuous deploys set up (Edit)

  
<https://master...amplifyapp.com>

Last deployment  
12/18/2022, 12:39:06 PM

Last commit  
Please visit AWS CodeCommit Co... | 5438018 | AWS CodeCommit - master

Provision Build Deploy

Previews Disabled

Feedback Looking for language selection? Find it in the new Unified Settings

© 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

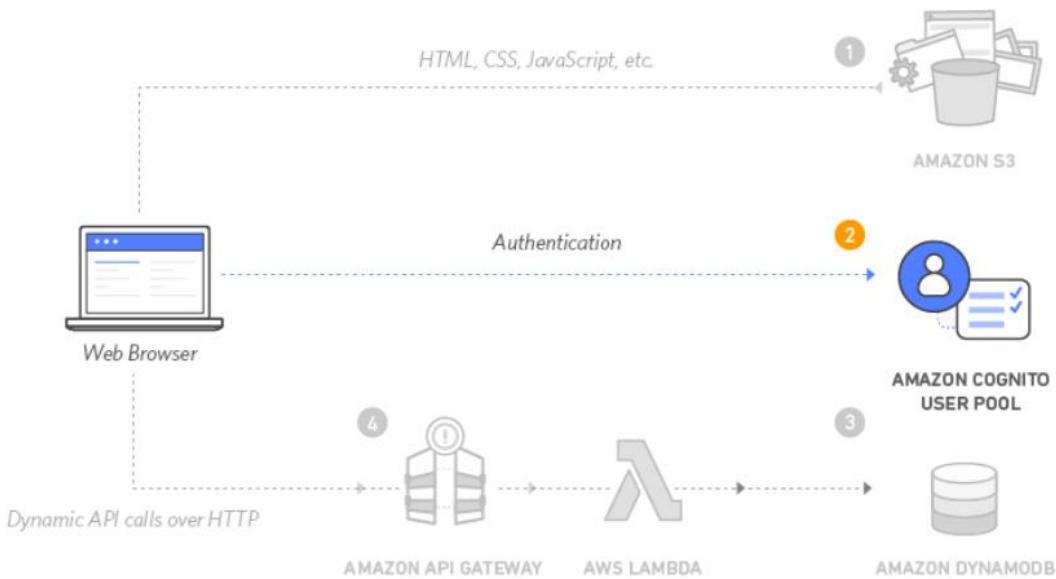
ENG IN 12:51 PM 12/18/2022 10

The image shows a vibrant, dynamic landing page for 'Wild Rydes'. The background features a composite of several horse-related scenes: a close-up of a white horse's head, a horse running through a lush green forest, and a horse leaping over a rocky, orange-red mountain ridge. The 'Wild Rydes' logo is prominently displayed in the center. The word 'Wild' is written in a large, flowing, white font with orange and red outlines. Below it, the word 'RYDES' is in a bold, white, sans-serif font. A small, rounded rectangular button with the text 'GIDDY UP!' in white is positioned below the main title. The overall aesthetic is energetic and modern, with a focus on the beauty and power of horses.

## Module 2: User Management

In this module we will create an Amazon Cognito user pool to manage users' accounts. We'll deploy pages that enable customers to register as a new user, verify their email address, and sign into the site.

## Architectural Overview



When users visit website, they will first register a new user account by providing an email address and password to register.

After users submit their registration, Amazon Cognito will send a confirmation email with a verification code to the address they provided. To confirm their account, users will return to our site and enter their email address and the verification code they received. We can also confirm user accounts using the Amazon Cognito console with a fake email address for testing.

After users have a confirmed account (either using the email verification process or a manual confirmation through the console), they will be able to sign in. When users sign in, they enter their username (or email) and password. A JavaScript function then communicates with Amazon Cognito, authenticates using the Secure Remote Password protocol (SRP), and receives back a set of JSON Web Tokens (JWT). The JWTs contain claims about the identity of the user and will be used in the next module to authenticate against the RESTful API we build with Amazon API Gateway.

## Step 1: Create an Amazon Cognito User Pool

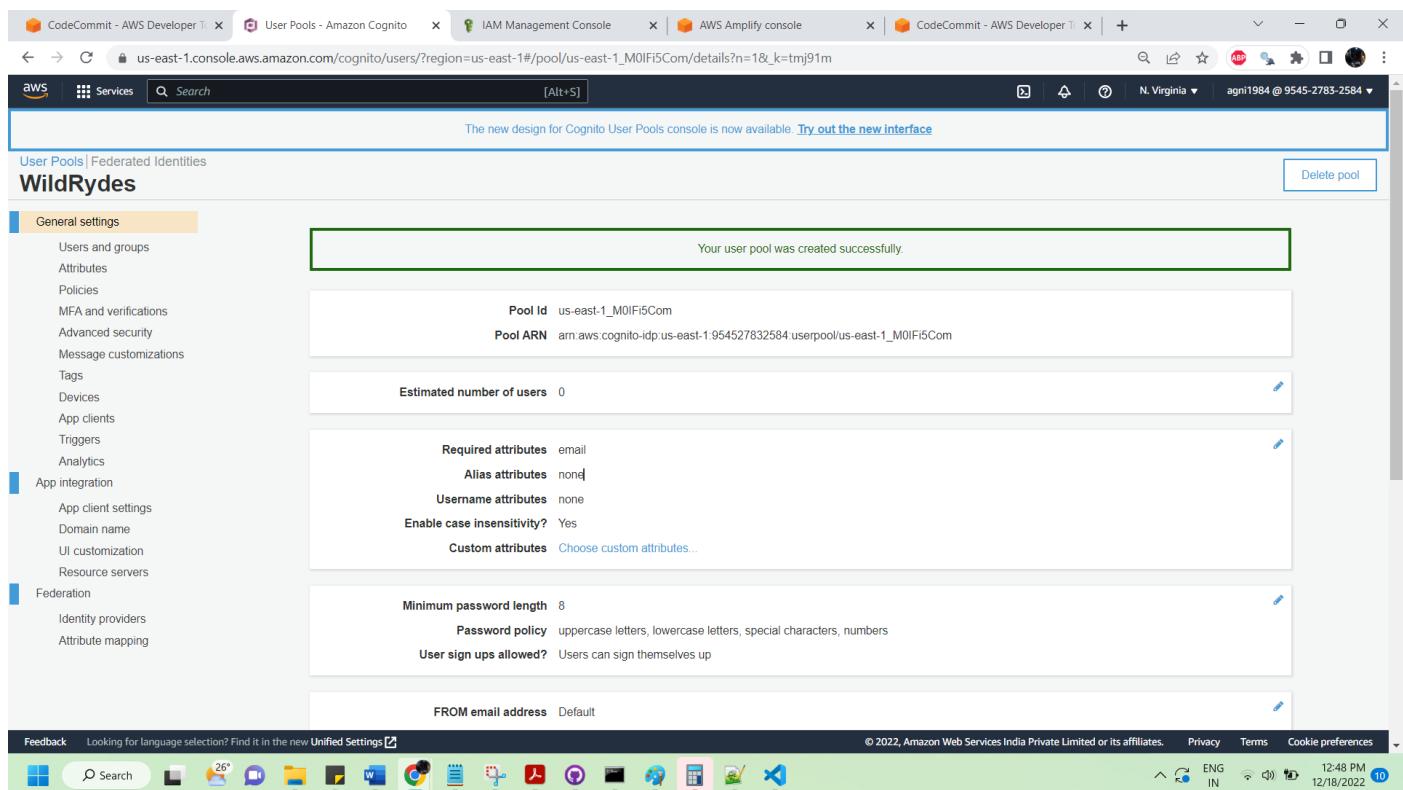
Amazon Cognito provides two different mechanisms for authenticating users. We can use Cognito User Pools to add sign-up and sign-in functionality to our application or use Cognito Identity Pools to authenticate

users through social identity providers such as Facebook, Twitter, or Amazon, with SAML identity solutions, or by using our own identity system. In this project, we are using, user pool as the backend for the provided registration and sign-in pages.

- From the AWS Console click Services then select Cognito under Mobile Services
- Choose Manage our User Pools
- Choose Create a User Pool
- Provide a name for the user pool as **WildRydes**, then select Review Defaults
- On the review page, click Create pool
- Note the Pool Id on the Pool details page of our newly created user pool.

**Pool Id us-east-1\_M0IFI5Com**

**Pool ARN arn:aws:cognito-idp:us-east-1:954527832584:userpool/us-east-1\_M0IFI5Com**



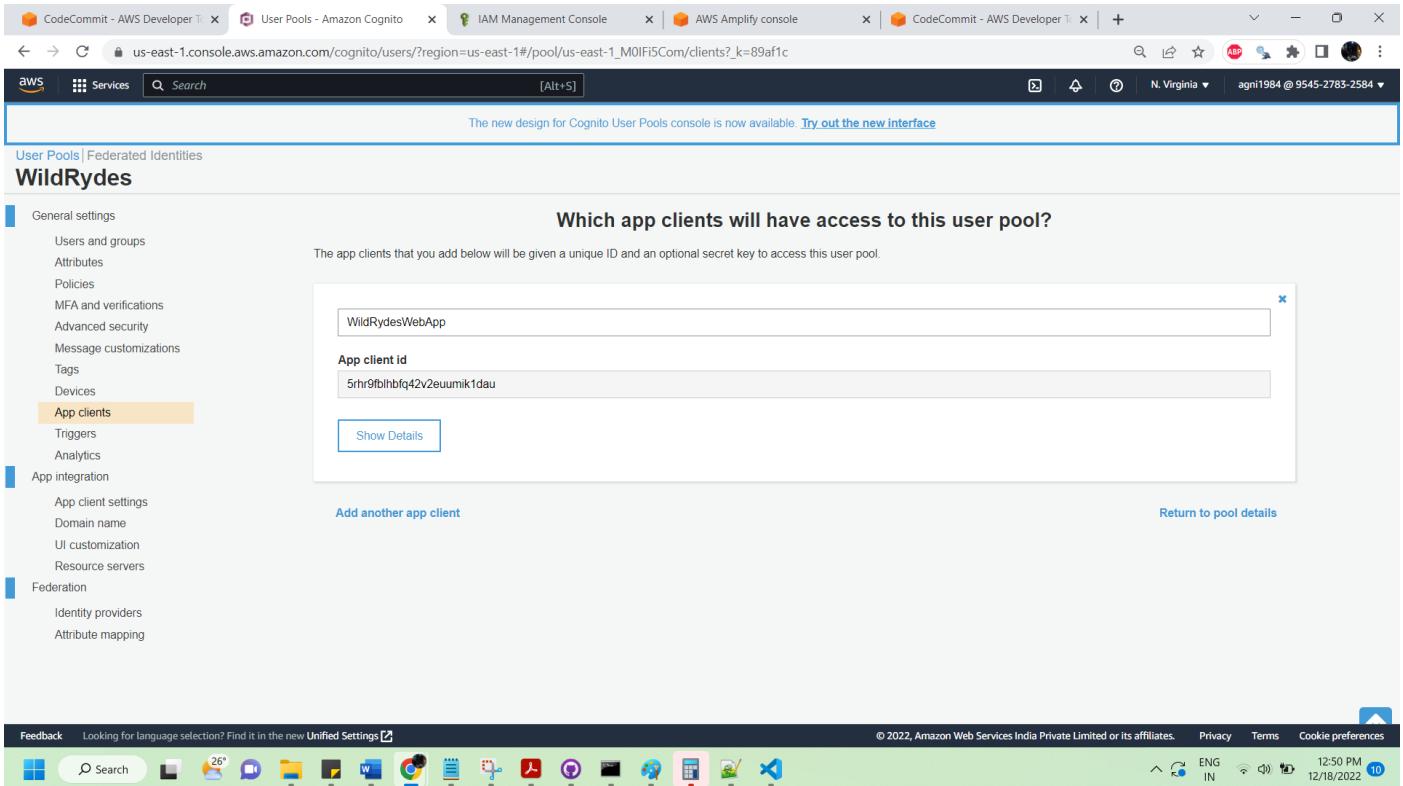
## Step 2: Add an App to User Pool

Follow the below step to add 'App client' to 'user pool',

- From the Pool Details page for the user pool, select App clients from the left General Settings section in the navigation bar.
- Choose Add an app client.
- Give the app client a name such as **WildRydesWebApp**.

- Uncheck the Generate client secret option. Client secrets aren't currently supported for use with browser-based applications. Keep the default settings for the other options.
- Choose Create app client.
- Note the App client id for the newly created application.

**app client id: 5rhr9fbhbfq42v2euumik1dau**



The screenshot shows the AWS Cognito User Pools console. The URL in the address bar is `us-east-1.console.aws.amazon.com/cognito/users/?region=us-east-1#/pool/us-east-1_M0lFi5Com/clients?_k=89af1c`. The sidebar on the left has several tabs: General settings, Users and groups, Attributes, Policies, MFA and verifications, Advanced security, Message customizations, Tags, Devices, **App clients** (which is selected), Triggers, Analytics, App integration, and Federation. The main content area is titled "Which app clients will have access to this user pool?" and contains a sub-section for "WildRydesWebApp". It shows the "App client id" field with the value "5rhr9fbhbfq42v2euumik1dau". There is a "Show Details" button and a "Add another app client" link. At the bottom right of the main content area is a "Return to pool details" link. The status bar at the bottom of the browser window shows "Feedback Looking for language selection? Find it in the new Unified Settings" and "© 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences". The system tray on the right shows icons for search, file explorer, task manager, and other system utilities, along with a battery level of 26% and a date/time of 12:50 PM 12/18/2022.

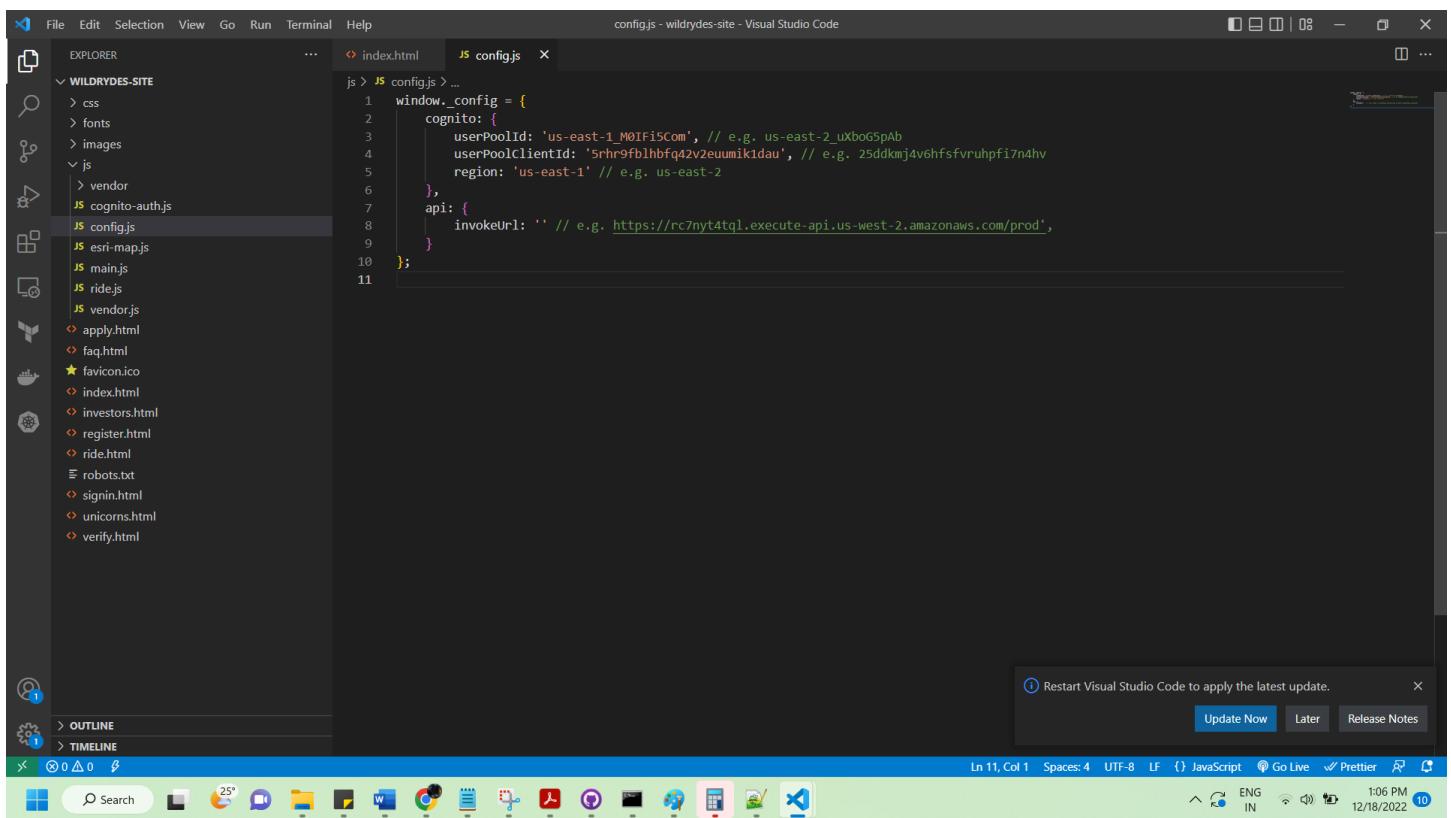
### Step 3: Update the website config

The `/js/config.js` file contains settings for the user pool ID, app client ID and Region.

**Pool Id us-east-1\_M0lFi5Com**

**app client id: 5rhr9fbhbfq42v2euumik1dau**

**Region: us-east-1**



```

File Edit Selection View Go Run Terminal Help
config.js - wildrydes-site - Visual Studio Code
EXPLORER JS config.js
WILDRYDES-SITE ...
> CSS
> fonts
> images
> JS vendor
JS config.js
JS config.js
JS esri-map.js
JS main.js
JS ride.js
JS vendor.js
apply.html
faq.html
★ favicon.ico
index.html
investors.html
register.html
ride.html
robots.txt
signin.html
unicorns.html
verify.html

```

The config.js file contains the following code:

```

js > JS config.js > ...
1 window._config = {
2   cognito: {
3     userPoolId: 'us-east-1_M0lFisCom', // e.g. us-east-2_uxboG5pAb
4     userPoolClientId: '5rhr9fb1hbqf42vzeumkidaU', // e.g. 25ddkmj4vhfsfvruhpfi7n4hv
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

```

Restart Visual Studio Code to apply the latest update.

Update Now Later Release Notes

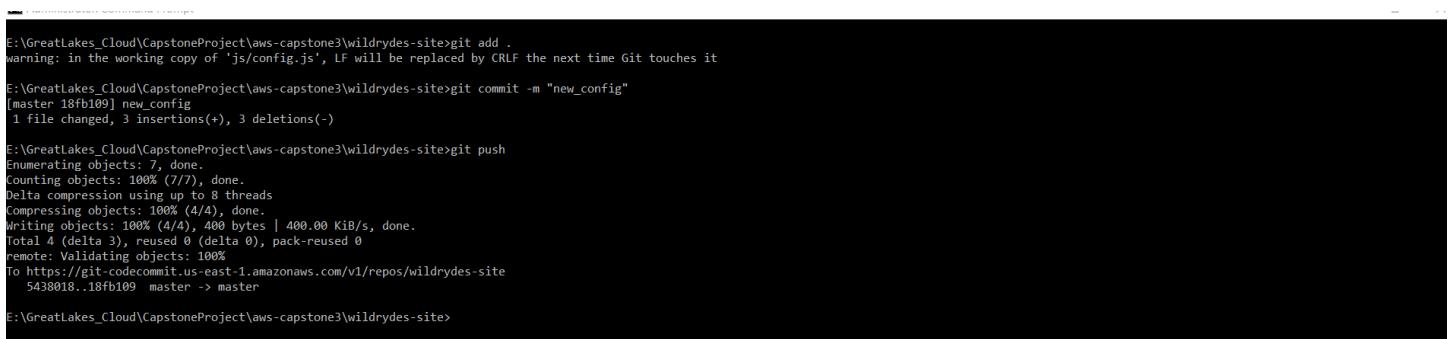
Ln 11, Col 1 Spaces: 4 UTF-8 LF {} JavaScript Go Live ✅ Prettier ⌂ ENG IN 12:06 PM 10 12/18/2022

Save the modified file and push it to our Git repository to have it automatically deploy to Amplify Console.

```

$ git add .
$ git commit -m "new_config"
$ git push

```



```

E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>git add .
warning: in the working copy of 'js/config.js', LF will be replaced by CRLF the next time Git touches it
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>git commit -m "new_config"
[master 18fb109] new_config
 1 file changed, 3 insertions(+), 3 deletions(-)

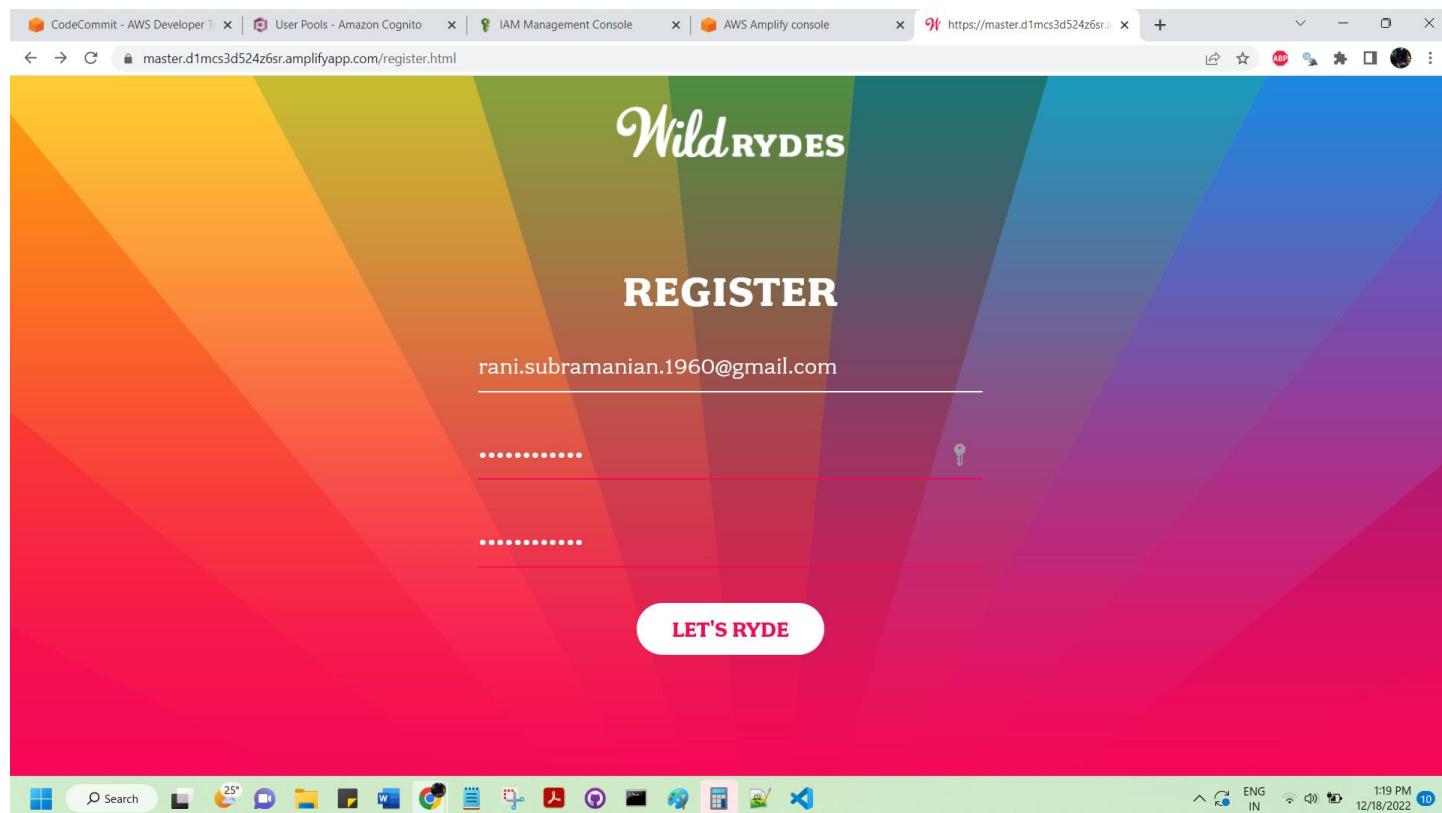
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 400 bytes | 400.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/wildrydes-site
 5438018..18fb109 master -> master
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>

```

## Step 4: Validate the Implementation

- Visit /register.html under the website domain, or choose the Giddy Up! button on the homepage of our site.

- Complete the registration form and choose Let's Ryde. We can use our own email or enter a fake email. Make sure to choose a password that contains at least one upper-case letter, a number, and a special character. Don't forget the password we entered for later. We should see an alert that confirms that our user has been created.
- Confirm our new user using one of the two following methods.
- For valid email address,
  - If we used an email address, we can complete the account verification process by visiting /verify.html under our website domain and entering the verification code that is emailed to us.





CodeCommit - AWS Developer Tools | User Pools - Amazon Cognito | IAM Management Console | AWS Amplify console | https://master.d1mcs3d524z6sr.a...

The new design for Cognito User Pools console is now available. Try out the new interface.

User Pools | Federated Identities | WildRydes

General settings

- Users and groups
- Attributes
- Policies
- MFA and verifications
- Advanced security
- Message customizations
- Tags
- Devices
- App clients
- Triggers
- Analytics

App integration

- App client settings
- Domain name
- UI customization
- Resource servers

Federation

- Identity providers
- Attribute mapping

Users Groups

Username	Enabled	Account status	Email	Email verified	Phone number verified	Updated	Created
rani.subramanian.1960-at-gmail.com	Enabled	UNCONFIRMED	rani.subramanian.1960@gmail.com	false	-	Dec 18, 2022 7:48:56 AM	Dec 18, 2022 7:48:56 AM
sample.102-at-gmail.com	Enabled	CONFIRMED	sample.102@gmail.com	false	-	Dec 18, 2022 7:45:36 AM	Dec 18, 2022 7:44:36 AM

Feedback Looking for language selection? Find it in the new Unified Settings

© 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

Inbox (7,844) - agni1984@gmail.com | Your verification code - rani.subramanian.1960@gmail.com | +

1:20 PM 12/18/2022

Gmail

Compose

Inbox 164

- Starred
- Snoozed
- Sent
- Drafts 2
- More

Labels +

Your verification code

no-reply@verificationemail.com to me

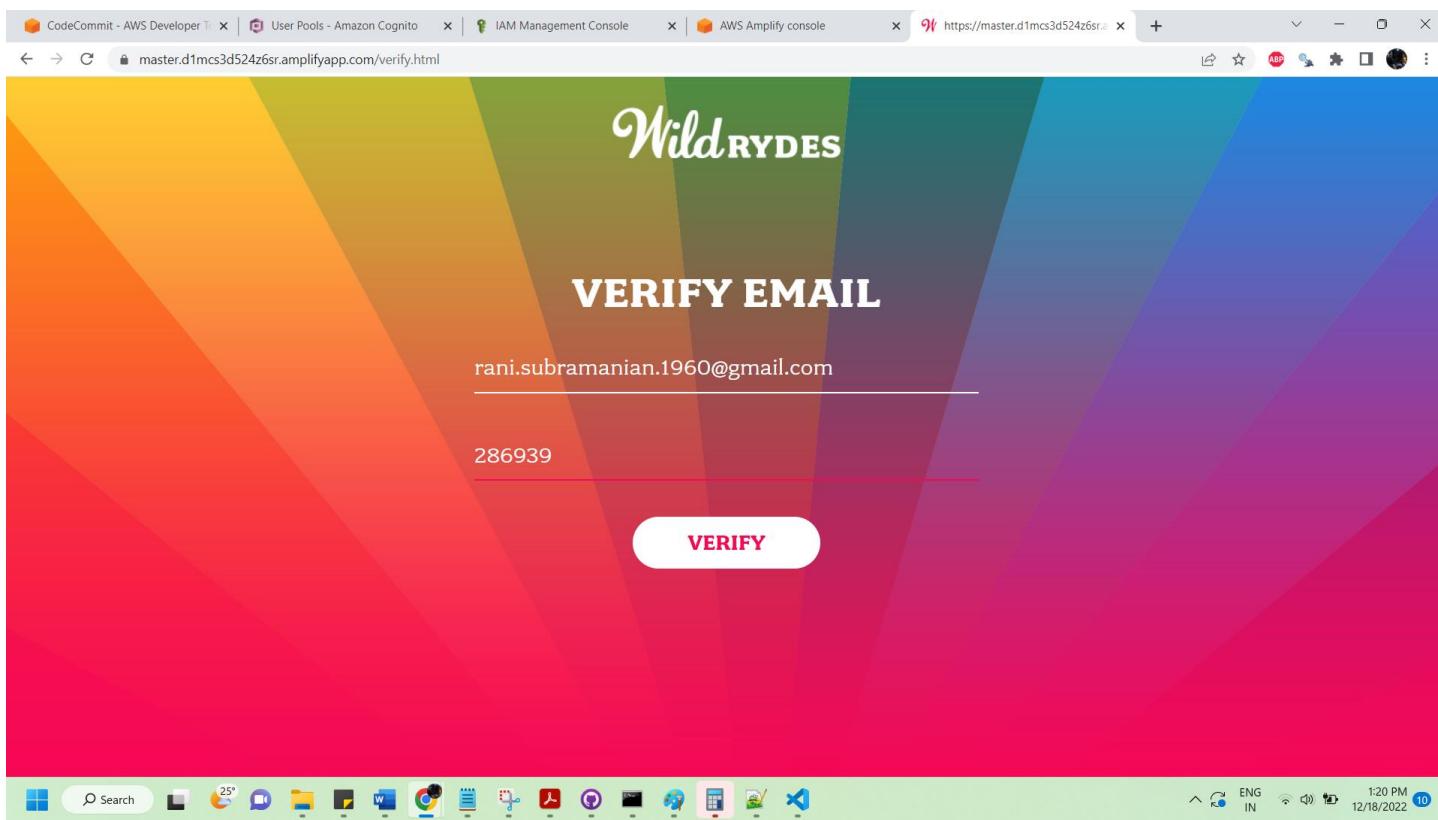
Your confirmation code is 286939

Reply Forward

1:18 PM (0 minutes ago)

1 of 261

ENG IN 1:19 PM 12/18/2022



CodeCommit - AWS Developer | User Pools - Amazon Cognito | IAM Management Console | AWS Amplify console | https://master.d1mcs3d524z6sr.amplifyapp.com/verify.html

25° N. Virginia ENG IN 12/18/2022 10

CodeCommit - AWS Developer | User Pools - Amazon Cognito | IAM Management Console | AWS Amplify console | https://master.d1mcs3d524z6sr.amplifyapp.com/verify.html

Services Search [Alt+S] N. Virginia ENG IN 12/18/2022 10

The new design for Cognito User Pools console is now available. Try out the new interface.

User Pools | Federated identities

## WildRydes

General settings

- Users and groups**
- Attributes
- Policies
- MFA and verifications
- Advanced security
- Message customizations
- Tags
- Devices
- App clients
- Triggers
- Analytics

App integration

- App client settings
- Domain name
- UI customization
- Resource servers

Federation

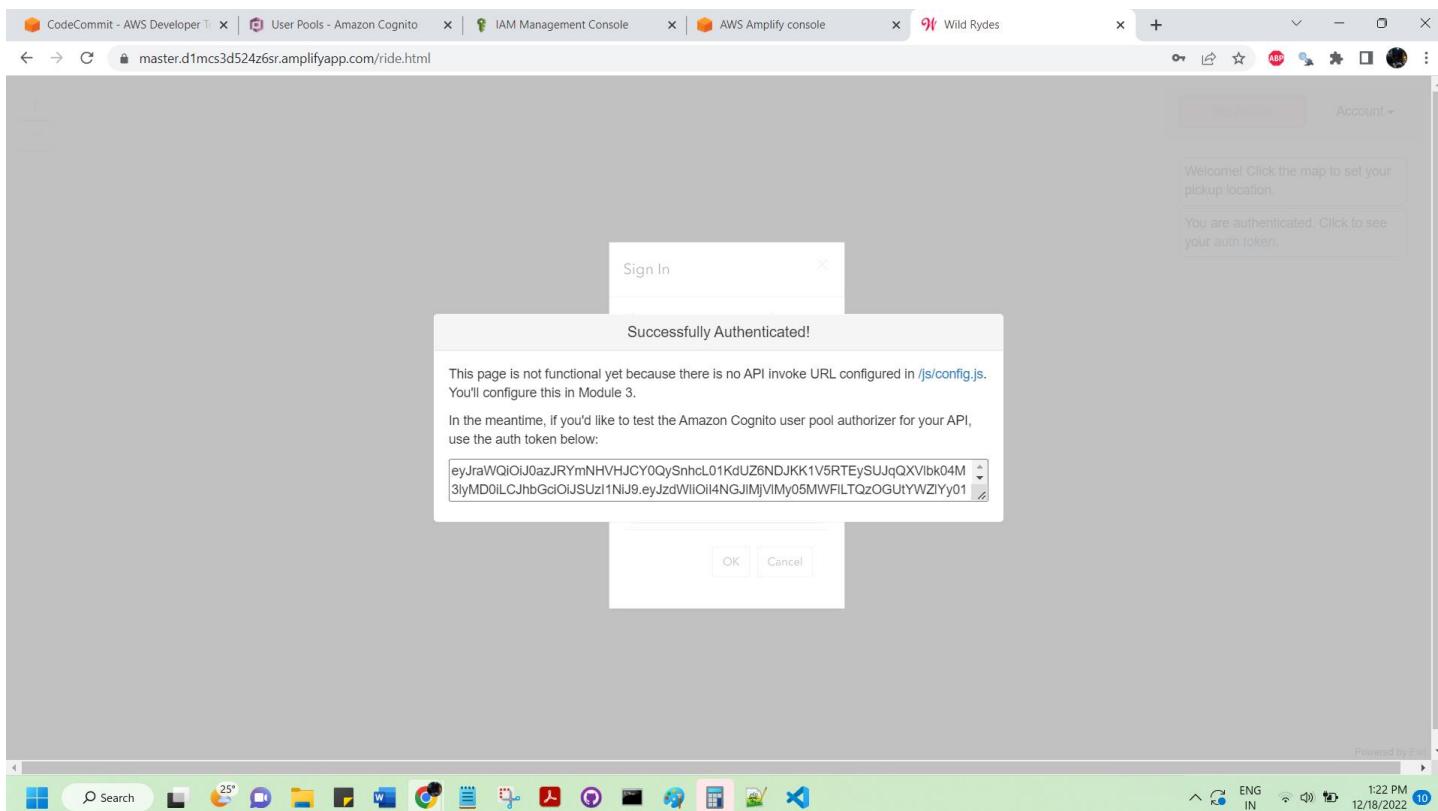
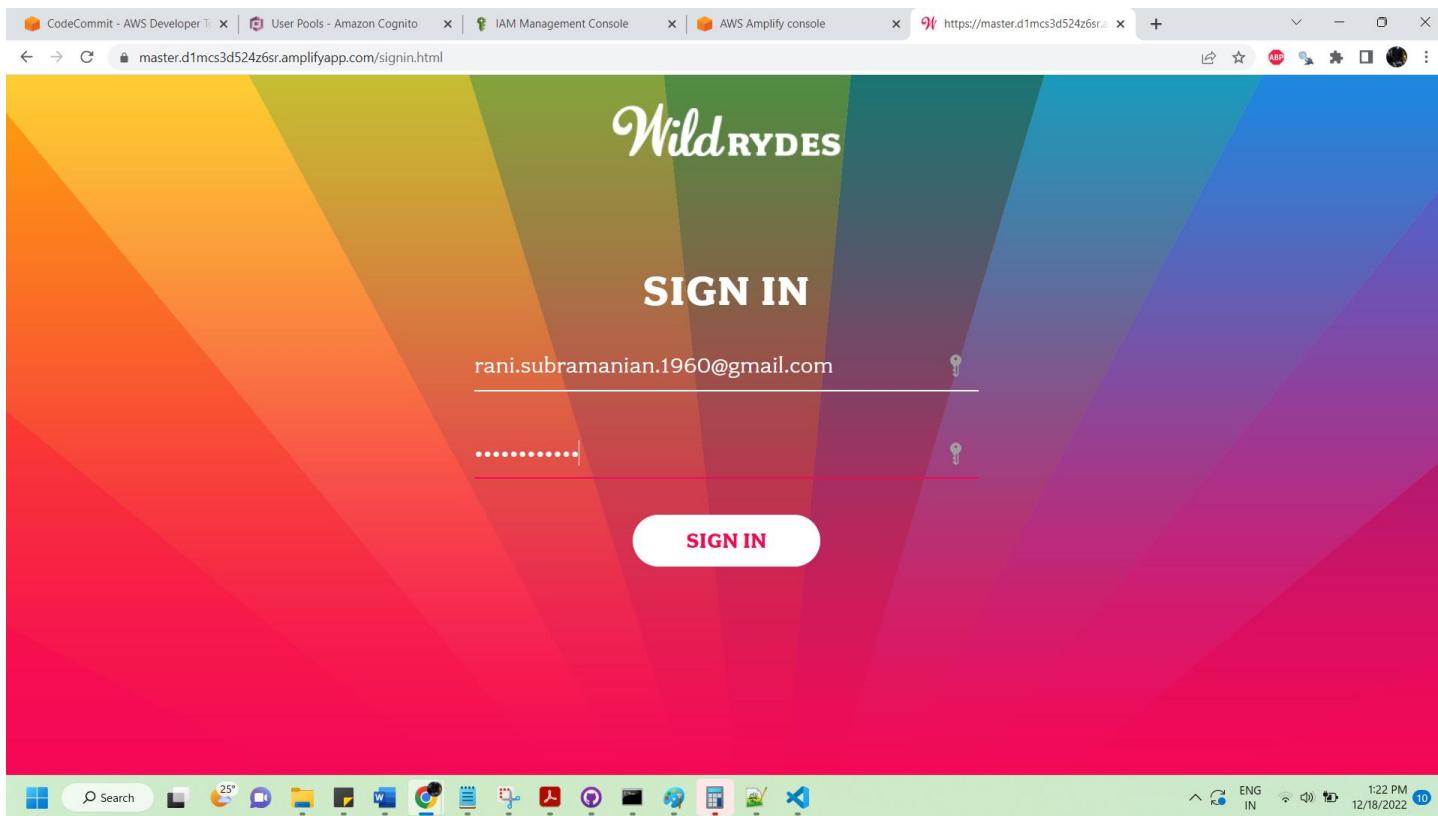
- Identity providers
- Attribute mapping

Feedback Looking for language selection? Find it in the new Unified Settings

© 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

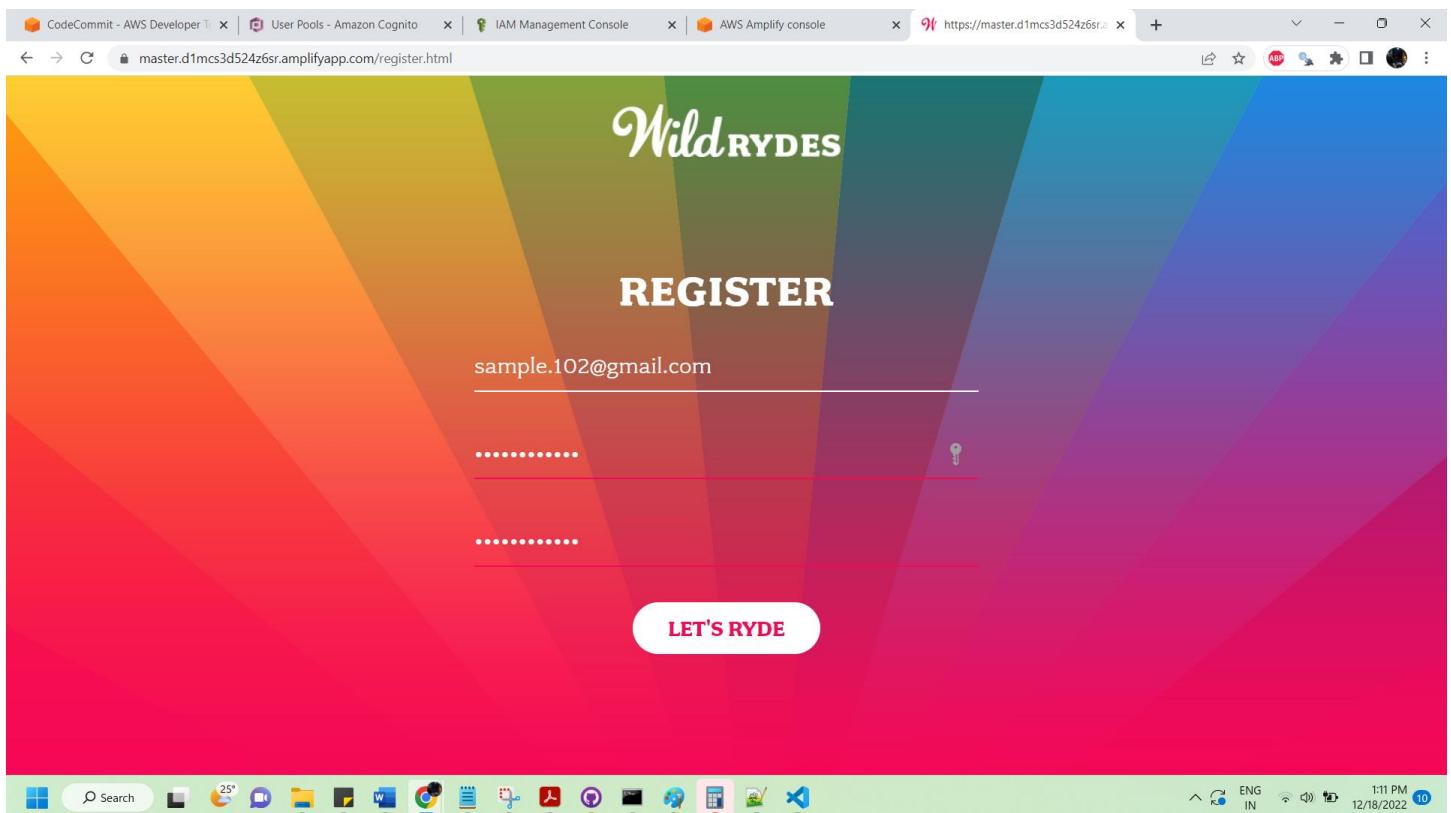
25° N. Virginia ENG IN 12/18/2022 10

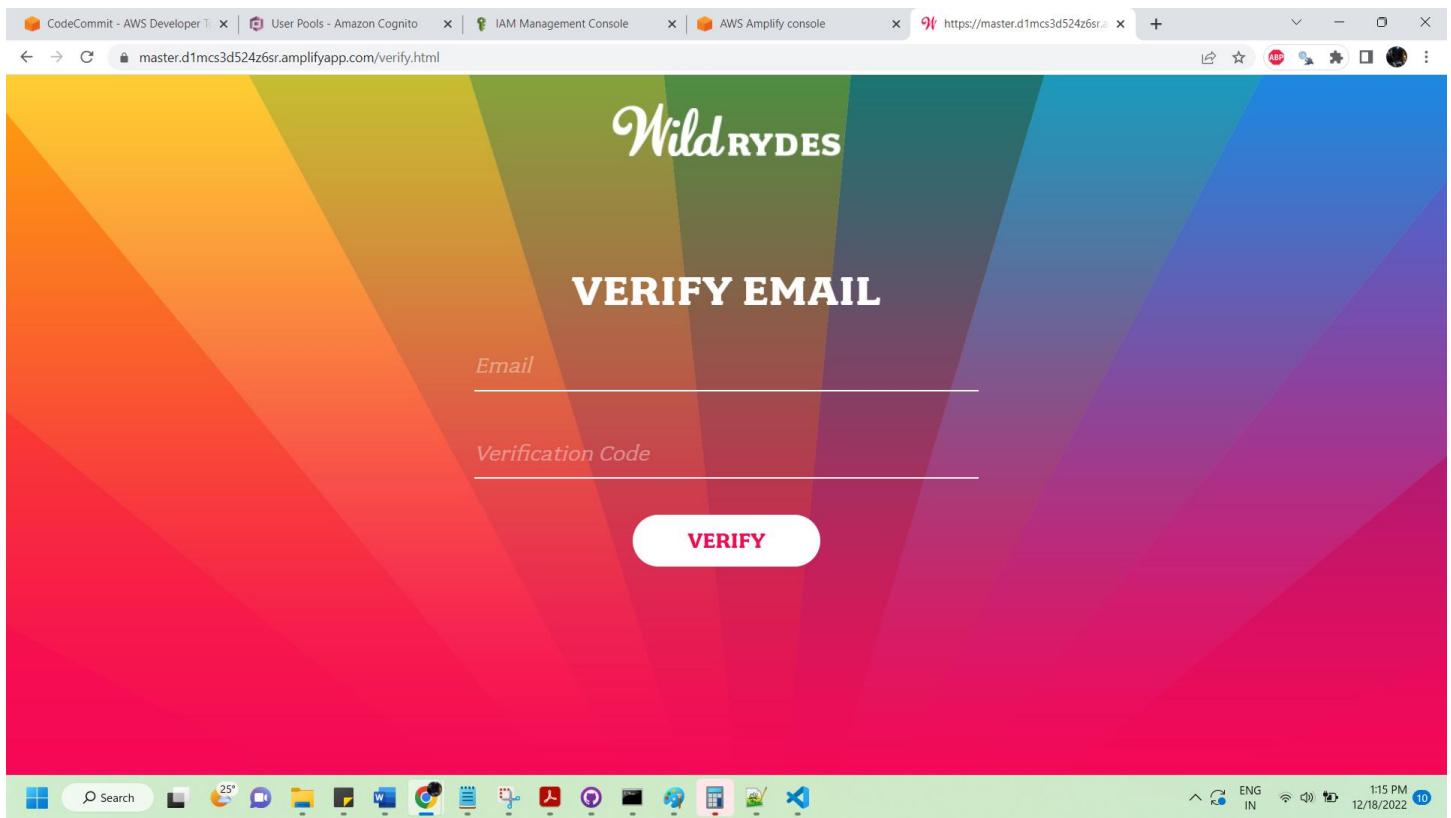
Username	Enabled	Account status	Email	Email verified	Phone number verified	Updated	Created
rani.subramanian.1960-at-gmail.com	Enabled	CONFIRMED	rani.subramanian.1960@gmail.com	true	-	Dec 18, 2022 7:48:56 AM	Dec 18, 2022 7:48:56 AM
sample.102-at-gmail.com	Enabled	CONFIRMED	sample.102@gmail.com	false	-	Dec 18, 2022 7:45:36 AM	Dec 18, 2022 7:44:36 AM



- For Dummy email address,
  - If dummy email address is used, confirm the user manually through the Cognito console.

- From the AWS console, click Services then select Cognito under Security, Identity & Compliance.
- Choose Manage our User Pools
- Select the WildRydes user pool and click Users and groups in the left navigation bar.
- User corresponding to the email address that we submitted through the registration page should be seen. Choose that username to view the user detail page.
- Choose Confirm user to finalize the account creation process.
- After confirming the new user using either the /verify.html page or the Cognito console, visit /signin.html and log in using the email address and password entered during the registration step.
- If successful , we should be redirected to /ride.html. we should see a notification that the API is not configured.





The new design for Cognito User Pools console is now available. [Try out the new interface](#)

User Pools | Federated Identities

## WildRydes

Username	Enabled	Account status	Email	Email verified	Phone number verified	Updated	Created
sample.102-at-gmail.com	Enabled	UNCONFIRMED	sample.102@gmail.com	false	-	Dec 18, 2022 7:44:36 AM	Dec 18, 2022 7:44:36 AM

**General settings**

- Users and groups** (selected)
- Attributes
- Policies
- MFA and verifications
- Advanced security
- Message customizations
- Tags
- Devices
- App clients
- Triggers
- Analytics

**App integration**

- App client settings
- Domain name
- UI customization
- Resource servers

**Federation**

- Identity providers
- Attribute mapping

The new design for Cognito User Pools console is now available. [Try out the new interface](#)

[User Pools](#) | Federated Identities  
**WildRydes**

General settings

- [Users and groups](#)
- [Attributes](#)
- [Policies](#)
- [MFA and verifications](#)
- [Advanced security](#)
- [Message customizations](#)
- [Tags](#)
- [Devices](#)
- [App clients](#)
- [Triggers](#)
- [Analytics](#)

App integration

- [App client settings](#)
- [Domain name](#)
- [UI customization](#)
- [Resource servers](#)

Federation

- [Identity providers](#)
- [Attribute mapping](#)

**Users > sample.102-at-gmail.com**

Add to group   Confirm user   Enable SMS MFA   Disable user

Groups	-
Account Status	Enabled / UNCONFIRMED
SMS MFA Status	Disabled
Last Modified	Dec 18, 2022 7:44:36 AM
Created	Dec 18, 2022 7:44:36 AM
sub	48531483-3749-423e-ac00-13cff5219be2
email_verified	false
email	sample.102@gmail.com

Device Key	Name	Last IP	Remembered	SDK	Last Seen

Feedback Looking for language selection? Find it in the new [Unified Settings](#)

© 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

ENG IN 12:15 PM 10 12/18/2022

The new design for Cognito User Pools console is now available. [Try out the new interface](#)

[User Pools](#) | Federated Identities  
**WildRydes**

General settings

- [Users and groups](#)
- [Attributes](#)
- [Policies](#)
- [MFA and verifications](#)
- [Advanced security](#)
- [Message customizations](#)
- [Tags](#)
- [Devices](#)
- [App clients](#)
- [Triggers](#)
- [Analytics](#)

App integration

- [App client settings](#)
- [Domain name](#)
- [UI customization](#)
- [Resource servers](#)

Federation

- [Identity providers](#)
- [Attribute mapping](#)

**Users > sample.102-at-gmail.com**

Add to group   Enable SMS MFA   Disable user

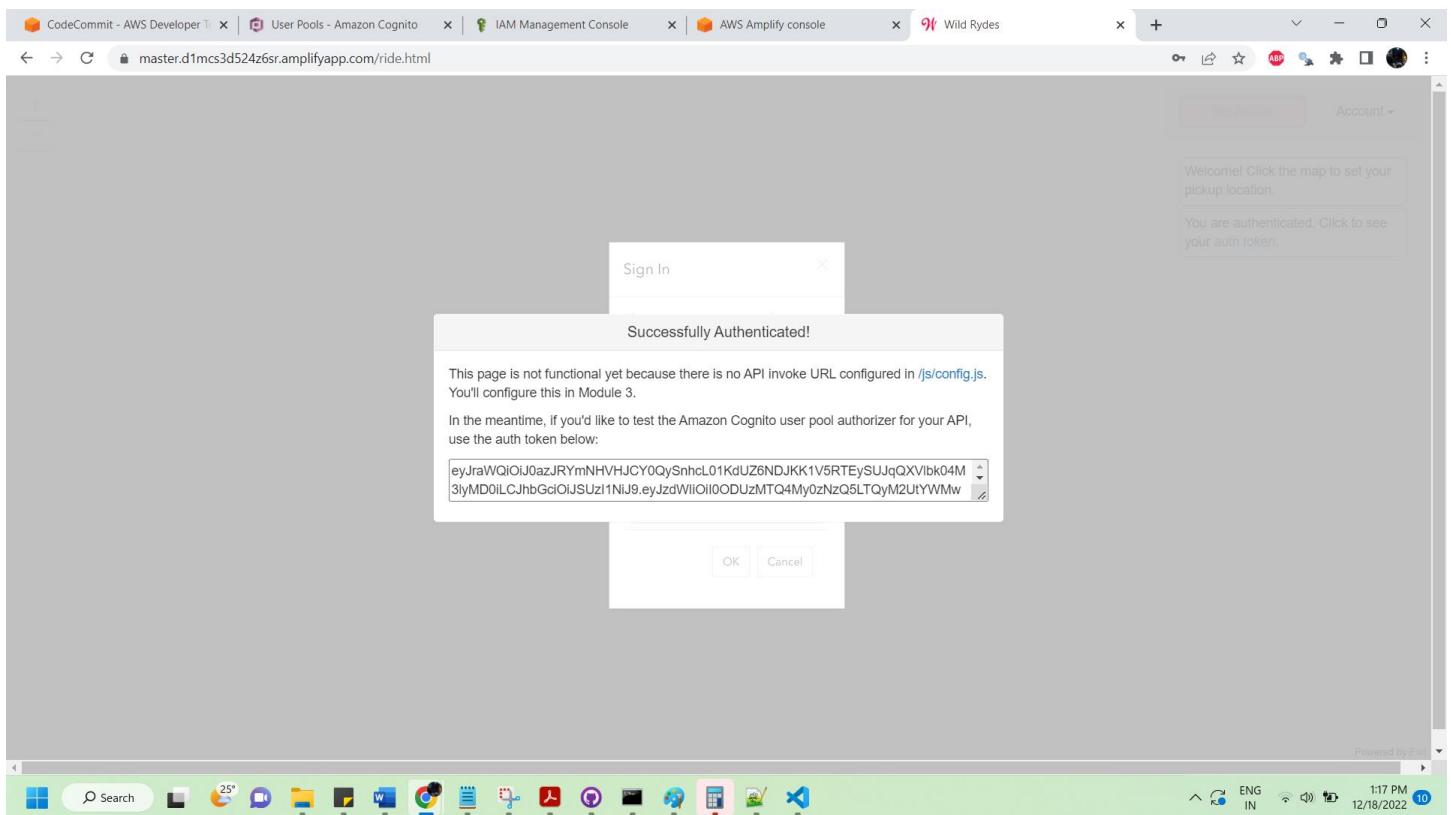
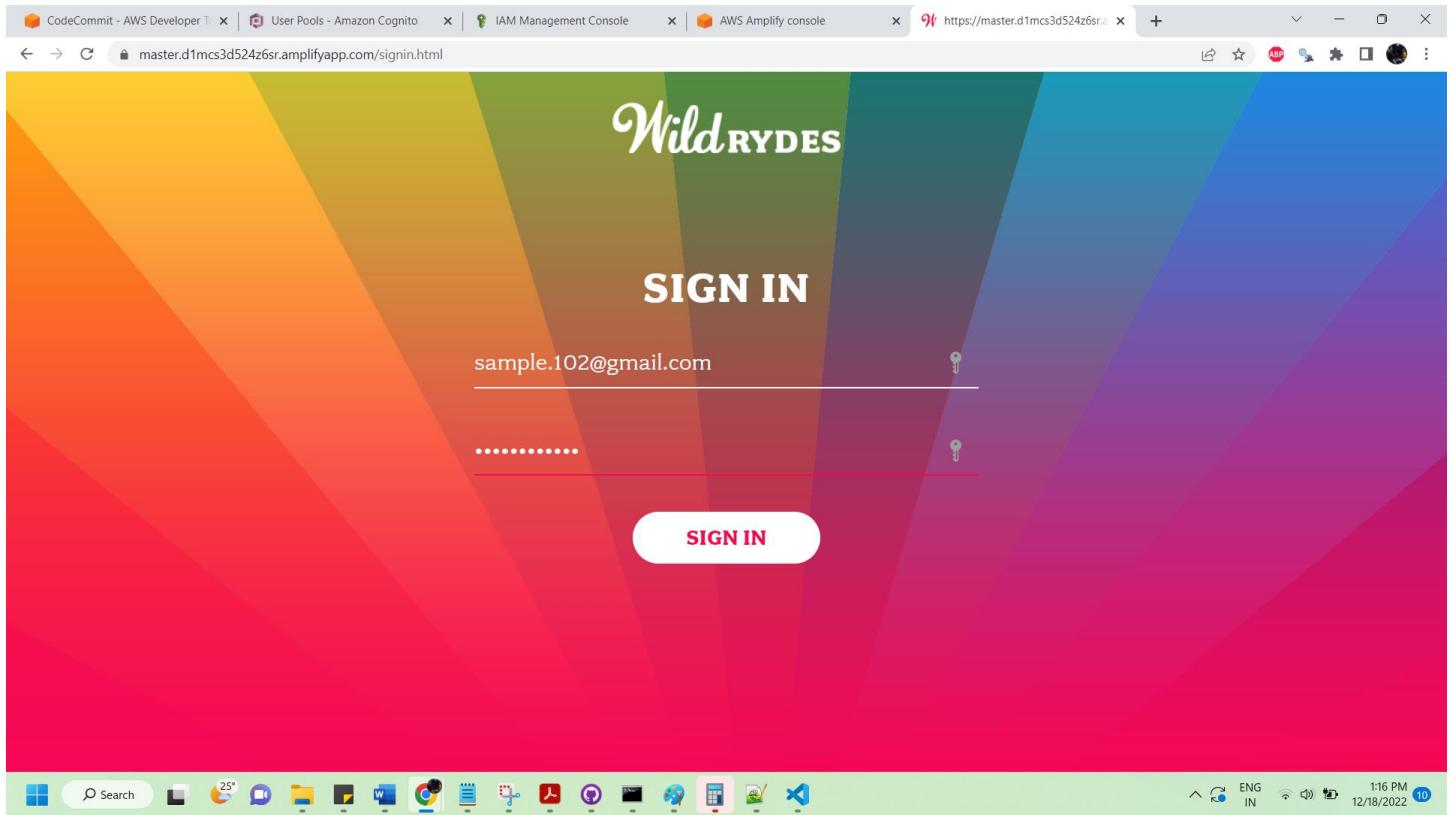
Groups	-
Account Status	Enabled / CONFIRMED
SMS MFA Status	Disabled
Last Modified	Dec 18, 2022 7:45:36 AM
Created	Dec 18, 2022 7:44:36 AM
sub	48531483-3749-423e-ac00-13cff5219be2
email_verified	false
email	sample.102@gmail.com

Device Key	Name	Last IP	Remembered	SDK	Last Seen

Feedback Looking for language selection? Find it in the new [Unified Settings](#)

© 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

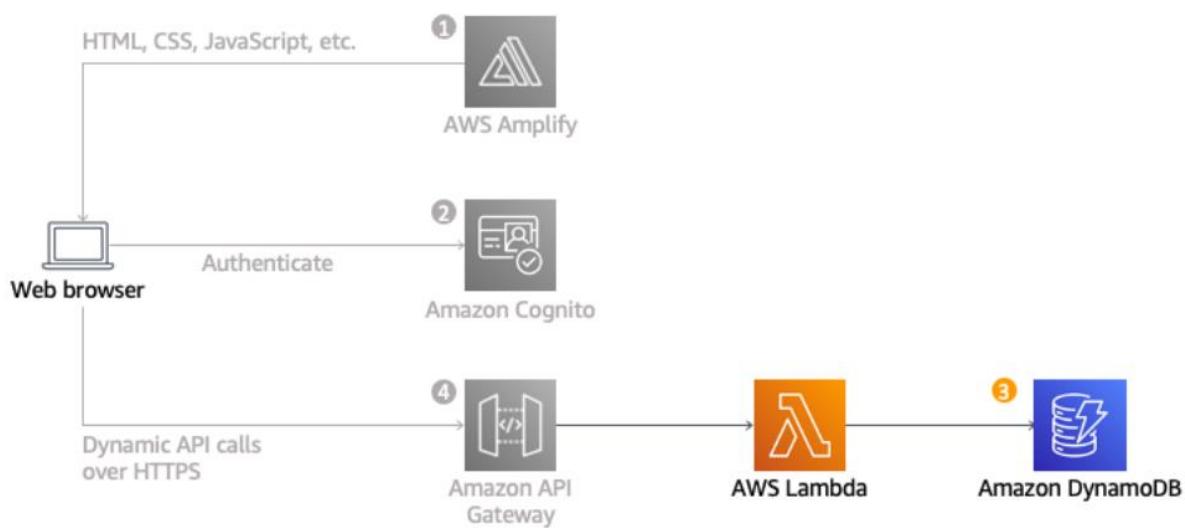
ENG IN 12:15 PM 10 12/18/2022



## Module 3: Serverless Backend

In this module, AWS Lambda and Amazon DynamoDB is used to build a backend process for handling requests for the web application. The browser application that we deployed in the first module allows users to request that a unicorn be sent to a location of their choice. To fulfill those requests, the JavaScript running in the browser will need to invoke a service running in the cloud.

### Architectural Overview



We are implementing a Lambda function that will be invoked each time a user requests a unicorn. The function will select a unicorn from the fleet, record the request in a DynamoDB table, and then respond to the frontend application with details about the unicorn being dispatched.

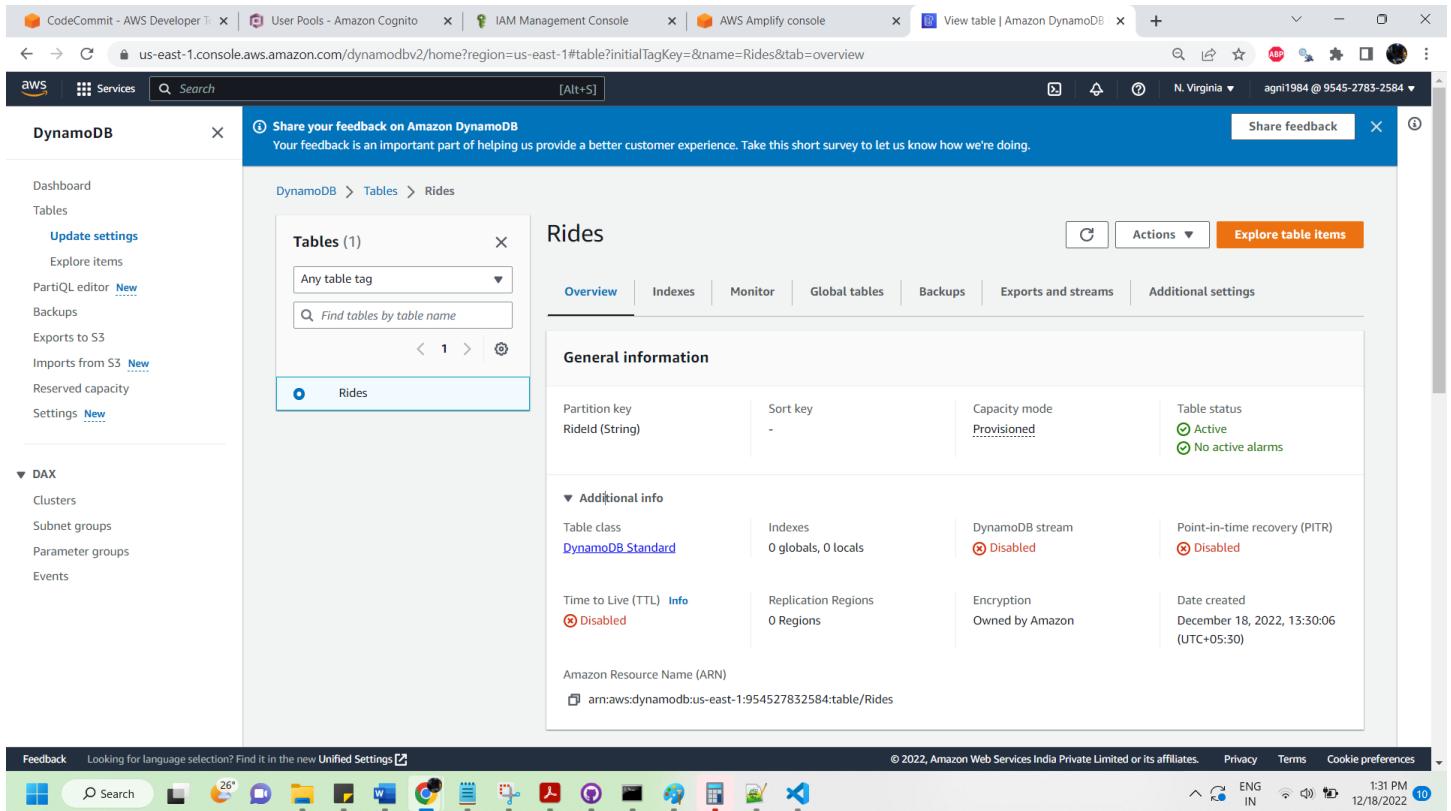
The function is invoked from the browser using Amazon API Gateway. API Gateway will be implemented in the next module.

### Step 1: Create an Amazon DynamoDB Table

Create Dynamo DB table called ‘Rides’ and give it a partition key called ‘RideId’ with type String. Follow the below steps to create DynamoDB table,

- From the AWS Management Console, choose Services then select DynamoDB under Databases.
- Choose Create table.
- Enter ‘Rides’ for the Table name. This field is case sensitive.
- Enter ‘RideId’ for the Partition key and select String for the key type. This field is case sensitive.
- Check the Use default settings box and choose Create.
- Scroll to the bottom of the Overview section of the new table and note the ARN.

**ARN: arn:aws:dynamodb:us-east-1:954527832584:table/Rides**



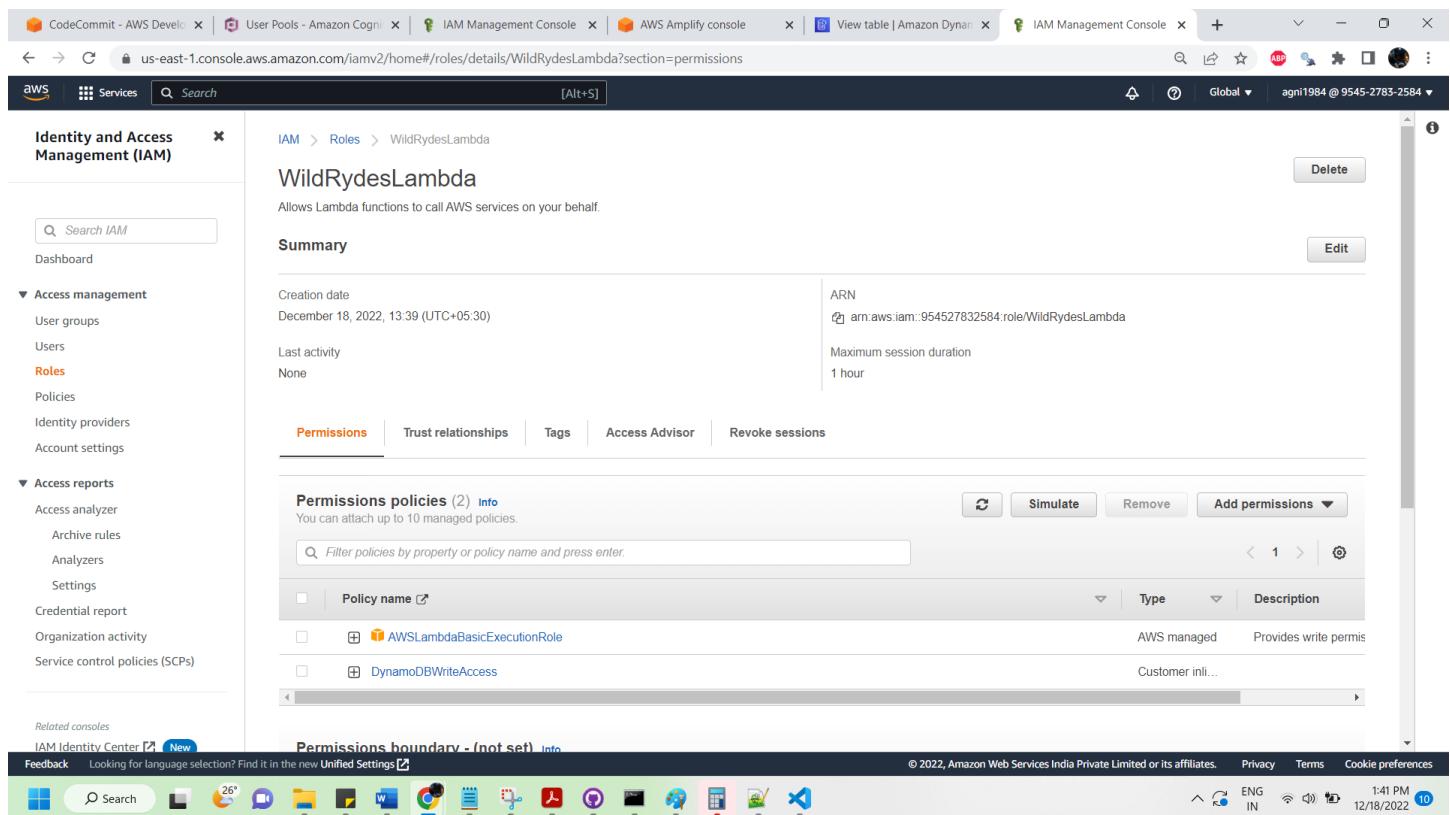
The screenshot shows the AWS DynamoDB console with the Rides table selected. The left sidebar shows navigation options like Dashboard, Tables, Update settings, Explore items, PartiQL editor, Backups, Exports to S3, Imports from S3, Reserved capacity, Settings, DAX, Clusters, Subnet groups, Parameter groups, and Events. The main content area displays the Rides table details. The table has one item, RidId (String). The General information section shows the Partition key as RidId (String), Sort key as -, Capacity mode as Provisioned, and Table status as Active with no active alarms. The Additional info section shows the Table class as DynamoDB Standard, Indexes as 0 globals, 0 locals, and DynamoDB stream as Disabled. It also shows Time to Live (TTL) as Disabled, Replication Regions as 0 Regions, Encryption as Owned by Amazon, and Date created as December 18, 2022, 13:30:06 (UTC+05:30). The ARN is listed as arn:aws:dynamodb:us-east-1:954527832584:table/Rides.

## Step 2: Create an IAM Role for the Lambda function

Every Lambda function has an IAM role associated with it. This role defines what other AWS services the function is allowed to interact with. Create an IAM role that grants our Lambda function permission to write logs to Amazon CloudWatch Logs and access to write items to our DynamoDB table. Follow below steps to create IAM Role,

- From the AWS Management Console, click on Services and then select IAM in the Security, Identity & Compliance section.
- Select Roles in the left navigation pane and then choose Create Role.
- Select Lambda for the role type from the AWS service group, then click Next: Permissions.
- Begin typing **AWSLambdaBasicExecutionRole** in the Filter text box and check the box next to that role.
- Choose Next Step.
- Enter **WildRydesLambda** for the Role Name. Keep other parameters as default.
- Choose Create Role.
- Type **WildRydesLambda** into the filter box on the Roles page and choose the role we just created.
- On the Permissions tab, on the left under Add permissions, choose Create Inline Policy.
- Select Choose a service.
- Begin typing DynamoDB into the search box labeled Find a service and select DynamoDB when it appears.

- Choose Select actions.
- Begin typing PutItem into the search box labeled Filter actions and check the box next to PutItem when it appears.
- Select the Resources section.
- With the Specific option selected, choose the Add ARN link in the table section.
- Paste the ARN of the table we created in the previous section in the Specify ARN for table field, and choose Add.
- Choose Review Policy.
- Enter DynamoDBWriteAccess for the policy name and choose Create policy.



The screenshot shows the AWS IAM Management Console interface. On the left, there's a sidebar with navigation links like Identity and Access Management (IAM), Access management, and Access reports. The main area displays the details for a role named "WildRydesLambda". The "Summary" tab is active, showing the creation date (December 18, 2022, 13:39 UTC+05:30), last activity (None), and ARN (arn:aws:iam::954527832584:role/WildRydesLambda). The "Maximum session duration" is set to 1 hour. Below the summary, the "Permissions" tab is selected, showing two attached policies: "AWSLambdaBasicExecutionRole" (AWS managed, provides write permissions) and "DynamoDBWriteAccess" (Customer initiated). There are buttons for "Edit", "Delete", "Simulate", and "Add permissions". At the bottom, there's a "Permissions boundary - (not set)" section and a footer with various AWS service icons and links.

CodeCommit - AWS Dev... | User Pools - Amazon Cogn... | IAM Management Console | AWS Amplify console | View table | Amazon Dynar... | IAM Management Console | + | - | X

us-east-1.console.aws.amazon.com/iamv2/home#/roles/details/WildRydesLambda?section=permissions

**Identity and Access Management (IAM)**

Search IAM

**Access management**

- User groups
- Users
- Roles**
- Policies
- Identity providers
- Account settings

**Access reports**

- Access analyzer
- Archive rules
- Analyzers
- Settings
- Credential report
- Organization activity
- Service control policies (SCPs)

Related consoles

IAM Identity Center | New

Feedback Looking for language selection? Find it in the new Unified Settings

© 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

ENG IN 142 PM 12/18/2022

**Permissions policies (2) Info**  
You can attach up to 10 managed policies.

Filter policies by property or policy name and press enter.

Policy name	Type	Description
AWSLambdaBasicExecutionRole	AWS managed	Provides write permissions to CloudWatch Logs.

**AWSLambdaBasicExecutionRole**  
Provides write permissions to CloudWatch Logs.

```

1 Version: "2012-10-17",
2 Statement: [
3   {
4     Effect: "Allow",
5     Action: [
6       "logs:CreateLogGroup",
7       "logs:CreateLogStream",
8       "logs:PutLogEvents"
9     ],
10    Resource: "*"
11  }
12 ]
13 ]
14 
```

**DynamoDBWriteAccess**  
Customer inl...

```

1 Version: "2012-10-17",
2 Statement: [
3   {
4     Sid: "VisualEditor0",
5     Effect: "Allow",
6     Action: "dynamodb:PutItem",
7     Resource: "arn:aws:dynamodb:us-east-1:954527832584:table/Rides"
8   }
9 ]
10 ]
11 
```

**Identity and Access Management (IAM)**

Search IAM

**Access management**

- User groups
- Users
- Roles**
- Policies
- Identity providers
- Account settings

**Access reports**

- Access analyzer
- Archive rules
- Analyzers
- Settings
- Credential report
- Organization activity
- Service control policies (SCPs)

Related consoles

IAM Identity Center | New

Feedback Looking for language selection? Find it in the new Unified Settings

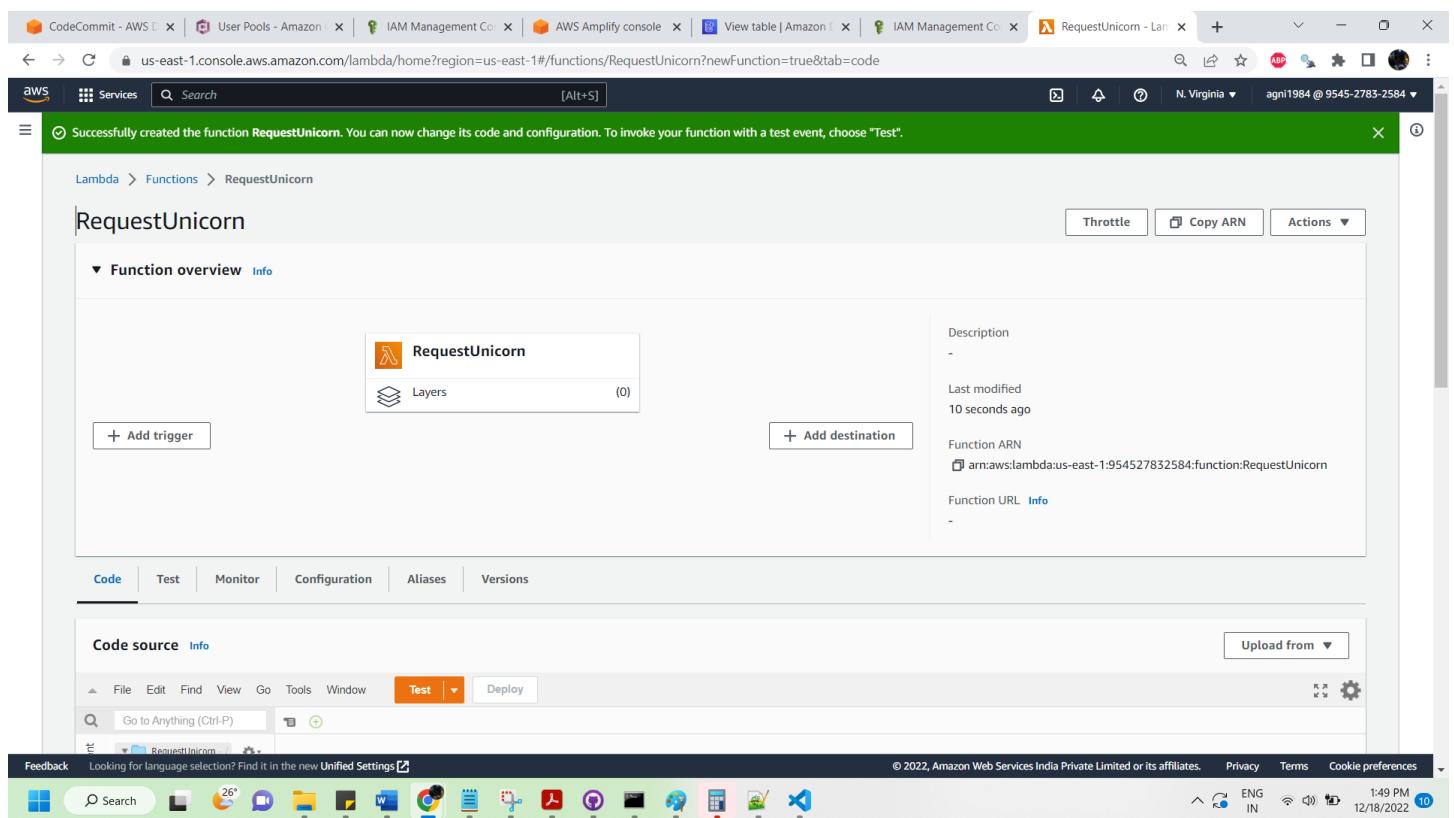
© 2022, Amazon Web Services India Private Limited or its affiliates. Privacy Terms Cookie preferences

ENG IN 2:09 PM 12/18/2022

## Step 3: Create a Lambda Function for Handling Requests

AWS Lambda will run our code in response to events such as an HTTP request. Build the AWS Lambda function that will process API requests from the web application to dispatch a unicorn. Follow the below steps, to create Lambda function,

- Choose Services then select Lambda in the Compute section.
- Click Create function.
- Keep the default Author from scratch card selected.
- Enter RequestUnicorn in the Name field.
- Select Node.js 16.x for the Runtime.
- Ensure Choose an existing role is selected from the Role dropdown.
- Select WildRydesLambda from the Existing Role dropdown.
- Click on Create function.



The screenshot shows the AWS Lambda console interface. At the top, there are several tabs: CodeCommit - AWS, User Pools - Amazon, IAM Management Co, AWS Amplify console, View table | Amazon, IAM Management Co, RequestUnicorn - Lambda, and another IAM Management Co tab. Below the tabs, the URL is us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/RequestUnicorn?newFunction=true&tab=code. The main header bar has 'Services' selected. 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 main content area shows the RequestUnicorn function details. It includes a thumbnail icon for the function, a 'Layers' section (0 layers), a 'Description' field (empty), a 'Last modified' field (10 seconds ago), a 'Function ARN' field (arn:aws:lambda:us-east-1:954527832584:function:RequestUnicorn), and a 'Function URL' field (empty). Below this, there are tabs for 'Code', 'Test', 'Monitor', 'Configuration', 'Aliases', and 'Versions'. The 'Code' tab is selected. Under 'Code source', there is a file browser with a search bar and a 'Test' button. The status bar at the bottom shows: Feedback, Looking for language selection? Find it in the new Unified Settings, © 2022, Amazon Web Services India Private Limited or its affiliates., Privacy, Terms, Cookie preferences, ENG IN, 1:49 PM, 12/18/2022, and a notification badge with the number 10.

- Scroll down to the Function code section and replace the existing code in the index.js code editor with the contents of requestUnicorn.js.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: MIT-0

const randomBytes = require('crypto').randomBytes;
```

```

const AWS = require('aws-sdk');

const ddb = new AWS.DynamoDB.DocumentClient();

const fleet = [
  {
    Name: 'Bucephalus',
    Color: 'Golden',
    Gender: 'Male',
  },
  {
    Name: 'Shadowfax',
    Color: 'White',
    Gender: 'Male',
  },
  {
    Name: 'Rocinante',
    Color: 'Yellow',
    Gender: 'Female',
  },
];
;

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

  const rideId = toUrlString(randomBytes(16));
  console.log('Received event (', rideId, ') : ', event);

  // Because we're using a Cognito User Pools authorizer, all of the claims
  // included in the authentication token are provided in the request context.
  // This includes the username as well as other attributes.
  const username = event.requestContext.authorizer.claims['cognito:username'];

  // The body field of the event in a proxy integration is a raw string.
  // In order to extract meaningful values, we need to first parse this string
  // into an object. A more robust implementation might inspect the Content-Type
  // header first and use a different parsing strategy based on that value.
  const requestBody = JSON.parse(event.body);

  const pickupLocation = requestBody.PickupLocation;

  const unicorn = findUnicorn(pickupLocation);

  recordRide(rideId, username, unicorn).then(() => {
    // You can use the callback function to provide a return value from your Node.js
    // Lambda functions. The first parameter is used for failed invocations. The
    // second parameter specifies the result data of the invocation.

    // Because this Lambda function is called by an API Gateway proxy integration
    // the result object must use the following structure.
    callback(null, {
      statusCode: 201,
      body: JSON.stringify({
        RideId: rideId,
        Unicorn: unicorn,
      })
    });
  });
};

function recordRide(rideId, username, unicorn) {
  return new Promise((resolve, reject) => {
    const params = {
      Item: {
        RideId: rideId,
        Username: username,
        Unicorn: unicorn,
      }
    };
    ddb.put(params, (err, data) => {
      if (err) {
        reject(err);
      } else {
        resolve();
      }
    });
  });
}

function errorResponse(message, awsRequestId, callback) {
  const error = {
    message: message,
    awsRequestId: awsRequestId
  };
  callback(error, null);
}

```

```

        UnicornName: unicorn.Name,
        Eta: '30 seconds',
        Rider: username,
    },
    headers: {
        'Access-Control-Allow-Origin': '*',
    },
},
));
}).catch((err) => {
    console.error(err);

    // If there is an error during processing, catch it and return
    // from the Lambda function successfully. Specify a 500 HTTP status
    // code and provide an error message in the body. This will provide a
    // more meaningful error response to the end client.
    errorResponse(err.message, context.awsRequestId, callback)
});
};

// This is where you would implement logic to find the optimal unicorn for
// this ride (possibly invoking another Lambda function as a microservice.)
// For simplicity, we'll just pick a unicorn at random.
function findUnicorn(pickupLocation) {
    console.log('Finding unicorn for ', pickupLocation.Latitude, ', ',
    pickupLocation.Longitude);
    return fleet[Math.floor(Math.random() * fleet.length)];
}

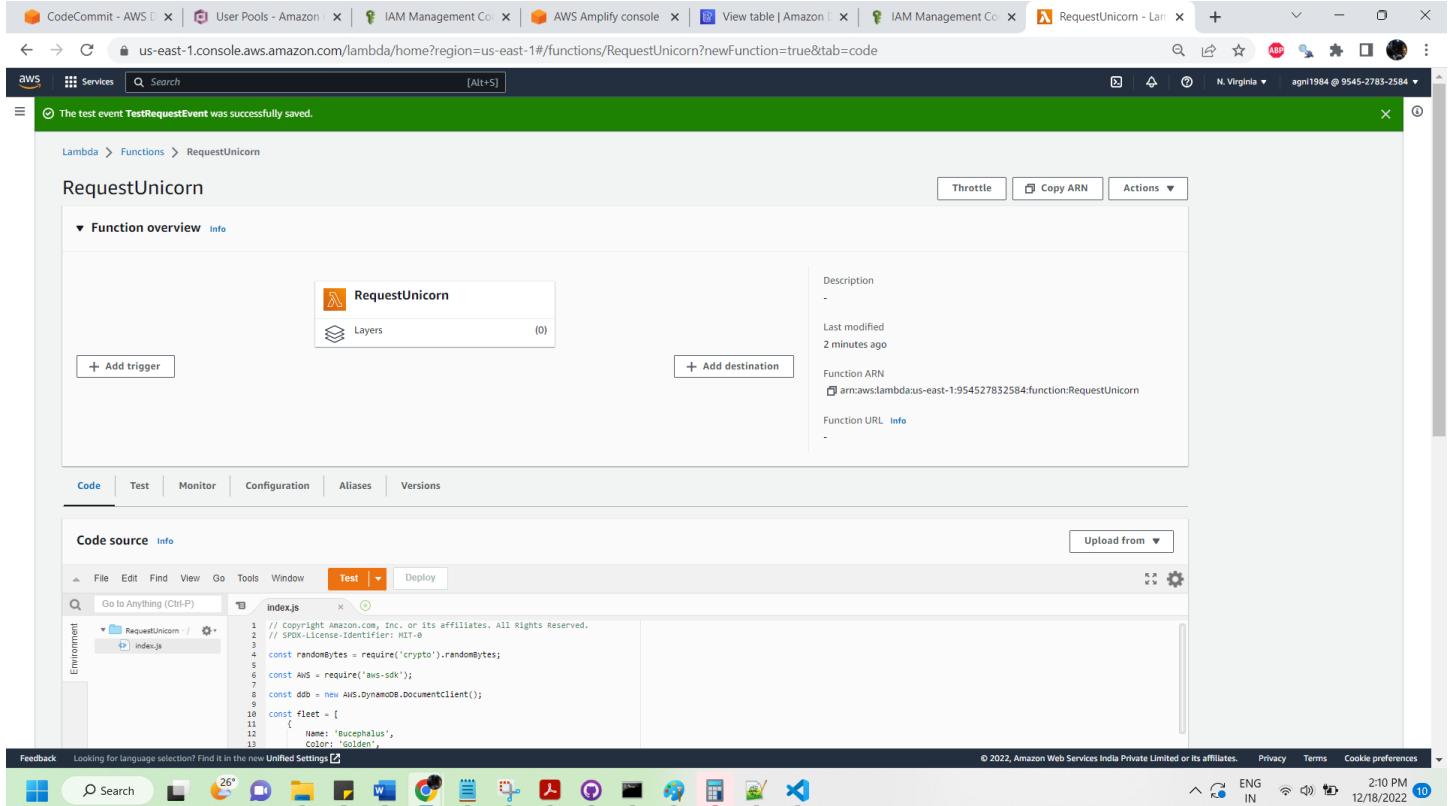
function recordRide(rideId, username, unicorn) {
    return ddb.put({
        TableName: 'Rides',
        Item: {
            RideId: rideId,
            User: username,
            Unicorn: unicorn,
            UnicornName: unicorn.Name,
            RequestTime: new Date().toISOString(),
        },
    }).promise();
}

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

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

```

➤ Choose Deploy.



```

1 // Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
2 // SPDX-License-Identifier: MIT-0
3
4 const randomBytes = require('crypto').randomBytes;
5
6 const AWS = require('aws-sdk');
7
8 const ddb = new AWS.DynamoDB.DocumentClient();
9
10 const fleet = [
11   {
12     Name: 'Bucephalus',
13     Color: 'Golden',
14   },
15 ];
16
17 module.exports = function(event, context) {
18   const item = fleet[Math.floor(Math.random() * fleet.length)];
19
20   const params = {
21     TableName: 'Unicorns',
22     Item: item,
23   };
24
25   ddb.put(params, function(err, data) {
26     if (err) {
27       context.fail(`Error putting item ${item} to table: ${err}`);
28     } else {
29       context.succeed(`Item ${item} added to table`);
30     }
31   });
32 };

```

## Step 4: Validate the Implementation

Test the created AWS lambda function, by following below steps,

- From the main edit screen for our function, select Test and choose Configure test event from the dropdown.
- Keep Create new event selected.
- Enter TestRequestEvent in the Event name field
- Copy and paste the following test event into the editor:

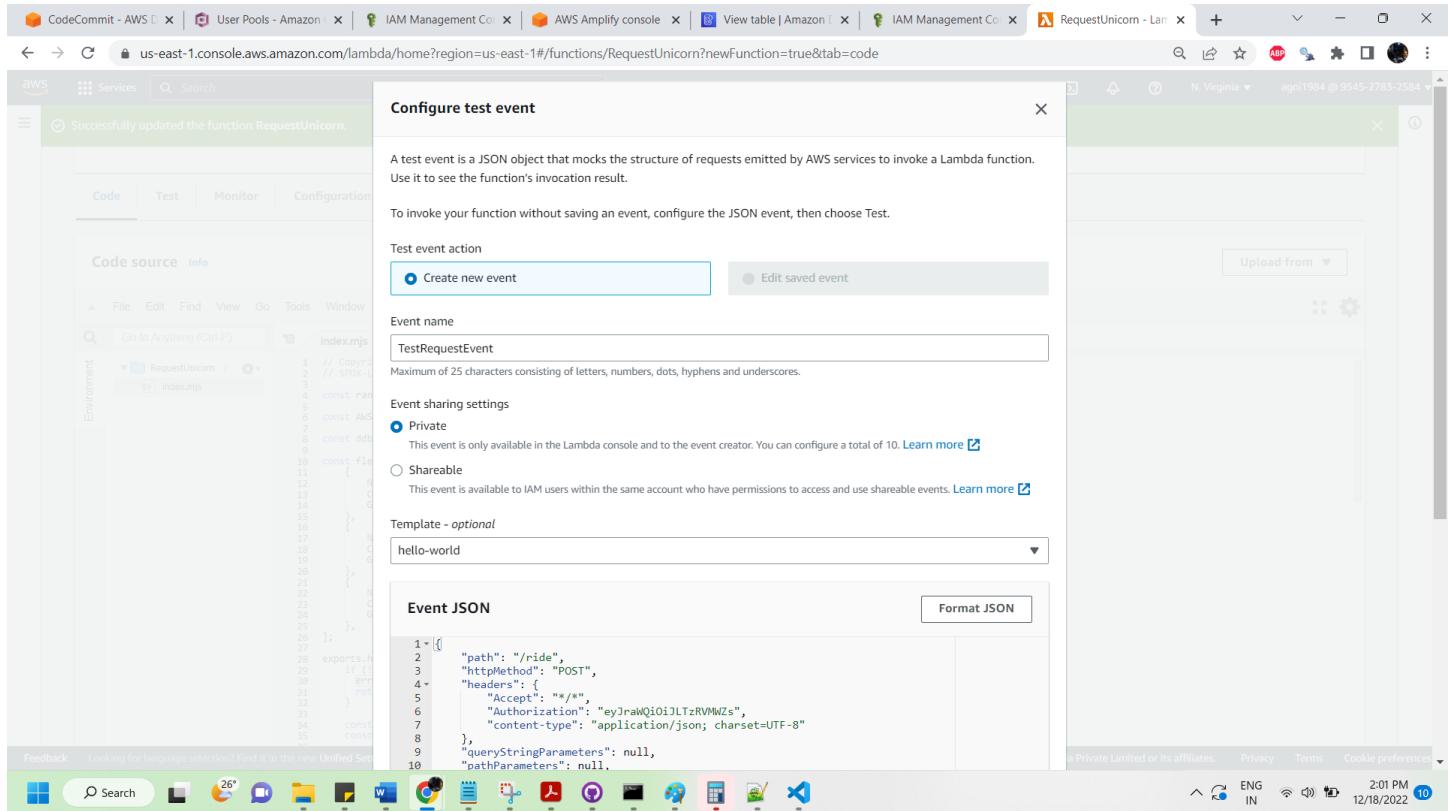
```
{
  "path": "/ride",
  "httpMethod": "POST",
  "headers": {
    "Accept": "*/*",
    "Authorization": "eyJRaWQiOiJLTzRVMWZs",
    "content-type": "application/json; charset=UTF-8"
  },
  "queryStringParameters": null,
  "pathParameters": null,
  "requestContext": {
    "authorizer": {
      "claims":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

```

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

```

➤ Choose Save.

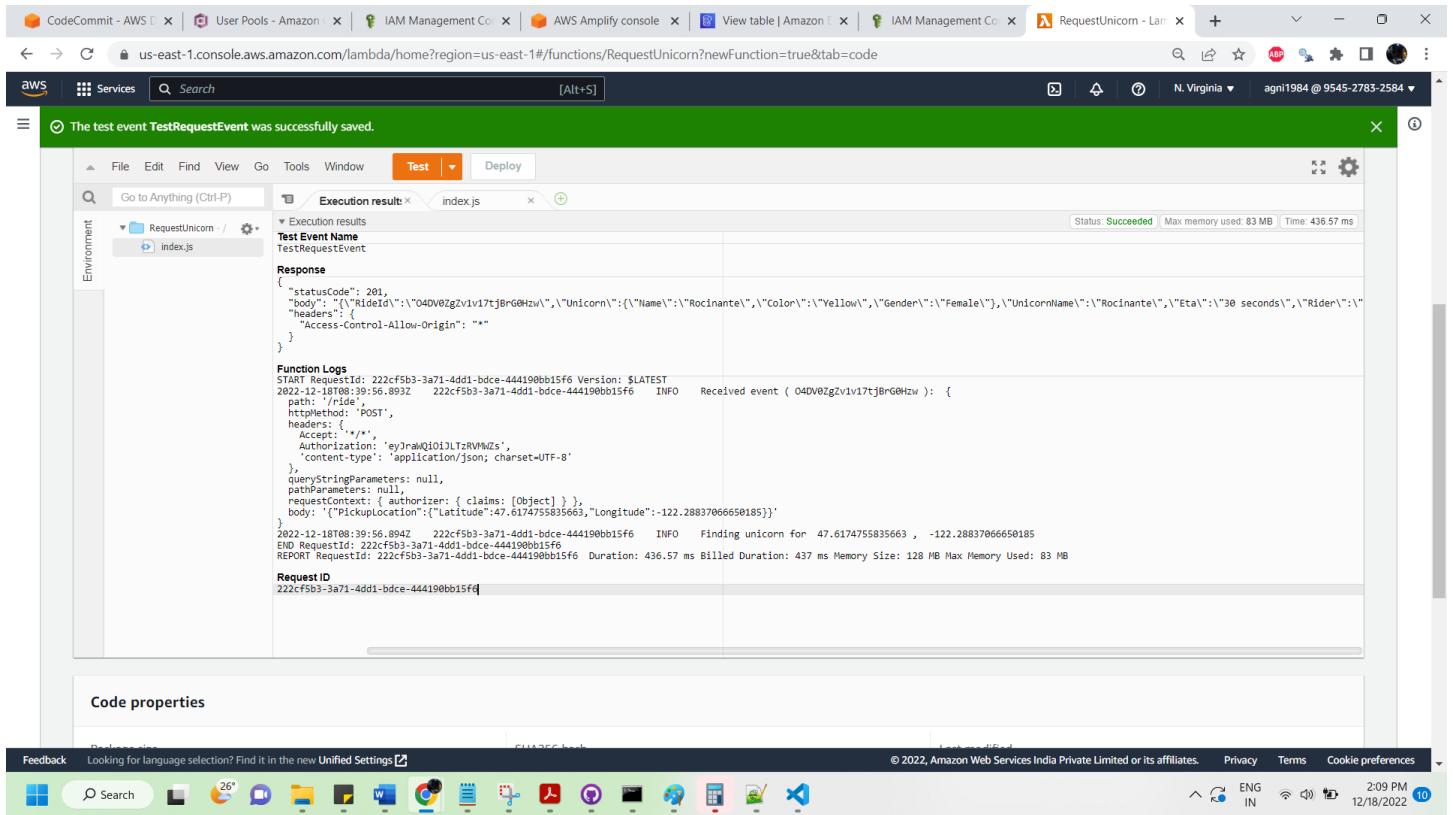


- On the main function edit screen click Test with TestRequestEvent selected in the dropdown.
- Scroll to the top of the page and expand the Details section of the Execution result section.
- Verify that the execution succeeded and that the function result looks like the following:

```

{
  "statusCode": 201,
  "body": "{\"RideId\":\"SvLnijIAtg6inAFUBRT+Fg==\", \"Unicorn\":{\"Name\":\"Rocinante\", \"Color\":\"Yellow\", \"Gender\":\"Female\"}, \"Eta\":\"30 seconds\"}",
  "headers": {
    "Access-Control-Allow-Origin": "*"
  }
}

```



The test event **TestRequestEvent** was successfully saved.

**Execution result:** index.js

**Test Event Name:** TestRequestEvent

**Response:**

```
{
  "statusCode": 201,
  "body": "{\"RideId\":\"04DV0ZgZv1v17tjBrG0Hzw\", \"Unicorn\": {\"Name\": \"Rocinante\", \"Color\": \"Yellow\", \"Gender\": \"Female\"}, \"UnicornName\": \"Rocinante\", \"Eta\": \"30 seconds\", \"Rider\": \"\"}
}
```

**Function Logs:**

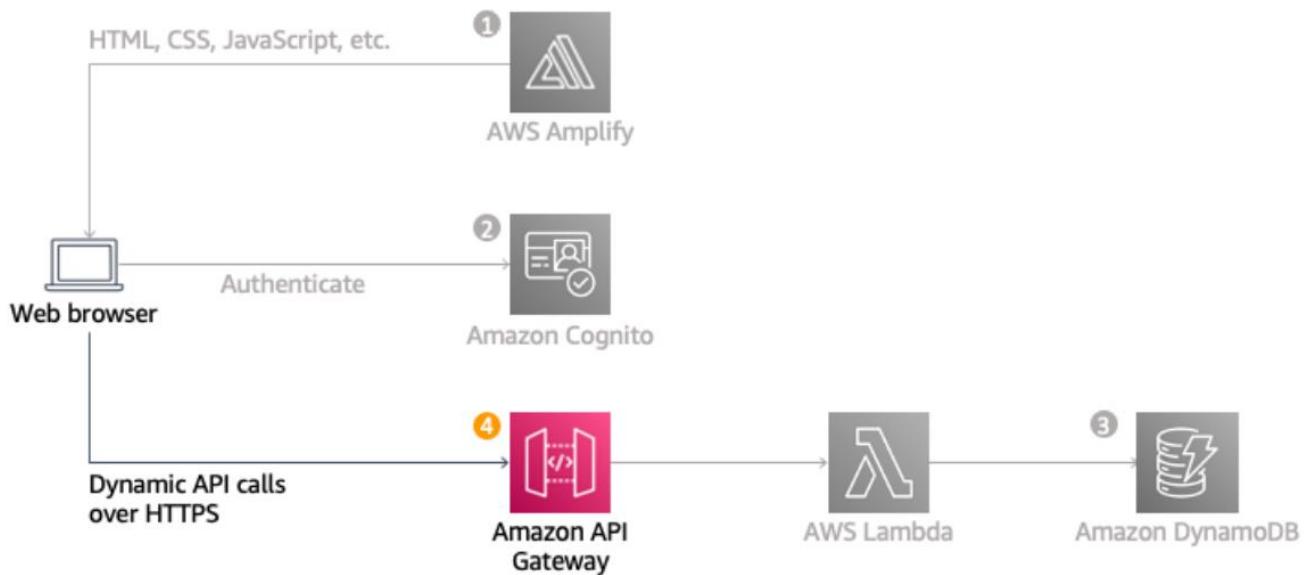
```
START RequestId: 222cf5b3-3a71-4dd1-bdce-444190bb15f6 Version: $LATEST
2022-12-18T08:39:56.893Z 222cf5b3-3a71-4dd1-bdce-444190bb15f6 INFO  Received event ( 04DV0ZgZv1v17tjBrG0Hzw ): {
  path: '/ride',
  httpMethod: 'POST',
  headers: { 'x-api-key': 'eyJraiwQ0j013LTzRVNwZs' },
  'Access-Control-Allow-Origin': '*'
}
queryStringParameters: null,
pathParameters: null,
requestContext: { authorizer: { claims: [Object] } },
body: '{"PickupLocation": {"Latitude": 47.6174755835663, "Longitude": -122.28837066650185}}'
}
2022-12-18T08:39:56.894Z 222cf5b3-3a71-4dd1-bdce-444190bb15f6 INFO  Finding unicorn for 47.6174755835663, -122.28837066650185
END RequestId: 222cf5b3-3a71-4dd1-bdce-444190bb15f6
REPORT RequestId: 222cf5b3-3a71-4dd1-bdce-444190bb15f6 Duration: 436.57 ms Billed Duration: 437 ms Memory Size: 128 MB Max Memory Used: 83 MB
```

**Request ID:** 222cf5b3-3a71-4dd1-bdce-444190bb15f6

## Module 4: RESTful API

In this module, Amazon API Gateway is used to expose the Lambda function that we built in the previous module as a RESTful API. This API will be accessible on the public Internet. It will be secured using the Amazon Cognito user pool we created in the previous module. Using this configuration, we will then turn our statically hosted website into a dynamic web application by adding client-side JavaScript that makes AJAX calls to the exposed APIs.

## Architectural Overview



The static website we deployed in the first module already has a page configured to interact with the API we will build in this module. The page at /ride.html has a simple map-based interface for requesting a unicorn ride. After authenticating using the /signin.html page, users will be able to select their pickup location by clicking a point on the map and then requesting a ride by choosing the "Request Unicorn" button in the upper right corner.

This module will focus on the steps required to build the cloud components of the API.

### Step 1: Create a New REST API

- In the AWS Management Console, click Services then select API Gateway under Application Services.
- Choose Create API.
- Select Build under REST API and enter WildRydes for the API Name.

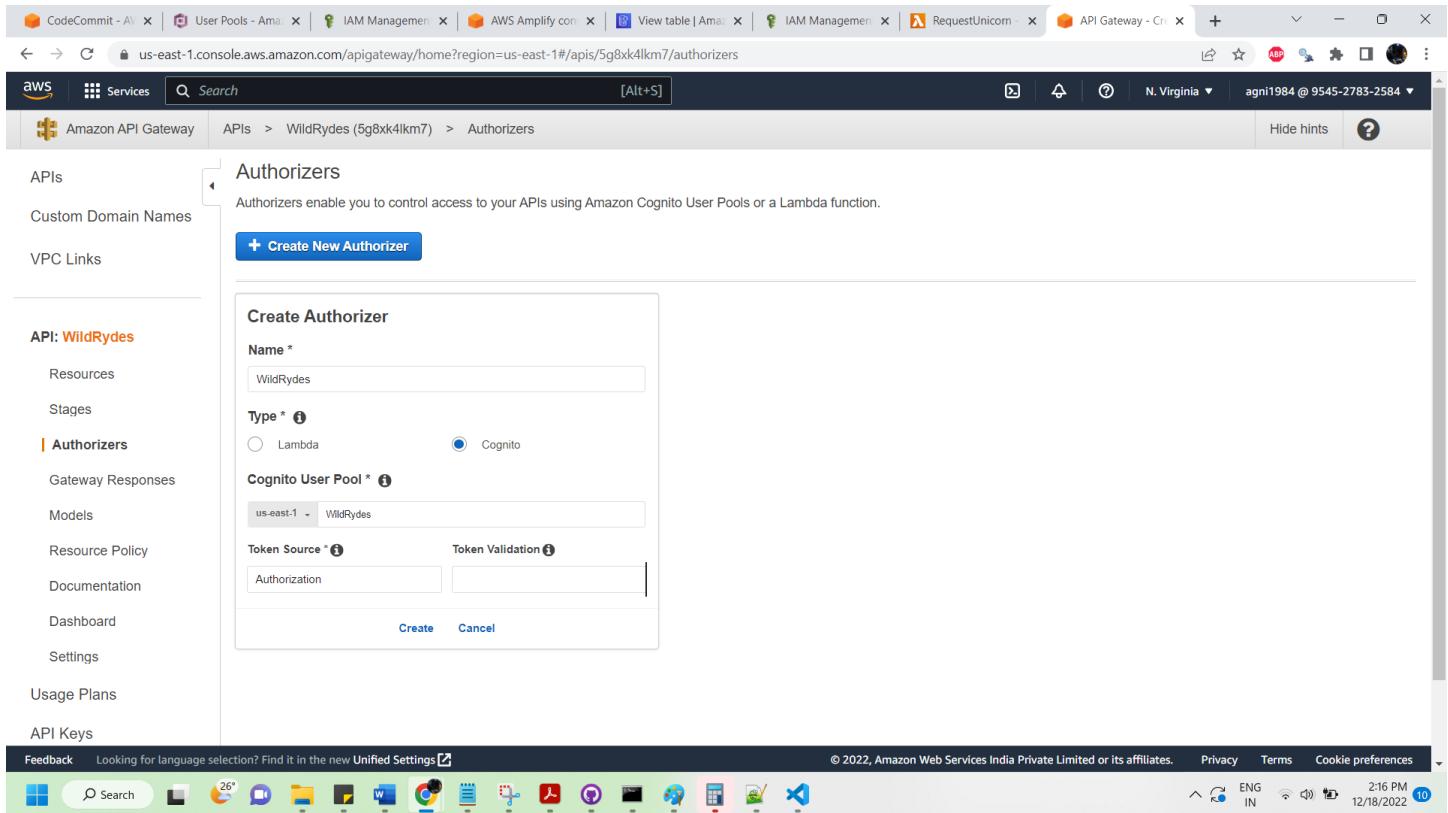
- Choose Edge optimized in the Endpoint Type dropdown. Note: Edge optimized are best for public services being accessed from the Internet. Regional endpoints are typically used for APIs that are accessed primarily from within the same AWS Region.
- Choose Create API

## Step 2: Create a Cognito User Pool Authorizer

Amazon API Gateway can use the JWT tokens returned by Cognito User Pools to authenticate API calls. In this step we'll configure an authorizer for our API to use the user pool we created in Module 2.

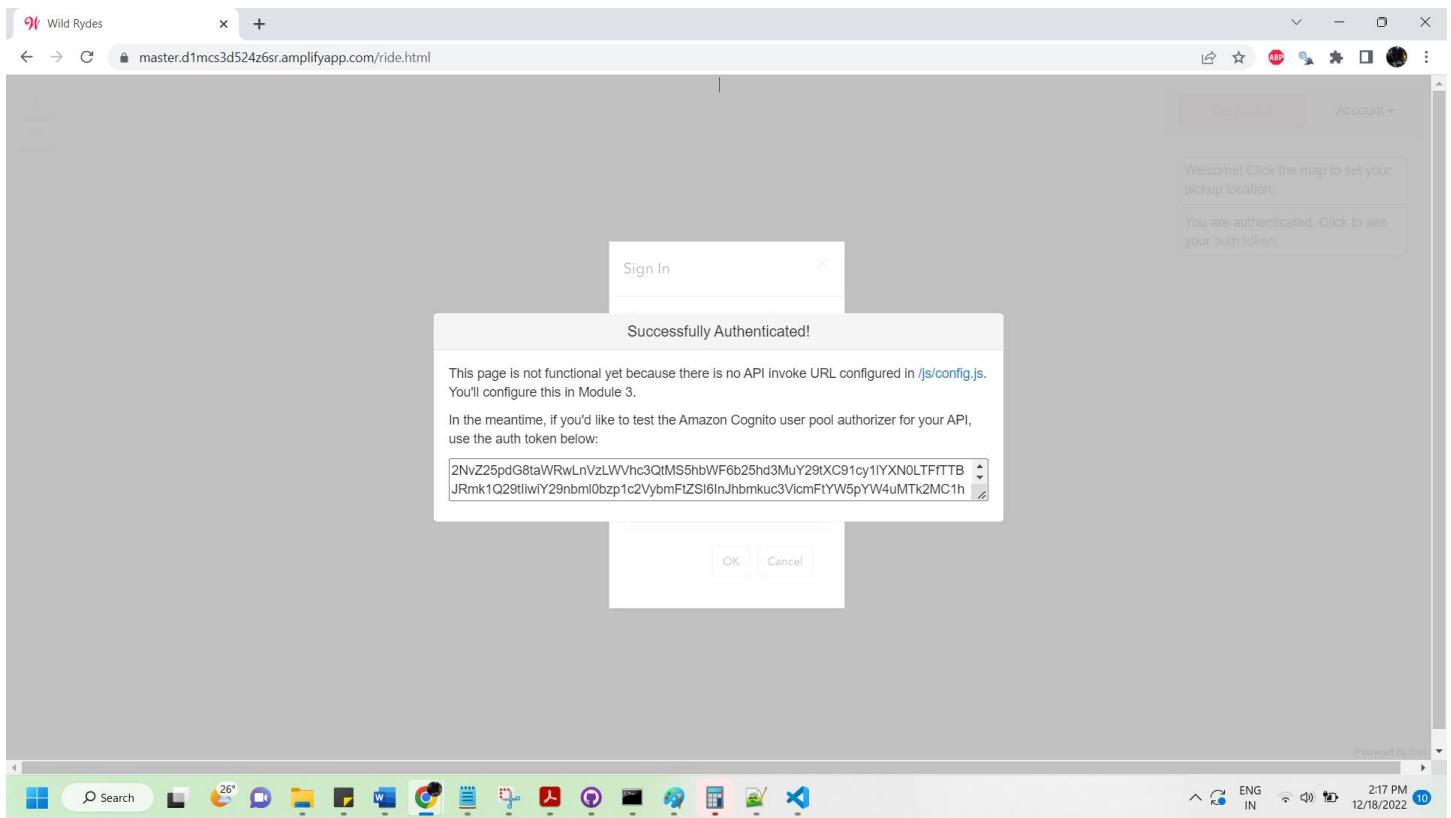
In the Amazon API Gateway console, create a new Cognito user pool authorizer for our API. Configure it with the details of the user pool that we created in the previous module. We can test the configuration in the console by copying and pasting the auth token presented to us, after we log in via the /signin.html page of our current website.

- Under our newly created API, choose Authorizers.
- Choose Create New Authorizer.
- Enter WildRydes for the Authorizer name.
- Select Cognito for the type.
- Enter WildRydes (or the name we gave our user pool) in the Cognito User Pool input.
- Enter Authorization for the Token Source.
- Choose Create.



The screenshot shows the AWS API Gateway interface. On the left, a sidebar lists various API management options like APIs, Stages, and Authorizers. The 'Authorizers' option is selected. The main content area displays a 'Create Authorizer' form. The 'Name' field is filled with 'WildRydes'. The 'Type' section has 'Cognito' selected. Under 'Cognito User Pool', 'us-east-1' is chosen, and the pool name is 'WildRydes'. The 'Token Source' dropdown is set to 'Authorization', and the 'Token Validation' field is empty. At the bottom of the form are 'Create' and 'Cancel' buttons.

- Verify our authorizer configuration
- Open a new browser tab and visit /ride.html under our website's domain.
- If we are redirected to the sign-in page, sign in with the user we created in the last module. We will be redirected back to /ride.html.
- Copy the auth token from the notification on the /ride.html,



- Go back to previous tab where we have just finished creating the Authorizer.
- Click Test at the bottom of the card for the authorizer.
- Paste the auth token into the Authorization Token field in the popup dialog.
- Click Test button and verify that the response code is 200 and that we see the claims for our user displayed.

CodeCommit - AI | User Pools - Amazon Cognito | IAM Management | AWS Amplify console | View table | IAM Management | RequestUnicorn - | API Gateway - Create API | + | - | Hide hints | ?

us-east-1.console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/5g8xk4lkm7/authorizers

aWS Services Search [Alt+S]

Amazon API Gateway APIs > WildRydes (5g8xk4lkm7) > Authorizers

**Authorizers**

Authorizers enable you to control access to your API.

+ Create New Authorizer

**WildRydes**

Authorizer ID: bpnr9b

**Cognito User Pool**

WildRydes - MOIFI5Com (us-east-1)

**Token Source**

Authorization

Authorization Token i

Authorization (header) C66jsqLEuv5N56TatqX5JtauzQH97j4ekEINSB8zccnYELTBReShKrw

Test

Close

Edit Test

Feedback Looking for language support

26° ENG IN 12/18/2022

CodeCommit - AI | User Pools - Amazon Cognito | IAM Management | AWS Amplify console | View table | IAM Management | RequestUnicorn - | API Gateway - Create API | + | - | Hide hints | ?

us-east-1.console.aws.amazon.com/apigateway/home?region=us-east-1#/apis/5g8xk4lkm7/authorizers

aWS Services Search [Alt+S]

Amazon API Gateway APIs > WildRydes (5g8xk4lkm7) > Authorizers

**Authorizers**

Authorizers enable you to control access to your API.

+ Create New Authorizer

**WildRydes**

Authorizer ID: bpnr9b

**Cognito User Pool**

WildRydes - MOIFI5Com (us-east-1)

**Token Source**

Authorization

**Response**

Response Code: 200

Latency 57

Claims

```
{
  "aud": "5rhr9fbhbhq42v2euumik1dau",
  "auth_time": "1671349927",
  "cognito:username": "rani.subramanian.1960-at-gmail.com",
  "email": "rani.subramanian.1960@gmail.com",
  "email_verified": "true",
  "event_id": "b044aa8a-87d4-424e-9946-055180da2e81",
  "exp": "Sun Dec 18 08:52:07 UTC 2022",
  "iat": "Sun Dec 18 07:52:07 UTC 2022",
  "iss": "https://cognito-idp.us-east-1.amazonaws.com/us-east-1_MOIFI5Com",
  "jti": "6c676036-d11b-4a8d-ba6a-5769802dd5e1",
  "origin_jti": "cfed71fb-b99d-4c62-a1e1-f40c58d755aa",
  "sub": "84be25e3-91ae-438e-afec-56475aa11e47",
  "token_use": "id"
}
```

Close

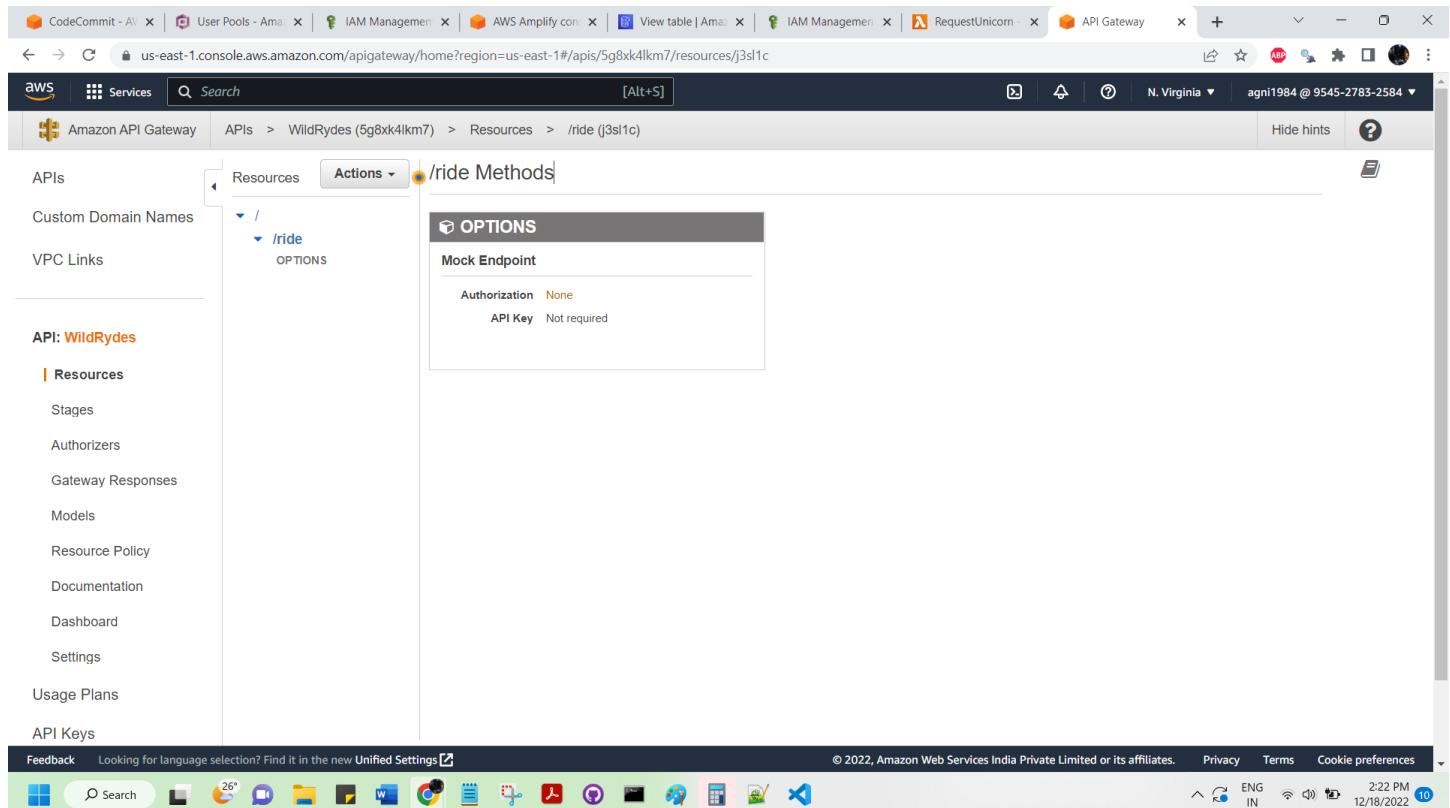
Feedback Looking for language support

26° ENG IN 12/18/2022

## Step 3: Create a new resource and method

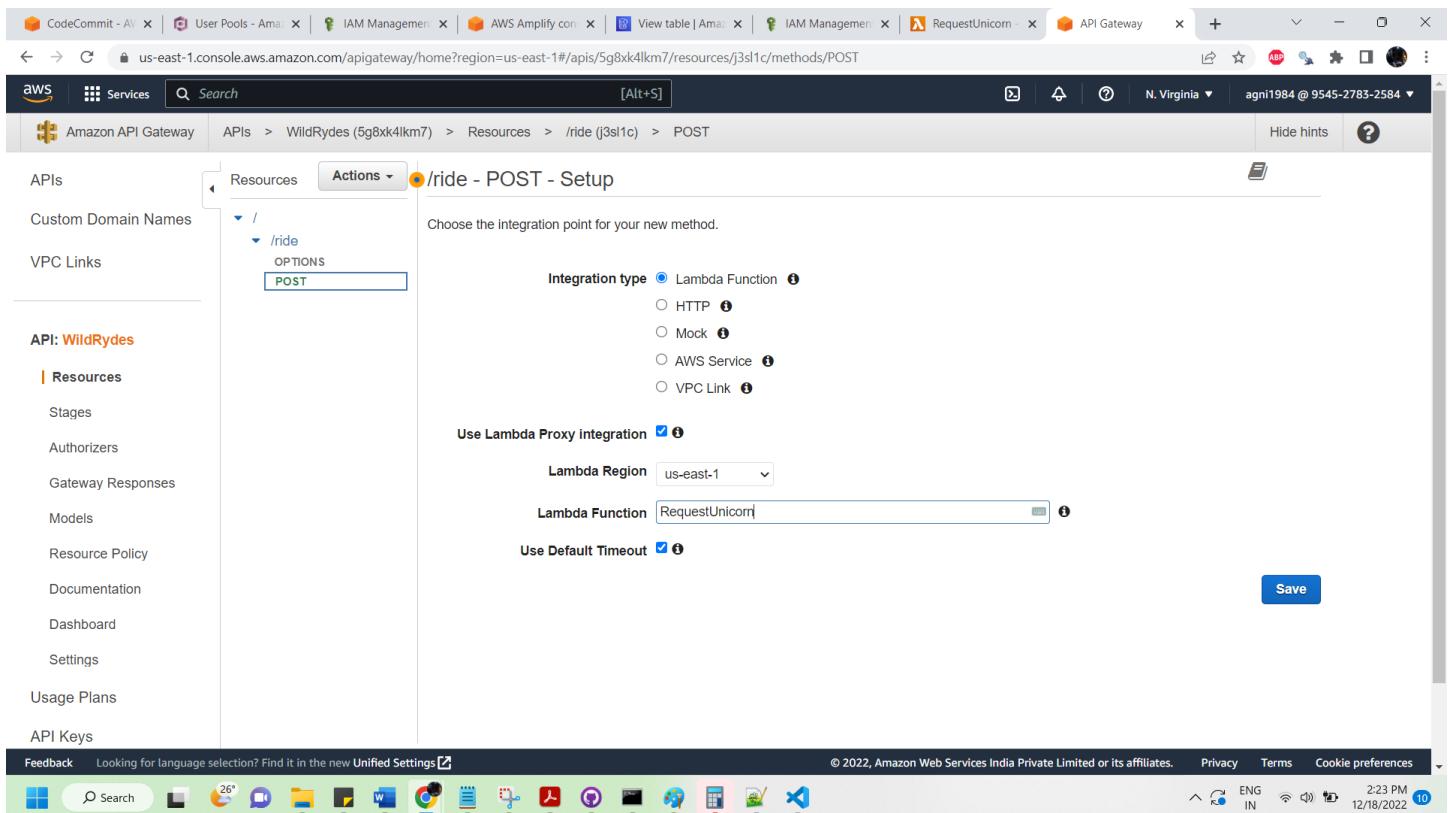
Create a new resource called /ride within our API. Then create a POST method for that resource and configure it to use a Lambda proxy integration backed by the RequestUnicorn function we created in the first step of this module.

- In the left nav, click on Resources under our WildRydes API.
- From the Actions dropdown select Create Resource.
- Enter ride as the Resource Name.
- Ensure the Resource Path is set to ride.
- Select Enable API Gateway CORS for the resource.
- Click Create Resource.



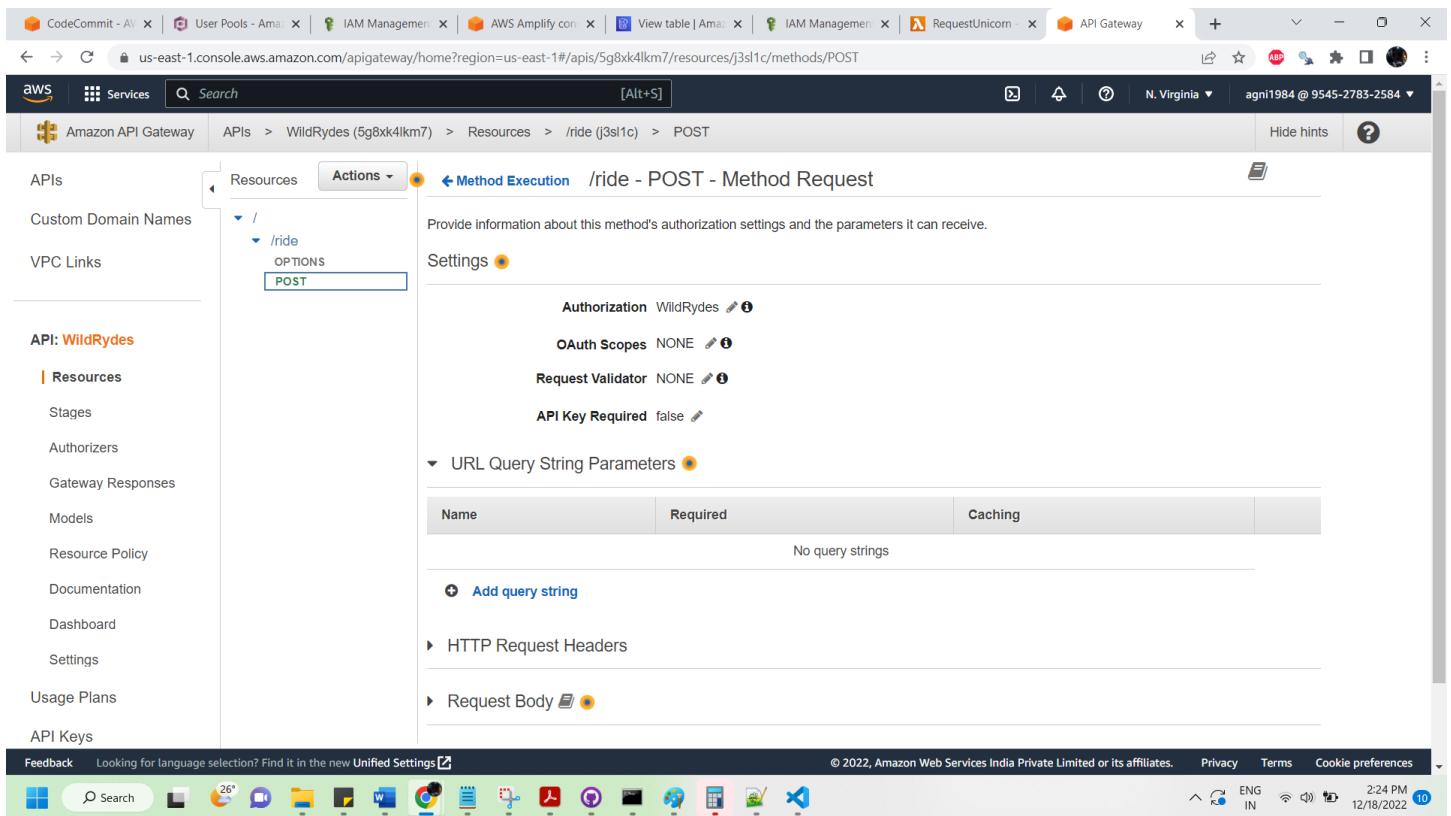
The screenshot shows the AWS API Gateway console. The left sidebar is expanded to show the 'Resources' section. Under the 'WildRydes' API, there is a single resource named 'ride' with a path of '/ride'. The 'Actions' dropdown is open, and 'Create Method' is selected. A modal window titled 'OPTIONS' is displayed, showing the configuration for the OPTIONS method. The 'Mock Endpoint' checkbox is checked. Under 'Authorization', 'None' is selected, and 'API Key' is noted as 'Not required'. The bottom of the screen shows the standard AWS navigation bar with various icons and links.

- With the newly created /ride resource selected, from the Action dropdown select Create Method.
- Select POST from the new dropdown that appears, then click the checkmark.
- Select Lambda Function for the integration type.
- Check the box for Use Lambda Proxy integration.
- Select the Region we are using for Lambda Region.
- Enter the name of the function we created in the previous module, RequestUnicorn, for Lambda Function.
- Choose Save. Please note, if we get an error that our function does not exist, check that the region we selected matches the one we used in the previous module.



The screenshot shows the AWS API Gateway console. On the left sidebar, under the 'APIs' section, the 'WildRydes' API is selected. In the main content area, the path 'APIs > WildRydes (5g8xk4lkm7) > Resources > /ride (j3sl1c) > POST' is shown. The 'Actions' dropdown is open, and the 'Setup' option is selected. The 'Integration type' is set to 'Lambda Function'. The 'Lambda Region' is 'us-east-1' and the 'Lambda Function' is 'RequestUnicorn'. The 'Save' button is visible at the bottom right.

- When prompted to give Amazon API Gateway permission to invoke our function, choose OK.
- Choose on the Method Request card.
- Choose the pencil icon next to Authorization.
- Select the WildRydes Cognito user pool authorizer from the drop-down list, and click the checkmark icon.



The screenshot shows the AWS API Gateway console. On the left sidebar, under the 'APIs' section, the 'WildRydes' API is selected. In the main content area, the path 'Amazon API Gateway > APIs > WildRydes (5g8xk4lkm7) > Resources > /ride (j3sl1c) > POST' is shown. The 'Actions' dropdown is open, and 'Method Execution' is selected. The page title is 'Method Execution /ride - POST - Method Request'. The 'Authorization' field is set to 'WildRydes'. The 'OAuth Scopes' and 'Request Validator' fields are both set to 'NONE'. The 'API Key Required' field is set to 'false'. Under 'URL Query String Parameters', there is a table with one row: 'Name' (empty), 'Required' (checkbox checked), and 'Caching' (checkbox checked). A note says 'No query strings'. Below the table are buttons for 'Add query string' and 'Edit'. Under 'HTTP Request Headers' and 'Request Body', there are expandable sections with icons.

## Step 4: Deploy our API

From the Amazon API Gateway console, choose Actions, Deploy API. We'll be prompted to create a new stage. Use prod for the stage name.

- In the Actions drop-down list select Deploy API.
- Select [New Stage] in the Deployment stage drop-down list.
- Enter prod for the Stage Name.
- Choose Deploy.
- Note the Invoke URL. We will use it in the next section.

Screenshot of the AWS API Gateway console showing the deployment process for the WildRydes API.

**Deploy API Dialog:**

- Deployment stage:** [New Stage]
- Stage name\***: prod
- Stage description**: (empty)
- Deployment description**: (empty)
- Buttons:** Cancel, Deploy

**API Stages View:**

- API: WildRydes
  - Stages: /, /ride
  - Actions: OPTIONS, POST

**prod Stage Editor:**

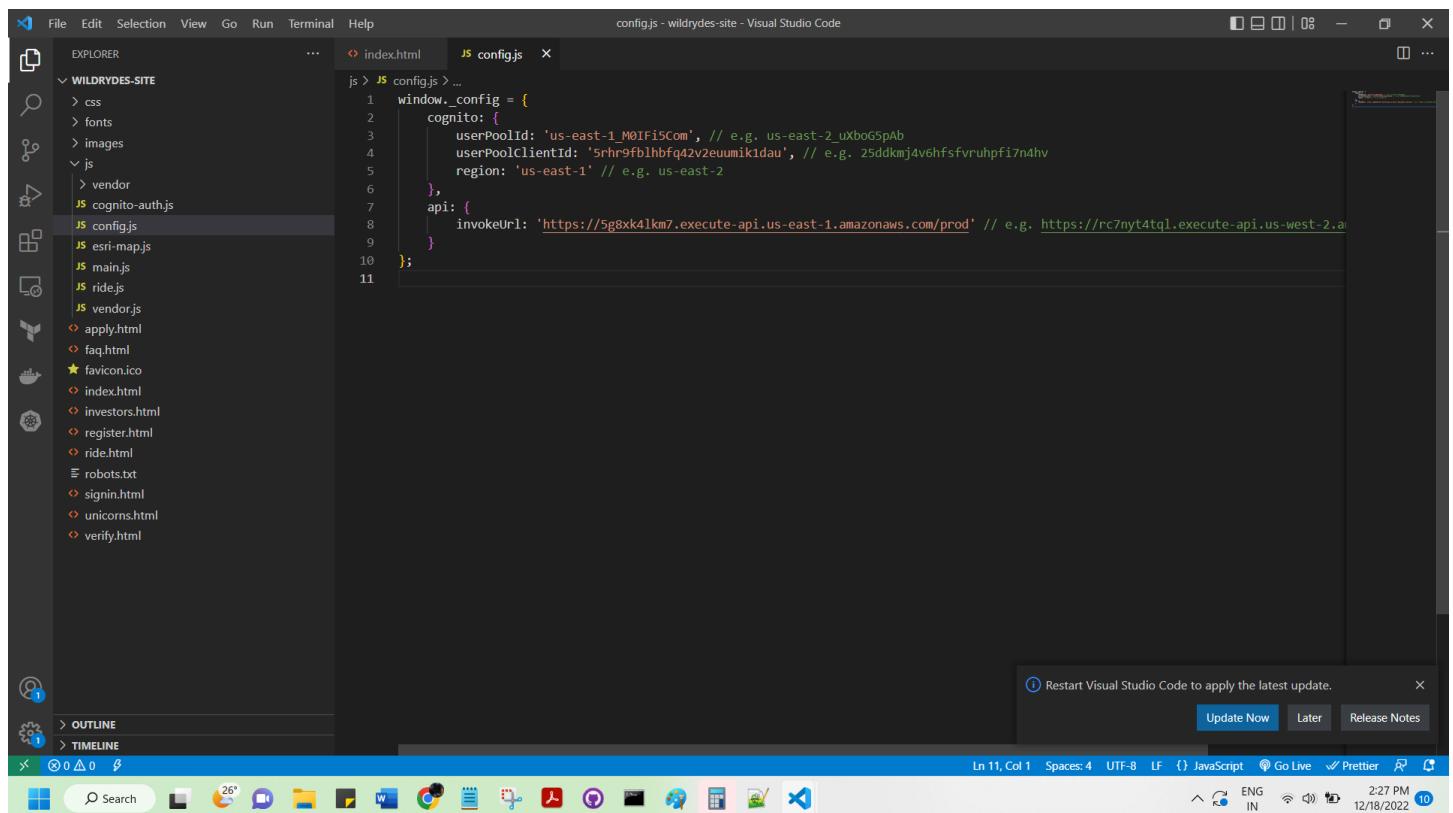
- Invoke URL:** <https://5g8xk4lkm7.execute-api.us-east-1.amazonaws.com/prod>
- Cache Settings:**
  - Enable API cache**:
- Default Method Throttling:**
  - Choose the default throttling level for the methods in this stage. Each method in this stage will respect these rate and burst settings. Your current account level throttling rate is **10000** requests per second with a burst of **5000** requests. [Read more about API Gateway throttling](#).
  - Enable throttling**:  [i](#)
  - Rate**: 10000 requests per second
  - Burst**: 5000 requests
- Web Application Firewall (WAF)**: [Learn more.](#)
- Select the Web ACL to be applied to this stage.**
- Web ACL**: None [Create Web ACL](#)

**Invoke URL:** <https://5g8xk4lkm7.execute-api.us-east-1.amazonaws.com/prod>

## Step 5: Update the Website Config

Update the `/js/config.js` file in our website deployment to include the invoke URL of the stage we just created. We should copy the invoke URL directly from the top of the stage editor page on the Amazon API Gateway console and paste it into the `_config.api.invokeUrl` key of our sites `/js/config.js` file.

- Open the config.js file in a text editor.
- Update the `invokeUrl` setting under the `api` key in the config.js file. Set the value to the Invoke URL for the deployment stage, which we created in the previous section.



- Save the modified file and push it to Git repository to have it automatically deploy to Amplify Console.

```
$ git add .
$ git commit -m "new_configuration"
$ git push
```

```
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>git add .
warning: in the working copy of 'js/config.js', LF will be replaced by CRLF the next time Git touches it
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>git commit -m "new_configuration"
[master 416cbbb] new_configuration
 1 file changed, 1 insertion(+), 1 deletion(-)

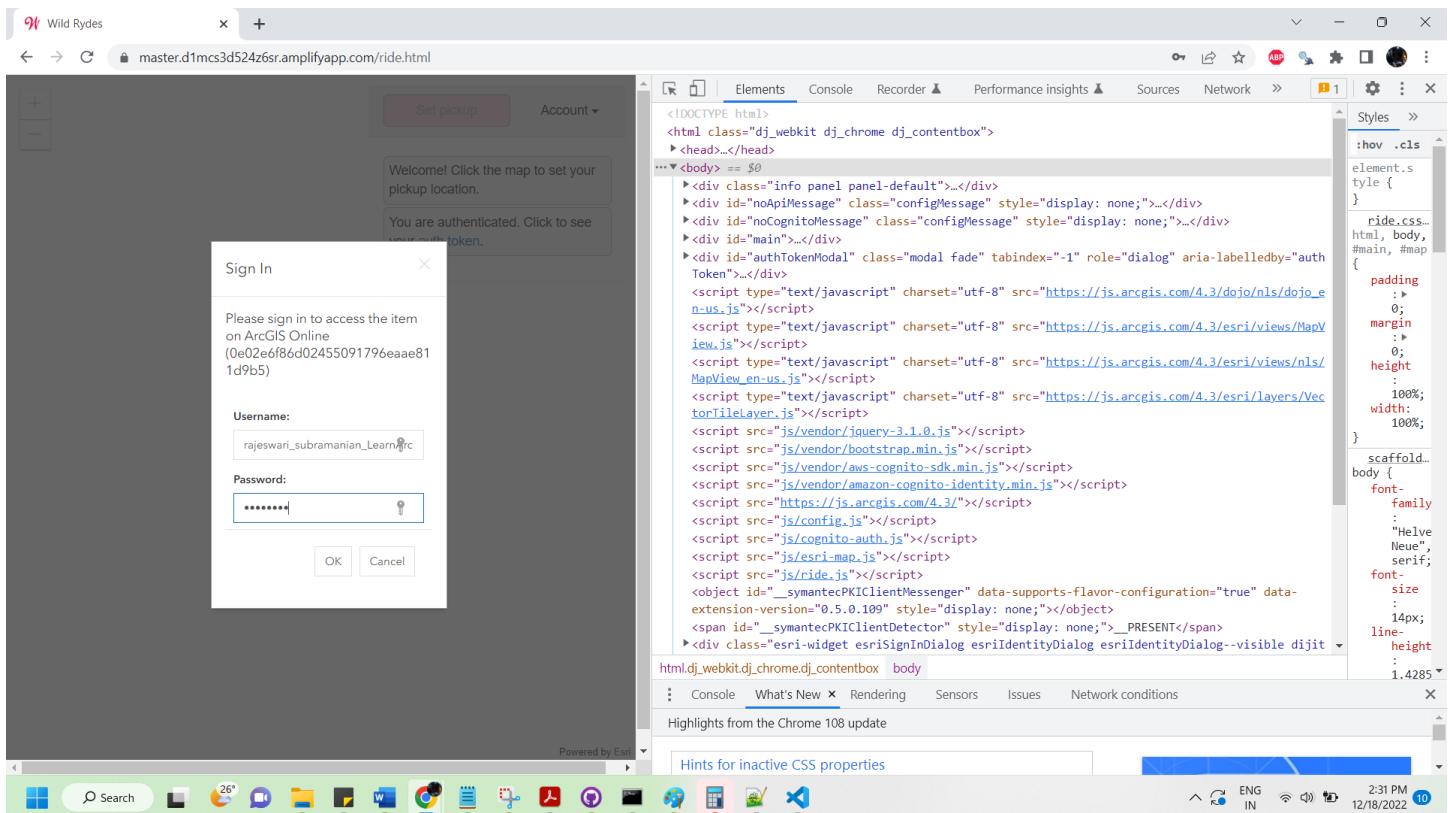
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 410 bytes | 410.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/wildrydes-site
 18fb109..416cbbb master -> master

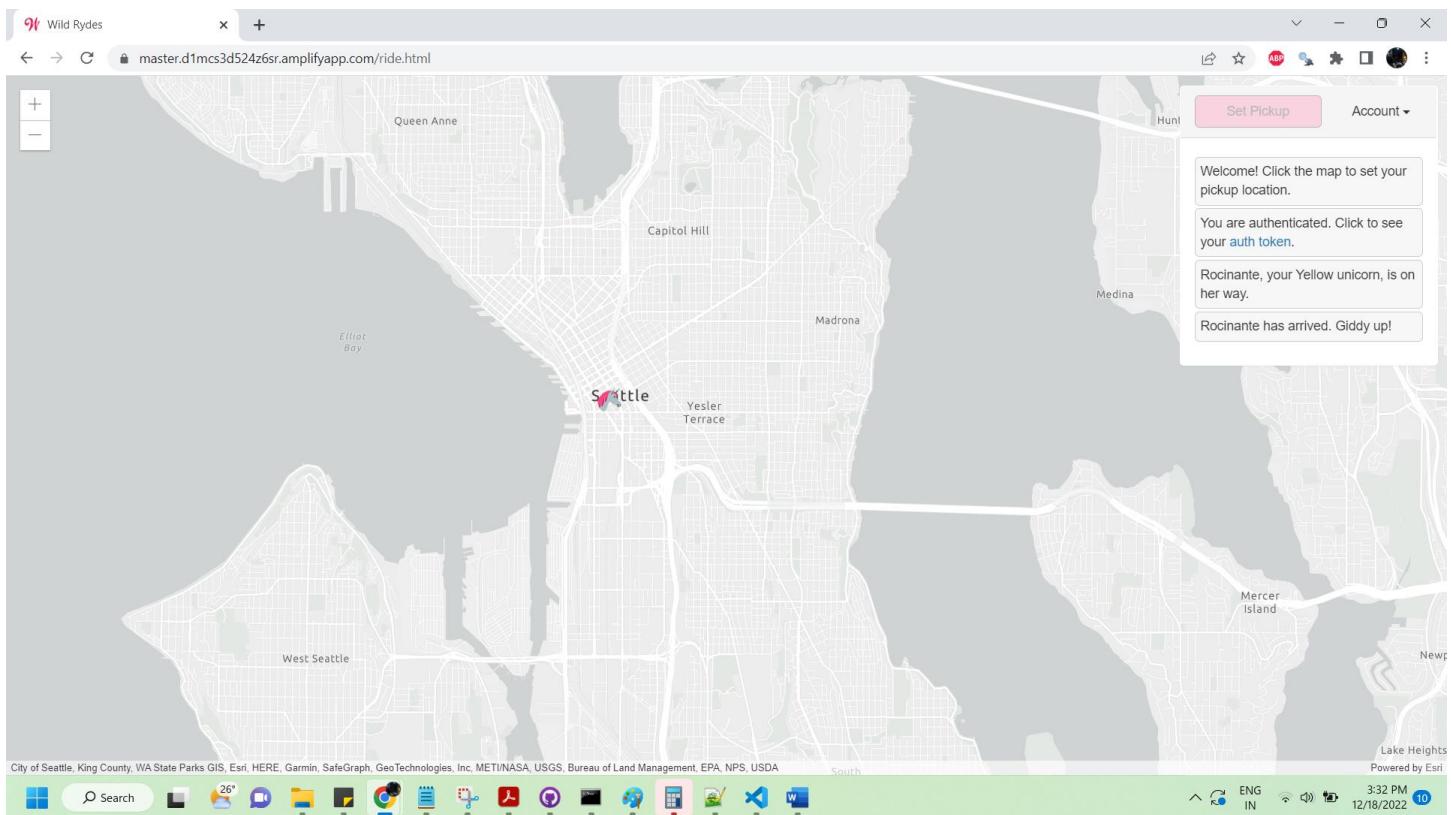
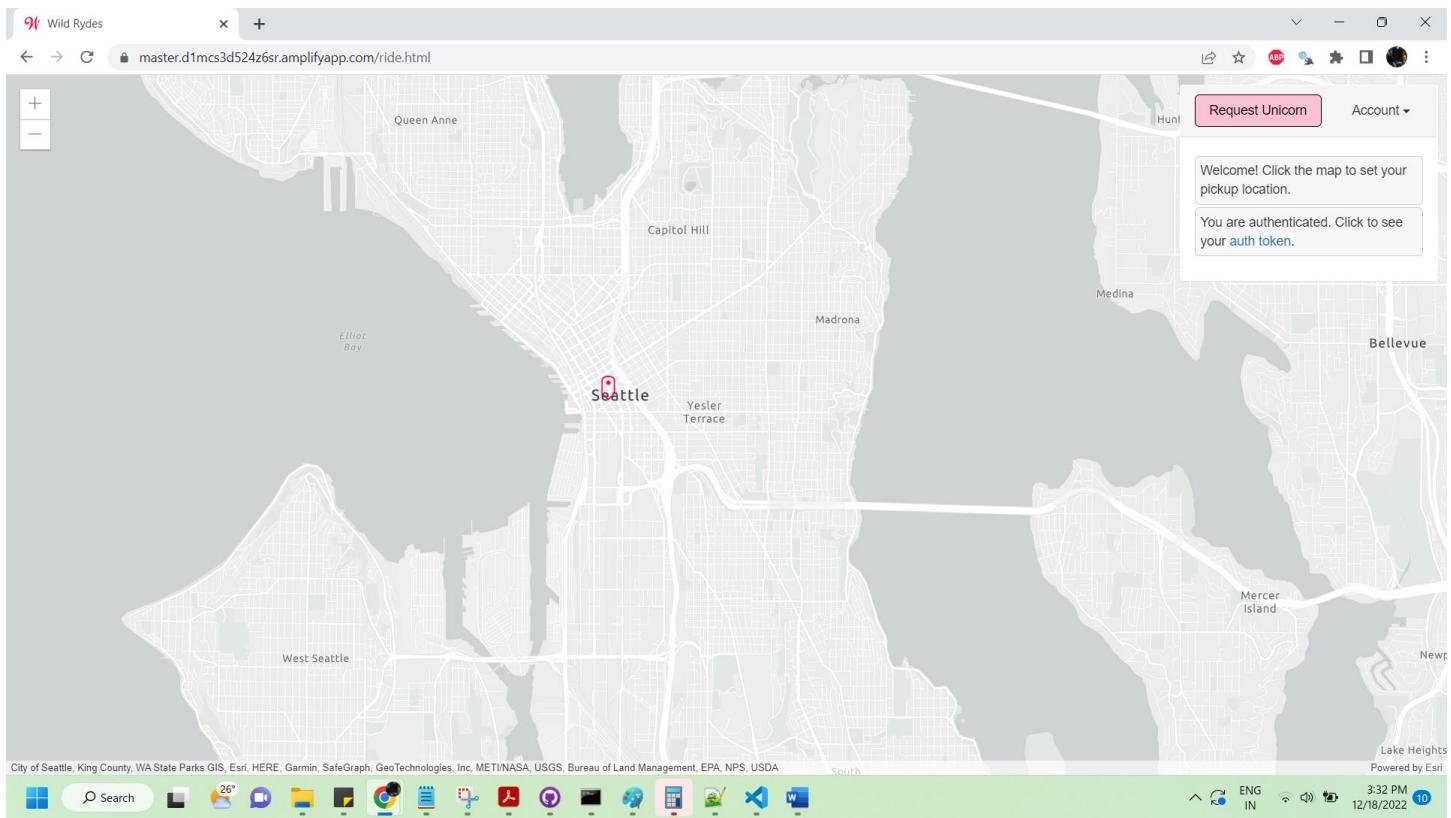
E:\GreatLakes_Cloud\CapstoneProject\aws-capstone3\wildrydes-site>
```

## Step 6: Validate the implementation

Follow below steps, to validate that, our implementation is working fine.

- Visit /ride.html under website domain.
- If we are redirected to the ArcGIS sign-in page, sign in with the user credentials of ArcGIS.
- After the map has loaded, click anywhere on the map to set a pickup location.
- Choose Request Unicorn. We should see a notification in the right sidebar that a unicorn is on its way and then see a unicorn icon fly to our pickup location.





Note: To view the map, change arcgis version from 4.3 to 4.6 in ride.html and push the code again. Open the app after redeployment.

## Cost Analysis

Problem: Cost Analysis of the implemented Solution. Assume your solution is used by 1000 users for a month, and give monthly billing estimates.

Solution: based on AWS pricing calculator, average cost per month for this solution is **\$256.423** including upfront cost (\$180) for first month, and **\$76.423** from next month onwards

Estimate summary			
Upfront cost	Monthly cost	Total 12 months cost	Currency
180	76.423	1097.08	USD

\* Includes upfront cost

Detailed Estimate						
Region	Service	Upfront	Monthly	First 12 months	Currency	Configuration summary
US East (N. Virginia)	DynamoDB provisioned capacity	180	26.39	496.68	USD	Table class (Standard), Average item size (all attributes) (1 KB), Write reserved capacity term (1 year), Read reserved capacity term (1 year), Data storage size (1 GB)
US East (N. Virginia)	Amazon API Gateway	0	0	0	USD	HTTP API requests units (millions), Average size of each request (34 KB), REST API request units (exact number), Cache memory size (GB) (None), WebSocket message units (thousands), Average message size (32 KB), Requests (0 per month), Requests (1000 per month)
US East (N. Virginia)	Amazon Cognito	0	50	600	USD	Advanced security features (Enabled), Number of monthly active users (MAU) (1000)
US East (N. Virginia)	AWS Lambda	0	0	0	USD	Architecture (x86), Architecture (x86), Amount of ephemeral storage allocated (512 MB)
US East (N. Virginia)	AWS Amplify	0	0.033	0.4	USD	Number of build minutes (1 per month), Data stored per month (1 GB)

### Acknowledgement

\* AWS Pricing Calculator provides only an estimate of your AWS fees and doesn't include any taxes that might apply. Your actual fees depend on a variety of factors, including your actual usage of AWS services.

Refer, AWS pricing estimation, for further details,

<https://calculator.aws/#/estimate?id=15b7a4ca7667d8e1b17963f494a590fb7c22399e>

## Lessons & Observations

- Learnt to create serverless web app application.
- Learnt to host the web application on a front-end web server and connect it to a backend database.
- Learnt to set up user authentication using AWS Cognito and will be able to collect and analyse user behaviour.