Creating server-less application(wild-rides):

- **Create iam user and give admin access- cli access**
- **Open terminal login with iam user with aws configer**

Create a repo in code commit:

First, create a new CodeCommit repository from within your Cloud9 terminal window:

```
aws codecommit create-repository \
  --repository-name wild-rydes
```

Clone the existing (not new) workshop repository from GitHub:

```
git clone https://github.com/aws-samples/aws-serverless-webapp-workshop.git
```

Change into the workshop repository directory:

```
cd aws-serverless-webapp-workshop
```

Split out the *WildRydesVue* code into its own branch:

```
sudo yum install git-subtree -y
git subtree split -P resources/code/WildRydesVue -b WildRydesVue
```

Note: You may get the error message `git: 'subtree' is not a git command` on Cloud9 and Amazon Linux 2. Run `sudo yum install git-subtree` as a workaround, as this is not installed by default with `git` in these environments.

Create a new directory for your CodeCommit repo:

```
mkdir ../wild-rydes && cd ../wild-rydes
```

Initialize a new git repository:

```
git init
```

Pull the *WildRydesVue* branch into your new repo:

```
git pull ../aws-serverless-webapp-workshop WildRydesVue
```

Add your CodeCommit repository as a remote:`git remote add origin codecommit://wild-rydes`

Push the code to your new CodeCommit repository:
`git push -u origin master`

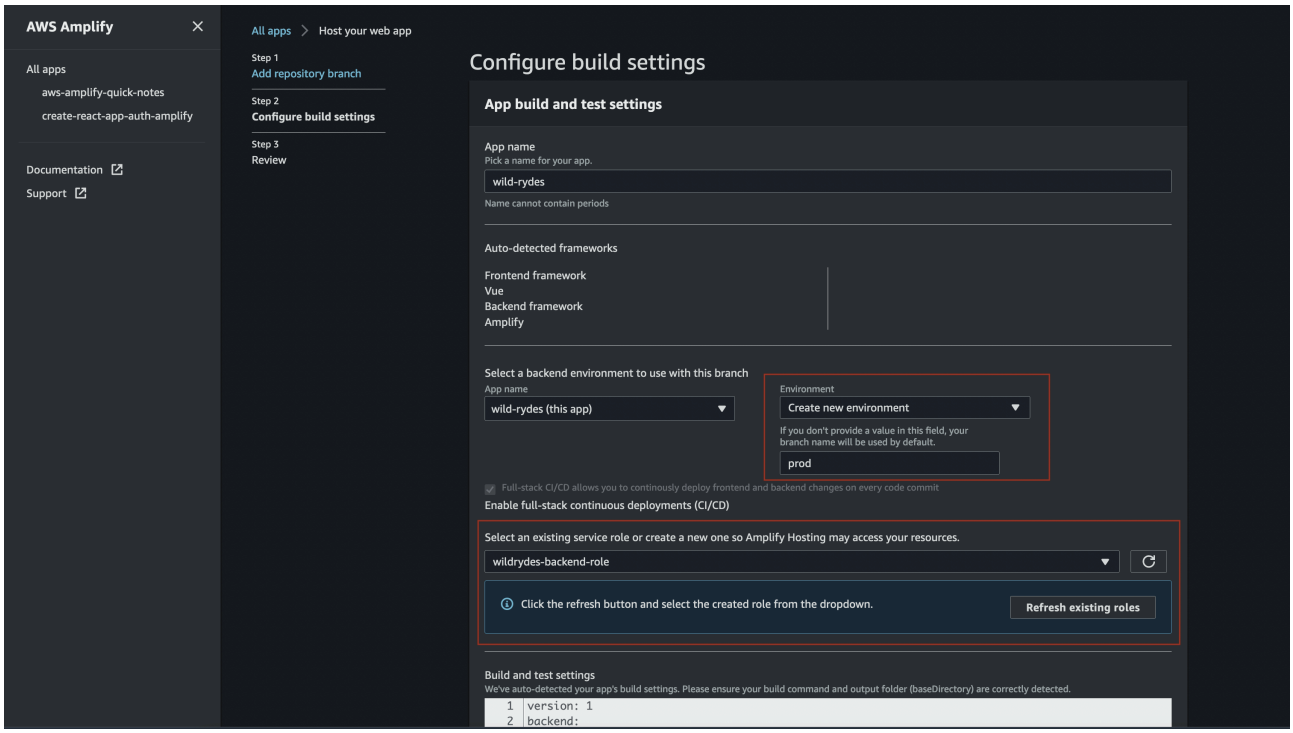
Deploy the site with AWS Amplify Console

Next you'll use the AWS Amplify Console to deploy the website you've just commited to git. The Amplify Console takes care of the work of setting up a place to store your 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.

1. Launch the Amplify Console
2. Underneath **Get Started**, you'll find a section for Amplify Hosting titled **Host your web app**. Click the **Get started** button within that section. If you are starting from the All apps page, choose **New app**, then **Host web app** in the upper right corner.

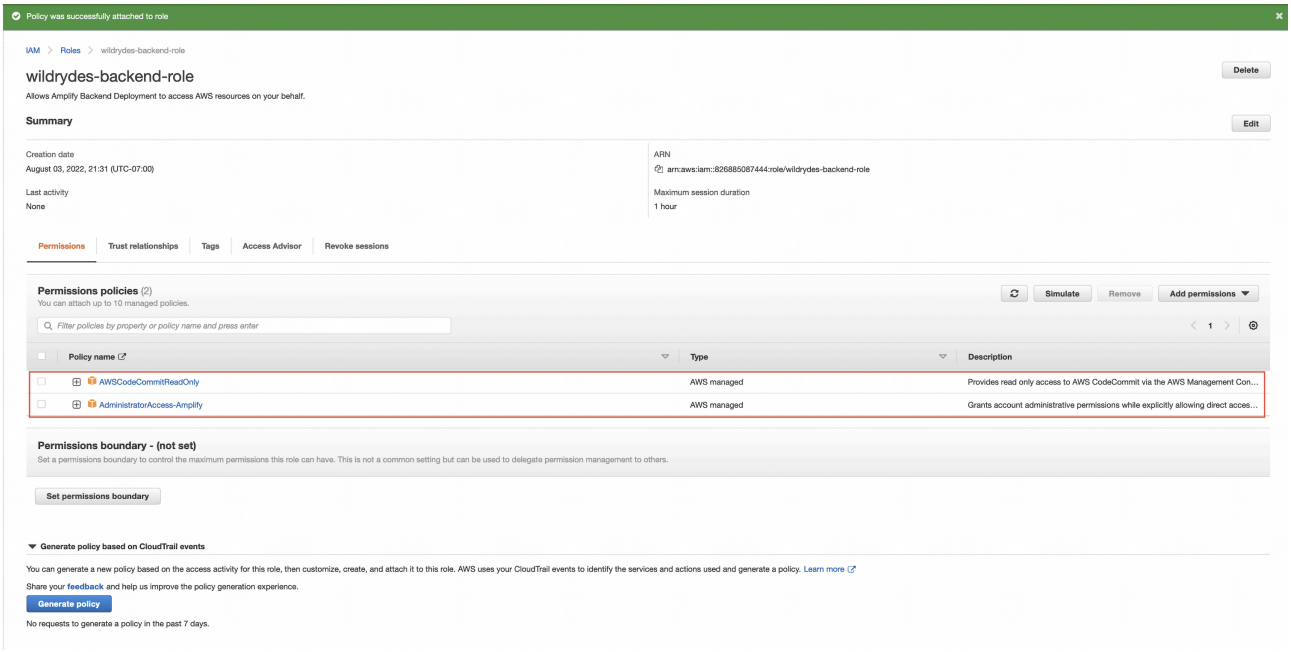
Depending on your console and account settings, you may not see the **Get Started** prompt. In this case, click on the **AWS Amplify** title link in the top of the left side navigation.

3. Connect a repository: Select **AWS CodeCommit** and choose **Continue**
4. From the drop down select the *Repository* and **master** Branch created today and select **Next**
5. Amplify will detect that the application has an existing Amplify backend. Select **Create New Environment** and name it **prod**



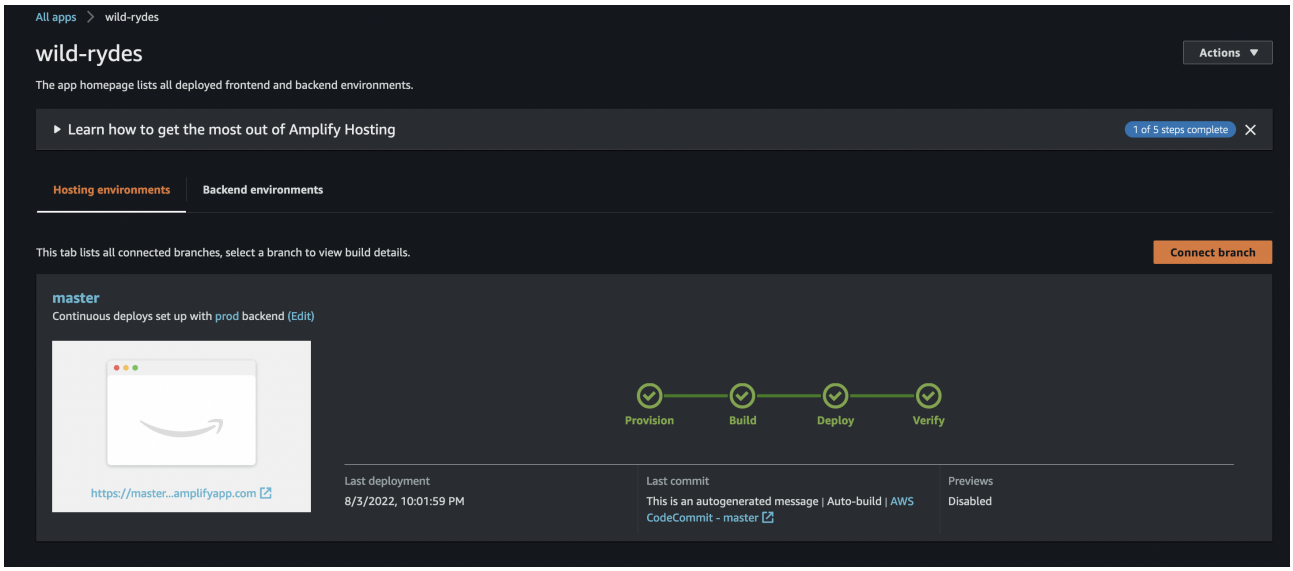
`Now you need to create a new service role with the permissions to deploy the application backend.`

1. Click on **Create new role**, check that **Amplify** is selected and click **Next permissions**, click **Next: Tags**, click **Next: Review**.
2. Give the Role a new name: `wildrydes-backend-role` and click **Create role**.
3. Search for `wildrydes-backend-role` from the search filter, and click the role name.
4. Click **Attach policies** under the ** Add Permissions** tab, search for `AWSCodeCommitReadOnly` policy, click on the checkbox next to the policy name, and click **Attach Policy**.
5. Close this tab and return to the AWS Amplify Build configure console.
6. Refresh the role list by clicking on the circular arrow button, and select the role created in the step above.



1. Select **Next**

2. On the **Review** page select **Save and deploy**
   This initial build and deploy process may take up to five minutes for Amplify Console to create the neccesary resources and to deploy your code.



Once completed, click the **site URL** to launch your Wild Rydes site.

Modify the website

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.

1. From your Cloud9 environment open the `index.html` file in the `/wild-rydes/public/` directory of the repository.
2. Modify the title line:

```
  <title>wildrydes</title>
```

So that it says:

```
<title>Wild Rydes - Rydes of the Future!</title>
```

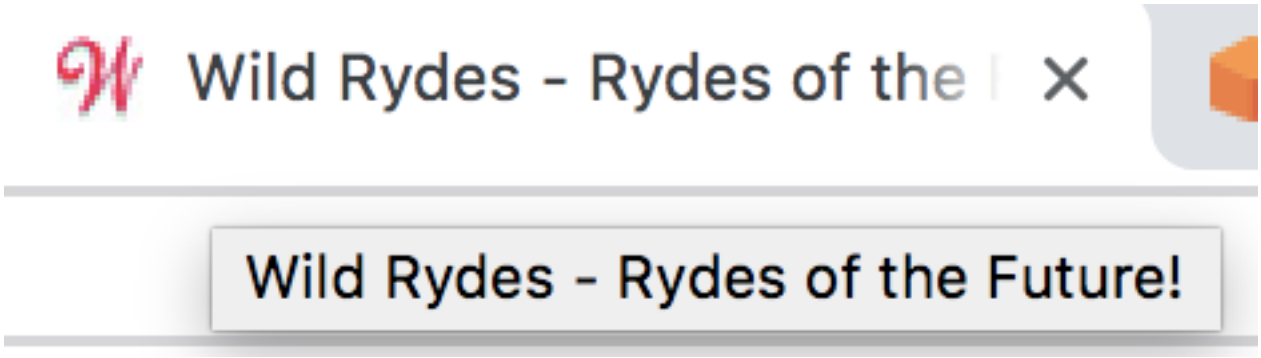
Save the file

Commit again to your git repository the changes:

```
git add .

git commit -m "updated title"

git push
```

```
Amplify Console will begin to build the site again
soon after it notices the update to the repository.
This happens pretty quickly! Head back to the Amplify
Console to watch the process.
```

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



Initialize AWS Amplify CLI

Background

The AWS Amplify Command Line Interface (CLI) is a unified toolchain to create, integrate, and manage the AWS cloud services for your app. The Amplify CLI toolchain is designed to work with

the Amplify JavaScript library as well as the AWS Mobile SDKs for iOS and Android.

AWS Amplify Authentication module provides Authentication APIs and building blocks for developers who want to create user authentication experiences.

Install the Amplify CLI by running the following command from within your Cloud9 terminal window:

amplify configure

Install the Amplify CLI by running the following command from within your Cloud9 terminal window:

Sudo npm install -g @aws-amplify/cli

Configure your default AWS profile.

```
echo '[profile default]' > ~/.aws/config
```

Make sure you are in the root `wild-rydes` directory of the repository:

Initialize amplify CLI by executing the following command:

```
amplify init
```

The terminal will now take a few moments to initialize your project:

```
Administrator:~/environment/wild-rydes (master) $ amplify init
Note: It is recommended to run this command from the root of your app directory
? Enter a name for the environment prod
? Choose your default editor: Visual Studio Code
Using default provider  awscloudformation

For more information on AWS Profiles, see:
https://docs.aws.amazon.com/cli/latest/userguide/cli-multiple-profiles.html

? Do you want to use an AWS profile? Yes
? Please choose the profile you want to use default
⠼ Initializing project in the cloud...
```

Verify that the initialization has finished by entering the following command. Version 4.29.3 or greater should be installed.

```
amplify version
```

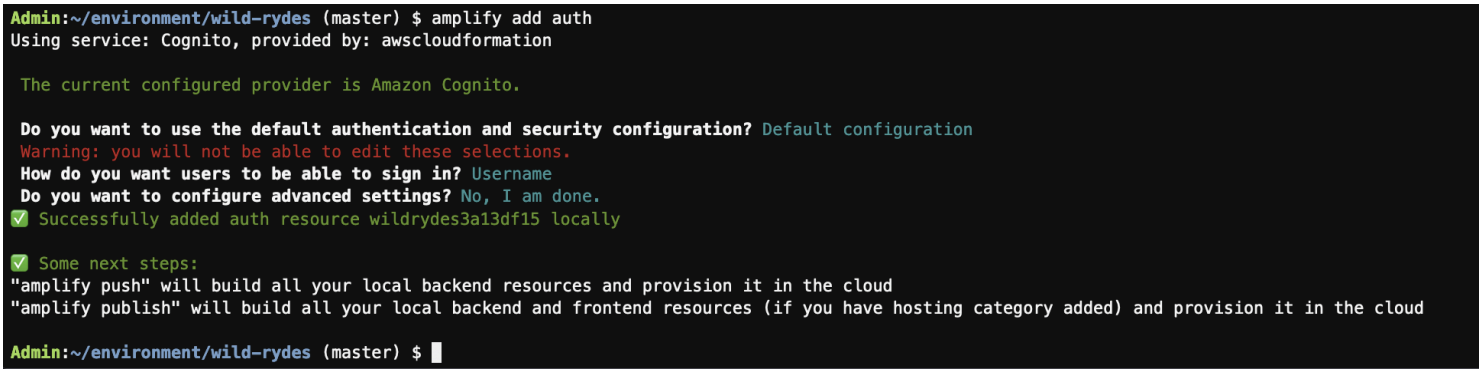
Amazon Cognito

```
amplify add auth
```

The AWS Amplify CLI will now run through the set up for Amazon Cognito, select the following:

```
> Do you want to use the default authentication and
security configuration? "Default configuration"
> How do you want users to be able to sign in?
"Username"
> Do you want to configure advanced settings? "No, I
am done."
```

Once configuration completes you see the following confirmation:



Committing your code updates to provision your Amplify backend resources in the cloud and kick off a new build.

Commit the changes to your git repository:

```
git add .

git commit -m "Configure Cognito"

git push
```

Amplify Console picks up the changes and begins building and deploying your web application. User Pool will get created in Amazon Cognito console once the application is deployed successfully.

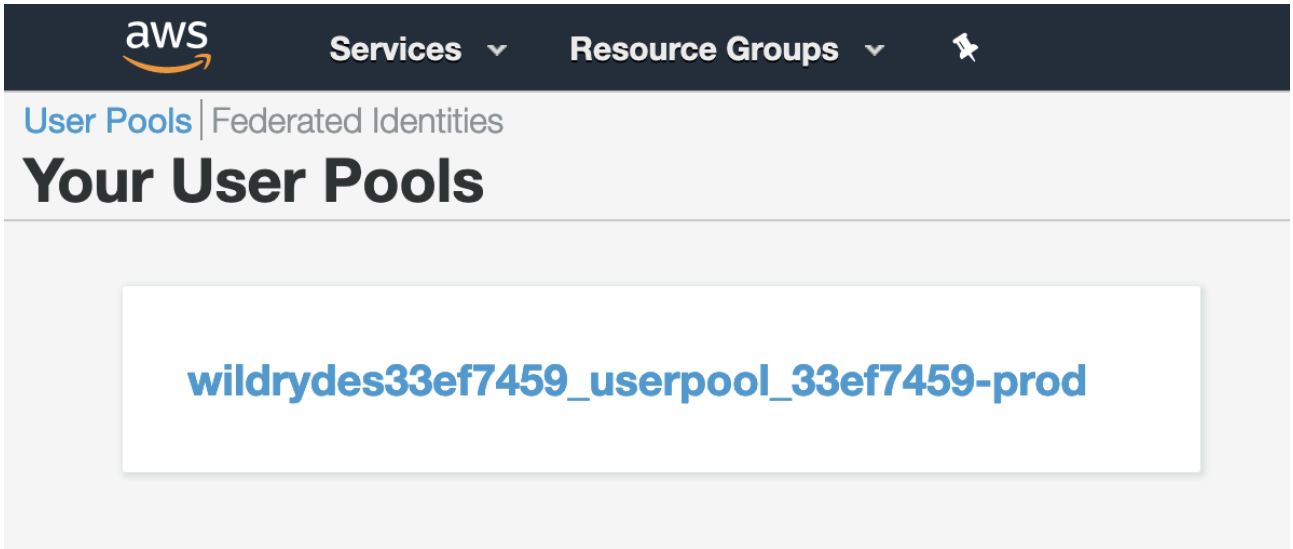
Check Your User Pool's App Client

A Cognito Userpool and a new App client has been created by the AWS Amplify build. Let's take a look at this app client.

Go to the Amazon Cognito Console

Choose **Manage User Pools**.

Here you will see a new userpool generated by the AWS Amplify CLI that looks something similar to the example below:



Click the new user pool to open the Pool Details page

From the Pool Details page, select **App clients** from the **General settings** section in the left navigation bar.

You will see that a new App client has been generated. Your web application is configured to use this App client via a config file located in `wild-rydes/src/aws-exports.js`.

How it Works:

Rather than configuring each service through a constructor or constants file, Amplify supports configuration through a centralized file called aws-exports.js which defines all the regions and service endpoints to communicate. Whenever you run `amplify push` or rebuild your web application by running a `git commit`, this file is automatically created, allowing you to focus on your application code. The Amplify CLI places this file in the appropriate source directory configured with amplify init.

❗ You won't see updates to this file in your local file store because it is included in the *.gitignore* file.

Implementation Validation

Visit `/auth` under your website domain, or choose the **Giddy Up!** button on the homepage of your site.

Click on the **Create Account** link at the bottom of the sign in box.

Complete the registration form and choose **Create Account**. You can use your 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 you entered for later. You should see an alert that confirms that your user has been created.

If you get an `Authentication Error` this is likely because your changes have not finished deploying.

Confirm your new user using one of the two following methods.

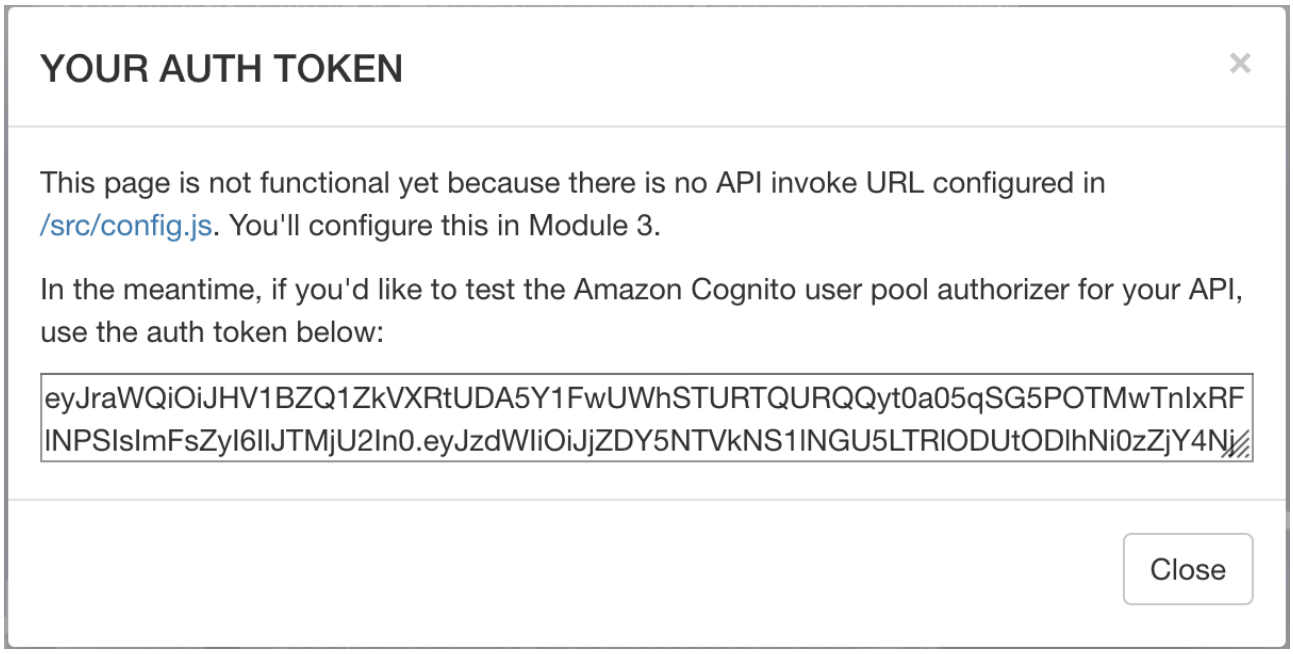
If you used an email address you control, you can complete the account verification process by entering the verification code that is emailed to you. Please note, the verification email may end up in your spam folder. For real deployments we recommend configuring your user pool to use Amazon Simple Email Service to send emails from a domain you own.

If you used a dummy email address, you must confirm the user manually through the Cognito console.

> From the AWS console, click Services then select **Cognito** under Security, Identity & Compliance.
> Choose **Manage your User Pools**
> Select the user pool prefixed with `wildrydes` and click **Users and groups** in the left navigation bar.
> You should see a user corresponding to the email address that you submitted through the registration page. 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 verrification code or the Cognito console, click on the **back to sign in** link or refresh the **/auth** page and log in using the email address and password you entered during the registration step.

If successful you should be redirected to `/ride`. You should see a notification that the API is not configured.



Create an Amazon DynamoDB Table

Use the Amazon DynamoDB Console to create a new DynamoDB table. Call your table `Rides` and give it a partition key called `RideId` with type String. The table name and partition key are case sensitive. Make sure you use the exact IDs provided. Use the defaults for all other settings.

After you've created the table, record the ARN for use in the next step.

1. Go to the Amazon DynamoDB Console
2. Choose **Create table**.
3. Enter `Rides` for the **Table name**. This field is case sensitive.
4. Enter `RideId` for the **Partition key** and select **String** for the key type. This field is case sensitive.
5. Choose the **Default settings** button for Table settings and choose **Create table**

DynamoDB > Tables > Create table

Create table

Table details Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

Rides

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.

RideId | 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 your table. You can modify these settings now or after your table has been created.

( ) Customize settings
Use these advanced features to make DynamoDB work better for your needs.

Default table settings

These are the default settings for your new table. You can change some of these settings after creating the table.

Setting	Value	Editable after creation
Capacity mode	Provisioned	Yes
Read capacity	5 RCU	Yes
Write capacity	5 WCU	Yes
Auto scaling	On	Yes
Local secondary indexes	-	No
Global secondary indexes	-	Yes
Encryption key management	Owned by Amazon DynamoDB	Yes
Table class	DynamoDB Standard	Yes

Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel | Create table

6. Once the table is Active, Click on “Rides” to open the table settings and under Overview > General information > Additional info section, you will find the **ARN** of the table. Record this ARN as you will use this in the next section.

Create an IAM Role for Your Lambda Function

Background

Every Lambda function has an IAM role associated with it. This role defines what other AWS services the function is allowed to interact with. In this workshop, you create an IAM role that grants your Lambda function permission to write logs to Amazon CloudWatch Logs and access to write items to your DynamoDB table.

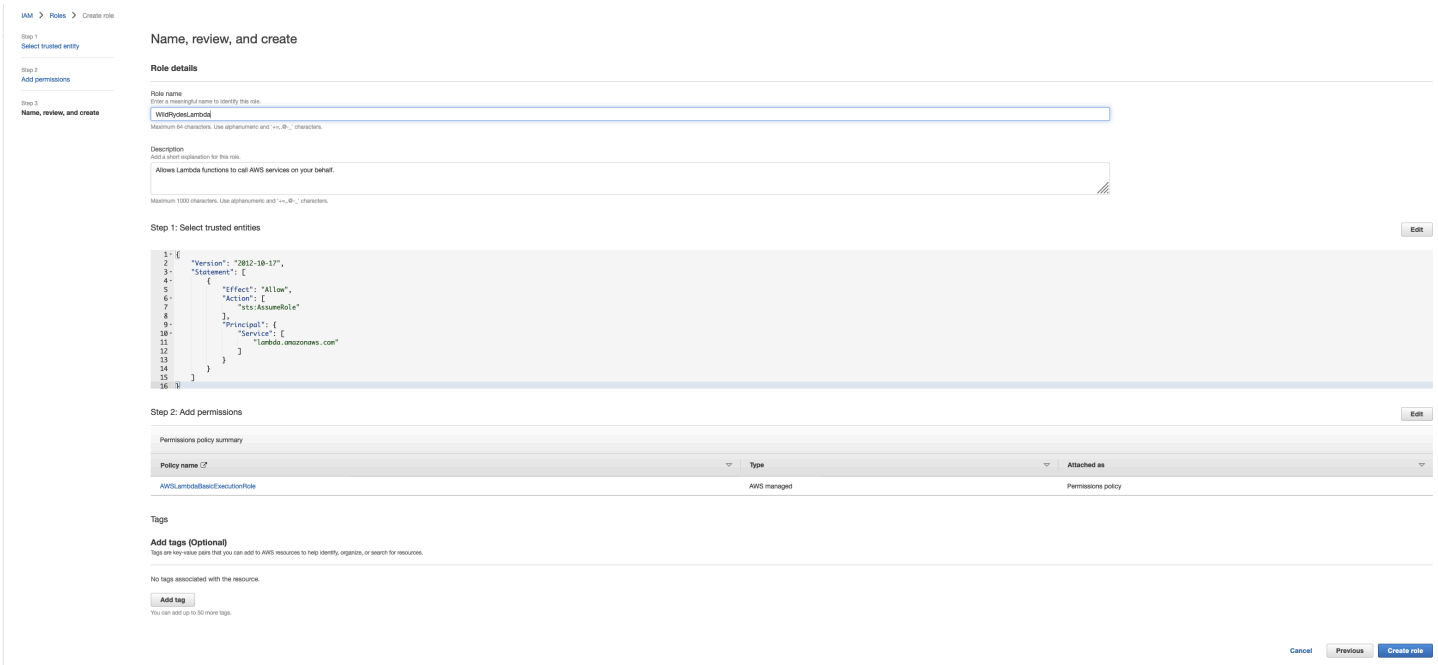
High-Level Instructions

Use the IAM console to create a new role. Name it `WildRydesLambda` and select AWS Lambda for the role type. You’ll need to attach policies that grant your function permissions to write to Amazon CloudWatch Logs and put items to your DynamoDB table.

Attach the managed policy called `AWSLambdaBasicExecutionRole` to this role to grant the necessary CloudWatch Logs permissions. Also, create a custom inline policy for your role that allows the `dynamodb:PutItem` action for the table you created in the previous section.

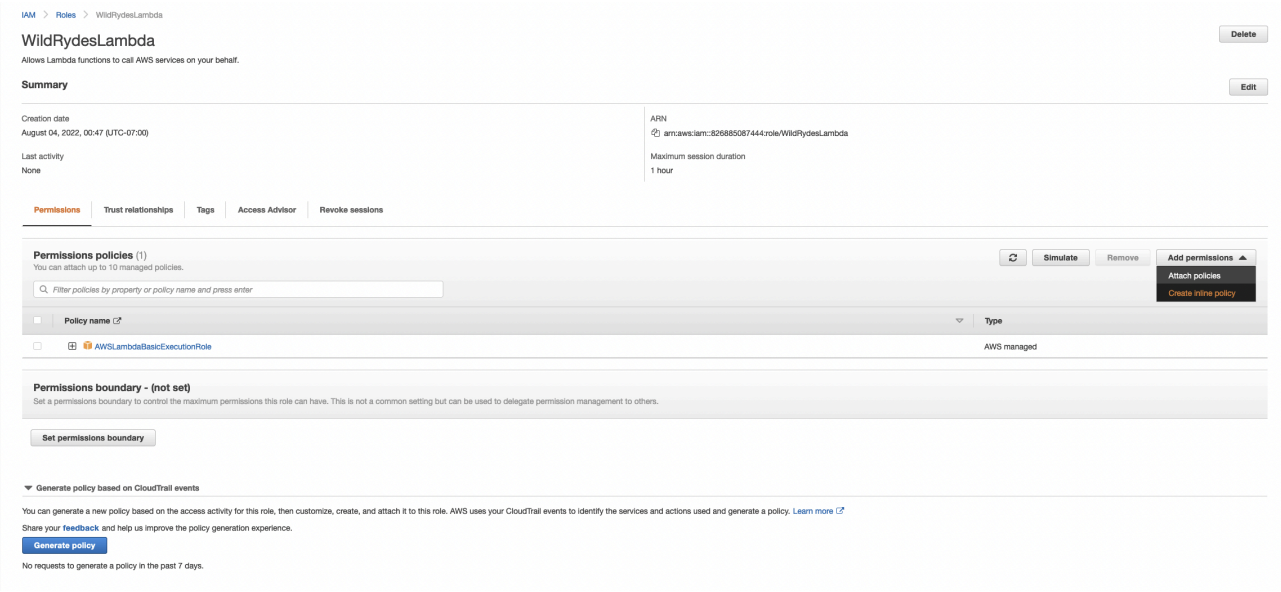
- Go to the AWS IAM Console
- Select **Roles** in the left navigation bar and then choose **Create role**.

- Under Use case, Select **Lambda** from the AWS services, then click **Next**
- Begin typing `AWSLambdaBasicExecutionRole` in the **Filter** text box and check the box next to that managed role, then Click **Next**
- Enter `WildRydesLambda` for the **Role name**. Add any tags that you wish.
- Choose **Create role**.

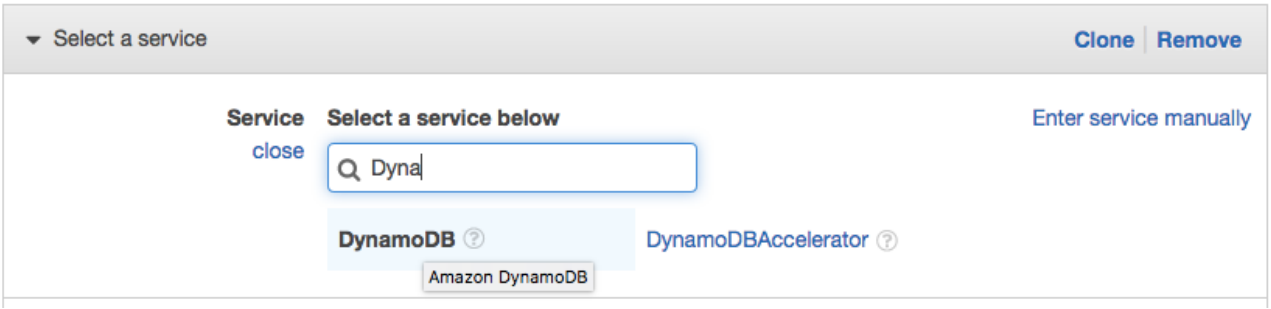


Next you need to add permissions to the role so that it can access your DynamoDB table.

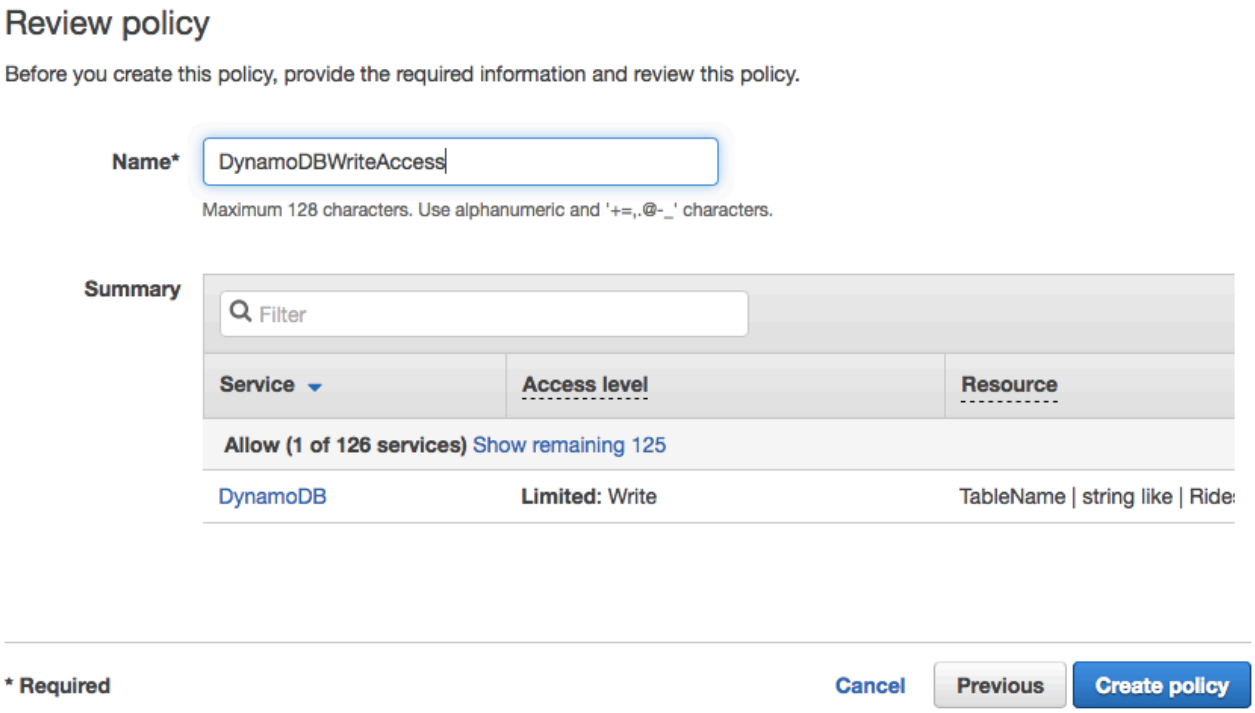
1. While in the IAM Console on the roles page type `WildRydesLambda` into the filter box and click the role name.
2. Under the Add Permissions tab, choose the **create inline policy**



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 you 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**.



Create a Lambda Function for Handling Requests

Background

AWS Lambda runs your code in response to events such as an HTTP request. In this step you build a function that processes API requests from the web application to dispatch a unicorn. In the next module you use Amazon API Gateway to create a RESTful API that exposes an HTTP endpoint that can be invoked from your users' browsers. Then you connect the Lambda function you create in this step to that API to create a fully functional backend for your web application.

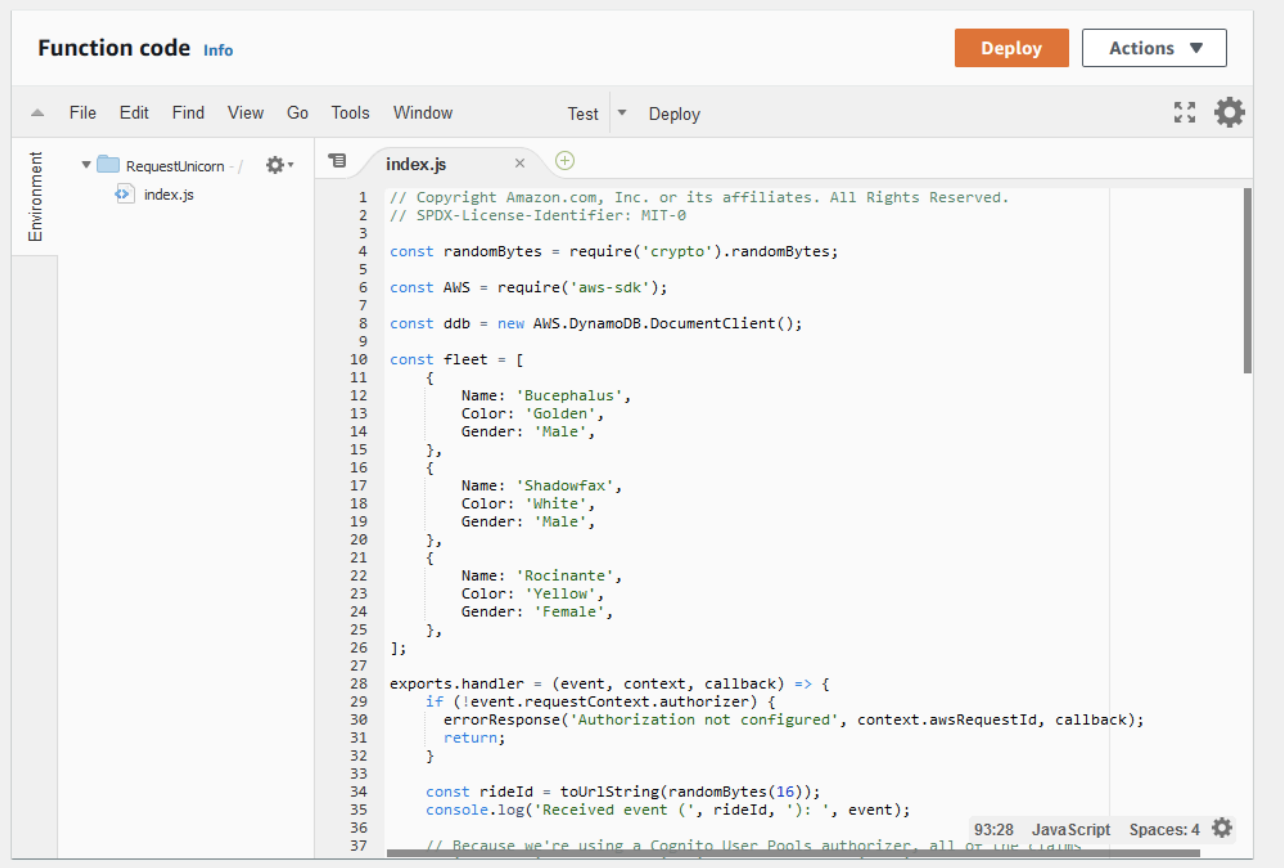
High-Level Instructions

Use the AWS Lambda console to create a new Lambda function called `RequestUnicorn` that processes API requests. Copy and paste this example implementation into the AWS Lambda console's editor for your function code.

Configure your function to use the `WildRydesLambda` IAM role you created in the previous section.

1. Go to the AWS Lambda console
2. Click **Create function**.
3. Keep the default **Author from scratch** card selected.
4. Enter `RequestUnicorn` in the **Name** field.
5. Select **Node.js 18.x** for the **Runtime**.
6. Expand *Change default execution role* under **Permissions**.
7. Ensure `Use an existing role` is selected from the **Role** dropdown.
8. Select `WildRydesLambda` from the **Existing Role** dropdown.

9. Choose **Create function**.
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.

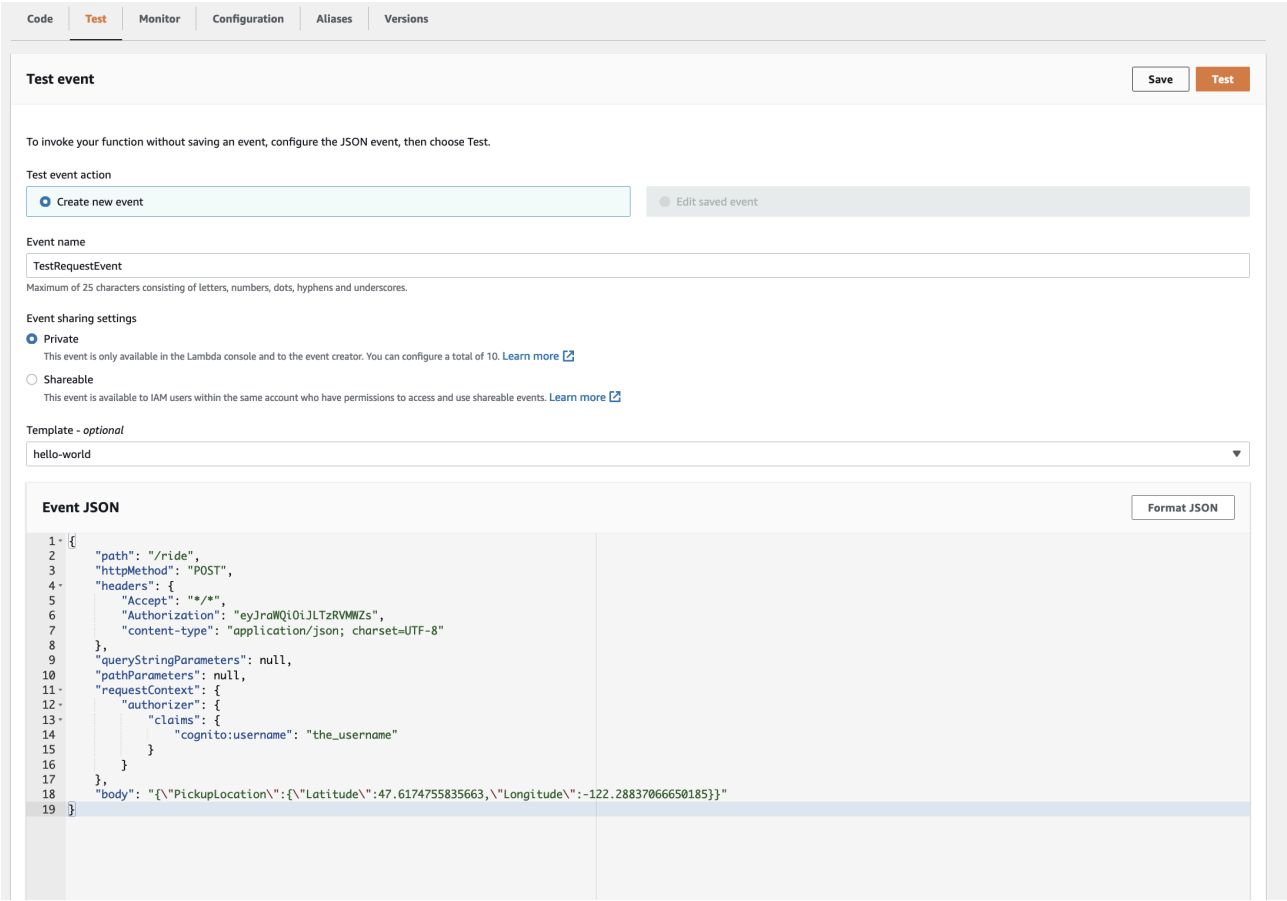


1. Click **Deploy** in the upper right above the code editor.

Implementation Validation

For this module you will test the function that you built using the AWS Lambda console. In the next module you will add a REST API with API Gateway so you can invoke your function from the browser-based application that you deployed in the first module.

1. From Test Tab, Configure test event.



Keep **Create new test 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\":-122.288370666
        50185}}"
}
```


Click **Save**. Click **Test**.
From Test Tab, 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\":\"1h0zDZ-6KLZaEQCPyqTxeQ\",\"Unicorn\":
        {\"Name\":\"Shadowfax\",\"Color\":\"White\",\"Gender\":\"Male\"},
        \"UnicornName\":\"Shadowfax\",\"Eta\":\"30 seconds\",\"Rider\":
        \"the_username\"}",
  "headers": {
    "Access-Control-Allow-Origin": "*"
  }
}
```

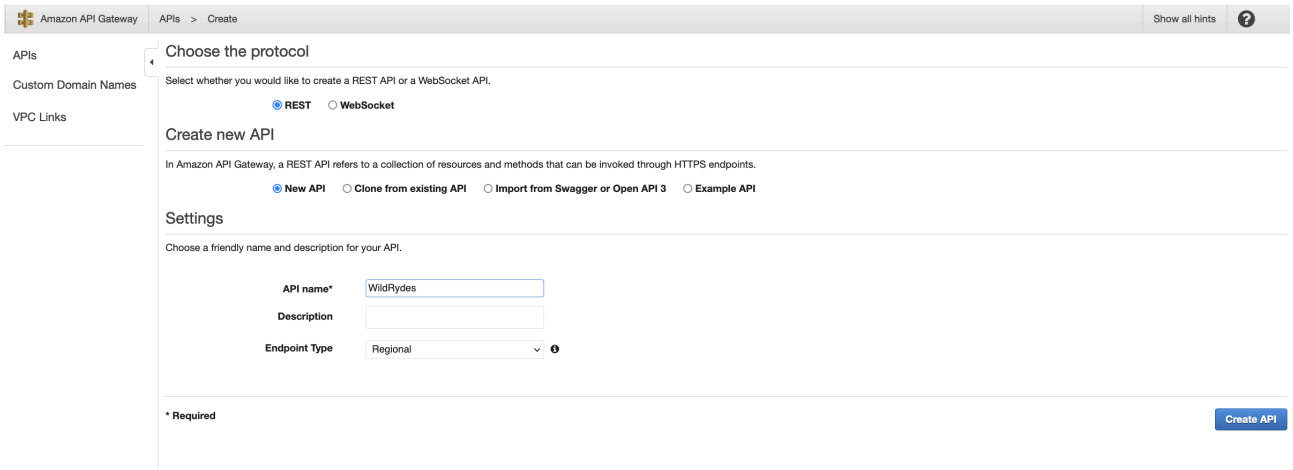

Create a New REST API

Use the Amazon API Gateway console to create a new API named `WildRydes`.

1. Go to the Amazon API Gateway Console, click **Create API**
2. On the **REST API** card, choose **Build**.

3. In the section Create new API select **New API** to clear the example API definition.
4. Enter `WildRydes` for the **API Name**.
5. Select `Regional` from the **Endpoint Type** dropdown.

   Choose **Create API**



Create a Cognito User Pools Authorizer

Background

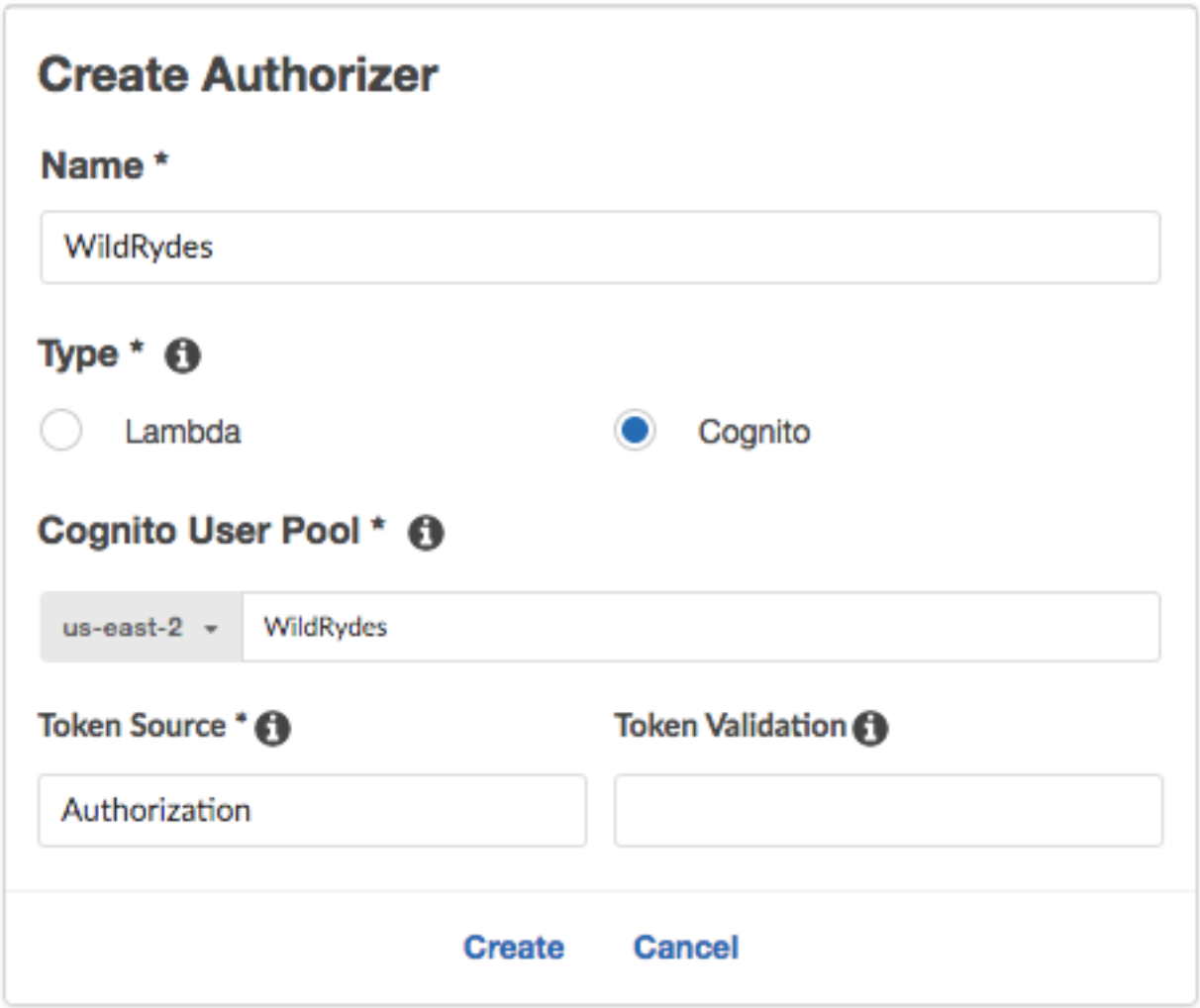
Amazon API Gateway can use the JWT tokens returned by Cognito User Pools to authenticate API calls. In this step you'll configure an authorizer for your API to use the user pool you created in User Management.

High-Level Instructions

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

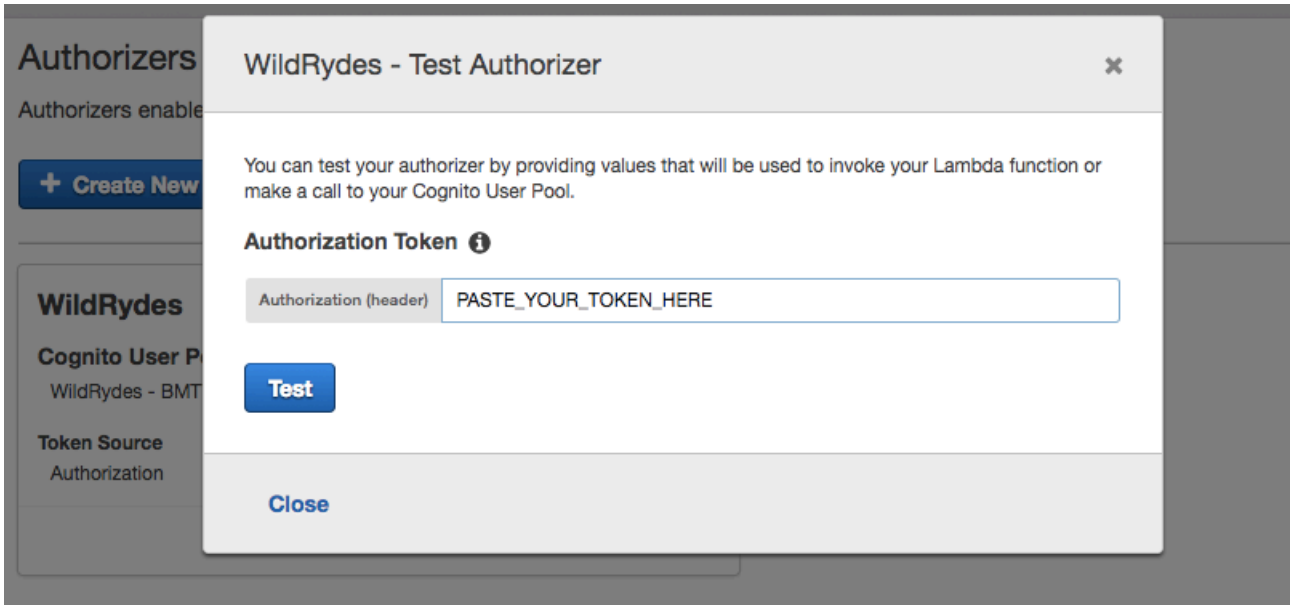
1. Under your newly created API, choose **Authorizers**.
2. Choose **Create New Authorizer**.

3. Enter `WildRydes` for the Authorizer name.
4. Select **Cognito** for the type.
5. In the Region drop-down under **Cognito User Pool**, select the Region where you created your Cognito user pool in the User Management module (by default the current region should be selected).
6. Enter `wildrydes` in the **Cognito User Pool** input, the name will auto-complete and allow you to select the name of the user pool that was generated when the user pool was created.
7. Enter `Authorization` for the **Token Source**.
8. Choose **Create**.



Verify your authorizer configuration

1. Open a new browser tab and visit `/ride` under your website's domain.
2. If you are redirected to the sign-in page, sign in with the user you created in the last module. You will be redirected back to `/ride`.
3. Copy the auth token from the notification on the `/ride`,
4. Go back to previous tab where you have just finished creating the Authorizer
5. Click **Test** at the bottom of the card for the authorizer.
6. Paste the auth token into the **Authorization Token** field in the popup dialog

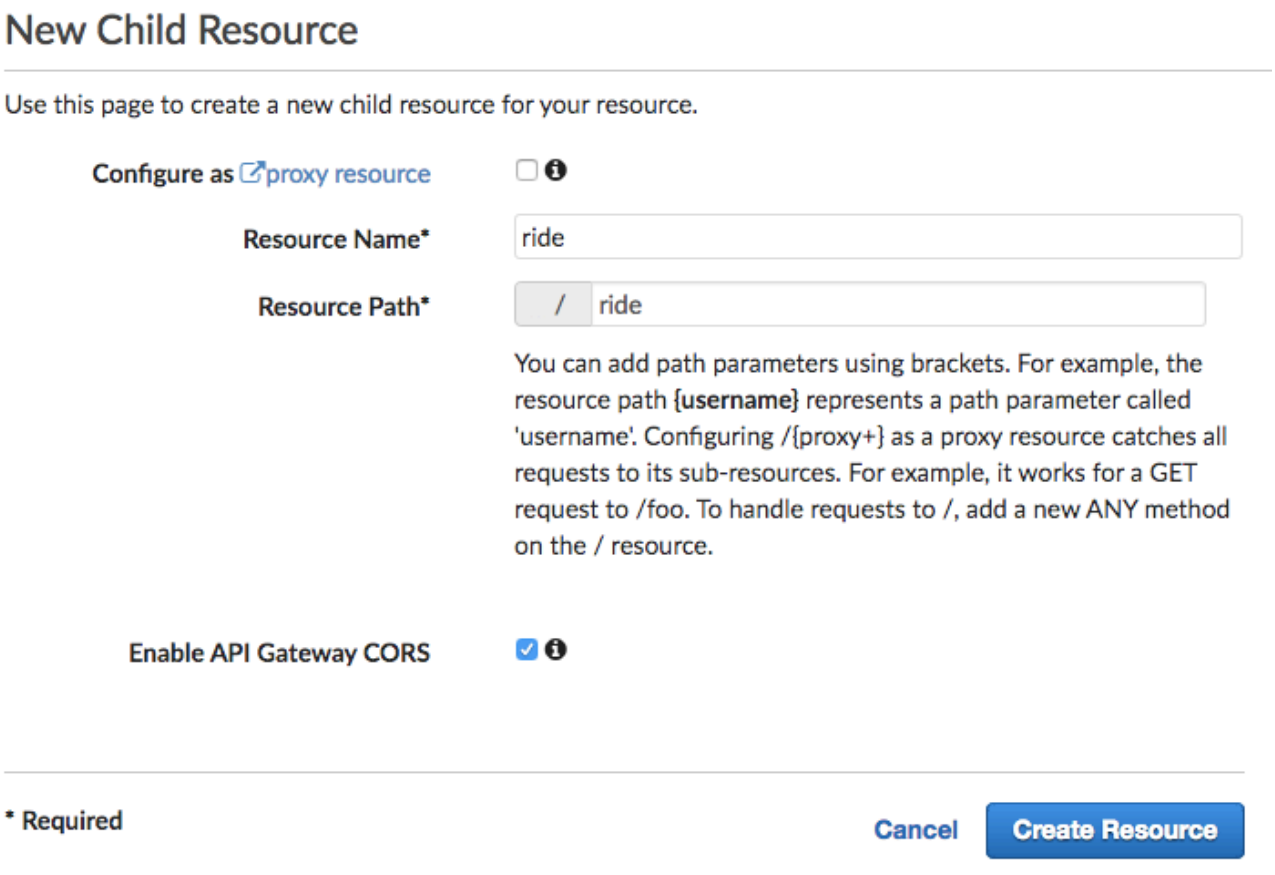


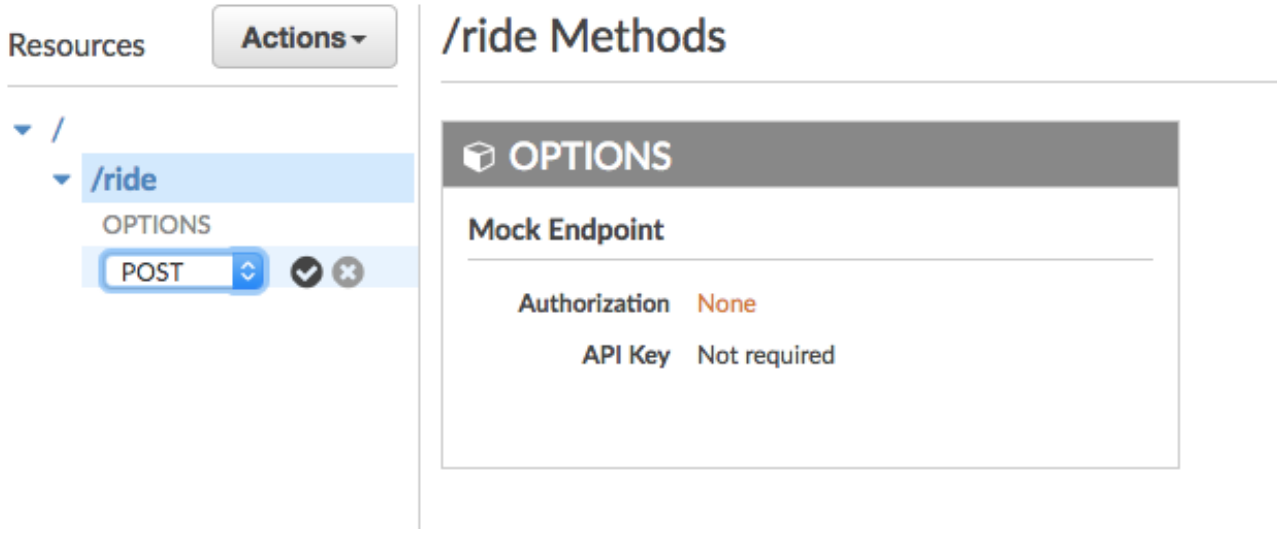
Click **Test** button and verify that the response code is 200 and that you see the claims for your user displayed.

Create a new resource and method

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

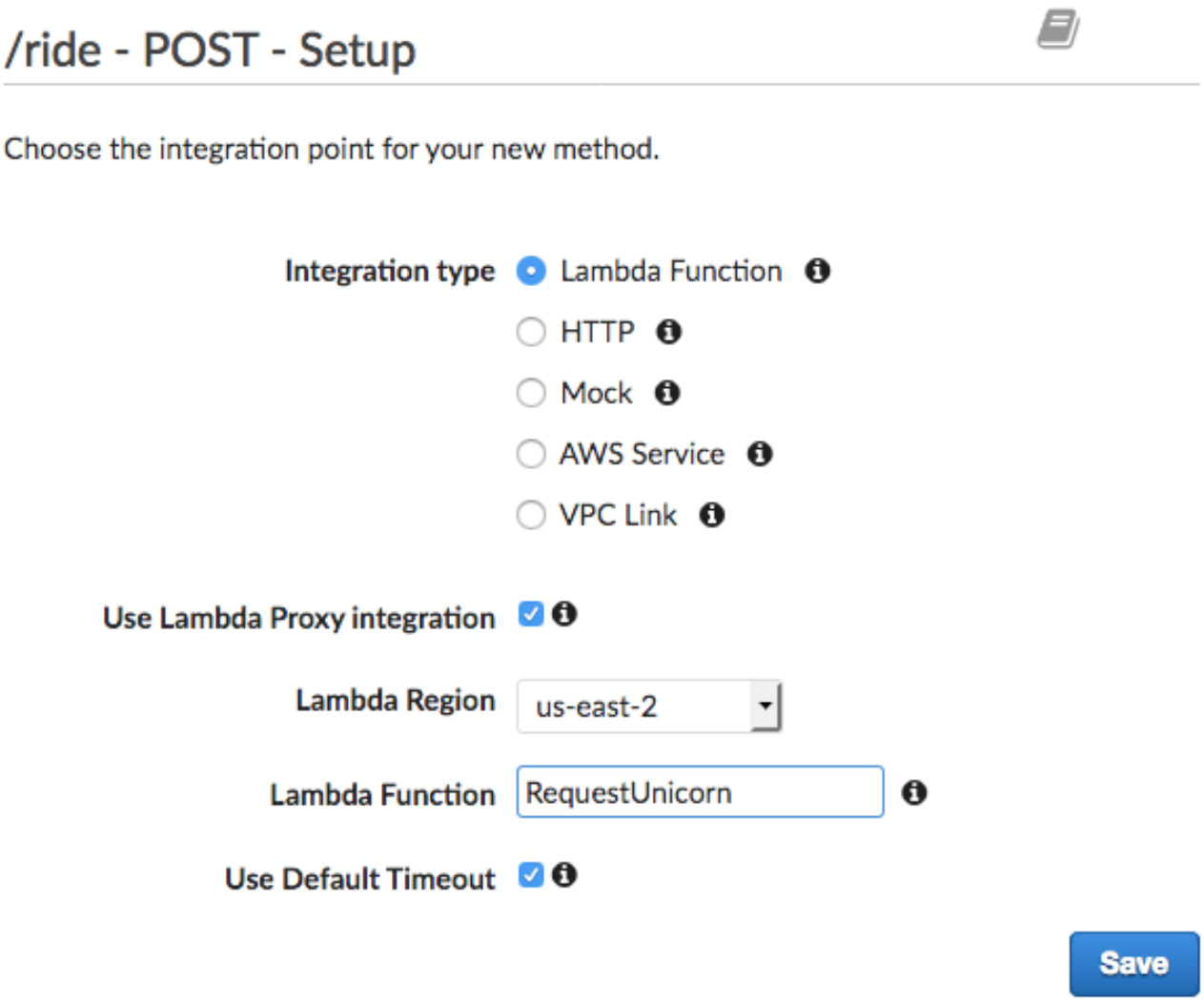
1. In the left nav, click on **Resources** under your WildRydes API.
2. From the **Actions** dropdown select **Create Resource**.
3. Enter `ride` as the **Resource Name**.
4. Ensure the **Resource Path** is set to `ride`.
5. Select **Enable API Gateway CORS** for the resource.
6. Choose **Create Resource**.

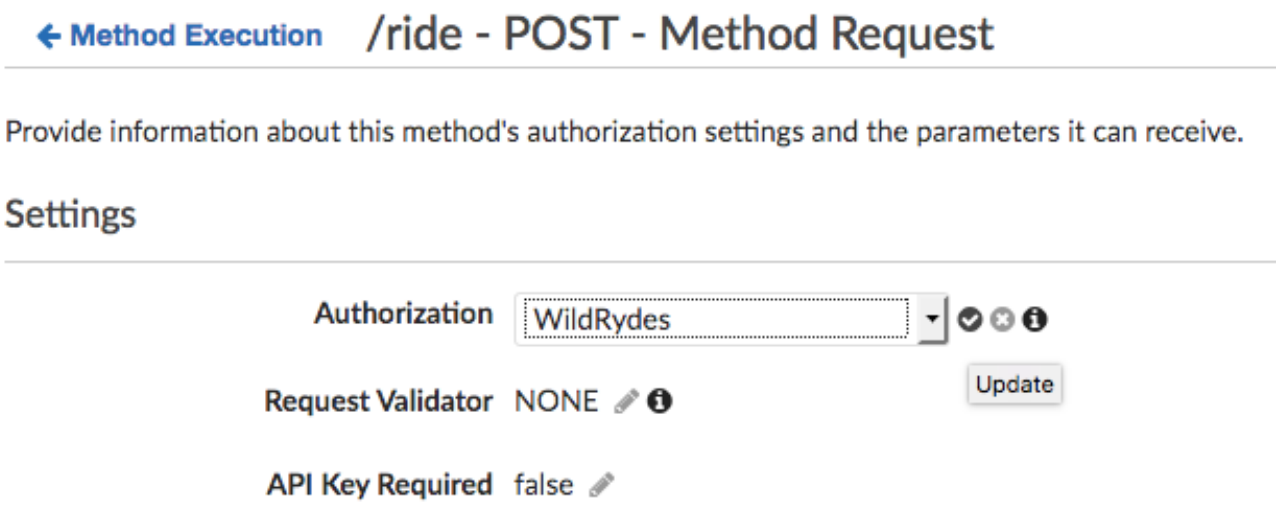




1. With the newly created `/ride` resource selected, from the **Action** dropdown select **Create Method**.
2. Select `POST` from the new dropdown that appears, then **click the checkmark**.
3. Select **Lambda Function** for the integration type.
4. Check the box for **Use Lambda Proxy integration**.
5. Select the Region you are using for **Lambda Region**.
6. Enter the name of the function you created in the previous module, `RequestUnicorn`, for **Lambda Function**.
7. Choose **Save**. Please note, if you get an error that you function does not exist, check that the region you selected matches the one you used in the previous module.
8.
9. When prompted to give Amazon API Gateway permission to invoke your function, choose **OK**.
10. Select the **Method Request** card.
11. Choose the pencil icon next to **Authorization**.

12. Select the WildRydes Cognito user pool authorizer from the drop-down list, and click the checkmark icon.





Deploy Your API

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

1. In the **Actions** drop-down list select **Deploy API**.
2. Select **[New Stage]** in the **Deployment stage** drop-down list.
3. Enter `prod` for the **Stage Name**.
4. Choose **Deploy**.
5. Record the **Invoke URL**. You will use it in the next section.

Update the Website Config

Update the */src/config.js* file in your website deployment to include the invoke URL of the stage you just created. You 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 your site's */src/config.js* file. Make sure when you update the config file it still contains the updates you made in the previous module for your Cognito user pool.

1. On your Cloud9 development environment open `src/config.js`
2. Update the **invokeUrl** setting under the **api** key in the config.js file. Set the value to the **Invoke URL** for the deployment stage your created in the previous section. An example of a complete `config.js` file is included below. **Note**: The actual URL in your file will be different.

```
module.exports = {
    api: {
        invokeUrl: 'https://rfk14x9w40.execute-api.us-
        east-1.amazonaws.com/prod'
    }
}
```

Save the modified file making sure the filename is still `config.js`.

Commit the changes to your git repository:

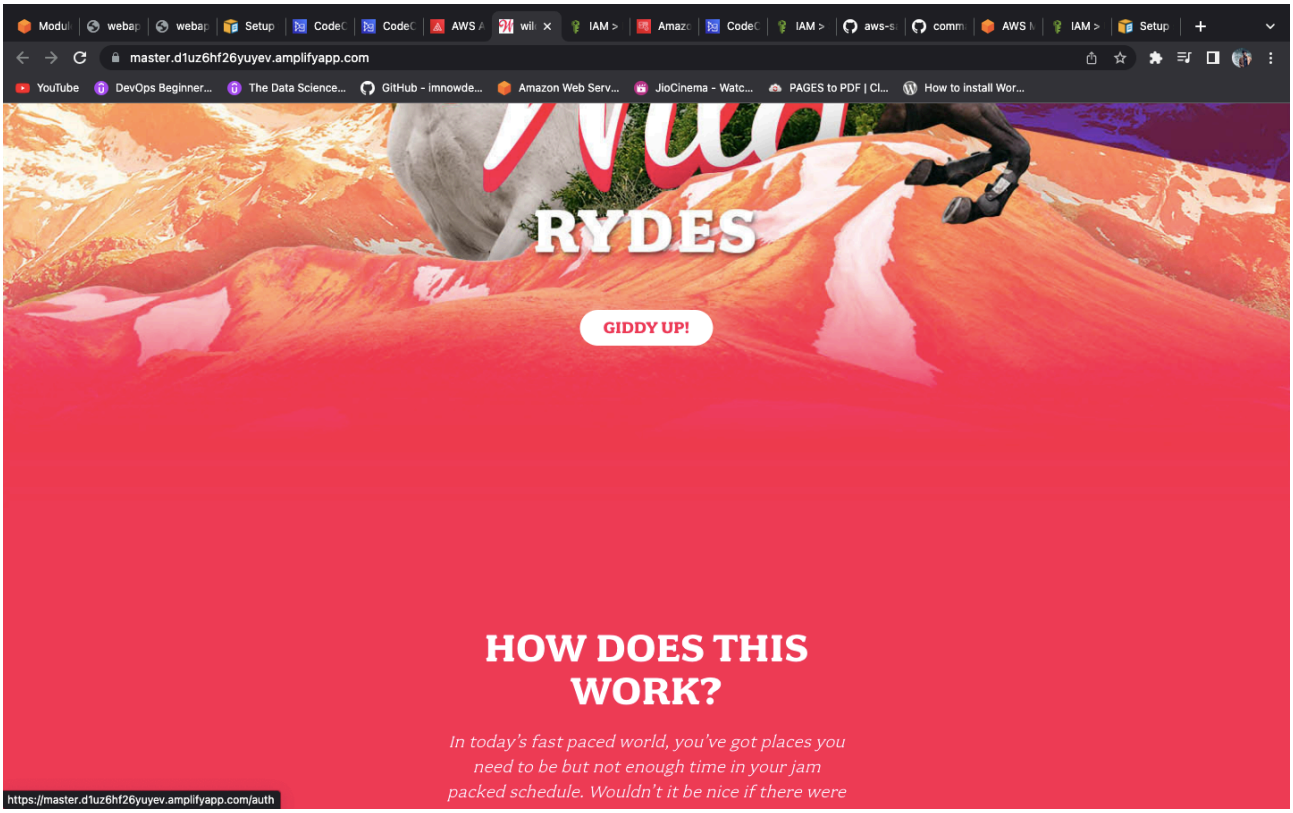
```
git add src/config.js

git commit -m "Configure API invokeURL"

git push
```

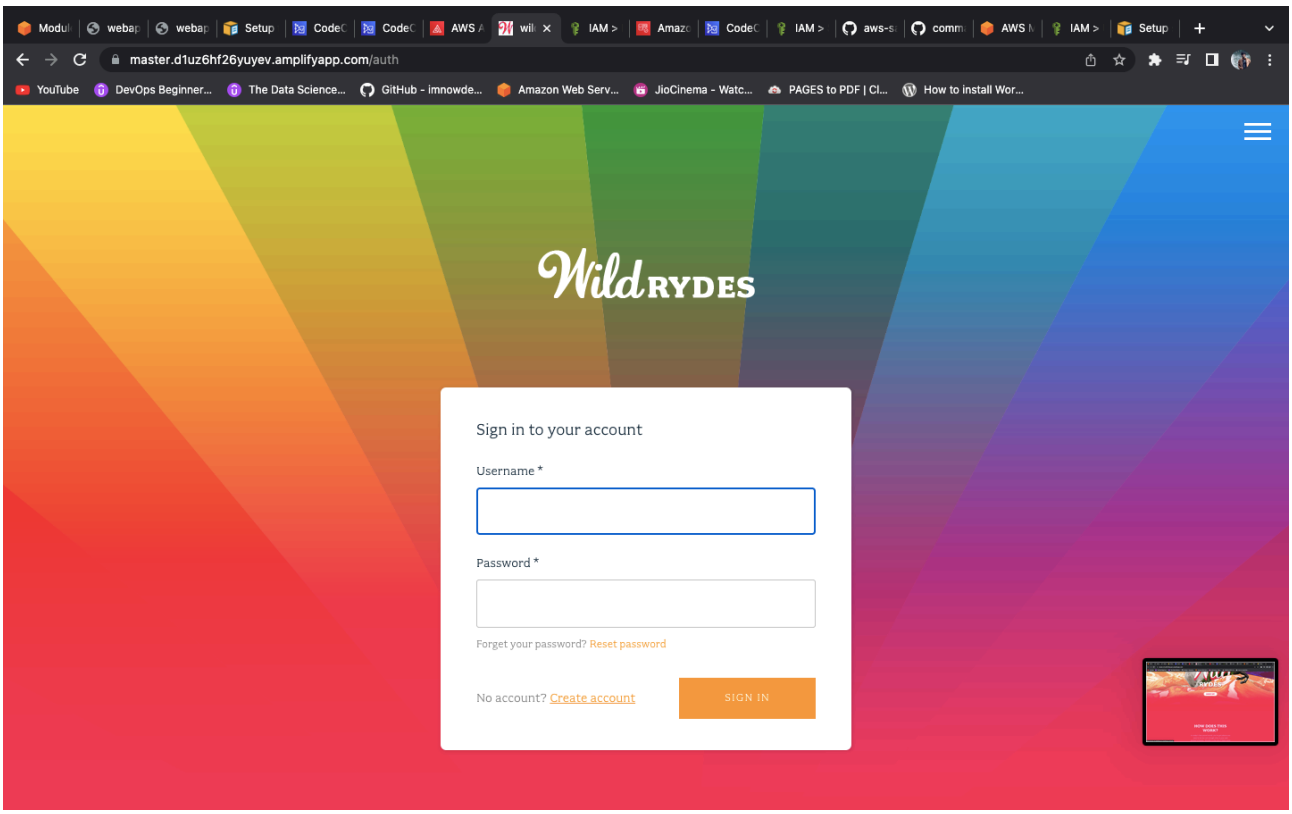
Amplify Console should pick up the changes and begin building and deploying your web application. Watch it to verify the completion of the deployment.

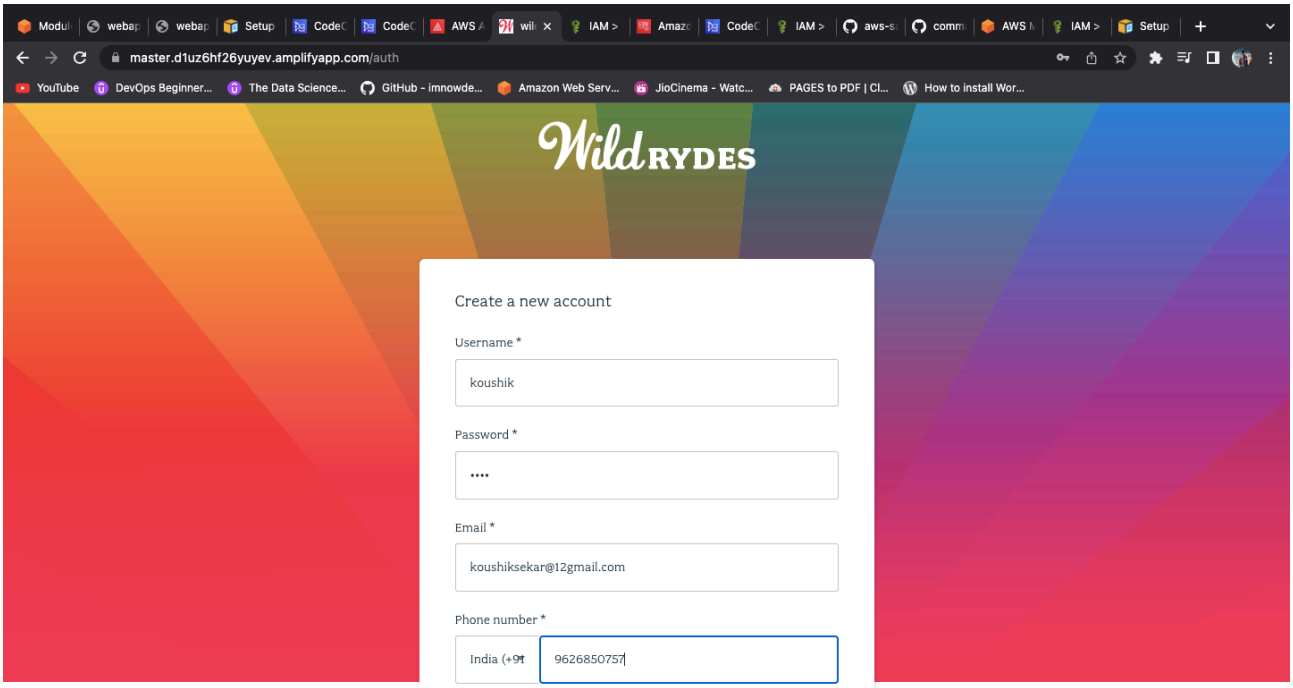
Implementation Validation

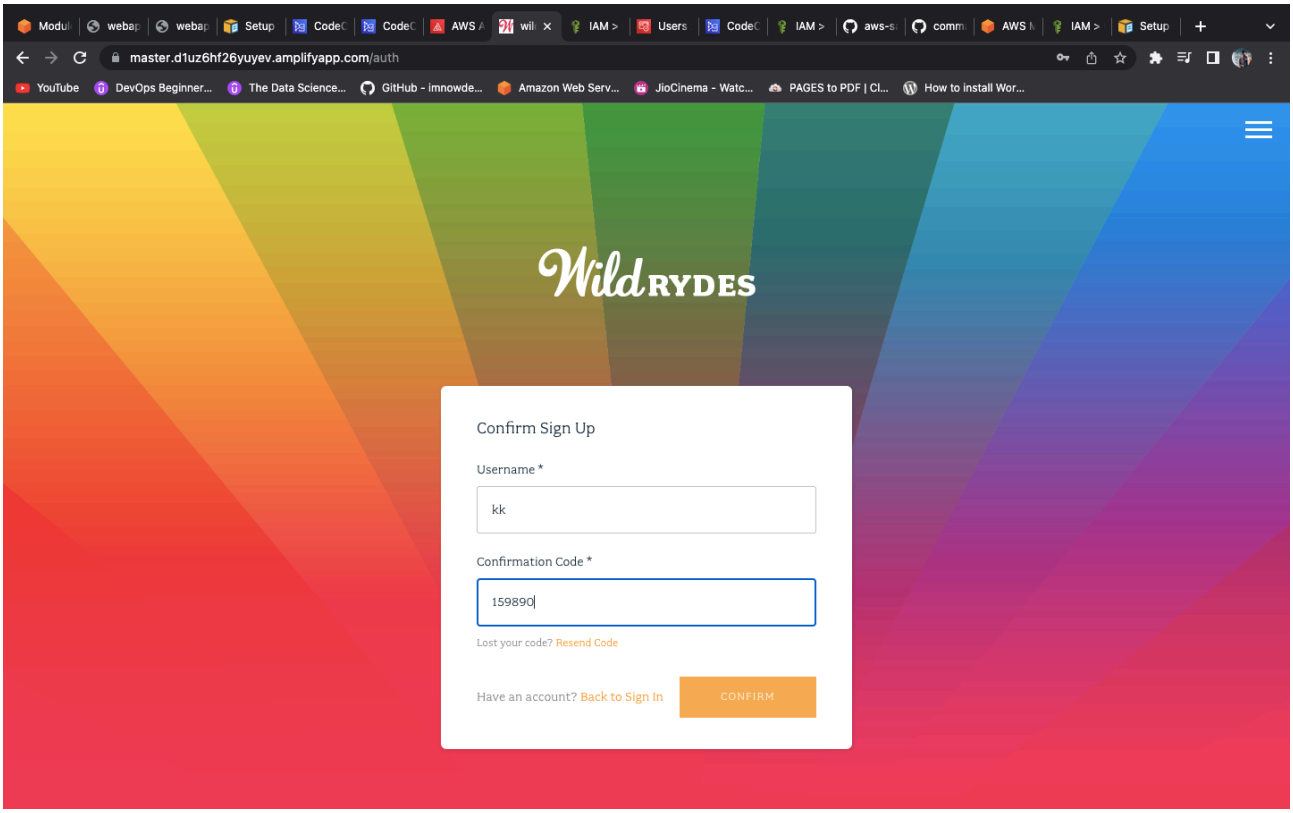
1. Visit `/ride` under your website domain.

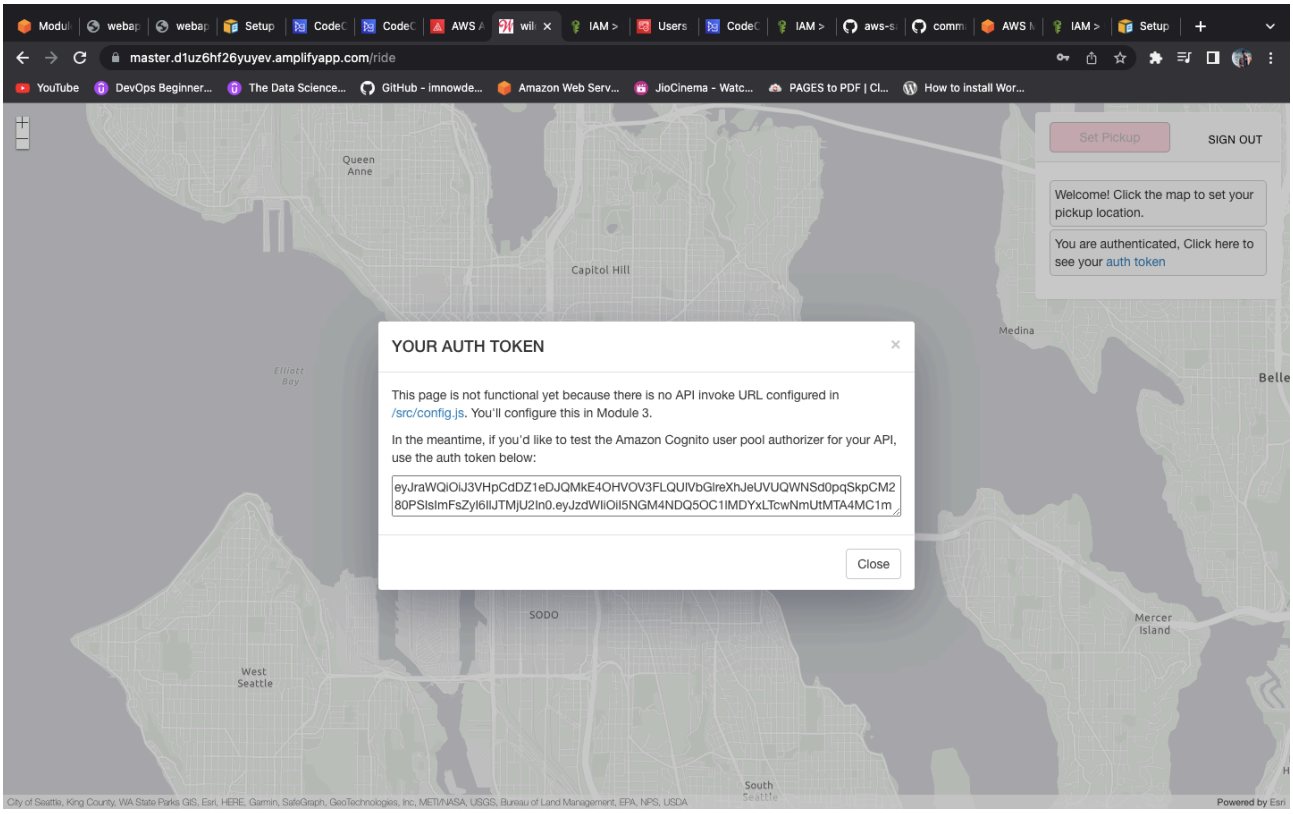
2. If you are redirected to the sign in page, sign in with the user you created in the previous module.
3. After the map has loaded, click anywhere on the map to set a pickup location.
4. Choose **Request Unicorn**. You should see a notification in the right sidebar that a unicorn is on its way and then see a unicorn icon fly to your pickup location.

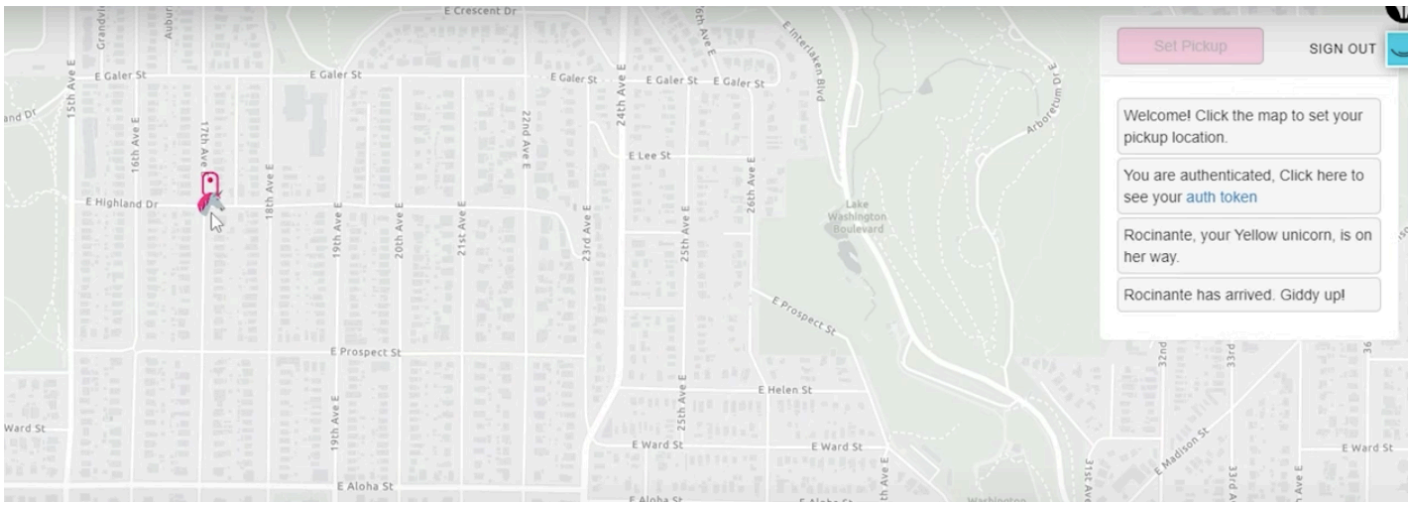



master.d1uz6hf26yuyev.amplifyapp.com/auth
WildRYDES
Sign in to your account
Username *
Password *
Forget your password? Reset password
No account? Create account
SIGN IN


master.d1uz6hf26yuyev.amplifyapp.com/auth
WildRYDES
Create a new account
Username *
koushik
Password *
Email *
koushiksekar@12gmail.com
Phone number *
India (+91
9626850757
Have an account? Sign in
CREATE ACCOUNT


master.d1uz6hf26yuyev.amplifyapp.com/auth
Wild Rydes
Confirm Sign Up
Username *
kk
Confirmation Code *
159890
Lost your code? Resend Code
Have an account? Back to Sign In
CONFIRM


master.d1uz6hf26yuyev.amplifyapp.com/ride
Set Pickup
SIGN OUT
Welcome! Click the map to set your pickup location.
You are authenticated, Click here to see your auth token
YOUR AUTH TOKEN
This page is not functional yet because there is no API invoke URL configured in /src/config.js. You'll configure this in Module 3.
In the meantime, if you'd like to test the Amazon Cognito user pool authorizer for your API, use the auth token below:
eyJraWQiOiJ3VHpCdDZ1eDJQMkE4OHVOV3FLQUlVbGlreXhJeUVUQWNSd0pqSkpCM280PSIsImFsZyI6IlJTMjU2In0.eyJzdWIiOiI5NGM4NDQ5OC1lMDYxLTcwNmUtMTA4MC1m
Close
Queen Anne
Capitol Hill
Medina
Elliott Bay
SODO
West Seattle
Mercer Island
South Seattle
City of Seattle, King County, WA State Parks GIS, Esri, HERE, Garmin, SafeGraph, GeoTechnologies, Inc, METI/NASA, USGS, Bureau of Land Management, EPA, NPS, USDA
Powered by Esri



Backend code:

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights
    Reserved.

// SPDX-License-Identifier: MIT-0


const { randomBytes } = require('crypto');

const { DynamoDBClient, PutItemCommand } = require('@aws-
    sdk/client-dynamodb');

const { marshall } = require('@aws-sdk/util-dynamodb');


const ddbClient = new DynamoDBClient({ region: 'us-east-1' });
    // Update with your desired region


const fleet = [
```

```
  {
    Name: 'Bucephalus',
    Color: 'Golden',
    Gender: 'Male',
  },
  {
    Name: 'Shadowfax',
    Color: 'White',
    Gender: 'Male',
  },
  {
    Name: 'Rocinante',
    Color: 'Yellow',
    Gender: 'Female',
  },
];

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

  const rideId = toUrlString(randomBytes(16));
```

```
console.log('Received event (', rideId, '): ', event);

const username =
    event.requestContext.authorizer.claims['cognito:usernam
    e'];

const requestBody = JSON.parse(event.body);

const pickupLocation = requestBody.PickupLocation;

const unicorn = findUnicorn(pickupLocation);

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

  return {
    statusCode: 201,
    body: JSON.stringify({
      RideId: rideId,
      Unicorn: unicorn,
      UnicornName: unicorn.Name,
      Eta: '30 seconds',
      Rider: username,
    }),
```

```
      headers: {
        'Access-Control-Allow-Origin': ‘*',
         "Access-Control-Allow-Headers" : "Content-Type",
           "Access-Control-Allow-Methods": "OPTIONS,POST,GET"

      },
    };
  } catch (err) {
    console.error(err);


    return errorResponse(err.message,
        event.requestContext.requestId);
  }
};


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


async function recordRide(rideId, username, unicorn) {
  const params = {
    TableName: 'Rides',
    Item: marshall({
```

```
      RideId: rideId,
      User: username,
      Unicorn: unicorn,
      UnicornName: unicorn.Name,
      RequestTime: new Date().toISOString(),
    }),
  };

  await ddbClient.send(new PutItemCommand(params));
}

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

function errorResponse(errorMessage, awsRequestId) {
  return {
    statusCode: 500,
    body: JSON.stringify({
```

```
      Error: errorMessage,
      Reference: awsRequestId,
    }),
    headers: {
      'Access-Control-Allow-Origin': '*',
    },
  };
}
```