BUILD A SERVERLESS WEB APPLICATION

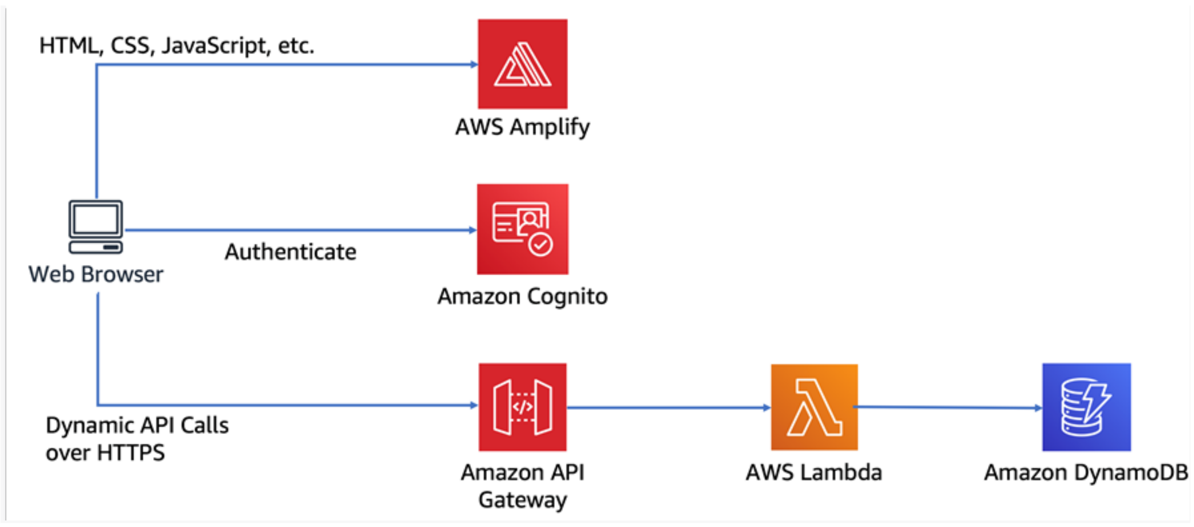
Introducing a serverless web application leveraging the power of AWS services, including Lambda, API Gateway, Amplify, DynamoDB, and Cognito. This project aims to provide a scalable and secure solution by utilizing Lambda functions for backend logic, API Gateway for RESTful endpoints, Amplify for frontend development, DynamoDB for data storage, and Cognito for user authentication and authorization.

Welcome to this project where we will learn how to build a serverless web application that revolutionizes the way users request unicorn rides from the Wild Rydes fleet. By combining the power of AWS services, including Lambda, API Gateway, Amplify, DynamoDB, and Cognito, we will create an intuitive HTML-based user interface that allows users to easily specify their pickup location, interact with a RESTful backend service to request rides, and even register and log in for a personalized experience. Get ready to embark on this exciting journey of building a modern, scalable, and secure unicorn ride service.

Application Architecture

The application architecture uses:

- AWS Lambda
- Amazon API Gateway
- Amazon DynamoDB
- Amazon Cognito
- AWS Amplify Console



Modules

- **Static Web Hosting**: AWS Amplify hosts static web resources including HTML, CSS, JavaScript, and image files which are loaded in the user's browser.
- **User Management**: Amazon Cognito provides user management and authentication functions to secure the backend API.
- **Serverless Backend**: Amazon DynamoDB provides a persistence layer where data can be stored by the API's Lambda function.
- **RESTful API**: JavaScript executed in the browser sends and receives data from a public backend API built using Lambda and API Gateway.

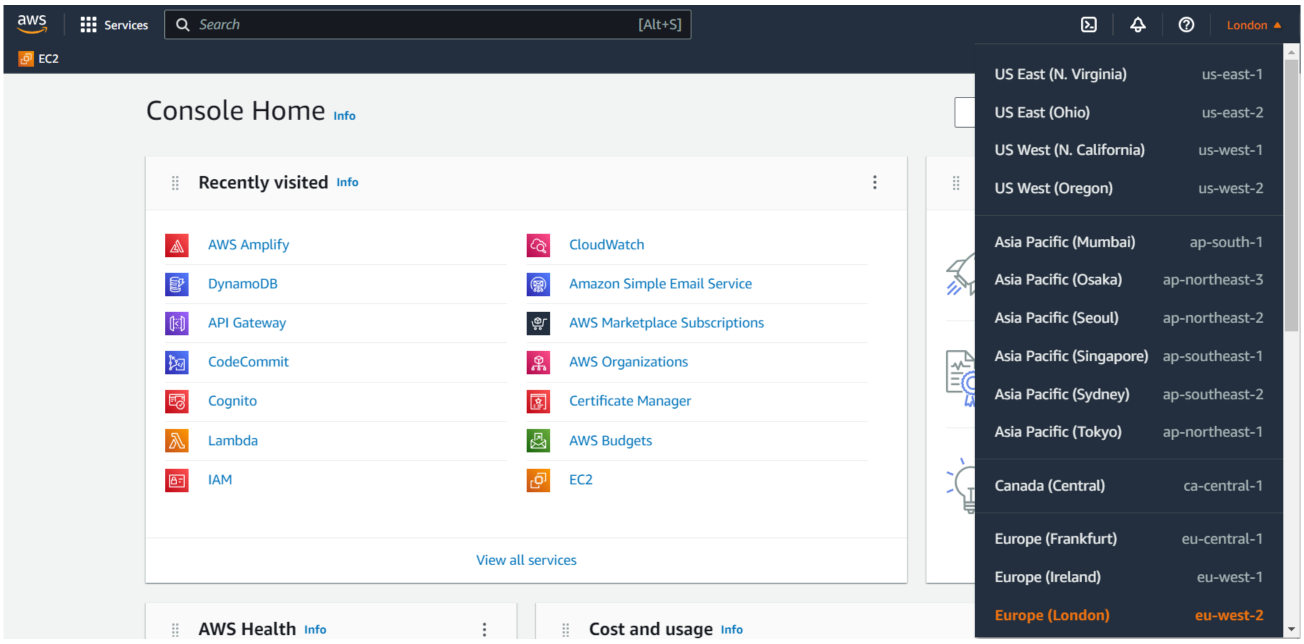
Module 1: Static Web Hosting

In this module, we'll leverage AWS Amplify to easily configure continuous deployment and hosting of our web application's static resources, while also preparing for future enhancements by incorporating JavaScript to interact with remote RESTful APIs powered by AWS Lambda and Amazon API Gateway.

Implementation

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 the Region from the dropdown in the upper right corner of the AWS Management Console.

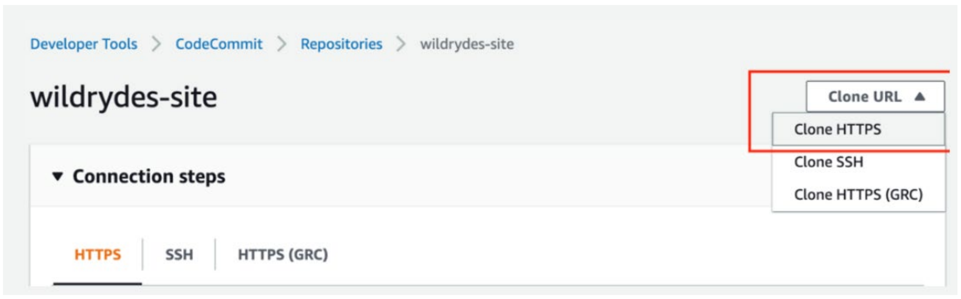

aws
Services
Search
[Alt+S]
London
EC2
Console Home Info
Recently visited Info
AWS Amplify
DynamoDB
API Gateway
CodeCommit
Cognito
Lambda
IAM
CloudWatch
Amazon Simple Email Service
AWS Marketplace Subscriptions
AWS Organizations
Certificate Manager
AWS Budgets
EC2
View all services
AWS Health Info
Cost and usage Info
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

Create Git Repository : We will use CodeCommit to store our application code, but you can do the same by creating a repository on GitHub

- Open the AWS CodeCommit console
- Select Create Repository
- Set the Repository name to "wildrydes-site"
- Select Create

Now that the repository is created, set up an IAM user with Git credentials in the IAM console.

Back in the CodeCommit console, from the Clone URL dropdown, select Clone HTTPS



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

```
git clone https://git-codecommit.eu-west2.amazonaws.com/v1/repos/wildrydes-site
Cloning into 'wildrydes-site'...
Username for 'https://git-codecommit.eu-west-2.amazonaws.com':XXXXXXXXXX
Password for 'USERID': XXXXXXXXXXXX
warning: You appear to have cloned an empty repository.
```

Populate Git Repository: Once we've used either AWS CodeCommit or GitHub.com to create your git repository and clone it locally, we'll need to copy the website content from an existing publicly accessible S3 bucket associated with this workshop and add the content to our repository.

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

```
aws s3 cp s3://wildrydes-us-east-1/WebApplication/1_StaticWebHosting/website ./ -- recursive
```

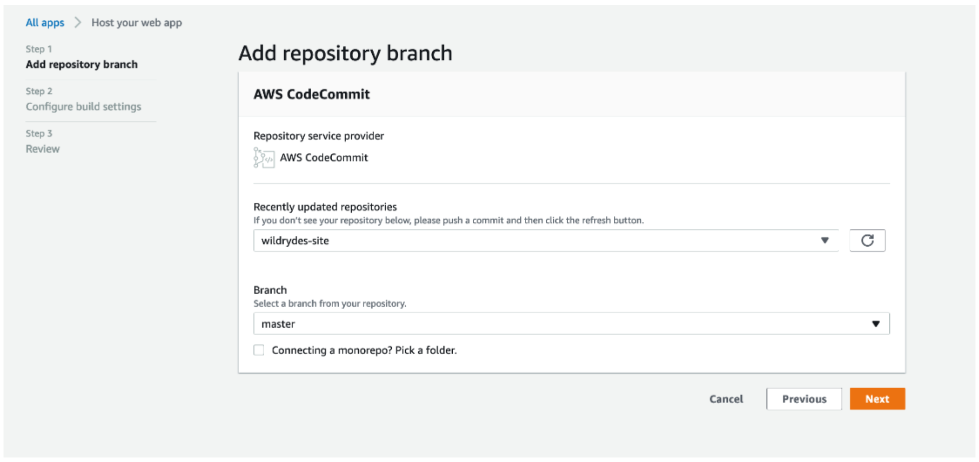
Commit the files to Git service

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

Enable Web Hosting with the AWS Amplify console: Next, we'll use the AWS Amplify Console to deploy the website you've just committed to git.

- Launch the Amplify Console 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 Continue.
- If you used GitHub, you'll need to authorize AWS Amplify to your GitHub account.
- From the dropdown, select the Repository and Branch you just created and select Next



All apps > Host your web app

Step 1
Add repository branch
Step 2
Configure build settings
Step 3
Review

Add repository branch

AWS CodeCommit

Repository service provider
AWS CodeCommit

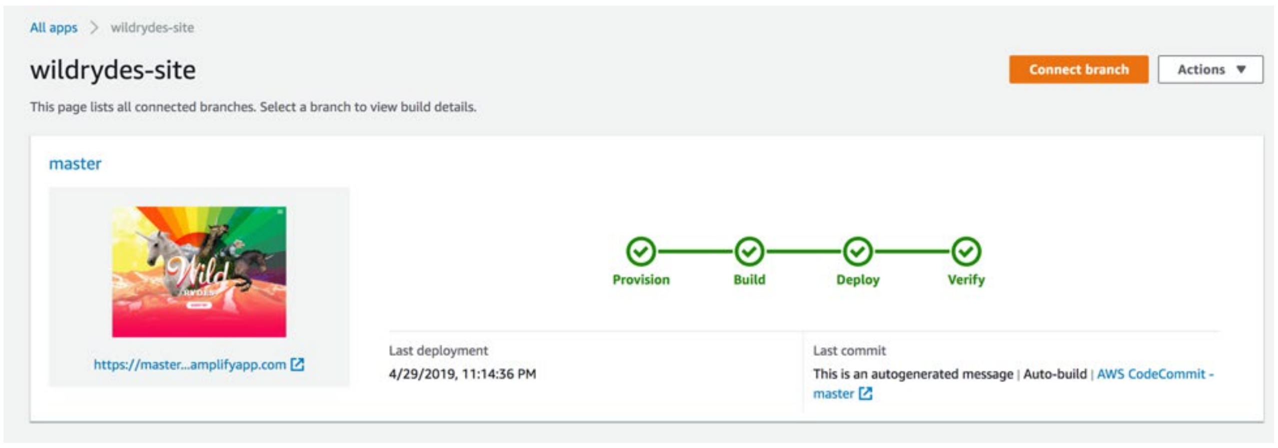
Recently updated repositories
If you don't see your repository below, please push a commit and then click the refresh button.
wildrydes-site

Branch
Select a branch from your repository.
master

☐ Connecting a monorepo? Pick a folder.

Cancel | Previous | Next

- On the Configure build settings page, leave all the defaults, Select Allow AWS Amplify to automatically deploy all files hosted in your project root directory and select Next.
- On the "Review" page select Save and deploy
- The process takes a couple of minutes for Amplify Console to create the necessary resources and to deploy the code.



All apps > wildrydes-site

wildrydes-site

Connect branch | Actions ▼

This page lists all connected branches. Select a branch to view build details.

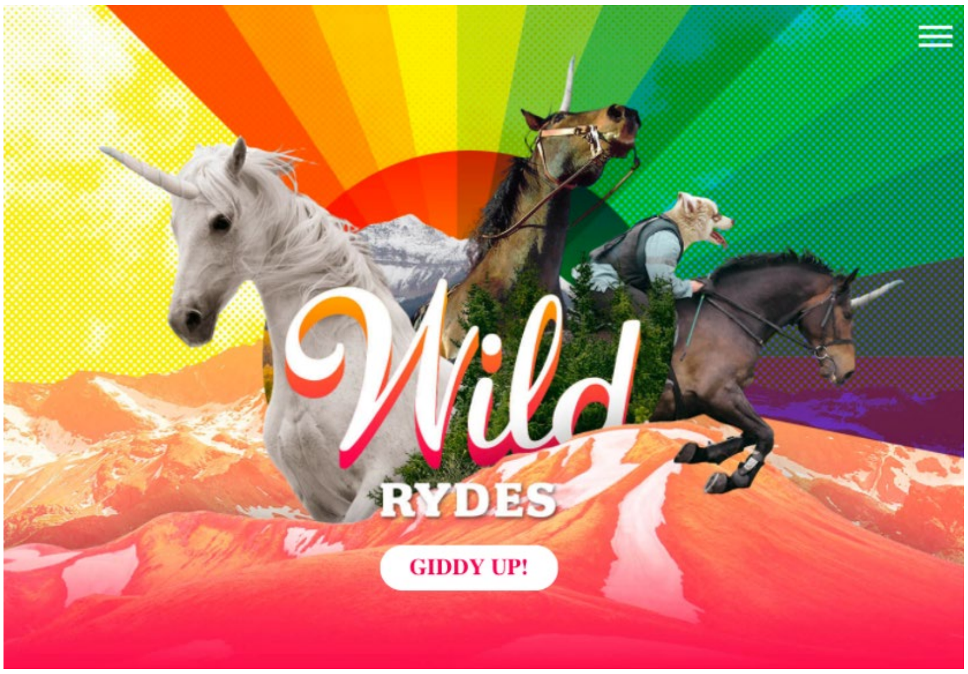
master

https://master...amplifyapp.com

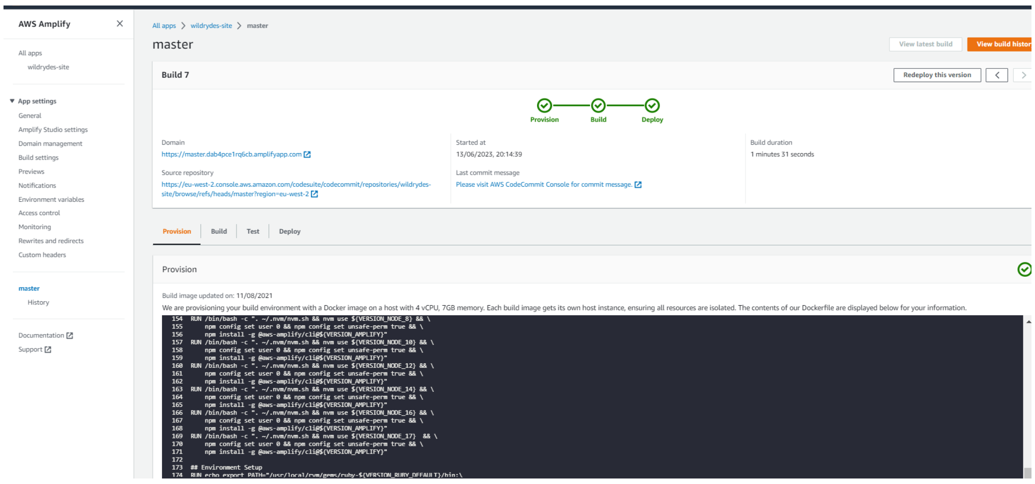
Last deployment
4/29/2019, 11:14:36 PM

Last commit
This is an autogenerated message | Auto-build | AWS CodeCommit - master

Once completed, click on the site image to launch your Wild Rydes site.



Clicking on the link for Master we'll see the build and deployment details related to our branch.



master

Build 7

Domain
https://master.dab4pce1rq6cb.amplifyapp.com

Started at
13/06/2023, 20:14:39

Build duration
1 minutes 31 seconds

Source repository
https://eu-west-2.console.aws.amazon.com/codesuite/codecommit/repositories/wildrydes-site/browse/refs/heads/master?region=eu-west-2

Last commit message
Please visit AWS CodeCommit Console for commit message.

Provision

Build image updated on: 11/08/2021

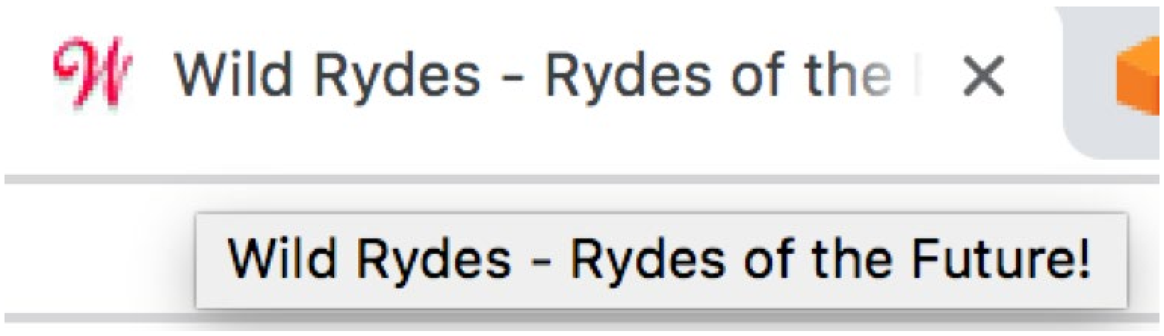
We are provisioning your build environment with a Docker image on a host with 4 vCPU, 7GB memory. Each build image gets its own host instance, ensuring all resources are isolated. The contents of our Dockerfile are displayed below for your information.

```
RUN /bin/bash -c ". ~/.nvm/nvm.sh && nvm use ${VERSION_NODE_8} && \
    npm config set user 0 && npm config set unsafe-perm true && \
    npm install -g @aws-amplify/cli@${VERSION_AMPLIFY}"
RUN /bin/bash -c ". ~/.nvm/nvm.sh && nvm use ${VERSION_NODE_10} && \
    npm config set user 0 && npm config set unsafe-perm true && \
    npm install -g @aws-amplify/cli@${VERSION_AMPLIFY}"
RUN /bin/bash -c ". ~/.nvm/nvm.sh && nvm use ${VERSION_NODE_12} && \
    npm config set user 0 && npm config set unsafe-perm true && \
    npm install -g @aws-amplify/cli@${VERSION_AMPLIFY}"
RUN /bin/bash -c ". ~/.nvm/nvm.sh && nvm use ${VERSION_NODE_14} && \
    npm config set user 0 && npm config set unsafe-perm true && \
    npm install -g @aws-amplify/cli@${VERSION_AMPLIFY}"
RUN /bin/bash -c ". ~/.nvm/nvm.sh && nvm use ${VERSION_NODE_16} && \
    npm config set user 0 && npm config set unsafe-perm true && \
    npm install -g @aws-amplify/cli@${VERSION_AMPLIFY}"
RUN /bin/bash -c ". ~/.nvm/nvm.sh && nvm use ${VERSION_NODE_17}  && \
    npm config set user 0 && npm config set unsafe-perm true && \
    npm install -g @aws-amplify/cli@${VERSION_AMPLIFY}"

## Environment Setup
RUN echo export PATH="/usr/local/rvm/gems/ruby-${VERSION_RUBY_DEFAULT}/bin:\
```

Modify the website title: To test the automatic rebuilding and redeployment process with AWS Amplify Console:

- Open wildryde-site/index.html in a text editor on your local machine.
- Modify the title line to <title>Wild Rydes - Rydes of the Future!</title>.
- Save the file and commit the changes to your Git repository.
- Amplify Console will detect the repository update and quickly begin the build process.
- Monitor the process on the Amplify Console page to see the changes reflected in the deployed site.


Wild Rydes - Rydes of the
Wild Rydes - Rydes of the Future!

Module 2: User Management

In this module we'll 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.

Implementation

Create Cognito User Pool and integrate an App to the User Pool: Amazon Cognito offers two authentication mechanisms: Cognito User Pools, which enable you to incorporate sign-up and sign-in features into your application, and Cognito Identity Pools, which allow authentication through social identity providers or your own identity system. In this module, we will utilize a user pool as the foundation for the registration and sign-in pages provided.

- In the AWS Console, search for and select Cognito, then choose "Create user pool."
- Configure the user pool's sign-in options, selecting "Username" as the sign-in method.
- Keep the default settings for other configurations such as authentication providers.
- Proceed to configure security requirements, optionally enabling multi-factor authentication (MFA).
- Keep the default settings for the sign-up experience configuration.
- Configure email delivery using Amazon SES as the email provider.
- Provide names for the user pool and initial app client, keeping other settings as default.
- Review the details and create the user pool.
- Take note of the Pool ID and App client ID on the Pool details page for future reference.

Update the website config.js file:

- Open the wildryde-site/js/config.js file on your local machine using a text editor.

- Update the cognito section in the file with the correct values for the user pool ID and app client ID obtained from the Amazon Cognito console.
- Save the changes and upload the updated config.js file push it to the Git repository to have it automatically deploy to Amplify Console.

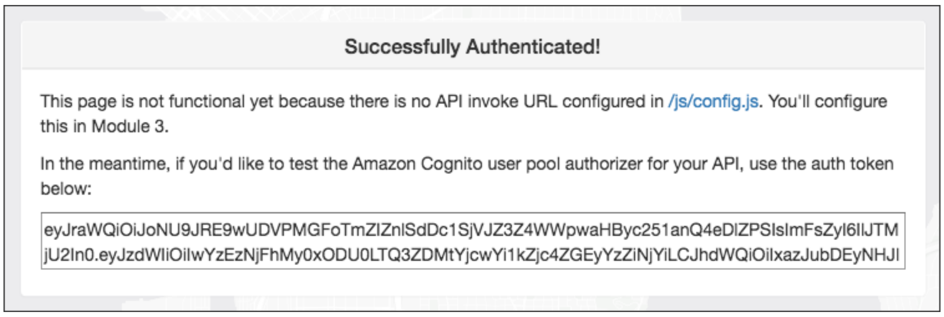
The updated config.js file

```
window._config = {
    cognito: {
        userPoolId: 'eu-west-2_HPCj6M54u',
        userPoolClientId: 'cqp6i9fqhk58soo0nu0culipg',
        region: 'eu-west-2'
    },
    api: {
        invokeUrl: '',
    }
};
```

Validate the Implementation:

- Access the /register.html page on your website or click on the "Giddy Up!" button on the homepage.
- Fill out the registration form with your desired information, including a valid or fake email address, and a password that meets the specified requirements.
- Submit the form, and you should receive an alert confirming the successful creation of your user.
- Confirm your new user either by following the verification process in the received email (if using a controlled email address) or manually through the Cognito console.
- In the Cognito console, select the WildRydes user pool, navigate to "Users and groups," locate the user associated with the registered email address, and confirm the user.
- Once the user is confirmed, visit the /signin.html page and log in using the registered email address and password.
- Upon successful login, you will be redirected to the /ride.html page.

We should see a notification that the API is not configured.



Successfully Authenticated!

This page is not functional yet because there is no API invoke URL configured in /js/config.js. You'll configure this in Module 3.

In the meantime, if you'd like to test the Amazon Cognito user pool authorizer for your API, use the auth token below:

```
eyJraWQiOiJoNU9JRE9wUDVPMGFoTmZlZnlSdDc1SjVJZ3Z4WWpwaHByc251anQ4eDlZPSIsImFsZyI6IlJTM
jU2In0.eyJzdWIiOiIwYzEzNjFhMy0xODU0LTQ3ZDMtYjcwYi1kZjc4ZGEyYzZiNjYiLCJhdWQiOiIxazJubDEyNHJI
```

Module 3: Serverless Service Backend

In this module, AWS Lambda and Amazon DynamoDB will be utilized to create a backend process for handling unicorn ride requests from the web application deployed in the previous module, allowing users to request and send unicorns to their desired locations.

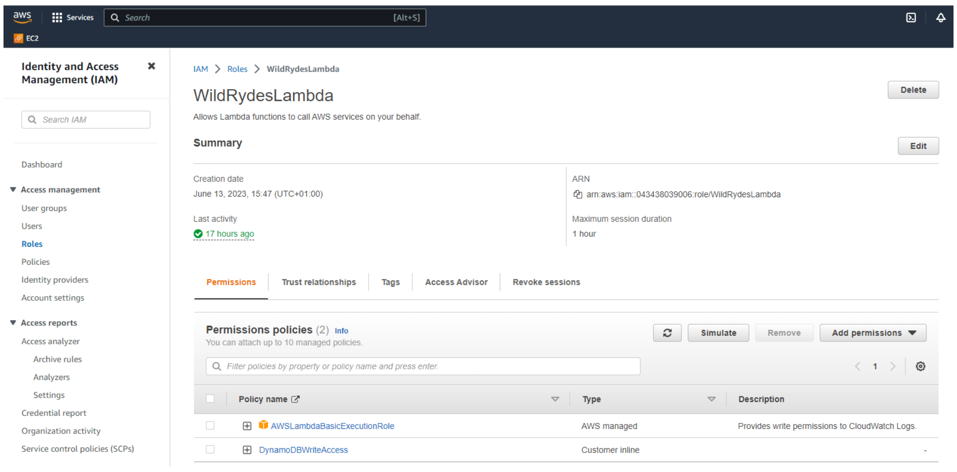
Implementation

Create an Amazon Dynamo DB table: Create a new DynamoDB table named "Rides" in the Amazon DynamoDB console with a partition key called "RideId" of type String and take note of the generated ARN for use in the subsequent step.

- Go to the AWS Management Console and select DynamoDB under Databases.
- Click on "Create table" and enter "Rides" as the Table name (case sensitive).
- Set "RideId" as the Partition key with the key type as String (case sensitive).
- Enable the "Use default settings" option and click on Create.
- Wait for the table creation to complete, then select the table name from the Tables page.
- In the Overview section, click on "Additional info" at the bottom, and note the ARN for future use.

Create an IAM Role for the Lambda function: To create the necessary IAM role for the Lambda function, go to the IAM console and select "Roles," then create a new role named "WildRydesLambda" with the role type set to AWS Lambda. Attach the managed policy "AWSLambdaBasicExecutionRole" to grant CloudWatch Logs permissions and create a custom inline policy allowing the "ddb: PutItem" action for your previously created DynamoDB table.

- Go to the AWS Management Console, select IAM in the "Security, Identity & Compliance" section, and choose "Roles" in the left navigation pane.
- Create a new role named "WildRydesLambda" with the role type set as AWS Lambda, attach the "AWSLambdaBasicExecutionRole" managed policy, and create a custom inline policy named "DynamoDBWriteAccess" allowing the "PutItem" action for the previously created DynamoDB table.



WildRydesLambda

Allows Lambda functions to call AWS services on your behalf.

Summary

Creation date
June 13, 2023, 15:47 (UTC+01:00)

Last activity
17 hours ago

ARN
arn:aws:iam::043438039006:role/WildRydesLambda

Maximum session duration
1 hour

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

Policy name	Type	Description
AWSLambdaBasicExecutionRole	AWS managed	Provides write permissions to CloudWatch Logs.
DynamoDBWriteAccess	Customer inline	-

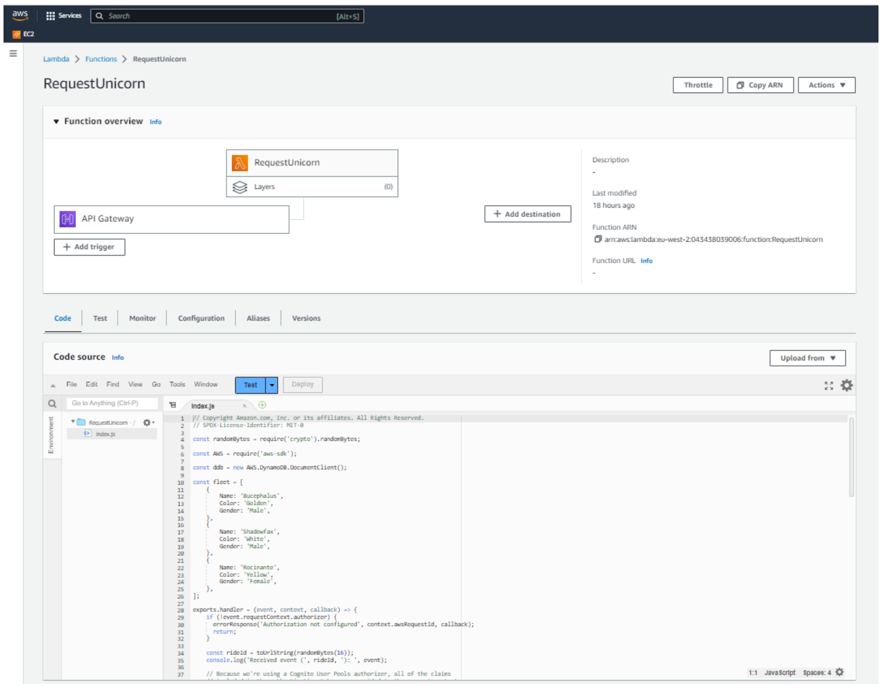
Create a Lambda function for handling request: To create the core function for processing API requests and dispatching unicorns, go to the AWS Lambda console and create a new Lambda function named "RequestUnicorn." Copy and paste the code from the provided example implementation (requestUnicorn.js) into the Lambda console's editor and ensure that you configure the function to use the previously created "WildRydesLambda" IAM role.

Access the AWS Management Console and select "Lambda" under the Compute section.

Click on "Create function," choose the "Author from scratch" option, and provide the name "RequestUnicorn" for the function.

Select the "Node.js 16.x" runtime and choose the existing role "WildRydesLambda" from the dropdown.

Scroll down to the "Code source" section, replace the existing code in the code editor with the contents of "requestUnicorn.js," and click on "Deploy."



Description
-

Last modified
18 hours ago

Function ARN
arn:aws:lambda:eu-west-2:043438039006:function:RequestUnicorn

Function URL
-

```
// 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
```

Validate the implementation: In this module, we will test the function that is created using the AWS Lambda console, and in the next module, we will integrate a REST API with API Gateway to invoke the function from our browser-based application.

- In the AWS Lambda console, go to the main edit screen of your function and click on "Test."
- Choose "Configure test event" from the dropdown and keep "Create new event" selected.
- Enter "TestRequestEvent" as the Event name and copy and paste the provided 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}}
}
```

- After saving the test event, click on "Test" with the "TestRequestEvent" selected in the dropdown.
- Scroll to the top of the page and expand the "Details" section of the "Execution result" to verify the successful execution of the function with the expected function result.

```
{
    "statusCode": 201,
    "body": "{\"RideId\":\"SvLnijIAtg6inAFUBRT+Fg==\",\"Unicorn\":{\"Name\":\"Rocinante\",\"Color\"
    "headers": {
        "Access-Control-Allow-Origin": "*"
    }
}
```

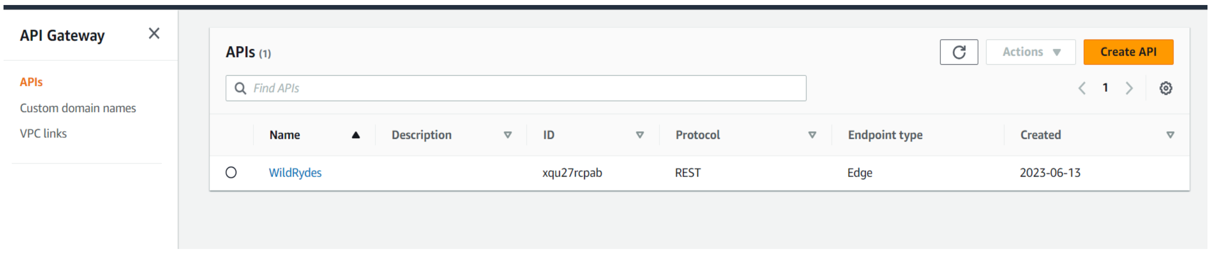
Module 4: Build a RESTful API

In this module, we will use Amazon API Gateway to expose the Lambda function you 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 you 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.

Implementation

Create a new REST API: In the AWS Management Console, follow these steps to create a new API in API Gateway:

- Click Services and select API Gateway under Application Services.
- Choose Create API and ensure that New API is selected in the Create new API section.
- Select Build under REST API and enter "WildRydes" as the API Name.
- Choose "Edge optimized" in the Endpoint Type dropdown (best for public services accessed from the Internet).
- Finally, click Create API.



API Gateway

APIs
Custom domain names
VPC links

APIs (1)

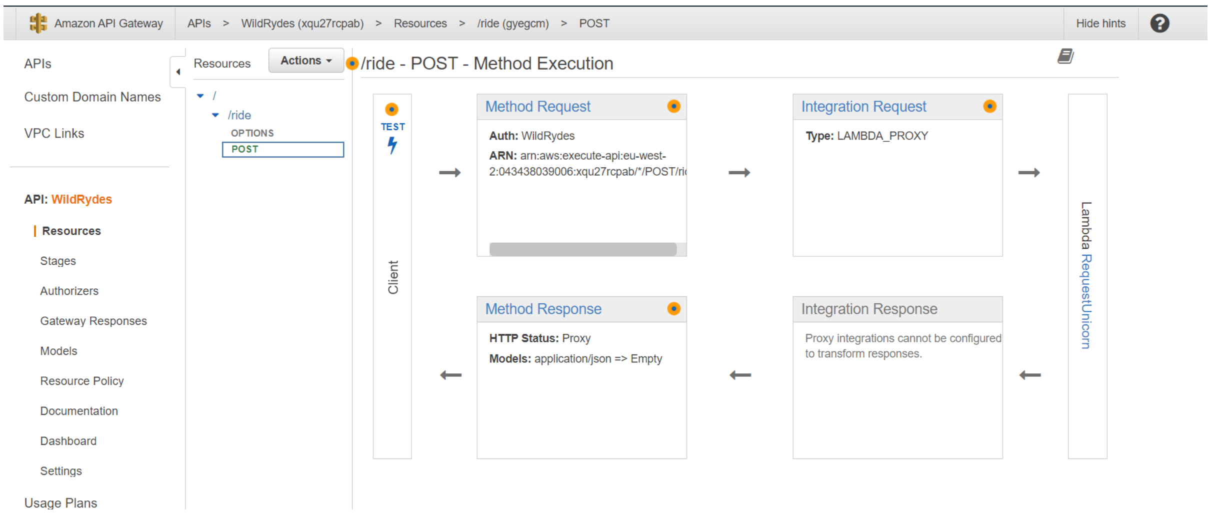
Find APIs

	Name	Description	ID	Protocol	Endpoint type	Created
○	WildRydes		xqu27rcpab	REST	Edge	2023-06-13

Create a new resource and method: Create /ride resource, add POST method, configure Lambda proxy integration with RequestUnicorn function.

Here are the steps to create the /ride resource, add a POST method, and configure it with a Lambda proxy integration using the RequestUnicorn function:

- Click on Resources in the left navigation menu of your WildRydes API.
- Select Create Resource from the Actions dropdown.
- Enter "ride" as the Resource Name and ensure the Resource Path is set to "ride".
- Enable API Gateway CORS for the resource.
- Click Create Resource.
- With the newly created /ride resource selected, choose Create Method from the Actions dropdown.
- Select POST from the dropdown and click the checkmark.
- Choose Lambda Function for the integration type.
- Enable the "Use Lambda Proxy integration" checkbox.
- Select the appropriate Lambda Region.
- Enter the name of the RequestUnicorn function in the Lambda Function field.
- Save the configuration. If you encounter an error, verify that the selected region matches the one used in the previous module.
- When prompted to grant Amazon API Gateway permission to invoke your function, select OK.
- Go to the Method Request card and click on the pencil icon next to Authorization.
- Select the WildRydes Cognito user pool authorizer from the dropdown list and click the checkmark icon.



Amazon API Gateway > APIs > WildRydes (xqu27rcpab) > Resources > /ride (gyegcm) > POST

/ride - POST - Method Execution

Method Request
Auth: WildRydes
ARN: arn:aws:execute-api:eu-west-2:043438039006:xqu27rcpab/*/POST/ri

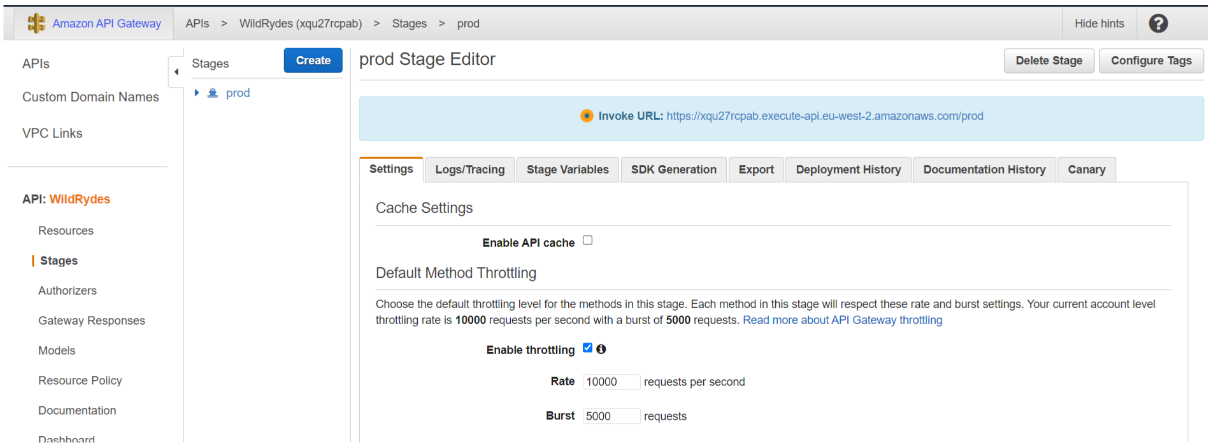
Integration Request
Type: LAMBDA_PROXY

Method Response
HTTP Status: Proxy
Models: application/json => Empty

Integration Response
Proxy integrations cannot be configured to transform responses.

Deploy the API: To deploy the API follow the below steps

- Click on the Actions dropdown and select Deploy API.
- Choose [New Stage] from the Deployment stage dropdown.
- Enter "prod" as the Stage Name.
- Click Deploy.
- Take note of the Invoke URL, as you will need it in the next section.



prod Stage Editor

Invoke URL: https://xqu27rcpab.execute-api.eu-west-2.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 ✓

Rate 10000 requests per second

Burst 5000 requests

Update the config.js file: To update the /js/config.js file:

- Open the config.js file in a text editor.
- Locate the "invokeUrl" setting under the "api" key.
- Update the value of "invokeUrl" with the Invoke URL obtained from the top of the stage editor page in the Amazon API Gateway console, corresponding to the deployment stage you created earlier.
- Ensure that the updated config file still contains the previous module's updates for your Cognito user pool.

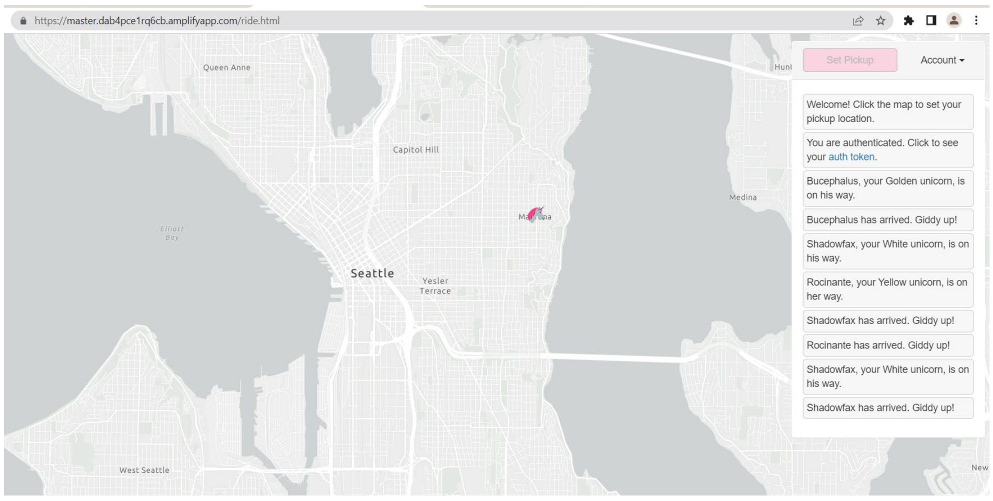
Here is the updated config.js file

```
window._config = {
    cognito: {
        userPoolId: 'eu-west-2_HPCj6M54u',
        userPoolClientId: 'cqp6i9fqhk58soo0nu0culipg',
        region: 'eu-west-2' // e.g. us-east-2
    },
    api: {
        invokeUrl: ' https://xqu27rcpab.execute-api.eu-west-2.amazonaws.com/prod',
    }
};
```

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

Validate the implementation: Clear your browser cache before proceeding, as there might be a delay in seeing the updated content of the conf ig.js file in your S3 bucket.

- Visit /ride.html under your website domain.
- If redirected to the ArcGIS sign-in page, sign in using the previously created user credentials.
- Once the map loads, click on the map to set a pickup location.
- Choose "Request Unicorn" and observe the right sidebar notification indicating a unicorn is on its way, followed by a flying unicorn icon moving towards your pickup location.



Welcome! Click the map to set your pickup location.

You are authenticated. Click to see your auth token.

Bucephalus, your Golden unicorn, is on his way.

Bucephalus has arrived. Giddy up!

Shadowfax, your White unicorn, is on his way.

Rocinante, your Yellow unicorn, is on her way.

Shadowfax has arrived. Giddy up!

Rocinante has arrived. Giddy up!

Shadowfax, your White unicorn, is on his way.

Shadowfax has arrived. Giddy up!

Congratulations! We have built a serverless web application using AWS.