Building a Serverless Web Application by Blake Gerold

Final Website Address: https://master.d2d248elxw0s3d.amplifyapp.com/

(This work is based on the "Build a Serverless Web Application" tutorial from the AWS website.)
https://aws.amazon.com/getting-started/hands-on/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/

Introduction: This tutorial requires a variety of AWS services that have not been used in class before. This includes Amazon API Gateway, AWS Amplify, Amazon Cognito and an ArcGIS account to add mapping to the app.

Section Zero: Adding all the new permission for IAM.

Create a new group and call "Project2''. Add the follow permissions to this group and then add it to the Cloud9Admisistrators policies:

AWSCodeCommitFullAccess

AmazonAPIGatewayAdministrator

AmazonAPIGatewayInvokeFullAccess

AmazonCognitoDeveloperAuthenticatedIdentities

AmazonCognitoPowerUser

AmazonESCognitoAccess

AWSLambda_FullAccess

AmazonDynamoDBFullAccess

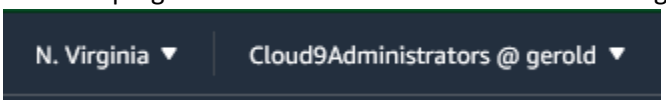
AmazonSESFullAccess

Section One: Host a Static Website

Part one: Create GitHub Repository

For this section, AWS Amplify with git to host a static website. Amplify will automatically update committed files if used with CodeCommit which makes rebuilding and redeploying websites easy.

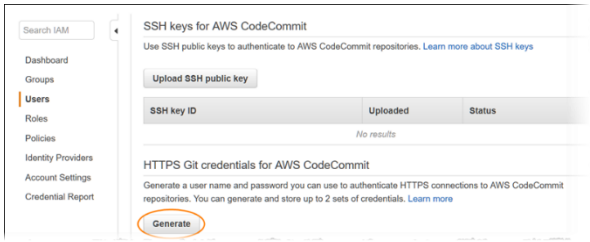
1. Select the region in the top right-hand corner. Make sure that it is N. Virginia!



2. Log into your Cloud9Admisistrators account.
3. Enter the AWS CodeCommit app either by clicking on the previous link or by searching for it on the AWS website.
4. Select **Create Repository** and name it "Project2" and hit **Create**.
5. Log back into your root account.

Part two: Cloning the Git Project and Entering Website Files

6. Navigate to the IAM app and select the **Cloud9Administrators** user.
7. Click on the **Security Credentials** tab and scroll down to the **HTTPS Git credentials for AWS CodeCommit** and select **Generate.**



8. Copy both the username and password that is generated. **IMPORTANT! THIS MUST BE DONE NOW! Once the password is generated, it is impossible to view again.**
9. Go to the pull-down bar labeled **Clone URL** and select **Clone HTTPS**
10. Go to your Cloud9 IDE and start a terminal. Enter the following:
    git clone YOUR CLONED URL HERE
    If this works, it will ask for a username and password. Entered the generated username and password generated from step 8.
11. The following code will obtain files from the S3 bucket used in this tutorial. Enter the following that is highlighted and hit enter after every paste:
    This opens your Project2 folder in the terminal.

    cd Project2

    This enters the static website file to the folder.

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

    Use the following Git commands to commit all of these files to the Project2 folder.

    git add .

    git commit -m 'new'

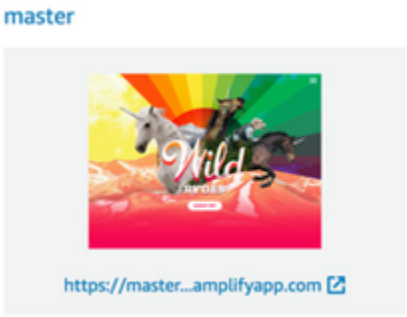
    git push

    The following show output in the terminal if it works:

    Counting objects: 95, done.
    Compressing objects: 100% (94/94), done.
    Writing objects: 100% (95/95), 9.44 MiB | 14.87 MiB/s, done.
    Total 95 (delta 2), reused 0 (delta 0)
    To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/Project2
    *** [new branch] master -> master**

Part 3: Web Hosting with Amplify Console

12. Either click on this AWS Amplify Console or search the AWS website for the app.
13. Select **Get Started**
14. Select **New App** and then **Host Web App**
15. Select **CodeCommit** under **Get started with Amplify Hosting**
16. Select **AWS CodeCommit** as the service provider and hit **Continue**
17. Select your "**Project2**" repository from the dropped down menu and hit **Next**
18. Under the Configure build settings tab, select **Allow AWS Amplify to automatically deploy all files hosted in your project root directory** and hit the **Next** button.
19. Hit **Save and deploy.**
20. Click on the picture to launch the website.



Part 4: Editing Files from CodeCommit

21. Go back to CodeCommit
22. Click **Project2**
23. Click **index.html**
24. Click **Edit**
25. Use CTRL + F "Wild Rydes" and change it to "Wild Rydes – Rydes of the Future!". Do this for the other "Wild Rydes" in this file as well.

    You can watch this Commit be updated in **Amplify** in real time!

Section Two: Manage Users

This section will allow for the user to set up a user pool with Amazon Cognito that can be accessed by the website. User name and email can either be entered manually with the Cognito app or register through the website. Cognito allows for email verification methods to make sure that bots do not spam user creation and along with SMS security options.

Part 1: Implementing a User Pool with Amazon Cognito

1. Search with the AWS search bar for the **Cognito** app.
2. Select **Create User Pool**
3. While in the **Configure sign-in experience** under the **Cognito user pool sign-in options,** select **User Name.** Then select **Next.**
4. While in **Configure security requirements** only change the **Multi-factor authentication** to **No MFA** for this project and select **Next.**
5. While in **Configure sign-up experience,** keep all defaults and select **Next.**
6. While in **Configure message delivery**, make sure **Send email with Amazon SES - Recommended** is selected. In addition, enter your email in the **FROM email address** field.
7. While in **Integrate your app**, enter "WildRydes" as the **User pool name** and enter "Project2WebApp" under the **App client name** and select **Next.**
8. While in **Review and create**, scroll to the bottom and select **Create user pool**.
9. This will take you to **User Pools**. Click on Info by the "**WildRydes**" user pool and record the **User pool ID** at the top (mine was **us-east-1_UB4UwXQMp**) and record the **Client ID** (mine was 6jqcsilfqq1hl1n21hm8tbqjbk) under the **App clients and analytics.**

Part 2: Update config.js

10. Either click on this AWS CodeCommit or search for **CodeCommit** in the AWS search bar.
11. Click on **Project2** under **Repositories,** click on the **js** folder, click on **config.js,** then click on **Edit.**
12. Enter the following code:

```
window._config = {
        cognito: {
        userPoolId: 'YOUR USER POOL ID',
        userPoolClientId: 'YOUR POOL CLIENT ID',
        region: 'us-east-1'
        },
        api: {
        invokeUrl: '' // e.g.
https://rc7nyt4tql.execute-api.us-west-2.amazonaws.com/prod',
```

```
    }
};
```

Mine looks like this:

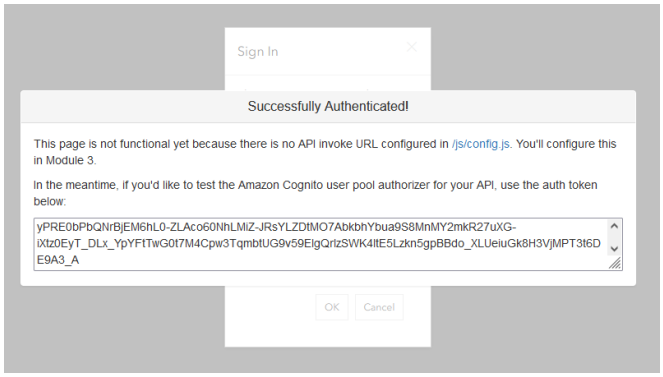
Project2 / js / config.js Info

```
window._config = {
    cognito: {
        userPoolId: 'us-east-1_UB4UwXQMp', // e.g. us-east-2_uXboG5pAb
        userPoolClientId: '6jqcsilfqq1hl1n21hm8tbqjbk', // e.g. 25ddkmj4v6hfsfvruhpfi7n4hv
        region: 'us-east-1' // e.g. us-east-2
    },
    api: {
        invokeUrl: '' // e.g. https://rc7nyt4tql.execute-api.us-west-2.amazonaws.com/prod',
    }
};
```

13. Enter your initials or name in the **Author Name** section and your email address in **Email address**. Click on **Commit changes**.

Part 3: Validate Emails

14. Click on AWS Amplify Console or search for the Amplify Console app in the AWS search bar.
15. Click on the webpage like you did **Section 1, Part 3, Step 20** after the Deploy is green.
16. Click on the **GIDDY UP!** on the front page and enter your Email, Password and Confirm Password then click on **LET'S RYDE.**
17. At the **VERIFY EMAIL** enter your Email and Verification Code sent to your email.
18. After the verification, the web page should look like this because do to no backend.



Section Three: Building a Serverless Backend

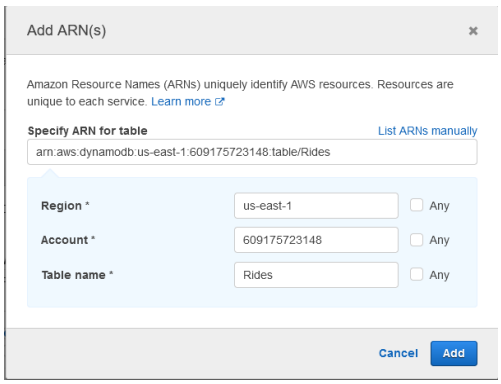
This section involves setting up AWS Lambda with Amazon DynamoDB. The Lambda function is triggered whenever a user requests a Unicorn using the Amazon API Gateway. The function will randomly fetch a unicorn from the DynamoDB table and return them to the frontend.

Part 1: Create a DynamoDB Table

1. Either click on **DynamoDB** or search for **DynamoDB** in the AWS search bar.
2. Click on **Create table** on the right hand side.
3. Under **Table name**, enter "Rides" and under **Partition key** enter "RideId" as a **String.**
4. Click on **Create table**
5. Click on your **Rides** table once it is created, click on **Additional info** and write down the ARN. Mine was arn:aws:dynamodb:us-east-1:609175723148:table/Rides

Part 2: Create a new IAM Role for Lambda

6. Enter your root account
7. Click on either IAM or type in the search bar **IAM.**
8. Click on **Roles** and then click on **Create Role.**
9. Under **Trusted entity type**, select **AWS service** and then for **Common use cases** select **Lambda.**
10. Under **Add permissions** search for **AWSLambdaBasicExecutionRole**, check the box and click on **Next.**
11. Enter under **Role Name** "WildRydesLambda" and click on **Create role**.
12. Click on **WildRydesLambda** on the **Roles** page.
13. Click on **Add permission** and select **Create Inline Policy.**
14. Click on **Service** and search for **DynamoDB.**
15. Click on **Actions** and search for **PutItem** and check the box.
16. Click on **Resources** and select **Add ARN.**
17. Copy and paste your **ARN** from before. When you enter the ARN, it should populate the table like this. Then click on **Add**.



18. Click on **Review policy.**
19. Name it "DynamoDBWriteAccess" and click on **Create policy.**

Part 3: Create Lambda Function for Handling Requests

20. Sign back into **Cloud9Administrators** role.
21. Click on **Lambda** or search AWS for **Lambda** and click on it.
22. Click on **Click Function.**
23. Make sure **Author from scratch** is selected. Enter "RequestUnicorn" in the **Function Name.**
24. Select **Node.js 16.x** under **Runtime.**
25. Under **Change default execution role**, go to **Execution rol**e, select **Use an existing role**, change this to **WildRydesLambda** and click on **Create Function.**
26. Once RequestUnicorn is created, scroll to Code source and copy/paste the following code from requestUnicorn.js.
27. Save the changes and click on **Deploy**.

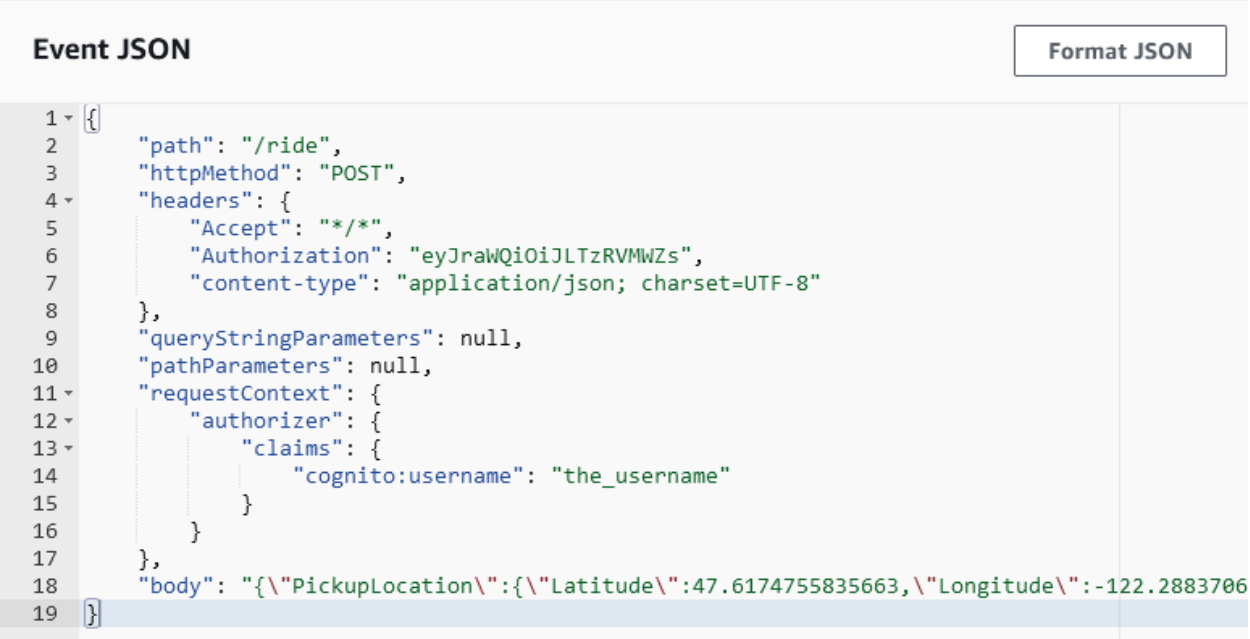
Part 4: Test your Code!

28. From your **RequestUnicorn Lambda Function,** click on **Test.**
29. Select **Create new event**, enter "TestRequestEvent**"** and enter the following code and save it:

```
{
  "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}}"
}
```

30. Click on **Test** and make sure **TestRequestEvent** is selected. If the test is successful it should come up with the following output. "statusCode": 201 is the desire. If it give you the error that the RideId cannot be found, it's because you put in **RideID** instead of **RideId**.

This is my code for the event file:



Section Four: Deploying a RESTful API

For this section, the user is going to use the API Gateway with ArcGIS which is an online geographic information system. ArcGIS will allow users to select a longitude and latitude location on a map anywhere in the world. Once selected, they can click Request Unicorn. The location will be sent through API Gateway and trigger the lambda function. This will in turn send a request to DynamoDB for a random unicorn which will be sent to the GPS location. .

Part 1: Creating a Rest API

1. Either click on API Gateway or search on the AWS website for **API Gateway**.
2. Scroll down to **REST API** (not the private one) and click on **Build**.
3. On the **Create** page, under **Choose the protocol,** make sure that **REST** is selected.
4. Under **Create new API,**select **New API**.
5. Under **Settings**, enter "**WildRydes**" for **API name** and select **Endpoint Type** to **Edge optimized**.
6. Click on **Create API.**
7. On the **API: WildRydes** page, select **Actions** then and select **Create Resource**.
8. Enter "ride" for **Resource Name**.
9. Check that the **Resource Path** is **ride**.
10. Check the box for **Enable API Gateway CORS**.
11. Click on **Create Resource**
12. On the new **/ride** resource, click on Actions and select **Create Method.**
13. Under **OPTIONS**, select **POST** and click on the checkmark.
14. Under **POST - Setup**, check that for the **Integration type**, **Lambda Function** is selected.

15. Check the box for **Use Lambda Proxy integration**
16. For the **Lambda Region,** make sure that **us-east-1** is selected.
17. For **Lambda Function**, enter "RequestUnicorn" and click on **Save.**
18. Press **OK** to give **Amazon API Gateway** permission.
19. Click on **Method Request**
20. Click on the pencil icon to the right of **Authorization** and select **WildRydes.**
21. Copy the **Invoke URL** at the top of the page. Mine was: https://t9lr46ys71.execute-api.us-east-1.amazonaws.com/prod

Part 2: Deploy the API

22. Go to **CodeCommit**
23. Click on **Project**
24. Click on the **js** folder
25. Click on **config.js**
26. In the "**api:** " part of the code update date it in the **Invoke URL.** Mine is shown below.

27. Enter your name under **Author name** and email address under **Email Address** and click on **Commit changes**.

Part 3: Test the REST API!!!

28. While still in **CodeCommit**, click on **Project2** then **ride.html**
29. Change all the **4.3** in the file to "4.6"
30. Enter your name under **Author name** and email address under **Email Address** and click on **Commit changes**.
31. Go to the website homepage and add ***/ride.html*** after the URL.
32. If you are redirected to ArcGIS sign-in page when testing the code, make sure to make a sign-in or make a free ArcGIS account.
33. Click the world map and hit **Request Unicorn.**

34. If this works, a unicorn will fly to that location! (Unfortunately this happens too quickly or a screen shot)