

Build a Serverless Web Application with AWS Lambda, Amazon API Gateway, AWS Amplify, Amazon DynamoDB, and Amazon Cognito

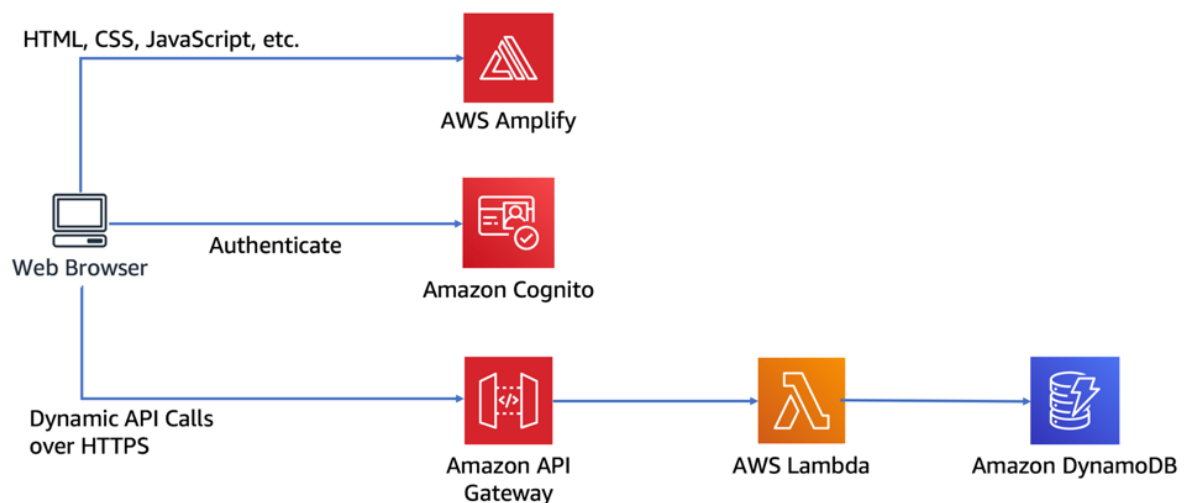
NOTE: Take care of **leading/trailing** spaces, while you copy and paste the values from this document.

Overview

In this tutorial, you'll create a 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.

Application Architecture

The application architecture uses [AWS Lambda](#), [Amazon API Gateway](#), [Amazon DynamoDB](#), [Amazon Cognito](#), and [AWS Amplify Console](#). Amplify Console provides continuous deployment and hosting of the static web resources including HTML, CSS, JavaScript, and image files which are loaded in the user's browser. JavaScript executed in the browser sends and receives data from a public backend API built using Lambda and API Gateway. Amazon Cognito provides user management and authentication functions to secure the backend API. Finally, DynamoDB provides a persistence layer where data can be stored by the API's Lambda function.



Modules

This workshop is divided into five modules. Each module describes a scenario of what we're going to build and step-by-step directions to help you implement the architecture and verify your work.

1. Host a Static Website (15 minutes): Configure AWS Amplify to host the static resources for your web application with continuous deployment built-in
2. Manage Users (30 minutes): Create an Amazon Cognito user pool to manage your users' accounts
3. Build a Serverless Backend (30 minutes): Build a backend process for handling requests for your web application
4. Deploy a RESTful API (15 minutes): Use Amazon API Gateway to expose the Lambda function you built in the previous module as a RESTful API
5. Terminate Resources (10 minutes): Terminate all the resources you created throughout this tutorial

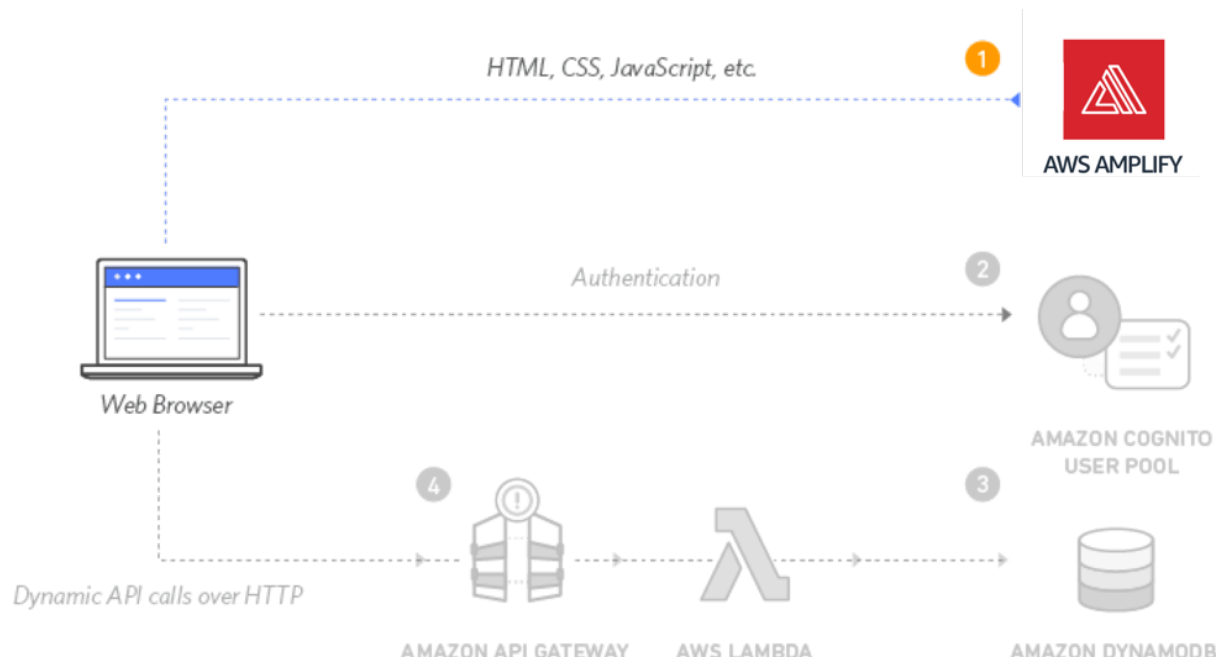
Module 1: Static Web Hosting with Continuous Deployment

To get started, you will configure AWS Amplify to host the static resources for your web application with continuous deployment built-in

Overview

In this module you'll configure AWS Amplify to host the static resources for your web application with continuous deployment built-in. The Amplify Console provides a git-based workflow for continuous deployment & hosting of fullstack web apps. In subsequent modules you'll add dynamic functionality to these pages using JavaScript to call remote RESTful APIs built with AWS Lambda and Amazon API Gateway.

Architecture Overview



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

Create a Git Repository

You have two options to manage the source code for this module: [AWS CodeCommit](#) (included in the AWS Free Tier) or [GitHub](#). In this tutorial we will use CodeCommit to store our application code, but you can do the same by [creating a repository](#) on GitHub.

- a. Open the [AWS CodeCommit console](#)
- b. Select Create Repository
- c. Set the Repository name* to "wildrydes-site"
- d. Select Create
- e. Now that the repository is created, use your existing IAM user's Git credentials in the IAM console (See these steps to create Git credentials > [follow these instructions](#)).
- f. Back in the CodeCommit console, From the Clone URL drop down, select Clone HTTPS

g. From a terminal window (**better use your Public EC2 instance**) run git clone and the HTTPS URL of the repository:

NOTE: to install git on your EC2 run this “`sudo yum install -y git.x86_64`”

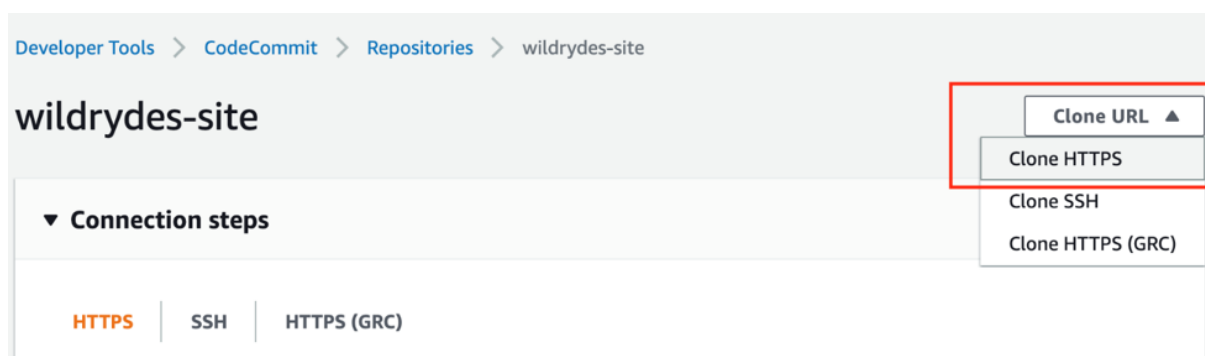
```
$ git clone https://git-codecommit.us-east-1.amazonaws.com/v1/repos/wildrydes-site
```

```
Cloning into 'wildrydes-site'...
```

```
Username for 'https://git-codecommit.us-east-1.amazonaws.com':XXXXXXXXXXXX
```

```
Password for 'USERID': XXXXXXXXXXXXXXXX
```

```
warning: You appear to have cloned an empty repository.
```



Populate the Git Repository

Once you've used either AWS CodeCommit or GitHub.com to create your git repository and clone it locally, you'll need to copy the web site content from an existing publicly accessible S3 bucket associated with this workshop and add the content to your repository.

1. Navigate to IAM Users > CloudLearner.
2. Navigate to the **Security Credentials** tab and click on the **Create access key** button.
3. Download and save the generated file on your laptop. This file contains secret access keys for your CloudLearner user.

4. Make sure you have removed the IAM role from your Public EC2 instance.
5. On the Public EC2 instance, **configure your credentials** for AWS CLI. Execute:

```
aws configure
```

#Provide the values for Access Key ID & Secret Access Key from the downloaded file.

Specify **us-east-1** for region.

Specify **json** for output format.

a. Change directory into your repository and copy the static files from S3:

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

b. Commit the files to your Git service

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

Additional Steps:

```
git config --global user.email "you@example.com"
```

```
git config --global user.name "Your Name"
```

```
$ git push
```

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/wildrydes-site

** [new branch] master -> master*

Enable Web Hosting

Next you'll use the [AWS Amplify Console](#) to deploy the website you've just committed 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.

You need to create a new service role with the permissions to deploy the application backend.

1. Under **IAM Dashboard**, click on Create new role, check that Amplify is selected and click Next permissions. Select the relevant permissions boundary (search for **DND_permission_boundary**). Now, 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 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.

- a. Launch the Amplify Console page. [Delete any apps \(if already available there\)](#).
- b. Scroll to the bottom of the page. Click Get Started button under **Host your web app** section. Then, select AWS CodeCommit radio button and press Continue button.
- c. From the dropdown select the *Repository and Branch* just created. Press Next.
- d. On the "Configure build settings" page, check the checkbox for "Allow AWS Amplify to automatically deploy all files hosted in your project root directory".
- e. For the Read-only access service role, you should choose second radio button. It will have the role name (previously created) auto-populated. Leave all the other fields as defaults and click Next.
- f. On the "Review" page select Save and deploy
- h. The process takes a couple of minutes for Amplify Console to create the necessary resources and to deploy your code.

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


All apps > wildrydes-site

wildrydes-site

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

Connect branch Actions ▼

master



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

Provision Build Deploy Verify

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

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

Keep this browser tab open throughout the exercise to check the progress around Amplify deployments.

Once completed, click the site image to launch your Wild Rydes site. (**You might have to copy the URL and open it in your phone**).



Modify your 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.

a. Open the `index.html` page and modify the title line so that it says: `<title>Wild Rydes - Rydes of the Future!</title>`

b. Save the file and commit to your git repository again. Amplify Console will begin to build the site again soon after it notices the update to the repository. It will happen pretty quickly! Head back to the [Amplify Console page](#) to watch the process.

```
$ git add index.html
```

```
$ git commit -m "updated title"
```

```
[master dfec2e5] updated title
```

```
1 file changed, 1 insertion(+), 1 deletion(-)
```

```
$ git push
```

```
Counting objects: 3, done.
```

```
Compressing objects: 100% (3/3), done.
```

```
Writing objects: 100% (3/3), 315 bytes | 315.00 KiB/s, done.
```

```
Total 3 (delta 2), reused 0 (delta 0)
```

```
remote: processing
```

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

```
2e9f540..dfec2e5 master -> master
```

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



Wild Rydes - Rydes of the Future!

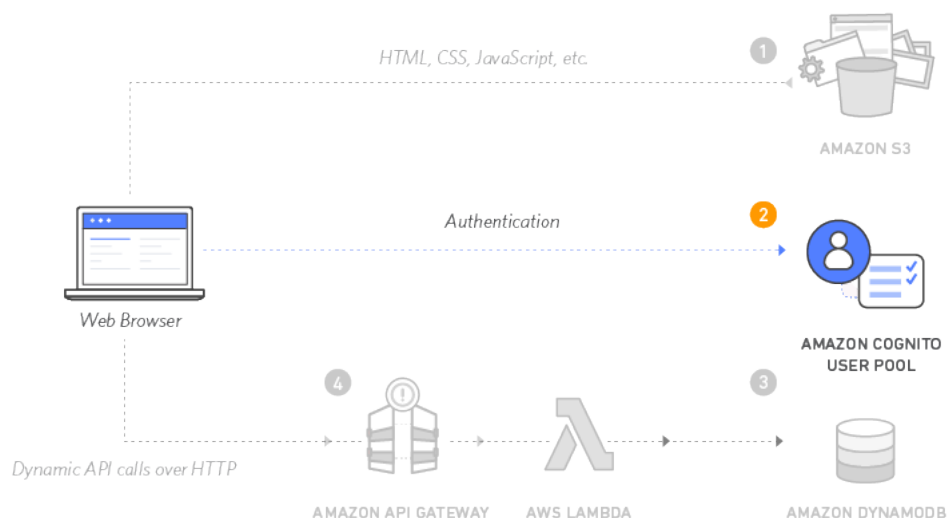
Module 2: User Management

You will create an Amazon Cognito user pool to manage your users' accounts

Overview

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

Architecture Overview



When users visit your website they will first register a new user account. For the purposes of this workshop we'll only require them to provide an email address and password to register. However, you can configure Amazon Cognito to require additional attributes in your own applications.

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 your site and enter their email address and the verification code they received. You can also confirm user accounts using the Amazon Cognito console if you want to use fake email addresses 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 you build with Amazon API Gateway.

Create an Amazon Cognito Pool

Amazon Cognito provides two different mechanisms for authenticating users. You can use Cognito User Pools to add sign-up and sign-in functionality to your 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 your own identity system. For this module you'll use a user pool as the backend for the provided registration and sign-in pages.

- a. From the AWS Console click Services then select Cognito under Mobile Services
- b. Choose Manage your User Pools
- c. Choose Create a User Pool
- d. Provide a name for your user pool such as **WildRydes**, then select **Review Defaults**
- e. On the review page, click **Create pool**
- f. Note the **Pool Id** on the Pool details page of your newly created user pool. Keep this in a notepad.

Add an app to your user pool

From the Amazon Cognito console select your user pool and then select the App clients section. Add a new app client and make sure the Generate client secret option is deselected. Client secrets aren't currently supported with the JavaScript SDK. If you do create an app with a generated secret, delete it and create a new one with the correct configuration.

- a. From the Pool Details page for your user pool, select **App clients** from the left **General Settings** section in the navigation bar.
- b. Choose Add an app client.
- c. Give the app client a name such as **WildRydesWebApp**.
- d. Uncheck the Generate client secret option. Client secrets aren't currently supported for use with browser-based applications.
- e. Choose Create app client button at the bottom of page.
- f. Note the App client id for the newly created application.

Update the Website Config

The `/js/config.js` file contains settings for the user pool ID, app client ID and Region. Update this file with the settings from the user pool and app you created in the previous steps and upload the file back to your bucket

- a. From your local machine, open `wild-ryde-site/js/config.js` in a text editor of your choice.
- b. Update the cognito section with the correct values for the user pool and app you just created.

You can find the value for `userPoolId` on the Pool details page of the Amazon Cognito console after you select the user pool that you created.

You can find the value for `userPoolClientId` by selecting App clients from the left navigation bar. Use the value from the App client id field for the app you created in the previous section. The value for region should be the AWS Region code where you created your user pool. E.g. **us-east-1** for the **N. Virginia** Region, or **us-west-2** for the Oregon Region. If you're not sure which code to use, you can look at the Pool ARN value on the Pool details page. The Region code is the part of the ARN immediately after `arn:aws:cognito-idp:`.

The updated `config.js` file should look like this. **Note that the actual values for your file will be different:**

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

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

```
$ git add .
```

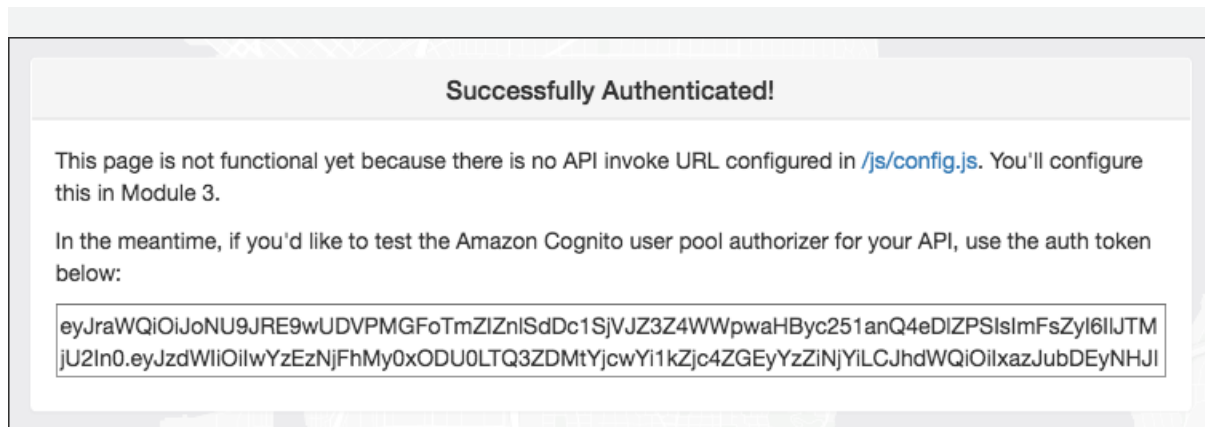
```
$ git commit -m "update2"
```

```
$ git push
```

Validate your implementation

- a. **Wait for the Amplify deployment to complete.** Visit */register.html* under your website domain, or choose the Giddy Up! button on the homepage of your site.
- b. Complete the registration form and choose Let's Ryde. 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.
- c. Confirm your new user using one of the two following methods.
- d. If you used an email address you control, you can complete the account verification process by visiting */verify.html* under your website domain and 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.
- e. If you used a dummy email address, you must confirm the user manually through the Cognito console.
- f. From the AWS console, click Services then select Cognito under Security, Identity & Compliance.
- g. Choose Manage your User Pools
- h. Select the *WildRydes* user pool and click Users and groups in the left navigation bar.
- i. 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.
- j. Choose Confirm user to finalize the account creation process.
- k. 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 you entered during the registration step.

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



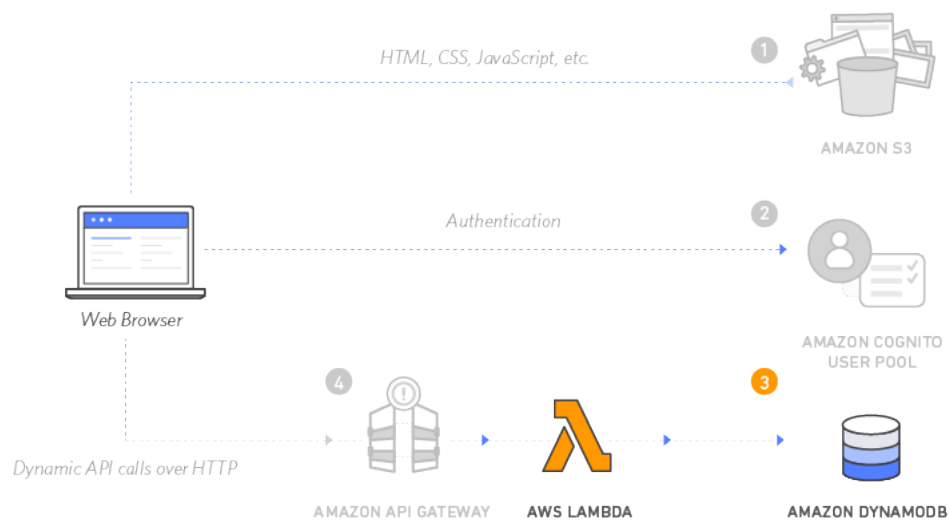
Module 3: Serverless Service Backend

You will use AWS Lambda and Amazon DynamoDB to build a backend process for handling requests for your web application

Overview

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

Architecture Overview



You'll implement 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 front-end application with details about the unicorn being dispatched.

The function is invoked from the browser using Amazon API Gateway. You'll implement that connection in the next module. For this module you'll just test your function in isolation.

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, note the ARN for use in the next step.

- 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.
- Once the table is created, click on the table. Go to Overview section of your new table and note the **ARN** (under Additional Info). You will use this in the next section.

Create an IAM Role for your 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. For the purposes of this workshop, you'll need to 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.

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 *ddb:PutItem* action for the table you created in the previous section.

- a. From the AWS Management Console, click on Services and then select IAM in the Security, Identity & Compliance section.
- b. Select Roles in the left navigation bar and then choose Create New Role.
- c. Select Lambda for the role type from the AWS service group, then click Next: Permissions.
Note: Selecting a role type automatically creates a trust policy for your role that allows AWS services to assume this role on your behalf. If you were creating this role using the CLI, AWS CloudFormation or another mechanism, you would specify a trust policy directly.
- d. Begin typing ***AWSLambdaBasicExecutionRole*** in the Filter text box and check the box next to that role. Also, select the relevant **Permissions Boundary** (search for **DND_permission_boundary**).
- e. Choose Next Step.
- f. Enter *WildRydesLambda* for the Role Name.
- g. Choose Create Role.
- h. Type *WildRydesLambda* into the filter box on the Roles page and choose the role you just created.
- i. On the **Permissions** tab, choose the Add **inline policy** link in the lower right corner to create a new inline policy.
- j. Select Choose a service.
- k. Begin typing *DynamoDB* into the search box labeled Find a service and select DynamoDB when it appears..
- l. Choose Select actions.
- m. Begin typing *PutItem* into the search box labeled Filter actions and check the box next to PutItem when it appears.
- n. Select the Resources section.
- o. With the Specific option selected, choose the Add ARN link in the table section.
- p. Paste the ARN of the table you created in the previous section in the Specify ARN for table field, and choose Add.
- q. Choose Review Policy.
- r. Enter ***DynamoDBWriteAccess*** for the policy name and choose Create policy.

Create a Lambda Function for handling requests

AWS Lambda will run your code in response to events such as an HTTP request. In this step you'll build the core function that will process API requests from the web application to dispatch a unicorn. In the next module you'll use Amazon API Gateway to create a RESTful API that will expose an HTTP endpoint that can be invoked from your users' browsers. You'll then connect the Lambda function you create in this step to that API in order to create a fully functional backend for your web application.

Use the AWS Lambda console to create a new Lambda function called *RequestUnicorn* that will process the API requests. Use the provided [requestUnicorn.js](#) example implementation for your function code. Just copy and paste from that file into the AWS Lambda console's editor.

Make sure to configure your function to use the *WildRydesLambda* IAM role you created in the previous section.

- a. Choose Services then select Lambda in the Compute section.
- b. Click Create function.
- c. Keep the default Author from scratch card selected.
- d. Enter *RequestUnicorn* in the Name field.
- e. Select **Node.js 12.x** for the Runtime.
- f. Expand Change default execution role under Permissions. Ensure Use an existing role is selected from the Role dropdown.
- g. Select *WildRydesLambda* from the Existing Role dropdown.

Create function [Info](#)

Choose one of the following options to create your function.

Author from scratch ☒

Start with a simple Hello World example.



Use a blueprint ☐

Build a Lambda application from sample code and configuration presets for common use cases.



Browse serverless app repository ☐

Deploy a sample Lambda application from the AWS Serverless Application Repository.



Basic information

Function name

Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)

Choose the language to use to write your function.

Permissions [Info](#)

By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ Change default execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

- ☐ Create a new role with basic Lambda permissions
- ☒ Use an existing role
- ☐ Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.



[View the WildRydesLambda role](#) on the IAM console.

► Advanced settings

Cancel

Create function

h. Click on Create function.

i. Scroll down to the Function **code source** section and replace the existing code in the index.js code editor with the contents of [requestUnicorn.js](#).

j. Click "**Deploy**" in the upper part of the editor and look for the "**Changes Deployed**" message.

Validate your Implementation

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.


- a. From the main edit screen for your function, look for the Test dropdown. Select Configure test event from the Select a test event... dropdown.
- b. Keep Create new test event selected.
- c. Enter *TestRequestEvent* in the Event name field
- d. 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}}"
```

- e. Click Create.
- f. On the main function edit screen click the **Test** button.
- g. View the Execution result section.
- h. Verify that the execution succeeded and that the function result looks like the following:

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

Recap

 **AWS Lambda** is a serverless Functions-as-a-Service (FaaS) product that removes the burden of managing servers to run your applications. You configure a trigger and set the role that the function can use and then can interface with almost anything you want from databases, to datastores, to other services either publicly on the internet or in your own Amazon Virtual Private Cloud (VPC). **Amazon DynamoDB** is a non-relational serverless database that can scale automatically to handle massive amounts of traffic and data without the need to manage any servers.

 In this module you created a DynamoDB table and a Lambda function to write data into it. In the next module, you create an Amazon API Gateway REST API and connect it to your application to capture ride details from your users.

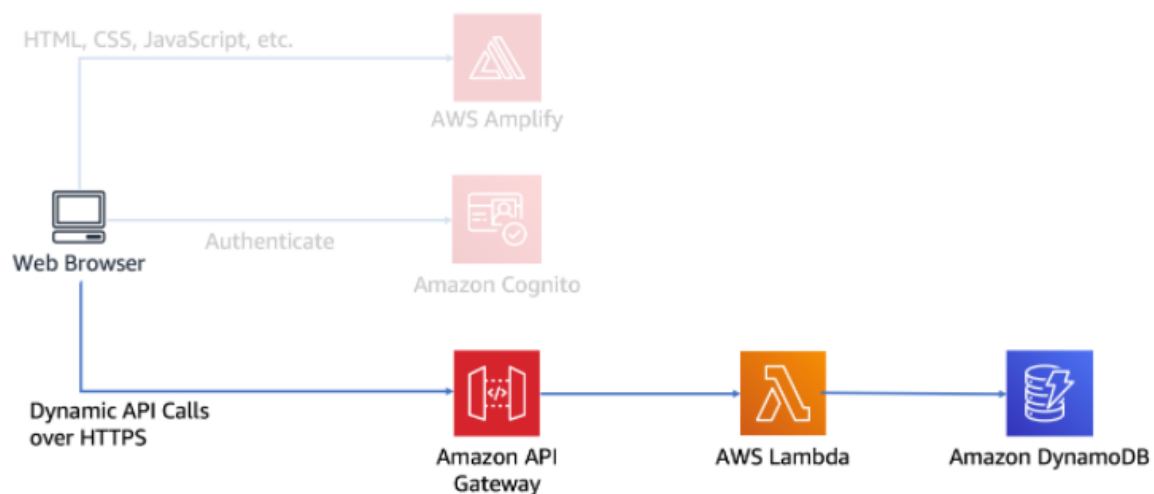
Module 4: Deploy a RESTful API

You will use Amazon API Gateway to expose the Lambda function you built in the previous module as a RESTful API

Overview

In this module you'll 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 you will then turn your statically hosted website into a dynamic web application by adding client-side JavaScript that makes AJAX calls to the exposed APIs.

Architecture Overview



The diagram above shows how the API Gateway component you will build in this module integrates with the existing components you built previously. The grayed out items are pieces you have already implemented in previous steps.

The static website you deployed in the first module already has a page configured to interact with the API you'll 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, your 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, but if you're interested in how the browser code works that calls this API, you can inspect the [ride.js](#) file of the website. In this case the application uses jQuery's [ajax\(\)](#) method to make the remote request.

Create a new REST API

- a. In the AWS Management Console, click Services then select API Gateway under Application Services.
- b. Under **REST API** section, click on the **Build** button.
- c. Select **New API** radio button, and enter **WildRydes** for the API Name.
- d. Keep **Edge optimized** selected 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.
- e. Choose **Create API**

Create a Cognito User Pools Authorizer

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 Module 2.

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.html` page of your current website.

- a. Under your newly created API, choose **Authorizers** (in the left menu).
- b. Choose Create New Authorizer.
- c. Enter **WildRydes** for the Authorizer name.
- d. Select **Cognito** for the type.
- e. In the Region drop-down under Cognito User Pool, select the Region where you created your Cognito user pool in module 2 (by default the current region should be selected).
- f. Enter **WildRydes** (or the name you gave your user pool) in the Cognito User Pool input.
- g. Enter **Authorization** for the Token Source. Take care of **leading/trailing** spaces.
- h. Choose Create.

Verify your authorizer configuration (this is an optional step, auth token below is really long and may get corrupted while you copy between devices)

- j. Open a new browser tab and visit `/ride.html` under your website's domain.
- k. 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.html`.
- l. Copy the auth token from the notification on the `/ride.html`,
- m. Go back to previous tab where you have just finished creating the Authorizer
- n. Click Test at the bottom of the card for the authorizer.
- o. Paste the auth token into the Authorization Token field in the popup dialog.
- p. Click the **Test** button and verify that the **response code is 200** and that you see the claims for your user displayed.

Create a new Resource & 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.

- a. In the left nav, click on **Resources** under your WildRydes API.
- b. From the **Actions** dropdown select **Create Resource**.
- c. Enter `ride` as the Resource Name.
- d. Ensure the Resource Path is set to `ride`.

New Child Resource

Use this page to create a new child resource for your resource.

Configure as ☒ proxy resource

☐ ⓘ

Resource Name*

ride

Resource Path*

/ ride

You can add path parameters using brackets. For example, the resource path `{username}` represents a path parameter called 'username'. Configuring `/[proxy+]` as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to `/foo`. To handle requests to `/`, add a new ANY method on the `/` resource.

Enable API Gateway CORS

☒ ⓘ

* Required

Cancel

Create Resource

- e. Select Enable API Gateway CORS for the resource.
- f. Click **Create Resource**.
- g. With the newly created `/ride` resource selected, from the Action dropdown select Create

Method.

h. Select *POST* from the new dropdown that appears, then **click** the checkmark.

The screenshot shows the AWS API Gateway console. On the left, under 'Resources', the path '/ride' is selected, and the 'OPTIONS' dropdown is set to 'POST'. On the right, the '/ride Methods' tab is active, showing the 'OPTIONS' configuration. Under 'Mock Endpoint', the 'Authorization' is set to 'None' and the 'API Key' is 'Not required'.

i. Select Lambda Function for the integration type.

j. Check the box for Use Lambda Proxy integration.

k. Select the Region you are using for Lambda Region.

l. Enter the name of the function you created in the previous module, *RequestUnicorn*, for Lambda Function.

/ride - POST - Setup



Choose the integration point for your new method.

The screenshot shows the 'Integration type' configuration page. The 'Integration type' is set to 'Lambda Function'. The 'Use Lambda Proxy integration' checkbox is checked. The 'Lambda Region' is set to 'us-east-2'. The 'Lambda Function' is set to 'RequestUnicorn'. The 'Use Default Timeout' checkbox is checked. A 'Save' button is at the bottom right.

m. 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.

n. When prompted to give **Amazon API Gateway permission** to invoke your function, choose **OK**.

o. Click on the **Method Request** card.

p. Choose the pencil icon next to Authorization.

[← Method Execution](#) /ride - POST - Method Request

Provide information about this method's authorization settings and the parameters it can receive.

Settings

Authorization ✓ ✕ ⓘ

Request Validator NONE ✎ ⓘ Update

API Key Required false ✎

q. 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.

a. In the Actions drop-down list select Deploy API.

b. Select [New Stage] in the Deployment stage drop-down list.

c. Enter *prod* for the Stage Name.

d. Choose Deploy.

e. Note the Invoke URL. You will use it in the next section.

Update the Website config

Update the `/js/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 sites `/js/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.

a. Open the config.js file in a text editor.

b. Update the invokeUrl setting under the api key in the config.js file. Set the value to the Invoke URL for the deployment stage you created in the previous section.

An example of a complete *config.js* file is included below. Note, the actual values in your file will be different.

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



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

```
git add .
```

```
git commit -m "Configure API invokeURL"
```

```
git push
```

Validate your Implementation

Note: It is possible that you will see a delay between updating the config.js file in your S3 bucket and when the updated content is visible in your browser. You should also ensure that you clear your browser cache before executing the following steps.

- a. Visit `/ride.html` under your website domain.
- b. If you are redirected to the sign in page, sign in with the user you created in the previous module.
- c. After the map has loaded, click anywhere on the map to set a pickup location.
- d. 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.

Module 5: Resource Cleanup

To finish this experiment, you will go through the steps to terminate all the resources you created throughout this tutorial

Resource Cleanup

In this module, you will go through the steps to terminate all the resources you created throughout this tutorial. You will terminate your AWS Amplify app, an Amazon Cognito User Pool, an AWS Lambda function, an IAM role, a DynamoDB table, a REST API, and a CloudWatch Log. It is a best practice to delete resources you are no longer using to avoid unwanted charges.

Delete your Amplify app.

- a. In the AWS Management Console choose Services then select AWS Amplify under Mobile.
- b. Select the app you created in module 1.
- c. On the app landing page, choose 'Actions > Delete app'. Enter 'delete' when prompted to confirm, then choose confirm.

Delete the CodeCommit repository.

Delete the **Amazon Cognito** user pool you created in module 2.

- a. From the AWS Console click Services then select Cognito under Mobile Services.
- b. Choose Manage your User Pools.
- c. Select the WildRydes user pool you created in module 2.
- d. Choose Delete Pool in the upper right corner of the page.
- e. Type *delete* and choose Delete Pool when prompted to confirm.

Delete the AWS Lambda function, IAM role and Amazon DynamoDB table you created in module 3.

Lambda Function

- a. In the AWS Management Console, click Services then select Lambda under Compute.
- b. Select the *RequestUnicorn* function you created in module 3.
- c. From the Actions drop-down, choose Delete function.
- d. Choose Delete when prompted to confirm.

IAM Role

- a. In the AWS Management Console, click Services then select IAM under Security, Identity & Compliance.

- b. Select Roles from the navigation menu.
- c. Type *WildRydesLambda* into the filter box.
- d. Select the role you created in module 3.
- e. From the Role actions drop-down, select Delete role.
- f. Choose Yes, Delete when prompted to confirm.

DynamoDB Table

- a. In the AWS Management Console, click Services then select DynamoDB under Databases
- b. Choose Tables in the navigation menu.
- c. Choose the Rides table you created in module 3.
- d. Choose Delete table from the Actions drop-down.
- e. Leave the checkbox to Delete all CloudWatch alarms for this table selected and choose Delete.

Delete the REST API created in module 4. There is a Delete API option in the Actions drop-down when you select your API in the Amazon API Gateway Console.

- a. In the AWS Management Console, click Services then select API Gateway under Application Services.
- b. Select the API you created in module 4.
- c. Expand the Actions drop-down and choose Delete API.
- d. Enter the name of your API when prompted and choose Delete API.

AWS Lambda automatically creates a new log group per function in Amazon CloudWatch Logs and writes logs to it when your function is invoked. You should delete the log group for the RequestUnicorn function. Also, if you launched any CloudFormation stacks, there may be log groups associated with custom resources in those stacks that you should delete.

- a. From the AWS Console click Services then select CloudWatch under Management Tools.
- b. Choose Logs in the navigation menu.
- c. Select the `/aws/lambda/RequestUnicorn` log group. If you have many log

groups in your account, you can type */aws/lambda/RequestUnicorn* into the Filter text box to easily locate the log group.

d. Choose Delete log group from the Actions drop-down.

e. Choose Yes, Delete when prompted to confirm.

f. If you launched any CloudFormation templates to complete a module, repeat steps 3-5 for any log groups which begin with */aws/lambda/wildrydes-webapp*.

THANK YOU