Building Serverless Solutions on AWS
Lab Guide
Version 1.0

AWS-200-BSS-10-EN

Corrections or feedback on the course, please email us at:
aws-course-feedback@amazon.com.
For all other questions, contact us at:
https://aws.amazon.com/contact-us/aws-training/.

Contents

Lab 1
Dynamic Web Application

Overview

In this lab session, you will configure a dynamic web application that will serve as a front-end for requesting a *Wild Ryde* from a unicorn. You will learn how Amazon Simple Storage Service (S3) can be used to host static websites, how to integrate the AWS JavaScript SDK to leverage services like Amazon API Gateway and Amazon Cognito, use AWS Lambda to write to and read from a database (DynamoDB), and how to route the Lambda interactions using Amazon API Gateway.

Objectives

After completing this lab, you will be able to:

- Task 1 – Create a static website in Amazon S3
    - 1.1 Register with Mapbox
    - 1.2 Modify JavaScript and upload to Amazon S3
    - 1.3 Verify basic functionality of your application
- Task 2 – Implement API Gateway functionality
    - 2.1 Observe GET method for /rides
    - 2.2 Create an AWS Lambda function
    - 2.3 Implement GET method for /rides/{requestid}
- Task 3 – Finalize code to request a ride
    - 3.1 Review existing Lambda code
    - 3.2 Add code to write to Amazon DynamoDB
    - 3.3 Add code to asynchronously dispatch a unicorn

Prerequisites

This lab requires the following:

- Access to a computer with Wi-Fi running Microsoft Windows, Mac OS X, or Linux (Ubuntu, SuSE, or Red Hat).

- The qwikLABS lab environment is not accessible using an iPad or tablet device, but you can use these devices to access the student guide.
- An Internet browser such as Chrome, Firefox, or IE9 or later (previous versions of Internet Explorer are not supported).

Duration

This lab will take approximately 45 minutes.

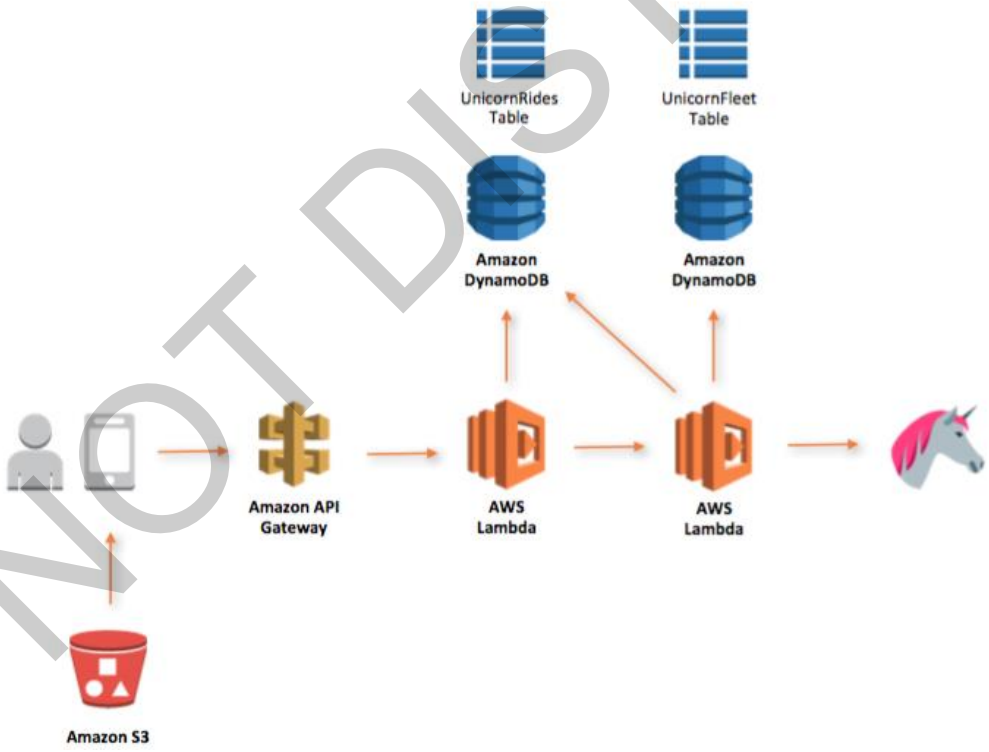
Task 1: Create a Static Website in Amazon S3

Overview

In this section you will modify a static website hosted in Amazon S3. Client-side websites hosted in S3 can be made interactive by taking advantage of client-side libraries and frameworks. For this application, we will use jQuery and the Amazon API Gateway SDK for JavaScript to add the required functionality for our serverless application.

Scenario

In this lab you will complete the following infrastructure:



Task 1.1: Register with Mapbox

A dynamic web application consisting of HTML, CSS, and JavaScript has been created in an Amazon S3 bucket for you. The application uses a mapping solution from Mapbox to allow users to travel between points on a live map of the world. In this task, you will register with Mapbox and retrieve your API key needed to enable the map.

1.1.1 Visit **https://www.mapbox.com** and click the **Sign Up** button

1.1.2 Create a **Username, Email**, and **Password, and** then click the **Sign Up** button.

1.1.3 After signing up with Mapbox, visit your Account settings at **https://www.mapbox.com/studio/account/.** Select the menu option for **API Access Tokens**, then click **Default Public Token** dropdown to reveal your public Mapbox token

1.1.4 Copy the token to your clipboard. You will need it in Task 1.2.

Task 1.2: Modify JavaScript and Upload to Amazon S3

In this task you will modify existing JavaScript code to include your Mapbox API key.

1.2.1 In the **AWS Management Console**, on the **Services** menu, click **Storage & Content Delivery > S3**

1.2.2 Click on the *wildrydes-xxxxxxxxxxxx* bucket, then click on the *js* folder within the bucket to list the contents.

1.2.3 Right-mouse click on *config.js*, then click **Download.**

1.2.4 Right-mouse click the download link and choose “Save Link as…” to save the file on your computer.

1.2.5 Using the text editor of your choice, enter the token copied in task 1.1 (from your MapBox account) to the **accessToken** key under the mapbox section of the config.js file.

1.2.6 Save *config.js*

1.2.7 Upload *config.js* back to the *js* folder in your bucket, overwriting the original.

Task 1.3: Verify Basic Functionality of Your Application

In this task you will verify that your application is being served correctly from your Amazon S3 bucket.

1.3.1 In the **AWS Management Console**, on the **Services** menu, click **Storage and Content Delivery > S3**

1.3.2 Click on the magnifying glass next to the *wildrydes-xxxxxxxxxxxx* bucket

1.3.3 Click the **Properties** button near the upper right corner of your console

1.3.4 Click the **Static Website Hosting** dropdown in the right pane of the S3 console, and click **Endpoint** to load your website in the browser. It will begin with *wildrydes-xxxxxxxxxxx.*

1.3.5 The serverless application uses Amazon Cognito User Pools to remove the heavy lifting from having to build and maintain your own user directories. In the **Register** section, enter your email address, then create and confirm a password for use on this application.

1.3.6 Click the **Register** button to create your account.

1.3.7 Check your email for a verification code and enter it on the /verify.html page.

1.3.8 After verifying your account you will be redirected to the sign-in page where you can log in using the credentials you just created.

1.3.9 Once signed in, you are taken the main application map page. Verify that the map shows, and that you're able to pan around the world, indicating that the mapping JavaScript loaded correctly and that your Mapbox API was successfully initialized

1.3.10 Click the list icon in the upper left corner of the page to show a list of previous rides.

1.3.11 Click on one of the rides in the list and notice that the details are not properly displayed. If you look in your browser's developer tools console (network activity) you will see a

request being made to /prod/rides/xxxxxxxx-xxxx-xxxx-xxxxxxxxxxxx. This method has not yet been implemented, but you will complete the implementation in the next task.

Task 2: Implement API Gateway Functionality

Overview

You have now created a user and enabled basic mapping for your serverless application. The application will interact with other AWS services by using Amazon API Gateway. API Gateway acts as a front door for applications by enabling standard HTTP methods—like GET and POST—to other endpoints, or other services like Amazon DynamoDB and AWS Lambda.

In your lab environment there is a pre-configured REST API that exposes a GET /rides method for listing previous rides. You were exercising this method in the previous task when viewing the ride list through the static website. In this task you'll implement an additional API method backed by a Lambda function to get the details of an individual ride.

Command Reference Files

Use the command_reference.txt file when copying text or function bodies referenced in this lab manual. This file is available in the Lab Instructions section of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

Task 2.1: Create an AWS Lambda Function

In this task, you will create a function in AWS Lambda to retrieve the details of a particular ride.

2.1.1 In the **AWS Management Console**, on the **Services** menu, click **Compute > Lambda**

2.1.2 Click the **Create a Lambda Function** button to begin a new function

2.1.3 Select **Blank Function** from the list of Blueprints

2.1.4 Skip the **Configure triggers** option by clicking **Next** beneath the optional trigger list

2.1.5 In the **Configure Function** screen, name your function **GetRideDetails**, give it a description, and choose a **Runtime** of “Node.js 4.3”

2.1.6 You will edit the Lambda function code inline. Refer to the command reference file to copy, paste, and modify your Lambda code.

2.1.7 Leave “index.handler” as the value for **Handler,** and “Choose an existing role” as the value for **Role.** In the **Existing Role** dropdown, select **WildRydesLambdaRole**.

2.1.8 Use the default values for **Memory** and **Timeout** and **VPC**, then click **Next**.

2.1.9 Review your Lambda function, then click the **Create Function** button.

2.1.10 Note the **ARN** of your newly created function which is shown in the upper right corner of the screen. You will need this value in the next task.

Task 2.2: Configure and Test Your Function

In this task you will add the necessary configuration for your function to run. After the configuration is complete you will test the function through the Lambda console.

2.2.1 In the **AWS Management Console**, on the **Services** menu, click **Database > DynamoDB.**

2.2.2 Click **Tables** in the left menu.

2.2.3 Select the **Rides-xxxxx** table and note the table name.

2.2.4 Select the **Fleet-xxxxx** table and note the table name.

2.2.5 Select the **ServerlessBootcampConfig** table, select the Items tab and click **Create Item**.

2.2.6 Enter the ARN of the **GetRideDetails** function you created in the previous task in the "Function" field.

2.2.7 Click the plus symbol and **Append** a String field. Name it "RidesTable" and set the value to the name of the table noted in step 2.2.3. Use the same process to create a FleetTable attribute with the noted value from step 2.2.4.

2.2.8 Click **Save** to persist the configuration record for that function.

2.2.9 Return to the Lambda console and select the **GetRideDetails** function you created in the previous task.

2.2.10 From the **Actions** menu select **Configure test event**.

2.2.11 Enter the following JSON document for the test event replacing the UserId value with the email address you used to register with the site.

```
{
  "UserId": "[Your email address]",
  "RequestId": "11111111-1111-1111-111111111111"
}
```

2.2.12 Click **Save and test** from the event editor dialog.

2.2.13 Verify your function returns valid ride data in the test result area.

Task 2.3: Implement GET method for /rides/{id}

In this task, you will create a new resource and method within the Wild Rydes API and connect it to the Lambda function created in the previous task.

2.3.1 In the **AWS Management Console**, select the **Services** menu, then select **Application Services** > **API Gateway**

2.3.2 Click **WildRydes-XXXX** under **APIs** on the navigation pane on the left and then select **Resources**

2.3.3 In the **Resources** pane, highlight the **{id}** resource under **/rides**, then click the **Actions** button and select **Create Method** to create a new method, choosing **GET** and save the method by clicking the checkmark.

2.3.4 In the **GET – Setup** pane, choose an **Integration Type** of "Lambda Function", then select eu-west-1 from the **Lambda Region** dropdown.

2.3.5 For **Lambda Function** type in the name of the Lambda function you created in task 2.2, "GetRideDetails". (API Gateway should automatically complete this value for you as you begin to type). Click the **Save** button to continue.

2.3.6 When prompted to **Add Permission to Lambda Function** click **OK**. This will grant the API Gateway service permission to invoke your Lambda function when a request is received.

2.3.7 Click on the **Method Request** section.

2.3.8 Click the pencil icon next to the **Authorization** field and select **CognitoAuthorizer**. Click the checkmark to save your selection.

2.3.9 Expand the **HTTP Request Headers** section and click **Add Header**.

2.3.10 Enter "Authorization" for the header name and click the checkmark to save.

Note: The Authorization header is used by the custom Cognito Authorizer that was created for you when the lab was launched. You can view the configuration details for this custom authorizer by clicking the **Authorizers** menu option under your API in the left panel.

2.3.11 Click the back arrow labeled **Method Execution** to return to the main method configuration screen.

2.3.12 Click on the **Integration Request** box and expand the **Body Mapping Templates** section.

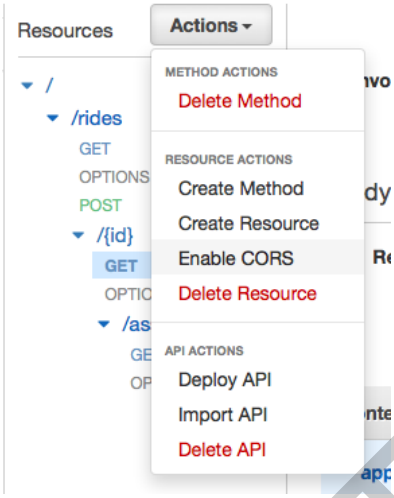
2.3.13 For the **Request body passthrough** option select "When there are no templates defined (recommended)."

2.3.14 Click **Add mapping template**.

2.3.15 In the **Content-Type** box that appears, type “application/json”, then click the checkbox to save.

2.3.16 In the template body box, add the following lines, and click **Save.** This template can be copy/pasted from the command reference file.

2.3.17 Select **Enable CORS** from the **Actions** dropdown menu.



2.3.18 Keep the default values for all items and click **Enable CORS and replace existing CORS headers**. Confirm your selection by clicking **Yes, replace existing values** on the confirmation dialog.

Note: For simplicity we are allowing access to all origins on this method, but in a real production deployment you should list only trusted origins in this header.

2.3.19 Deploy the changes you have made to the API by selecting **Deploy API** from the **Actions** dropdown.

2.3.20 For **Deployment stage,** choose “prod.” Give a description that indicates what you have done, like “Enabled GET method for rides/{id}”, then click **Deploy.**

2.3.21 In the **Prod Stage Editor** click on the **SDK Generation** tab, then choose “JavaScript” from the **Platform** dropdown. Click **Generate SDK** to download a ZIP file containing your latest API resources and method functions.

2.3.22 Unzip the file that downloaded, and upload “apigClient.js” to the “js” folder of the Amazon S3 bucket where your website code is begin hosted

2.3.23 Verify that your new code is working by refreshing the map page in your browser. (You may have to do a hard-refresh in order for the browser to empty the cached version of your old code. This method varies across operating systems and browsers, please search for the method to hard reset if you are having difficulties). Click on the menu icon once again to display a list of rides, then click on a particular ride. You should now see ride details loading in the details modal that pops up. If you do, you have successfully integrated Lambda with API Gateway, and generated a new JavaScript SDK that is helping your programmatically access this integration.

Task 2.4: Add Error Response Mappings

In this task we'll update the new GET /rides/{id} method to properly handle and report errors by mapping the error messages returned by Lambda to appropriate HTTP status codes.

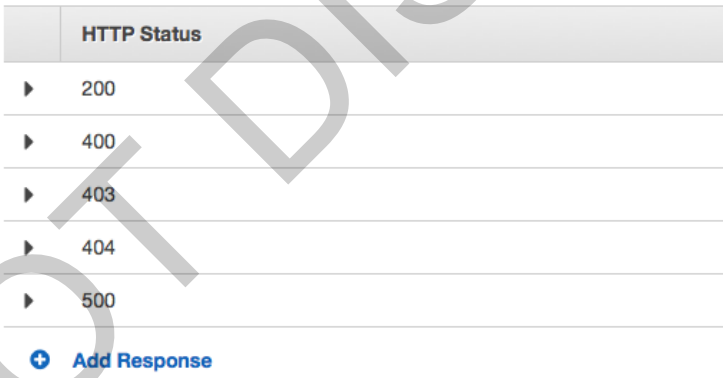
The code provided for the GetRideDetails function does some basic input validation and error handling during the course of retrieving the ride details from DynamoDB. If an error occurs, the function invokes the Lambda callback method with an error message to indicate the type of failure. With the current API Gateway configuration for the GET /rides/{id} method, even when the Lambda function returns an error, the HTTP call will still return a 200 OK status to the client. This task will change that behavior to use the appropriate HTTP status codes for each type of error.

2.4.1 In the **AWS Management Console**, select the **Services** menu, then select **Application Services** > **API Gateway**

2.4.2 Select the WildRydes API and navigate to the GET method of the /rides/{id} resource.

2.4.3 Click on the **Method Response** box.

2.4.4 Use the **Add response** link to create entries for 400, 403, 404 and 500 HTTP Status codes.



2.4.5 Click the back arrow to return to the **Method execution** and click the **Integration Response** box.

2.4.6 Add the following integration response entries by clicking the **Add integration response** link and then saving each one. These values are available to copy paste from the command reference.

Lambda Error Regex	Method response status
^[BadRequest].*	**400**
^[Forbidden].*	**403**

^[Not Found].*	**404**
^[Internal].*	**500**

The final table should look like this when you are finished.

	Lambda Error Regex	Method response status	Output model	Default mapping	
▸	-	200		Yes	⊗
▸	^[BadRequest].*	400		No	⊗
▸	^[Forbidden].*	403		No	⊗
▸	^[Not Found].*	404		No	⊗
▸	^[Internal].*	500		No	⊗

2.4.7 Deploy your API by selecting **Actions > Deploy API**. Select the prod stage and click **Deploy**.

Task 3: Finalize Code to Request a Ride

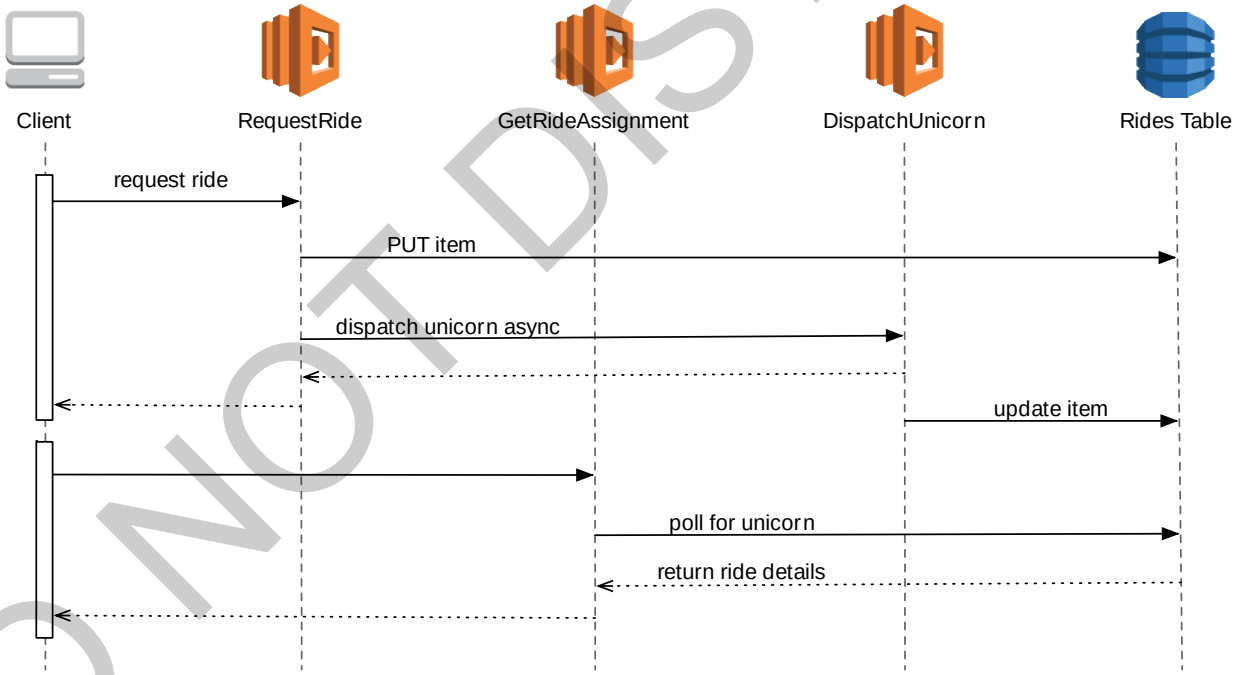
Overview

In Task 2, you wired up an API Gateway method to Lambda which retrieves the details of an existing ride. In this task you will complete the implementation of a method for requesting a new ride.

In your lab environment there is a stub Lambda function called RequestRide that is already connected to the POST /rides method of the WildRydes API. The current implementation of the RequestRide function extracts data about the pickup and destination locations from the received event and builds a ride object. It also does some basic input validation, but the function does not persist anything to DynamoDB or call the unicorn dispatch service. In this task you will complete the implementation such that a new ride item is persisted to DynamoDB and the dispatch function is invoked.

Dispatch architecture

A sequence diagram depicting the flow of a ride request is shown below.



When a client requests a ride the RideRequest function first adds a new ride item to the Rides table, then invokes the DispatchUnicorn function asynchronously and returns. The DispatchUnicorn function will then find an available unicorn and update the ride item in the Rides table with to reflect the new unicorn assignment.

Command Reference File

The command_reference.txt file contains snippets of code used in this task. Use this files when copying text provided in this lab manual. These files are available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste code directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference files to your computer instead.

Task 3.1: Review and Test Existing Lambda Code

In this task you will configure and test the existing RequestRide implementation. The current function extracts the necessary data from the received event and builds a ride object. It also does some basic input validation, but the function does not persist anything to DynamoDB or call the unicorn dispatch service. Adding in these steps will be done in subsequent tasks.

3.1.1 In the **AWS Management Console**, select the **Services** menu, then select **Compute > Lambda**

3.1.2 Choose **RequestRide** from the list of existing Lambda functions.

Note: The following test will result in an error. This will be corrected in subsequent tasks.

3.1.3 Select **Actions > Configure test event** and enter the following JSON for the event body. Replace the placeholder text with your email address for the UserID.

```
{
  "UserId": "[Your email address]",
  "PickupLocation": {
      "Lat": 56.2313,
      "Lon": 23.4002,
      "Address": "Pickup address"
  },
  "DestinationLocation": {
      "Lat": 43.9012,
      "Lon": 54.3342,
      "Address": "Destination address"
  }
}
```

3.1.4 Click **Save and test** to test the function with the new event. The output of the test should be an error indicating that the function has not been fully implemented yet. You will complete this implementation in the following steps.

Task 3.2: Add Code to Write a New Record to DynamoDB

In this task you will update the doRequest() function to write a new item to DynamoDB

3.2.1 From the Lambda function editor console for the RequestRide function, find the doRequest() function on line 57. Add code to write the ride object as a new item in the Rides DynamoDB table.

You can use the docClient object to make this request using the DynamoDB document client. See the documentation for the put method for more information. You will need the full table name for your environment's Rides table which is available in the **config.RidesTable** property.

A completed version of the doRequest() function is available in the command_reference.txt file.

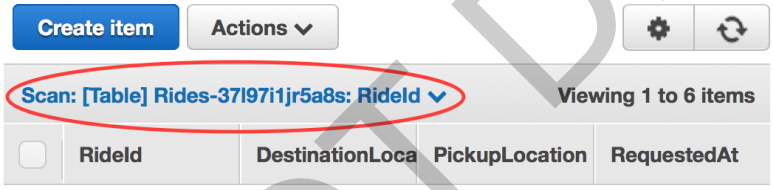
3.2.2 When you have completed your implementation click the **Save and Test** button and verify the log output is what you expected. Note the new ride ID which will be the same as the Lambda request ID so you can verify this record was created in DynamoDB.

3.2.3 Select the **Services** menu, then select **Database > DynamoDB**

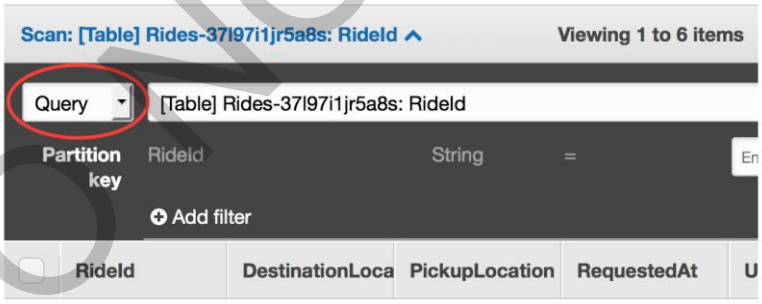
3.2.4 Select **Tables** in the left menu and then select the Rides-xxxxx table.

3.2.5 Click the **Items** tab.

3.2.6 Expand the query dialog by clicking the **Scan: [Table]…** heading above the items list.



3.2.7 Select **Query** from the action dropdown.



3.2.8 Enter the ride ID noted in step 3.2.2 in the **Partition key** value and click **Start Search**. You may need to scoll down in query dialog to see the search button.

3.2.9 Verify the item created by your test function is complete.

Task 3.3: Add Code to Asynchronously Dispatch a Unicorn

In the previous task, you added code to write a new ride item to DynamoDB, but the unicorn dispatch service is still not being invoked. In this task you will add code to invoke the DispatchUnicorn function asynchronously after ride item is persisted.

3.3.1 From the Lambda function editor console for the RequestRide function, edit the doRequest() function so that the DispatchUnicorn function is invoked asynchronously after the ride item is persisted. An instance of the AWS.Lambda service is already instantiated and assigned to the global variable 'lambda'. Be sure to use the 'Event' invocation type in order to make the request asynchronous. See the documentation for the invoke method for more information.

When invoking the DispatchUnicorn function you should pass the ride object as the invocation payload. When using the JavaScript SDK to invoke a Lambda function, remember to use the JSON.stringify() method to convert the payload to a string before passing it to the invoke method.

You can see a completed version of the doRequest() function in the command reference file and a complete version of the entire RequestRide function is available in the command reference file.

3.3.2 When you have completed your implementation click the **Save and Test** button and verify the log output is what you expected. Note the new ride ID which will be the same as the Lambda request ID so you can verify this record was created and updated in DynamoDB.

3.3.3 Select **Functions** from the left menu of the Lambda console and then select the **DispatchUnicorn** function.

3.3.4 Click the **Monitoring** tab and then click **View logs in CloudWatch** to open the log group for the DispatchUnicorn function.

3.3.5 Click on the most recent log stream to view the invocation logs for the DispatchUnicorn function.

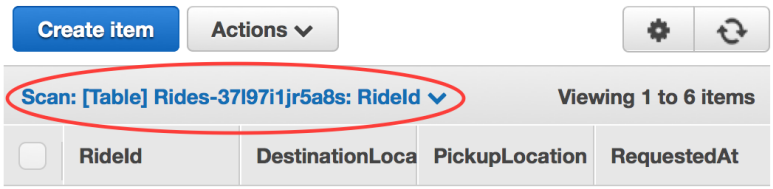
3.3.6 Verify that the ride logs do not show any errors and that a unicorn was successfully dispatched for the ride.

3.3.7 Select the **Services** menu, then select **Database > DynamoDB**

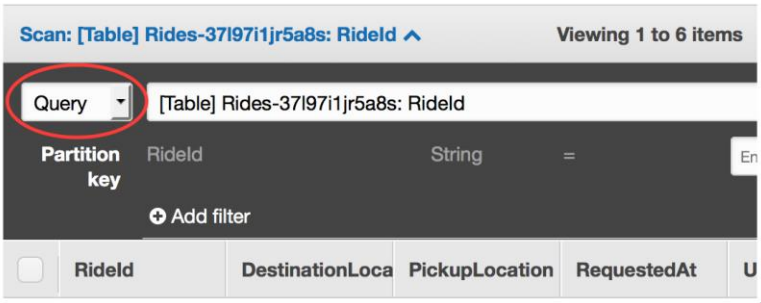
3.3.8 Select **Tables** in the left menu and then select the Rides-xxxxx table.

3.3.9 Click the **Items** tab.

3.3.10 Expand the query dialog by clicking the **Scan: [Table]…** heading above the items list.



3.3.11 Select **Query** from the action dropdown.



3.3.12 Enter the ride ID noted in step 3.2.2 in the **Partition key** value and click **Start Search**. You may need to scoll down in query dialog to see the search button.

3.3.13 Verify the item has a UnicornId field populated.

3.3.14 Visit your web application and sign in using the credentials you created in task 1.

3.3.15 On the map page, click an area on the map to set your destination and then click **Request a ride.**

3.3.16 Observe that your ride is requested and a unicorn is dispatched. You can also view the Rides table in the DynamoDB console to see the new ride item and inspect the logs for the RequestRide, DispatchUnicorn and GetRideAssignment functions in CloudWatch Logs.

Lab Complete

Congratulations! You have successfully completed the lab for Dynamic Web Application.

1. Log out of the **AWS Management Console** by clicking **awsstudent** in the top right corner and click **Sign Out**.
2. Return to the **qwikLABS** page where you launched your lab and click **End**.

Lab 2
Serverless Stream Processing

Overview

In this lab, you will use DynamoDB streams to capture changes to items stored in the rides DynamoDB table. This stream will be used to create event notifications using SNS, and indexing into the Amazon Elasticsearch Service for backend analysis later. We'll also see how Lambda uses the IAM role to be authorized to particular permissions in the account.

Objectives

After completing this lab, you will be able to:

- Task 1 – Creating DynamoDB Streams
    - 1.1 Turn on the DynamoDB streams for the rides table
    - 1.2 Set up the event sources for the two lambda functions
    - 1.3 Test inserting a message to populate the change event on the stream
- Task 2 – Investigating Logs and Calling SNS
    - 2.1 Examine Streaming Lambda Logs in CloudWatch
    - 2.2 Edit function to call SNS for a notification
- Task 3 – Setting permission in IAM
    - 3.1 Add IAM permissions
    - 3.2 Run another test
    - 3.3 Inspect messages on SQS

Prerequisites

This lab requires the following:

- Access to a computer with Wi-Fi, running Microsoft Windows, Mac OS X, or Linux (Ubuntu, SuSE, or Red Hat). The qwikLABS lab environment is not accessible using an iPad or tablet device, but you can use these devices to access the student guide.
- An Internet browser such as Chrome, Firefox, or IE9 or later (previous versions of Internet Explorer are not supported).

Duration

This lab will take approximately 45 minutes.

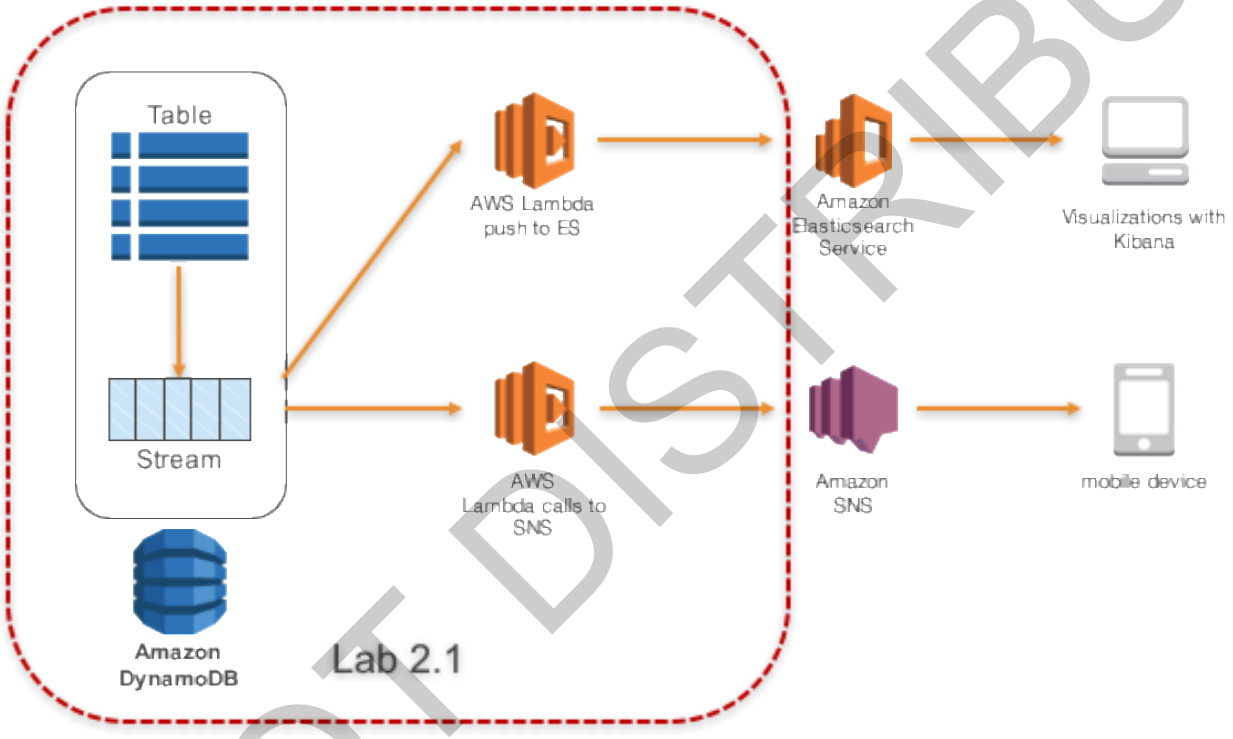
Task 1: Creating and Configuring DynamoDB Streams

Overview

In this section, you will turn on the rides DynamoDB Stream.

Scenario

In this lab, you will build the following infrastructure:



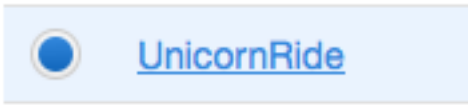
Task 1.1: Create the UnicornRides DynamoDB Stream

In this task, you will create the change item stream from the DynamoDB table.

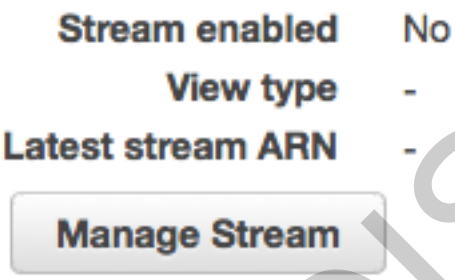
1.1.1 In the **AWS Management Console**, on the **Services** menu, click **DynamoDB**.

1.1.2 In the navigation pane, click **Tables**.

1.1.3 Click on **UnicornRide** in the list.

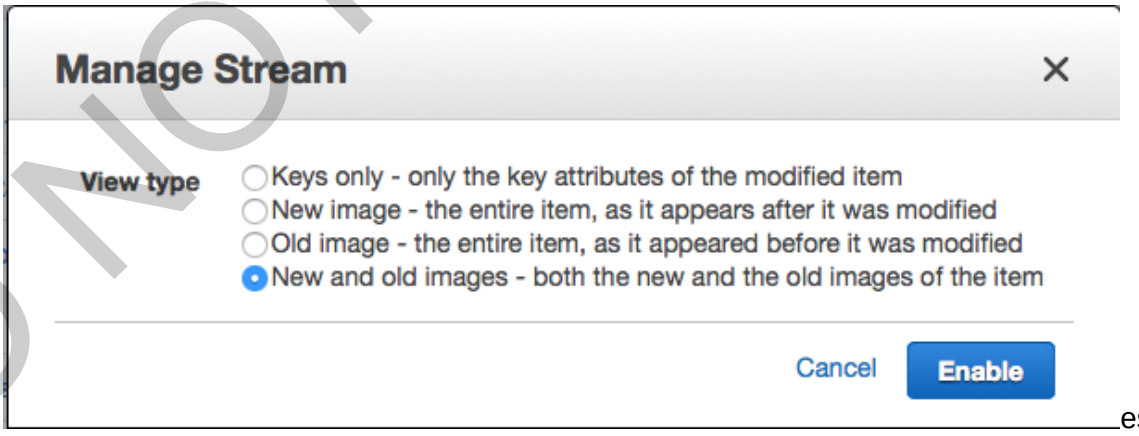


1.1.4 Click **Manage Stream** in the streams section of the Table properties.



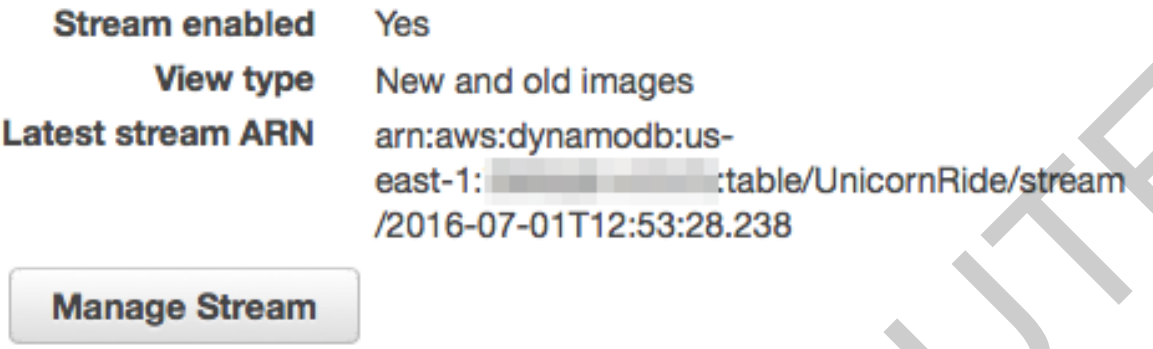
1.1.5 Keep the default type selected and click **Enable**.

This allows you to specify what data gets pushed during item changes to the DynamoDB Stream from the UnicornRides table.



es

1.1.6 After creating the stream, you will see the ARN listed in the details of the DynamoDB Table.

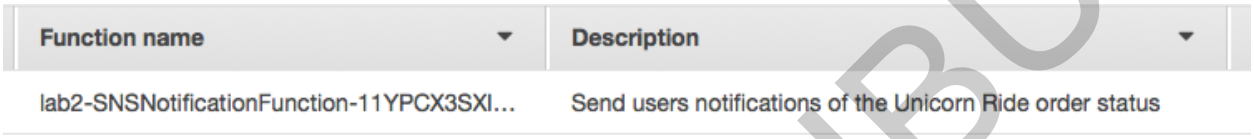


Task 1.2: Configure DynamoDB Stream Triggering Lambda

In this task, you will configure the UnicornRides DyanmoDB Stream as an event source for two Lambda functions.

1.2.1 On the **Services** menu, click **Lambda**.

1.2.2 Click the **[*CloudformationStackName*]-SNSNotificationFunction-*** function name.



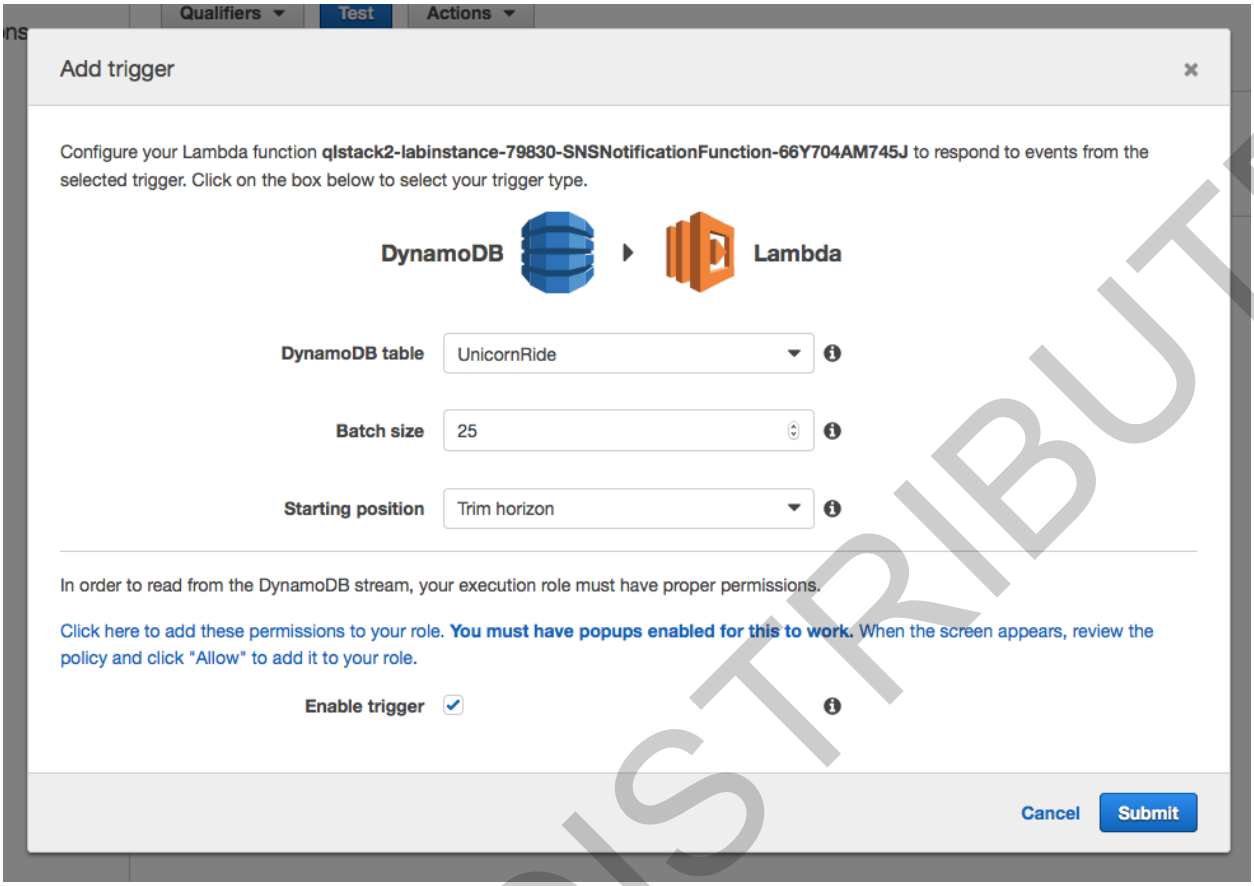
1.2.3 Click the **Triggers** tab under the function details.

1.2.4 Click **Add trigger**.

1.2.5 In the **Add trigger** dialog box, fill out the following information:

Field	Value
Event source type	DynamoDB
DynamoDB table	UnicornRide
Batch size	25
Starting position	Trim horizon
Enable trigger	Checked (Default)

1.2.6 Click **Submit** to add the new trigger.



1.2.7 Click the **[*CloudformationStackName*]-WriteToESFunction-*** function name.

1.2.8 Perform the same steps [1.2.3 – 1.2.6] for that function to add the stream as an event source.

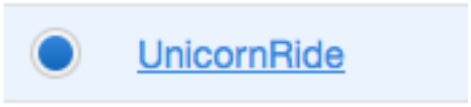
Task 1.3: Test the Stream

Next, you'll test the stream by manually inserting a new record into the UnicornRide DynamoDB table.

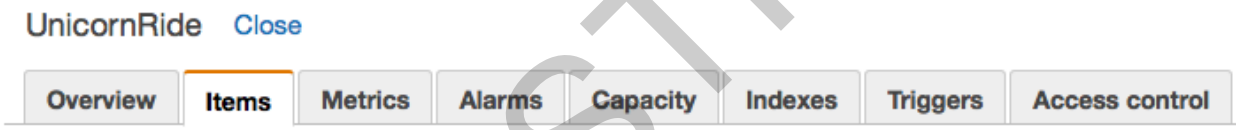
1.3.1 Go back to the **AWS Management Console** and on the **Services** menu, click **DynamoDB**.

1.3.2 In the navigation pane, click **Tables**.

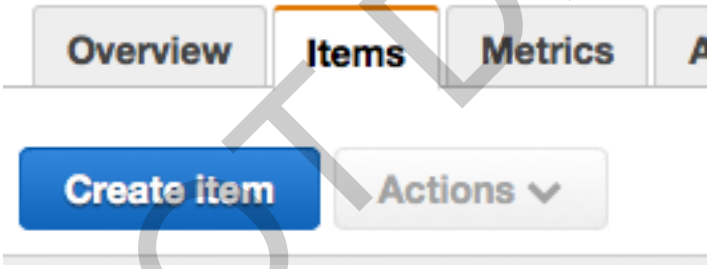
1.3.3 Click **UnicornRides**.



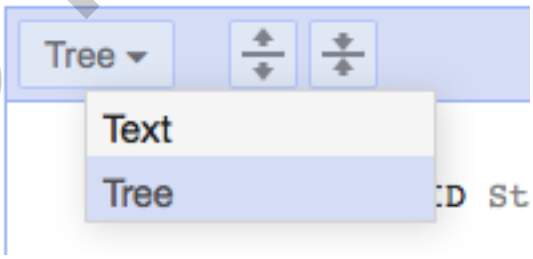
1.3.4 Click **Items**.



1.3.5 Click **Create item**.



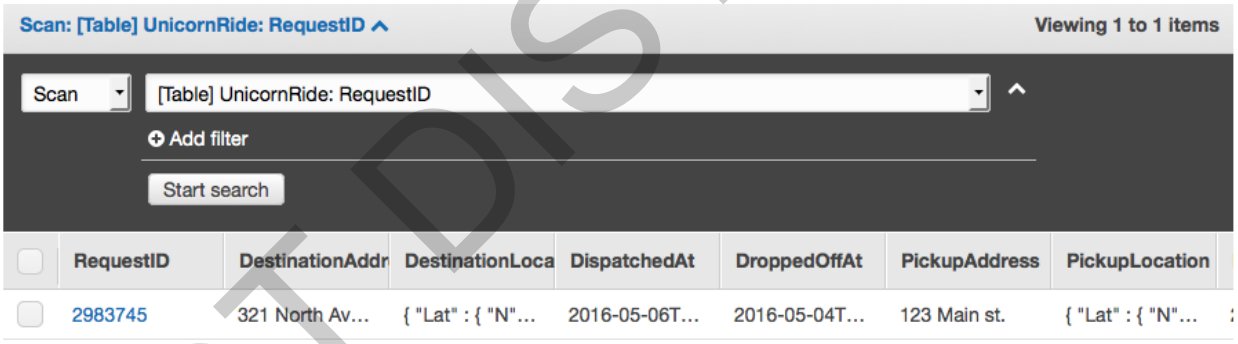
1.3.6 In the **Create item** dialog box, for "**Tree**", click "**Text**".



1.3.7 Paste the following text to store as an item in DynamoDB and then click **Save**.

```
{
  "DestinationAddress": "321 North Avenue",
  "DestinationLocation": {
    "Lat": 34,
    "Lon": 43
  },
  "DispatchedAt": "2016-05-06T13:27:14Z",
  "DroppedOffAt": "2016-05-04T13:27:14Z",
  "PickupAddress": "123 Main st.",
  "PickupLocation": {
    "Lat": 24,
    "Lon": 54
  },
  "RequestedAt": "2016-04-04T13:27:14Z",
  "RequestID": "2983745",
  "UnicornId": 7000,
  "UserID": "Favorite Customer"
}
```

1.3.8 You'll see the request added to the table.



1.3.9 This will automatically trigger an event on the stream and the lambda functions will execute. The batch size specified is a max value, the event will trigger even with the single message inserted into the DynamoDB table.

1.3.10 On the **Services** menu, click **Elasticsearch Service**.

1.3.11 Notice under the Amazon Elasticsearch Service dashboard that there is a new searchable document in Elasticsearch.

Task 2: Investigating Logs and Calling SNS

Overview

So, now you've turned on DynamoDB streams for your UnicornRide table, and edited the configuration of 2 Lambda functions to pull from that stream and process that data. One of those functions has code to put the unicorn ride data into an ElasticSearch cluster, which you viewed the results of in the last task. The other Lambda function does a push notification using Amazon Simple Notification Service (SNS) to notify users of ride confirmations, completions, etc. In this task, you'll see how to debug a function that's not working properly.

Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

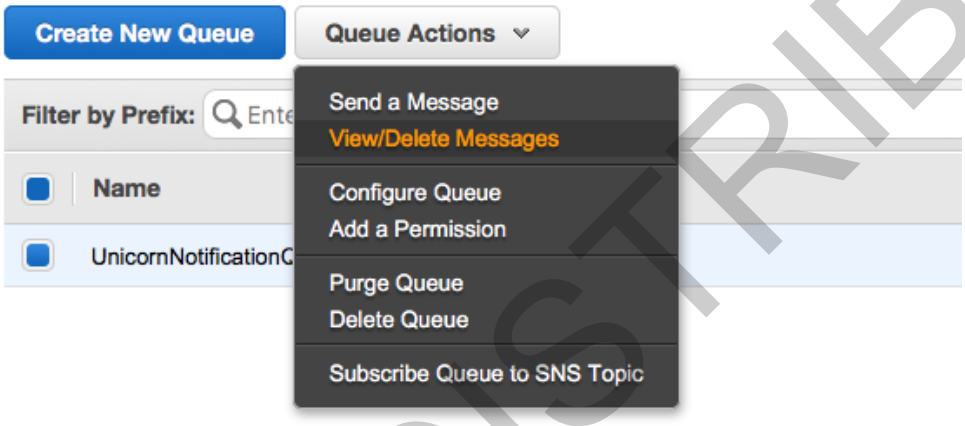
Task 2.1: Examine Streaming Lambda Logs in CloudWatch

This task walks you through examining the logs of your Lambda function.

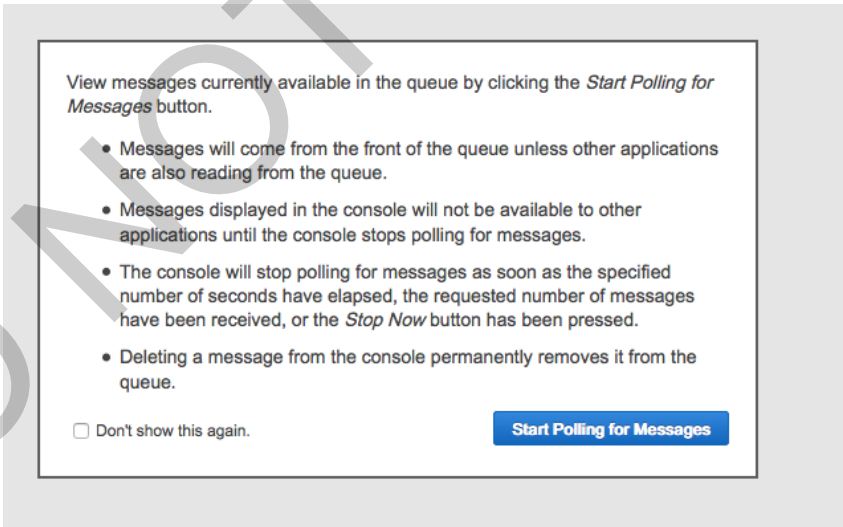
2.1.1 We have configured a queue in our Simple Queue Service to listen to the SNS topic that has been created. Alternatively, you could set up SNS to send you SMS or email messages and many other options. On the **Services** menu, click **SQS**.

2.1.2 Select **UnicornNotificationQueue.**

2.1.3 For **Queue Actions**, click **View/Delete Messages**.



2.1.4 Click **Start Polling for Messages**.



2.1.5 No notification message is displayed. Let's inspect the Lambda code to find out why.

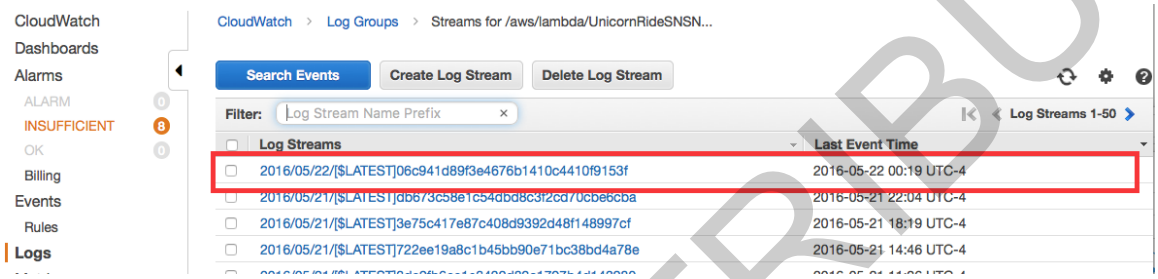
2.1.6 On the **Services** menu, click **Lambda**.

2.1.7 Click the **[*CloudformationStackName*]-SNSNotificationFunction-*** function name.

2.1.8 Click **Monitoring**.

2.1.9 Here you can see metrics from the different invocations of your Lambda function. Click on **View Logs in CloudWatch** in the upper-right corner, to see logs from the latest run of your function. This will take you to the CloudWatch Logs console.

2.1.10 The latest CloudWatch Logs stream (at the top), will have the logs from your latest run. Click on the first log stream in the list as shown below:



2.1.11 In here you can see the logs from the different runs of your Lambda function. You should see a message that looks like: **"Fix me! I am not publishing messages yet!"**

Task 2.2: Edit function to call SNS for a notification

This task walks you through launching an EC2 instance into your VPC. This instance will act as your web server.

2.2.1 So, it looks like the code needs to be fixed. You'll need to go into the Lambda function's code and configure it to send the messages to SNS. On the **Services** menu, click **Lambda**.

2.2.2 Select the **[*CloudformationStackName*]-SNSNotificationFunction-*** function name.

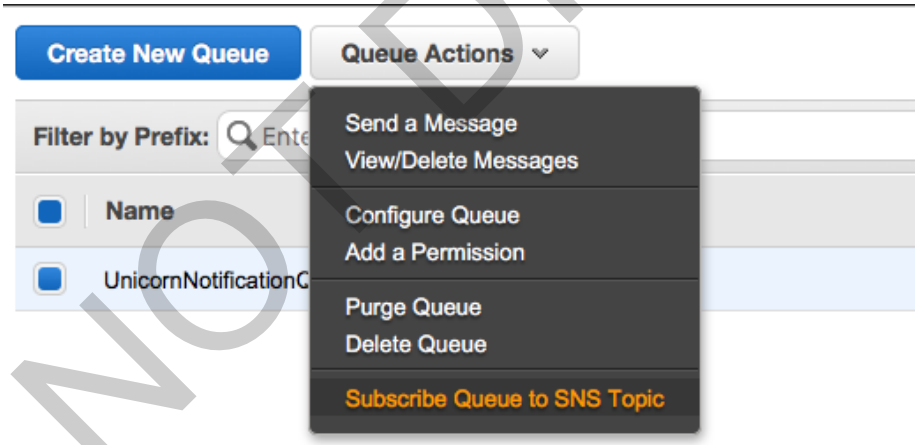
2.2.3 On the **Code** tab, take a minute to inspect the code to see what it is doing with the records on the DynamoDB stream. It handles insertions, modifications, and deletions to the Dynamo table differently, sending different messages in each case. Scroll down to the bottom and where it says "//INSERT SNS PUBLISH CODE HERE", replace that line with the code from the command references file. Also, delete the line with the log message that says "Fix me…".

2.2.4 Click **Save**.

2.2.5 On the **Services** menu, click **SQS**.

2.2.6 Select the queue named **UnicornNotificationQueue.**

2.2.7 For **Queue Actions**, click **Subscribe Queue to SNS Topic**.



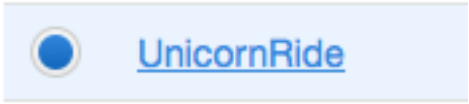
2.2.8 For **Choose a Topic**, click **UnicornRideNotification** SNS topic.

2.2.9 Click **Subscribe** and then click **OK**.

2.2.10 On the **Services** menu, click **DynamoDB**. Now, you're going to repeat the steps from Task 1.3 to test the function.

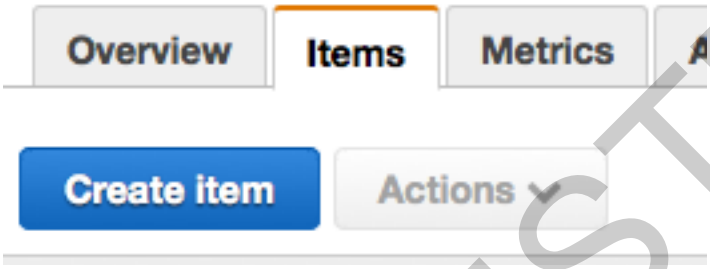
2.2.11 In the navigation pane, click **Tables**.

2.2.12 Click **UnicornRides**.

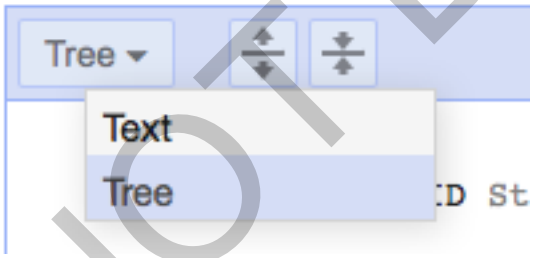


2.2.13 Click **Items**.

2.2.14 Click **Create item**.



2.2.15 In the **Create item** dialog box, for **Tree**, click **Text**.



2.2.16 Paste in the following text to store as an item in DynamoDB and click **Save**.

```
{
  "DestinationAddress": "321 South Avenue",
  "DestinationLocation": {
    "Lat": 34,
    "Lon": 43
  },
  "DispatchedAt": "2016-06-06T13:27:14Z",
  "DroppedOffAt": "2016-06-04T13:27:14Z",
  "PickupAddress": "123 Main st.",
```



```
  "PickupLocation": {
    "Lat": 24,
    "Lon": 54
  },
  "RequestedAt": "2016-06-04T13:27:14Z",
  "RequestID": "1234567",
  "UnicornId": 8000,
  "UserID": "2nd Favorite Customer"
}
```

2.2.17 You'll see the request added to the table.

2.2.18 This will automatically trigger an event on the stream and the lambda functions will execute. The batch size specified is a max value, the event will trigger even with the single message inserted into the DynamoDB table.

2.2.19 On the **Services** menu, click **SQS**.

2.2.20 Select **UnicornNotificationQueue.**

2.2.21 For **Queue Actions**, click **View/Delete Messages**.

2.2.22 Click **Start Polling for Messages**.

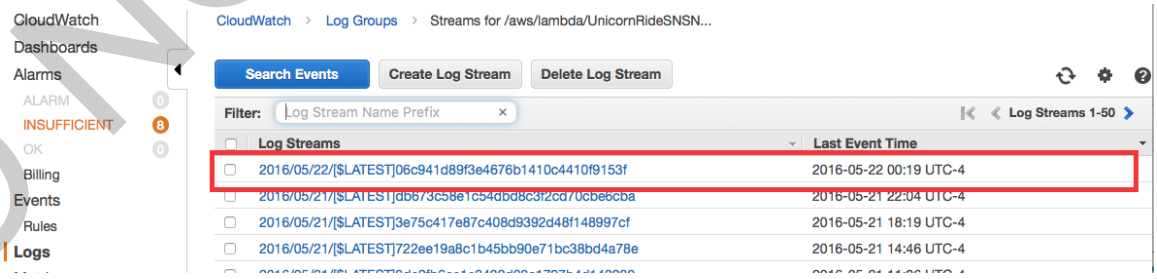
2.2.23 There's still no message there though. So, go back to the Lambda logs to figure out why. On the **Services** menu, click **Lambda**.

2.2.24 Click the **[*CloudformationStackName*]-SNSNotificationFunction-*** function name.

2.2.25 Click **Monitoring**.

2.2.26 Here you can see metrics from the different invocations of your Lambda function. Click on **View Logs in CloudWatch** in the upper-right corner, to see logs from the latest run of your function. This will take you to the CloudWatch Logs console.

2.2.27 The latest CloudWatch Logs stream (at the top), will have the logs from your latest run. Click on the first log stream in the list as shown below:



Here, you can see the logs from the different runs of your Lambda function. Scroll all the way to the bottom. There should be an error message, indicating an Authorization Error like the following:

[AuthorizationError: User: <IAM ROLE FOR YOUR LAMBDA FUNCTION> is not authorized to perform: SNS:Publish on resource: <SNS TOPIC>]

This indicates that the IAM role for the Lambda functions does not have permissions to publish to SNS. You'll learn how to fix this in the next task.

Task 3: Setting Permissions in IAM

Overview

At the end of the last task, you saw your Lambda function fail due to an authorization error. Lambda functions assume an IAM role when executing, and so the IAM role must have permissions to call any AWS services or APIs that you want the Lambda function to interact with. You added in the ability to the Lambda function to call the Simple Notification Service to publish a message. However, you did not add the ability to publish a message to the IAM role. You'll need to check the permissions in the IAM role of the Lambda function to ensure it has permissions to call SNS and publish a message.

Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

Task 3.1: Add IAM Permissions

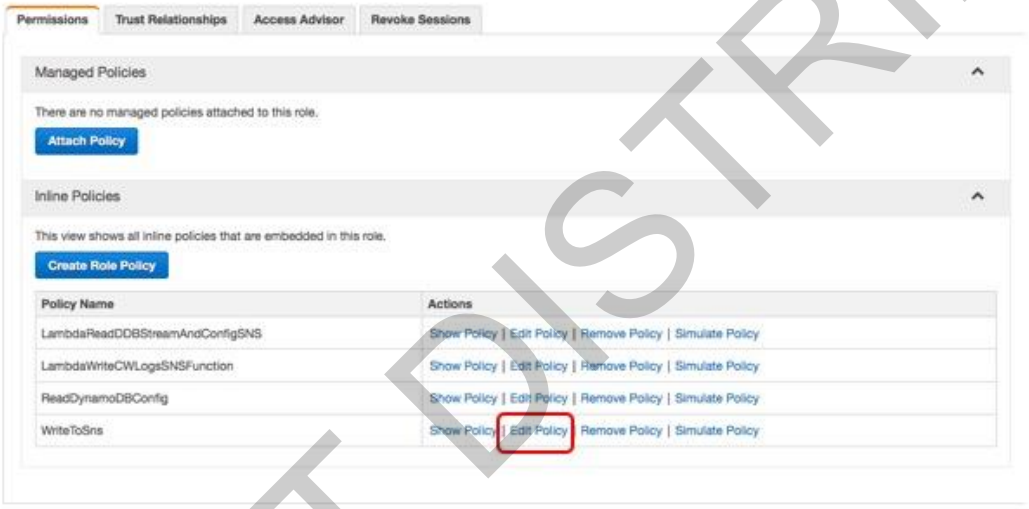
This task walks you through editing the IAM role that the Lambda executes under.

3.1.1 On the **Services** menu, click **IAM**.

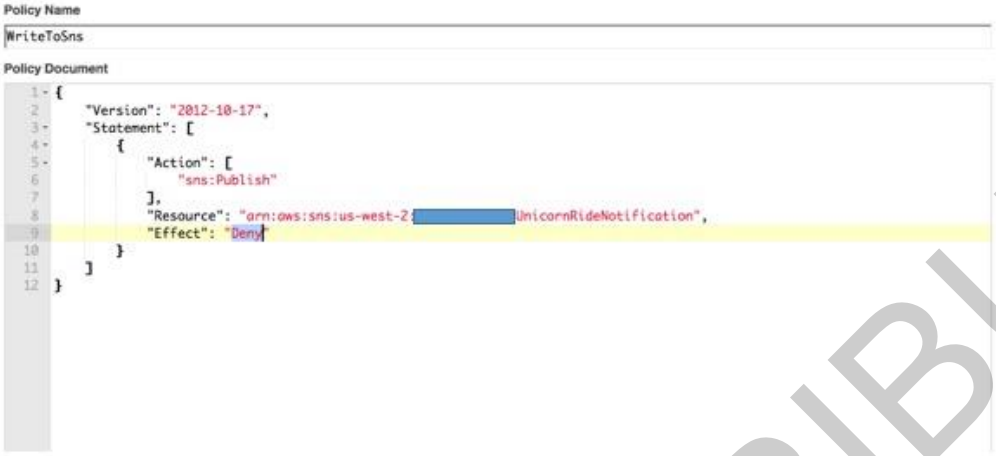
3.1.2 In the navigation pane, click **Roles**.

3.1.3 Click on the role named "**[CloudFormation Stack Name]-LambdaSnsRole-[Generated String]**".

3.1.4 On the **Permissions** tab, click on **Edit Policy** next to the Inline Policy named **WriteToSns**.



3.1.5 Change the **Effect** from "Deny" to "**Allow**".



3.1.6 Click **Validate Policy.**

3.1.7 Click **Apply Policy.**

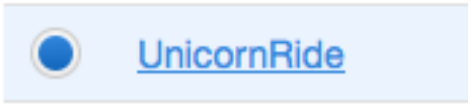
Task 3.2: Run a Test Message

This task walks you through running a test message through the system again.

3.2.1 On the **Services** menu, click on **DynamoDB**.

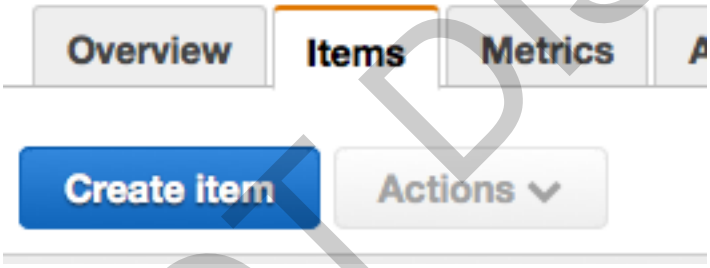
3.2.2 In the navigation pane, click **Tables.**

3.2.3 Click **UnicornRides**.

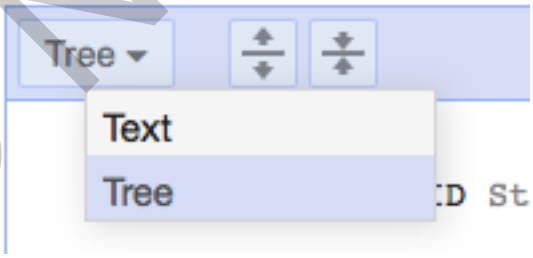


3.2.4 Click **Items**.

3.2.5 Click **Create item**.



3.2.6 In the **Create item** dialog box, for **Tree**, click **Text**.



3.2.7 Paste the following text to store as an item in DynamoDB and click **Save**.


```
{
  "DestinationAddress": "456 West Avenue",
  "DestinationLocation": {
    "Lat": 34,
    "Lon": 43
  },
  "DispatchedAt": "2016-06-06T13:27:14Z",
  "DroppedOffAt": "2016-06-04T13:27:14Z",
  "PickupAddress": "789 Rainbow St.",
  "PickupLocation": {
    "Lat": 24,
    "Lon": 54
  },
  "RequestedAt": "2016-06-04T13:27:14Z",
  "RequestID": "2983723",
  "UnicornId": 9000,
  "UserID": "unicornsAreCool"
}
```


3.2.8 You'll see the request added to the table.

3.2.9 This will automatically trigger an event on the stream and the lambda functions will execute. The batch size specified is a max value, the event will trigger even with the single message inserted into the DynamoDB table.

Task 3.3: View the Message in SQS

This task walks you through viewing your successfully sent message. Remember that SQS is just an example, and you could have different subscriptions set up for your SNS topic, such as email or SMS messages, etc.

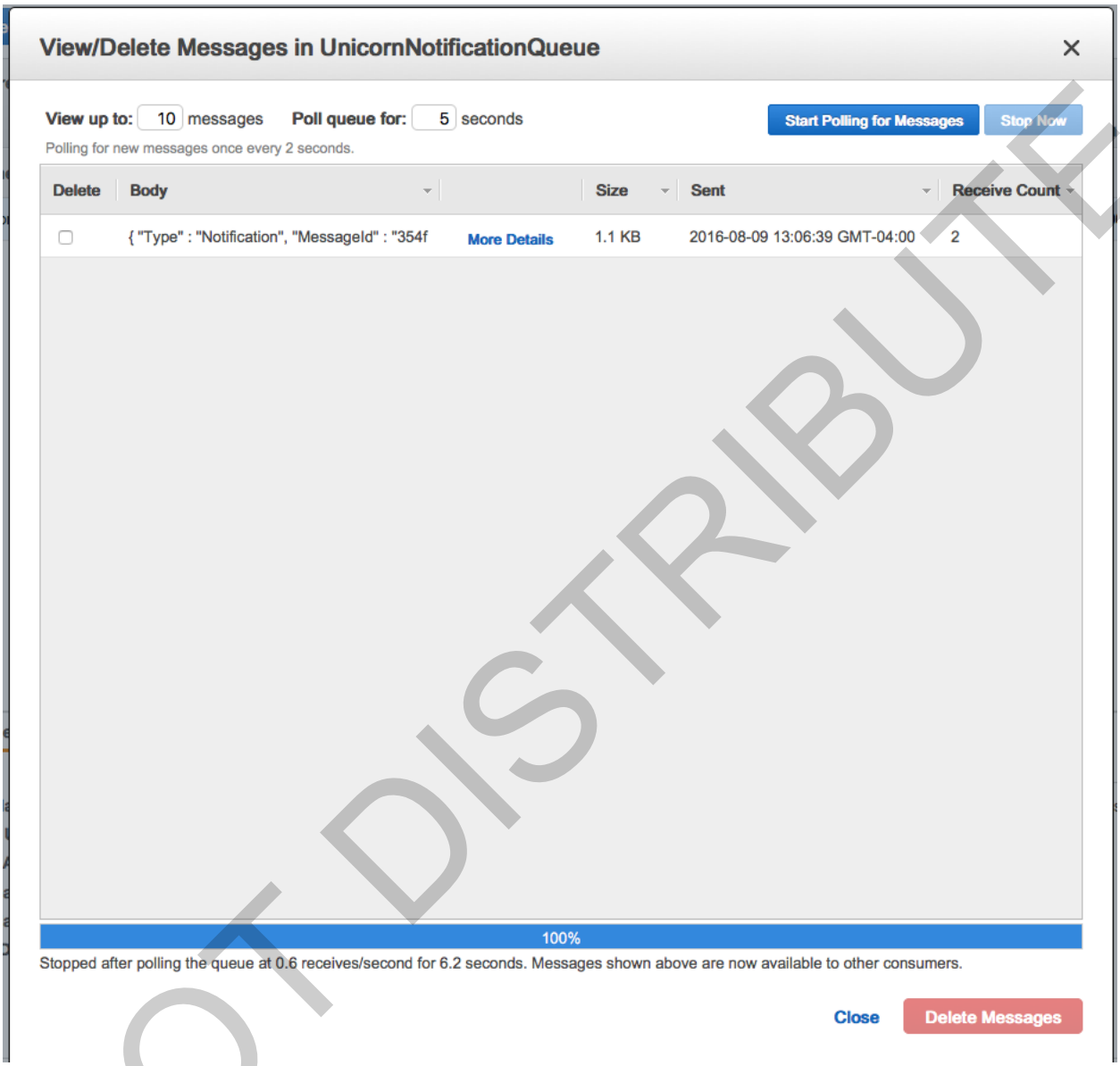
3.3.1 On the **Services** menu, click **SQS**.

3.3.2 Select **UnicornNotificationQueue.**

3.3.3 For **Queue Actions**, click **View/Delete Messages**.

3.3.4 Click **Start Polling for Messages**.

3.3.5 You should see your messages listed in the window.



Lab Complete

Congratulations! You have successfully completed Serverless Stream Processing:

1. Log out of the **AWS Management Console** by clicking **awsstudent** in the top-right corner and then click **Sign Out**.

2. Return to the **qwikLABS** page where you launched your lab and click **End**.

Lab 3
Scheduled Lambda and Batch Processing

Overview

In this lab session, you will orchestrate a group of scheduled lambda functions to create internal operations' metrics based on the state of the rides system. In the first part of the lab, you will create a pipeline using AWS Lambda for updating Amazon CloudWatch Metrics. You will then use Amazon Simple Notification Service (SNS) to create a broadcast channel for other Lambda functions and Amazon CloudWatch Metrics to persist custom metrics. Then, you will use Amazon Simple Email Service to send an email to administrators in the event of an unhealthy system. In the second section of the lab, you will create a set of batch-processing Lambda functions that are responsible for creating time-series indexes of rides data using Amazon ElasticSearch Service.

Objectives

After completing this lab, you will be able to:

- Task 1 – Configure SNS as an Event Source for Lambda
    - 1.1 Retrieve SNS topic for administrator system alerts
    - 1.2 Create Lambda Function with SNS Event Source
    - 1.3 Manually trigger an unsuccessful message
- Task 2 – Configure SES Email Notifications with Lambda
    - 2.1 Configure SES Lambda Function
    - 2.2 Manually trigger an SES email
- Task 3 – Enable Scheduled Operational Lambda Function
    - 3.1 Enable a scheduled Lambda Function in the Console
    - 3.2 View the Scheduled Lambda Function Results
    - 3.3 Experiment with changing successful and error threshold metrics
- Task 4 – Configure Time Series Lambda Functions
    - 4.1 Configure Scheduled ElasticSearch Rides Data
    - 4.2 View and Test Time Series Worker
    - 4.3 Create Coordinator Lambda Function

Prerequisites

This lab requires the following:

- Access to a computer with Wi-Fi, running Microsoft Windows, Mac OS X, or Linux (Ubuntu, SuSE, or Red Hat). The qwikLABS lab environment is not accessible using an iPad or tablet device, but you can use these devices to access the student guide.
- An Internet browser such as Chrome, Firefox, or IE9 or later (previous versions of Internet Explorer are not supported).

Duration

This lab will take approximately **45 minutes**.

Task 1: Configuring SNS as an Event Source for Lambda

Overview

In this section, you will create the SNS workflow for sending error alerts to a Lambda Function via SNS. In later tasks, you'll use a similar broadcast pattern to send an email of the health of the system.

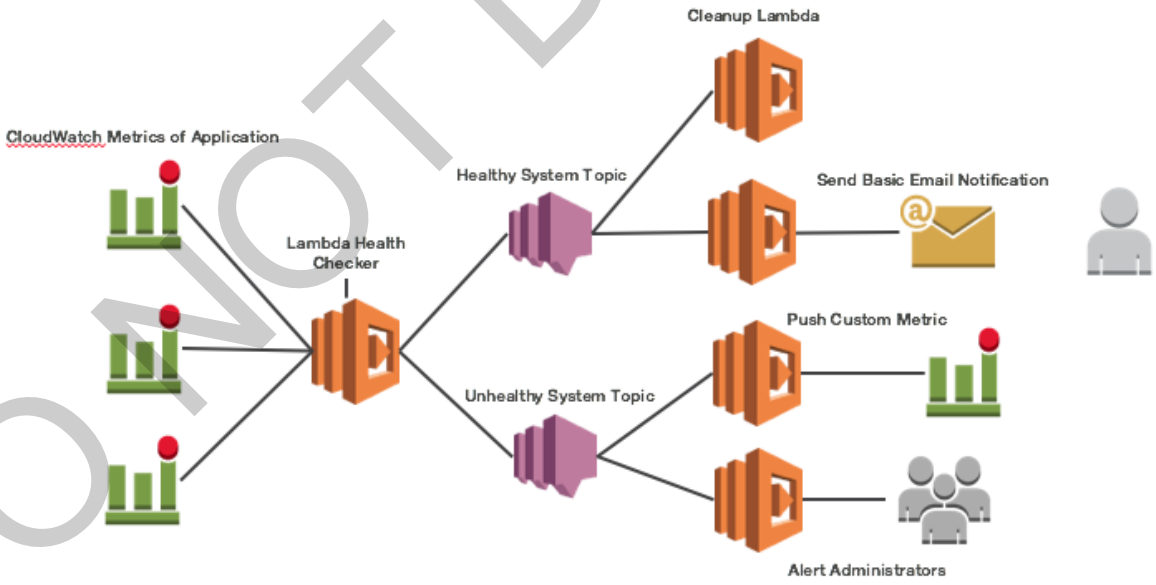
Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

Scenario

In this lab, you will build the following infrastructure:



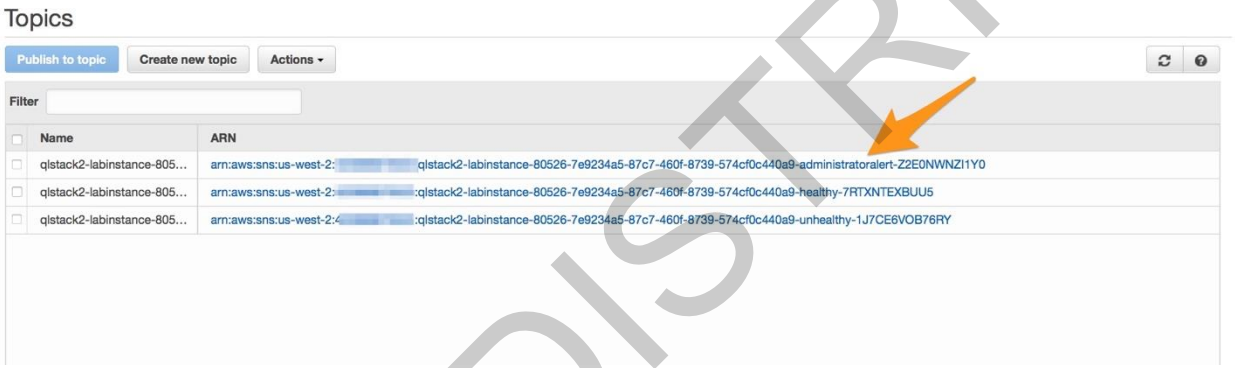
Task 1.1: Retrieve SNS topic for administrator system alerts

In this task, you will trigger an SNS topic for your operational Lambda Functions.

1.1.1 In the **AWS Management Console**, on the **Services** menu, click **Mobile Services→SNS**.

1.1.2 On the navigation pane, click **Topics**.

1.1.3 Copy the ARN for the administrator alert topic, which will be used later on in the lab, into a text editor. The topic ARN will have a similar format to the below and will have a topic that contains the words "**administrator**" as part of the SNS topic name.



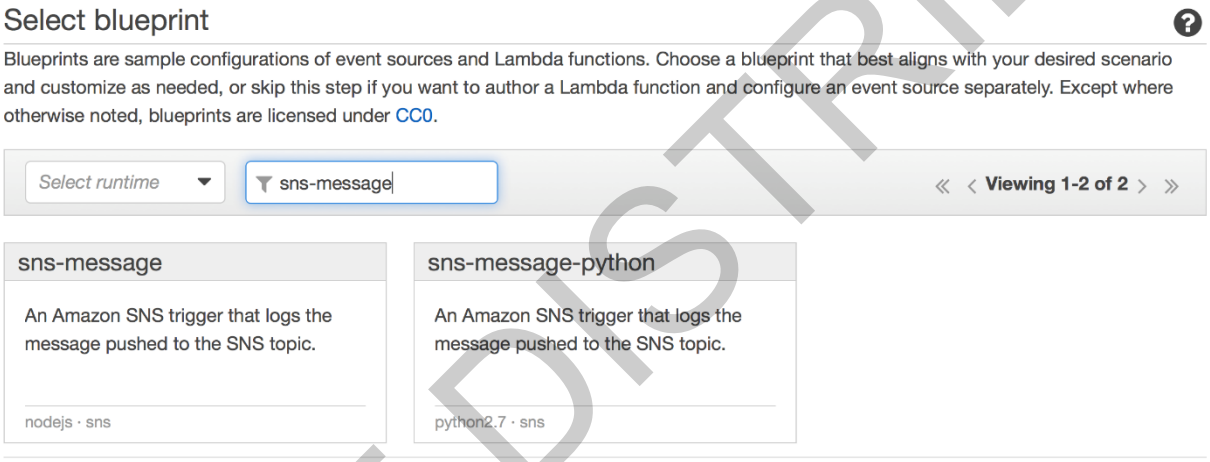
Task 1.2: Create Administrator Lambda Function with SNS Event Source

In this task, you will create a sample Lambda function that will be associated with your SNS topic. This will be the first step in automating and chaining together your Lambda functions.

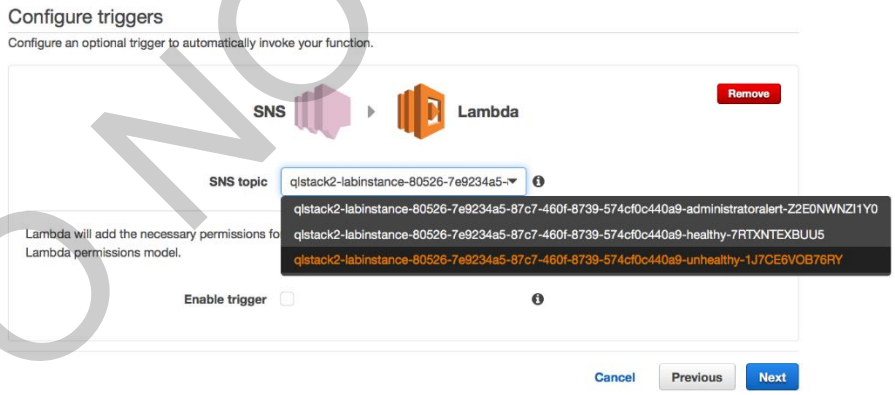
1.2.1 In the **AWS Management Console**, on the **Services** menu, click **Compute→Lambda**.

1.2.2 On the navigation pane, click **Functions**.

1.2.3 Click **Create a Lambda Function** and enter “sns-message” in the filter textbox for blueprints. This search will return two functions. Click “sns-message” blueprint with nodejs as the language.

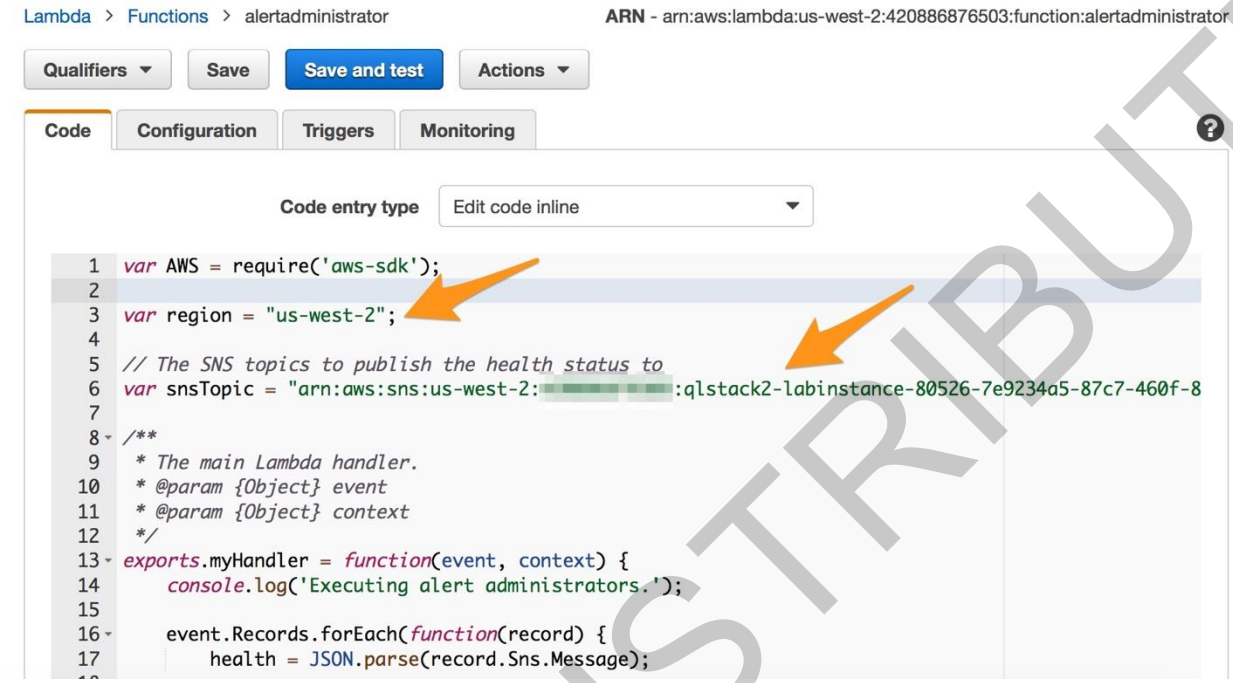


1.2.4 For the **Configure triggers** page, for SNS topic, click the topic arn that contains the word “unhealthy” and then select **Enable Trigger**. Click **Next**.



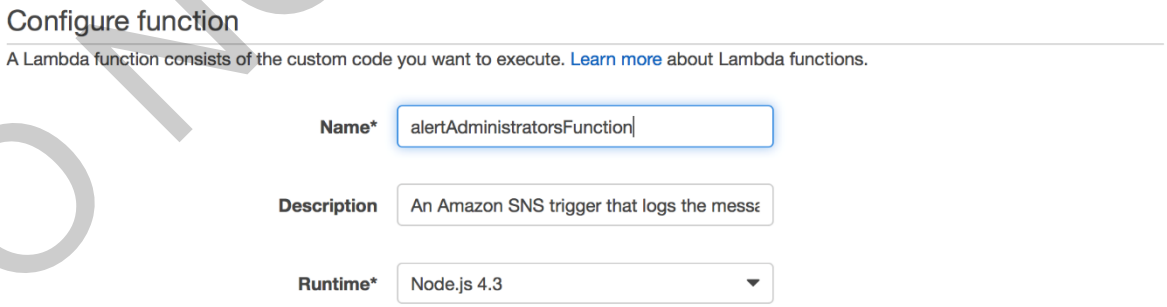
1.2.5 On the **Configure function** page, create your notification Lambda Function by copying and pasting the alertadministrator.js code from the reference file into the inline editor.

Enter in two variables. The first variable is "**region**", which you will populate with the region that is being used for the lab. The second variable is in the lambda code, which you will paste in your "**alert administrator**" SNS topic ARN.



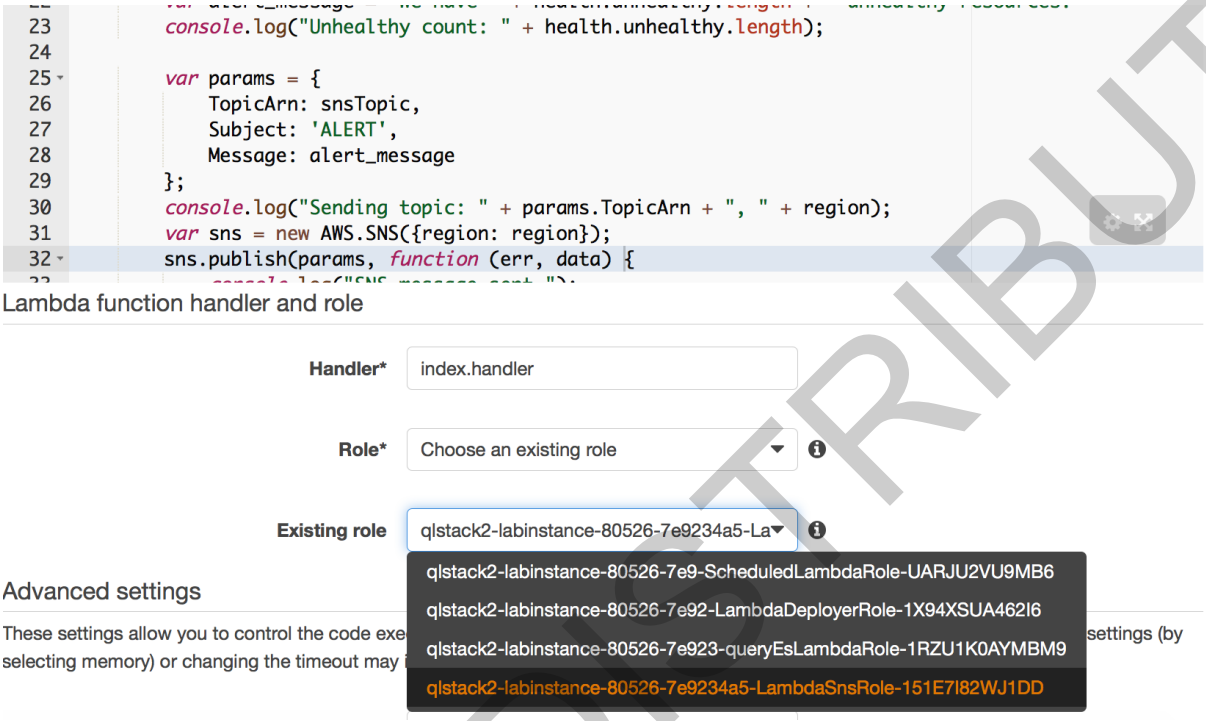
This Lambda function code is part of a broadcast pattern. It is one of potentially many listeners that are subscribed to error notifications that are being generated by your operational system. The Lambda Function code looks for errors and then passes a message to a downstream SNS topic to alert administrators.

1.2.6 On the **Configure function** page, for **Name**, type "alertAdministratorsFunction".:



1.2.7 In the **Lambda function handler and role** section, for **Role**, click **Choose an existing role**, and for **Existing role**, click the role containing **LambdaSnsRole**, which has been

preconfigured with IAM permissions for this lab. Leave the memory settings but increase the timeout to 15 seconds. Click **Next** and click **Create function** to create a new Lambda function.



Task 1.3: Trigger an Unsuccessful Message Manually

In this task, you will test your SNS-triggered Lambda Function by invoking a message from SNS to Lambda and viewing the logs for Lambda.

1.3.1 In the **AWS Management Console**, select the **Services** menu, then select **Mobile Services→SNS**.

1.3.2 In the navigation pane, click **Topics**.

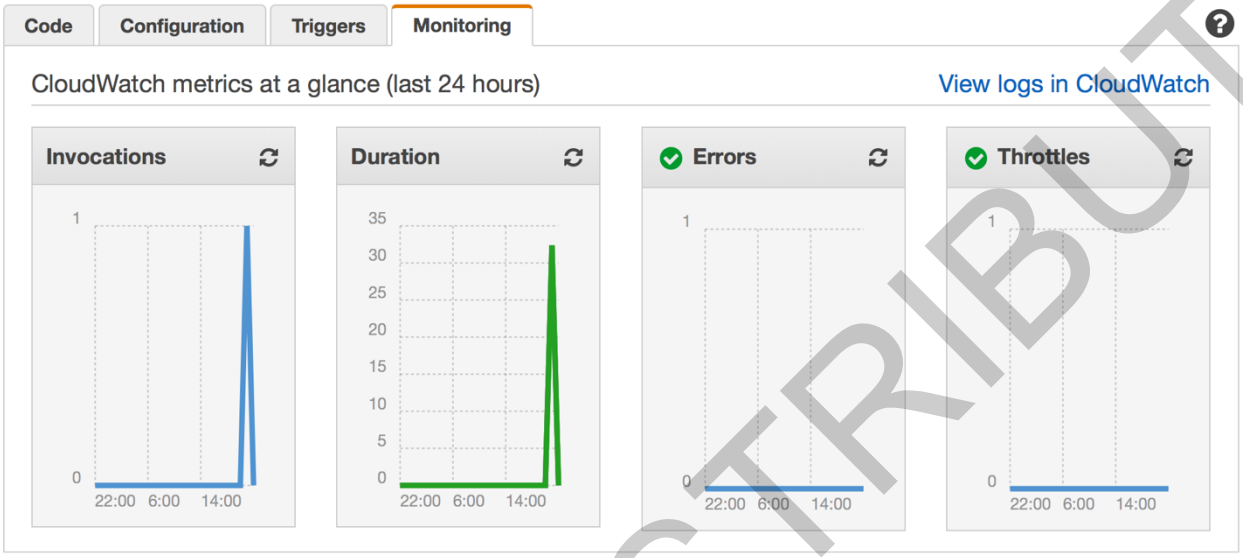
1.3.3 Click the unhealthy SNS topic and then click **Publish to topic**.

1.3.4 Enter a JSON Payload that mirrors the syntax used for operations. There's a sample JSON payload in the command reference text file. Then, click **Publish Message**.

1.3.5 To confirm that the SNS Message triggered your Lambda Function, in the **AWS Management Console**, on the **Services** menu, click **Compute→Lambda**.

1.3.6 In the navigation pane, click **Functions**, select **alertAdministratorsFunction** from the list of available Lambda Functions.

1.3.7 While viewing the alertAdministratorsFunction, click the **Monitoring** tab and confirm that the Lambda Function has now been triggered one time in the CloudWatch Metrics.



Task 2: Configuring SES Email Notifications with Lambda

Overview

So, now you've configured SNS as a trigger for an alert Lambda Function that will message all users that are subscribed to the alert channel. You've also created your first pipeline for any unhealthy events that are created from the rides system. In addition to monitoring and alerting on unhealthy messages that will be created from our operational Lambda Functions, we also want to create an email notification that sends a message whenever the system has information to share based on status. In this task, you'll create a Simple Email Service (SES) configuration using Lambda to send emails to yourself, whenever the system calculates its health. This Lambda Function will also be triggered via SNS to continue to enhance the pipeline and broadcast Lambda patterns.

Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

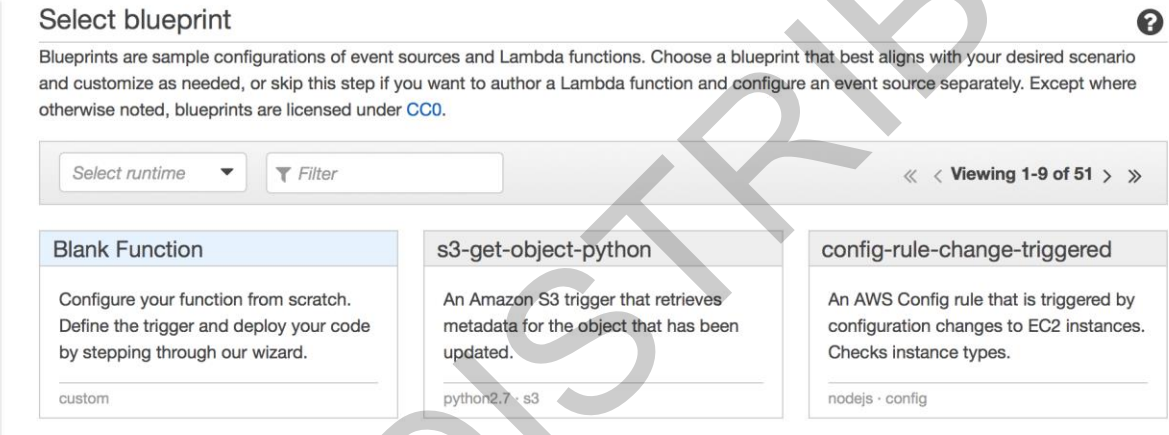
Task 2.1: Configure SES Lambda Function

This task walks you through setting up SES and SNS for triggering a Lambda Function and sending an email.

2.1.1 In the **AWS Management Console**, on the **Services** menu, click **Compute→Lambda**.

2.1.2 In the navigation pane, click **Functions**.

2.1.3 Click **Create a Lambda Function** and then click **Blank Function**. You will add the event sources in a later part of this task.

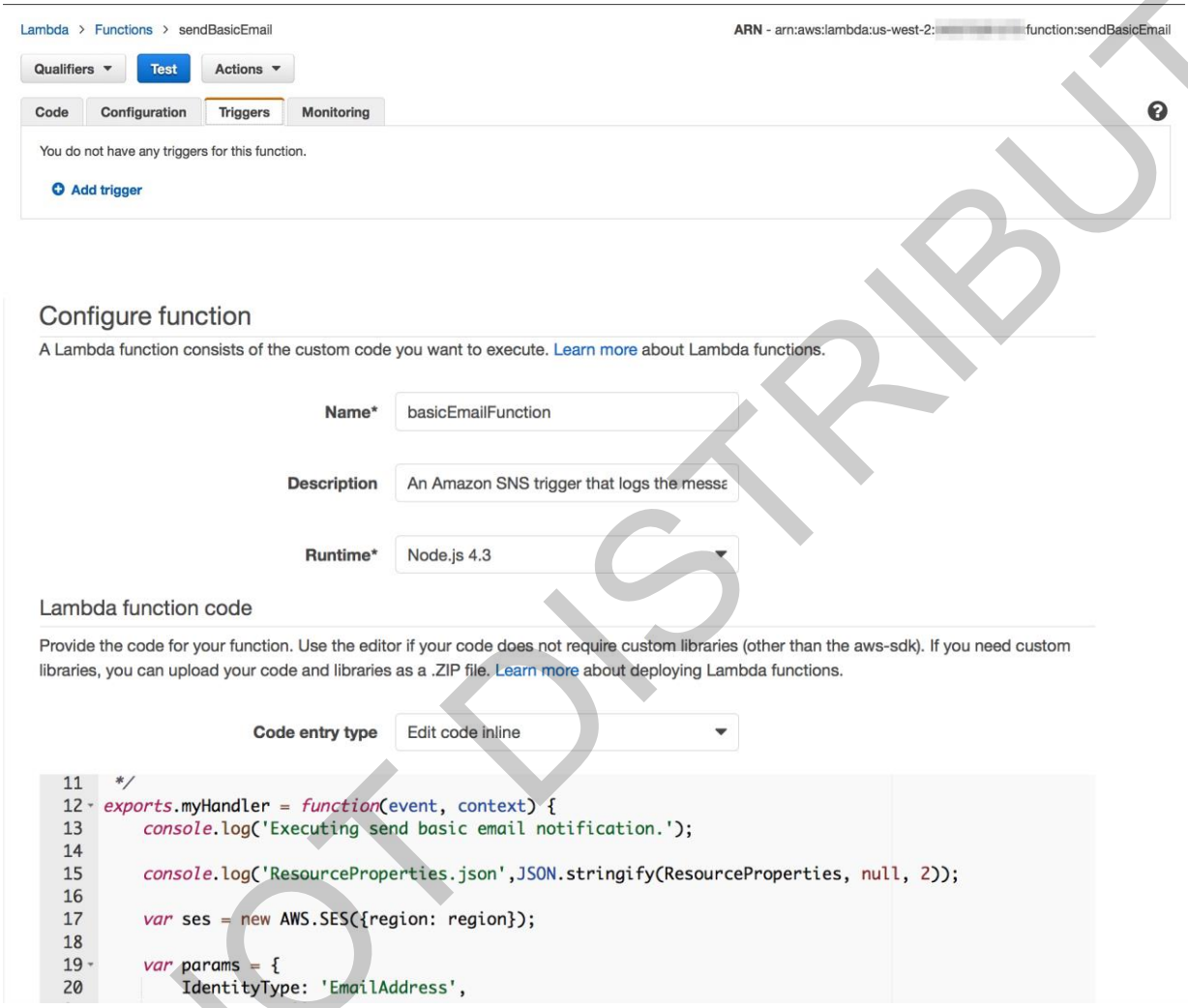


2.1.4 On the **Configure triggers** page, click **Next**.

2.1.5 On the **Configure function** page, you will create your notification Lambda Function by copying and pasting the basicemailnotification.js code from the reference file into the inline editor.

Note You will then enter one variable in inline editor, i.e., the "**region**" variable that you will populate with the region that is being used for the lab.

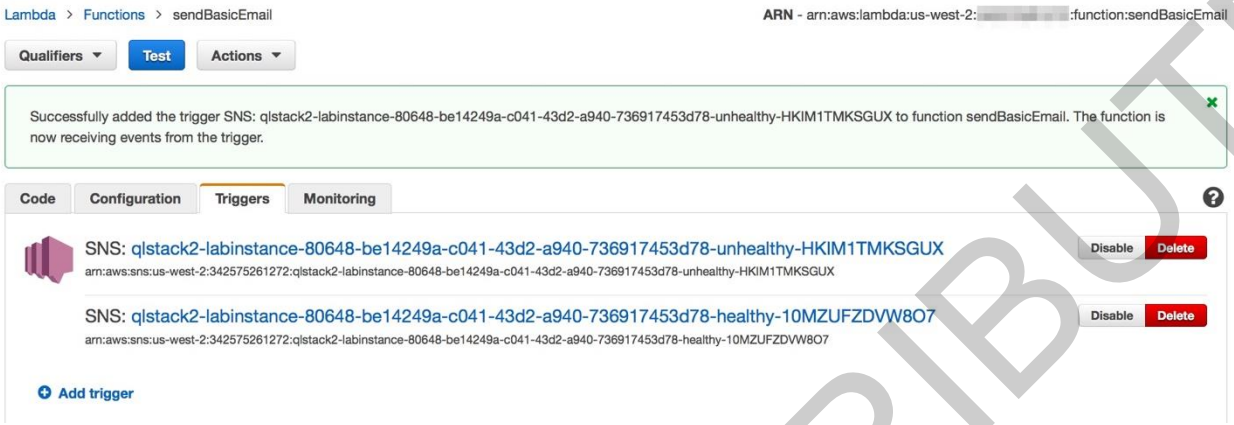
2.1.6 On the **Configure function** page, for **Name**, enter "**basicEmailFunction**" for the name of your Lambda function.



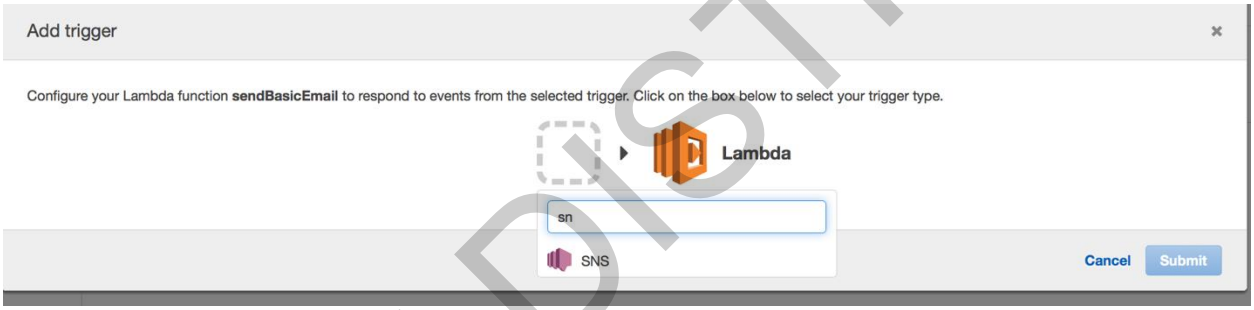
2.1.7 In the **Lambda function handler and role** section for Lambda, for **Role**, click **Choose an existing role**, and for **Existing role**, click the role containing the words **emailRole** in the ARN. Then, click **Next** and then click **Create function** button to create a new Lambda function.

2.1.8 Next, you will configure both unhealthy and healthy notifications to trigger your Lambda function to execute. To achieve this, click **Functions** in the navigation pane on the left. Then, select the **basicEmailFunction** that you created and click the "**Triggers**" tab.

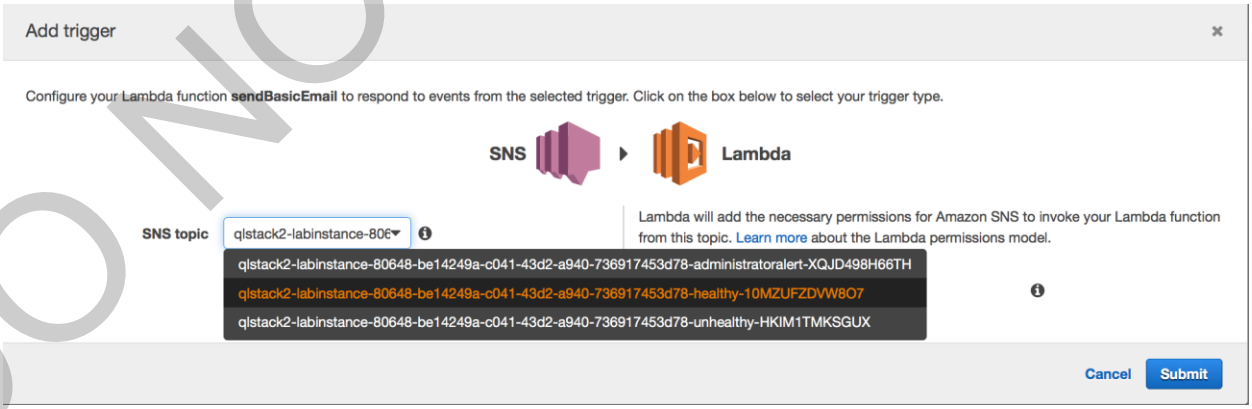
2.1.9 Then, click "**Add trigger**" and then search/select SNS in the dropdown.



2.1.10 For **SNS topic**, click the topic containing word "**healthy**" metrics and then click **Submit**.



2.1.11 You'll then follow the same steps again to add the "**unhealthy**" sns topic. After adding two event sources, your triggers section should resemble the below.



Task 2.2: Trigger an SES Email Manually

This task walks you through triggering an SES email from your Lambda function.

2.2.1 In the **AWS Management Console**, on the **Services** menu, click **Application Services→SES**.

2.2.2 In the navigation pane, under **Identity Management,** click **Email Addresses**.

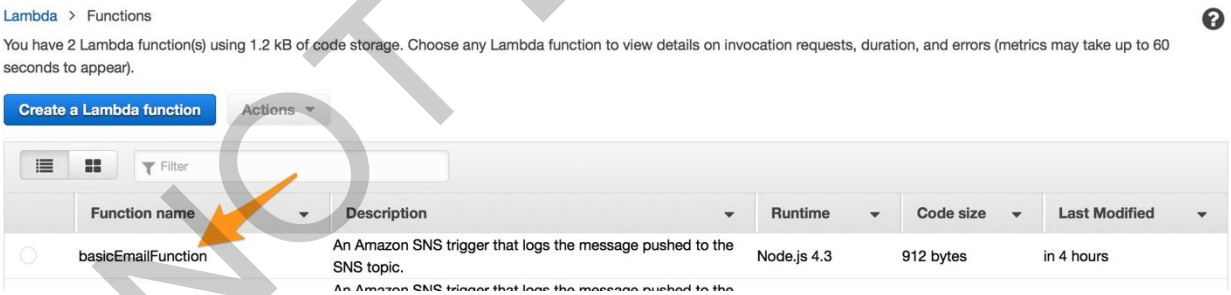
2.2.3 Click **Verify a New Email Address,** enter an email address to retrieve a status report, and click **Verify This Email Address**.

2.2.4 In your email client, open the email message from Amazon SES asking you to confirm that you are the owner of this email address. Click the link in the email message.

2.2.5 The status of the email address in the Amazon SES console will change from "*pending* verification" to "*verified*".

2.2.6 Now that we've configured a test email that will be used for sending a status report, you can test if your basic email Lambda function can correctly configure and send an operational email for the rides service. In the **AWS Management Console**, on the **Services** menu, select **Compute→Lambda.**

2.2.7 In the navigation pane, click **Functions** and then click the **basicEmailFunction** from the Amazon Lambda Function list. This Lambda Function is already linked to the email you created earlier.



2.2.8 Click **Test** and then from the Input test event modal, for **Sample event template**, click **SNS**. For a sample input that we will next configure to send automatically, copy and paste a sample operational message from the reference file:

```
{
        "Records": [
        { "EventVersion": "1.0",
```



```
        "EventSubscriptionArn": "arn:aws:sns:EXAMPLE",
        "EventSource": "aws:sns",
        "Sns": { "Message": "{\"status\":\"healthy\",\"healthy\":[\"healthyFunction1\"]}",
                "Subject": "Healthy"
        }
      }
    ]
}
```

Then, click **Save and test** at the bottom of the modal.

2.2.9 You will shortly receive emails from SES that have been sent from your SNS triggered Lambda function. As you inspect the email and the Lambda function, you'll notice that this Lambda function is decoupled from the input message (SNS) and the output through SES. This demonstrates how you can leverage Lambda functions to work in parallel when events are triggered in your workflow.

Task 3: Enabling Scheduled Operational Lambda Functions

Overview

During the initial lab, several Lambda Functions were created within your AWS account. In addition to these lambda functions, through the first few sections you configured several additional downstream Lambda functions that created an end-to-end workflow. In this task, you will enable one of the Lambda Functions that was provisioned at the start of the lab. This Lambda function is responsible for periodically checking the metrics of the rides system and will publish to the other SNS topics like the healthy and unhealthy topics that you've been working on throughout the first two tasks. During this lab, you'll start working with scheduled Lambda Functions by first enabling a scheduled function and then viewing the logs of your Lambda Function.

Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

Task 3.1: Enable a scheduled lambda function in the console

This task walks you through enabling the primary Lambda function that is responsible for scheduling the operational system checks for the rides healthchecker.

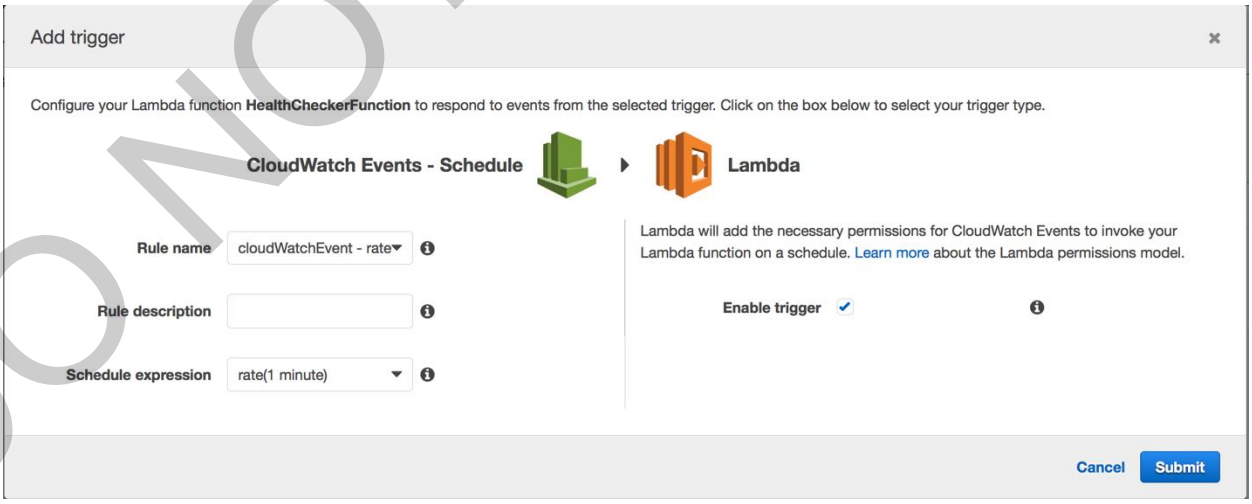
3.1.1 In the **AWS Management Console**, select the **Services** menu, then select **Compute→Lambda**.

3.1.2 In the navigation pane, click **Functions**.

3.1.3 Click the Lambda function containing the words "**HealthChecker**" from the list of available Functions. The HealthChecker function is pre-configured to read metrics from the rides service, including elasticSearch metrics and other Lambda Function metrics.

3.1.4 To enable a scheduled Lambda Function, you need to create a CloudWatch event that executes on a given interval. Next, click the **Triggers** tab and click **Add trigger.**

3.1.5 When you click **Add Trigger**, a modal will appear. Select **CloudWatch Events – Schedule** from the drop-down list, for **Rule name**, type "**cloudWatchEvent**", for **schedule expression**, click "**rate(1 minute)**, then select "**Enable Trigger**" and then click **Submit**.



3.1.6 When this function is enabled, the HealthChecker will automatically trigger every minute, then will publish messages downstream to your SNS topics.

Task 3.2: View the scheduled lambda function results

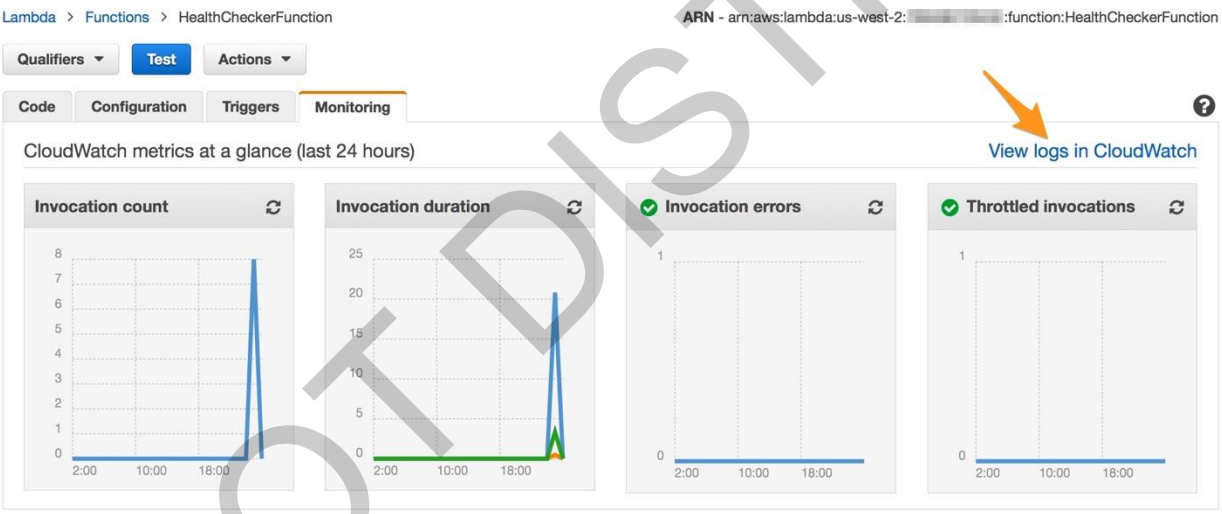
This task walks you through verifying the scheduled Lambda functions via CloudWatch Logs and reviewing your SES emails.

3.2.1 In the **AWS Management Console**, on the **Services** menu, click **Compute→Lambda**.

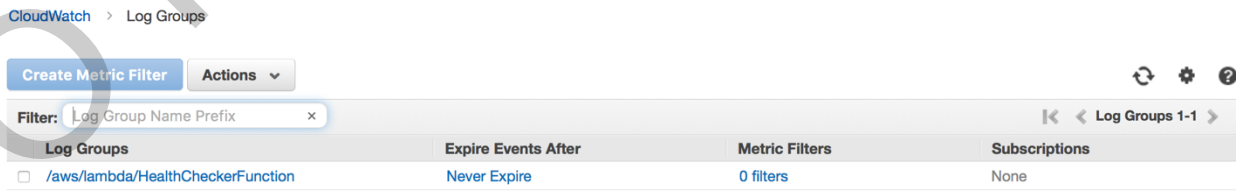
3.2.2 In the navigation pane, click **Functions**.

3.2.3 Click the Lambda function containing the words "**HealthChecker**" from the list of available Functions.

3.2.4 Click on the last tab for **Monitoring**. This will display the logs and a link to the CloudWatch Logs. Click on the **View Logs in CloudWatch** in the upper-right corner to view the logs from the latest Lambda run. This will navigate you to the CloudWatch Logs console.

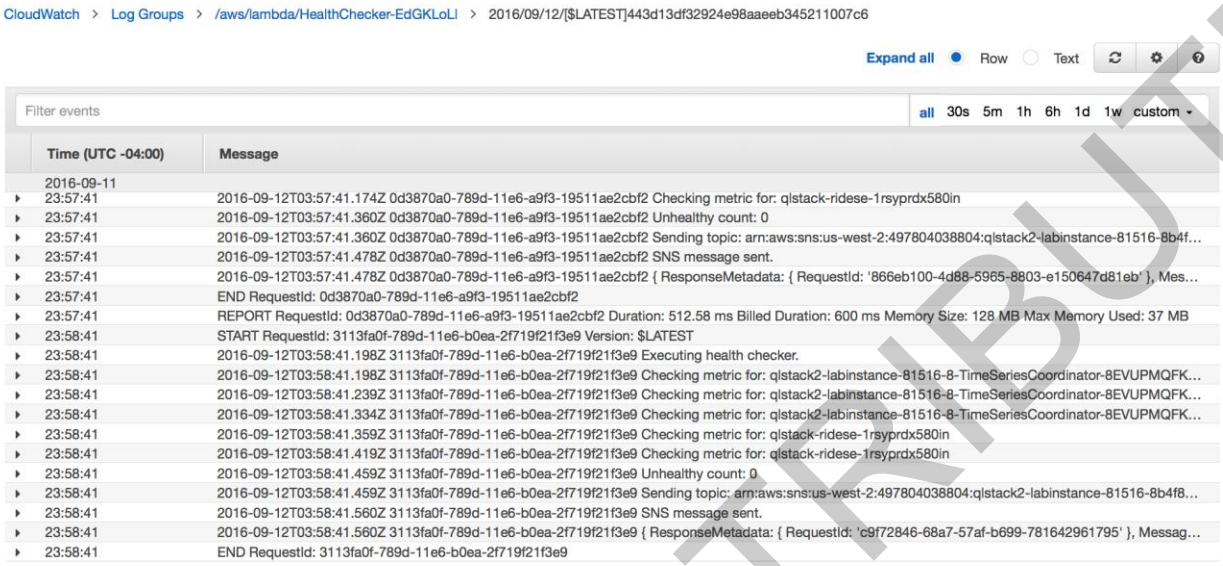


3.2.5 In the Cloudwatch Logs, select the latest CloudWatch Log Group to view the ongoing metrics from this specific Lambda function.



3.2.6 In the logs for the operational health checker, you will see log messages that show the health checker executing. The Lambda function is tallying the number of healthy and

unhealthy resources in the application, and sending messages to specific SNS topics that the HealthChecker has pre-configured. One of the SNS topics is associated with the SES email function that you configured in the previous lab.



3.2.7 You have now configured a pipeline, by first creating the downstream Lambda Functions triggered by SNS then connecting additional notifications and communications using new SNS topics and SES emails. And then, you enabled a scheduled Lambda Function that triggers the entire end-to-end pipeline.

3.2.8 In addition to the HealthChecker logs, spend time reviewing the additional Lambda Function logs that are executed as part of the operational pipeline.

Task 4: Configuring Time Series Lambda Functions

Overview

Having configured internal operational and system metrics for the rides system, you are now ready to start expanding the operational metrics for rides by creating additional data that analyses time series information about rides. In this task, you will configure a time series Lambda worker function that will look at timestamps over a period of time and store them in a time series index in ElasticSearch. After configuring the individual time indexes in a single function, you will create a second Lambda pipeline by creating a Lambda Coordinator Function that will handle synchronously calling the time series function. After creating the coordinator, you will view the output of the time series data using Kibana.

Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

Task 4.1: Configure Scheduled ElasticSearch Rides Data

This task walks you through creating a Lambda function that simulates ride data throughout the course of the day and sends it to the ElasticSearch Service. You will configure the function to trigger on a schedule via a cron style syntax. Each time the schedule invokes the Lambda function, it will generate ride data and index it in ElasticSearch.

4.1.1 In order to test the ElasticSearch time series function, you first need to retrieve the preconfigured ElasticSearch Domain that has been created for the rides system. In the **AWS Management Console**, on the **Services** menu, click **Analytics→ElasticSearch Service**.

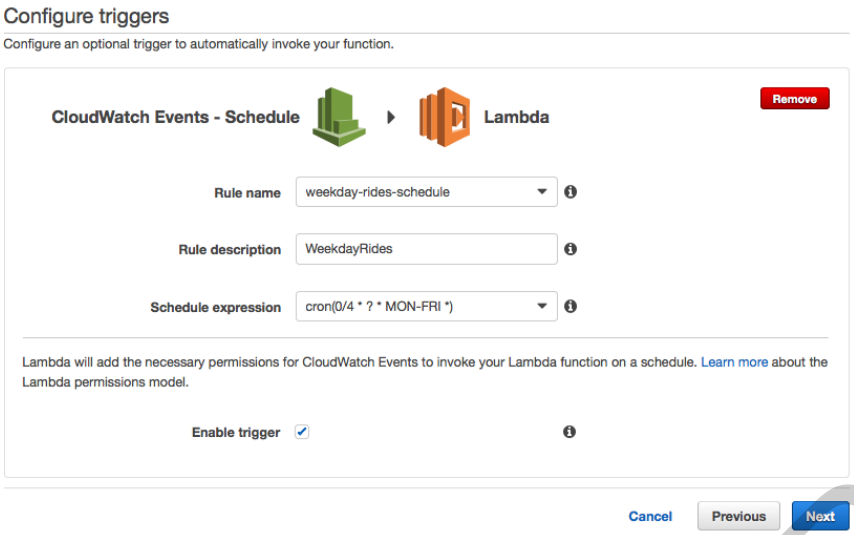
4.1.2 From the ElasticSearch Service console, click the available search domain, then copy and paste the elastic search **endpoint** url. The elastic search endpoint url will be used in the next step as input for our time series worker.

4.1.3 In the **AWS Management Console**, on the **Services** menu, click select **Compute→Lambda**.

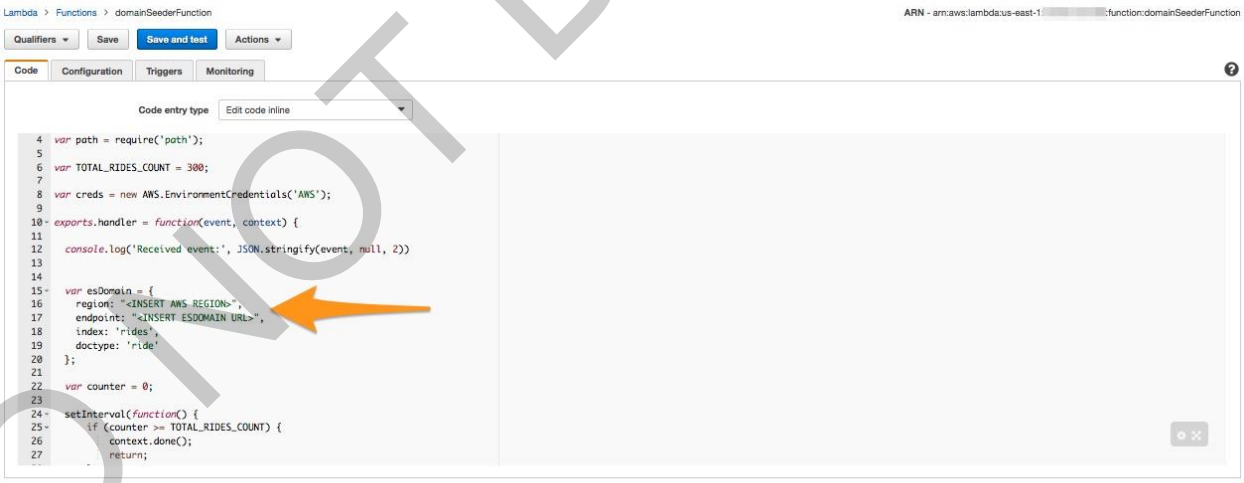
4.1.4 In the navigation pane, click **Functions**.

4.1.5 Click **Create a Lambda Function** and click **Blank Function**.

4.1.6 Next, you will create a scheduled event that will trigger your Lambda Function every 4 minutes between Monday through Friday. For the rides scheduling, unicorns will send data only during the weekdays, so you will configure your schedule to use a cron style syntax to only push new rides every 4 minutes. Click **CloudWatch Events - Schedule** from the drop-down list, for **Rule name**, type **weekday-rides-schedule**, for **schedule expression,** type **cron(0/4 * ? * MON-FRI *)**, select **Enable trigger** and then click **Next.**



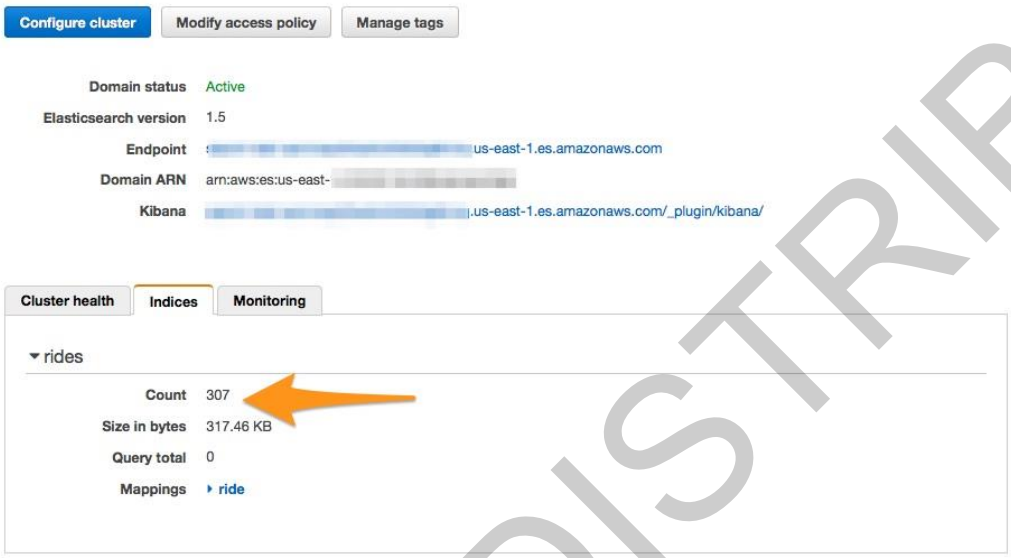
4.1.7 On the **Configure function** page, you will create your notification Lambda Function by copying and pasting the **SampleRidesFunction.js** code from the reference file into the inline editor. You will then look for two variables in the source code named "**region**" and "**endpoint**". You will replace those values with the region your AWS lab is currently running in as well as the elasticsearch endpoint you copied from step 4.1.2.



4.1.8 On the **Configure function** page, for **Name**, type "rideSimulationFunction". In the **Lambda function handler and role** section for **Role**, click **Choose an existing role**, and then for **Existing role**, click the role that contains the words

ScheduledLambdaRole in the ARN. Then, for **Timeout** set this to **5 min**. Click **Next** and click **Create function** to create a new Lambda function.

4.1.9 After several minutes, you can confirm that your rides are being indexed by navigating back to the ElasticSearch console. In the **AWS Management Console**, on the **Services** menu, click **Analytics→ElasticSearch Service.** From the ElasticSearch Service console, select the ElasticSearch Domain for the rides, then click the **Indices** tab. You should see an index labeled “rides” and there should be a few hundred rides in it. This will continue to increase as the function continues to get triggered on the schedule.



Task 4.2: View and Test Time Series Worker

This task walks you through enabling the primary Lambda function that is responsible for

4.2.1 In order to test the ElasticSearch time series function, you first need to retrieve the preconfigured ElasticSearch Domain that has been created for the rides system. In the **AWS Management Console**, select the **Services** menu, then select **Analytics→ElasticSearch Service.**

4.2.2 From the ElasticSearch Service console, click the Search domain, then copy and paste the elastic search **endpoint** url to a text editor. The elasticsearch endpoint url will be used in the next step as input for the time series worker.

4.2.3 In the **AWS Management Console**, on the **Services** menu, select **Compute→Lambda**.

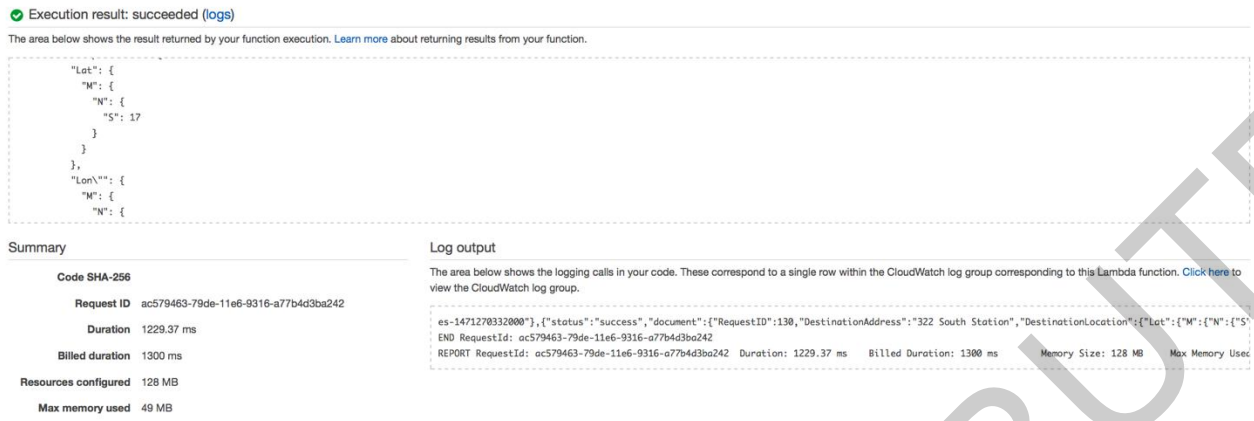
4.2.4 In the navigation pane, click **Functions**.

4.2.5 Click the Lambda function containing the name "**TimeSeriesWorker**" from the list of available functions. The TimeSeriesWorker function is pre-configured to take a start and end date. Now, search for results from the rides table between the two indexes. After finding search results in the main index, the TimeSeriesWorker will create new time series indexes in elastic search.

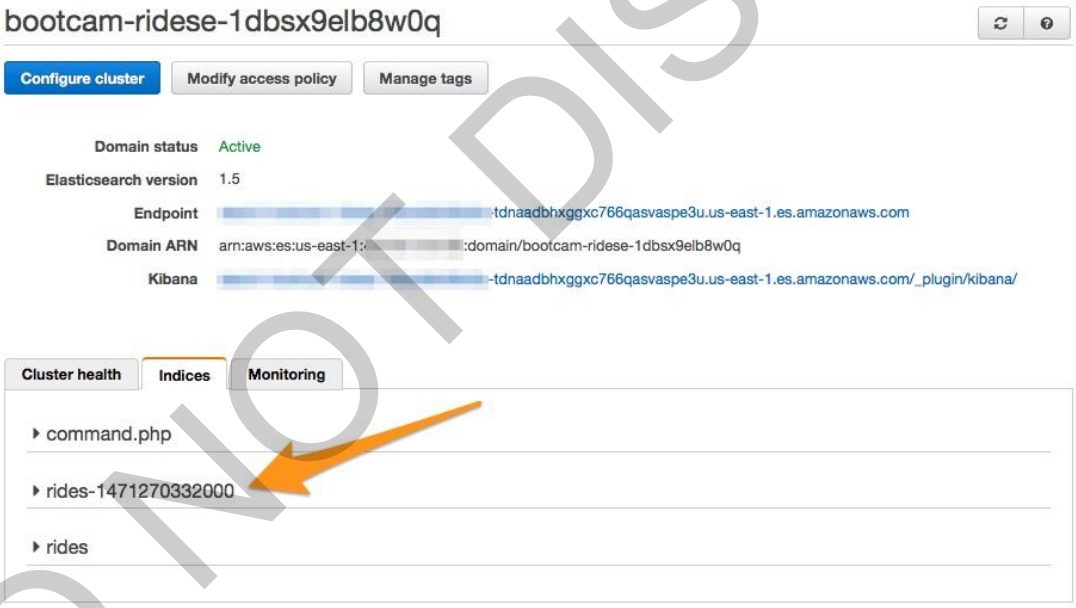
4.2.6 To test the worker functionality, click **Test** and enter in a json document similar to what is given below, which is also referenced in the command reference file for the lab. Replace the region attribute and the endpoint attribute with the region your lab is deployed with, and the endpoint, and then click **Save and test**.

```
{
  "region": "<INSERT AWS REGION>",
  "endpoint": "https://<INSERT ESDOMAIN URL>/",
  "index": "rides",
  "doctype": "ride",
  "startTime":"2016-08-15T14:12:12",
  "endTime":"2016-08-24T14:12:12",
  "searchResultSize": 10
}
```

Now that you've run the lambda function, scroll down to the bottom of the console to view the output from the running function. You should see two messages that log the index created and the number of records processed.



4.2.7 To confirm that your lambda function is now querying and creating time series data in Elasticsearch, you will validate that your Elasticsearch index now has more indexes created with unique timestamps. In the **AWS Management Console**, on the **Services** menu, select **Analytics→ElasticSearch Service.** From the ElasticSearch Service console, select the ElasticSearch Domain for the rides, click the **Indices** tab. You should see a new index labeled "*rides-<timestamp>*".



4.2.8 Now that you've configured a running Lambda function that creates new records for searching ride information, you will use the next task to build a pipeline in order to scale your search requests across more rides documents.

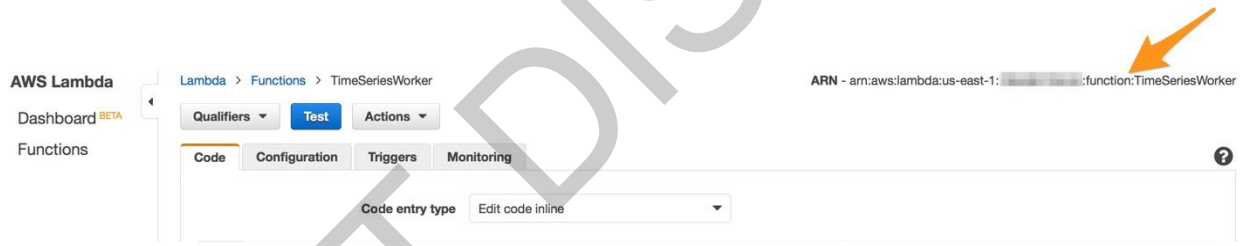
Task 4.3: View and Test Time Series Coordinator

In the second task, you invoked a single Lambda function that could search against a subset of the entire rides information and then create a time series based index. Using Lambda, you can create a pipeline that connects multiple Lambda functions together in a broadcast pattern. In order to support larger search queries, you will have one Lambda function, the coordinator, orchestrate the execution of other worker lambda functions. In this next task, you will create a coordinator Lambda function. This coordinator function will synchronously invoke the Lambda function you created in task 4.2. This approach demonstrates the pipeline approach of one Lambda function directly invoking other lambda functions.

4.3.1 In the **AWS Management Console**, select the **Services** menu then select **Compute→Lambda**.

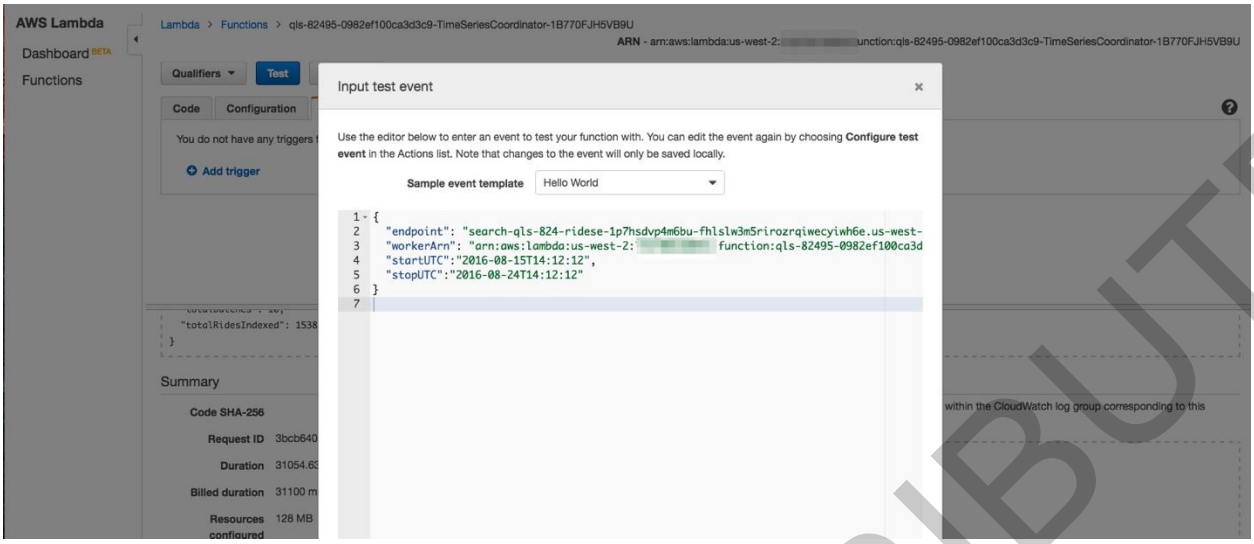
4.3.2 In the navigation pane, click **Functions**.

4.3.3 Select the Lambda function named TimeSeriesWorker from the list of available Functions and copy and paste the full function ARN that is located in the top right of the AWS Console to a text file. You'll use this ARN as an input variable when creating your coordinator Lambda Function.



4.3.4 Click the Lambda function containing the name "TimeSeriesCoordinator" from the list of available functions. The TimeSeriesCoordinator function is pre-configured to take a start and end date, and then split the time into multiple time ranges. After the coordinator has created an array of time periods, the coordinator function invokes multiple TimeSeriesWorker functions that perform the majority of the batch processing, working on only their batch of work and reporting back to the coordinator.

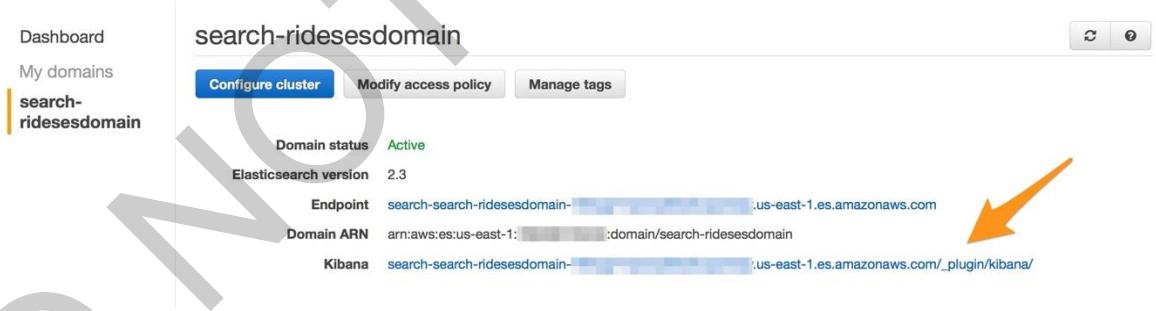
4.3.5 To test the coordinator functionality, click **Test** and enter in a json document similar to the one given below, which is also referenced in the command reference file for the lab. Replace the region attribute and the endpoint attribute with the region your lab is deployed with, and the endpoint.



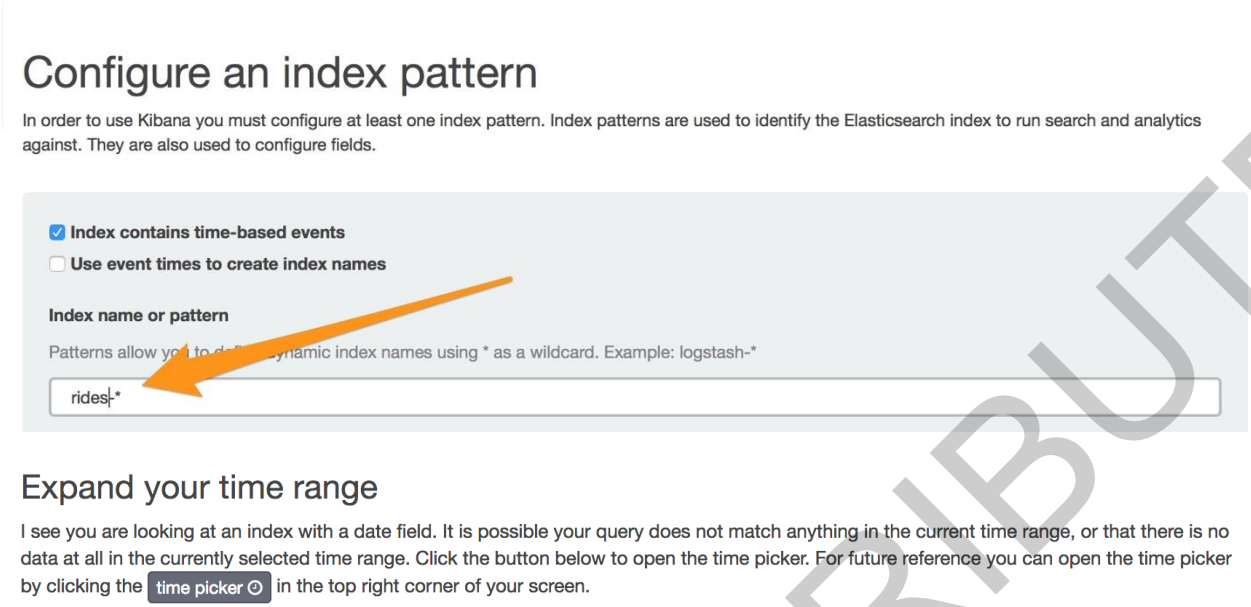
For the startUTC and stopUTC, enter a month time range that corresponds to the previous month. For this lab, use November 1st through November 29th. UTC months are represented as 0 – 11. For **startUTC** input “2016-10-01T00:00:00” and for **stopUTC** input “2016-10-29T23:59:59”. Then**,** click **Save and test** at the bottom of the modal.

4.3.6 After the TimeSeriesCoordinator executes, you should see a string of output in the Lambda console.

4.3.7 Before completing the lab, you can explore the individual records in kibana UI. In the AWS Management Console, select the **Services** menu, then select **Analytics→ElasticSearch Service**. Select the Kibana UI link in the console and wait for the index to build.



4.3.8 On the “Configure an index pattern” page that you see at the start, enter “rides-*” as the “Index name or pattern” and select “RideDate” under “Time-field name”. Kibana will then list the fields found under the “rides” index. After a few minutes, rides will become visible for search on the “Discover” tab. Click the grey button titled “time picker” in the “Expand your time range” section as shown below. In the time filter dropdown, select “Last 1 year” to display rides that were requested within the last year.



Lab Complete

Congratulations! You have successfully completed Serverless Scheduled and Batch Processing Lambda.

1. Log out of the **AWS Management Console** by clicking **awsstudent** in the top-right corner and click **Sign Out**.

2. Return to the **qwikLABS** page where you launched your lab and click **End**.

Lab 4
VPC and Hybrid Architectures

Overview

In this lab session, you will become familiar with how to configure AWS Lambda and the relevant Amazon VPC features to give a Lambda function network connectivity to the public Internet, as well as connectivity to private resources within your VPC. You will enable your serverless web application to integrate with the public Amazon DynamoDB APIs to retrieve configuration information, as well as integrate with a MySQL Database, hosted on Amazon RDS and deployed in a private subnet of your Amazon VPC.

Objectives

After completing this lab, you will be able to:

- Configure a VPC for use with AWS Lambda including creating a NAT Gateway and configuring route tables.
- Configure Lambda functions to access resources within a VPC.
- Optimize Lambda function code to cache connections between invocations.

Prerequisites

This lab requires the following:

- Access to a computer with Wi-Fi running Microsoft Windows, Mac OS X, or Linux (Ubuntu, SuSE, or Red Hat).
- The qwikLABS lab environment is not accessible using an iPad or tablet device, but you can use these devices to access the student guide.
- An Internet browser such as Chrome, Firefox, or IE9 or later (previous versions of Internet Explorer are not supported).

Duration

This lab will take approximately 45 minutes.

Task 1: Configure the Web Application and Register a User

Overview

In this task you will overwrite the configuration file stored in Amazon S3 to contain the MapBox API key associated with your new account. This step is similar to those executed in previous labs.

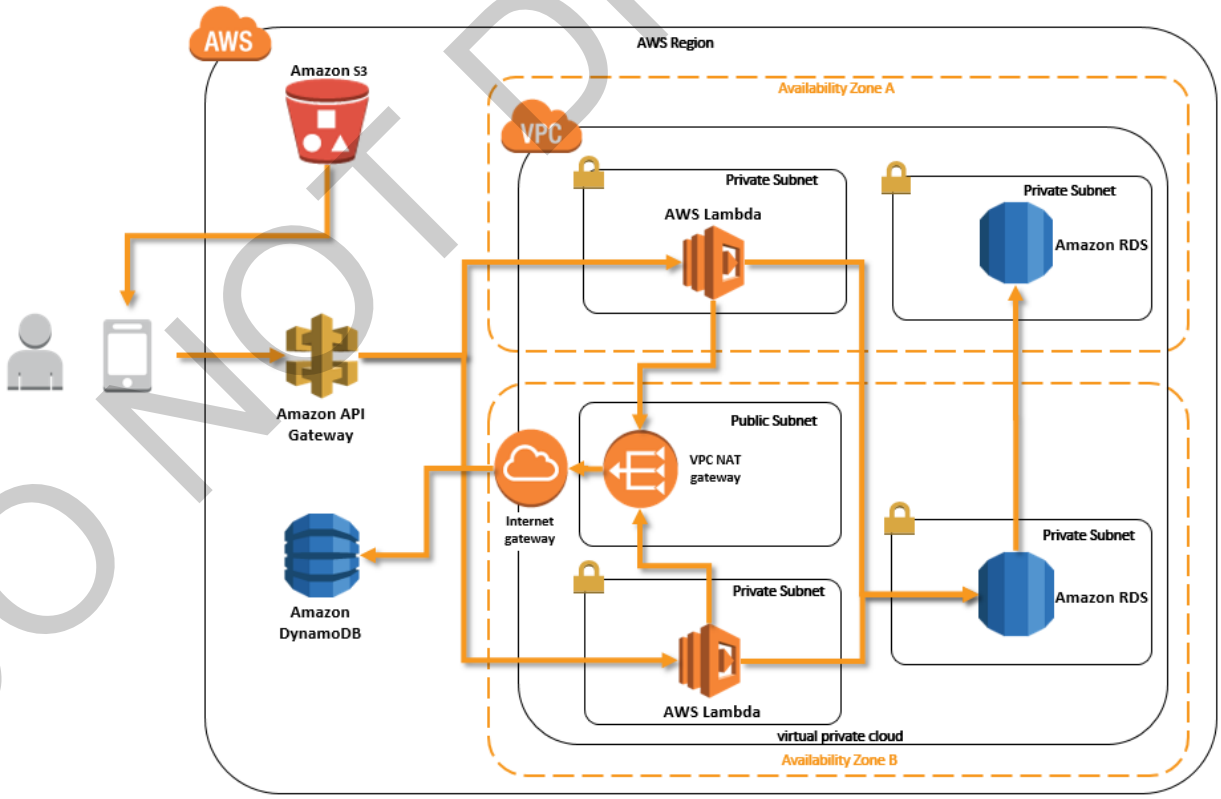
Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

Scenario

In this lab you will build the following infrastructure:



Task 1.1: Retrieve MapBox Token

In this task you will retrieve the MapBox developer token from the account you created in Lab 1. If you did not complete Lab 1, follow the alternate instructions to register with the service.

1.1.1 Visit **https://www.mapbox.com** and click **Log In**. Use the credentials you created in Lab 1 to log into your account.

1.1.2 If you do not have an existing account, click **Sign Up**. Enter a **Username, Email**, and **Password**, on the following dialog and then click the **Sign Up** button.

1.1.3 Visit your Account settings at **https://www.mapbox.com/studio/account/.** Select the menu option for **API Access Tokens**, then click **Default Public Token** dropdown to reveal your public Mapbox token

1.1.4 Copy the token to your clipboard. You will need it in Task 1.2.

Task 1.2: Update the config.js JavaScript file and Upload to Amazon S3

In this task you will modify existing JavaScript code to include your Mapbox API key.

1.2.1 In the **AWS Management Console**, on the **Services** menu, click **Storage & Content Delivery > S3**

1.2.2 Click on the *wildrydes-xxxxxxxxxxxx* bucket, then click on the *js* folder within the bucket to list the contents.

1.2.3 Right-mouse click on *config.js*, then click **Download.**

1.2.4 Right-mouse click the download link and choose “Save Link as…” to save the file on your computer.

1.2.5 Using the text editor of your choice, enter the token copied in task 1.1 (from your MapBox account) to the **accessToken** key under the mapbox section of the config.js file.

1.2.6 Save *config.js*

1.2.7 Upload *config.js* back to the *js* folder in your bucket, overwriting the original.

Task 1.3: Register a user

In this task you will register a new user for the web application. This process is identical to the one you followed in Lab 1, but because this is a new environment you will need to create a separate user account.

1.3.1 In the AWS Management Console, on the Services menu, click **Storage and Content Delivery > S3**

1.3.2 Click on the magnifying glass next to the *wildrydes-xxxxxxxxxxx* bucket

1.3.3 Click the **Properties** button near the upper right corner of your console

1.3.4 Expand the **Static Website Hosting** section in the right pane of the S3 console, and click the **Endpoint** link to load your website in the browser. It will begin with *wildrydes-xxxxxxxxxx.*

1.3.5 The serverless application uses Amazon Cognito User Pools to remove the heavy lifting from having to build and maintain your own user directories. In the **Register** section, enter your email address, then create and confirm a password for use on this application.

1.3.6 Click the **Register** button to create your account.

1.3.7 Check your email for a verification code and enter it on the /verify.html page.

1.3.8 After verifying your account, you will be redirected to the sign-in page where you can log in using the credentials you just created.

1.3.9 Once signed in, you are taken the main application map page. Verify that the map shows, and that you're able to pan around the world, indicating that the mapping JavaScript loaded correctly and that your Mapbox API was successfully initialized.

Task 2: Database Initialization

Overview

In this section you will seed the MySQL database provisioned for this lab with test datayou're your user and connect to it via a bastion host. This same process can be used in subsequent tasks to run queries against the database in order to test the other functions you develop and deploy.

Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

Task 2.1: Connect to the bastion host

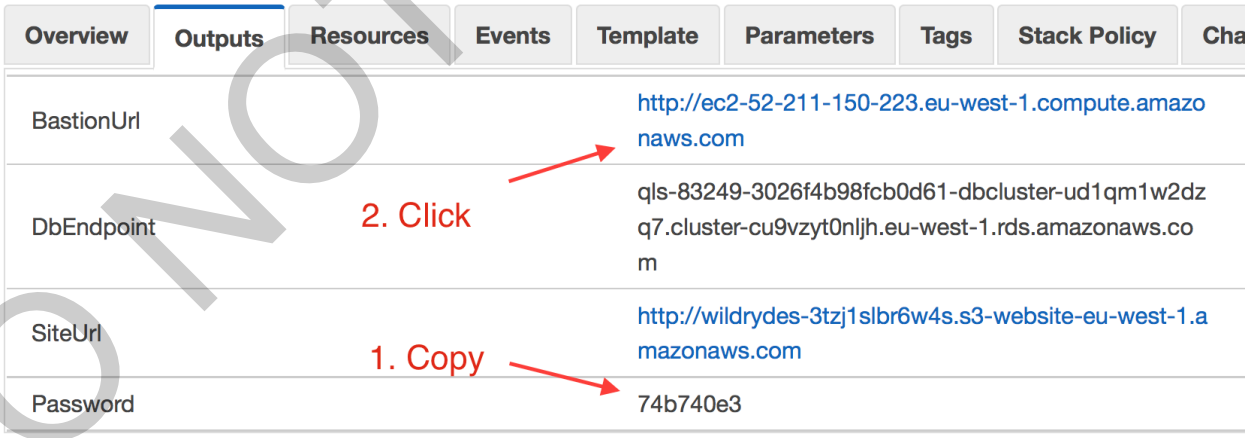
In this task you will connect to the bastion server provisioned in your VPC in order to access the Amazon Aurora database provisioned by this lab. For the sake of simplicity for this lab, the bastion server is running a web-based terminal emulator which enables you to log in and issue shell commands from a browser without needing to configure an SSH client. Typically, for normal operations you would use SSH to access your bastion hosts.

2.1.1 In the **AWS Management Console**, select the **Services** menu, then select **Management Tools** > **CloudFormation**.

2.1.2 Check the box next to the stack with a name like qls-12345-12345678abcdef. Do not select the CognitoHelperStack or ConfigHelperStack with the same prefix.

2.1.3 Select the **Outputs** tab**.**

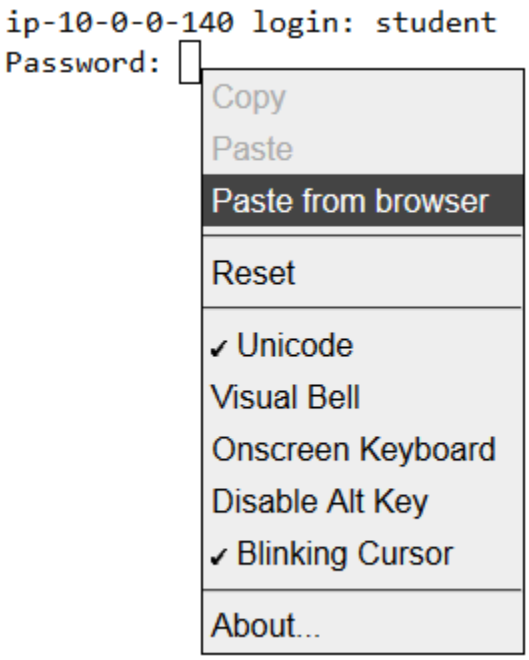
2.1.4 Copy the **Password** value to your clipboard and then click the **BastionUrl** to load the bastion host's web shell.



2.1.5 Enter the username **student** at the login prompt and hit Enter.

2.1.6 When prompted for a password, right click the browser page and select "Paste from browser". Paste the password you copied previously into the prompt and click ok. No

characters will appear, but the password will be transmitted to the server. Hit **Enter** to finalize the login.




```
ip-10-0-0-140 login: student
Password:
Last login: Fri Oct  7 18:41:58 from 205.251.233.52

       __|  __|_  )
       _|  (     /   Amazon Linux AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-ami/2016.03-release-notes/
11 package(s) needed for security, out of 71 available
Run "sudo yum update" to apply all updates.
Amazon Linux version 2016.09 is available.
[student@ip-10-0-0-140 ~]$
```


Task 2.2: Log in to MySQL Database and Confirm Schema

In this task you will connect to the deployed MySQL database and confirm that the schemas needed for this lab are correctly configured.

You can use this process during later Lab tasks in order to query the database and verify the functionality of the functions you interact with.

2.2.1 From the bastion shell, run the **inituser** script passing in your email address as the only parameter. You should use the same email address you used in Task 1 to register with the webapp.

```
[student@ip-10-0-0-140 ~]$ inituser you@domain.com
```

2.2.2 Type **db** from the command prompt of the bastion shell to open a connection to the MySQL database. **db** is a bash alias that has been preconfigured to execute the mysql client with the correct database endpoint, username and password.

```
[student@ip-10-0-0-139 ~]$ db
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 19
Server version: 5.6.10 MySQL Community Server (GPL)

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

2.2.3 Use the MySQL **DESCRIBE** command to validate the **fleet** and **ride** tables exist and match the following table structure:


```
mysql> DESCRIBE fleet;
+-------------------+-------------+------+-----+-------------------+----------------+
| Field             | Type        | Null | Key | Default           | Extra          |
+-------------------+-------------+------+-----+-------------------+----------------+
| id                | int(11)     | NO   | PRI | NULL              | auto_increment |
| available         | tinyint(1)  | NO   |     | 1                 |                |
| current_location  | point       | NO   |     | NULL              |                |
| last_status       | timestamp   | NO   |     | CURRENT_TIMESTAMP |                |
| color             | varchar(20) | NO   |     | white             |                |
| commissioned_date | date        | NO   |     | NULL              |                |
+-------------------+-------------+------+-----+-------------------+----------------+
6 rows in set (0.00 sec)

mysql> DESCRIBE ride;
+---------------------+--------------+------+-----+-------------------+----------------+
| Field               | Type         | Null | Key | Default           | Extra          |
+---------------------+--------------+------+-----+-------------------+----------------+
| id                  | int(11)      | NO   | PRI | NULL              | auto_increment |
| user_id             | varchar(30)  | YES  |     | NULL              |                |
| requested_at        | timestamp    | NO   |     | CURRENT_TIMESTAMP |                |
| pickup_address      | varchar(100) | NO   |     | NULL              |                |
| pickup_lat          | decimal(8,6) | NO   |     | NULL              |                |
| pickup_lon          | decimal(9,6) | NO   |     | NULL              |                |
| destination_address | varchar(100) | YES  |     | NULL              |                |
| destination_lat     | decimal(8,6) | YES  |     | NULL              |                |
| destination_lon     | decimal(9,6) | YES  |     | NULL              |                |
| dispatched_at       | timestamp    | YES  |     | NULL              |                |
| arrived_at          | timestamp    | YES  |     | NULL              |                |
| unicorn_id          | int(11)      | YES  | MUL | NULL              |                |
+---------------------+--------------+------+-----+-------------------+----------------+
12 rows in set (0.00 sec)
```

2.2.4 Run **SELECT * FROM ride;** to verify that the ride table has been initialized for your user.


```
mysql> SELECT * FROM ride;
+----+---------------+---------------------+---------------
-----------------------+-----------------+-----------------+
| id | user_id       | requested_at        | pickup_address
ress                   | destination_lat | destination_lon |
+----+---------------+---------------------+---------------
-----------------------+-----------------+-----------------+
|  1 | test@test.com | 2016-11-23 14:10:10 | Space Needle, S
ilding, New York, NY   |       40.748408 |      -73.985632 |
|  2 | test@test.com | 2016-11-24 16:23:23 | Empire State Bu
Dallas, TX             |       32.775355 |      -96.809057 |
|  3 | test@test.com | 2016-11-25 22:53:02 | Reunion Tower, 
dge, San Francisco, CA |       37.819954 |     -122.478502 |
|  4 | test@test.com | 2016-01-26 12:43:23 | Golden Gate Bri
ellowstone, WY         |       44.460463 |     -110.828138 |
|  5 | test@test.com | 2016-11-27 18:43:33 | Old Faithful, Y
as Vegas, NV           |       36.121781 |     -115.166198 |
+----+---------------+---------------------+---------------
-----------------------+-----------------+-----------------+
5 rows in set (0.00 sec)
```

Task 3: Configure VPC and IAM for new AWS Lambda Function

Overview

In this task you will configure your Amazon VPC so that it is well designed for the use of AWS Lambda functions that be deployed within it. This involves defining the security groups that will permit traffic to leave your Lambda functions, creating VPC subnets that your Lambda functions will be provisioned to, and configuring the subnet route tables to properly route traffic originating from your Lambda functions. Finally you will create a new Identity and Access Management (IAM) role that your Lambda function will assume during execution, defining which AWS services your function is authorized to interact with.

Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

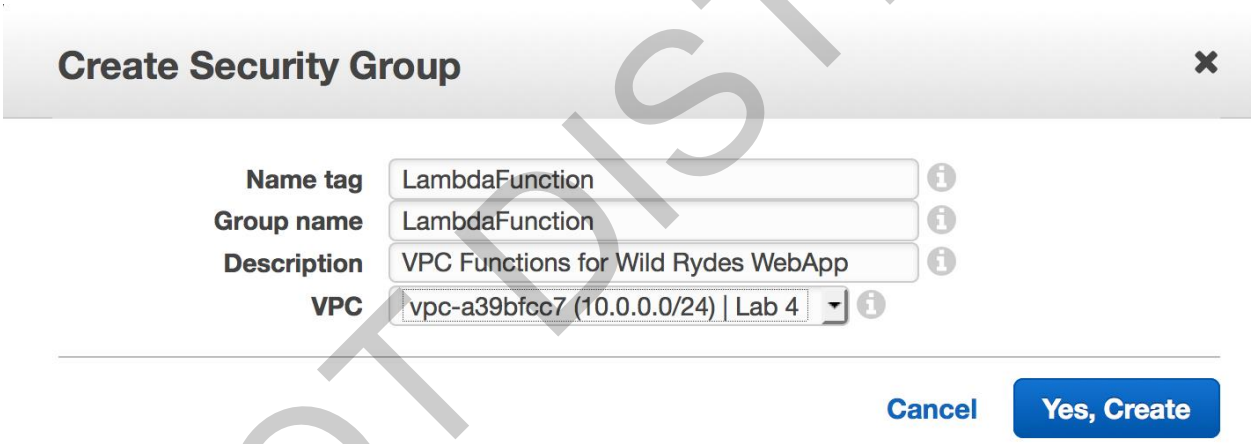
Task 3.1: Create Security Group

In this task you will create a new **Security Group** that will be assigned to the AWS Lambda function you will create in a later task. When configuring an AWS Lambda function to be provision to your own VPC, they will communicate through an Elastic Network Interface (ENI), similar to EC2 instances. In order to define what outbound traffic is permitted to be communicated from your Lambda function, a Security Group is used, also similar to EC2 instances. Once a Security Group has been created for the intended VPC for your Lambda function, you will be able to select that Security Group during Lambda Function creation/modification to define what traffic will be permitted.

3.1.1 In the **AWS Management Console**, on the **Services** menu, click **Networking > VPC.**

3.1.2 On the left navigation panel, under **Security,** click **Security Groups**. Click the blue **Create Security Group** button.

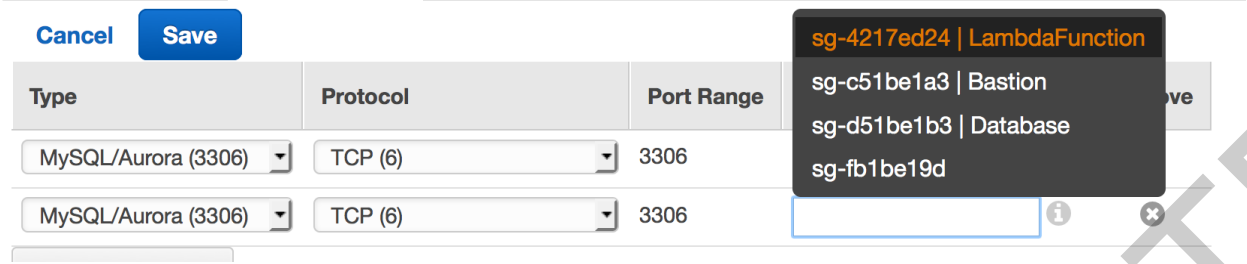
3.1.3 Enter the Security Group configuration details to match the following, ensuring that the VPC selected is the VPC named **Lab 4** (as Security Groups are VPC-specific)**:**



3.1.4 Select the **Database** security group from the Security Groups list and click the **Inbound Rules** tab.

3.1.5 Click **Edit** and then click **Add another rule**.

3.1.6 Select **MySQL/Aurora (3306)** from the Type dropdown and select the **LambdaFunction** security group you created previously for the source.



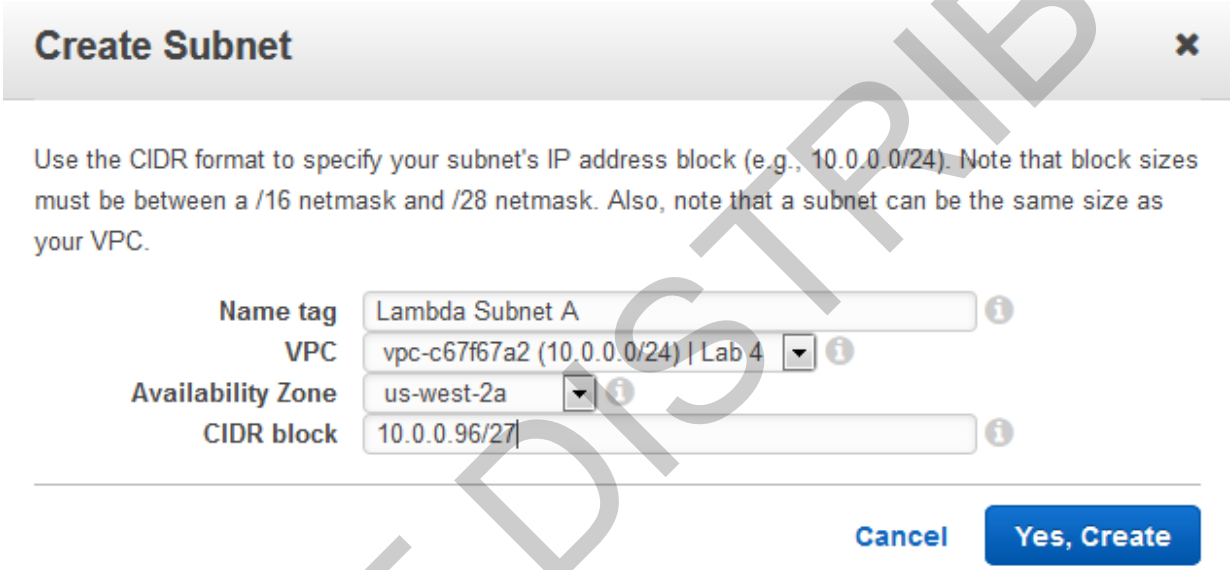
3.1.7 Click **Save**.

Task 3.2: Create VPC Subnets

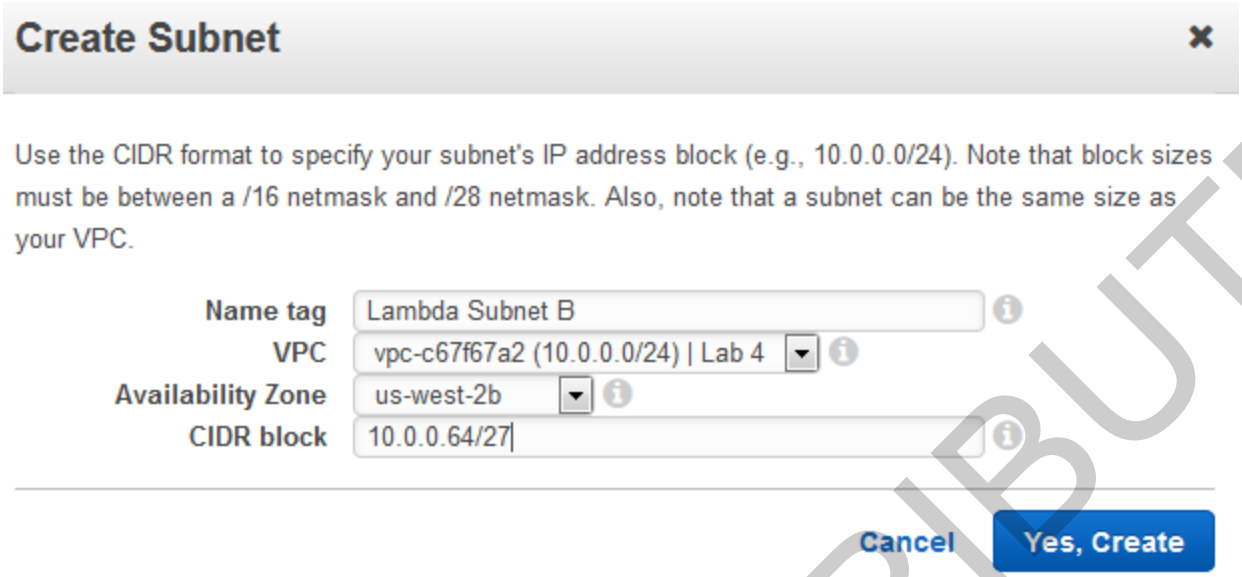
In this task you will create two new subnets that your Lambda Functions will be provisioned in. By creating a subnet in two separate Availability Zones, we can ensure our functions have high availability in the event of any AWS service event that is isolated to a single AZ.

3.2.1 Click **Subnets** on the left navigation panel of the **VPC** console, then click the blue **Create Subnet** button.

3.2.2 Input the following Subnet configuration details to create a subnet in one of the availability zones your VPC has access to and click **Yes, Create**:



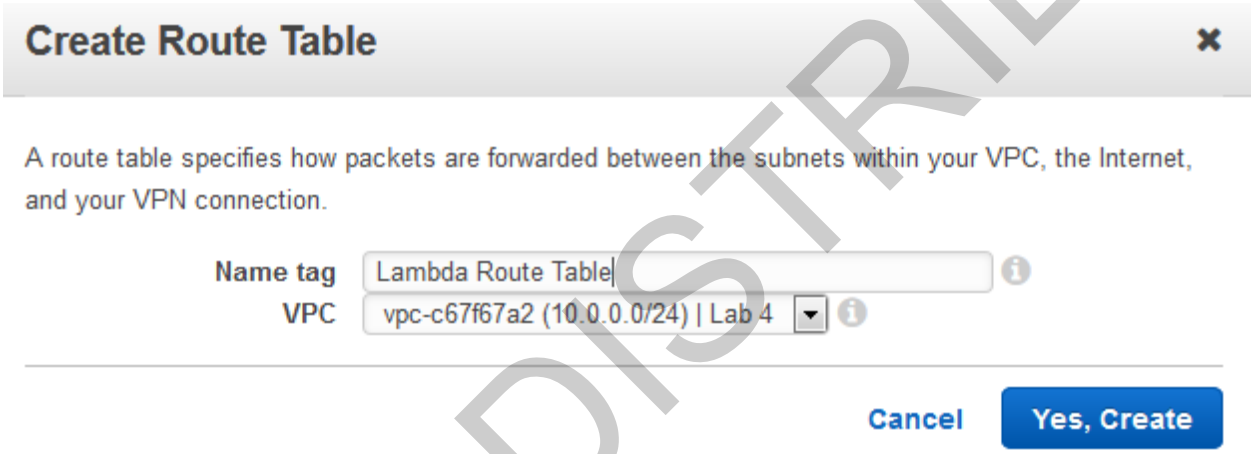
3.2.3 Create a second subnet with the following information, notice the change in **CIDR block** and choice of a different **Availability Zone**:



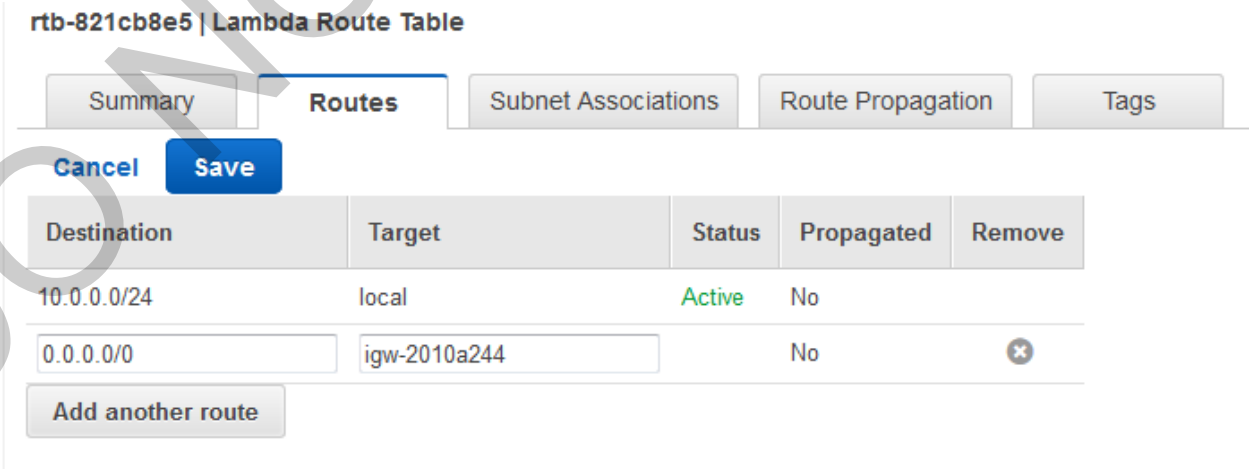
Task 3.3: Create new route table with route from new subnets to IGW

In this task you will create a new VPC route table that will direct traffic from the subnets just created to the Internet Gateway (IGW) attached to the Lab 4 VPC. Internet Gateways are what permit traffic from Elastic Network Interfaces (ENIs) that have associated Public IP addresses to communicate outbound to the Internet. However, provisioned Lambda Functions are not assigned Public IP Addresses, so the behavior will be different – as you will see later in this lab.

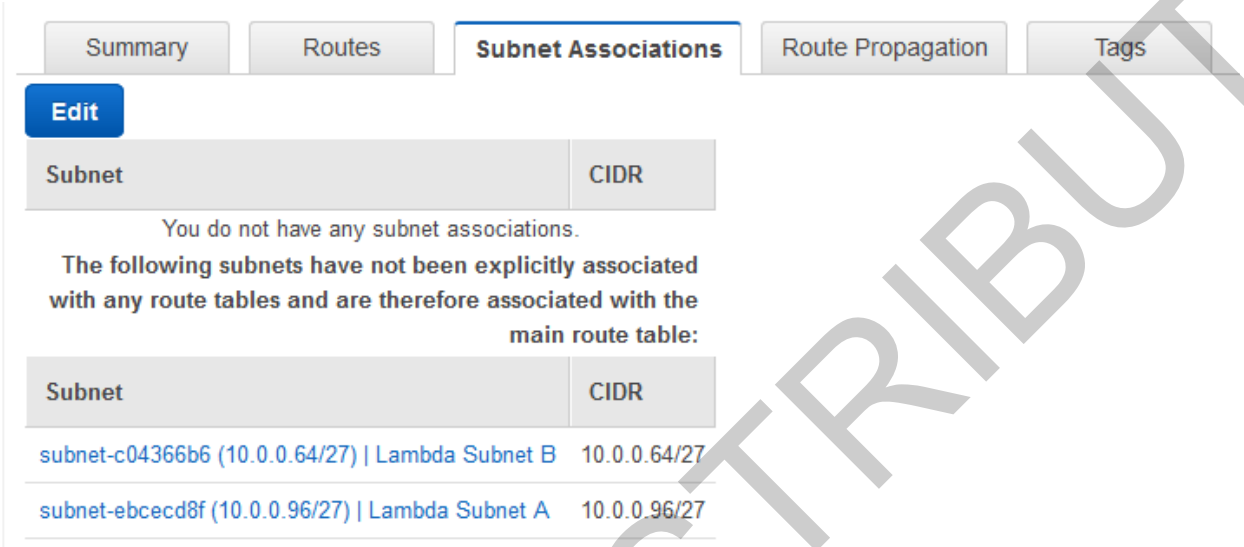
3.3.1 Click **Route Tables** on the left navigation panel of the **VPC** console. Click the blue **Create Route Table** button and enter the following information before clicking **Yes, Create:**



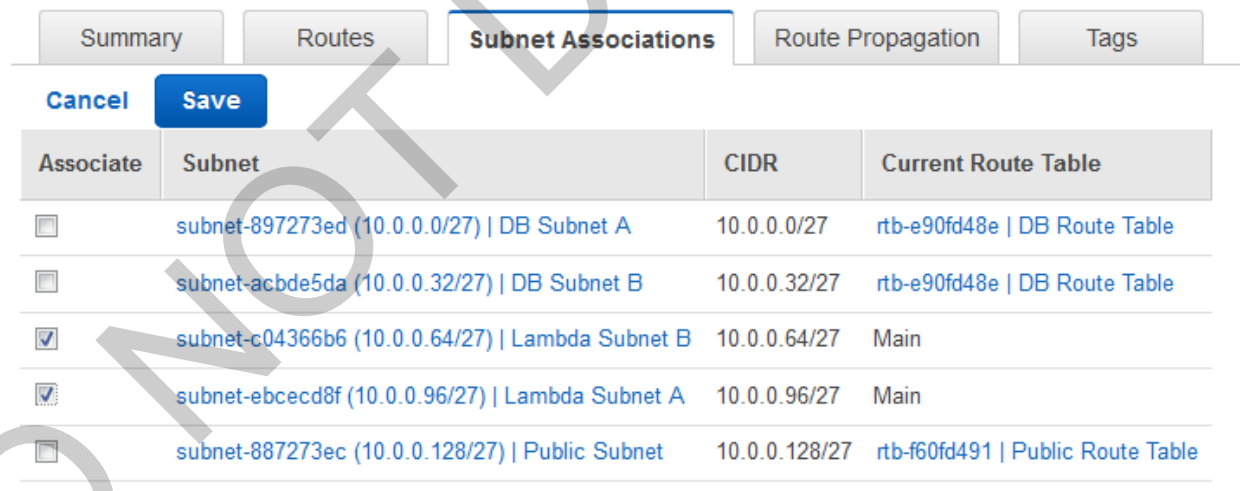
3.3.2 Click the checkbox on the left of the newly created **Lambda Route Table** and then click the **Routes** tab on the bottom details panel. Click the blue **Edit** button followed by clicking the grey **Add another route** button so that we can add a route for Internet bound traffic to the IGW in our VPC. Enter the wildcard destination IP range and select the IGW in your VPC as the target, like below, and click the blue **Save** button:



3.3.3 Now, the route table must be associated with the two Lambda Subnets we have created previously. Click the **Subnet Associations** tab in the Subnet details panel. Here, you will be presented with details about all of the subnets that are using this route table to route traffic (currently none), and which subnets exist that are still using the VPC main route table and can still be associated with your new route table:



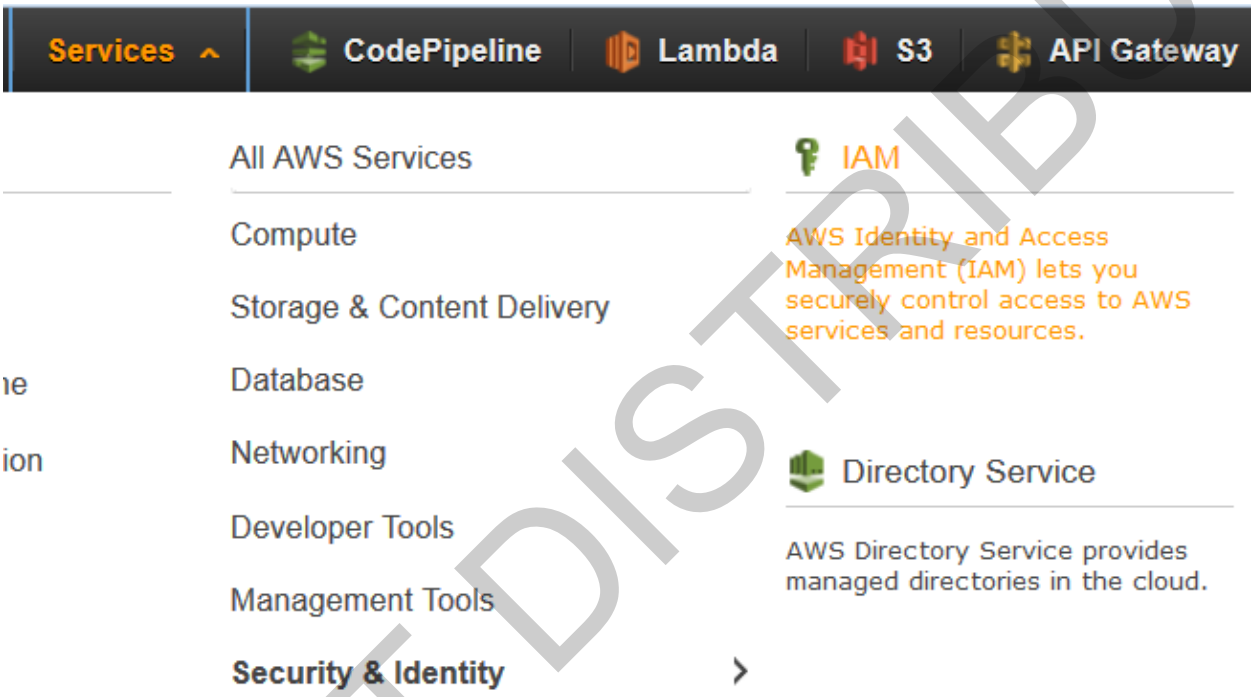
3.3.4 Click the blue **Edit** button so that we can associate the two Lambda Subnets with this new route table. Check the boxes next to **Lambda Subnet A,** and **Lambda Subnet B** and click **Save:**



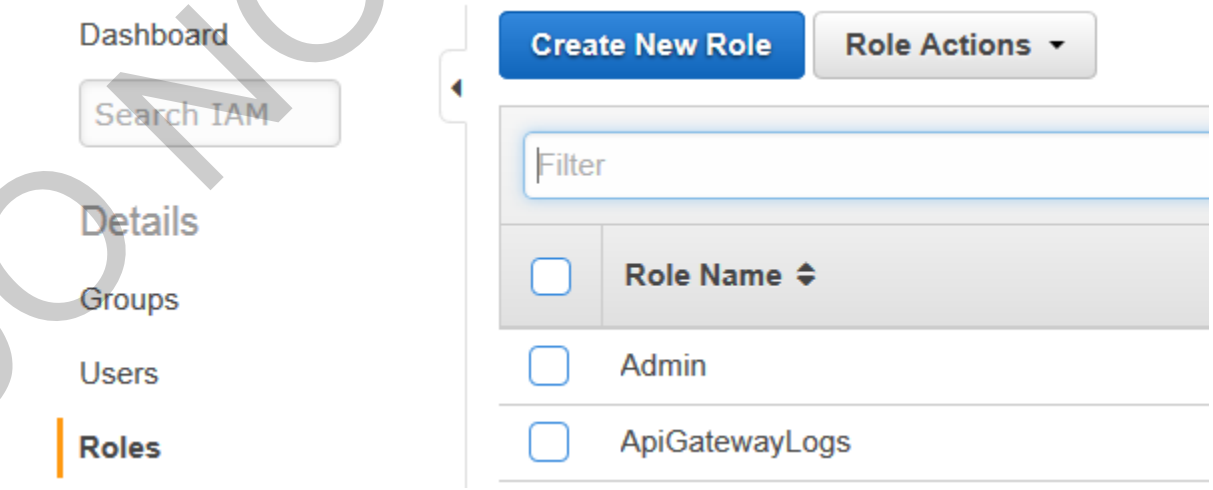
Task 3.4: Create IAM Role to be used as Lambda Execution Role

In this task you will create a new AWS Identity and Access Management Role that will be used as the Execution Role for your Lambda Function.

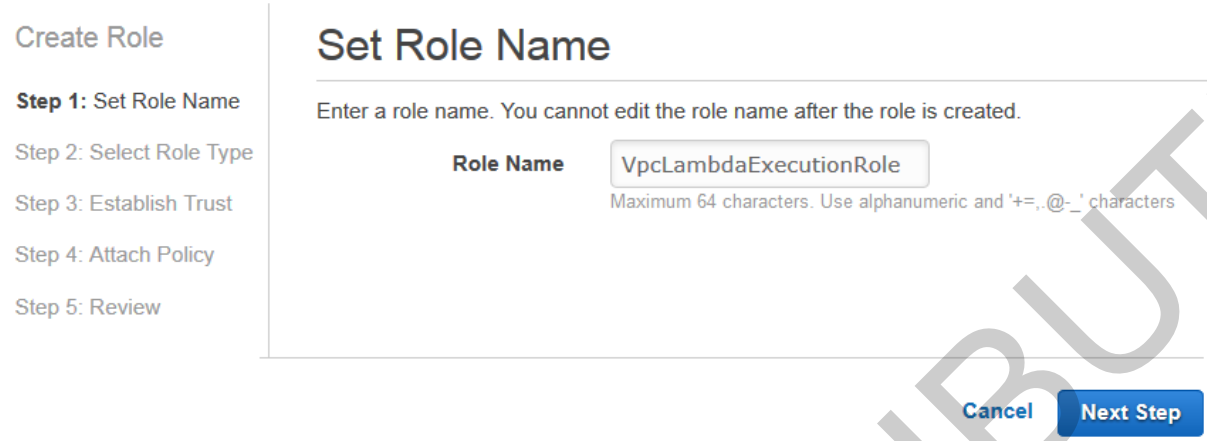
3.4.1 Navigate to the Identity and Access Management (IAM) console. Click **Services** on the top-left, then click **Security & Identity**, then **IAM:**



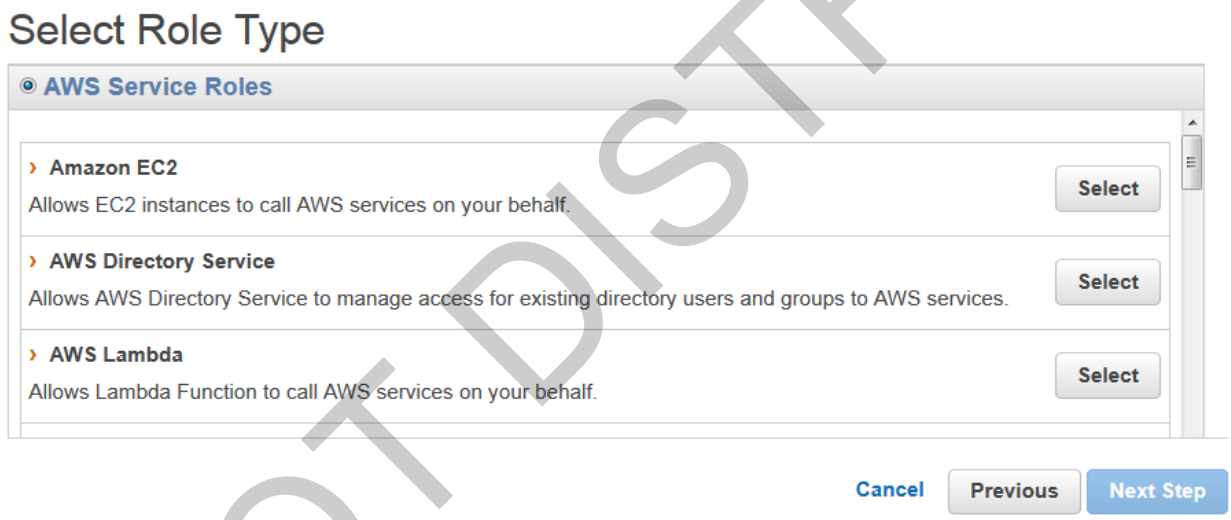
3.4.2 Click **Roles,** then click **Create New Role**.



3.4.3 Enter the Role Name **VpcLambdaExecutionRole:**

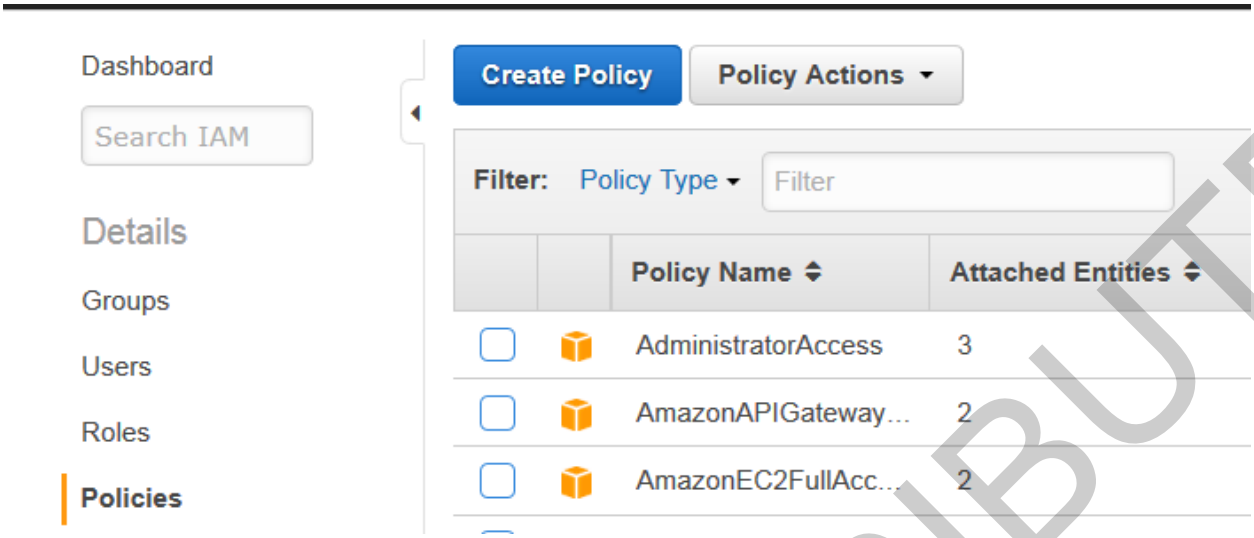


3.4.4 Select **AWS Lambda** as the AWS Service Role Type:

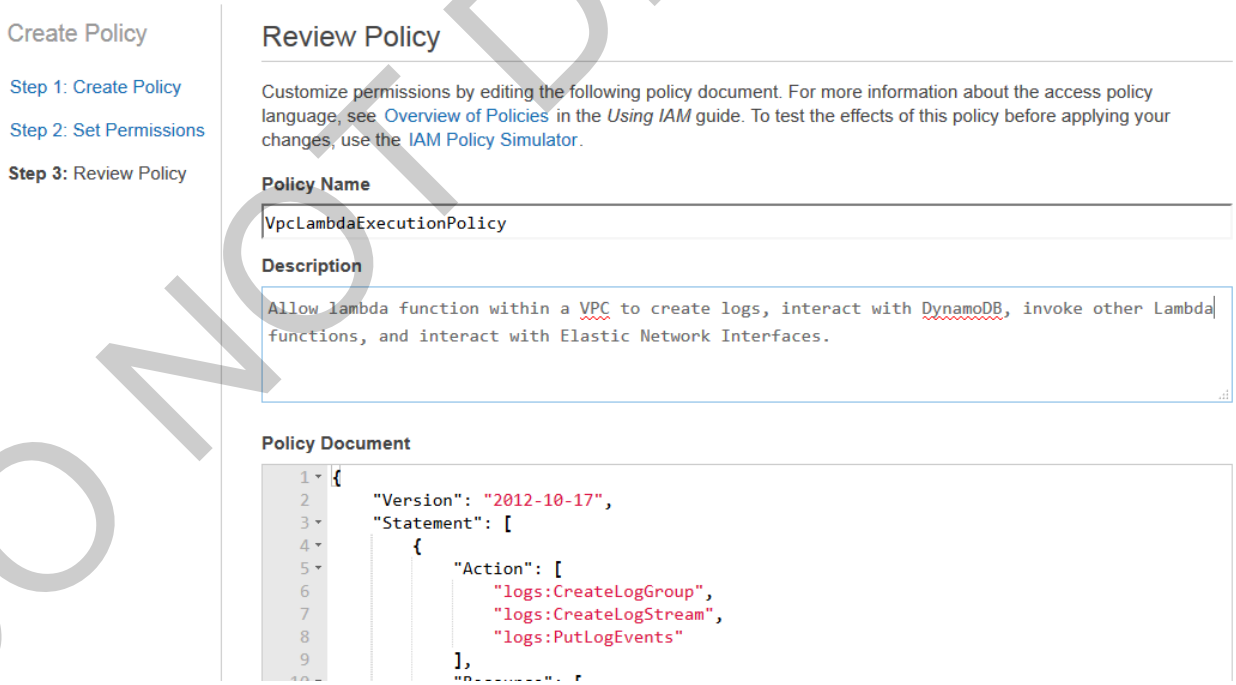


3.4.5 Leave the remaining options as their default values, and click through **Next Step** and **Create Role.**

3.4.6 Now, we will create an IAM Policy that will be attached to this new role and define what permissions your Lambda function will have. Click **Policies** on the left navigation panel, then click the blue **Create Policy** button.

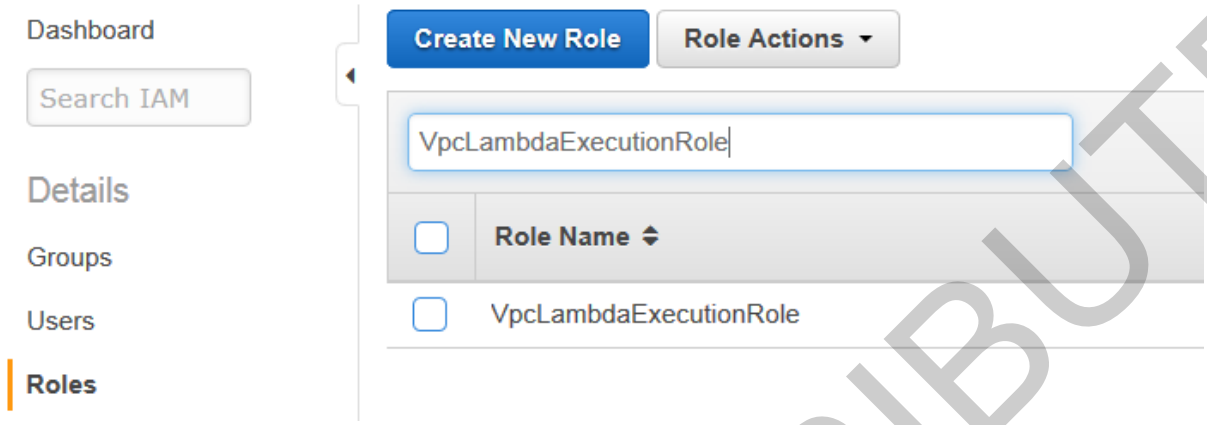


3.4.7 Click **Create Your Own Policy**. On the subsequent page, set the **Policy Name** to **VpcLambdaExecutionPolicy** give it a description to your liking, and paste in the IAM Policy json available in the **Command Reference File** for this lab. Take a minute to review the policy document to discover what our Lambda function requires permissions to do. One statement to make particular note is the statement that allows actions related to EC2 Network Interfaces. Because this AWS Lambda function will be deployed within a VPC and communicate through ENIs. Your function needs to be authorized to interact with the ENI APIs, unlike Lambda functions that will be provisioned in the default networking environment.

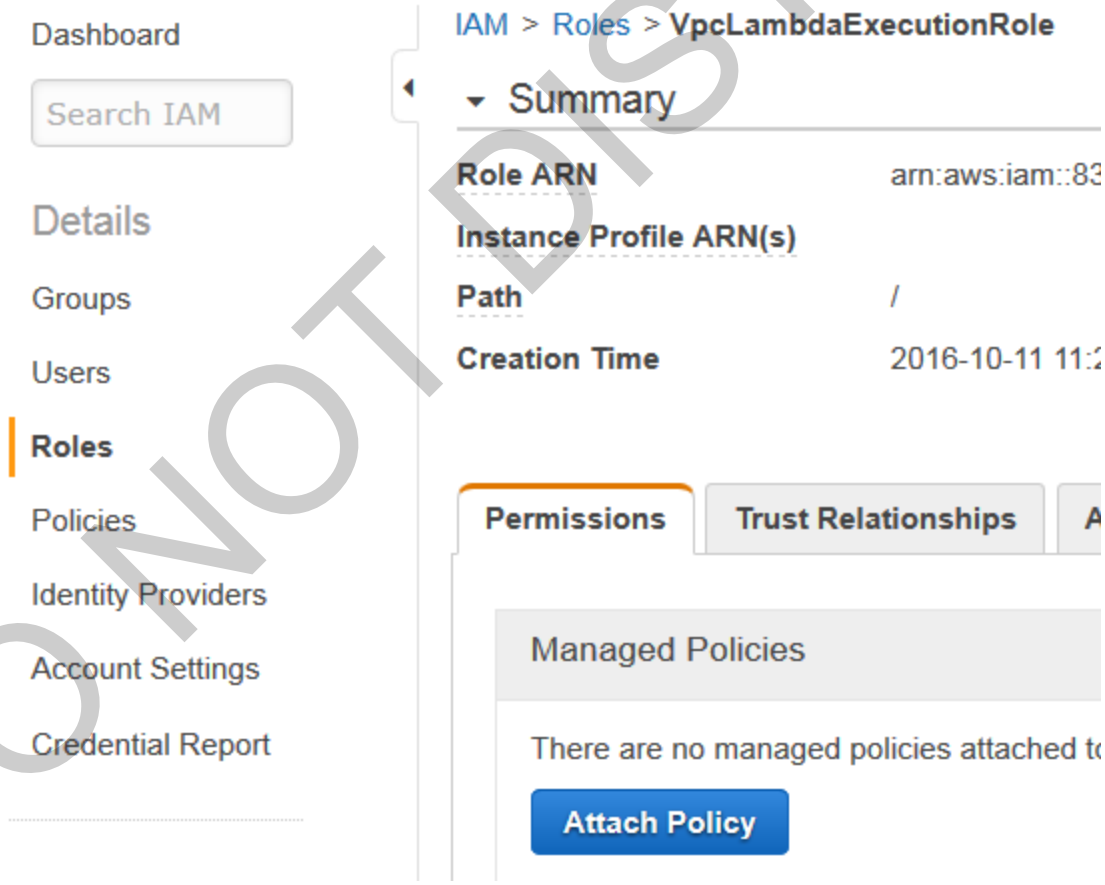


3.4.8 Click the blue **Create** button once the Policy information has been entered.

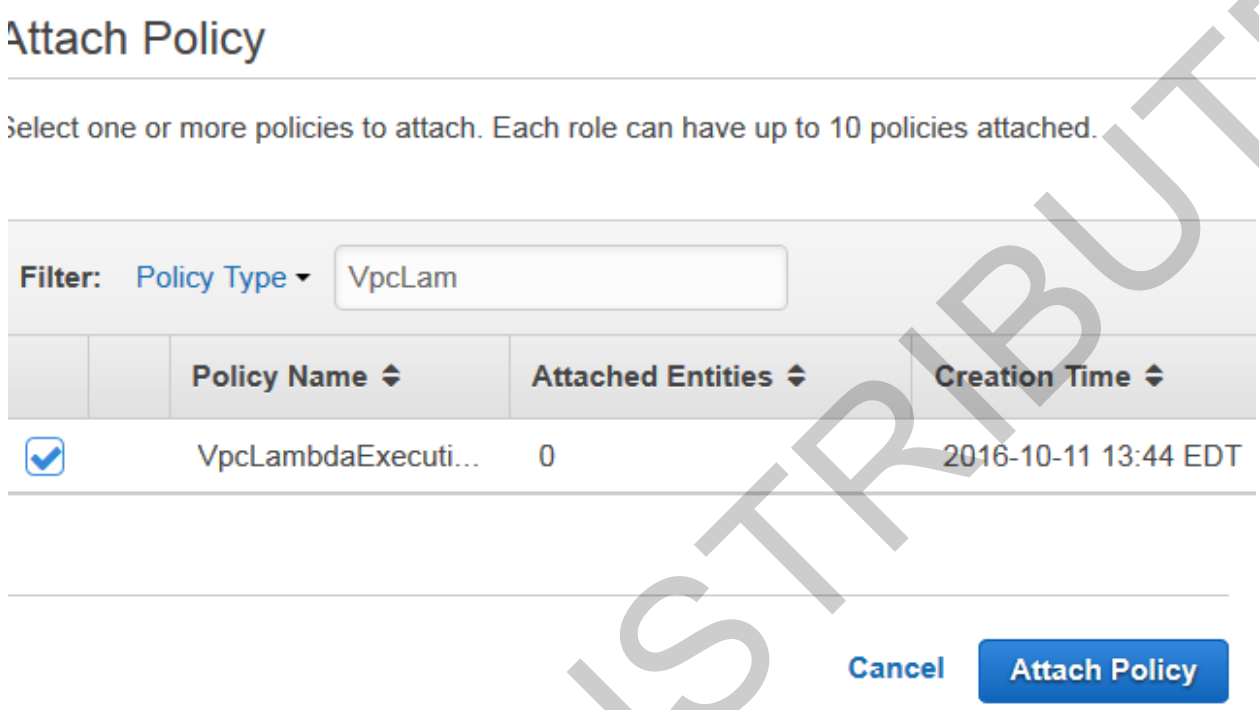
3.4.9 Click **Roles** on the left navigation panel and select the role **VpcLambdaExecutionRole** that you just created previously.



3.4.10 Under the **Permissions** tab, click **Attach Policy** so that we may attach the new policy we just created with this role.



3.4.11 Type **VpcLambdaExecutionPolicy** in the filter box, check the box next to this policy, and then the blue **Attach Policy** button in order to successfully attach it to the **VpcLambdaExecutionRole.**



Task 4: Integrate ListRides with the VPC

Overview

In this task you will update the existing ListRides function to run within the existing VPC so it can reach the Amazon RDS database you validated in Task 1. The function code has already been updated to integrate with MySQL instead of DynamoDB, but until the function has network connectivity to the database instance, the function won't work.

Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

Task 4.1: Update ListRides VPC Configuration

In this task you will update the ListRides function in order to give it network access to your VPC.

4.1.1 In the **AWS Management Console**, on the **Services** menu, click **Compute > Lambda.**

4.1.2 Select the **ListRides** function in the Lambda console and select the **Configuration** tab.

4.1.3 Select the **VpcLambdaExecutionRole** you created in Task 3 from the **Existing role** dropdown.

4.1.4 Scroll down and expand the **Advanced settings** section.

4.1.5 Select the **Lab 4** VPC from the VPC dropdown.

4.1.6 From the **Subnets** dropdown, select the two subnets you created in Task 3 (**Lambda Subnet A** and **Lambda Subnet B**).

4.1.7 Select the **LambdaFunction** security group you created in Task 3 from the **Security Groups** dropdown.

4.1.8 Scroll back to the top of the page and click **Save** to save your changes.

Task 4.2: Perform Test Execution of ListRides (Failure)

In this task you will sent a Test Execution Event to your new Lambda function and inspect the results. With the current configuration this test will fail because your function is not correctly configured to access the internet. Internet access is required for all of the functions in this lab because they use DynamoDB to retrieve configuration data when they first start.

The current route table configuration for the subnets you created in Task 3 provides a route to an Internet gateway directly. Because the Elastic Network Interfaces used by Lambda functions that run in VPCs do not have public IP addresses assigned to them, your function will not be able to send traffic to the Internet via this route successfully.

4.2.1 Select the **Configure test event** option from the **Actions** dropdown on the ListRides function detail page.

4.2.2 Replace the existing JSON document with the following content, replacing the UserId with the email address you used to register with the site.

```
{ "UserId": "you@domain.com" }
```

Note: The test you execute in the following step is expected to fail.

4.2.3 Click **Save and test** to test the function with this event. The function execution will time out because the query to DynamoDB to fetch configuration data cannot complete. You will fix this issue in the next task.

Task 4.3: Reconfigure Subnet Route Table to use NAT Gateway

In this task you will reconfigure your VPC Subnet Route Table such that Internet-bound traffic is routed to a NAT Gateway rather than directly to an Internet Gateway (IGW). Because Lambda functions are not directly assigned Public IP addresses, they are unable to communicate directly out to the Internet and must utilize a NAT Gateway as their door out to the Internet.

4.3.1 In the **AWS Management Console**, select the **Services** menu, then select **Networking** > **VPC**.

4.3.2 Select **NAT Gateways** from the left menu and then click **Create NAT Gateway**.

4.3.3 Click on the **Subnet** text box and select the Public Subnet from the dropdown that appears.

4.3.4 Click **Create New EIP** to create a new Elastic IP and associate it with the new NAT Gateway.

4.3.5 Click **Create a NAT Gateway** to finalize the creation process.

4.3.6 Click **Edit Route Tables** from the confirmation dialog to take to you the route tables console.

4.3.7 Select the Lambda Route Table you created in Task 3, click the **Routes** tab and then click **Edit**.

4.3.8 Move your cursor to the text box for the target of the 0.0.0.0/0 route. The current route target should to an Internet Gateway (e.g. igw-xxxxxx).

4.3.9 Delete the text in the target text box and select the NAT Gateway you created previously.

4.3.10 Click **Save** to save your changes.

Task 4.4: Perform Test Execution of ListRides (Success)

Perform the same test execution from Task 4.2 that previously failed. Now that the function has an appropriate route to a NAT Gateway for Internet access the test should succeed.

4.4.1 In the **AWS Management Console**, select the **Services** menu, then select **Compute** > **Lambda**.

4.4.2 Click on the **ListRides** function.

4.4.3 Select the Configure test event option from the Actions dropdown and confirm that the test event you used in task 4.2 is still correctly configured. If it is not, enter the following JSON document in the text area replacing the UserId value with your email address.

```
{ "UserId": "you@domain.com" }
```

4.4.4 Click **Save and Test.**

4.4.5 Confirm that the function returns an array of ride objects and that there are no errors reported.

4.4.6 Log into the Wild Rydes web application using the user you registered in Task 1 and verify that the ride history list functionality works as expected.

Task 4.5: Update ListRides to cache database connections

In this task you will update the code for the ListRides function to cache database connections between invocations within the same container. When AWS Lambda executes your function it will reuse containers in certain situations. You can take advantage of this fact by storing the connection object in a variable defined in the global scope. By doing this, function invocations that execute in an existing container will not need to recreate a separate database connection which will limit the total number of connections to your database and improve performance.

The completed code for the ListRides function is available in the list-rides-updated.js file in the Lab Instructions menu.

4.5.1 In the **AWS Management Console**, select the **Services** menu, then select **Compute** > **Lambda**.

4.5.2 Click on the **ListRides** function.

4.5.3 In the code editor create a variable named **config** in the global scope above the handler function definition.

4.5.4 Create a function with the signature initConnection(context). This function will initialize our database connection or return the existing, cached value. Copy and paste the function definition from the command reference.

4.5.5 Update the handler function to call the initConnection function instead of calling fetchConfig and creating the connection inline.

4.5.6 The changes needed and resulting code should look like the following:


```
let config;

function initConnection(context) {
    if(config) {
        console.log("Using cached connection");
        return Promise.resolve(config.connection);
    } else {
        return fetchConfig(context).then(c => {
            config = c;
            console.log("Creating new connection");
            config.connection = mysql.createConnection({
                host     : config.DbHost,
                user     : "admin",
                password : config.DbPassword,
                database : 'wildrydes'
            });
            return config.connection;
        })
    }
}

exports.handler = (event, context, callback) => {
    context.callbackWaitsForEmptyEventLoop = false;

    initConnection(context).then( conn => {
        return conn.query(
            'SELECT id, pickup_address, pickup_lat, pickup_lon, ' +
            '  destination_address, destination_lat, destination_lon, ' +
            '  requested_at ' +
            'FROM wildrydes.ride ' +
            'WHERE user_id = ?',
            [event.UserId]
        );
    }).then( results => {
        let rows = results[0];
```

4.5.3 - variable defined

4.5.4 - function pasted from command reference

4.5.5 - initConnection call added

4.5.7 Click **Save and Test** to validate the new function implementation.

4.5.8 Click **Test** again and notice in the logs that the function now uses the cached connection.

Task 5: Update Remaining Lambda Functions for VPC

Overview

In order for the remaining functionality of the Wild Rydes web site to be functional, the remaining Lambda functions need to have their configuration updated to also be deployed within the VPC Subnets that have access to the Amazon Aurora database on RDS.

Command Reference File

Use the command reference file when copying text provided in this lab manual. The command reference file is available by clicking the ADDL. INFO button of your lab in qwikLABS.

You should not copy and paste commands directly from this lab manual, because the manual's rich formatting may inject characters that could introduce errors to your lab experience. Download the reference file to your computer instead.

Task 5.1: Reconfigure Remaining Lambda Functions

5.1.1 Repeat all of the steps of **Task 4.1** other than **Step 4.1.3** (leave existing Lambda execution Role as is) for the following Lambda Functions:

(1) **DispatchUnicorn**

(2) **GetRide**

(3) **RequestRide**

(4) **GetRideAssignment**

Note Manually testing each of these functions through the console is outside the scope of this lab. Because each of these functions requires different fields for its input event, you will see failures if you try to test these functions using the default test event.

5.1.2 Open the Wild Rydes web application and confirm that all functionality for reserving Unicorns is now functional using a private database hosted in Amazon RDS!

Lab Complete

Congratulations! You have successfully completed Serverless Solutions: VPC and Hybrid Architectures

1. Log out of the **AWS Management Console** by clicking **awsstudent** in the top right corner and click **Sign Out**.

2. Return to the **qwikLABS** page where you launched your lab and click **End**.

