



Module	Assessment Type
Distributed and Cloud Systems Programming	Individual Report

Workshop 3

Student Id : 2049867 (NP03A190017)
Student Name : Roshan Parajuli
Section : L5CG3
Module Leader : Rupak Koirala
Lecturer /Tutor : Saroj Sharma
Submitted on : 2020-03-30

Table of contents

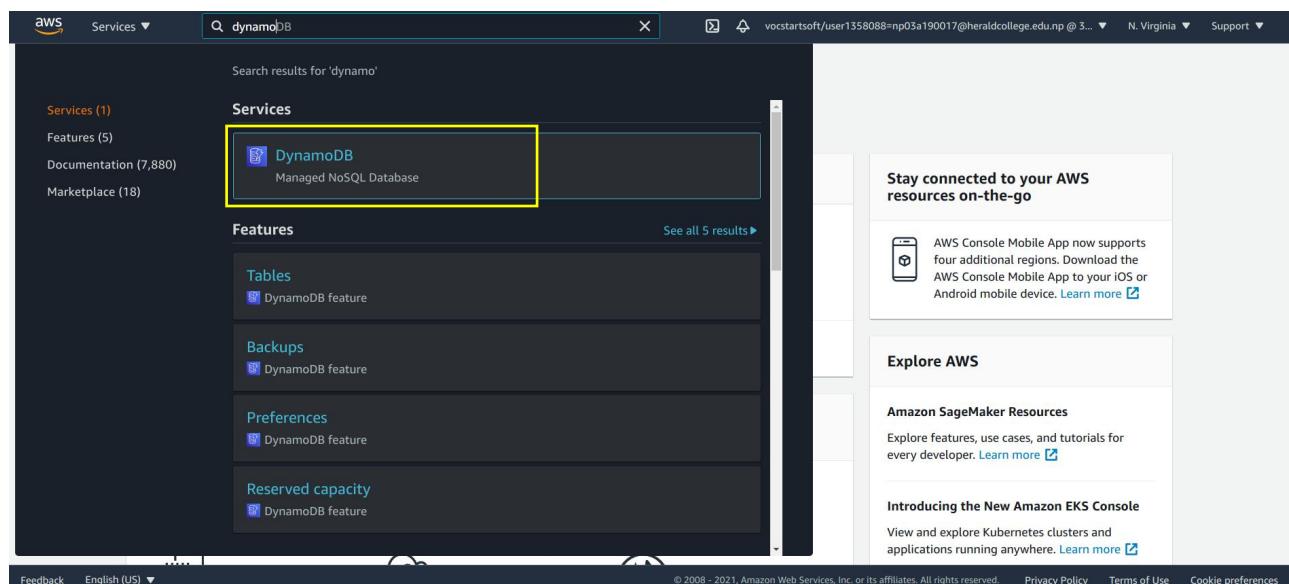
Task 1: Serverless Service Backend.....	3
Creating a DynamoDB table.....	3
Creating an IAM Role for the Lambda function.....	5
Creating a Lambda Function for Handling requests.....	9
Validating the implementation.....	11
Task 2: Serverless Service Backend.....	13
Creating a new REST API.....	13
Creating a Cognito User Pools Authorizer.....	14
Verifying the authorizer configuration.....	15
Creating a new resource and method.....	16
Deploying the API.....	19
Updating the Website Config.....	20
Validating the implementation.....	22

Task 1: Serverless Service Backend

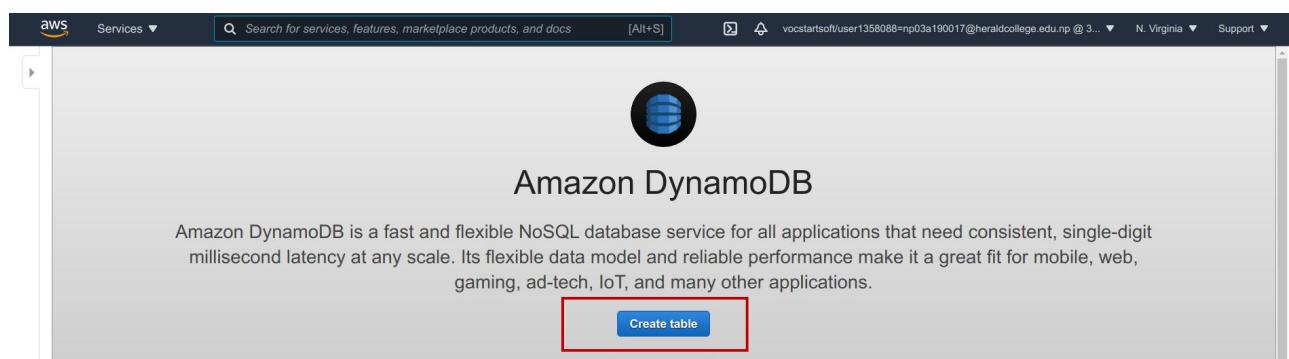
Continuing the workshop 2, a **lambda function** is to be implemented which needs to be invoked everytime a user requests a ride. As a first part of the task, Amazon DynamoDB table is created.

Creating a DynamoDB table

First, DynamoDB service is selected from the search box. **Amazon DynamoDB** is a fully managed **NoSQL database** service that provides fast and predictable performance with seamless scalability.



The screenshot shows the AWS search interface with the search term 'dynamo' entered. The 'DynamoDB' service is highlighted with a yellow box. Other search results include 'Tables', 'Backups', 'Preferences', and 'Reserved capacity' under the 'Features' section. To the right, there are promotional banners for the AWS mobile app and the new Amazon EKS console.



The screenshot shows the Amazon DynamoDB landing page. It features a large 'Create table' button at the top. Below it, there is descriptive text about the service's capabilities and a 'Getting started guide' link. Three icons are displayed at the bottom: a database icon with a plus sign, a cloud and search icon, and a monitor with a graph and checkmark icon.

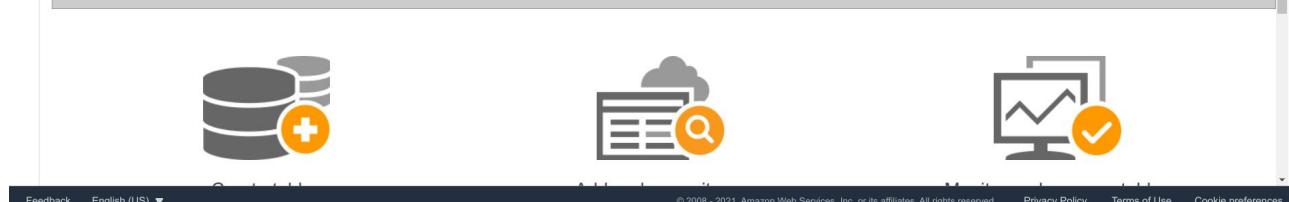


Table name* Rides

Primary key* Partition key

Rideld String

Add sort key

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

Use default settings

- No secondary indexes.
- Provisioned capacity set to 5 reads and 5 writes.
- Basic alarms with 80% upper threshold using SNS topic "dynamodb".
- Encryption at Rest with DEFAULT encryption type.

Info You do not have the required role to enable Auto Scaling by default.
Please refer to documentation.

+ Add tags NEW!

Additional charges may apply if you exceed the AWS Free Tier levels for CloudWatch or Simple Notification Service. Advanced alarm settings are available in the CloudWatch management console.

Create

The table is named “Rides” and a partition key called “Rideld” is provided to the table. Partition key is a simple primary key which helps in identify each record uniquely in the database.

Rides Close

Overview Items Metrics Alarms Capacity Indexes Global Tables Backups Contributor Insights Triggers Access control Tags

Table details

Table name	Rides
Primary partition key	Rideld (String)
Primary sort key	-
Point-in-time recovery	DISABLED Enable
Encryption Type	DEFAULT Manage Encryption
KMS Master Key ARN	Not Applicable
Encryption Status	
CloudWatch Contributor Insights	DISABLED Manage Contributor Insights NEW
Time to live attribute	DISABLED Manage TTL
Table status	Active
Creation date	March 31, 2021 at 8:16:24 AM UTC+5:45
Read/write capacity mode	Provisioned
Last change to on-demand mode	-
Provisioned read capacity units	5 (Auto Scaling Error)
Provisioned write capacity units	5 (Auto Scaling Error)
Last decrease time	-
Last increase time	-
Storage size (in bytes)	0 bytes
Item count	0 Manage live count
Region	US East (N. Virginia)
Amazon Resource Name (ARN)	arn:aws:dynamodb:us-east-1:360180145664:table/Rides

Storage size and item count are not updated in real-time. They are updated periodically, roughly every six hours.

The **Amazon Resource Name (ARN)** was noted for further use. ARN is a file naming convention used to identify a particular resource in the Amazon Web Services (AWS) public cloud.

Creating an IAM Role for the Lambda function

Before a lambda function is created for handling requests, an Identity and Access Management (IAM) role is to be created for giving the appropriate permissions to the lambda function. It defines what other AWS services the function is allowed to interact with.

The screenshot shows two screenshots of the AWS IAM interface. The top screenshot is a search results page for 'iam'. It has a sidebar with 'Services (1)', 'Features (10)', 'Documentation (62,934)', and 'Marketplace (206)'. The main area shows a 'Services' section with 'IAM' highlighted and a 'Features' section with 'Groups', 'Roles', 'Policies', and 'Users'. A red box highlights the 'IAM' item in the Services list. The bottom screenshot shows the 'Roles' page under the 'Identity and Access Management (IAM)' navigation. It has a 'Create role' button highlighted with a red box. The main table lists 13 existing roles, each with a checkbox, the role name, the trusted entity (e.g., AWS service: cloud9), and the last activity date (e.g., None, Today). The table has columns for 'Role name', 'Trusted entities', and 'Last activity'.

Role name	Trusted entities	Last activity
AWSServiceRoleForAWSCloud9	AWS service: cloud9 (Service-Linked role)	None
AWSServiceRoleForCloudWatchEvents	AWS service: events (Service-Linked role)	None
AWSServiceRoleForElastiCache	AWS service: elasticache (Service-Linked role)	None
AWSServiceRoleForOrganizations	AWS service: organizations (Service-Linked r...	None
AWSServiceRoleForSupport	AWS service: support (Service-Linked role)	None
AWSServiceRoleForTrustedAdvisor	AWS service: trustedadvisor (Service-Linked ...	None
EMR_AutoScaling_DefaultRole	AWS service: elasticmapreduce and 1 more	None
EMR_DefaultRole	AWS service: elasticmapreduce	None
EMR_EC2_DefaultRole	AWS service: ec2	None
lambda_basic_execution	AWS service: lambda	Today
robomaker_students	AWS service: robomaker and 3 more	None
vocareum	Account: 769722020711	You need permissions
vocstartsoft	Account: 769722020711	You need permissions

Create role

Select type of trusted entity

AWS service EC2, Lambda and others

Another AWS account Belonging to you or 3rd party

Web identity Cognito or any OpenID provider

SAML 2.0 federation Your corporate directory

Allows AWS services to perform actions on your behalf. Learn more

Choose a use case

Common use cases

EC2 Allows EC2 instances to call AWS services on your behalf.

Lambda Allows Lambda functions to call AWS services on your behalf.

Or select a service to view its use cases

API Gateway	CloudWatch Events	EKS	IoT SiteWise	RDS
AWS Backup	CodeBuild	EMR	IoT Things Graph	Redshift
AWS Chatbot	CodeDeploy	ElastiCache	KMS	Rekognition

* Required

Cancel Next: Permissions

https://console.aws.amazon.com/iam/home?region=us-east-1#/roles\$new

We now have to select what permissions are to be given for the role.

Create role

Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy

Filter policies ▾ Q awslambda Showing 14 results

Policy name	Used as
<input checked="" type="checkbox"/> AWSLambdaBasicExecutionRole	None
<input type="checkbox"/> AWSLambdaBasicExecutionRole-1e7e65aa-434f-40e8-9cf4-d68d3291313e	Permissions policy (1)
<input type="checkbox"/> AWSLambdaDynamoDBExecutionRole	None
<input type="checkbox"/> AWSLambdaNIMManagementAccess	None
<input type="checkbox"/> AWSLambdaExecute	None
<input type="checkbox"/> AWSLambdaInvocation-DynamoDB	None
<input type="checkbox"/> AWSLambdaKinesisExecutionRole	None
<input type="checkbox"/> AWSLambdaMSKExecutionRole	None

* Required

Cancel Previous Next: Tags

Feedback English (US) ▾ © 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use Cookie preferences

As the name suggests, **AWSLambdaBasicExecutionRole** provides very basic permissions for the lambda function to work. Using this would be an easier option instead of manually adding CloudWatch Logs permissions as it handles the logs on its own.

Create role

Review

Provide the required information below and review this role before you create it.

Role name*	TaxiRidesLambda
Use alphanumeric and '+-, @-' characters. Maximum 64 characters.	
Role description	Allows Lambda functions to call AWS services on your behalf.
Maximum 1000 characters. Use alphanumeric and '+-, @-' characters.	
Trusted entities	AWS service: lambda.amazonaws.com
Policies	AWSLambdaBasicExecutionRole Edit
Permissions boundary	Permissions boundary is not set
No tags were added.	

* Required

[Cancel](#) [Previous](#) [Create role](#)

The role is created successfully. Now, we need to give the role appropriate permissions for adding items to the DynamoDB.

Identity and Access Management (IAM)

Dashboard

Access management

- Groups
- Users
- Roles**
- Policies
- Identity providers
- Account settings

Access reports

- Access analyzer
- Archive rules
- Analyzers
- Settings

Credential report

Organization activity

Service control policies (SCPs)

Feedback English (US) ▾

Search for services, features, marketplace products, and docs [Alt+S]

Roles > TaxiRidesLambda Summary [Delete role](#)

Role ARN: arn:aws:iam::360180145664:role/TaxiRidesLambda [Edit](#)

Role description: Allows Lambda functions to call AWS services on your behalf. | Edit

Instance Profile ARNs: [Edit](#)

Path: /

Creation time: 2021-03-31 08:26 UTC+0545

Last activity: Not accessed in the tracking period

Maximum session duration: 1 hour [Edit](#)

Permissions [Trust relationships](#) [Tags](#) [Access Advisor](#) [Revoke sessions](#)

▼ Permissions policies (1 policy applied)

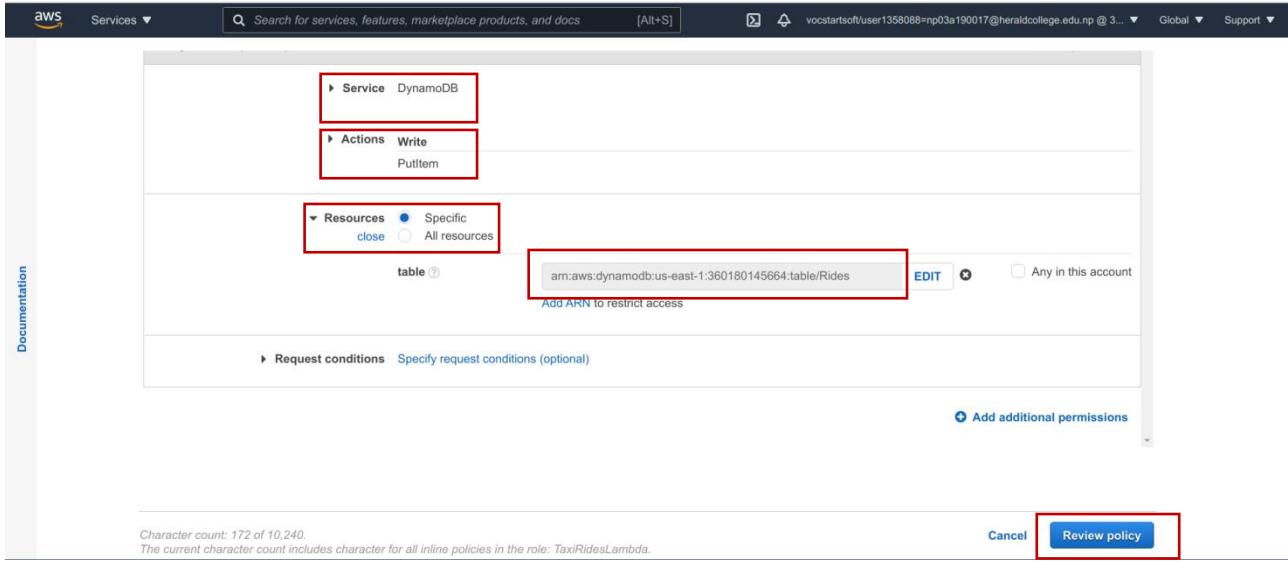
[Attach policies](#) [Add inline policy](#)

Policy name ▾	Policy type ▾
AWSLambdaBasicExecutionRole	AWS managed policy Edit X

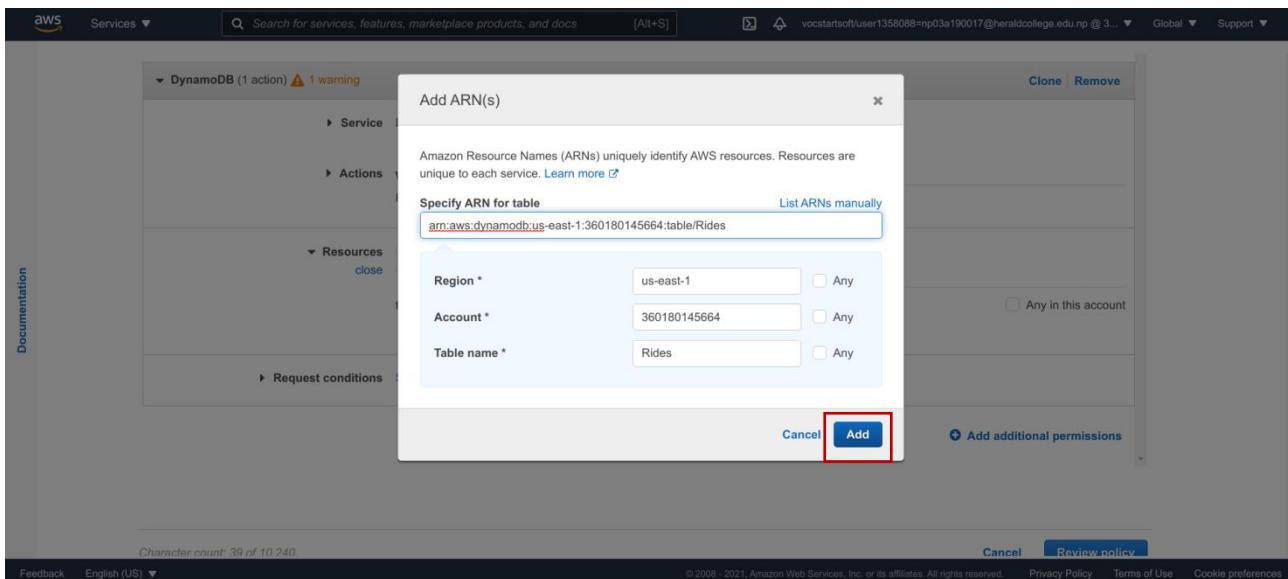
▼ Permissions boundary (not set)

© 2006 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. [Privacy Policy](#) [Terms of Use](#) [Cookie preferences](#)

This is a summary page for the role just created. For adding the permissions for the DynamoDB, we need to add an inline policy. **Inline policies** are policies that the user creates and manages and is embedded directly into a single user, group, or role.



First of all, DynamoDB service is selected and the action that the role needs is the write access. The role is allocated a specific role and to identify the DynamoDB table, its ARN which was noted before is pasted.



The ARN for table helps to identify the AWS Region, Account and table name. It is added into the policy.

Create policy

Review policy

Before you create this policy, provide the required information and review this policy.

Name* Maximum 128 characters. Use alphanumeric and '+', '@', '-' characters.

Summary

Service	Access level	Resource	Request condition
Allow (1 of 276 services) Show remaining 275	DynamoDB	Limited: Write	TableName string like Rides
			None

* Required

Cancel Previous Create policy

Policy is created with the name DynamoDBWriteAccess and is now attached to the lambda function.

Creating a Lambda Function for Handling requests

Lambda function is created so that it would respond to the HTTP request by executing the code in it.

Lambda > Functions > Create function

Create function [Info](#)

Choose one of the following options to create your function.

Author from scratch Start with a simple Hello World example.

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

Container image Select a container image to deploy for your function.

Browse serverless app repository Deploy a sample Lambda application from the AWS Serverless Application Repository.

Basic information

Function name
Enter a name that describes the purpose of your function.

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

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Permissions [Info](#)

Feedback English (US) ▾ © 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use Cookie preferences

A new lambda function named RequestRide is created and the runtime is the Node.js 12.x which is one version behind from the latest release.

The screenshot shows the 'Permissions' step in the AWS Lambda function creation wizard. A red box highlights the 'Change default execution role' section. Under 'Execution role', the 'Use an existing role' option is selected, and 'TaxiRidesLambda' is chosen from the dropdown. Below it, a link 'View the TaxiRidesLambda role on the IAM console.' is visible.

The role which is just created few steps ago i.e. "TaxiRidesLambda" is assigned to the newly created lambda function.

The screenshot shows the AWS Lambda code editor with the file 'index.js' open. A red box highlights the 'File' menu, specifically the 'Save' option under 'Environment'. The code in 'index.js' is a sample Lambda function for managing a fleet of vehicles using AWS SDK and DynamoDB.

```
const randomBytes = require('crypto').randomBytes;
const AWS = require('aws-sdk');
const ddb = new AWS.DynamoDB.DocumentClient();
const fleet = [
  {
    Name: 'John',
    Color: 'Golden',
    Fuel: 'Diesel',
  },
  {
    Name: 'Bob',
    Color: 'White',
    Fuel: 'Diesel',
  },
  {
    Name: 'Diane',
    Color: 'Yellow',
    Fuel: 'Petrol',
  },
];
exports.handler = (event, context, callback) => {
  if (!event.requestContext.authorizer) {
    errorResponse('Authorization not configured', context.awsRequestId, callback);
    return;
  }
  const rideId = toUrlString(randomBytes(16));
  console.log(`Received event ${rideId}, ${event}`);
  // Resumes writing to the CloudWatch Logs subscriber at 1000ms intervals
}
```

The default index.js code is replaced with the code from the provided file RequestRide.js which is pre-populated with the necessary code. The change is now saved.

Validating the implementation

The screenshot shows the AWS Lambda console. A green success message at the top states: "Successfully created the function RequestRide. You can now change its code and configuration. To invoke your function with a test event, choose 'Test'." Below this, the "Test event" section is displayed. A red box highlights the "New event" button. The "Template" dropdown is set to "hello-world". The "Name" field contains "TestRequestEvent". The "Code" pane shows a JSON template for a POST request to "/ride". The "body" field includes a placeholder for a pickup location: "{\"PickupLocation\": {\"Latitude\": 47.62, \"Longitude\": -122.28}}". At the bottom right of the page, a red box highlights the "Invoke" button.

A new test event is created with the provided template for invoking the lambda function.

The screenshot shows the AWS Lambda Code source editor. A green success message at the top states: "The test event TestRequestEvent was successfully saved." Below this, the "Code source" tab is active. A red box highlights the "Test" tab in the navigation bar. The "Execution result" dropdown is set to "TestRequestEvent". The code editor shows an index.js file with a function that handles a POST request to "/ride". The "fleet" variable is defined as an array of three cars: John (Golden, Diesel), Bob (White, Diesel), and Diane (Yellow, Petrol). The "handler" export is defined to check if the request context has an "authorizer" and respond with an error if it's missing. The "Changes deployed" status is shown as green.

After invoking the function will the created test event, the following result is seen:

The screenshot shows the AWS Lambda Test Execution results page. The execution was successful with a status code of 201. The response body contains JSON data representing a taxi ride. The summary table includes details like Request ID, Duration, and Resources configured.

```

    "statusCode": 201,
    "body": "{\"RideId\":\"0Rpf2rWpOrGPN8yIiPeUg\",\"Taxi\":{\"Name\":\"Diane\",\"Color\":\"Yellow\",\"Fuel\":\"Petroil\"},\"TaxiName\":\"Diane\", \"Eta\":\"30 seconds\", \"Rider\":\"the_username\"}",
    "headers": {
        "Access-Control-Allow-Origin": "*"
    }
}

```

Code SHA-256	Request ID
CerbglFuj2zPXzVi74KwRmtlm/ZAXKhamKrMj3Bq5fs=	9d4c9b58-3039-4ffa-91eb-6295dc94c85b

Init duration	Duration
446.20 ms	756.61 ms

Billed duration	Resources configured
757 ms	128 MB

The status code returned is 201. The HTTP 201 Created success status response code indicates that the request has succeeded and has led to the creation of a resource.

The screenshot shows the AWS Lambda Test Execution results page with the 'Changes deployed' tab selected. The function logs are displayed, showing the request event and the processing steps. A red box highlights the log output, which matches the response shown in the previous screenshot.

```

    "statusCode": 201,
    "body": "{\"RideId\":\"0Rpf2rWpOrGPN8yIiPeUg\",\"Taxi\":{\"Name\":\"Bob\",\"Color\":\"White\",\"Fuel\":\"Diesel\"},\"TaxiName\":\"Bob\", \"Eta\":\"30 seconds\", \"Rider\":\"the_username\"}",
    "headers": {
        "Access-Control-Allow-Origin": "*"
    }
}

Function Logs
START RequestId: 4b399c2e-0003-4b3d-903f-16ae7dbf86bd Version: $LATEST
2021-03-31T02:14:43.248Z 4b399c2e-0003-4b3d-903f-16ae7dbf86bd INFO Received event ( QN40Vgvp4aPoiW-34_w1Q ): {
  path: '/ride',
  httpMethod: 'POST',
  headers: {
    Accept: '*/*',
    Authorization: 'eyJraWQiOiJLTzRVMWzs',
    'Content-Type': 'application/json; charset=UTF-8'
  },
  queryStringParameters: null,
  pathParameters: null,
  requestContext: { authorizer: { claims: [Object] } },
  body: {"PickupLocation":{"latitude":47.62,"longitude":-122.28}}
}
2021-03-31T02:14:43.248Z 4b399c2e-0003-4b3d-903f-16ae7dbf86bd INFO Finding taxi for 47.62, -122.28
END RequestId: 4b399c2e-0003-4b3d-903f-16ae7dbf86bd
REPORT RequestId: 4b399c2e-0003-4b3d-903f-16ae7dbf86bd Duration: 744.24 ms Billed Duration: 745 ms Memory Size: 128 MB Max Memory Used: 87 MB Init Durat
4b399c2e-0003-4b3d-903f-16ae7dbf86bd

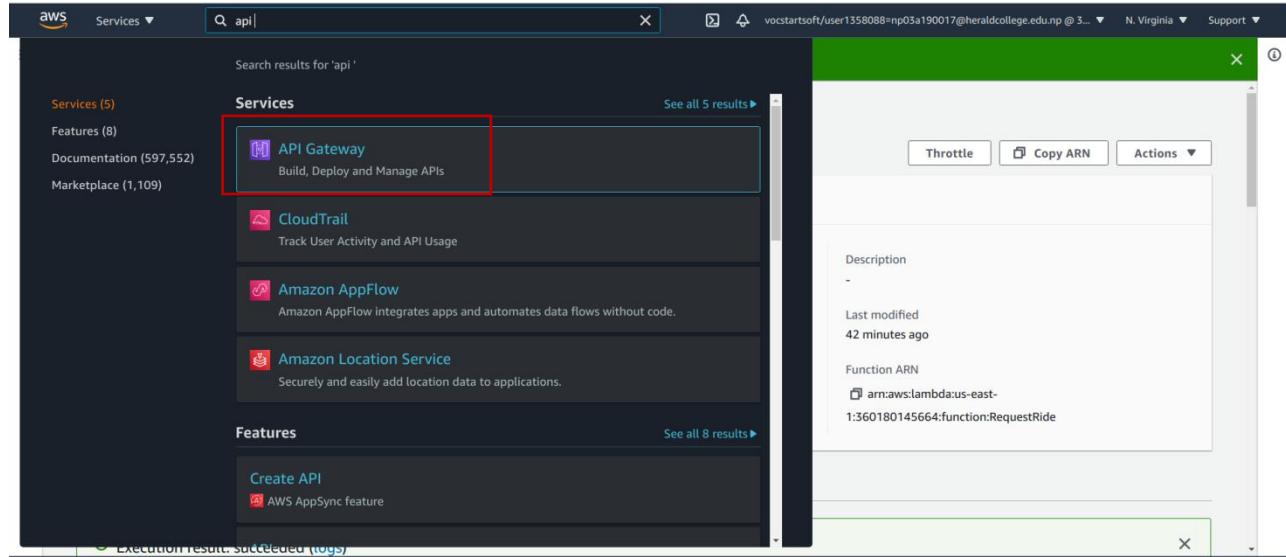
```

It is the same response with the actions that the function performed i.e. the function logs.

Task 2: Serverless Service Backend

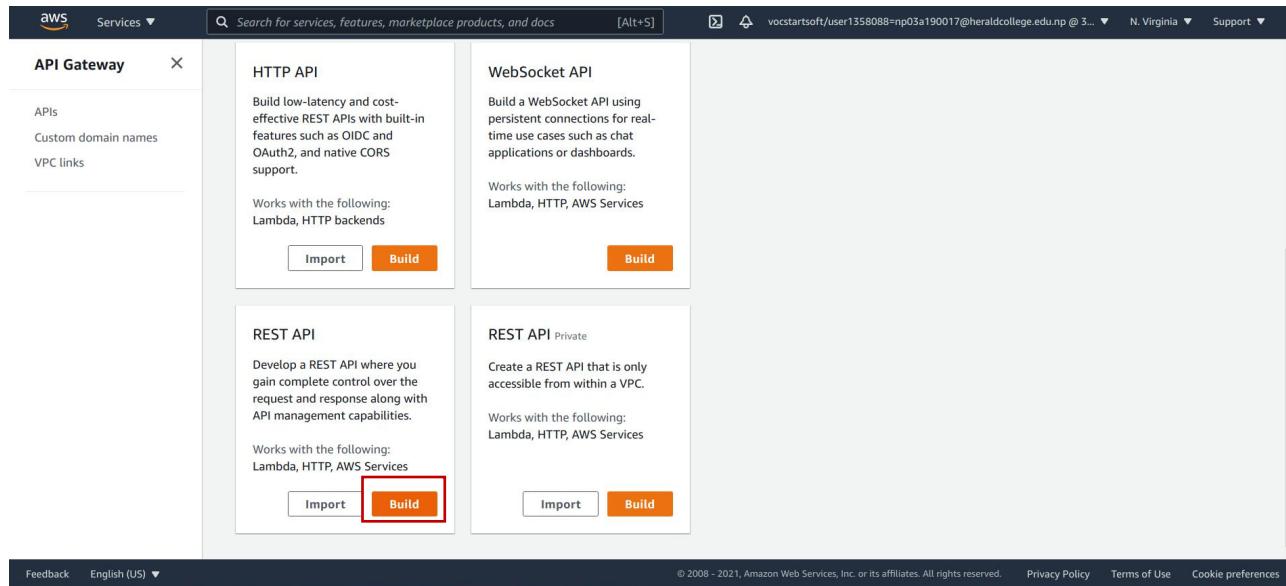
Amazon API Gateway is used here to expose the Lambda function that is built in the previous part as a RESTful API.

Creating a new REST API



The screenshot shows the AWS search interface with the search term 'api' entered. The results are filtered under the 'Services' category. The 'API Gateway' service is highlighted with a red box. To the right, a detailed view of a specific API is shown, including fields for 'Description', 'Last modified' (42 minutes ago), and 'Function ARN' (arn:aws:lambda:us-east-1:360180145664:function:RequestRide). There are also 'Throttle', 'Copy ARN', and 'Actions' buttons.

For creating a new API Gateway, the required service is selected and a new REST API is created through the click of a “Build” button.



The screenshot shows the AWS API Gateway service page. On the left, there's a sidebar with options like 'APIs', 'Custom domain names', and 'VPC links'. The main area displays four API types: 'HTTP API', 'WebSocket API', 'REST API', and 'REST API Private'. The 'REST API' section is expanded, showing its description and supported services (Lambda, HTTP, AWS Services). Below this, there are 'Import' and 'Build' buttons. The 'Build' button for the 'REST API' section is highlighted with a red box. At the bottom, there are links for 'Feedback', 'English (US)', and various AWS footer links.

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API Import from Swagger or Open API 3 Example API

Settings

Choose a friendly name and description for your API.

API name* TaxiRides

Description

Endpoint Type Edge optimized

* Required

Create API

Feedback English (US) © 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use Cookie preferences

A new API is created here where the endpoint type is selected to be edge optimized. Edge-optimized APIs are endpoints that are accessed through a CloudFront distribution that is created and managed by API Gateway.

Creating a Cognito User Pools Authorizer

An authorizer needs to be configured so that the Amazon API Gateway can use the tokens returned by Cognito User Pools to authenticate API calls. The authorizer is named TaxiRides and the token source is “Authorization” as it is made for authorization purposes.

Services ▾ Search for services, features, marketplace products, and docs [Alt+S]

Amazon API Gateway APIs > TaxiRides (c0gerjehzi) > Authorizers Show all hints ?

APIs Custom Domain Names VPC Links

API: TaxiRides Resources Stages Authorizers Gateway Responses Models Resource Policy Documentation Settings Usage Plans API Keys

Authorizers

Authorizers enable you to control access to your APIs using Amazon Cognito User Pools or a Lambda function.

+ Create New Authorizer

Create Authorizer

Name * TaxiRides

Type * Cognito Lambda

Cognito User Pool * us-east-1 / TaxiRides

Token Source * Authorization

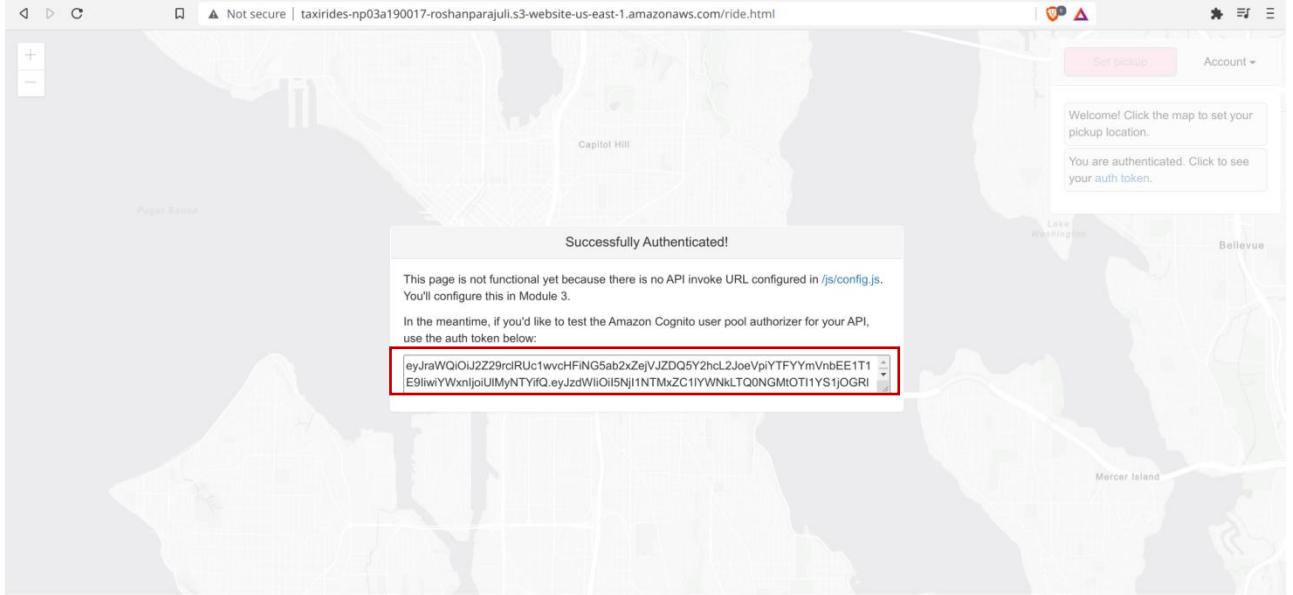
Token Validation

Create Cancel

Feedback English (US) © 2008 - 2021, Amazon Web Services, Inc. or its affiliates. All rights reserved. Privacy Policy Terms of Use Cookie preferences

Verifying the authorizer configuration

The auth token is copied from the /ride.html webpage. The token includes the details of the request including the cognito details and many other details.

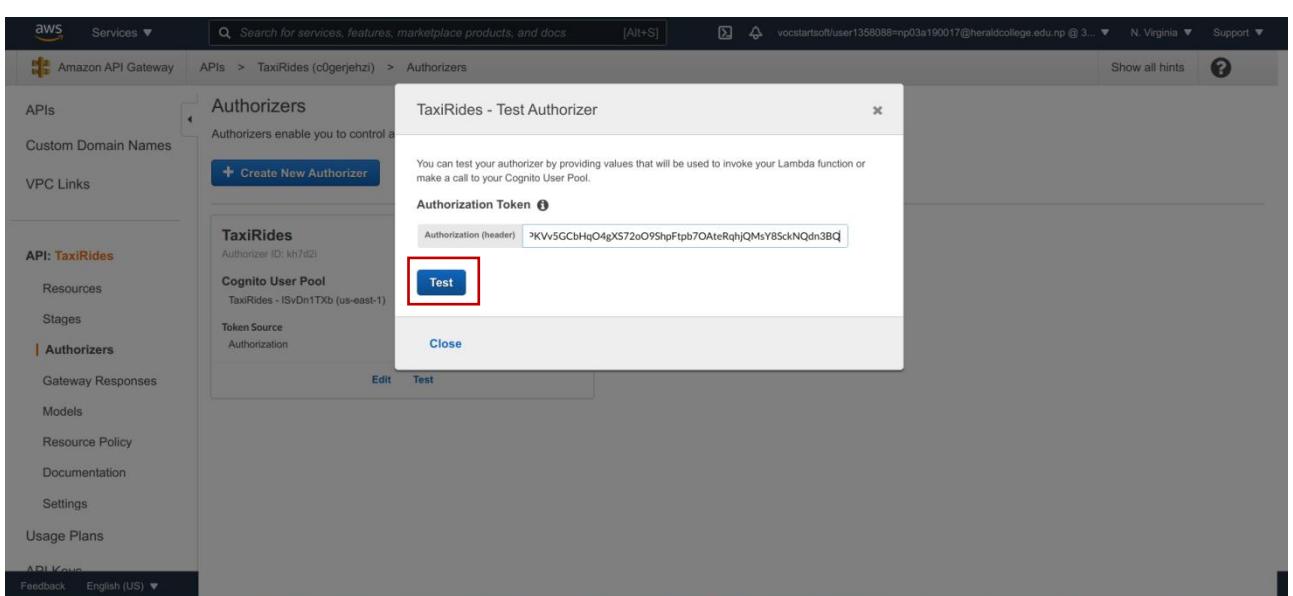


Successfully Authenticated!

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

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

eyJraWQiOiJ2Z29rcjRUC1wvCHFING5ab2xZejVJZDQ5Y2hcL2JoeVpiYTFYYmVnbEE1T1E9liwiYWxnjoIuMyNTYifQ.eyJdWiiOil5NjI1NTMxzC1YWVnLTQ0NGMIOT1YS1jOGRI



TaxiRides - Test Authorizer

You can test your authorizer by providing values that will be used to invoke your Lambda function or make a call to your Cognito User Pool.

Authorization Token

Authorization (header) `>KVv5GCbHqO4gX572oO9ShpFpb7OAtcRqhjQMzYBScKQdn3BQ`

Test

Close

API: TaxiRides

Authorizers

Cognito User Pool

Token Source

Edit Test

After the token is pasted in the authorizer created above, we can observe all the details about the request including response code 200 which means that everything was successful.

The screenshot shows the AWS Lambda console interface. On the left, a sidebar lists 'APIs', 'Custom Domain Names', 'VPC Links', and 'API: TaxiRides' (selected). Under 'API: TaxiRides', there are links for 'Resources', 'Stages', 'Authorizers' (selected), 'Gateway Responses', 'Models', 'Resource Policy', 'Documentation', 'Settings', and 'Usage Plans'. At the bottom of the sidebar are 'Feedback' and 'English (US)'. The main content area shows the 'Authorizers' page for the 'TaxiRides' API, with a sub-menu for 'Authorizers' (selected) and 'Create New Authorizer'. A modal window titled 'TaxiRides - Test Authorizer' is open, showing a JSON response with a status code of 200 and a latency of 12ms. The 'Claims' section displays a large JSON object representing an AWS Cognito ID token. A 'Close' button is at the bottom of the modal.

The details from the token is displayed in the screenshot above.

Creating a new resource and method

A new resource called /ride is created within the api and then create a POST method with the configuration to use a Lambda proxy integration backend by the RequestRide function that was created.

The screenshot shows the AWS Lambda console interface. The sidebar is identical to the previous screenshot, showing 'API: TaxiRides' selected. The main content area shows the 'Resources' page for the 'TaxiRides' API. A modal window titled 'Actions' is open over the resources list, with the 'Create Resource' option highlighted with a red box. Below the modal, the resources list shows a single item: '/ (zuh05cspmd)'. To the right of the resources list, a message says 'No methods defined for the resource.' The 'Actions' modal also includes options for 'Create Method', 'Enable CORS', and 'Edit Resource Documentation'.

APIs > TaxiRides (c0gerjehzi) > Resources > / (zuh05cspmd) > Create

New Child Resource

Configure as [proxy resource](#)

Resource Name*

Resource Path*

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

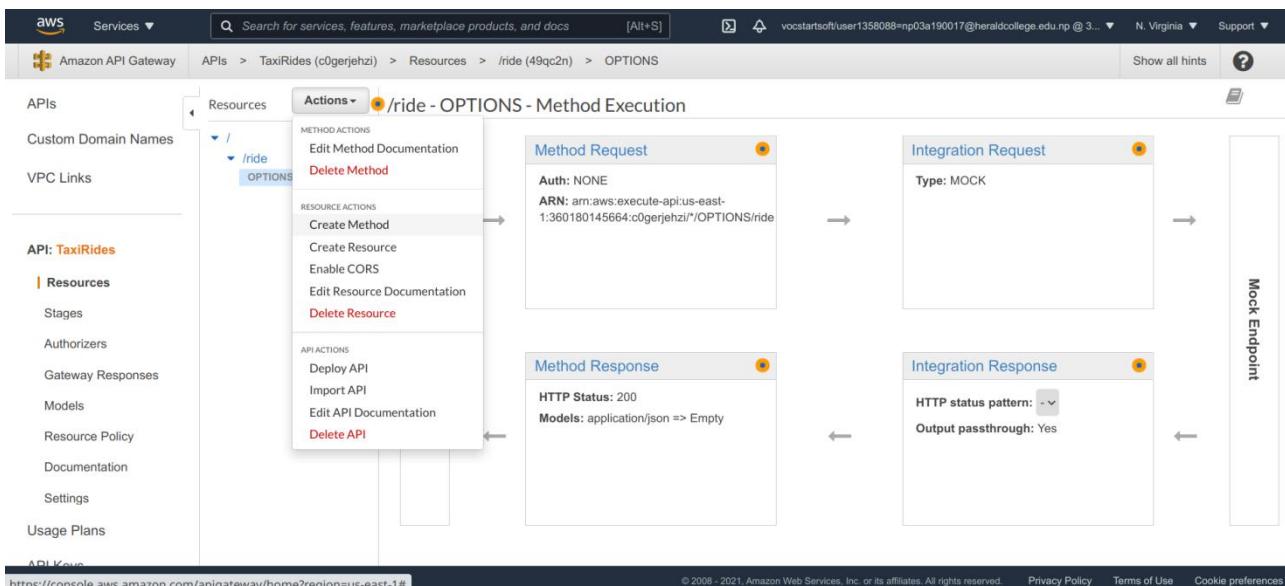
Enable API Gateway CORS

* Required

[Cancel](#) [Create Resource](#)

After a new resource path `/ride` is created, the resource is created.

Now, A new method is created for the POST action which is going to be integrated to the Lambda Function and the Lambda Proxy integration is enabled.



The screenshot shows the AWS Lambda Function integration setup for a POST method. The 'Integration type' is set to 'Lambda Function'. The 'Lambda Region' is 'us-east-1' and the 'Lambda Function' is 'RequestRide'. The 'Use Default Timeout' checkbox is checked. A red box highlights the 'Integration type' dropdown and the 'Lambda Function' input field. A blue box highlights the 'Save' button.

Here the method is integrated to the Lambda Function and the Lambda Proxy integration is enabled.

The screenshot shows the Method Execution settings for the POST method. Under 'Authorization', 'TaxiRides' is selected. Under 'Request Validator', 'NONE' is selected. Under 'API Key Required', 'TaxiRides' is selected. A red box highlights the 'Request Validator' dropdown and the 'API Key Required' dropdown.

The POST method is authorized with the help of the authorizer created above.

Deploying the API

The screenshots illustrate the steps to deploy an API stage:

- Screenshot 1: Deploy API Dialog**

A modal dialog titled "Deploy API" is open. It shows a dropdown for "Deployment stage" set to "[New Stage]" with "prod" selected. There are fields for "Stage name*" and "Stage description". A "Deployment description" field is empty. At the bottom are "Cancel" and "Deploy" buttons.
- Screenshot 2: Stage Settings**

The "Stage Settings" page for the "prod" stage is shown. It includes sections for "Cache Settings" (with "Enable API cache" unchecked), "Default Method Throttling" (with "Enable throttling" checked, "Rate" set to 10000 requests per second, and "Burst" set to 5000 requests), and "Web Application Firewall (WAF)" (with a "Web ACL" dropdown and a "Create Web ACL" button). A "Client Certificate" section allows selecting a certificate for integration endpoints.
- Screenshot 3: prod Stage Editor**

The "prod Stage Editor" page is displayed. It shows the "Invoke URL" as <https://c0gerjehzi.execute-api.us-east-1.amazonaws.com/prod>. The "Settings" tab is selected, showing the same "Cache Settings" and "Default Method Throttling" configurations as the previous screenshot. Other tabs include "Logs/Tracing", "Stage Variables", "SDK Generation", "Export", "Deployment History", "Documentation History", and "Canary".

The API is deployed for production for the website to work with the API correctly.

Updating the Website Config

On the S3 bucket, the config file is updated with the new API invocation URL provided in the previous stage. Also, ride.js file is updated in this stage.

The screenshot shows two views of the AWS S3 console. The top view is the 'Objects' list for a bucket named 'JS/'. It lists several files: 'cognito-auth.js', 'config.js', 'esri-map.js', 'main.js', 'ride.js', 'vendor.js', and a 'vendor/' folder. The 'config.js' file is highlighted with a red box. The bottom view is the 'Upload' interface, where a single file 'config.js' is selected for upload to the same 'JS/' bucket. The 'config.js' file is also highlighted with a red box in this interface.

Name	Type	Last modified	Size	Storage class
cognito-auth.js	js	March 24, 2021, 11:08:00 (UTC+05:45)	5.0 KB	Standard
config.js	js	March 24, 2021, 15:54:41 (UTC+05:45)	360.0 B	Standard
esri-map.js	js	March 24, 2021, 11:07:59 (UTC+05:45)	3.8 KB	Standard
main.js	js	March 24, 2021, 11:07:57 (UTC+05:45)	168.0 B	Standard
ride.js	js	March 24, 2021, 11:08:02 (UTC+05:45)	3.7 KB	Standard
vendor.js	js	March 24, 2021, 11:08:04 (UTC+05:45)	84.8 KB	Standard
vendor/	Folder	-	-	-

Here, the new config file with updation is uploaded.

Upload succeeded
View details below.

Upload: status

The information below will no longer be available after you navigate away from this page.

Summary

Destination	Succeeded	Failed
s3://taxirides-np03a190017-roshanparajuli/js/	0 files, 0 B (0%)	0 files, 0 B (0%)

Files and folders Configuration

Files and folders (1 Total, 419.0 B)

Find by name
< 1 >

All files and folders in this table will be uploaded.

Name	Folder	Type	Size
ride.js	-	text/javascript	3.7 KB

Destination

Destination
s3://taxirides-np03a190017-roshanparajuli/js/

Destination details

The following bucket settings impact new objects stored in the specified destination.

Bucket Versioning When enabled, multiple variants of an object can be stored in the bucket to easily recover from unintended user actions and application failures. Learn more	Enabled
Default encryption When enabled, new objects stored in this bucket are automatically encrypted. Learn more	Disabled
Object Lock When enabled, objects in this bucket might be prevented from being deleted or overwritten for a fixed amount of time or indefinitely. Learn more	Disabled

► Additional upload options

Cancel **Upload**

The config file and ride file is updated successfully.

Validating the implementation

The following is the ride.html configured properly.

