

Serverless 2FA Token Delivery Using AWS

TABLE OF CONTENTS

1. PROBLEM STATEMENT	3
2. CORE SERVICES	3
3. THE SOLUTION.....	3
4. THE ARCHITECTURE	4
5. SYSTEM DEMONSTRATION	5
6. CODE ARTEFACTS	9
7. S3 FRONTEND ARTEFACTS	11
8. SECURITY CONSIDERATIONS.....	14

1. Problem Statement

The task is to design a backend system for generating single-use tokens to support a two-factor authentication (2FA) workflow. The system assumes that users and their credentials are valid, and the second factor of authentication needs to be triggered.

The solution should prioritize simplicity, cost-effectiveness, security, and usability. The design should meet the following core requirements:

- Entirely built on AWS.
- Leverages serverless architecture.
- Is resilient and can handle failures gracefully.
- Is modular to enable easy extensibility and maintenance.

Additional deliverables for enhanced evaluation include:

- An architecture diagram outlining the system design.
- Code snippets or configuration details illustrating key implementation components.
- Metrics to assess system health, availability, and performance.
- Adherence to security best practices.

The expected output is a written document (in MS Word or similar format) detailing the architecture and implementation steps for the proposed solution.

2. Core Services

Here is a breakdown of the core AWS Services used.

AWS Service	Intended Use
AWS DynamoDB	Used to store the generated tokens with a TTL (time-to-live) feature to automatically expire tokens after 5 minutes.
AWS Lambda	A stateless function generates, stores, and dispatches the token.
AWS API Gateway	API Gateway provides a secure HTTP endpoint to invoke Lambda function. It also allows to modularise the function and can be published to end customers for integration.
IAM	To give the lambda function permission to access the required AWS. This policy allows Lambda functions to perform CRUD operations on a DynamoDB table.
S3	Acts a S3 frontend to invoke the Lambda function. Since the problem statement didn't include a frontend, I thought of having a frontend to showcase the overall functionality of using API Gateway to create a HTTP endpoint that invokes a lambda function.

By combining these AWS services, the architecture provides a seamless way to interact with the database through serverless functions, enhancing scalability and reducing overhead.

3. The Solution

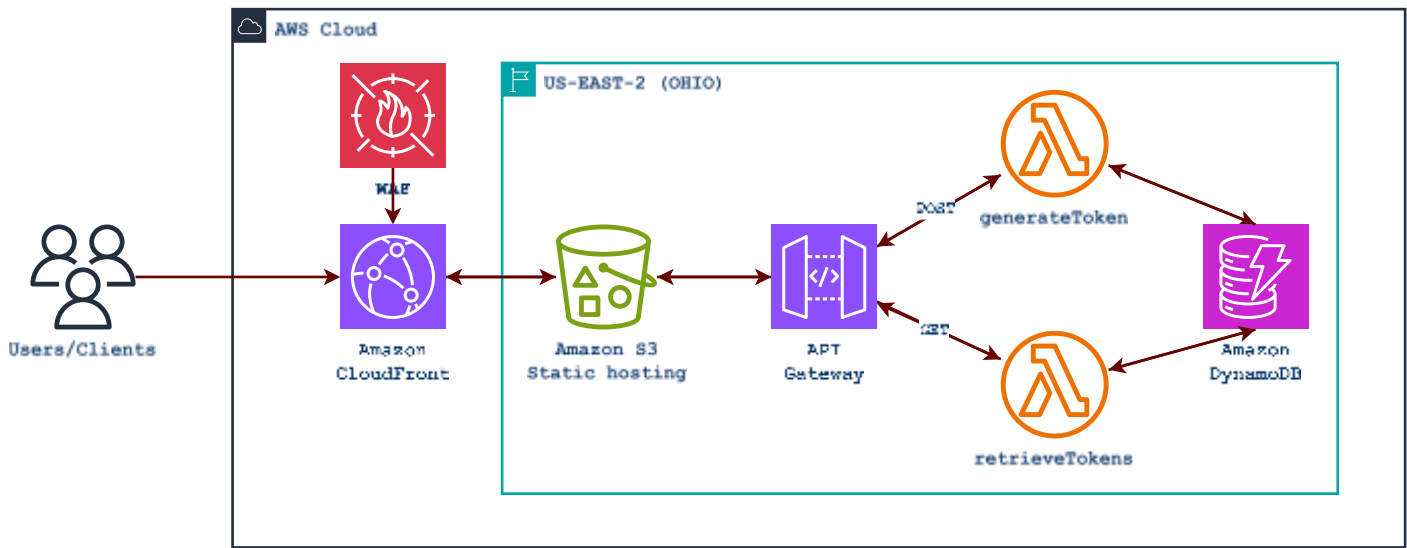
Serverless computing enables applications to run without the need to manage underlying infrastructure, eliminating the tasks of provisioning, scaling, and maintaining servers for applications and databases. AWS Lambda exemplifies this with its serverless compute platform designed to execute functions, often referred to as "Function as a Service."

In this solution, we will create a REST API to invoke a Lambda function via HTTP requests. The Lambda function will perform CRUD (create, read, update) operations on a DynamoDB table creating temporary access tokens. These access tokens will be valid for 300s before automatically expiring.

Amazon API Gateway will serve as the secure HTTP endpoint for invoking the Lambda function. The solution involves setting up a Lambda function, creating a REST API in API Gateway, and configuring a DynamoDB table to test Lambda functionality. After deploying the API, we will be able to create & read the tokens generated. These tokens will automatically expiry at the end of 5 mins and record will be removed from DynamoDB.

This implementation is a serverless, modular, and low-cost 2FA backend system built entirely on AWS.

4. The Architecture



Components	Intended Use
User Authentication Trigger	Once the primary authentication is verified, a trigger (e.g., HTTP API call) initiates the 2FA workflow. Right now this trigger happens with a click of a button on the S3 frontend.
Token Generation	Use AWS Lambda to generate a secure, random single-use token. The Lambda function ensures the token has sufficient entropy and is stored securely for verification.
DynamoDB	Use AWS Lambda to generate a secure, random single-use token. The Lambda function ensures the token has sufficient entropy and is stored securely for verification.
Token Delivery	To give the lambda function permission to access the required AWS. This policy allows Lambda functions to perform CRUD operations on a DynamoDB table.

5. System Demonstration

Open the S3 frontend URL hosted behind AWS CloudFront distribution.

dkhrs8ptq95sy.cloudfront.net

Generate & Retrieve Security Token

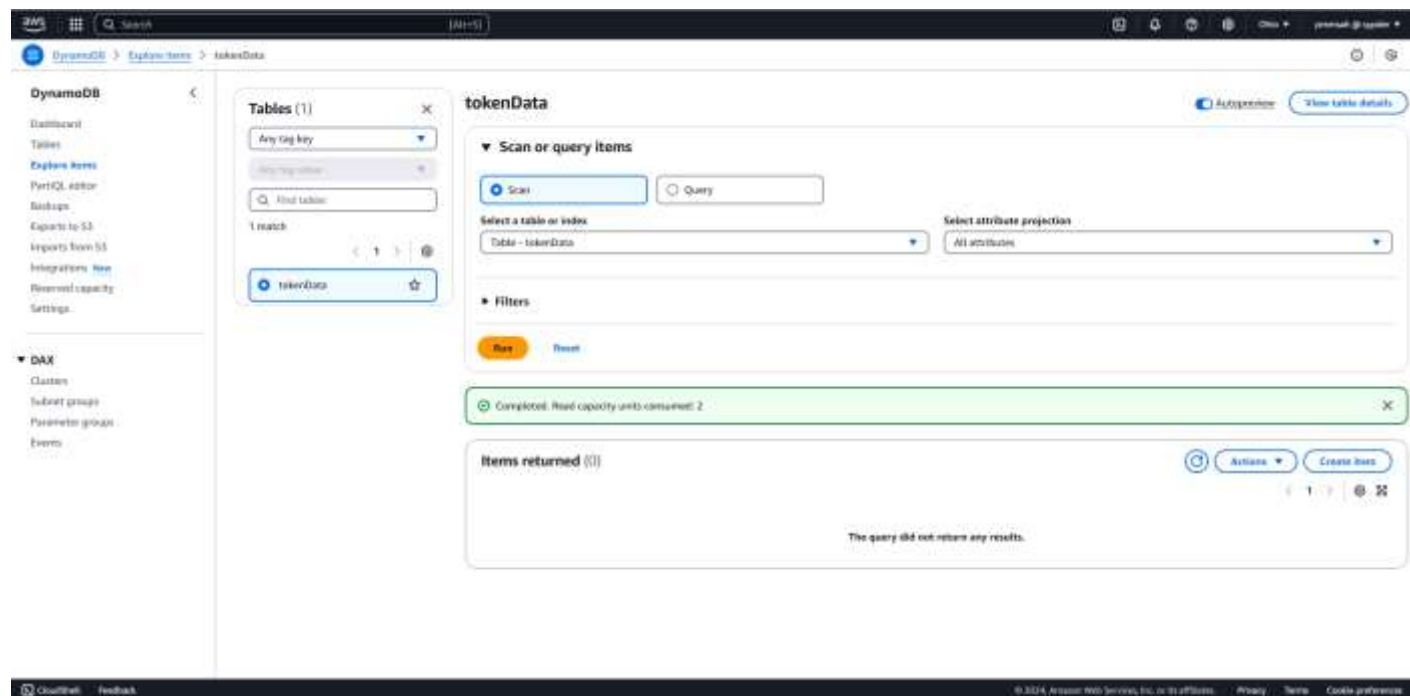
User ID:

Generate Security Token

View issued Security Token

User ID	Token	Expiry Time
---------	-------	-------------

Currently there are no records in the DynamoDB table.



Let's Input the User ID, premjeet.sahu and click on Generate Security Token

← → ↻ 🔍 dkhrs8ptq95sy.cloudfront.net ☆ 📁 👤 ⋮

Generate & Retrieve Security Token

User ID:

Generate Security Token

View issued Security Token

User ID	Token	Expiry Time
---------	-------	-------------

Let's Input the User ID, premjeet.sahu and click on Generate Security Token. This generate a random 15 char token and prints on the screen. The security token generation process can be triggered on any event.

← → ↻ 🔍 dkhrs8ptq95sy.cloudfront.net ☆ 📁 👤 ⋮

Generate & Retrieve Security Token

User ID:

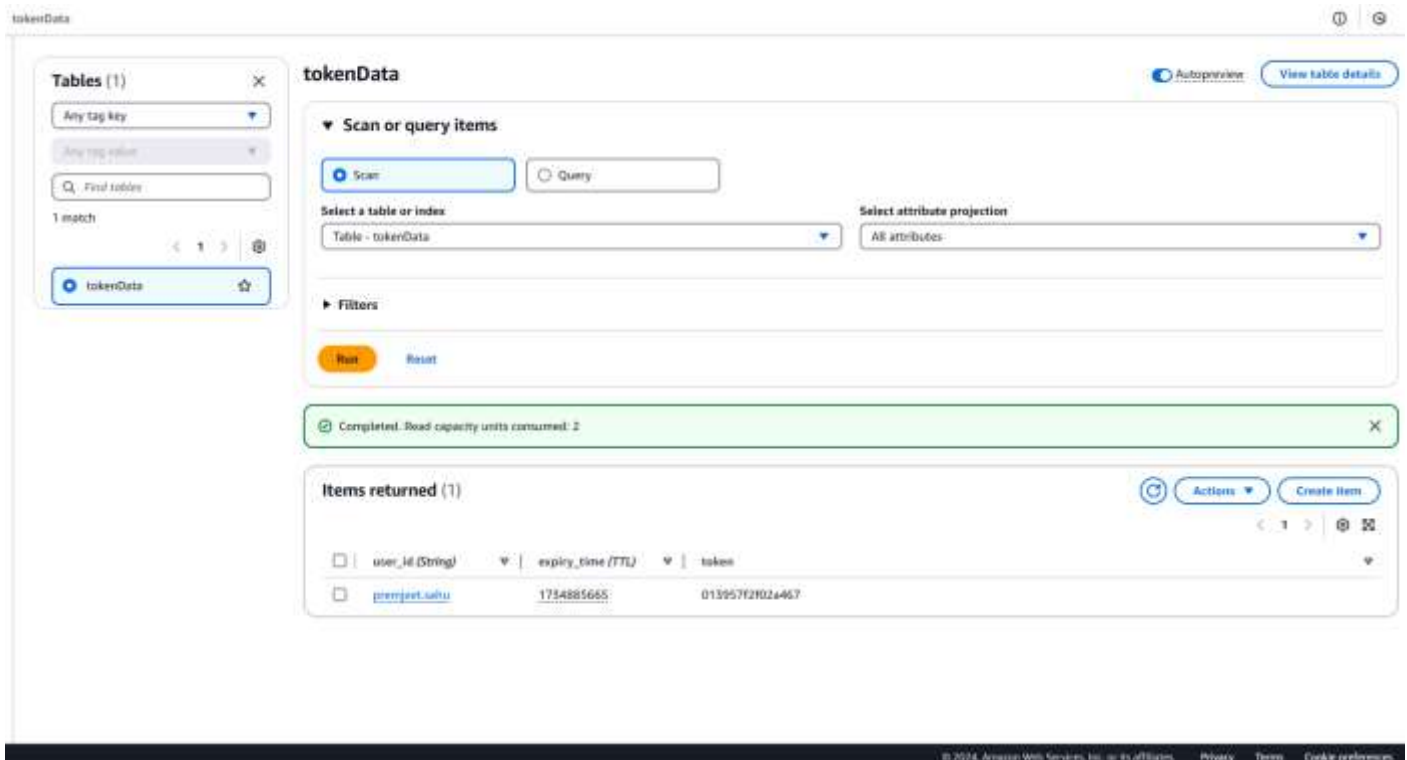
Generate Security Token

Token generated successfully. Token#: '013957f2f02a467' expires in 5 minutes.

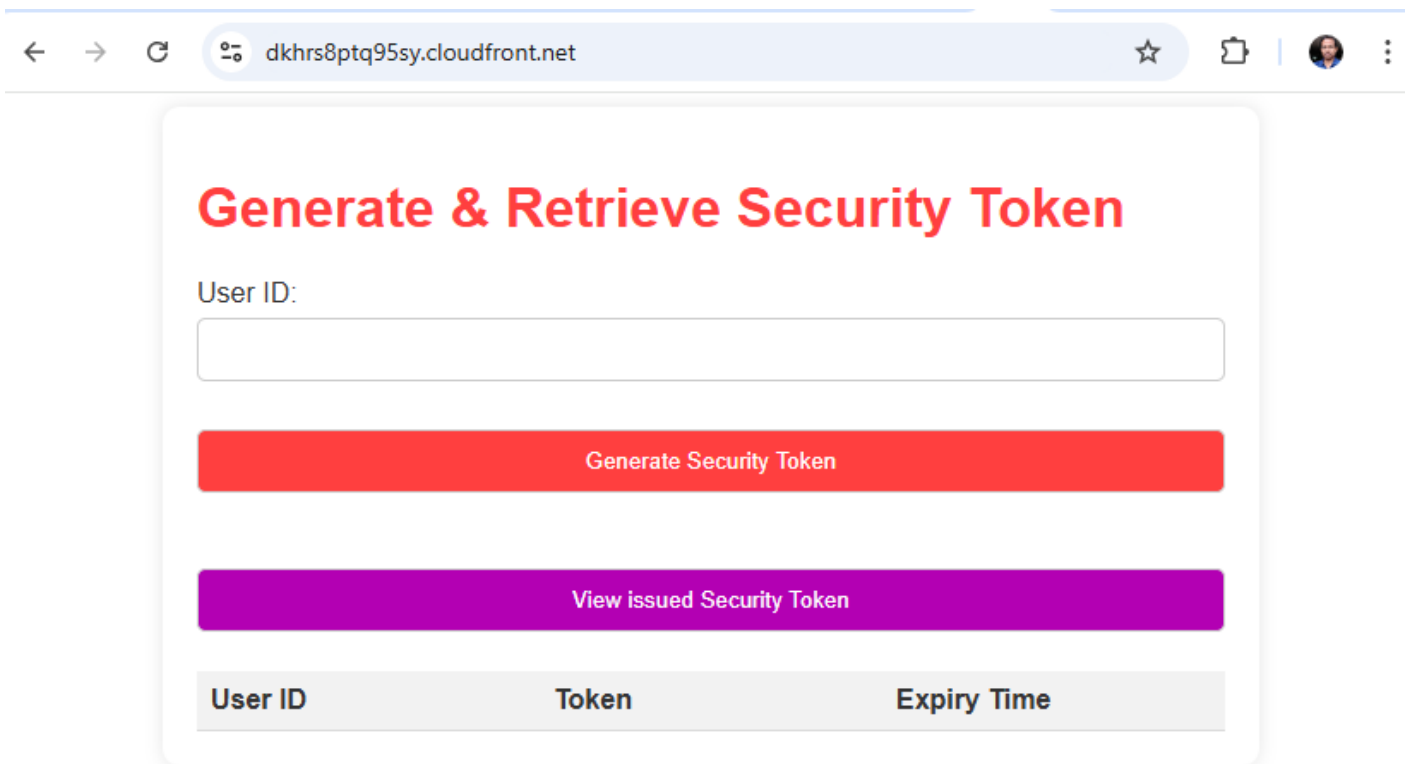
View issued Security Token

User ID	Token	Expiry Time
---------	-------	-------------

We can see a record is added to the DynamoDB with the token and time to expiry.



To view existing token generated, click on View Issued security token button.



This will list all the tokens that are generated and stored in the database.

dkhrs8ptq95sy.cloudfront.net

☆

Generate & Retrieve Security Token

User ID:

Generate Security Token

View issued Security Token

User ID	Token	Expiry Time
premjeet.sahu	013957f2f02a467	1734885665

After a span of 5 mins, we can see the token has automatically been removed from DynamoDB

dkhrs8ptq95sy.cloudfront.net

☆

Generate & Retrieve Security Token

User ID:

test.test

Generate Security Token

Token generated successfully. Token#: '8086f957c17f42e' expires in 5 minutes.

View issued Security Token

User ID	Token	Expiry Time
---------	-------	-------------

6. Code Artefacts

Generate 2FA.py

```
import boto3
import os
import uuid
from botocore.exceptions import ClientError
from datetime import datetime, timedelta

# Constants
TOKEN_TTL_SECONDS = 300 # Tokens expire after 5 minutes
DYNAMODB_TABLE = os.environ["DYNAMODB_TABLE"]

# Initialize AWS clients
dynamodb = boto3.client("dynamodb")

def generate_token():
    """Generates a random single-use token."""
    return str(uuid.uuid4()).replace("-", "")[:15]

def save_token(user_id, token):
    """Saves the token in DynamoDB."""
    expiry_time = int((datetime.utcnow() +
timedelta(seconds=TOKEN_TTL_SECONDS)).timestamp())
    try:
        dynamodb.put_item(
            TableName=DYNAMODB_TABLE,
            Item={
                "user_id": {"S": user_id},
                "token": {"S": token},
                "expiry_time": {"N": str(expiry_time)}
            }
        )
    except ClientError as e:
        print(f"Error saving token: {e}")
        raise

def lambda_handler(event, context):
    """Lambda handler to generate and save 2FA tokens."""
    user_id = event["user_id"]

    # Generate a new token
    token = generate_token()

    # Save the token in the database
    save_token(user_id, token)

    return {
        "statusCode": 200,
        "body": "Token generated successfully. Token#: '" + token + "' expires
in 5 minutes"
    }
```

Retrieve2FAToken.py

```
import json
```

Author: Premjeet Sahu

```

import boto3

def lambda_handler(event, context):
    # Initialize a DynamoDB resource object for the specified region
    dynamodb = boto3.resource('dynamodb', region_name='us-east-2')

    # Select the DynamoDB table named 'studentData'
    table = dynamodb.Table('tokenData')

    # Scan the table to retrieve all items
    response = table.scan()
    data = response['Items']

    # If there are more items to scan, continue scanning until all items
    # are retrieved
    while 'LastEvaluatedKey' in response:
        response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
        data.extend(response['Items'])

    # Return the retrieved data
    return data

```

IAM Policy

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ListAndDescribe",
      "Effect": "Allow",
      "Action": [
        "dynamodb:List*",
        "dynamodb:DescribeReservedCapacity*",
        "dynamodb:DescribeLimits",
        "dynamodb:DescribeTimeToLive"
      ],
      "Resource": "arn:aws:dynamodb:us-east-2:XXXXXXXXXXXX:table/tokenData"
    },
    {
      "Sid": "SpecificTable",
      "Effect": "Allow",
      "Action": [
        "dynamodb:BatchGet*",
        "dynamodb:DescribeStream",
        "dynamodb:DescribeTable",
        "dynamodb:Get*",
        "dynamodb:Query",
        "dynamodb:Scan",
        "dynamodb:BatchWrite*",
        "dynamodb:CreateTable",
        "dynamodb>Delete*",
        "dynamodb:Update*",

```

```

        "dynamodb:PutItem"
      ],
      "Resource": "arn:aws:dynamodb:us-east-2:329599624450:table/tokenData"
    }
  ]
}

```

7. S3 Frontend Artefacts

Below two files Index.html and script.js needs to be placed in a S3 bucket with hosting enabled.

Index.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="securitytokenaap" content="width=device-width, initial-scale=1.0">
  <title>Security Token App</title>
  <style>
    body {
      background-color: #ffffff; /* Light gray background */
      color: #333; /* Dark gray text */
      font-family: Arial, sans-serif; /* Use Arial font */
    }

    h1 {
      color: #ff0000c0; /* Blue heading text */
    }

    .container {
      max-width: 600px; /* Limit width to 600px */
      margin: 0 auto; /* Center the container */
      padding: 20px; /* Add padding */
      background-color: #fff; /* White background */
      border-radius: 10px; /* Rounded corners */
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1); /* Add shadow */
    }

    input[type="text"], input[type="submit"] {
      width: 100%;
      padding: 10px;
      margin: 5px 0;
      box-sizing: border-box;
      border: 1px solid #ccc;
      border-radius: 5px;
    }

    input[type="submit"] {
      background-color: #ff0000c0; /* Blue submit button */
      color: #fff; /* White text */
      cursor: pointer; /* Add pointer cursor */
    }

    input[type="submit"]:hover {
      background-color: #b300b3; /* Darker blue on hover */
    }
  </style>

```

```

        table {
            width: 100%;
            border-collapse: collapse;
        }

        th, td {
            padding: 8px;
            text-align: left;
            border-bottom: 1px solid #ddd;
        }

        th {
            background-color: #f2f2f2; /* Light gray header background */
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Generate & Retrieve Security Token</h1>
        <label for="user_id">User ID:</label><br>
        <input type="text" name="user_id" id="user_id"><br>

        <br>
        <input type="submit" id="generateToken" value="Generate Security
Token">
        <p id="tokenSaved"></p>

        <br>
        <input type="submit" id="retrieveToken" value="View issued Security
Token">
        <br><br>
        <div id="showTokens">
            <table id="tokenData">
                <colgroup>
                    <col style="width:20%">
                    <col style="width:20%">
                    <col style="width:20%">
                </colgroup>
                <thead>
                    <tr>
                        <th>User ID</th>
                        <th>Token</th>
                        <th>Expiry Time</th>
                    </tr>
                </thead>
                <tbody>
                    <!-- Student data will be displayed here -->
                </tbody>
            </table>
        </div>
    </div>

    <script src="scripts.js"></script>
    <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.6.0/jquery.min.js"></scri
pt>
</body>
</html>

```

Script.js

```
var API_ENDPOINT = "https://g3fnf5lxl5.execute-api.us-east-2.amazonaws.com/Production";

// AJAX POST request to generate token data
document.getElementById("generateToken").onclick = function(){
    var inputData = {
        "user_id": $('#user_id').val()
    };
    $.ajax({
        url: API_ENDPOINT,
        type: 'POST',
        data: JSON.stringify(inputData),
        contentType: 'application/json; charset=utf-8',
        success: function (response) {
            document.getElementById("tokenSaved").innerHTML = "Token
Generated!";
        },
        error: function () {
            alert("Error generating security token.");
        }
    });
}

// AJAX GET request to retrieve all tokens
document.getElementById("retrieveToken").onclick = function(){
    $.ajax({
        url: API_ENDPOINT,
        type: 'GET',
        contentType: 'application/json; charset=utf-8',
        success: function (response) {
            $('#tokenData tr').slice(1).remove();
            jQuery.each(response, function(i, data) {
                $('#tokenData').append("<tr> \
                <td>" + data['user_id'] + "</td> \
                <td>" + data['token'] + "</td> \
                <td>" + data['expiry_time'] + "</td> \
                </tr>");
            });
        },
        error: function () {
            alert("Error retrieving security token.");
        }
    });
}
```


IAM Role

Serverless-2FA-Lambda-Execution-Role Info

Allows Lambda functions to call AWS services on your behalf.

Summary

Creation date
December 22, 2024, 11:41 (UTC+05:30)

Last activity
 14 hours ago

Permissions

Trust relationships




Tags

Last Accessed

Revoke sessions

Permissions policies (2) Info

You can attach up to 10 managed policies.

<input type="checkbox"/>	Policy name <small>?</small>	Type
<input type="checkbox"/>	  AWSLambdaDynamoDBExecutionRole	AWS managed
<input type="checkbox"/>	 Serverless-2FA-DynamoDB-Policy	Customer managed

8. Security Considerations

- Used AWS Key Management Service (**KMS**) to encrypt tokens stored in DynamoDB.
- Ensured HTTPS is enforced for API Gateway endpoints.
- Applied IAM policies with the principle of least privilege for Lambda and other services.
- Set a short expiration time for tokens (e.g., 5-10 minutes).
- Integrated WAF with CloudFront to prevent against malicious attacks