# **Deploying AWS Backend for Secure S3 Presigned URLs**

This guide will walk you through setting up the AWS Lambda function and API Gateway endpoint necessary to generate real presigned URLs for your interactive guide. This will enable the "Interactive Demo" section of your web application to request and receive actual time-limited download links from your AWS environment.

**Before You Begin:**

* Ensure you have an AWS account and have configured your S3 bucket as outlined in the "Secure S3 File Transfer & Temporary Access Documentation" (specifically, remember your S3 bucket name and its region).
* You will need the **IAM Role ARN** that you created for your backend (e.g., S3FileTransferLambdaRole).

## **Step 1: Create the AWS Lambda Function**

This Lambda function will house the logic for generating the presigned S3 URLs.

1. **Go to the AWS Lambda Console:**
   * Sign in to the AWS Management Console and navigate to the **Lambda** service.
2. **Create a New Function:**
   * Click the **"Create function"** button.
   * Choose **"Author from scratch"**.
   * **Function name:** S3PresignedUrlGenerator (or a name of your choice).
   * **Runtime:** Node.js 20.x (or the latest LTS Node.js version).
   * **Architecture:** x86\_64 (default).
   * **Execution role:**
     + Choose **"Use an existing role"**.
     + Select the IAM role you created previously (e.g., S3FileTransferLambdaRole). This role needs permissions for s3:GetObject and s3:PutObject on your S3 bucket, and logs:CreateLogGroup, logs:CreateLogStream, logs:PutLogEvents for CloudWatch Logs.
   * Click **"Create function"**.
3. **Configure the Lambda Function Code:**
   * Once the function is created, you'll be on its configuration page. Scroll down to the **"Code source"** section.
   * Replace the existing index.js content with the following Node.js code. This code generates a presigned URL for getObject (download).

const AWS = require('aws-sdk');  
const S3 = new AWS.S3({ signatureVersion: 'v4', region: process.env.AWS\_REGION });  
  
exports.handler = async (event) => {  
 // Retrieve the S3 bucket name from environment variables  
 const bucketName = process.env.S3\_BUCKET\_NAME;  
 // The object key (file path/name) will come from the API Gateway query parameters  
 const objectKey = event.queryStringParameters ? event.queryStringParameters.key : null;  
 // URL expiration in seconds (e.g., 5 minutes = 300 seconds)  
 const expiresSeconds = 300;  
 // Default action is 'getObject' for download. Can be extended for 'putObject' for upload.  
 const action = event.queryStringParameters && event.queryStringParameters.action ? event.queryStringParameters.action : 'getObject';  
  
 // Basic validation for the object key  
 if (!objectKey) {  
 return {  
 statusCode: 400,  
 headers: {  
 'Access-Control-Allow-Origin': '\*', // IMPORTANT: Adjust for production to specific origins  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify({ message: 'Missing "key" query parameter for the S3 object.' })  
 };  
 }  
  
 // Validate the action type  
 if (!['getObject', 'putObject'].includes(action)) {  
 return {  
 statusCode: 400,  
 headers: {  
 'Access-Control-Allow-Origin': '\*',  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify({ message: 'Invalid "action" specified. Must be "getObject" or "putObject".' })  
 };  
 }  
  
 try {  
 const params = {  
 Bucket: bucketName,  
 Key: objectKey,  
 Expires: expiresSeconds  
 };  
  
 // Generate the presigned URL based on the action  
 const url = await S3.getSignedUrlPromise(action, params);  
  
 return {  
 statusCode: 200,  
 headers: {  
 'Access-Control-Allow-Origin': '\*', // IMPORTANT: Adjust for production to specific origins  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify({ presignedUrl: url, key: objectKey, action: action })  
 };  
 } catch (error) {  
 console.error('Error generating presigned URL:', error);  
 return {  
 statusCode: 500,  
 headers: {  
 'Access-Control-Allow-Origin': '\*',  
 'Content-Type': 'application/json'  
 },  
 body: JSON.stringify({ message: 'Failed to generate presigned URL.', error: error.message })  
 };  
 }  
};

1. **Add Environment Variables:**
   * Still on the function's configuration page, scroll down to the **"Configuration"** tab.
   * Click on **"Environment variables"** in the left menu.
   * Click **"Edit"**.
   * Add a new environment variable:
     + **Key:** S3\_BUCKET\_NAME
     + **Value:** Your S3 bucket name (e.g., my-secure-file-transfer-bucket-your-name).
   * Click **"Save"**.
2. **Deploy the Lambda Function:**
   * After making changes to the code or environment variables, click the orange **"Deploy"** button at the top right of the Code source section.

## **Step 2: Create the API Gateway Endpoint**

This will provide an HTTPS endpoint for your frontend application to call the Lambda function.

1. **Go to the AWS API Gateway Console:**
   * Sign in to the AWS Management Console and navigate to the **API Gateway** service.
2. **Create a New API:**
   * Choose **"REST API"** (not "WebSocket" or "HTTP API" for this example, as REST offers more granular control for beginners).
   * Click **"Build"** under "REST API".
   * **Choose the protocol:** Select **"New API"**.
   * **API name:** S3PresignedUrlApi (or a name of your choice).
   * **Endpoint type:** Regional (default, suitable for most cases).
   * Click **"Create API"**.
3. **Create a Resource:**
   * In the API Gateway console, under your new API, go to **"Resources"** in the left menu.
   * Click on **"Actions"** dropdown, then **"Create Resource"**.
   * **Resource Name:** presigned-url (or link).
   * **Resource Path:** /presigned-url (or /link).
   * Click **"Create Resource"**.
4. **Create a GET Method:**
   * Select the newly created /presigned-url resource.
   * Click **"Actions"** dropdown, then **"Create Method"**.
   * Select **"GET"** from the dropdown and click the checkmark.
   * **Integration type:** Select **"Lambda Function"**.
   * **Use Lambda Proxy integration:** Check this box (simplifies data passing).
   * **Lambda Region:** Select the region where you created your Lambda function.
   * **Lambda Function:** Start typing the name of your Lambda function (e.g., S3PresignedUrlGenerator) and select it from the dropdown.
   * Click **"Save"**.
   * You will be prompted to grant API Gateway permissions to invoke your Lambda function. Click **"OK"**`.
5. **Enable CORS (Cross-Origin Resource Sharing):**
   * This is crucial for your GitHub Pages frontend to be able to call this API.
   * Select the /presigned-url resource.
   * Click **"Actions"** dropdown, then **"Enable CORS"**.
   * For **"Access-Control-Allow-Headers"**, you can leave the defaults ('Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token').
   * For **"Access-Control-Allow-Methods"**, ensure GET and OPTIONS are selected. You might also add POST if you plan to extend this for upload link generation.
   * For **"Access-Control-Allow-Origin"**:
     + **For development/testing:** You can use \* (asterisk) to allow all origins. This is **NOT recommended for production**.
     + **For production:** Replace \* with the specific domain of your GitHub Pages site (e.g., https://yourusername.github.io).
   * Click **"Enable CORS and replace existing CORS headers"**.
   * Click **"Yes, replace existing values"** if prompted.
6. **Deploy the API:**
   * In the API Gateway console, click **"Actions"** dropdown, then **"Deploy API"**.
   * **Deployment stage:** Select **"[New Stage]"**.
   * **Stage name:** prod (or dev, v1, etc.).
   * Click **"Deploy"**.
7. **Get the Invoke URL:**
   * After deployment, you'll be redirected to the "Stages" page.
   * Click on your deployed stage (e.g., prod) in the left menu.
   * You'll see the **"Invoke URL"** at the top. This is the base URL for your API. It will look something like https://xxxxxxxxxx.execute-api.your-region.amazonaws.com/prod.

## **Step 3: Update Your Frontend (index.html)**

Now you need to tell your interactive guide (the index.html file) where your new API Gateway endpoint is.

1. **Open your index.html file** (from your local Git clone).
2. Locate the JavaScript section at the bottom of the file.
3. Find the generateBtn.addEventListener('click', ...) function.
4. **Replace the existing presigned URL generation logic** inside the click event listener with a fetch call to your new API Gateway endpoint.  
   **Original (simulated) code in index.html:**  
   generateBtn.addEventListener('click', () => {  
    const bucketName = 'my-secure-file-transfer-bucket';  
    const region = 'us-east-1';  
    const selectedFile = fileSelect.value;  
    const timestamp = new Date().toISOString().replace(/[:.]/g, '-');  
    const fakeSignature = Math.random().toString(36).substring(2, 15) + Math.random().toString(36).substring(2, 15);  
    const fakeAccessKey = 'ASIA' + Math.random().toString(36).substring(2, 18).toUpperCase();  
     
    const presignedUrl = `https://${bucketName}.s3.${region}.amazonaws.com/${selectedFile}?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=${fakeAccessKey}%2F${timestamp.substring(0,8)}%2F${region}%2Fs3%2Faws4\_request&X-Amz-Date=${timestamp}&X-Amz-Expires=300&X-Amz-SignedHeaders=host&X-Amz-Signature=${fakeSignature}`;  
     
    urlOutput.textContent = presignedUrl;  
    demoOutput.style.display = 'block';  
   });  
     
   **New (real) code for index.html:**  
   generateBtn.addEventListener('click', async () => { // Add 'async' here  
    const apiGatewayEndpoint = 'YOUR\_API\_GATEWAY\_INVOKE\_URL\_HERE'; // <--- REPLACE THIS  
    const selectedFileKey = fileSelect.value; // e.g., 'reports/quarterly-earnings.pdf'  
     
    if (apiGatewayEndpoint === 'YOUR\_API\_GATEWAY\_INVOKE\_URL\_HERE') {  
    urlOutput.textContent = "Please update 'YOUR\_API\_GATEWAY\_INVOKE\_URL\_HERE' in the JavaScript with your actual API Gateway Invoke URL.";  
    demoOutput.style.display = 'block';  
    return;  
    }  
     
    urlOutput.textContent = 'Generating real presigned URL...';  
    demoOutput.style.display = 'block';  
     
    try {  
    // Construct the URL to your API Gateway endpoint, passing the S3 key as a query parameter  
    const response = await fetch(`${apiGatewayEndpoint}/presigned-url?key=${encodeURIComponent(selectedFileKey)}`);  
     
    if (!response.ok) {  
    const errorData = await response.json();  
    throw new Error(`Error from backend: ${errorData.message || response.statusText}`);  
    }  
     
    const data = await response.json();  
    const realPresignedUrl = data.presignedUrl;  
     
    urlOutput.textContent = realPresignedUrl;  
     
    // Optionally, open the link directly for demonstration  
    // window.open(realPresignedUrl, '\_blank');  
     
    } catch (error) {  
    console.error('Failed to get presigned URL:', error);  
    urlOutput.textContent = `Error generating link: ${error.message}. Check browser console and Lambda logs.`;  
    }  
   });  
     
   **Important:** Replace YOUR\_API\_GATEWAY\_INVOKE\_URL\_HERE with the actual **Invoke URL** you obtained from API Gateway (Step 2.7). Make sure to include /prod or whatever stage name you used. For example: https://xxxxxxxxxx.execute-api.your-region.amazonaws.com/prod.
5. **Save and Push Changes to GitHub:**
   * Save your modified index.html file.
   * Open your terminal in your local repository directory.
   * Stage the changes: git add index.html
   * Commit the changes: git commit -m "Update frontend to use real AWS backend for presigned URLs"
   * Push the changes to your GitHub repository: git push origin main (replace main with your branch name if different).

## **Step 4: Test the End-to-End Flow**

1. **Wait for GitHub Pages to Deploy:** After pushing your changes, give GitHub Pages a few minutes to redeploy your updated index.html.
2. **Access Your GitHub Pages Site:** Open your GitHub Pages URL (e.g., https://yourusername.github.io/yourrepositoryname/) in your browser.
3. **Interact with the Demo:**
   * Scroll down to the "Interactive Demo" section.
   * Select a mock file.
   * Click **"Generate Temporary Download Link"**.
   * Observe the Generated Presigned URL output. This should now be a real S3 presigned URL.
   * Try opening this URL in a new tab. It should download the corresponding mock file from your S3 bucket!

If you encounter any issues:

* Check your Lambda function's **CloudWatch logs** for errors during execution.
* Verify your API Gateway **CORS configuration**.
* Double-check the **IAM permissions** for your Lambda function.
* Ensure the S3\_BUCKET\_NAME environment variable in Lambda is correct.

You now have a live, interactive demo backed by real AWS services!

## **Step 5: Add this Guide to Your GitHub Project**

To ensure this guide is easily accessible within your GitHub repository, you should add it as a Markdown file.

1. **Save this content locally:** Copy the entire content of *this* Canvas document.
2. **Create a new file in your local repository:**
   * In your cloned GitHub repository (e.g., s3-secure-transfer-guide), create a new directory called docs if it doesn't exist:  
     mkdir docs
   * Navigate into the docs directory:  
     cd docs
   * Create a new Markdown file, for example, aws-backend-deployment.md:
     + **Windows (Notepad):** notepad aws-backend-deployment.md
     + **macOS/Linux (Text Editor/Vim/Nano):** nano aws-backend-deployment.md or open with your preferred code editor.
3. **Paste the content:** Paste the entire content of this Canvas into the aws-backend-deployment.md file and save it.
4. **Commit and Push to GitHub:**
   * Navigate back to the root of your repository:  
     cd ..
   * Stage the new file:  
     git add docs/aws-backend-deployment.md
   * Commit your changes:  
     git commit -m "Add AWS backend deployment guide to documentation"
   * Push your changes to GitHub:  
     git push origin main  
       
     (Replace main with your branch name if different.)

Now, anyone visiting your GitHub repository will be able to navigate to the docs folder and find this comprehensive guide on deploying the AWS backend for your secure S3 file transfer demo.