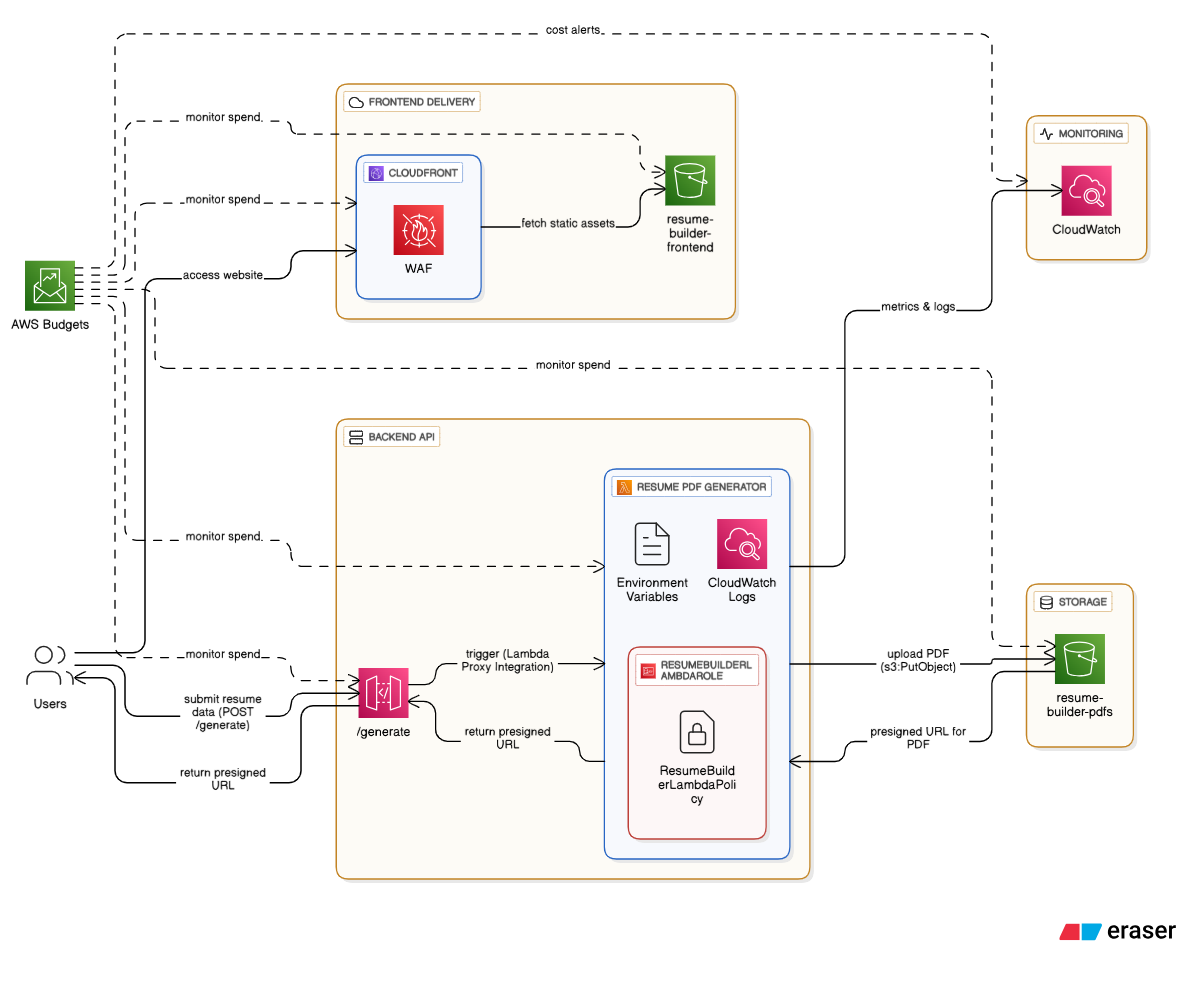
**Self-Hosted Resume Builder (AWS based)**

**Overview**

A complete self-hosted Resume Builder project where users can fill out an HTML form with their information to receive a professionally formatted PDF resume. The backend uses AWS Lambda to generate the PDF, which is then uploaded to S3. The frontend is served via CloudFront.

**Architecture Flow (Translated in English):**

1. A user accesses your website through **CloudFront**. CloudFront serves the static files (like index.html, script.js) from **S3**.
2. The user fills out the form and clicks the **"Generate PDF"** button.
3. The script.js file uses **JavaScript** to collect the form data in **JSON format**.
4. This JSON data is sent via a **POST request** to the **API Gateway endpoint**.
5. **API Gateway** receives the request and triggers your **Lambda function** (index.js).
6. The Lambda function takes the JSON data and uses **Puppeteer** to generate HTML (via the generateResumeHtml function in your code).
7. Puppeteer then generates a **PDF** from that HTML.
8. The Lambda function uploads the PDF file to an **S3 bucket**.
9. Once the upload is successful, the Lambda function creates a **Presigned URL** for the PDF from S3.
10. The Lambda function returns a **JSON response** with the Presigned URL back to the frontend via **API Gateway**.
11. The script.js file takes the URL from the response and opens the PDF in a new tab, allowing the user to **view or download** it.

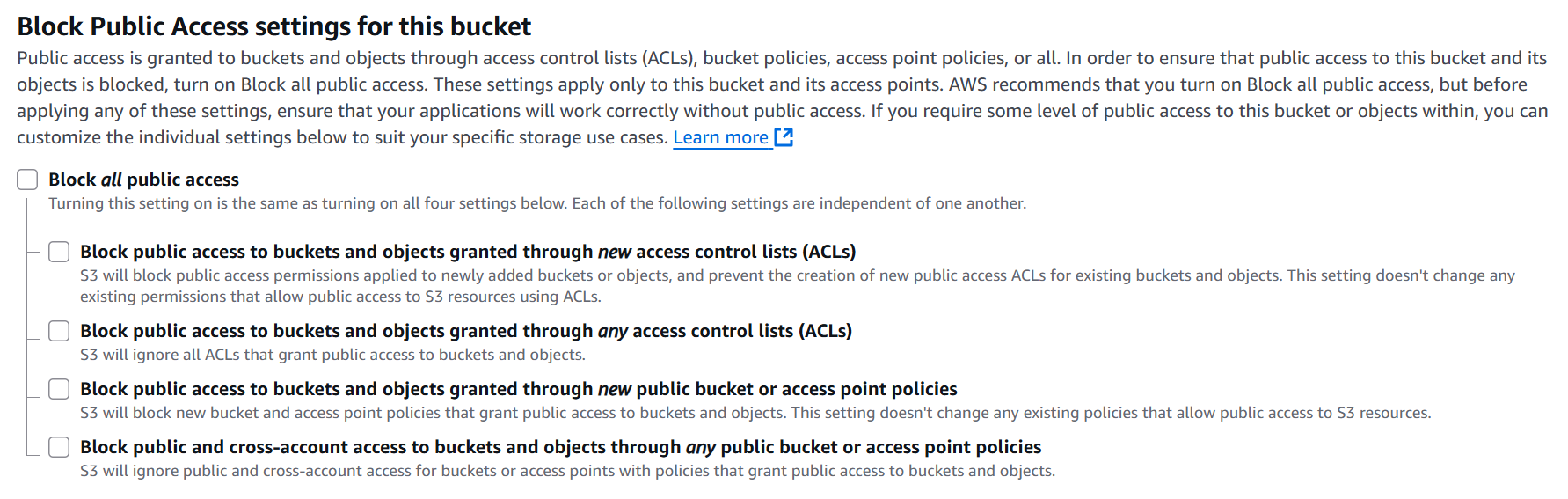


**Step-by-Step Setup Guide**

**Step 1: Create S3 Buckets**  
Purpose: Two buckets are required — one for storing the **website files** and another for **PDF storage**.

1st bucket name is **aws-resume-builder-frontend.** Here we have to do some steps:

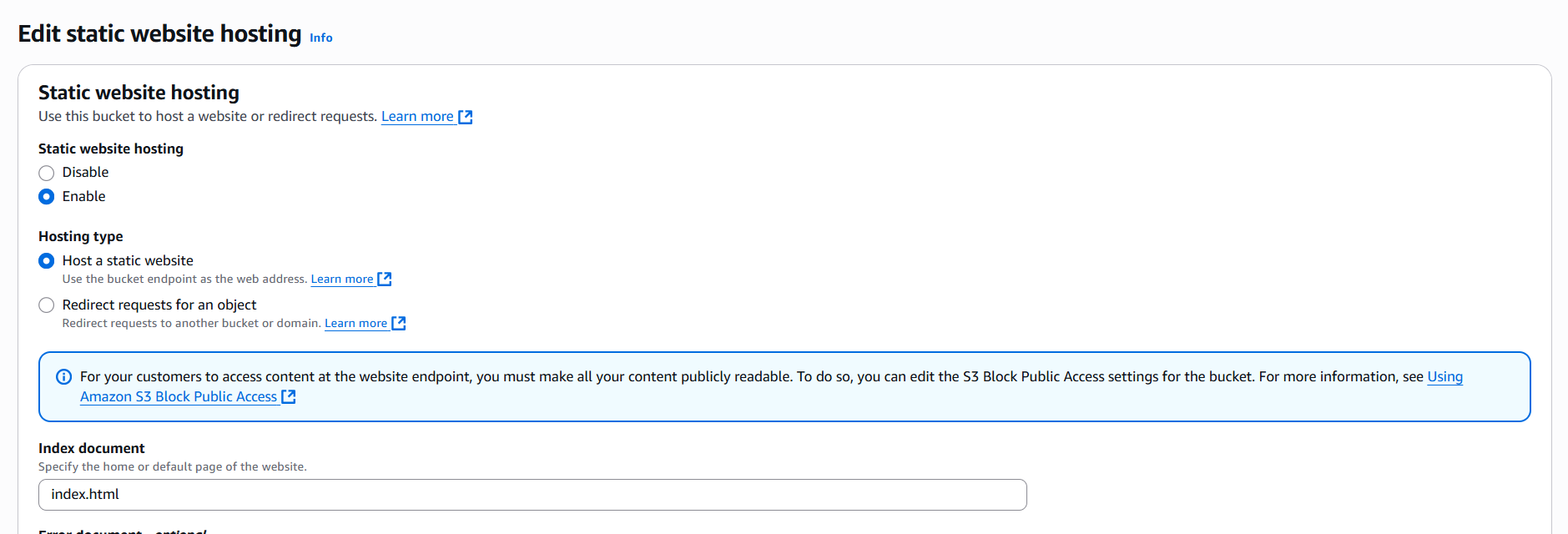
Block all public access (OFF)



2

**Enable Static Website Hosting**

1. Go to the **S3 Console** and select your frontend bucket (e.g., resume-builder-frontend).
2. From the top tabs, click on the **“Properties”** tab.
3. Scroll down to find the **“Static website hosting”** section. It will likely be in **Disabled** state.
4. Click the **“Edit”** button.
5. In the settings under “Static website hosting”, select the **“Enable”** option.
6. In the **“Index document”** field, type: index.html
7. Finally, click the **“Save changes”** button.



3

**Add a Public Read Bucket Policy**

**Important Note:**  
Even if "Block all public access" is disabled, you must add a bucket policy to allow public read access to your website files.

**Steps:**

1. Go to the **S3 Console** and open your frontend bucket (resume-builder-frontend).
2. Click on the **“Permissions”** tab.
3. Scroll down to the **“Bucket policy”** section.
4. Click **“Edit”**.
5. Copy and paste the following JSON policy:

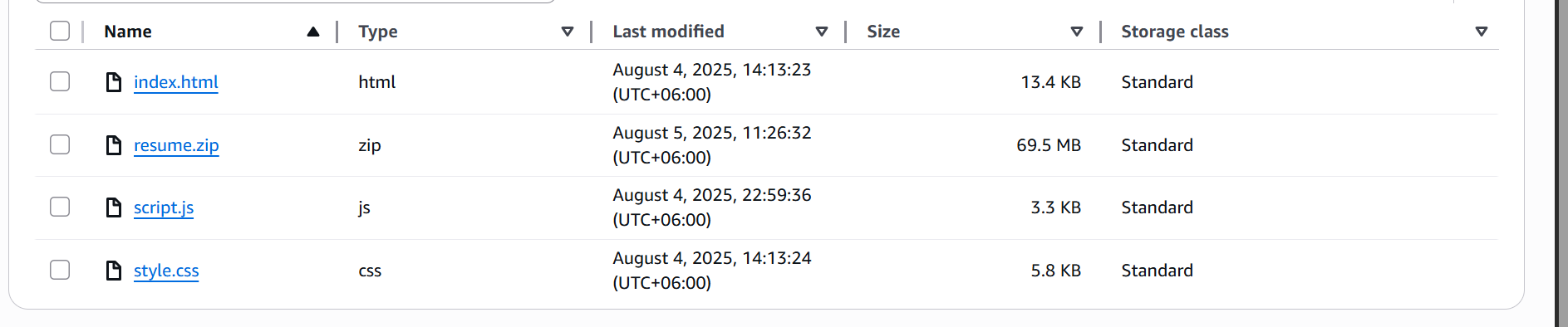


4

1. Click **“Save changes”**.

Once these two steps are completed, your **S3 bucket** will be fully ready to function as a **static website**.

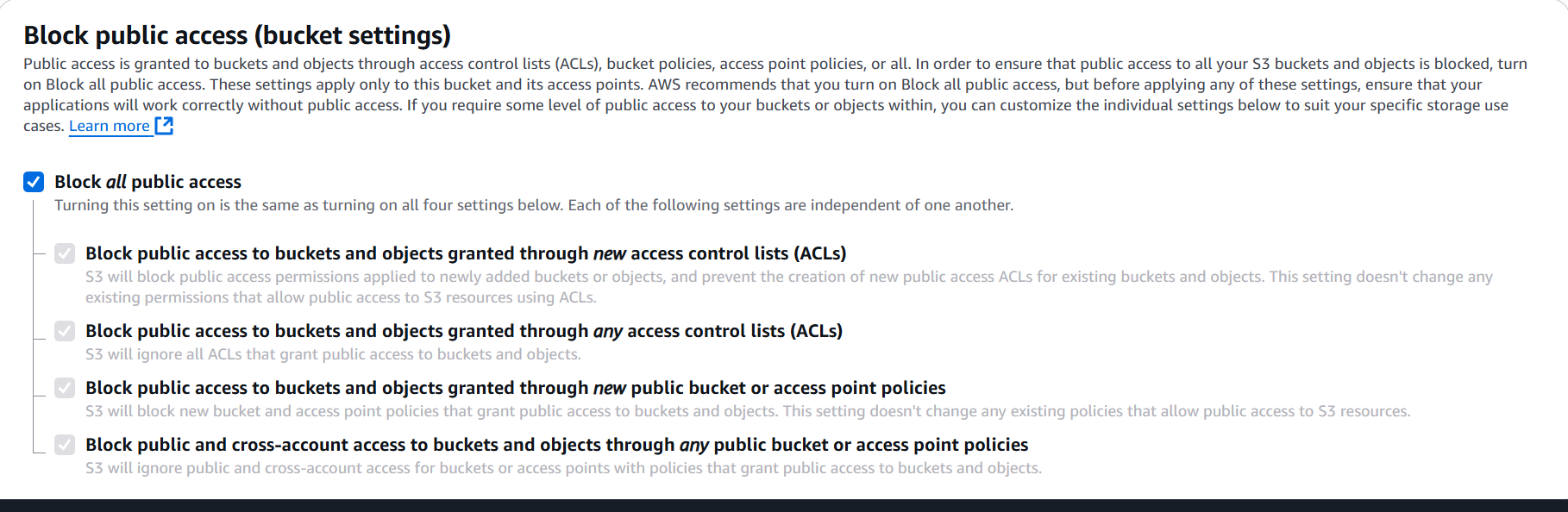
Then upload the frontend code in this bucket.



5

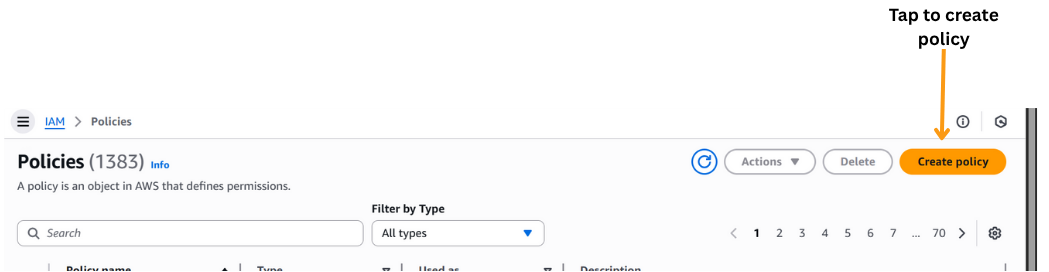
2nd bucket name is **aws- resume-builder-pdfs**

Block all public access (ON)



6

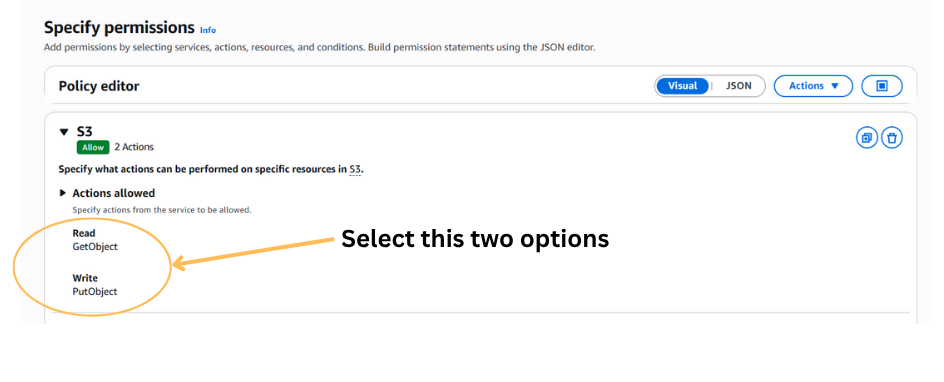
**Step 2: Create IAM Role & Policy**  
**Purpose:** To grant Lambda permission to access S3 and CloudWatch.



7

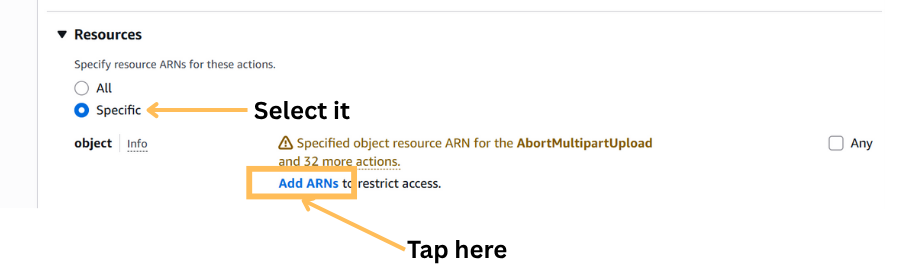
**For S3 Section:**

Select service: **S3**. Go to Action and select PutObject & GetObject

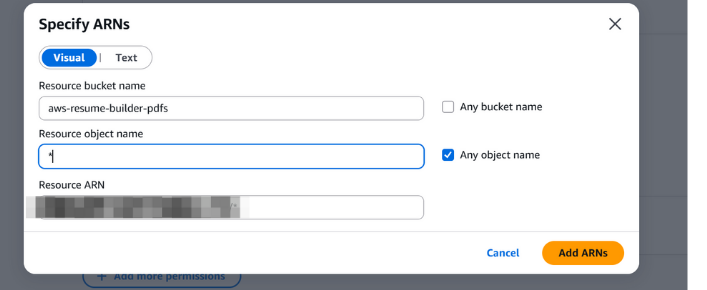


**8**

**In resource:**



9

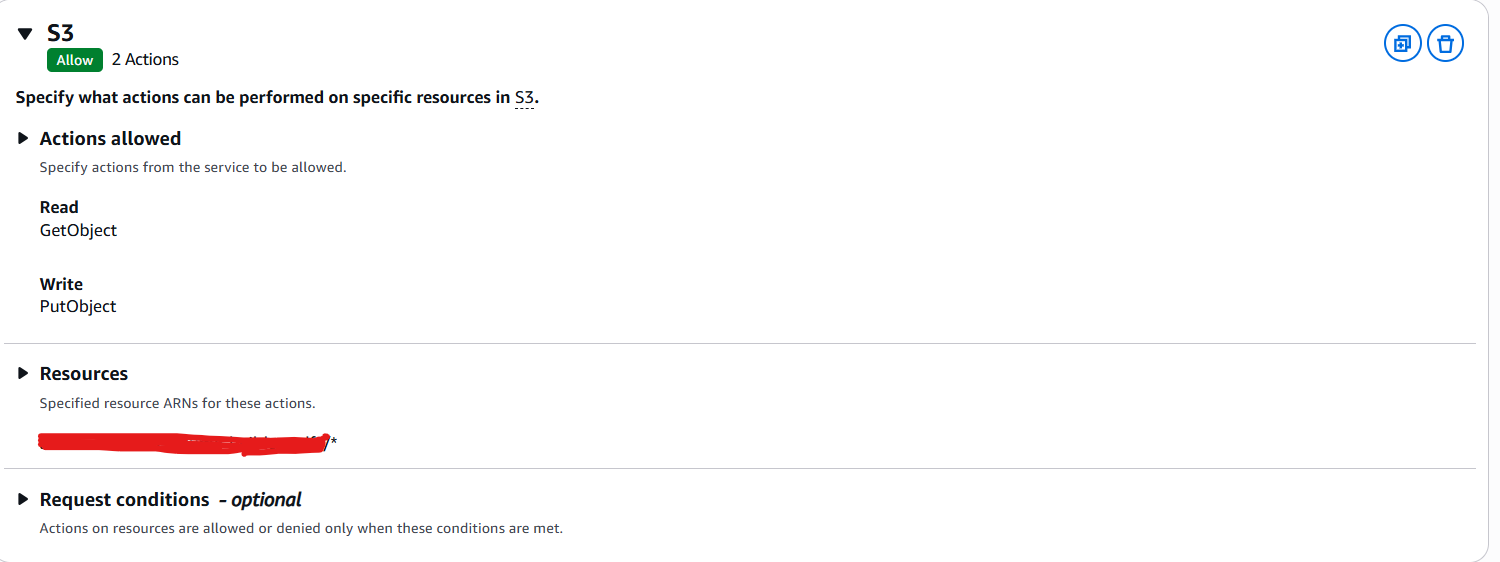


10

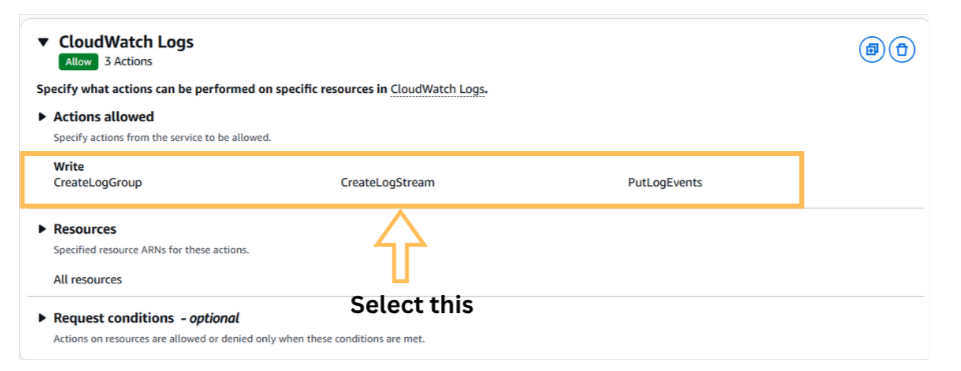
Here also provide **JSON Code:**



11

**Final Overview:**  


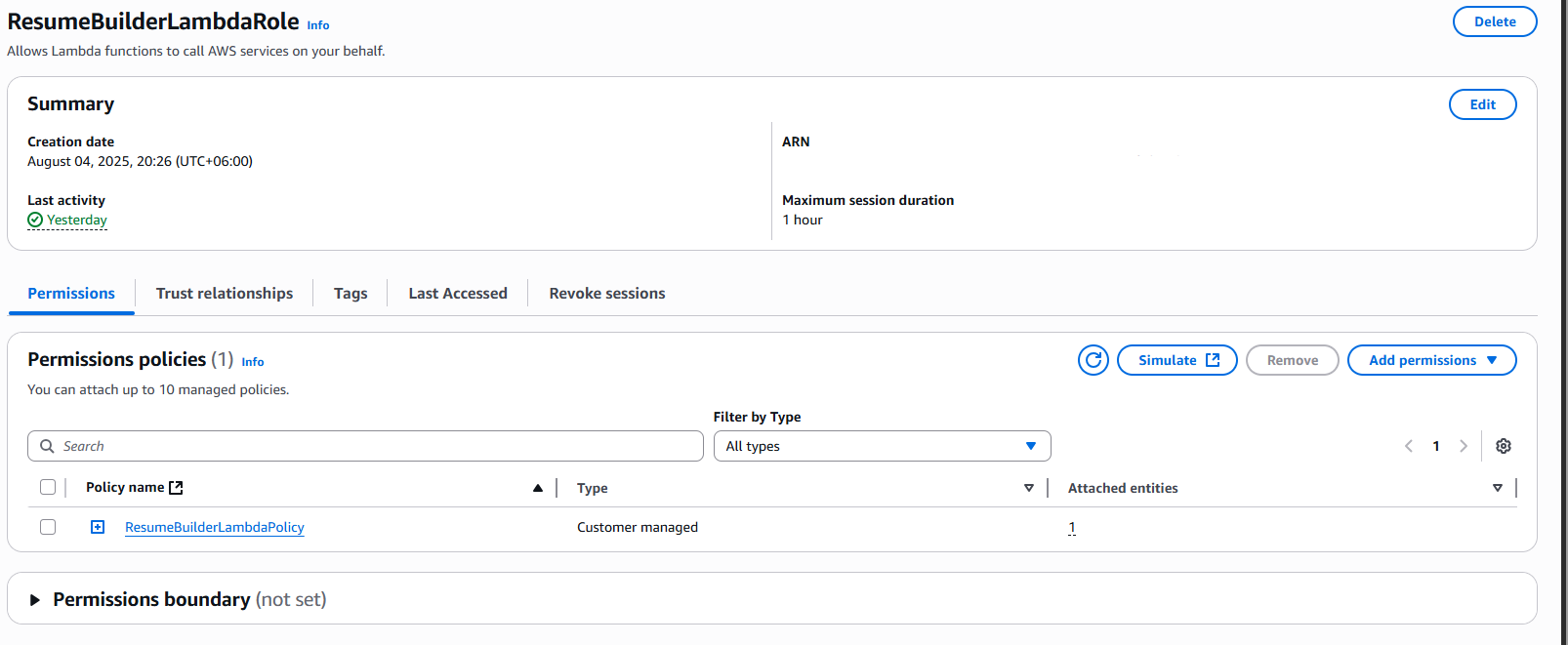
**12**

**For CloudWatch Logs:**  


**13**

Then save as the policy as **ResumeBuilderLambdaPolicy.**

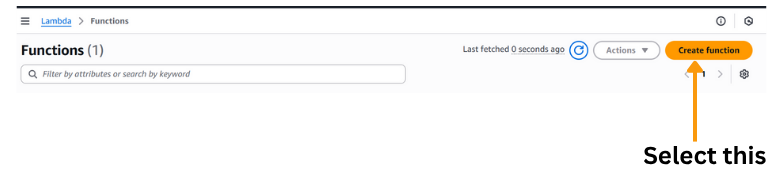
Then go to Roles: Create roles & select use case **Lambda**.Attach above policy and save it as **ResumeBuilderLambdaRole**

**Final Overview:**  


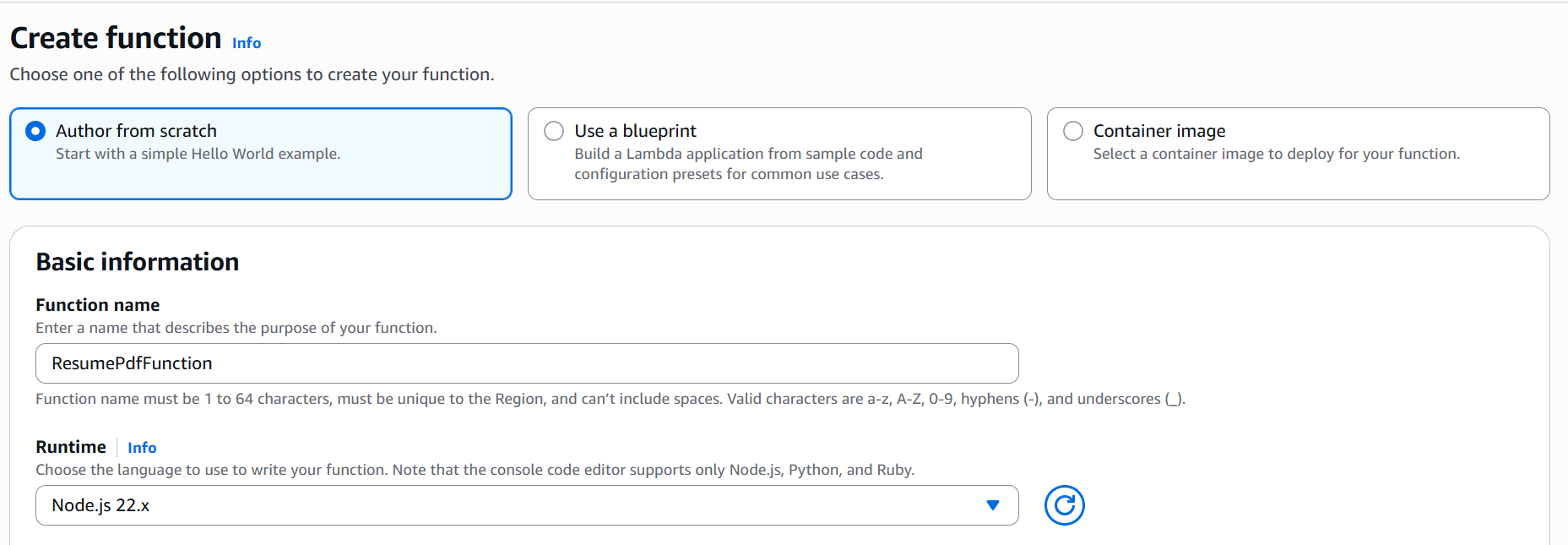
**14**

**Step 3: Create Lambda Function**  
**Purpose:** To generate a **PDF** from the form data and upload it to **S3**.

AWS Console → Lambda → Create Function → “Author from scratch”

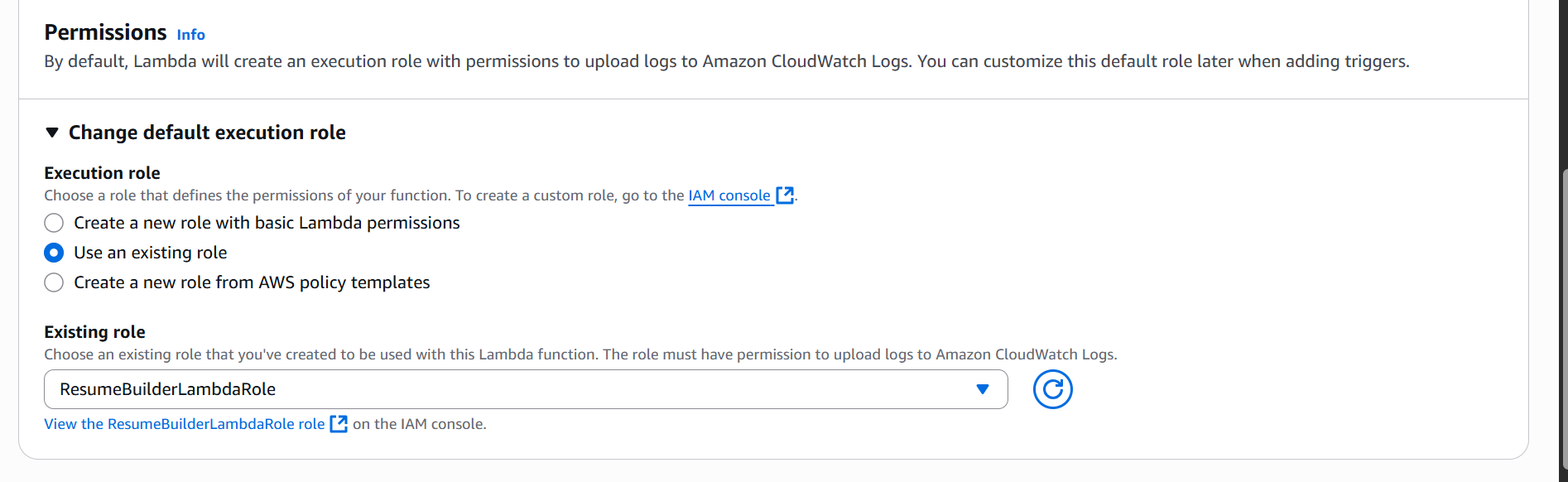


15



16

Then Create Function

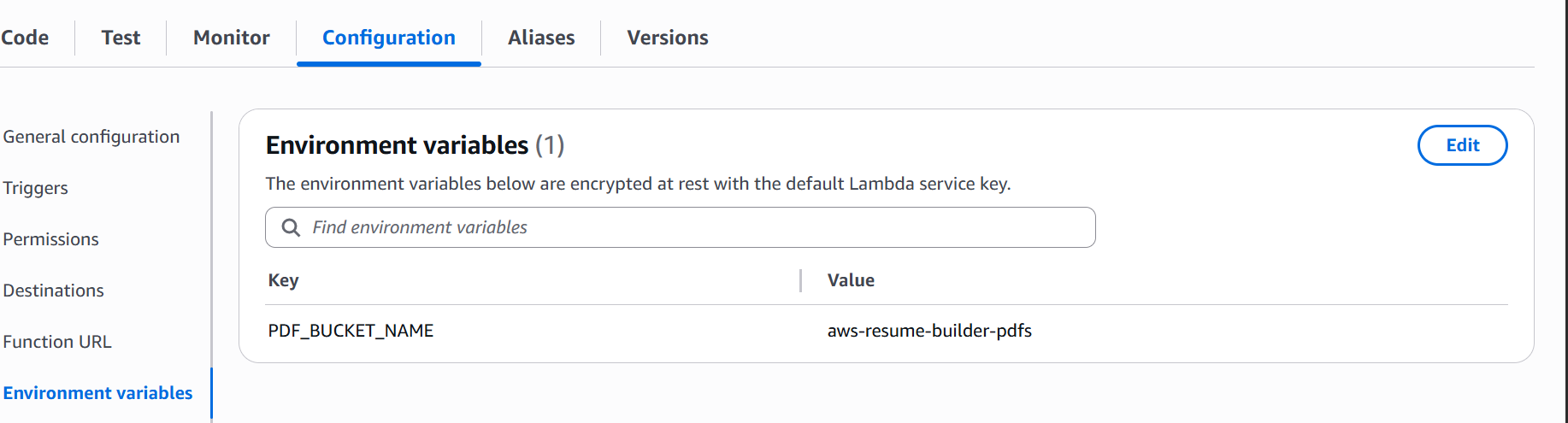


17

Permissions: Attach **ResumeBuilderLambdaRole**

**Add Environment Variable to Lambda Function**

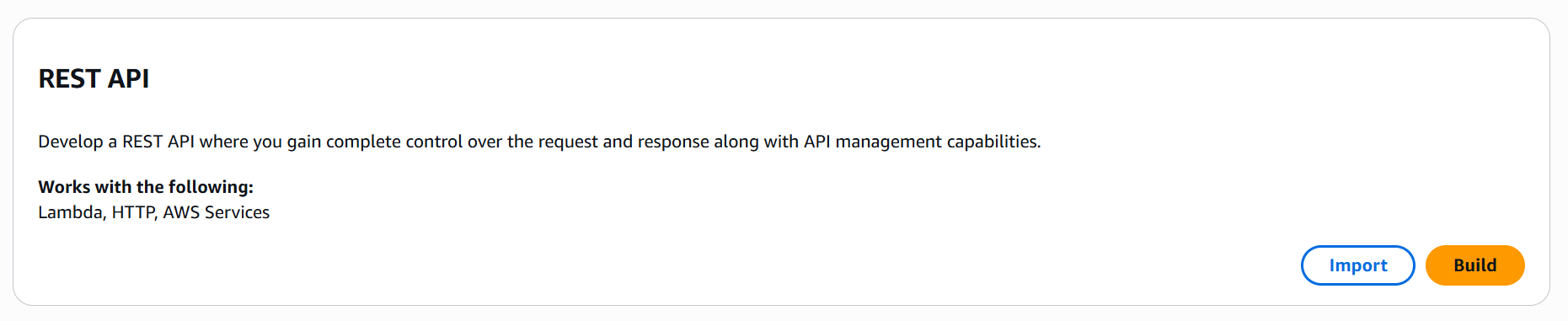
To allow your Lambda function to dynamically access the PDF bucket name:



18

**Step 4: Create API Gateway**  
**Purpose:** In this step, you will create a **secure HTTP endpoint** between your **frontend** and the **Lambda function**.This endpoint will receive **POST requests** from your website’s script.js file and forward the data to the **Lambda function** for processing.

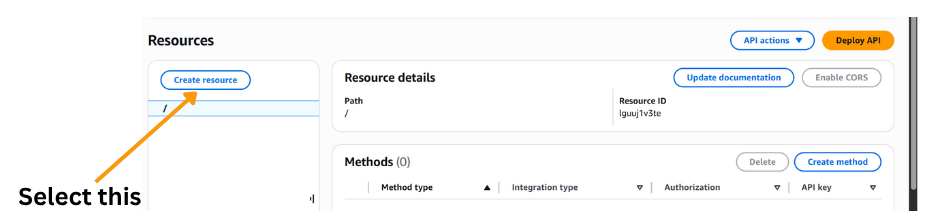
Go to API Gateway → Create API → REST API → Build



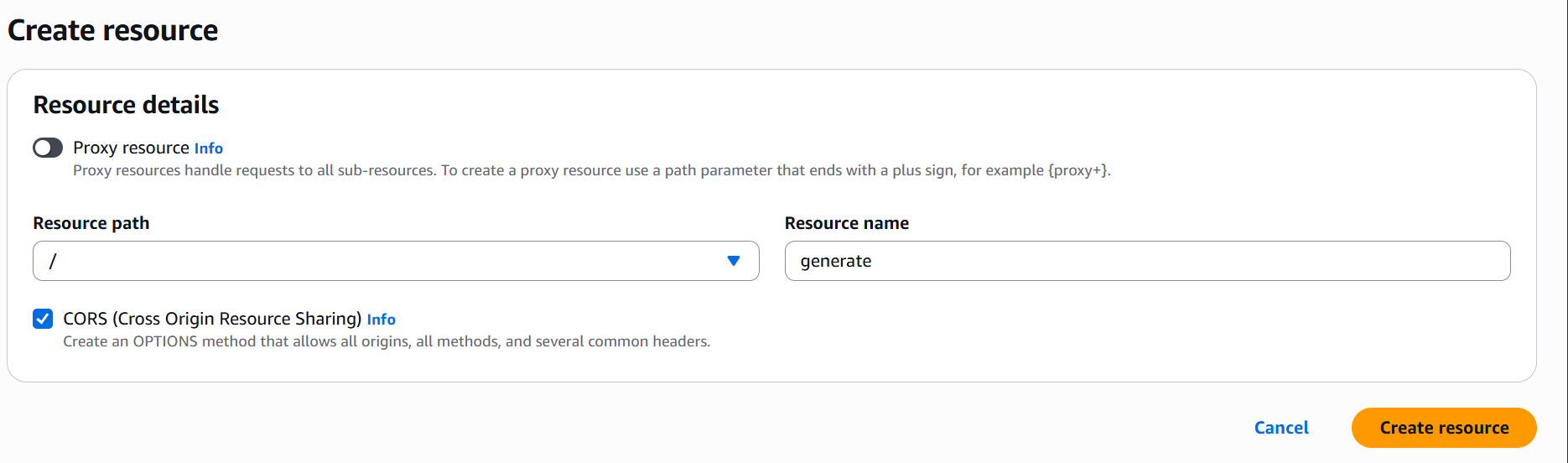
19

Name: **ResumeApi**

Actions → Create Resource → Name: **generate**



20



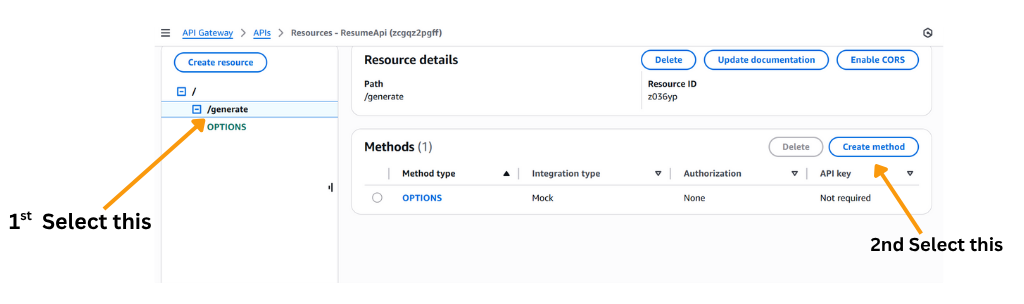
21

Click the **“Create Resource”** button to create a new resource (e.g., /generate-pdf) under your API in **API Gateway**. This resource will serve as the **path** for your HTTP endpoint.

**Create Method and Integrate with Lambda:**

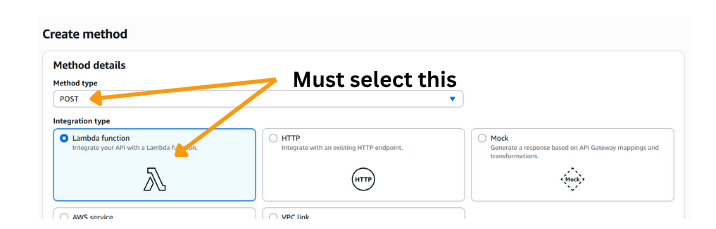
**Steps:**

1. Select the newly created resource (/generate) from the left panel in **API Gateway** and select **“Create Method”**.



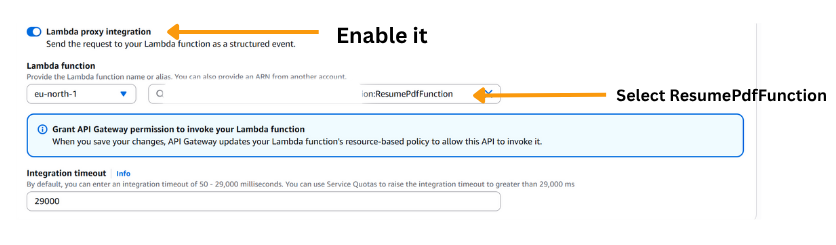
22

1. From the dropdown list, choose **POST**.
2. Configure the method with the following settings:
   * **Integration type:** Lambda Function



23

* + ✅ Enable **“Use Lambda Proxy integration”** — this allows your Lambda to access the full HTTP request data.
  + **Lambda Function:** Type and select your function name (e.g., ResumePdfFunction)



24

1. Click **“Save”**.

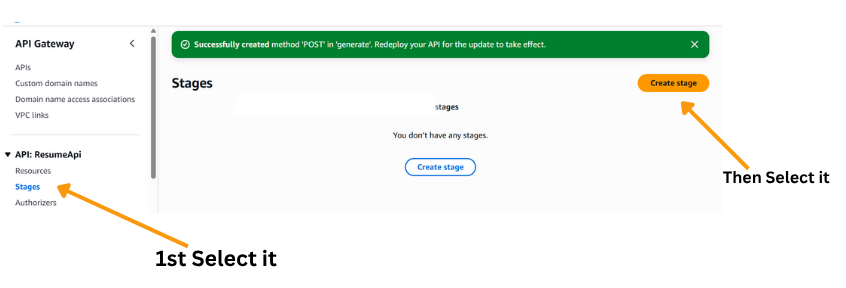
A pop-up will appear asking for permission to allow API Gateway to invoke your Lambda function. Click **OK** to grant access.

**Enable CORS (Short Version)**

1. In API Gateway, select the /generate resource.
2. Click **Actions → Enable CORS**.
3. In the popup:
   * Ensure **OPTIONS** and **POST** are selected.
   * Set **Access-Control-Allow-Origin** to \*.
4. Click **“Enable CORS and replace existing CORS headers”**.

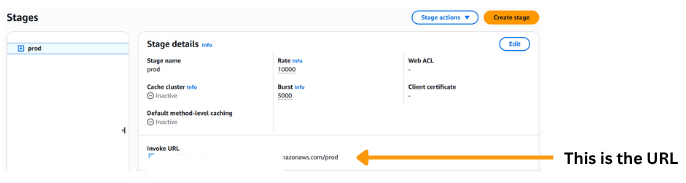
**Deploy API**

1. In API Gateway, click **Actions → Deploy API**.
2. Select **[New Stage]** from the dropdown.



25

1. Set **Stage name** to prod.
2. Click **Create Stage**.
3. After deployment, you'll see an **Invoke URL** like:  
   <https://your-api-id.execute-api.region.amazonaws.com/prod>



26

1. Use this URL in your script.js as:  
   <https://.../prod/generate>

Your API is now live and ready to accept requests!

**Step 5: Set API URL in script.js**

Replace your fetch URL in script.js



27

This will send the form request directly to your **Lambda function** via the API Gateway.

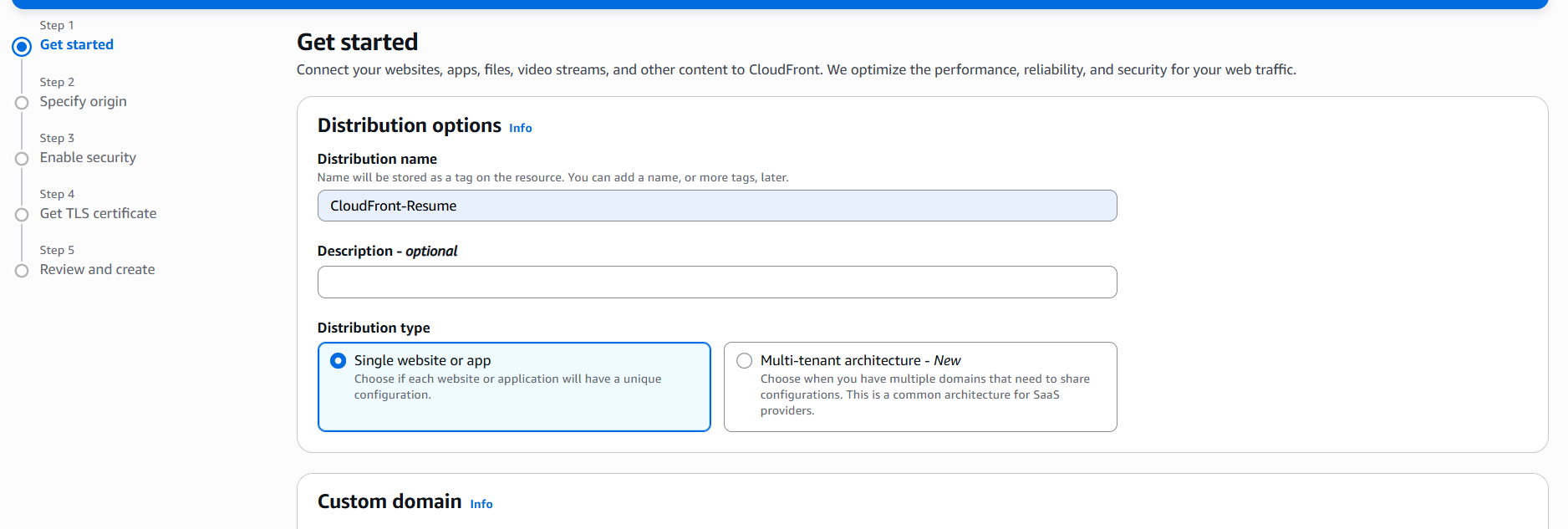
**Step 6: Set Up CloudFront + WAF**

To make your frontend fast, secure, and globally accessible:

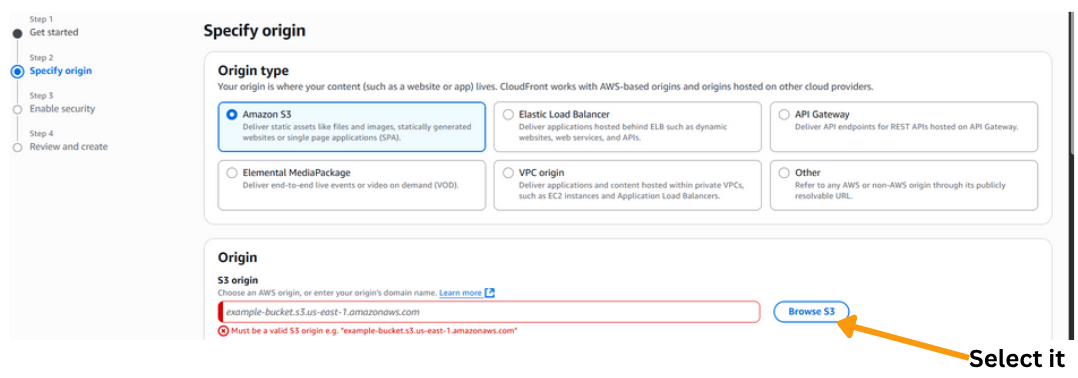
**Create CloudFront Distribution (Step-by-Step)**

1. **Go to CloudFront Console:**  
   In the AWS Management Console, search for **CloudFront** and click on **Create Distribution**.
2. **Select Origin Domain:**
   * In the **Origin domain** field, click the dropdown.
   * Select your **S3 frontend bucket** (e.g., resume-builder-frontend).
   * CloudFront will automatically set this as the **origin**.

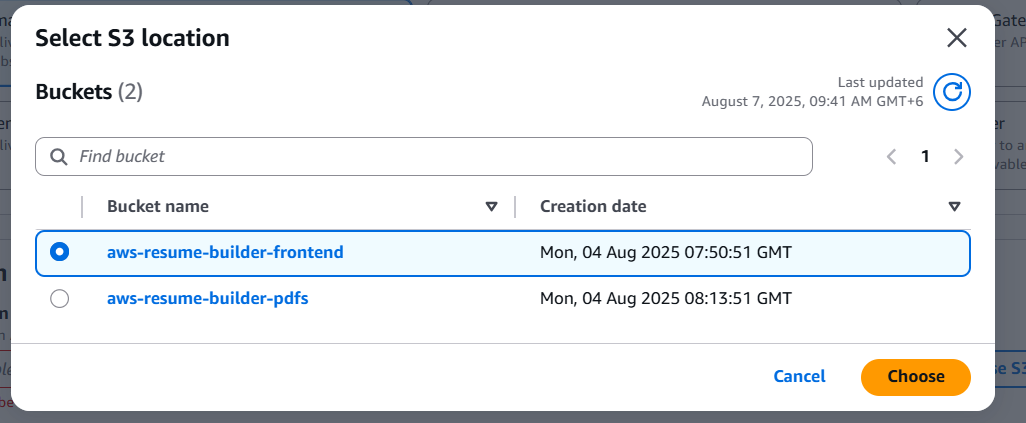
You’ve now connected CloudFront to your S3 static website bucket.



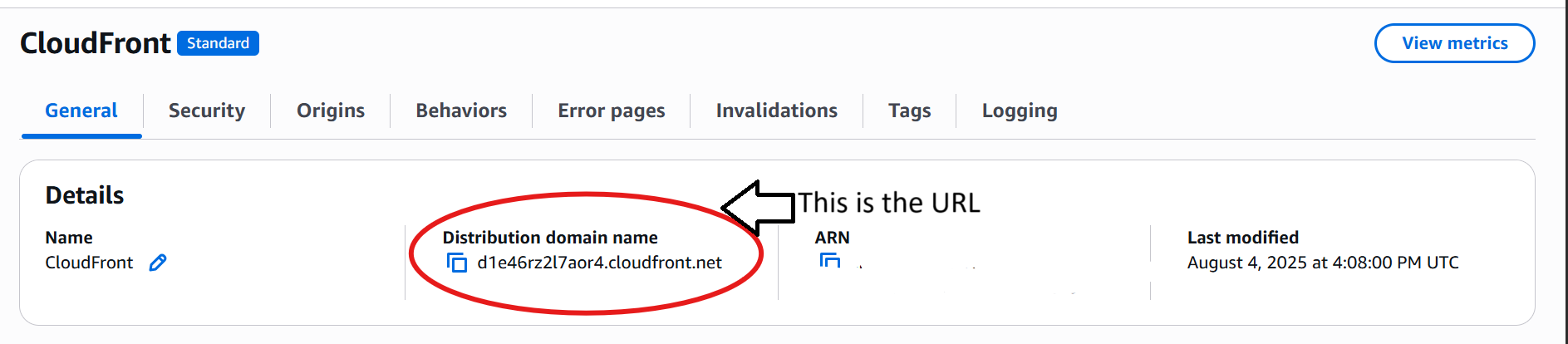
28



29



30



31

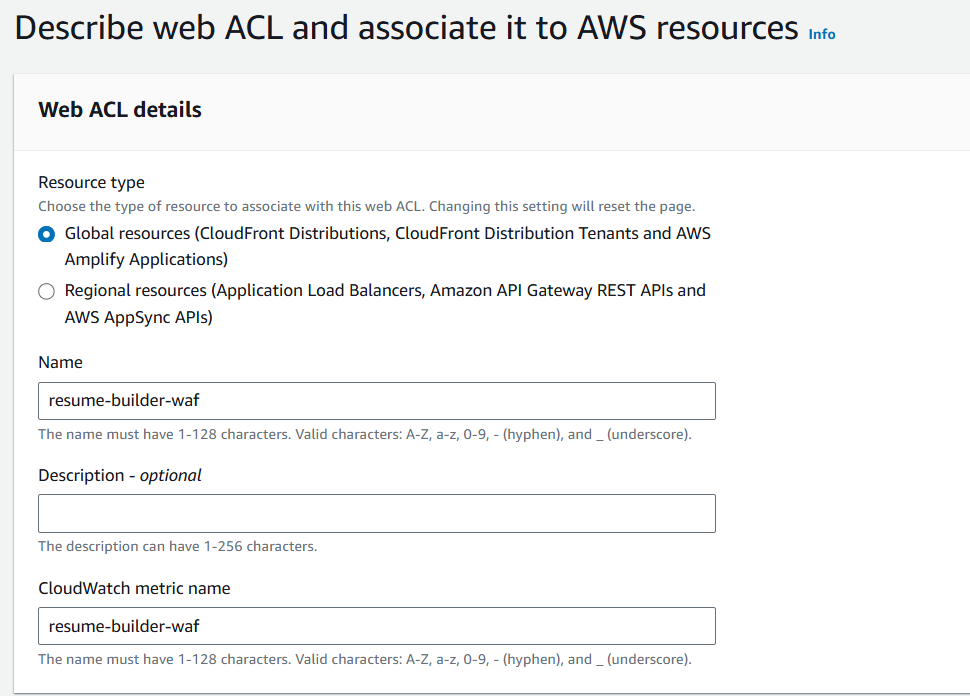
**WAF Setup for CloudFront**

**Open WAF Console**

* Go to AWS Console → Search **WAF & Shield** → Click **WAF**

**Create Web ACL**

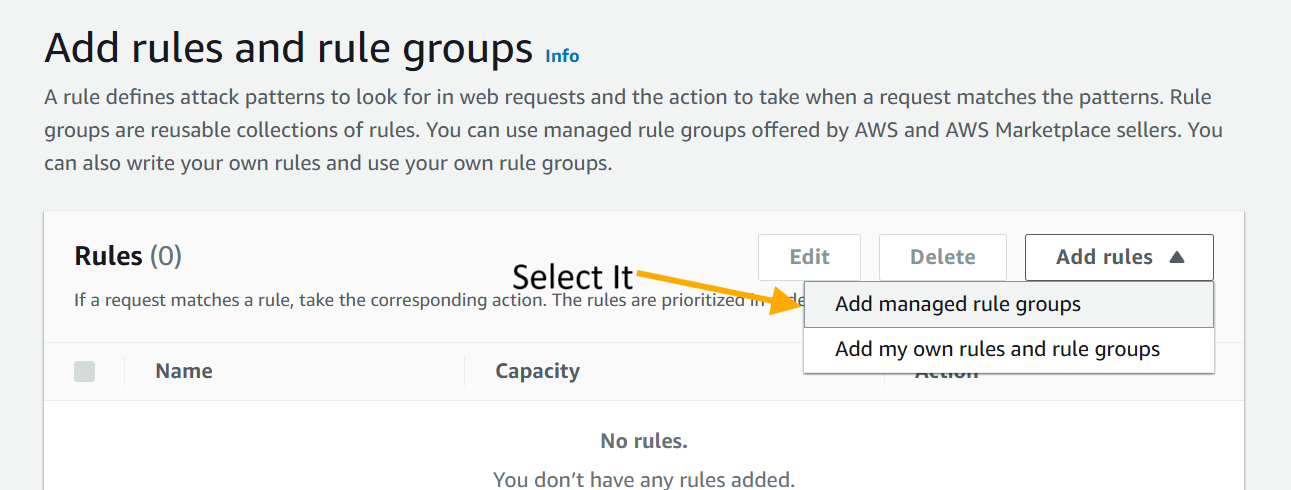
* Click **Create web ACL**
* Name: resume-builder-waf
* Resource type: **CloudFront distributions**
* Region: **Global (CloudFront only)** → Next



32

**Add Rule**

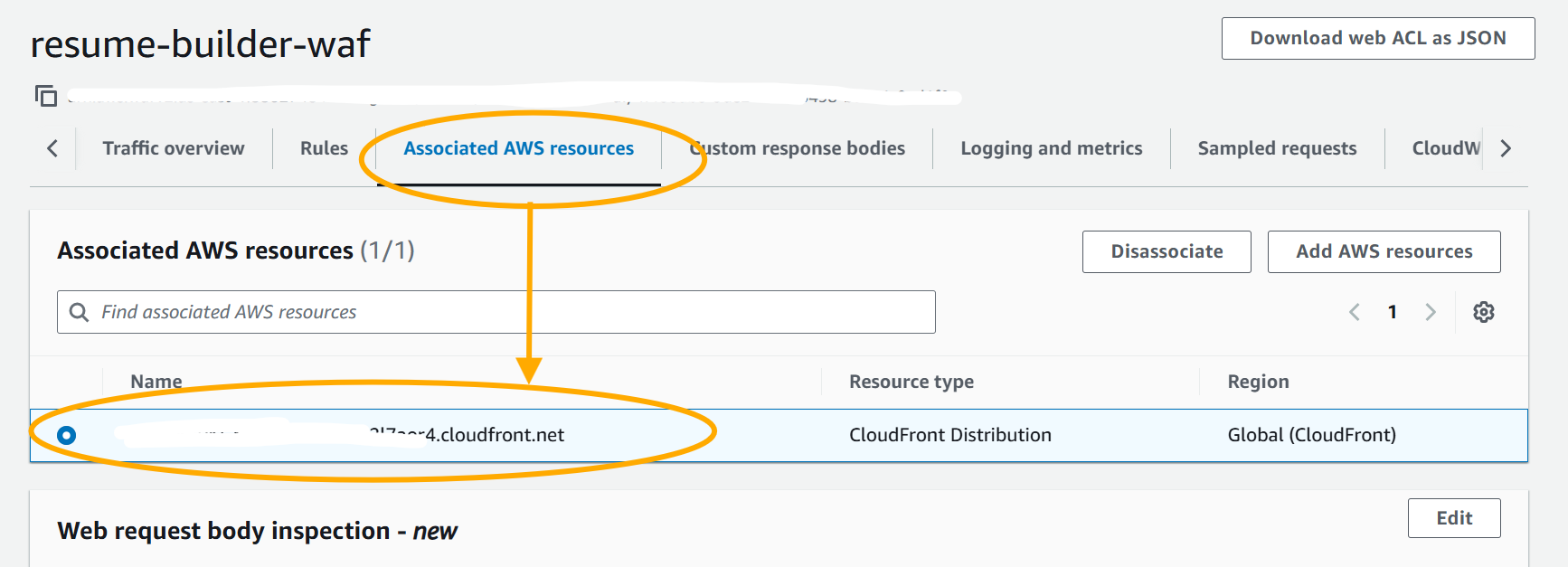
* Choose **Add managed rule groups**



33

**Associate with CloudFront**

* Select your CloudFront distribution under **Associated AWS resources**
* Save and confirm

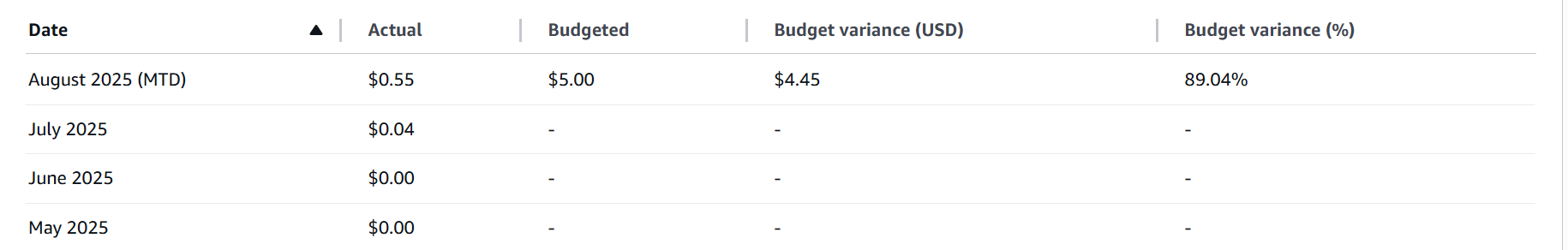


34

**Step 7: Detailed Analysis of CloudWatch + Budget Control**  
Helps monitor logs and control costs.

• **Lambda execution logs** → View in **CloudWatch**  
• **AWS Budgets** → Create a budget (e.g., $5)

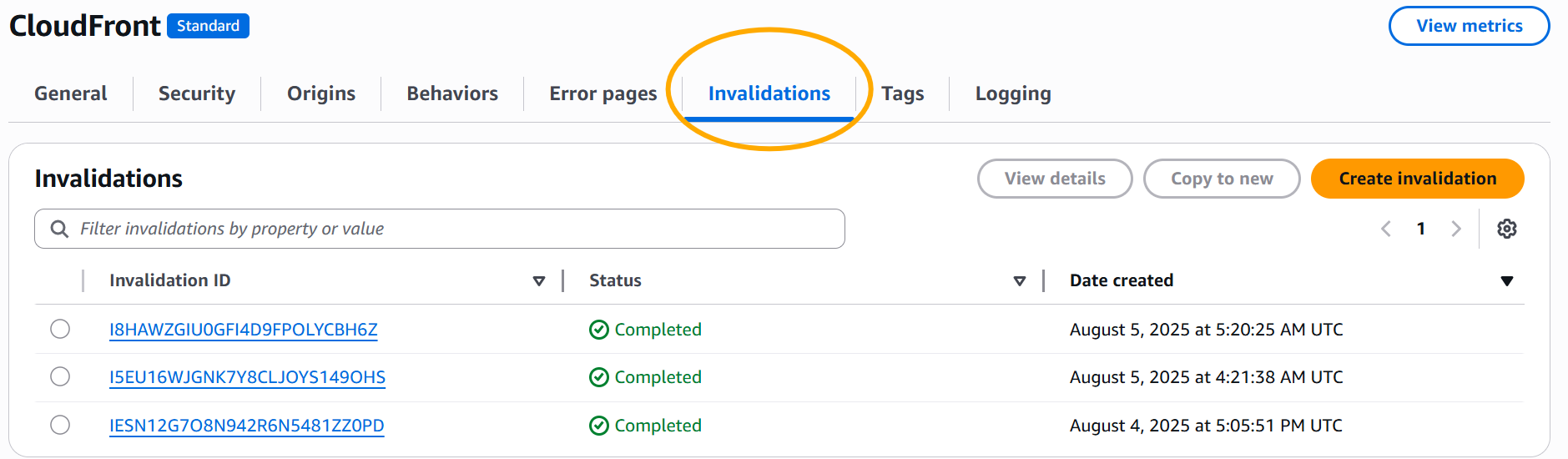
* **Notification**: When actual cost > 100% → Send email alert

  
35

**Troubleshooting**

**Problem 1: My website wasn't updating even after I changed the frontend code.**

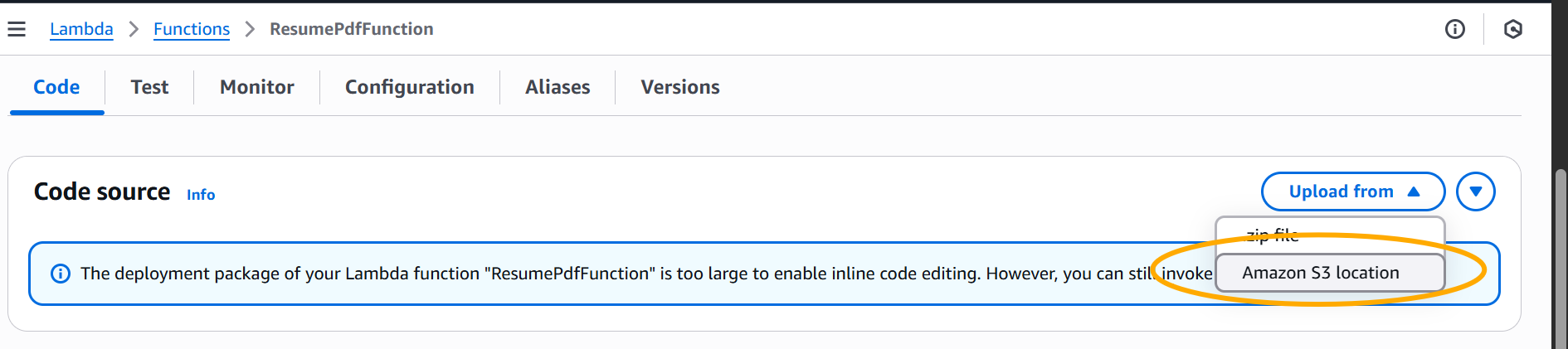
* **Cause:** CloudFront was caching my static files (script.js, style.css). As a result, even when I uploaded a new file to the S3 bucket, CloudFront was still serving the old, cached version to users.
* **Solution:** I went to the CloudFront console and created an **"Invalidation"** for my distribution. By invalidating specific files like /index.html or /script.js, CloudFront cleared the old files from its cache and started serving the new ones from S3.



36

**Problem 2: I got a The selected file is too large. The maximum size is 50 MB. error when deploying the Lambda function.**

* **Cause:** My .zip file was larger than the 50 MB limit for a direct upload to Lambda.
* **Solution:** Instead of uploading the .zip file directly to Lambda, I first uploaded it to an S3 bucket. Then, from the Lambda console, I updated the code by providing the S3 Object URL. This worked perfectly.



37

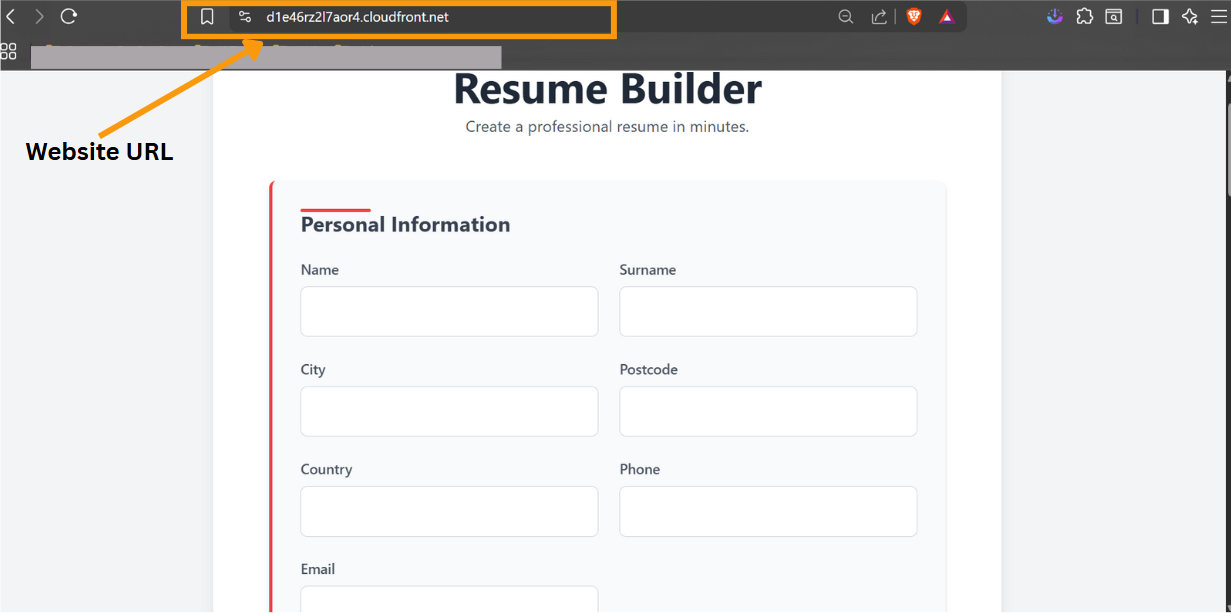
**Problem 3: I was seeing a net::ERR\_CONNECTION\_REFUSED error.**

* **Cause:** The Invoke URL for my API Gateway in the script.js file was incorrect. I had either made a typo or omitted a crucial part of the URL, such as the /prod/generate path.
* **Solution:** I opened my script.js file and replaced the incorrect URL with the correct Invoke URL copied from the API Gateway console. I made sure to include /generate at the end and then re-uploaded the file to S3.



38

**Final Look**



41