

Deployment Guide (Manual via Console)

URL Shortener

Overview

When a user enters a long URL, the system generates a short, unique identifier (like abc123). The user can then share this short URL. When someone clicks on it, the system looks up the identifier, finds the original URL, and redirects the visitor to that URL.

Example:

Original Long URL	Short URL
https://www.amazon.com/product/B08KH53NKR/ref	https://short.ly/abc123

Goals & Components Explained

1. Serverless URL Shortener

- **Serverless** means you don't manage servers — AWS handles it.
- The logic runs in **AWS Lambda**, which only executes when triggered (you only pay for usage).
- No EC2, no backend servers — making it **scalable and cost-effective**.

2. Frontend: Static Website

- The user interface (form to enter URL, display short URL) is hosted as a **static website** in **Amazon S3**.
- It might be a simple HTML/JavaScript app.
- You can enable **S3 static website hosting** and optionally use **CloudFront** to serve it globally.

3. Backend: URL Shortening & Redirection via Lambda

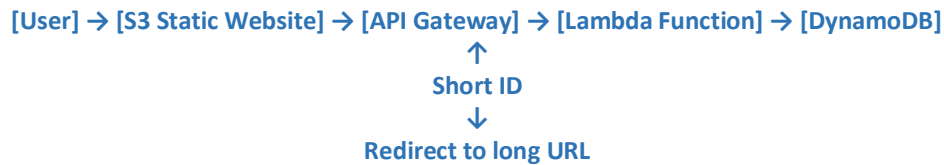
- When a user submits a long URL:
 - The frontend sends it to an **API Gateway endpoint**.
 - The request triggers an **AWS Lambda function**.
 - The Lambda function:
 - Generates a **short unique ID**.
 - Stores the mapping (short_id → long_url) in **DynamoDB**.
 - Returns the short URL to the frontend.
- When someone visits the short URL:
 - The short ID is passed to another **API Gateway + Lambda** endpoint.
 - Lambda looks up the short ID in DynamoDB.
 - If found, it **redirects** the user to the original long URL using a 301 redirect.

4. Data Storage: DynamoDB

- A **NoSQL database** to store mappings:
 - Partition key: short_id

- Attribute: long_url
- Fast lookups, fully managed, and scales automatically.

Architecture Summary



AWS Services Used

Service	Purpose
Amazon S3	Host static frontend
API Gateway	Expose REST endpoints for frontend to call
AWS Lambda	Handle logic for URL shortening and redirection
Amazon DynamoDB	Store short URL to long URL mappings

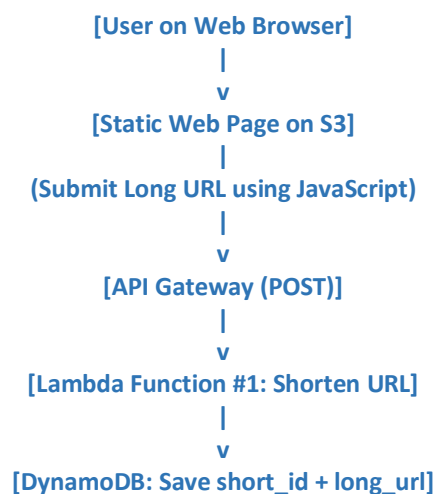
Testing Goals

- Focus is on **building and testing the core logic**:
 - Submit long URL → receive short URL
 - Visit short URL → get redirected to long one

Architecture Overview (with Diagram)

Here's how the system works, visually:

System Flow



|
v
← Return short URL to frontend

Now when the user visits the short URL:

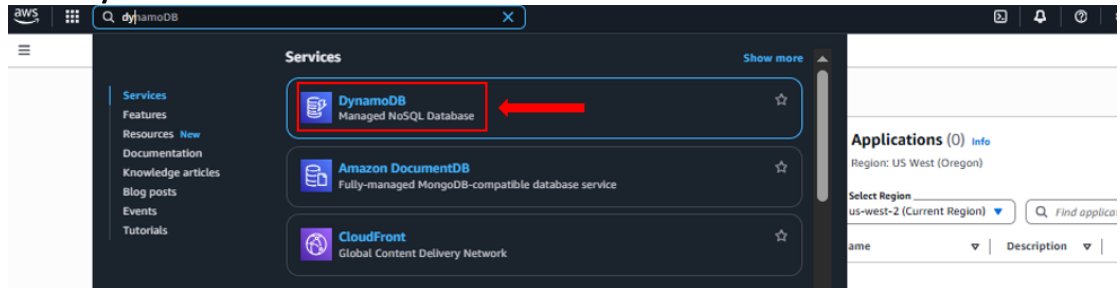
[User clicks short URL: /abc123]
|
v
[API Gateway (GET) with path /abc123]
|
v
[Lambda Function #2: Redirect]
|
v
[DynamoDB: Lookup abc123 → long_url]
|
v
← HTTP 301 Redirect to original URL

📄 URL Shortener Project Tutorial with Screenshots (AWS Serverless)

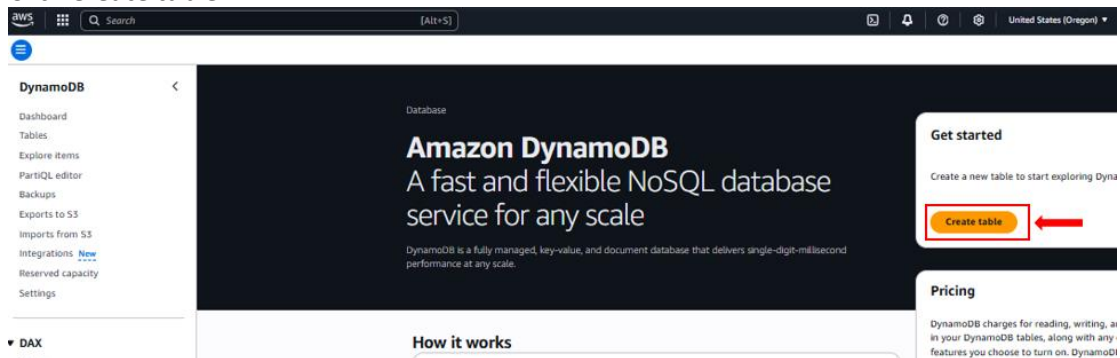
This guide walks you through creating a **serverless URL shortener** using AWS services: **Lambda**, **DynamoDB**, and **S3** — with visuals for every step.

✅ Step 1: Create a DynamoDB Table

1. Go to **DynamoDB Console**.



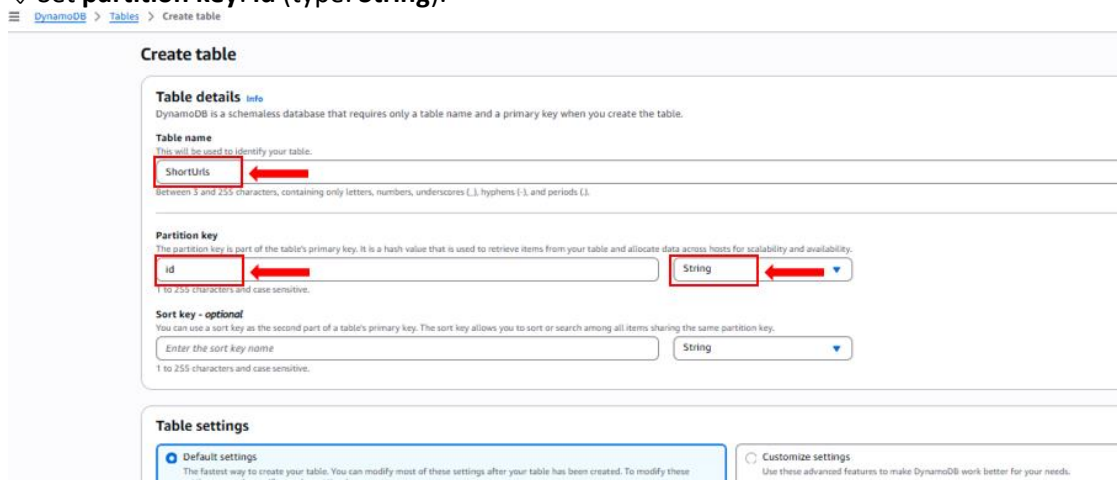
2. Click **Create table**.



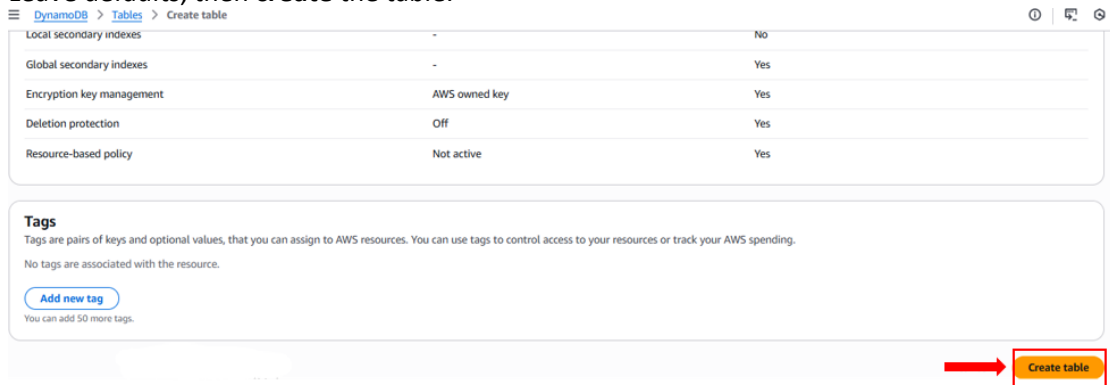
3. Set these values :

✅ Table Name "**ShortUrls**".

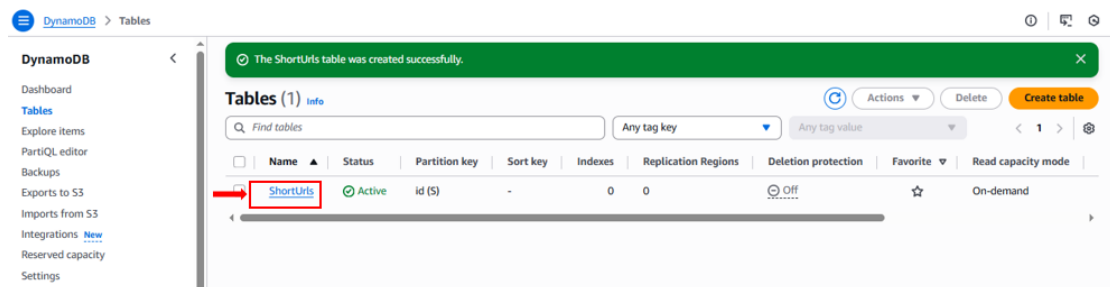
✅ Set **partition key**: **id** (type: **String**).

A screenshot of the 'Create table' form in the AWS DynamoDB console. The form is titled 'Create table' and has sections for 'Table details', 'Partition key', 'Sort key - optional', and 'Table settings'. In the 'Table details' section, the 'Table name' is 'ShortUrls'. In the 'Partition key' section, the 'id' is selected as the partition key with a type of 'String'. In the 'Sort key - optional' section, 'String' is selected as the sort key type. A red arrow points to the 'id' field in the partition key section.

4. Leave defaults, then **create** the table.

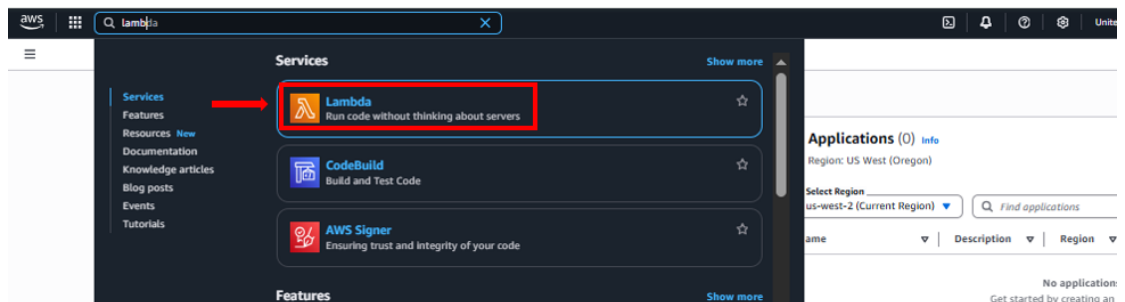


5. Successfully created "**ShortUrls**" table.

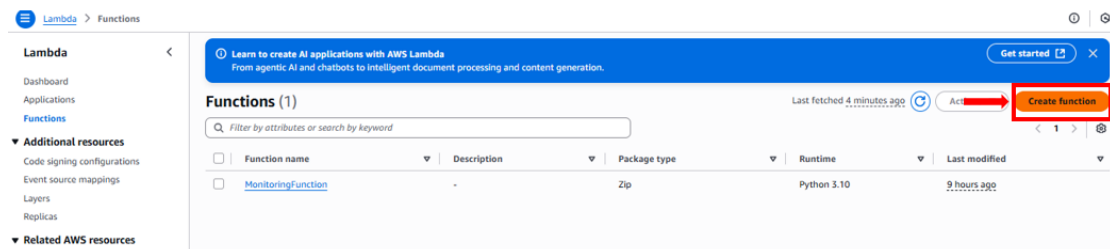


✓ **Step 2: Create the Lambda Function**

1. Go to **Lambda Console**.



2. Click **Create Function**.



3. Choose:

- ✓ **Author from scratch.**
- Name: **url-shortener-elham-lambda.**
- Runtime: **Python 3.10.**

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.

Basic information

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

(Function names must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).)

Runtime info
Choose the runtime to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Architecture info
Choose the instruction set architecture you want for your function code.
☒ x86_64
 ☐ arm64

5. Execution role: **Use an existing role** → Select **labrole** , and then click **create**.

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions
 ☒ **Use an existing role**
☐ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

View the LabRole role on the IAM console.

Additional configurations
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Create function

6. Successfully created "url-shortener-elham-lambda" function .

Functions (2)

Filter by attributes or search by keyword

<input type="checkbox"/>	Function name	Description	Package type	Runtime	Last modified
<input type="checkbox"/>	MonitoringFunction	-	Zip	Python 3.10	10 hours ago
<input type="checkbox"/>	url-shortener-elham-lambda	-	Zip	Python 3.10	11 minutes ago

7. Upload the code into the editor and then **deploy** the function.

Code source info

Open in Visual Studio Code [Upload from](#)

EXPLORER

- URL-SHORTENER-ELHAM-LAMBDA
 - lambda_function.py

```

1 import json
2 import boto3
3 import string
4 import random
5
6 dynamodb = boto3.resource('dynamodb')
7 table = dynamodb.Table('ShortURLs')
8
9 def generate_short_id(length=6):
10     return ''.join(random.choices(string.ascii_letters + string.digits, k=length))
11
12 def lambda_handler(event, context):
13     method = event['requestContext']['http']['method']
14
15     # Handle CORS preflight (OPTIONS request)
16     if method == "OPTIONS":
17         return {
18             "statusCode": 200,
19             "headers": {
20                 "Access-Control-Allow-Origin": "*",
21                 "Access-Control-Allow-Methods": "OPTIONS,POST,GET",
22                 "Access-Control-Allow-Headers": "Content-Type"
23             },
24             "body": ""
25         }
26
27     if method == "POST":
28         try:

```

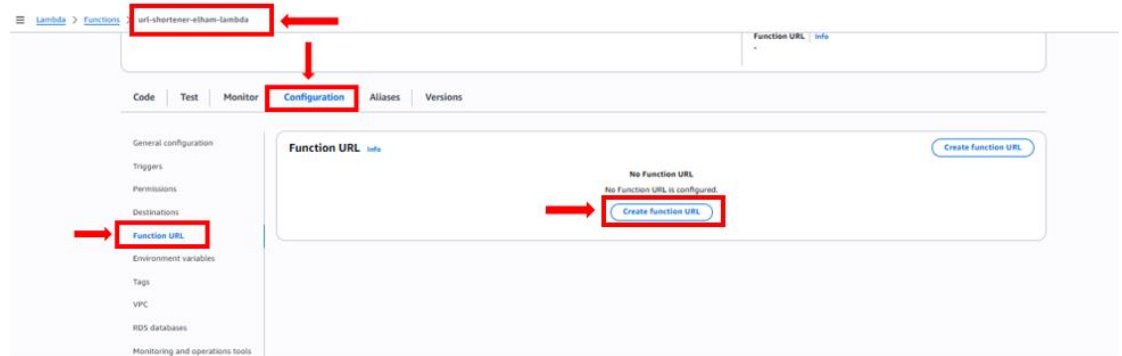
Deploy (Ctrl+Shift+L)

Test (Ctrl+Shift+G)

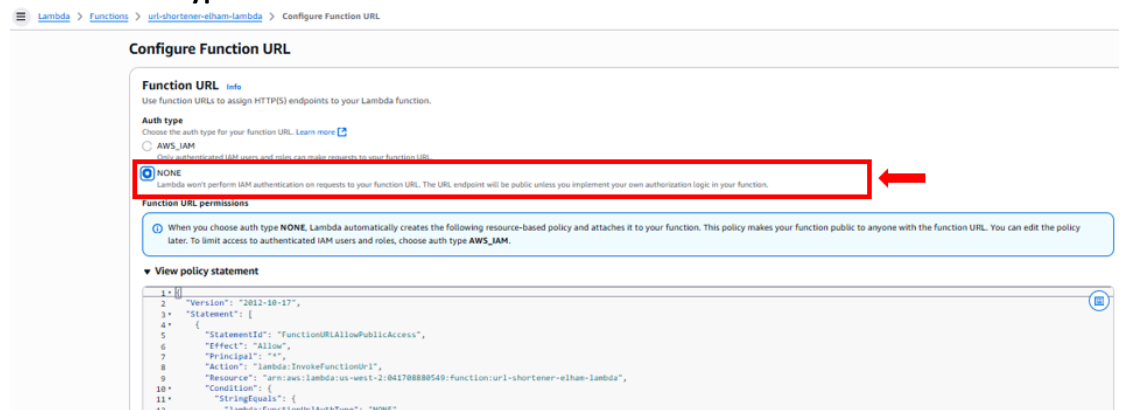
TEST EVENTS (NONE SELECTED)
 + Create new test event...

✓ Step 3: Enable Lambda Function URL

1. Go to your Lambda → Select **Configuration** → **Function URL** → Click **Create Function URL**.

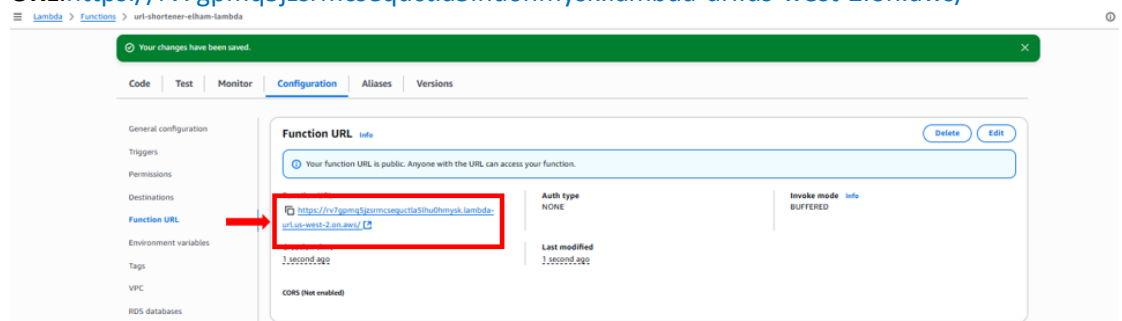


2. Choose: **Auth type: NONE**.



3. Copy the **Function URL**.

URL: <https://rv7gpmq5jzsrmscseuctla5lhu0hmysk.lambda-url.us-west-2.on.aws/>



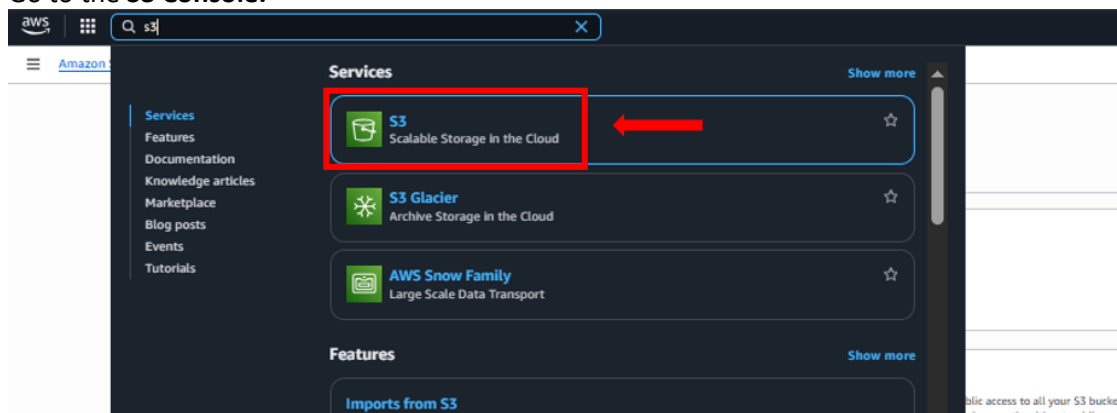
✓ Step 4: Connect Frontend to Lambda

1. In your `index.html`, update this line: `const lambdaUrl = "https://your-lambda-url.on.aws/";`

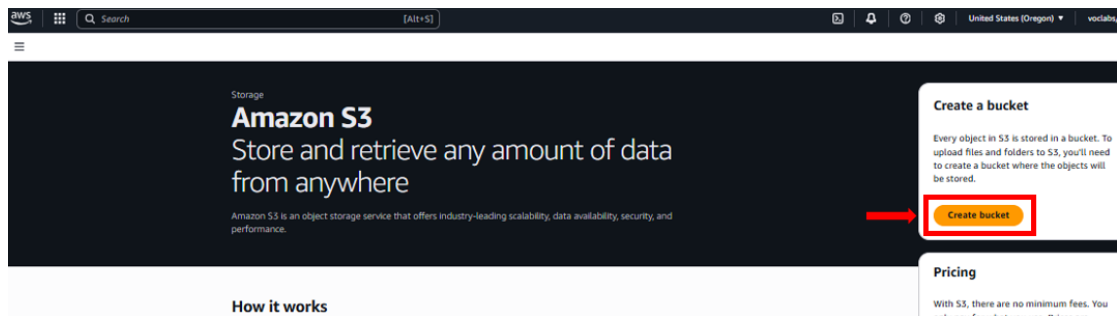
```
<script>  
  const lambdaUrl = "https://rv7gpmq5jzsrcmcsequctla5lhu0hmysk.lambda-url.us-west-2.on.aws/";
```

✓ Step 5: Create an S3 Bucket and Host the Frontend

- a. Go to the **S3 Console**.



- b. Click **Create bucket**.



- c. Choose a name : "url-shortener-elham-site".

Amazon S3 > Buckets > Create bucket

Create bucket [Info](#)

Buckets are containers for data stored in S3.

General configuration

AWS Region
US West (Oregon) us-west-2

Bucket type [Info](#)

☒ **General purpose**
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

☐ **Directory**
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class to store objects and process data within a single Availability Zone.

Bucket name [Info](#)

url-shortener-elham-site

Bucket names must be 3 to 63 characters and unique within the global namespace. Bucket names must also begin and end with a letter or number. Valid characters are a-z, 0-9, periods (.), and hyphens (-). [Learn More](#)

Copy settings from existing bucket - optional
Only the bucket settings in the following configuration are copied.

[Choose bucket](#)

Format: s3://bucket/prefix

- d. Under object ownership choose ACLs enabled.

Object Ownership [Info](#)

Control ownership of objects written to this bucket from other AWS accounts and the use of access control lists (ACLs). Object ownership determines who can specify access to objects.

☐ **ACLs disabled (recommended)**
All objects in this bucket are owned by this account. Access to this bucket and its objects is specified using only policies.

☒ **ACLs enabled**
Objects in this bucket can be owned by other AWS accounts. Access to this bucket and its objects can be specified using ACLs.

Object Ownership

☒ **Bucket owner preferred**
If new objects written to this bucket specify the bucket-owner-full-control canned ACL, they are owned by the bucket owner. Otherwise, they are owned by the object writer.

☐ **Object writer**
The object writer remains the object owner.

Warning: We recommend disabling ACLs, unless you need to control access for each object individually or to have the object writer own the data they upload. Using a bucket policy instead of ACLs to share data with users outside of your account simplifies permissions management and auditing.

Info: If you want to enforce object ownership for new objects only, your bucket policy must specify that the bucket-owner-full-control canned ACL is required for object uploads. [Learn more](#)

- e. Under **Block public access (bucket settings)**, Uncheck **Block all public access** and then :
- ✓ Uncheck the **first & second option only**.
 - ✓ Acknowledge the public warning.

Amazon S3 > Buckets > Create bucket

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. If you require public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☐ **Block all public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings is independent of one another.

☐ **Block public access to buckets and objects granted through new access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources.

☐ **Block public access to buckets and objects granted through any access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.

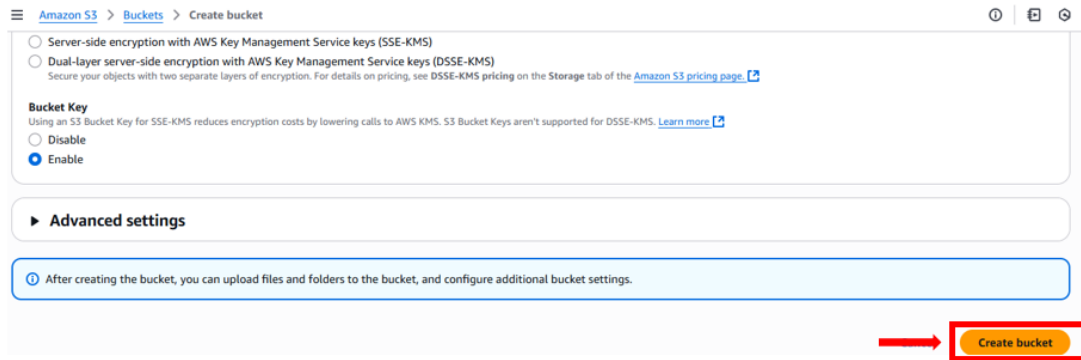
☒ **Block public access to buckets and objects granted through new public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

☒ **Block public and cross-account access to buckets and objects through any public bucket or access point policies**
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

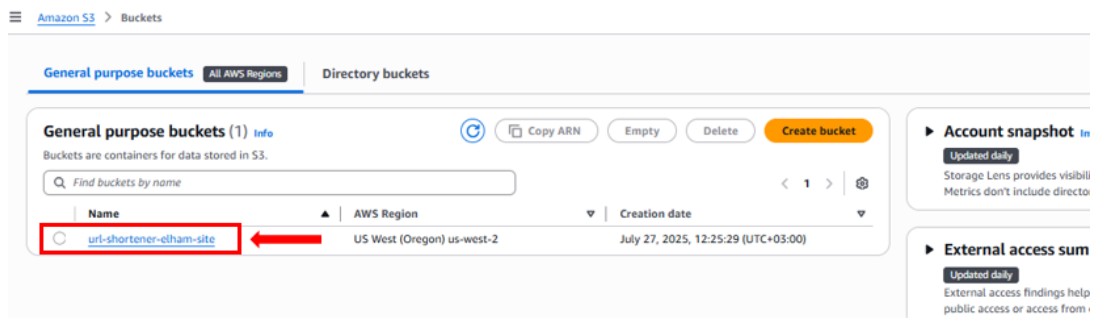
Warning: Turning off block all public access might result in this bucket and the objects within becoming public. If you have sensitive data, you should consider using cases such as static website hosting.

☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

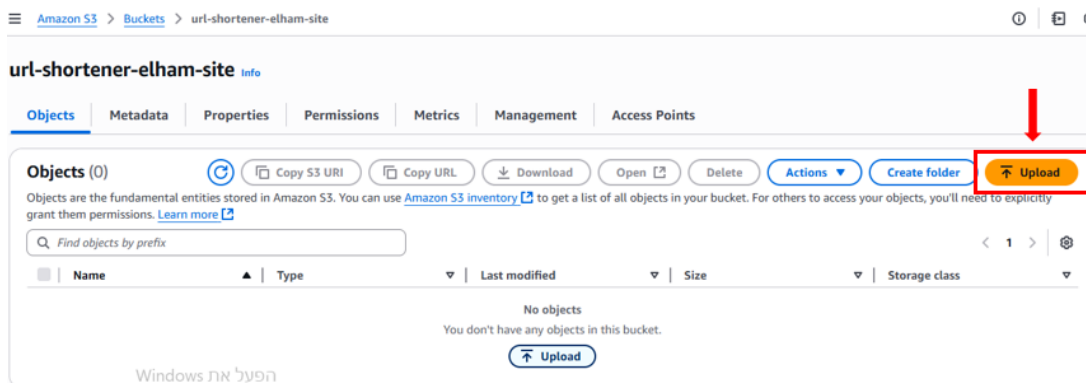
f. Click create.



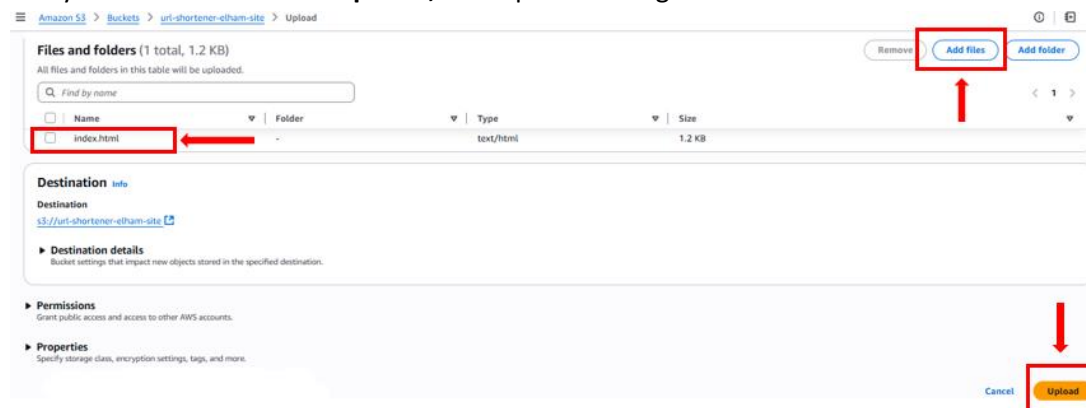
g. Successfully created "url-shortner-elham-site" bucket.



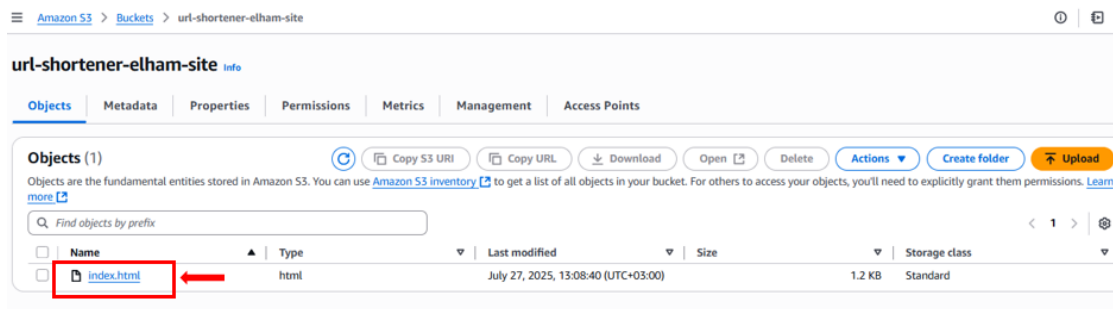
h. Click **upload** so you can add your files (**index.html** , **Style.css** , **background.jpg**) .



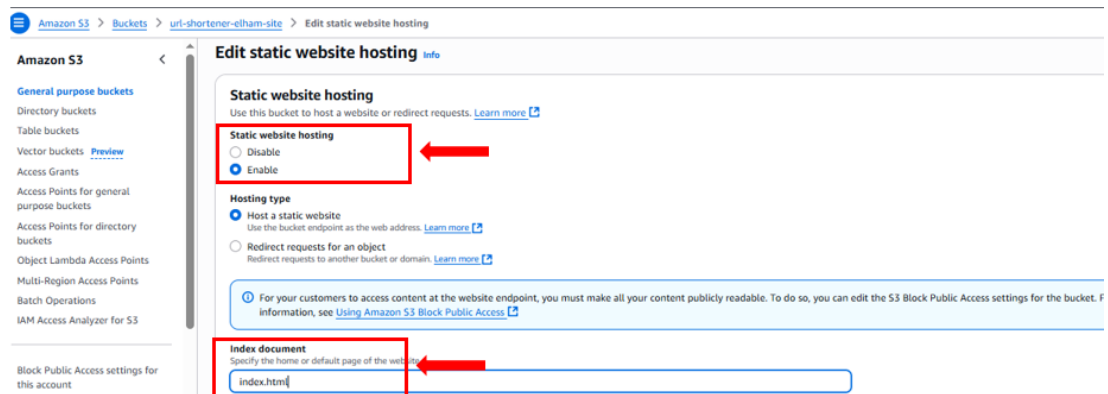
i. **Add** your file and then click **upload** , example of adding **index.html** file .



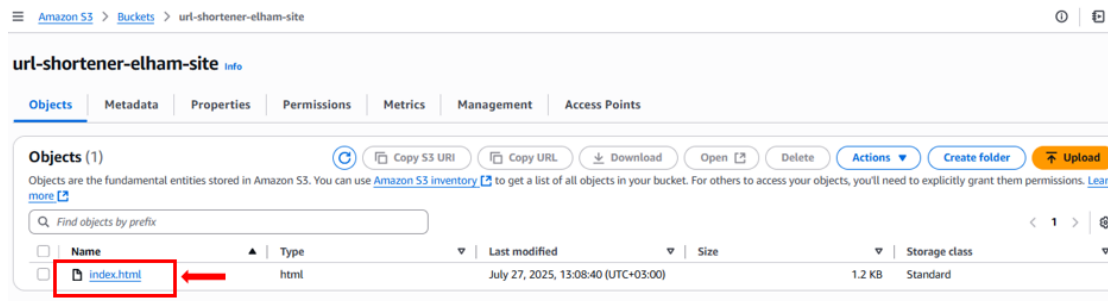
- j. Successfully uploaded the **index.html** file , add the rest files in the same way.



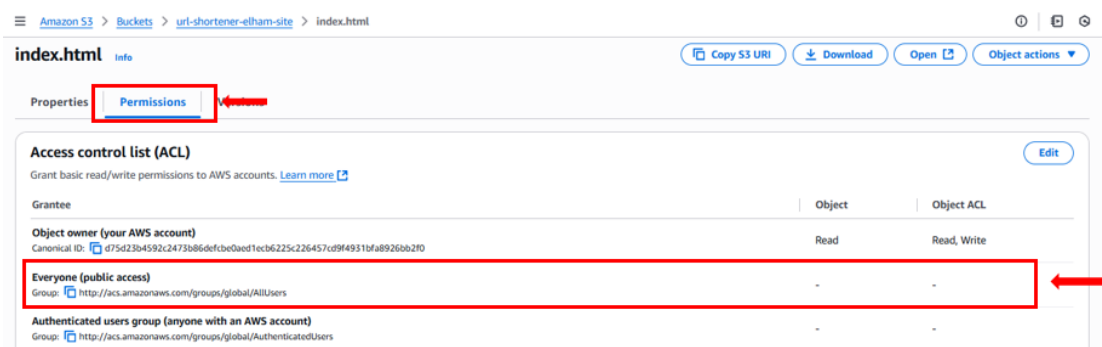
- k. Enable **Static website** hosting in the **Properties** tab and Set index document to **index.html**.



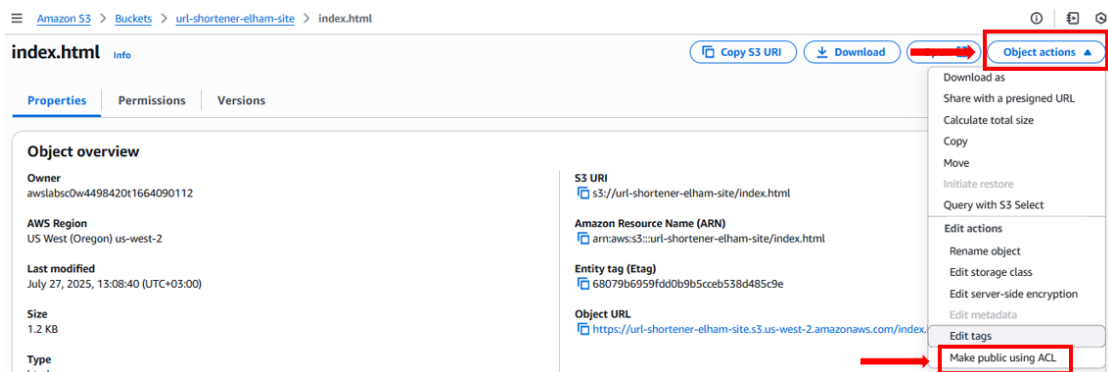
- l. Check the file permissions (for example : **index.html** file), click on the file name .



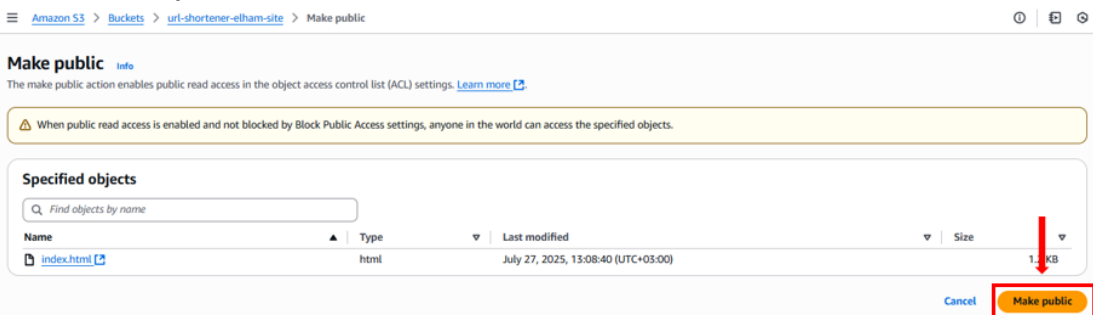
- m. Click on **permissions**, you can see that there is no **public read access for everyone**.



- n. Make the files **public** to allow public read access for everyone, click on **object actions** for every file and then choose "**Make public using ACL**", for example : **index.html**.

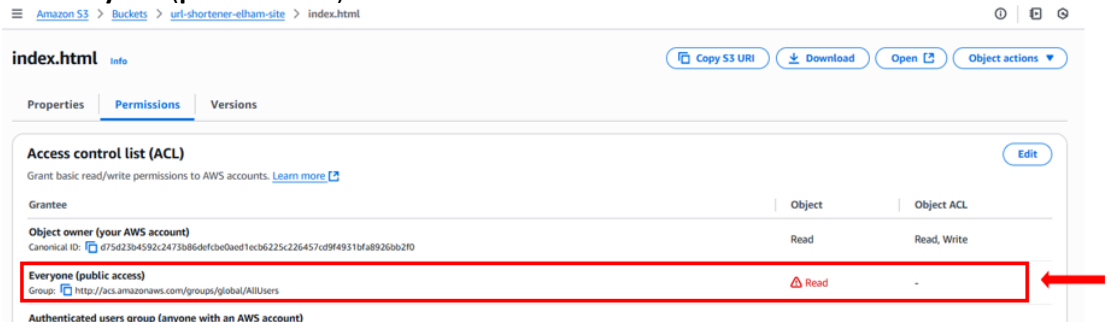


- o. Click on **make public**.

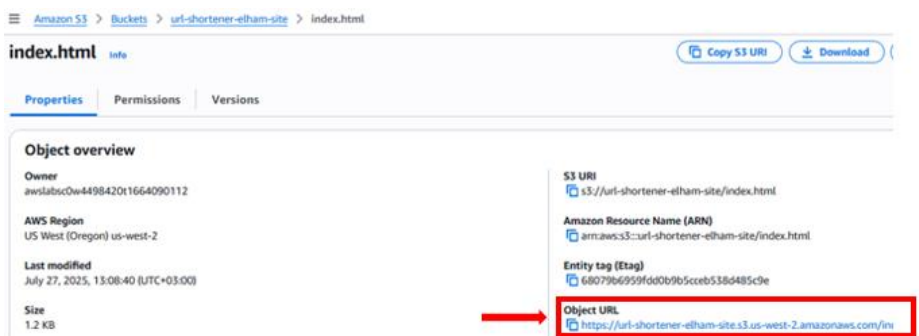


- p. You can check all files **permissions** , for example : **index.html**.

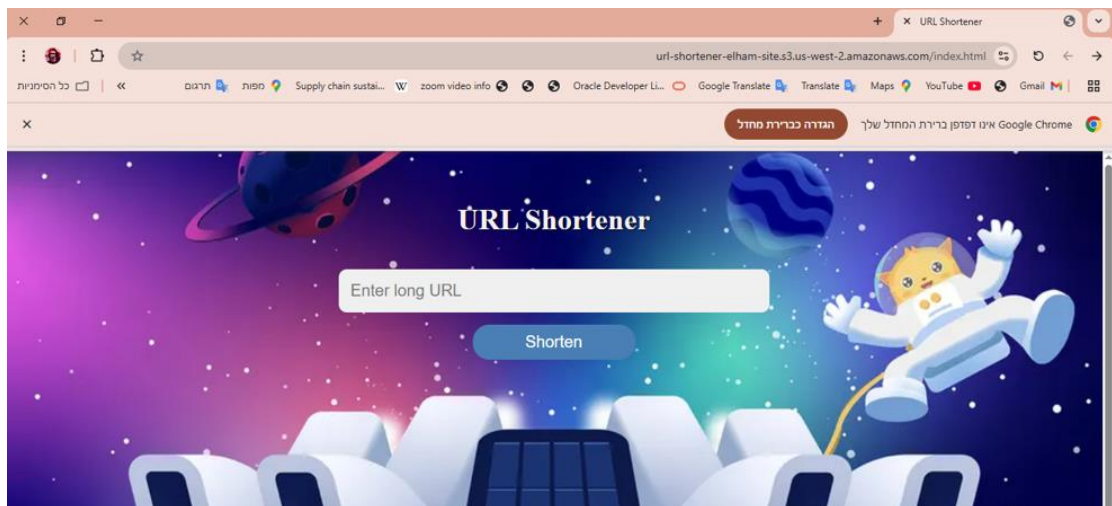
✓ "**Everyone (public access)**" → Read



- q. Get the **website URL** : <https://url-shortener-elham-site.s3.us-west-2.amazonaws.com/index.html> .



- r. Open the **URL** in new tab.



✓ Step 6: Test the Flow

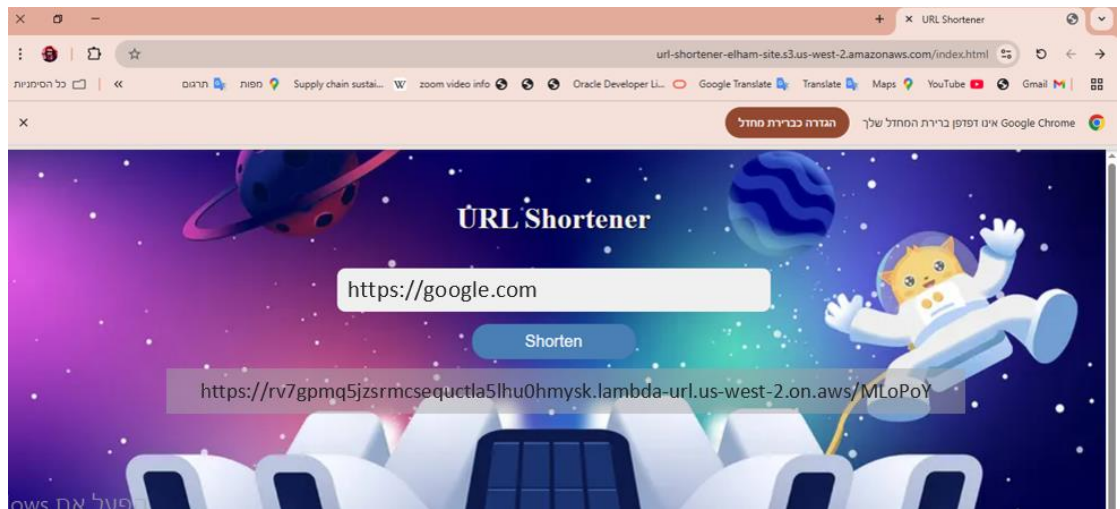
1. Open your static site URL (from S3 hosting tab): <https://url-shortener-elham-site.s3.us-west-2.amazonaws.com/index.html>
2. Paste a long URL like <https://google.com>
3. The table stores mappings like:

id	url
MLoPoY	https://google.com

The screenshot shows the AWS DynamoDB console interface. On the left, the 'ShortUrls' table is selected. The main panel shows a scan of the table with 4 items returned. One item is highlighted with a red box and a red arrow pointing to it from the table above. This item has the 'id' 'MLoPoY' and the 'url' 'https://google.com'.

id (String)	url
Tru8D	https://translate.google.com/tl-rw&hl=ar&text=Block%20public%20access%20to%20buckets%20and%20objects%20granted%20through%20new%20access%20...
MLoPoY	https://google.com
Em1X0d	https://mail.google.com/mail/u/0/#inbox

4. Click **Shorten**
5. You'll get a short URL like:
<https://rv7gpmq5jzsrcmsequctla5lhu0hmysk.lambda-url.us-west-2.on.aws/MLoPoY> .



6. Click the short URL — it should redirect to the original one ✓