S3 - Creating a personalised and publicly accessed website using S3 bucket

->> Name of the bucket should be UNIQUE
->> ACLs should be enabled for the general access of the objects like other AWS accounts

> An **ACL** = a set of rules that decides who can get access to a resource.
>
> Enabling ACLs in this S3 setup lets you control who can access and do things with the objects (i.e. website files) you upload into your bucket.
>
> With ACLs, different AWS accounts can own and control different files in your bucket.

->> Uncheck the _Block all public access_

->> Upload the project files and folders to the bucket

->> Configure your S3 bucket for static website **hosting**

- By configuring your S3 bucket for hosting, we're telling this bucket: "please create a URL that will take anyone to a page that displays the HTML file I just uploaded."

Go to bucket -> Properties -> Configure the following settings:

- **Static web hosting: Choose Enable.**
- **Hosting type: Choose Host a static website.**
- **Index document: Enter index.html**

What's a bucket website endpoint?
A bucket website endpoint is just like a regular website URL. It lets people visit your S3 bucket as a website.

->> Visit your public website link.

Upon clicking the link - > we get the 404 error:

403 Forbidden

- Code: AccessDenied
- Message: Access Denied
- RequestId: 71XX0GXVGK09GEHZ
- HostId: STfqzrZJBlWU0PyClvuwDxsrubauxQVx8Fqu0WQH7cy5KBN1aHge5oDB9Z7P2nSZQW9T3rIP1+I=
-

Why did I get this error?
Objects (in this case, the HTML and images files you uploaded) are private by default. This default setting helps keep your account's data secure.

The error message you're seeing is telling you that your static website is being hosted by S3, but the actual HTML/image files you've uploaded are still private. It's kind of like having a bucket on display, so everyone can see the bucket - but the contents are covered up, preventing anyone from seeing what's inside.

To solve this error, we need to set the permission of the objects to public - this is why we enabled **ACLs** in Task 1!

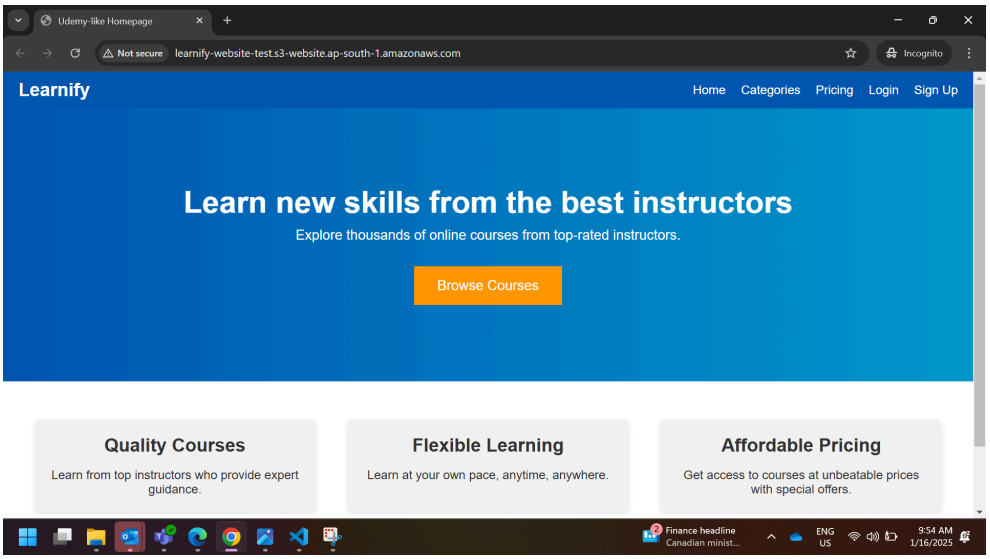
Making the objects public:

->> go to objects and select index.html -> go to Actions -> select ***Make public using ACL***

Similarly do it for the folder as well..

That's it. Now the website is live.

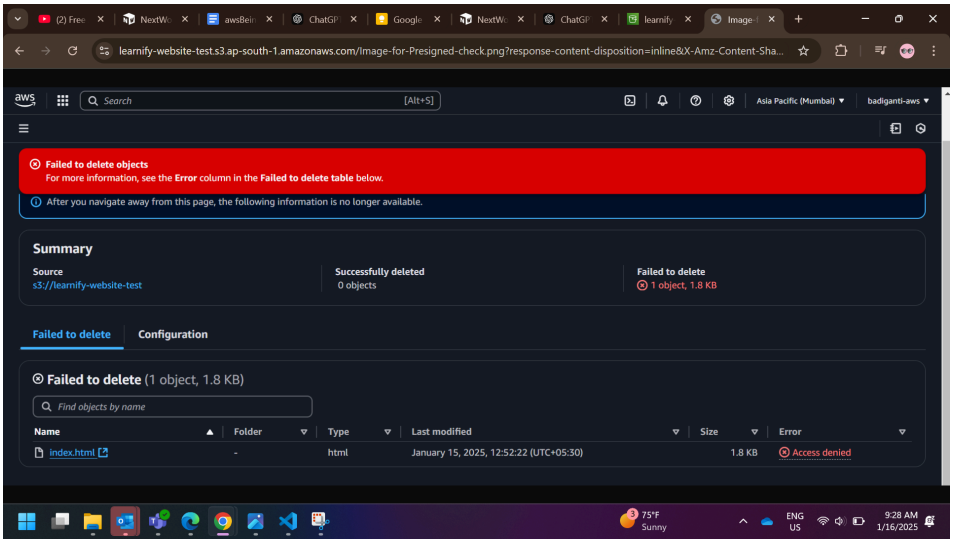
Below is the screenshot of our sample website hosted statically on aws.



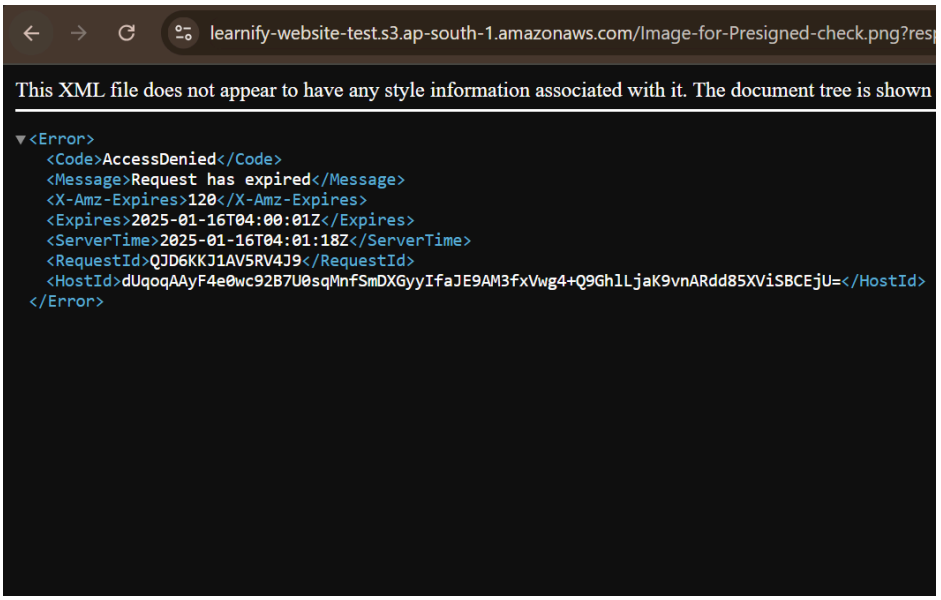
More into S3

Pre-signed URLs :

- **Definition**: A pre-signed URL is a temporary URL generated to grant time-limited access to objects in a private S3 bucket without requiring AWS credentials.
- **Use Cases**:
    - Sharing objects securely with external users.
    - Enabling temporary access to files for downloads or uploads.
- **Features**:
    - Time-limited: You can specify the expiration time for the URL.
    - Permission-limited: Access is restricted to the specified operation (e.g., GET or PUT).



The above image is obtained by using the created Pre-signed URL.



This error occurs after the expiration time set in the pre-signed URL has passed. (Request has expired)

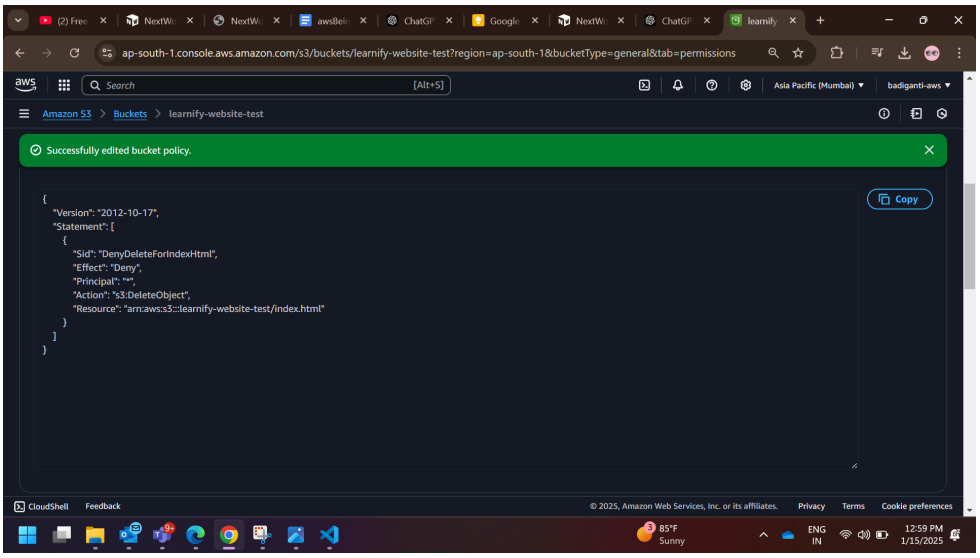
Bucket policy for secure access control :

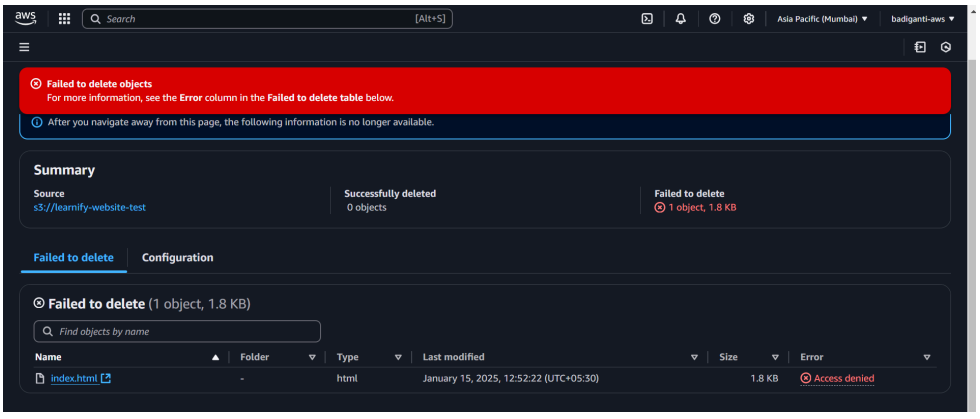
Using the bucket policy to secure the objects.

Example :

Let's say we need to secure our **index.html** file and that we do not want anyone to DELETE the file..

So now we can write the bucket policy to do this.





When tried to delete the index.html

Key Components in Policy making :

1. `Version`: **Policy language version (always `"2012-10-17"`).**
2. `Statement`: **List of rules; can include multiple.**
3. `Sid`: **Optional unique identifier for the statement.**
4. `Effect`: **Determines if the rule `Allow`s or `Deny`s the action.**
5. `Principal`: **Specifies who the policy applies to (`*` for everyone or specific users/roles).**
6. `Action`: **Defines what actions are allowed/denied (e.g., `s3:GetObject`, `s3:DeleteObject`).**
7. `Resource`: **Specifies which bucket or objects the rule applies to.**
8. `Condition`: **Optional constraints (e.g., IP, date, user-agent).**

Updating your website on Amazon S3:

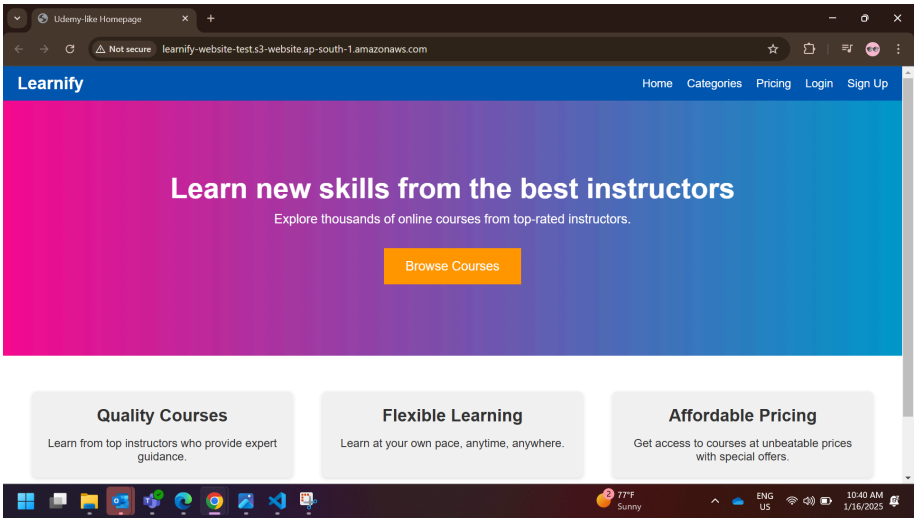
Upload the index.html without deleteing the existing file..
And make the file public using ALCs again.

That's how we can simply update our files. (a simple reupload and making ALCs public)

Bucket Versioning:

Bucket versioning in Amazon S3 is a feature that allows you to keep multiple versions of an object in the same bucket. When versioning is enabled for a bucket, S3 stores every version of an object, including the previous ones, whenever you upload or modify an object. This ensures that you can retrieve older versions of an object if needed.

By default, bv is not enabled. By enabling the bv, changed and deleted versions of the files are saved.



Updated styles.css

Route 53:

Route 53 is a scalable and highly available Domain Name System (DNS) web service offered by Amazon Web Services (AWS). It performs the following functions:

1. **DNS Management**: It translates friendly domain names (like `www.example.com`) into IP addresses (like `192.0.2.1`) that computers use to connect to each other.
2. **Domain Registration**: It allows you to register new domain names and manage your domain settings.
3. **Health Checks and Monitoring**: It can monitor the health of your resources (like web servers) and automatically route traffic to healthy endpoints.
4. **Traffic Routing**: It provides different routing policies for distributing traffic across multiple resources, such as weighted routing, latency-based routing, and geolocation routing.

Certainly! Here's a summarized table that includes key **features** of **Amazon Route 53**, **common record types**, and their descriptions for easy reference in your AWS notes:

Route 53 Summary Table

Feature/Concept	Description
DNS Management	Translates domain names to IP addresses to enable communication between devices on the internet.
Domain Registration	Allows you to register new domain names or transfer existing ones to Route 53.
Health Checks	Monitors the health of your resources (e.g., web servers) and routes traffic away from unhealthy resources.
Routing Policies	Defines how Route 53 handles DNS requests based on different policies: Simple, Weighted, Latency, Geolocation, Failover.
Scalability	Can scale to handle a large volume of DNS queries, suitable for applications of any size.

Integration with AWS	Fully integrates with other AWS services (e.g., EC2, S3, CloudFront, Elastic Load Balancing) for streamlined resource management.
Global Infrastructure	Provides low-latency DNS resolution through a globally distributed network of DNS servers.
Traffic Routing	Route traffic based on user location, load balancing, or health checks to improve performance and availability.
Cost-Effective	Pay-as-you-go pricing model, with charges based on the number of hosted zones, queries, and health checks.

Common Record Types in Route 53

Record Type	Description	Use Case Example
A Record	Maps a domain name to an **IPv4** address (e.g., `192.0.2.1`).	`www.example.com -> 192.0.2.1` (Static website)
AAAA Record	Maps a domain name to an **IPv6** address (e.g., `2001:0db8::1`).	`www.example.com -> 2001:0db8::1` (IPv6 website)
CNAME Record	Maps a domain name to another domain name (e.g., redirecting traffic to another domain). **Does not point to an IP address.**	`www.example.com -> example.com` (Alias record)
TXT Record	Used to store arbitrary text data, often for verification purposes (e.g., SPF, DKIM).	`example.com -> "v=spf1 include:_spf.google.com ~all"`

MX Record	Specifies mail servers that handle email for the domain.	`example.com -> mail.example.com` (Email routing)
NS Record	Specifies authoritative DNS servers for a domain. Typically provided by your DNS hosting provider.	`example.com -> ns-123.awsdns.com`
SOA Record	Specifies the **Start of Authority** for the domain, including information about the DNS zone, such as the primary DNS server and the time-to-live (TTL).	Auto-generated in hosted zones
PTR Record	Used for **reverse DNS lookups**, mapping an IP address to a domain name.	`192.0.2.1 -> example.com` (Reverse DNS lookup)
SRV Record	Specifies services offered by a domain, including the protocol, port, and target machine for the service.	`_sip._tcp.example.com` (Service discovery)
CAA Record	Specifies which Certificate Authorities (CAs) are allowed to issue SSL/TLS certificates for the domain.	`example.com -> 0 issue "letsencrypt.org"` (Security)
Alias Record	Similar to a CNAME record, but allows mapping to AWS resources (e.g., ELB, CloudFront, S3 buckets). Can be used at the root domain (which CNAME cannot).	`example.com -> s3-website-us-east-1.amazonaws.com`

Routing Policies

Routing Policy	Description	Use Case Example
Simple Routing	Directs traffic to a single resource.	Pointing `www.example.com` to one EC2 instance.
Weighted Routing	Distributes traffic across multiple resources based on assigned weights.	80% traffic to primary EC2, 20% to a secondary instance.
Latency-based Routing	Routes traffic to the resource with the lowest latency based on the user's location.	Direct traffic to EC2 in closest AWS region.
Geolocation Routing	Routes traffic based on the geographical location of the user.	Route U.S. traffic to U.S. servers, EU traffic to EU.
Failover Routing	Routes traffic to a secondary resource if the primary resource becomes unhealthy.	Primary web server goes down, traffic automatically rerouted to backup server.

See you on my next one! 👋