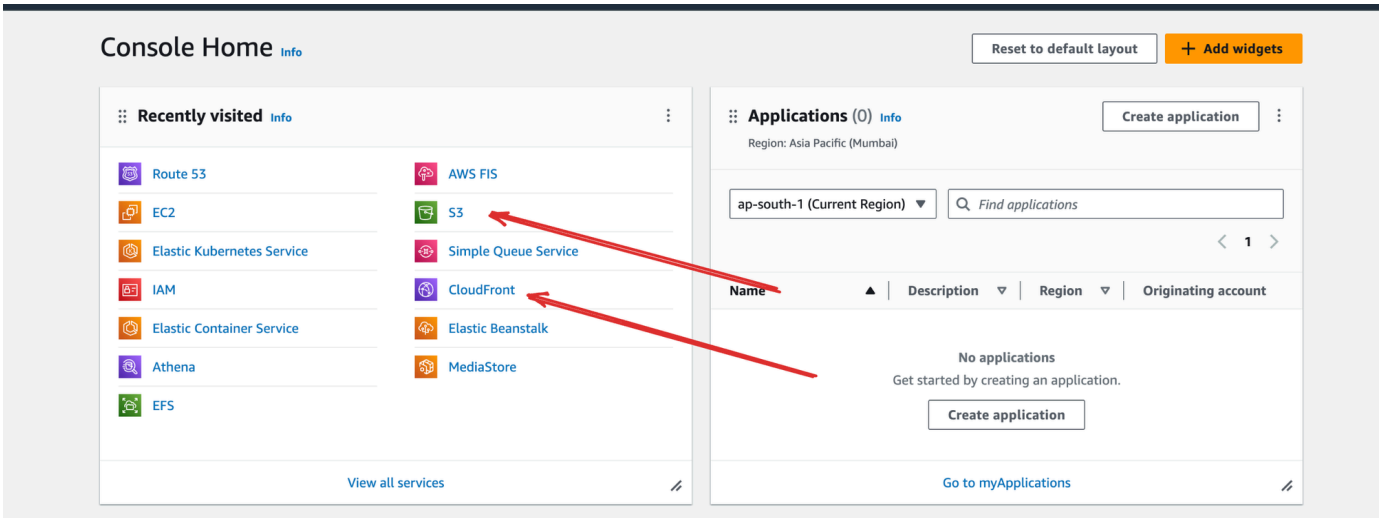
# Step 1 - How to deploy Frontends to AWS

New things we will learn include
1. Object stores (S3)
2. CDNs (Cloudfront)

Step 1 - Signup and get an AWS account.

Step 2 - Make sure you can access S3 and cloudfront (this will automatically happen if you are the root user of that account)



# Step 2 - Build your React frontend

💡 This approach will not work for frameworks that use Server side rendering (like Next.js)
This will work for basic React apps, HTML/CSS/JS apps

## Go to your react project

```
cd /link/to/your/react/project
```
Copy

## Build your project

```
npm run build
```
Copy

## Try serving the HTML/CSS/JS locally

```
npm i -g serve
serve
```
Copy

At this point you have basic HTML/CSS/JS code that you can deploy on the internet.

You might be tempted to host this on an EC2 instance, but that is not the right approach

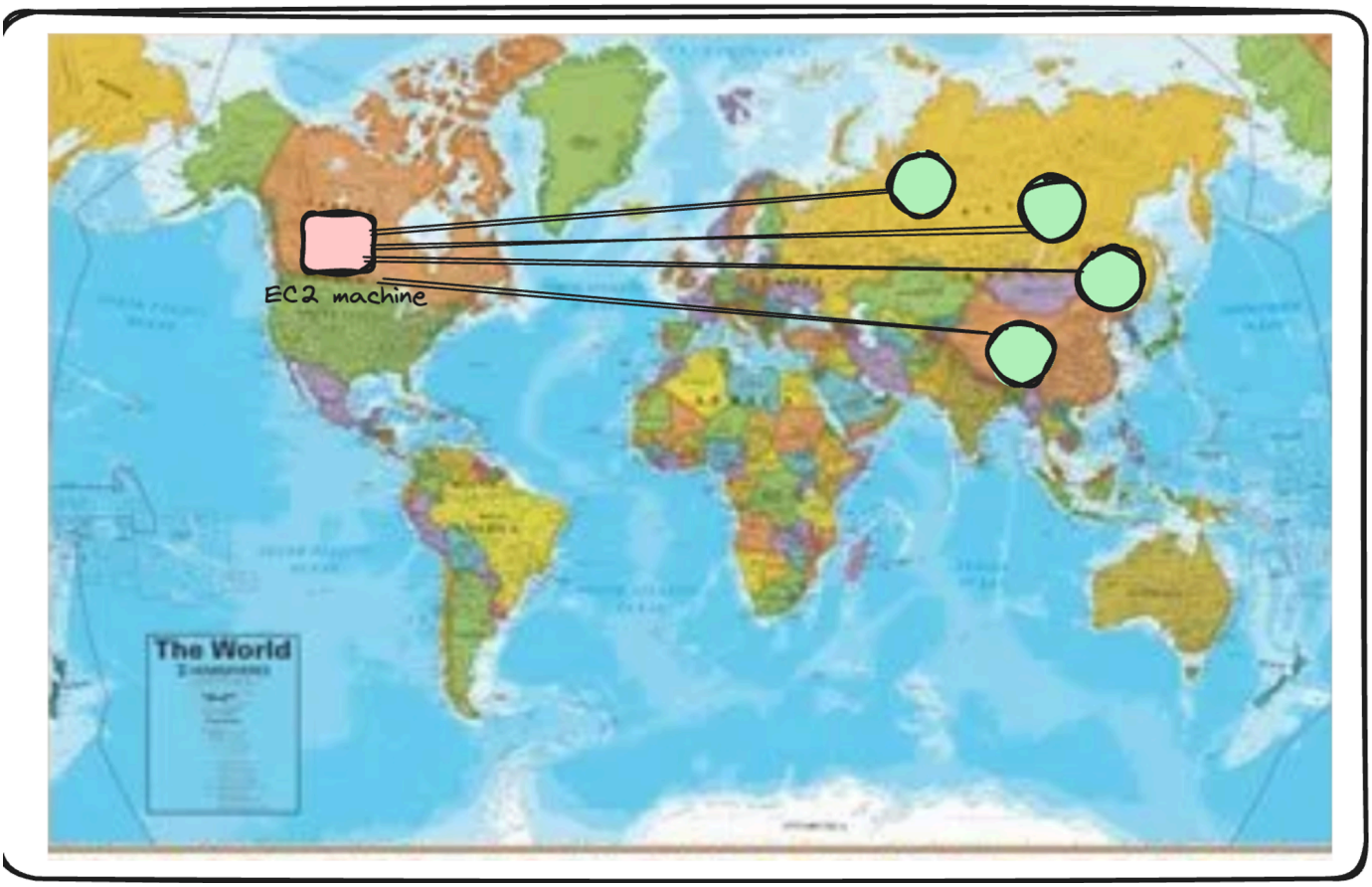
The next slide explains why

# Step 3 - What are CDNs?

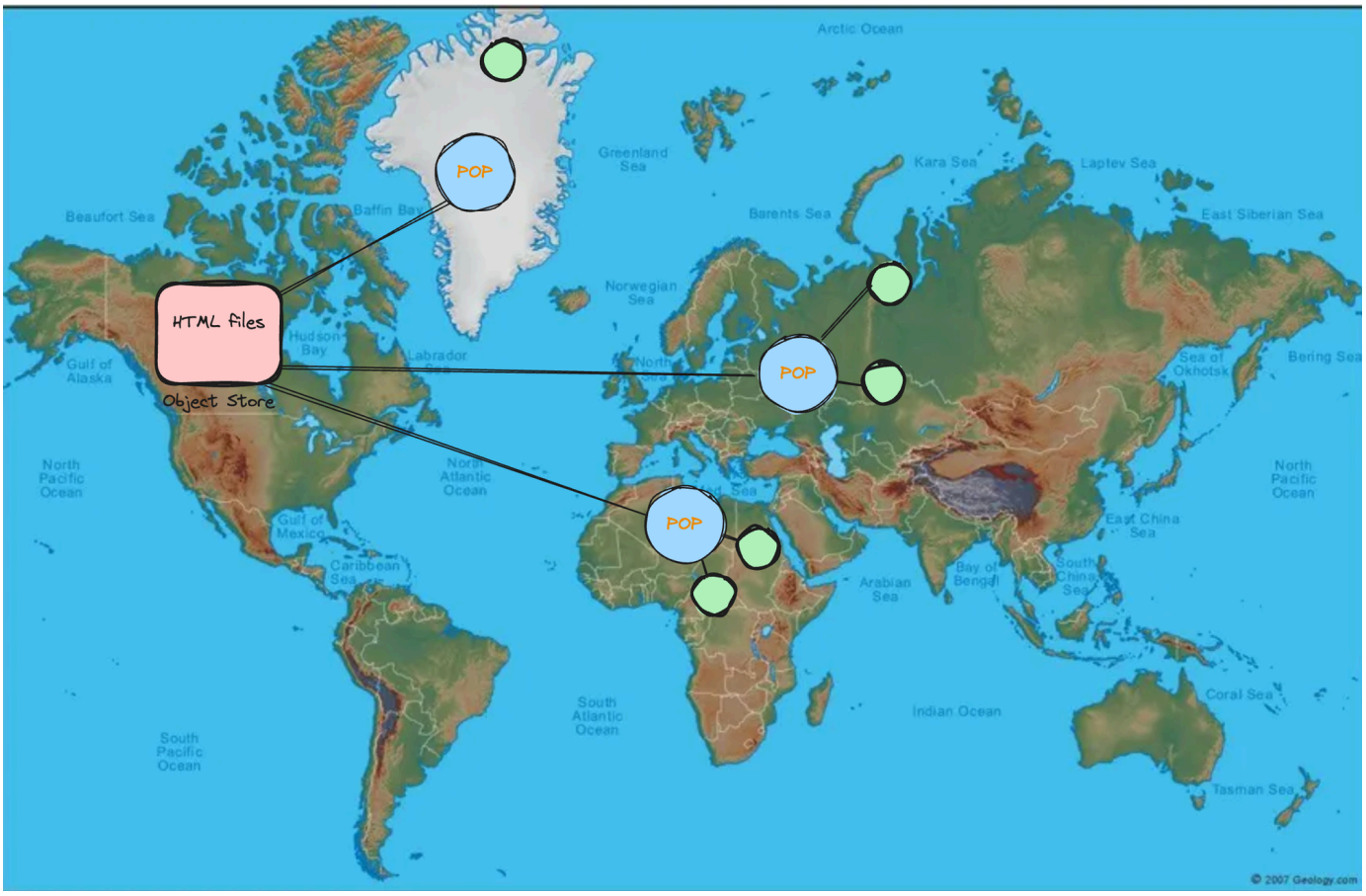
A CDN stands for `Content Delivery Network` .
As the name suggests, it's an optimal way for you to deliver content (mp4 files, jpgs and even HTML/CSS/JS files) to your users.

It is better than serving it from a VM/EC2 instances because of a few reasons -

## 1. EC2 machine apprach



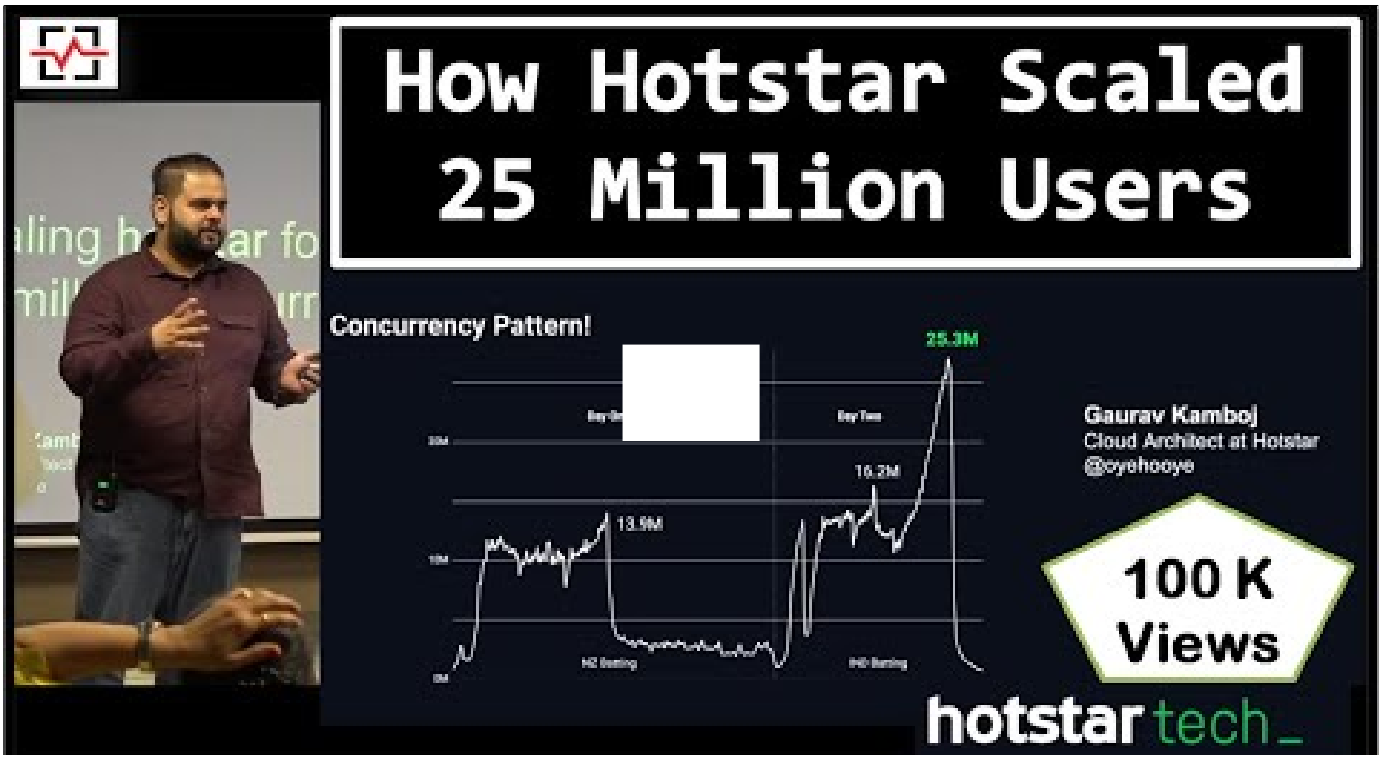
## 2. CDN approach




1. For frontends, mp4 files, images, `Object stores` + `CDNs` are a better approach.
2. You can't use the same for backends, since every request returns a different response. Caching doesn't make any sense there.

> 💡 You can use edge networks for backends (deploy your backend on various servers on the internet) but data can't be cached in there.

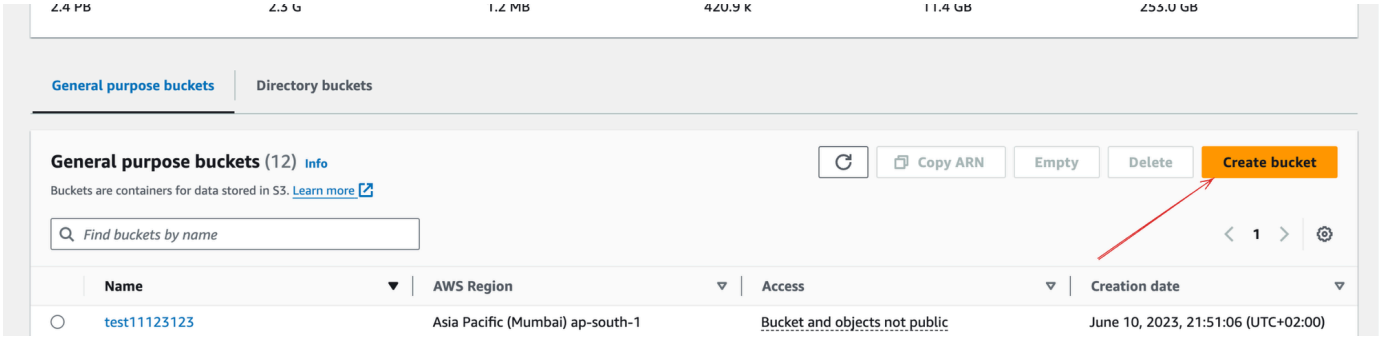
Great video on how Hotstar scales their infrastructure during cricket matches (they use CDNs heavily)

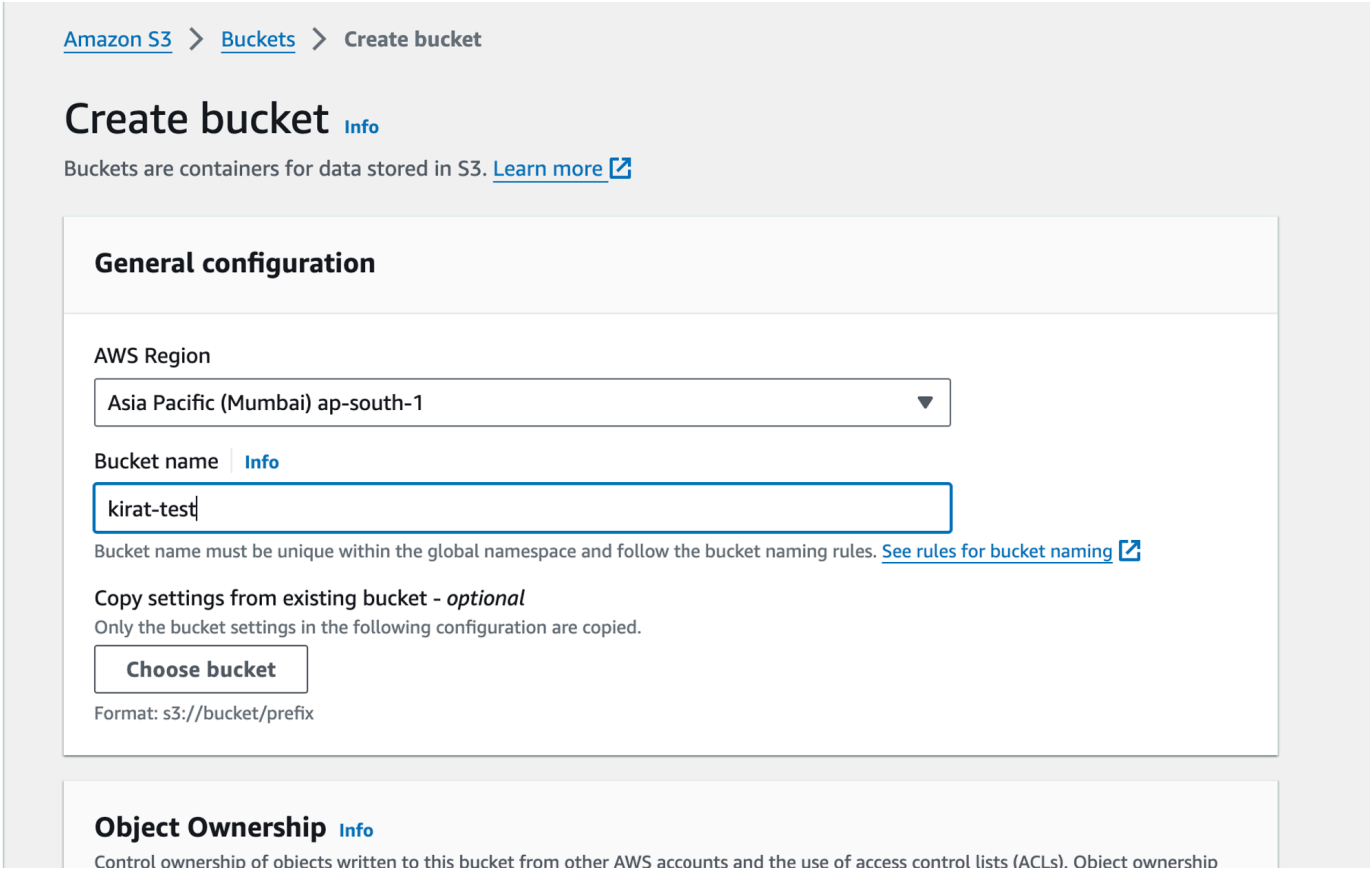


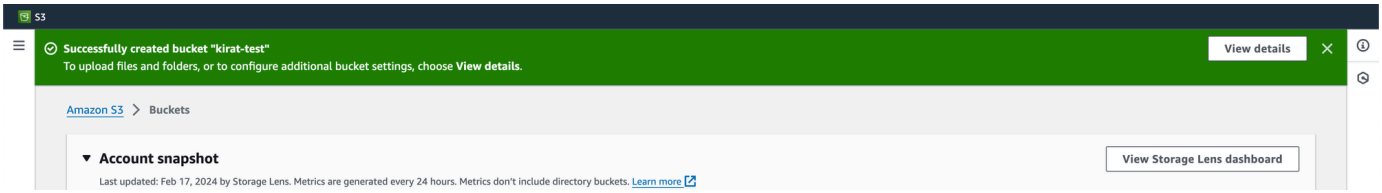

# Step 4 - Creating an object store in AWS

In AWS, S3 is their object store offering.

You can create a `bucket` in there. A `bucket` represents a logical place where you store all the files of a certain project.

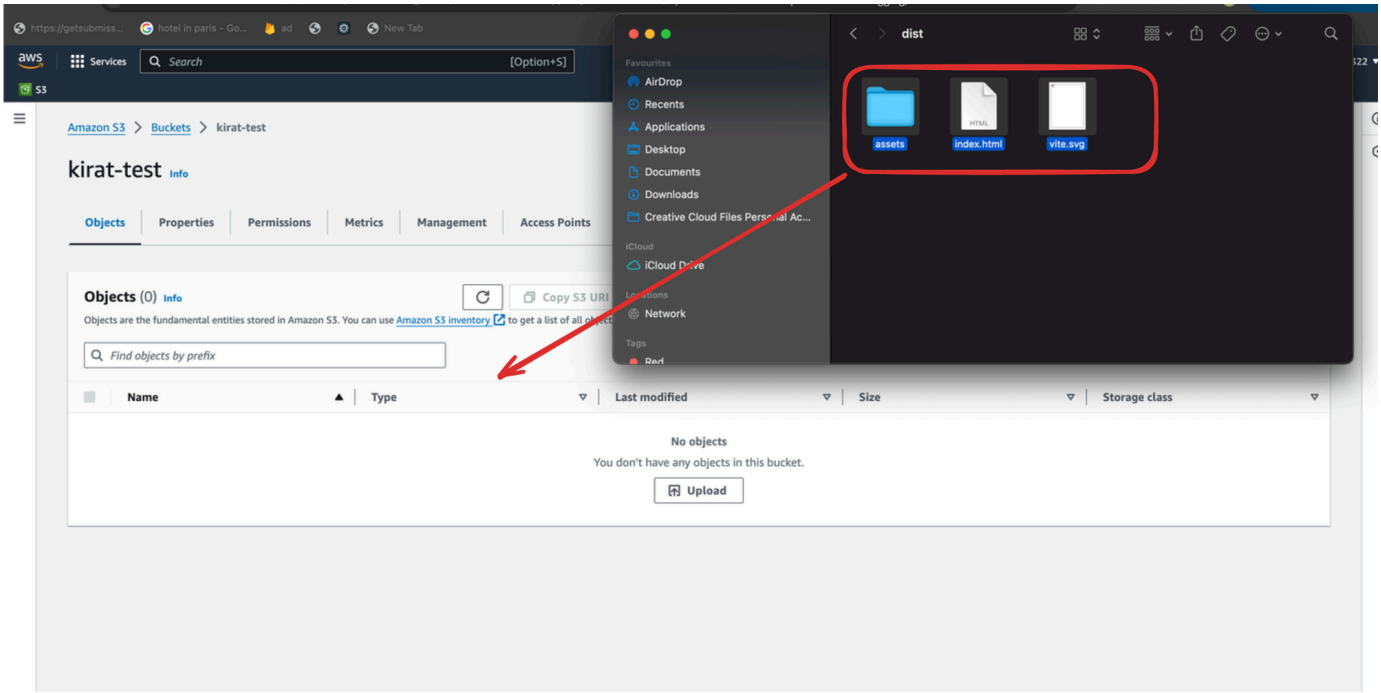




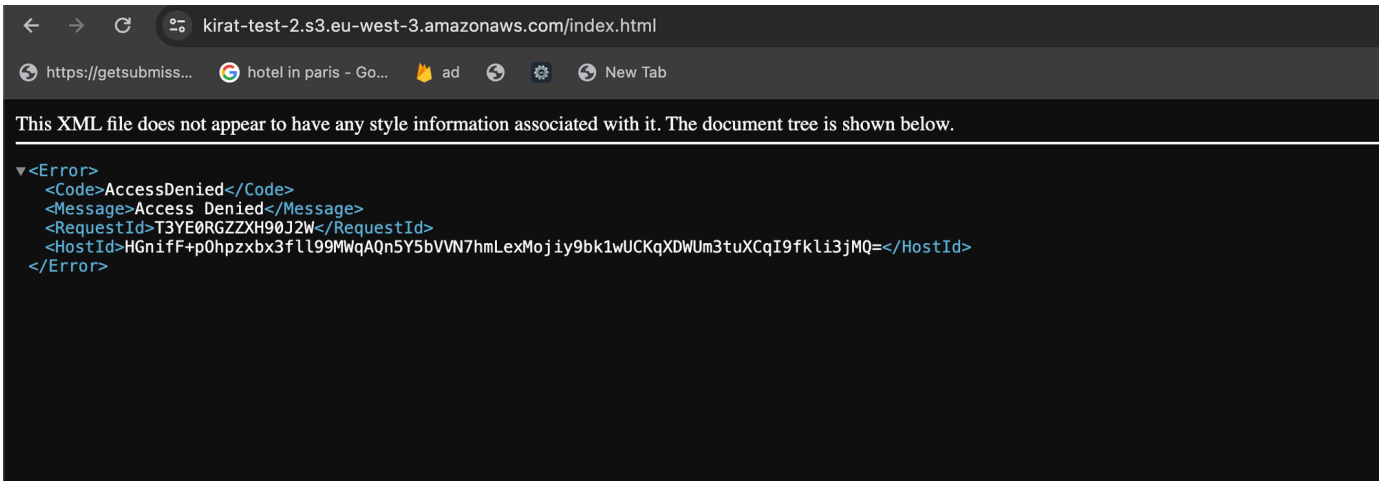




# Step 5 - Upload the file bundle to S3

Upload all the files in the `dist` folder of your react project to S3




# Step 6 - Try accessing the website



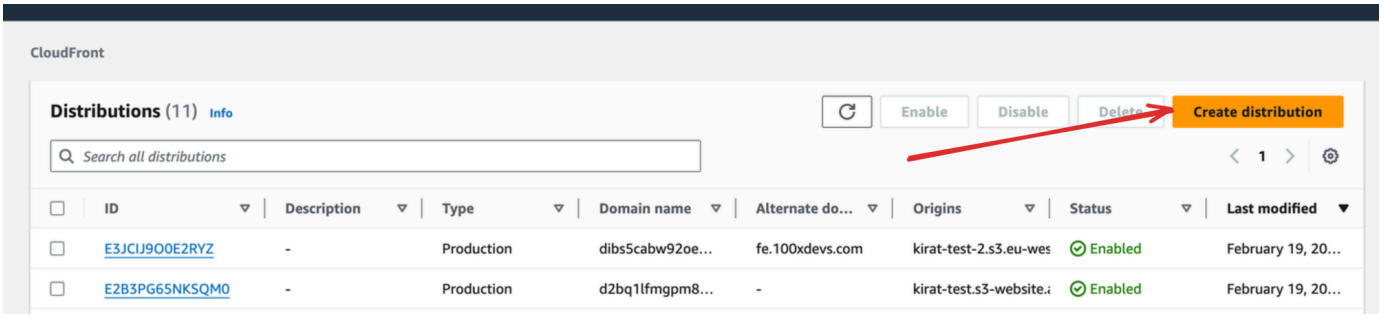

You might be tempted to open your S3 bucket at this point, but don't

Your S3 bucket should be blocked by default, and you should allow cloudfront (CDN) to access it.

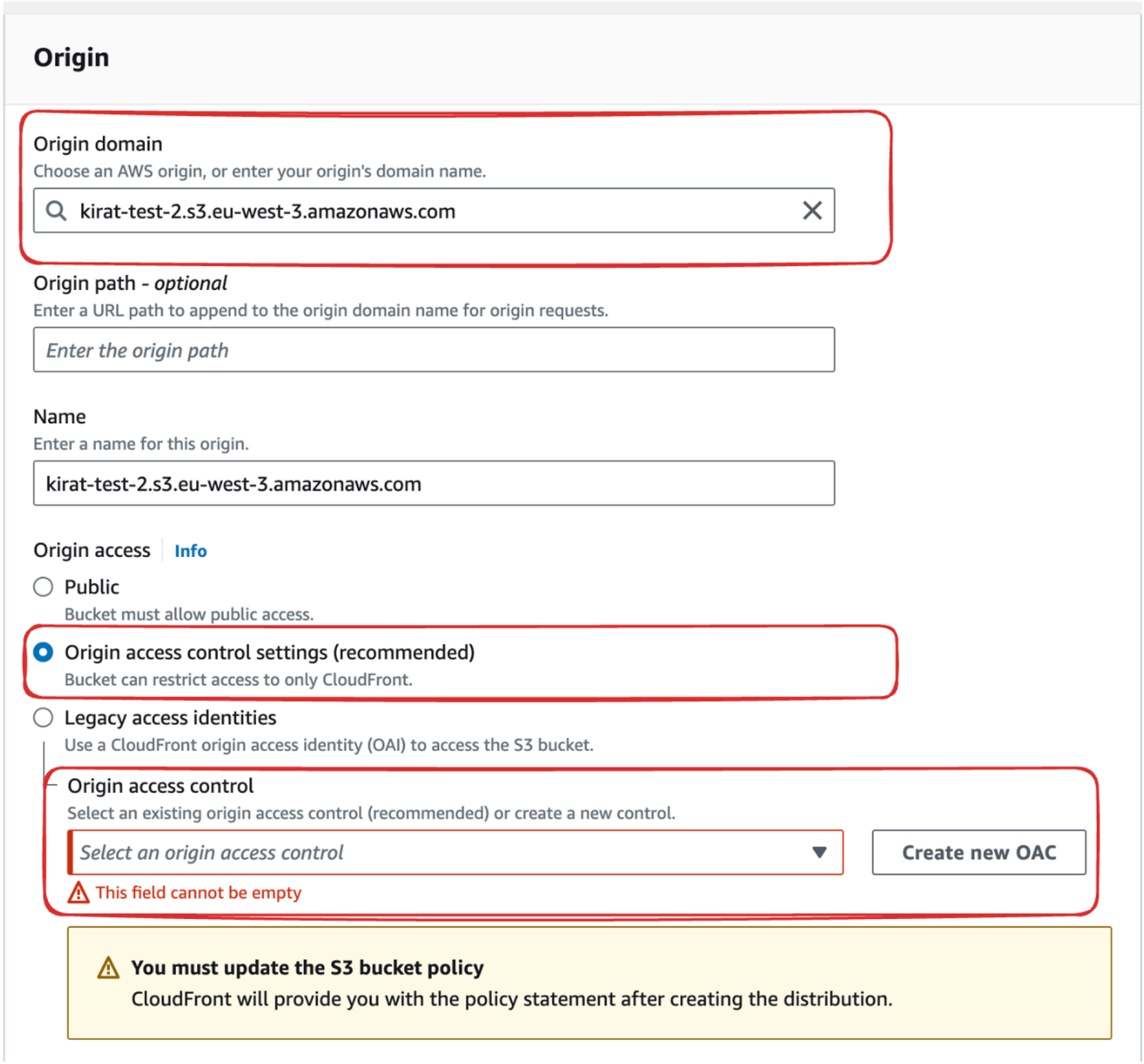
# Step 7 - Connecting Cloudfront

## Step 1 - Create cloudfront distribution

Go to cloudfront and create a new distribution. A `distribution` here means you're creating a place from where `content` can be distributed.



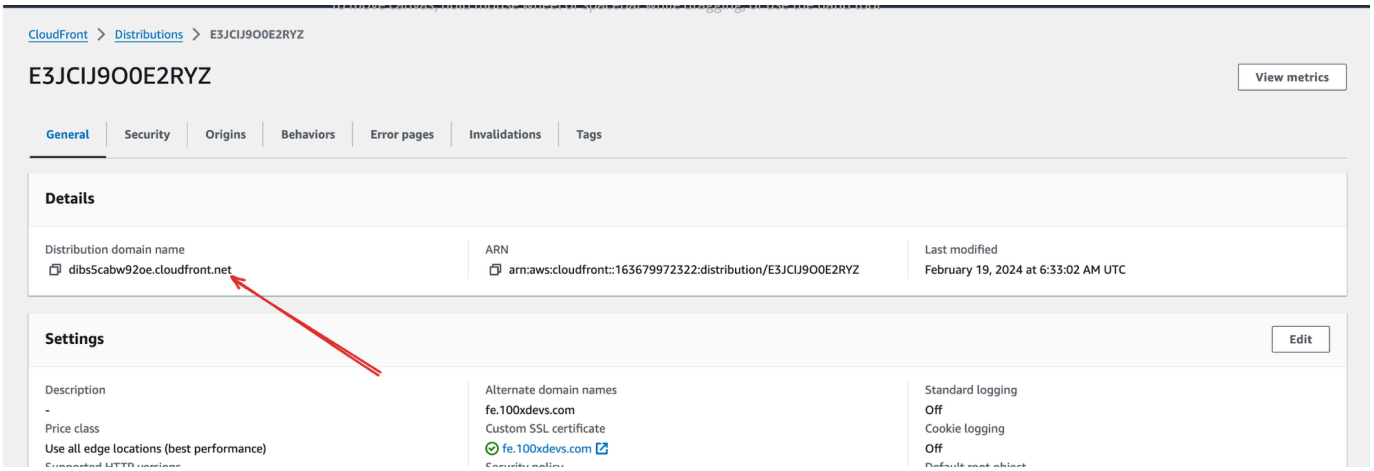

## Step 2 - Select your S3 bucket as the source




Origin Access Control (OAC) is a feature in Cloudfront, which allows you to restrict direct access to the content stored in your origin, such as an Amazon S3 bucket or a

web server, ensuring that users can only access the content through the CDN distribution and not by directly accessing the origin URL.

By the end of this, you should have a working cloudfront URL.

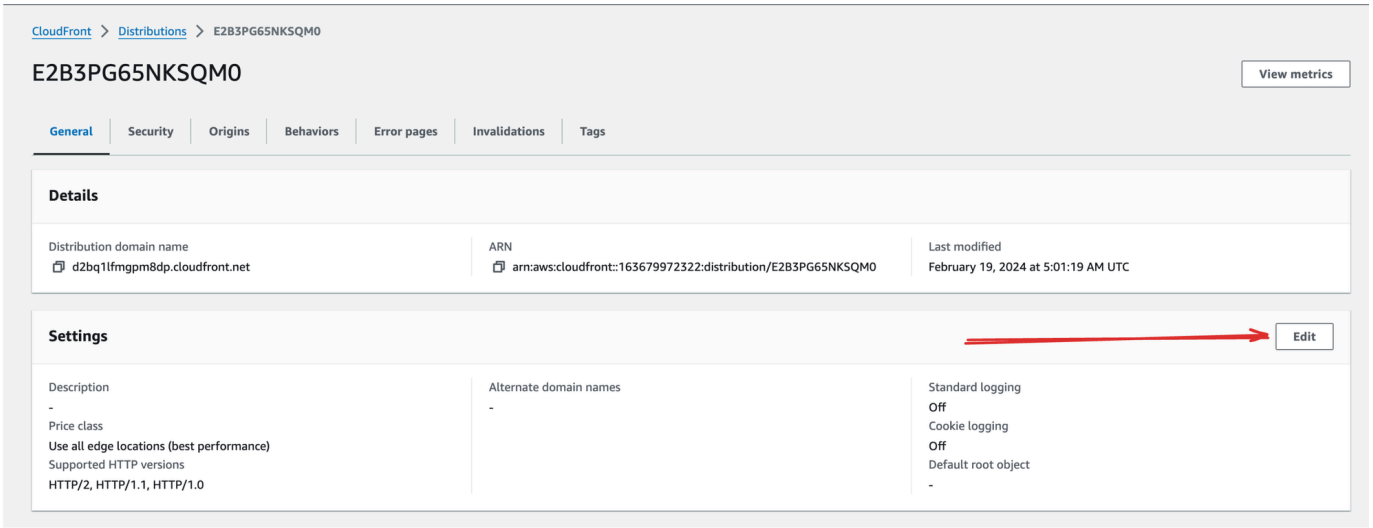



# Step 8 - Connect your own domain to it

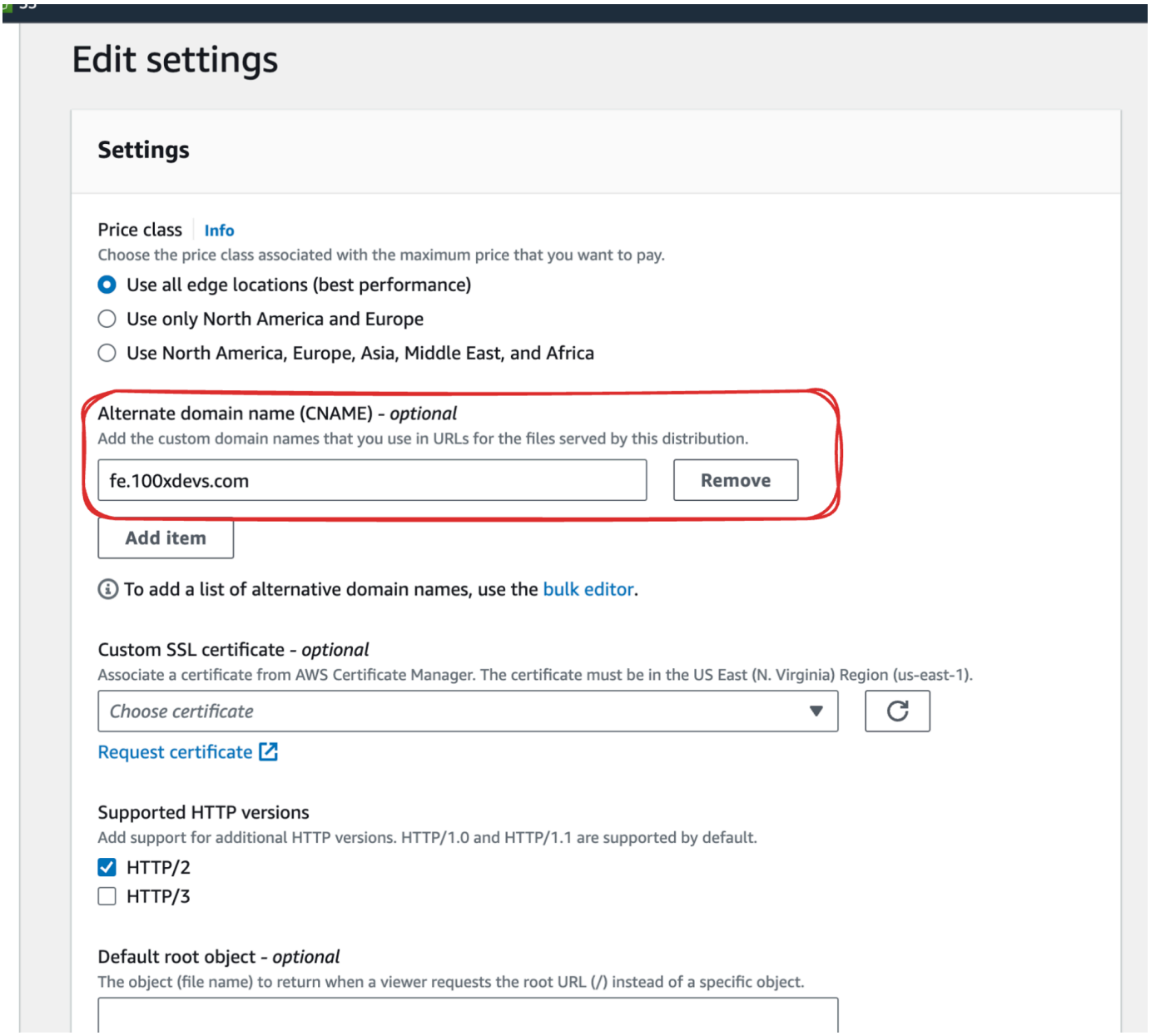
Websites aren't fun if you have to go to a URL that looks like this - https://dibs5cabw92oe.cloudfront.net

Connect your own custom domain by following the given steps -

## 1. Select edit on the root page

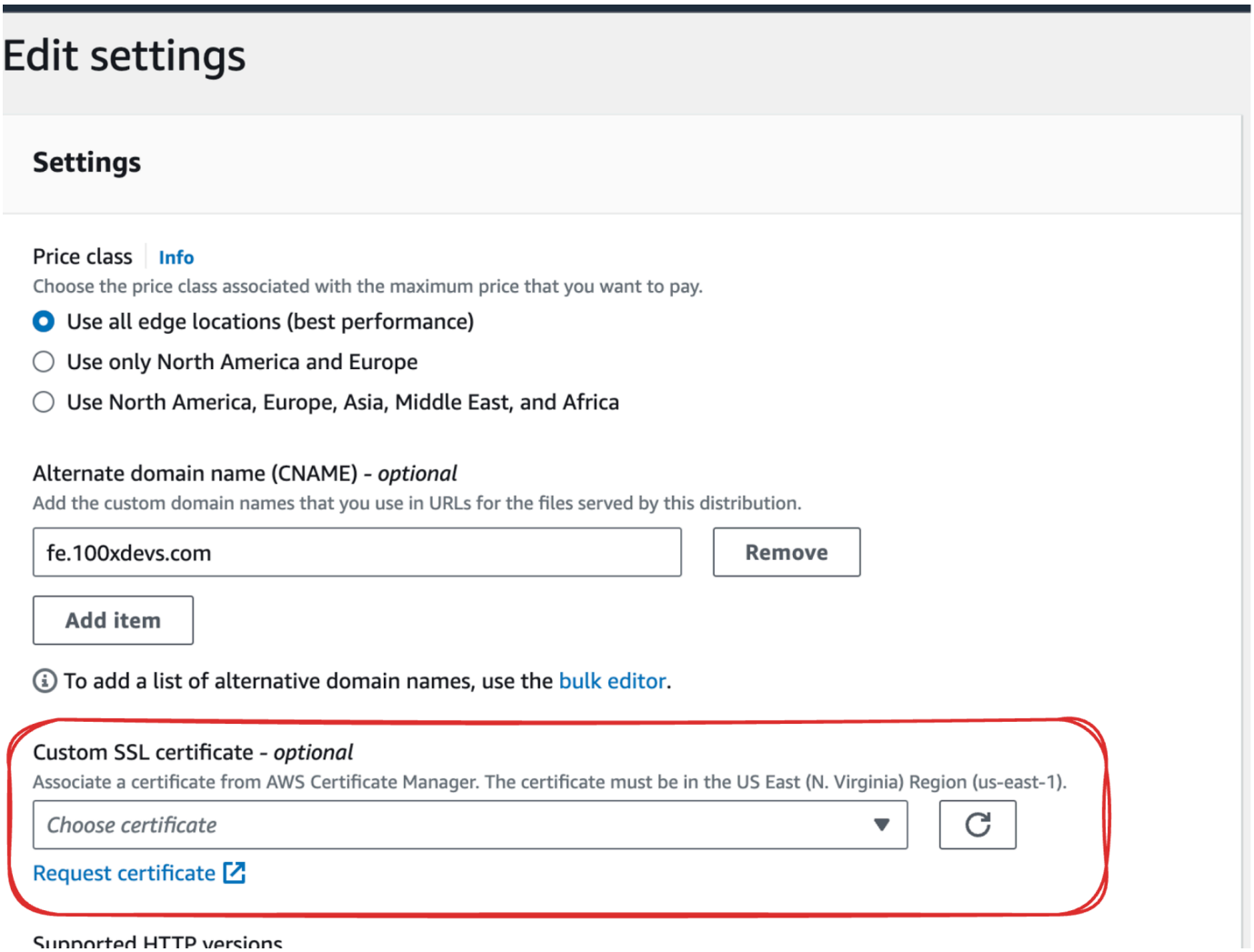


## 2. Attach a domain name to the distribution

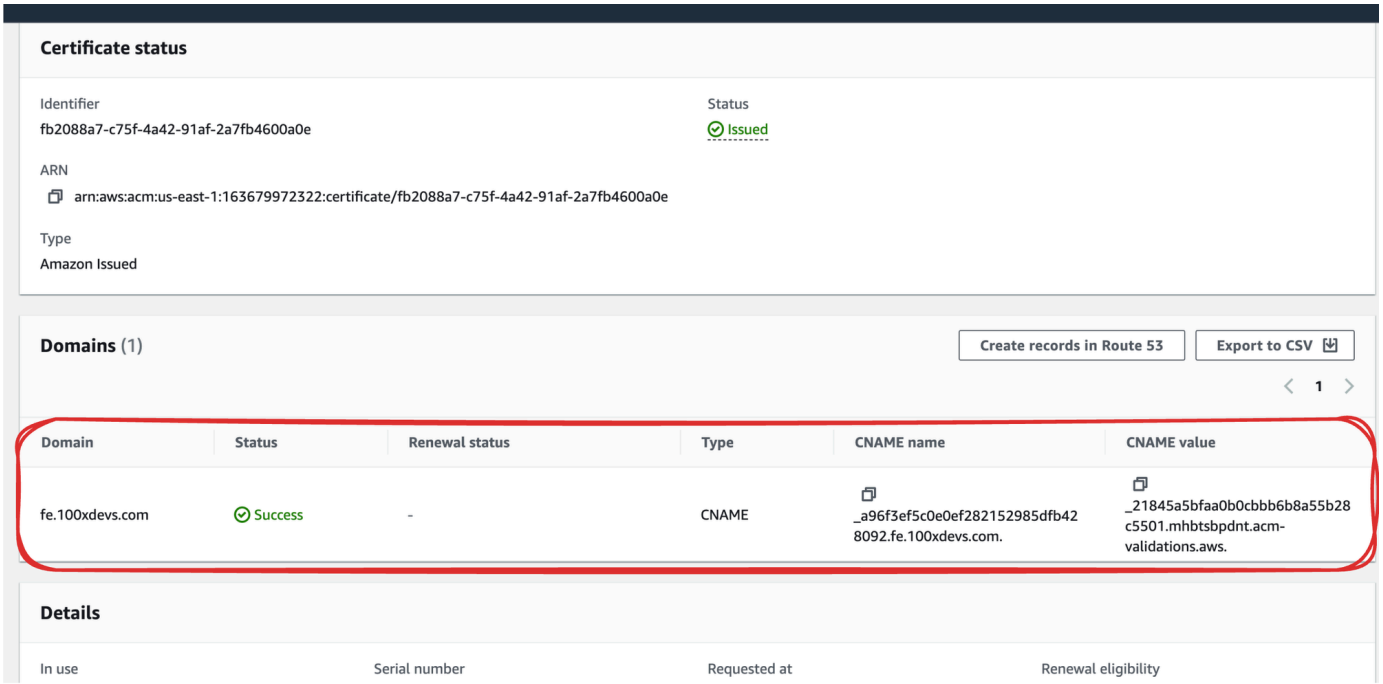


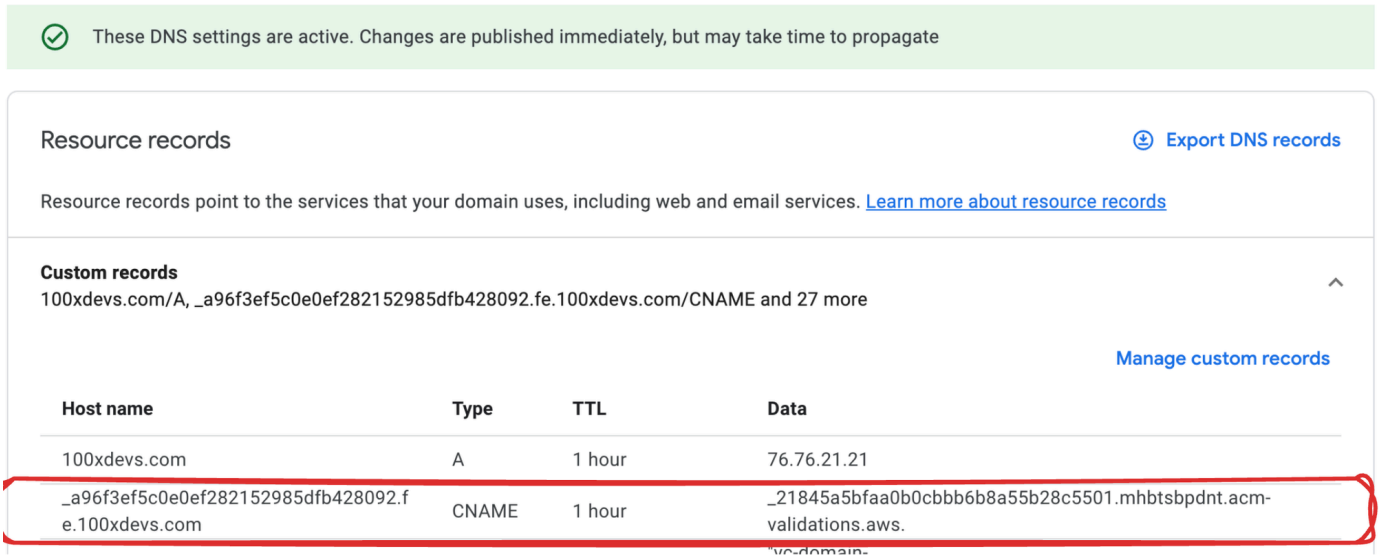

## 3. Create a certificate

Since we want our website to be hosted on HTTPS, we should request a certificate for our domain




## Step 4 - Follow steps to create the certificate in the certificate manager

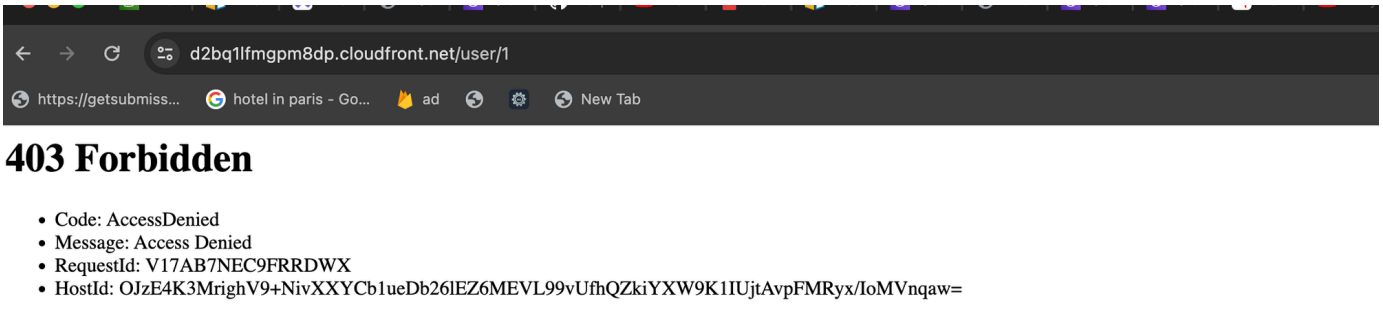





## Step 5 - Add a CNAME record for the website to point to your cloudfront URL

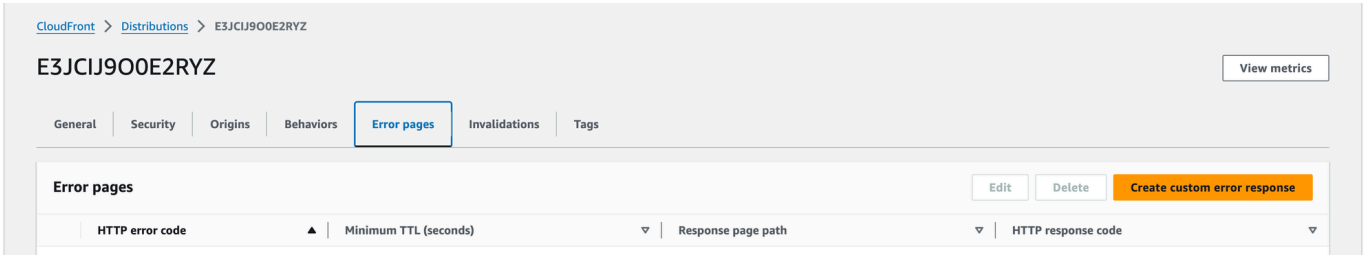

# Step 9 - Error pages

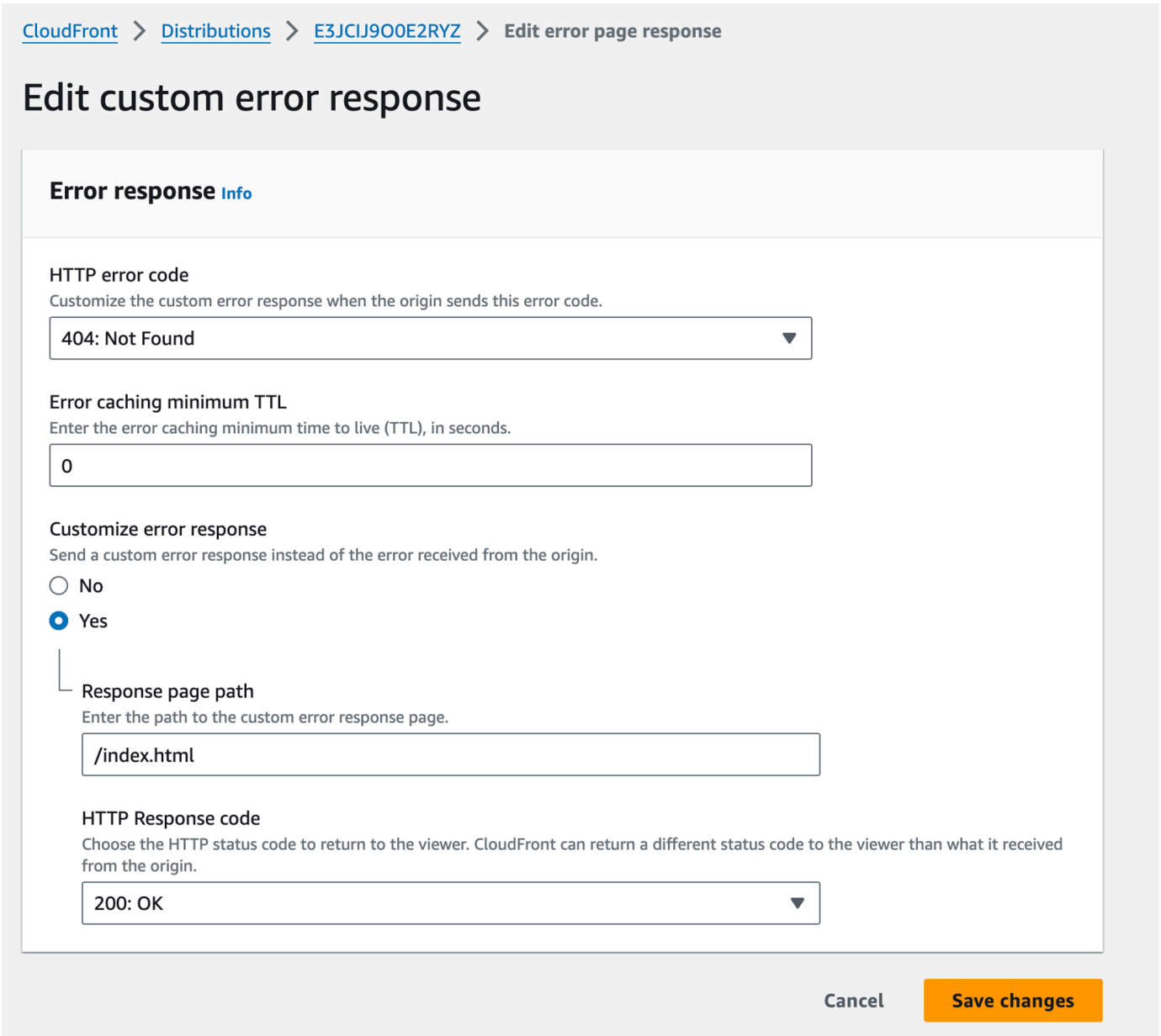
You will notice a problem, whenever you try to access a route on your page that isn't the index route (/user/1) , you reach an error page



This is because cloudfront is looking for a file `/user/1` in your S3, which doesn't exist.

To make sure that all requests reach `index.html`, add an `error page` that points to `index.html`




You might have to invalidate cache to see this in action.