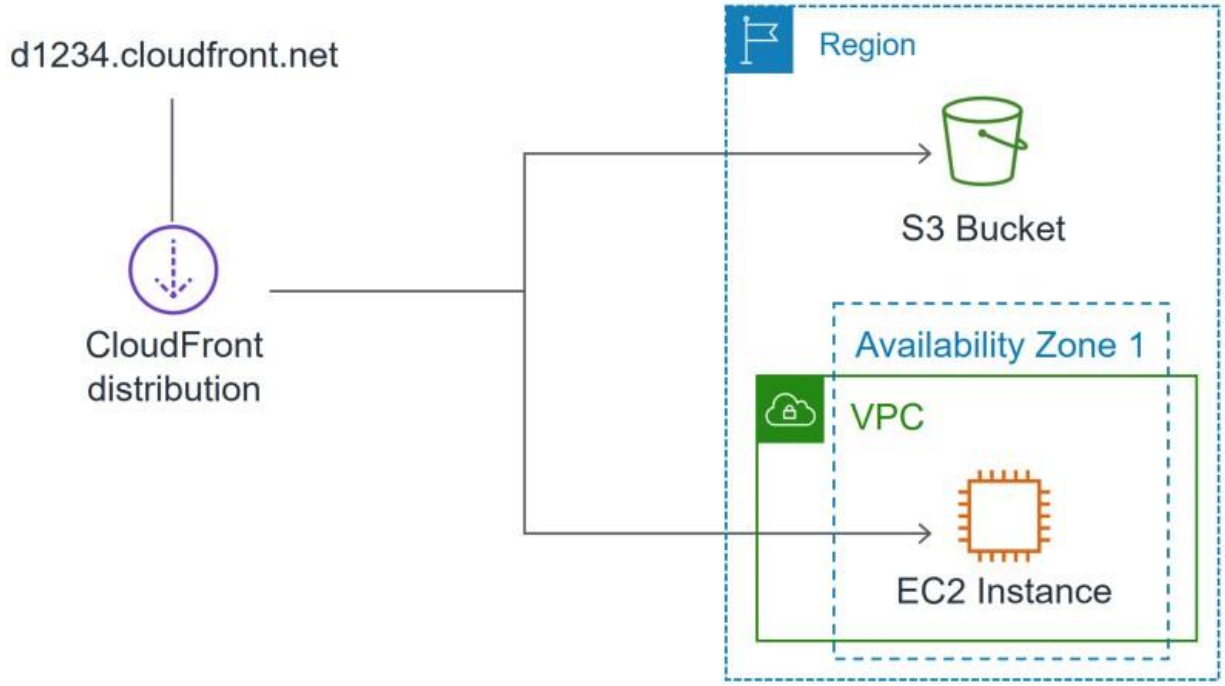
Accelerate Content using Edge Computing

In this experiment, you will learn how to set up a CloudFront Edge Computing distribution to front a simple web application with static and dynamic content hosted on Amazon S3 and on an Amazon EC2 instance respectively, as per the below diagram. You will learn how to test it, and check what are the special headers sent by CloudFront. Finally, you will invalidate the Edge Cached content and configure graceful Edge Computing Content failures using custom error pages.



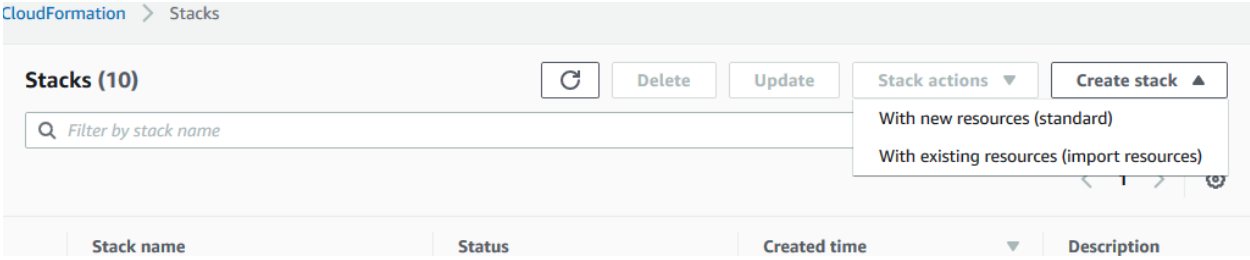
Create Edge Origins

In this section, you will create both S3 and EC2 Edge Cache Origins using a provided CloudFormation template.

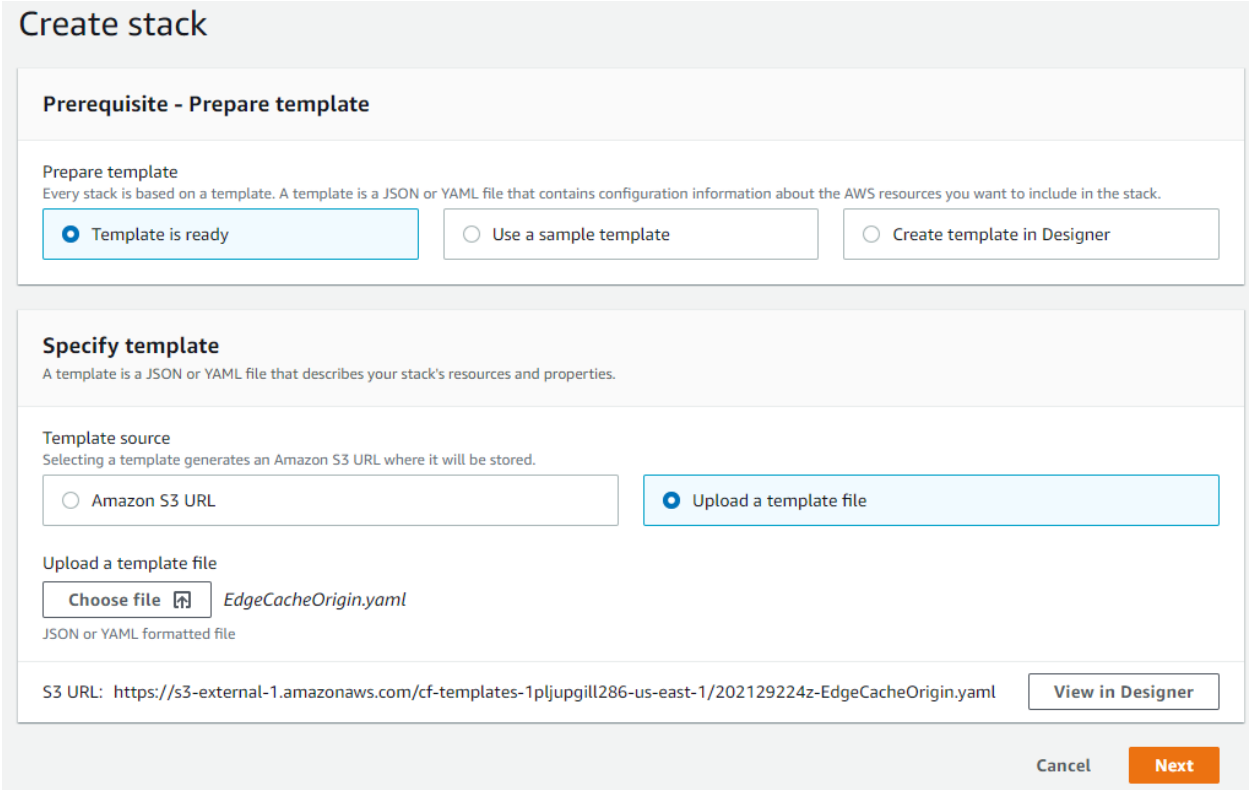
Go to CloudFormation console in North Virginia ***us-east-1***.

You can use this URL
https://console.aws.amazon.com/cloudformation/home?region=us-east-1#

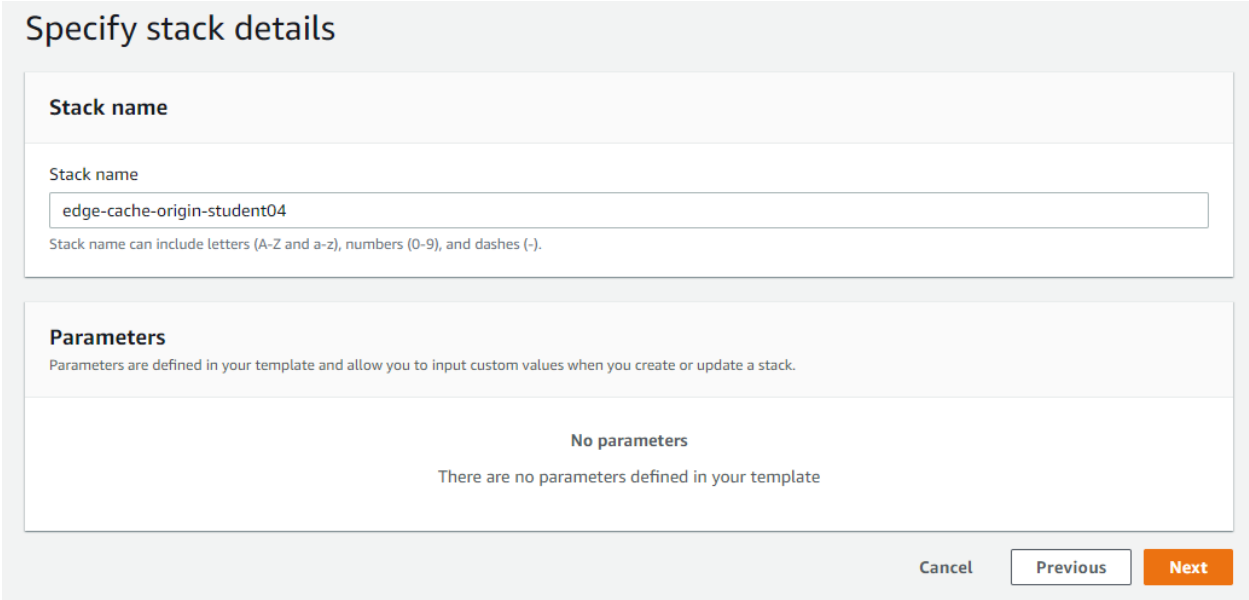
Or use the service search as we've shown in earlier experiments with **CloudFormation**



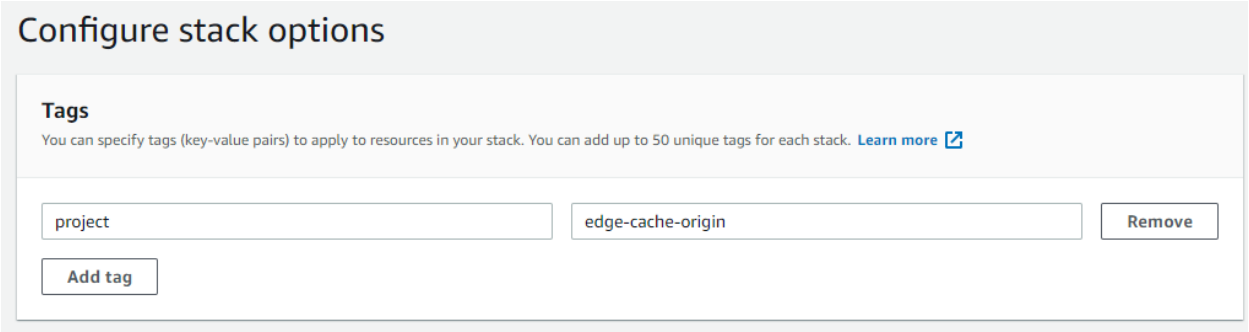
1. Select **Create stack with new resources** ('Create stack' > 'With new resources (standard)').



2. Select **Template is ready** and **Upload a template file** options, then choose and upload the following template file: **EdgeCacheOrigin.yaml**, which is available in the Course Github experiments folder
3. Select **Next**

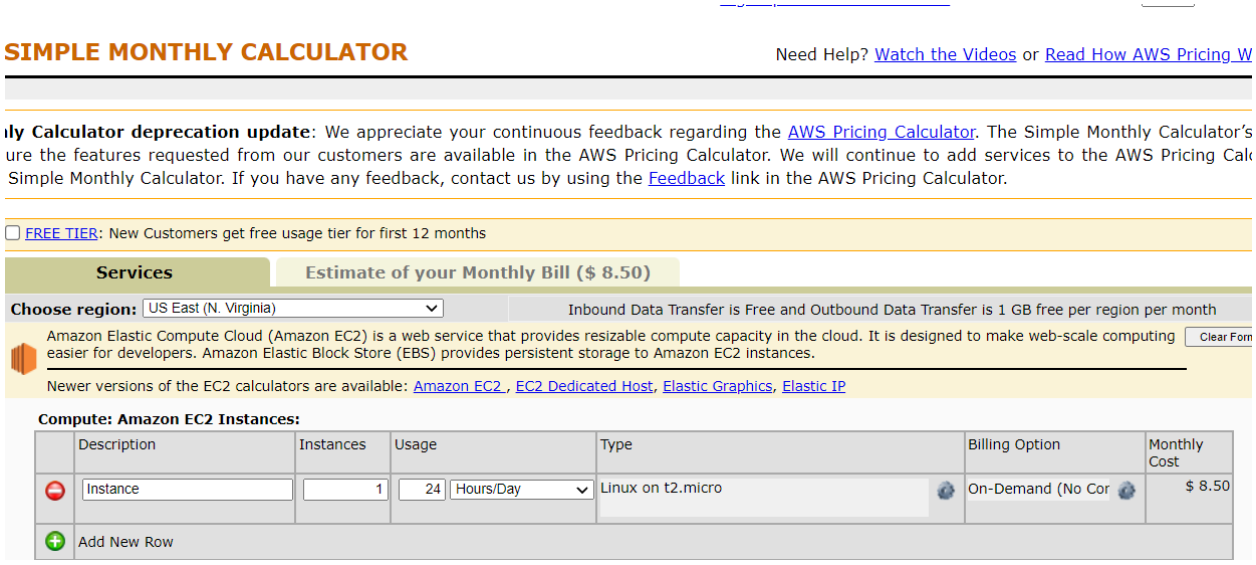


4. Enter a name for your stack as **edge-cache-origin-studentXX** and select **Next**

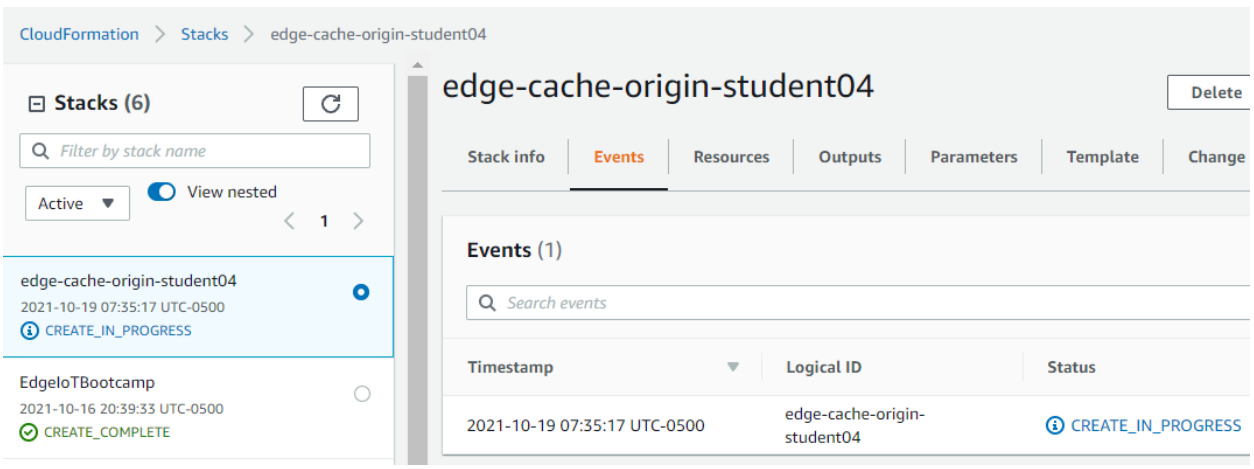


5. Enter **project** for our tag key name and **edge-cache-origin** for our tag key value, leave the rest of the configurations as default and select **Next**

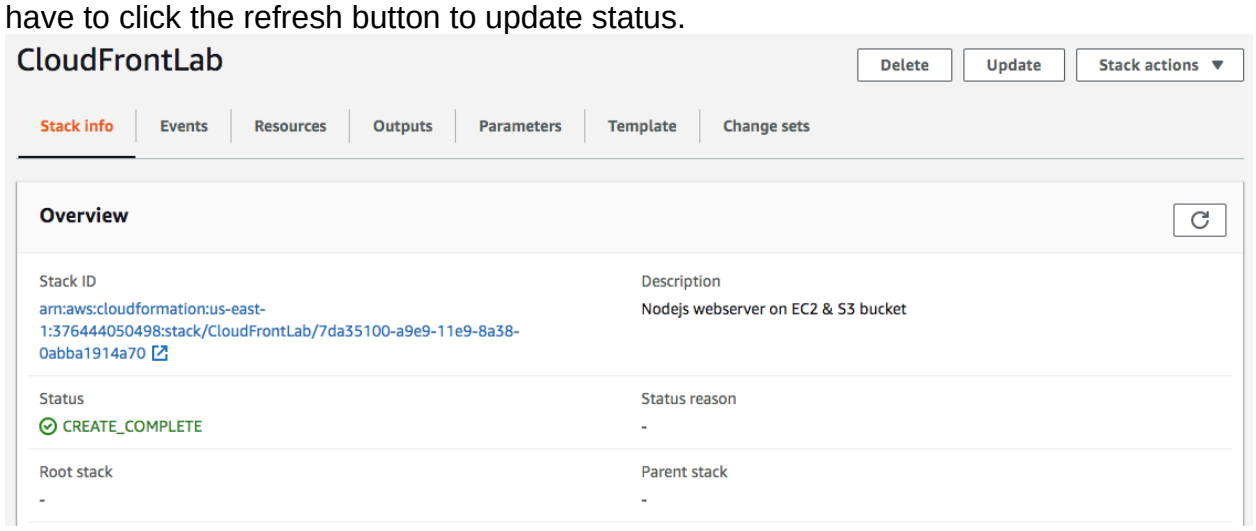
6. Click on the **Estimate cost** link to view the Pricing Calculator in a new browser tab



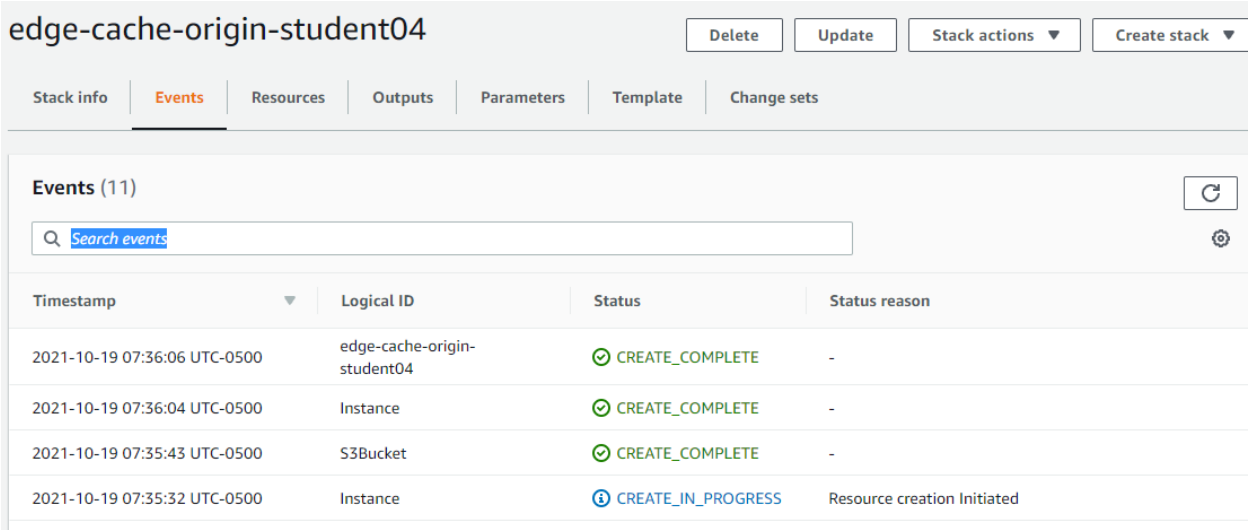
7. Return to the CloudFormation browser tab and select **Create stack**.



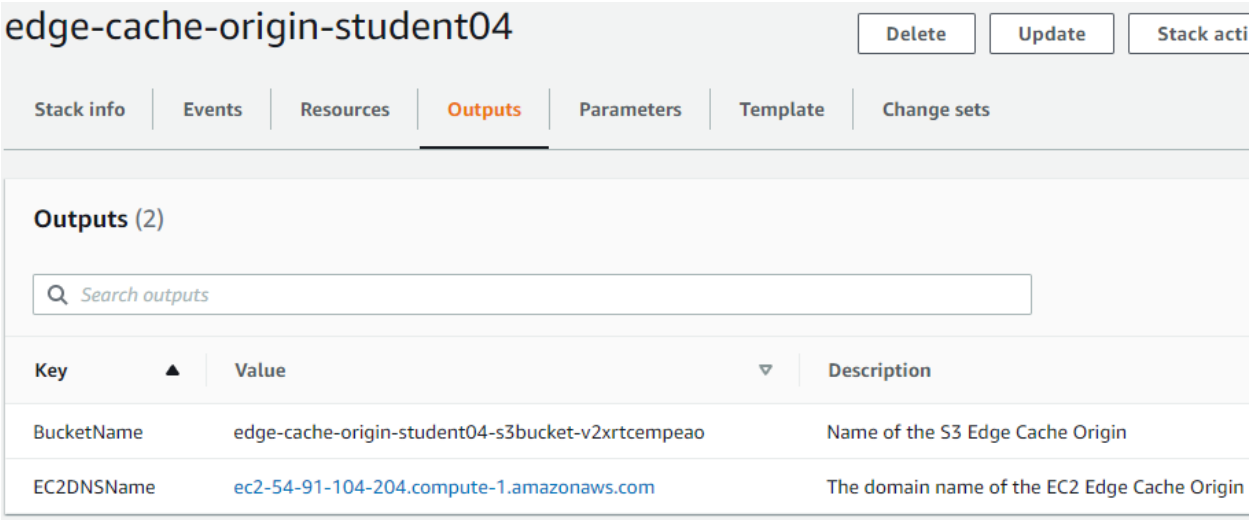
8. Wait for the stack status to show CREATE_COMPLETE, and remember you have to click the refresh button to update status.



9. Viewing the Events Tab we note details or our CloudFormation stack including the web server instance, S3 bucket for object storage (web content) and security group information. This information can be useful for debugging in case of any errors.



10. View the Outputs Tab and note the DNS name of your EC2 web server as well as the S3 bucket name. During the stack launch process, you can check the progress via Events Tab and get more details for debugging in case of any errors.



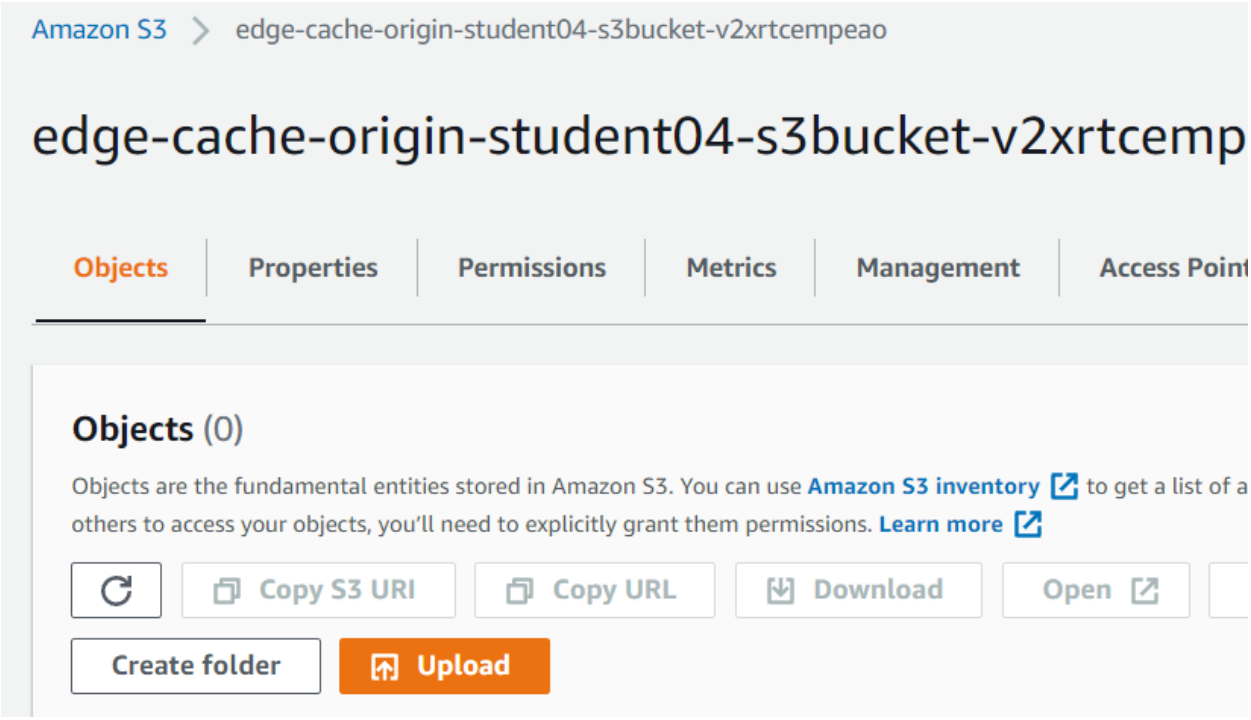
Create Edge Origin Content

1. Create an **index.html** file on your computer using a text editor of your choice and paste the following HTML content in it. This HTML calls the dynamic content using an iframe tag. In fact, when a user makes a request for index.html, the browser sends a subsequent request to /api. Alternatively the Edge Origin content **index.html** is also available in the experiments folder of the **course GitHub repository**.

```
<!DOCTYPE html>
<html lang="en">
  <body>
    <table border="1" width="100%">
    <thead>
        <tr><td><h1>Edge Computing Experiment</h1></td></tr>
    </thead>
    <tfoot>
        <tr><td>Edge Services - Origin Content</td></tr>
    </tfoot>
    <tbody>
        <tr><td>Response sent by Edge API</td></tr>
    </tbody>
    <tbody>
        <tr><td> <iframe src='/api' style="width:100%;
height:100%;"></iframe></td></tr>
    </tbody>
    </table>
  </body>
</html>
```

2. Note the S3 Edge Content Origin Bucket name in the Outputs tab for CloudFormation stack we created. It should be similar to **`edge-cache-origin-student04-s3bucket-v2xrtcempeao`**

3. Browse to the S3 console through the Service Search dialog or via the following deep link https://us-east-1.console.aws.amazon.com/s3



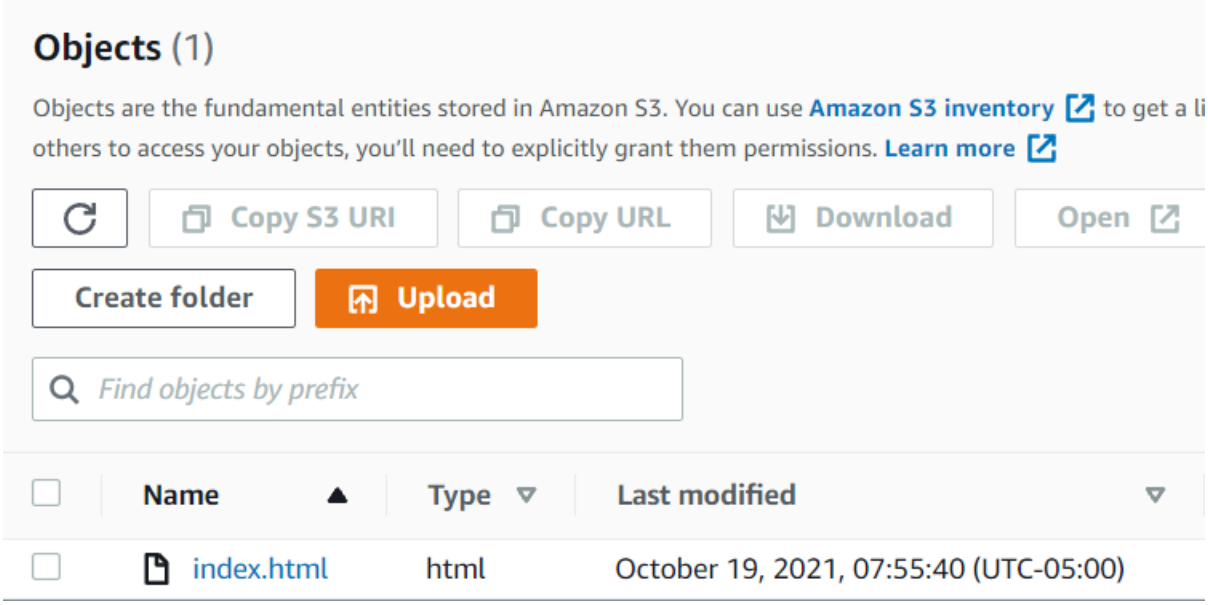
4. Select **Upload**

5. Select **Add files**

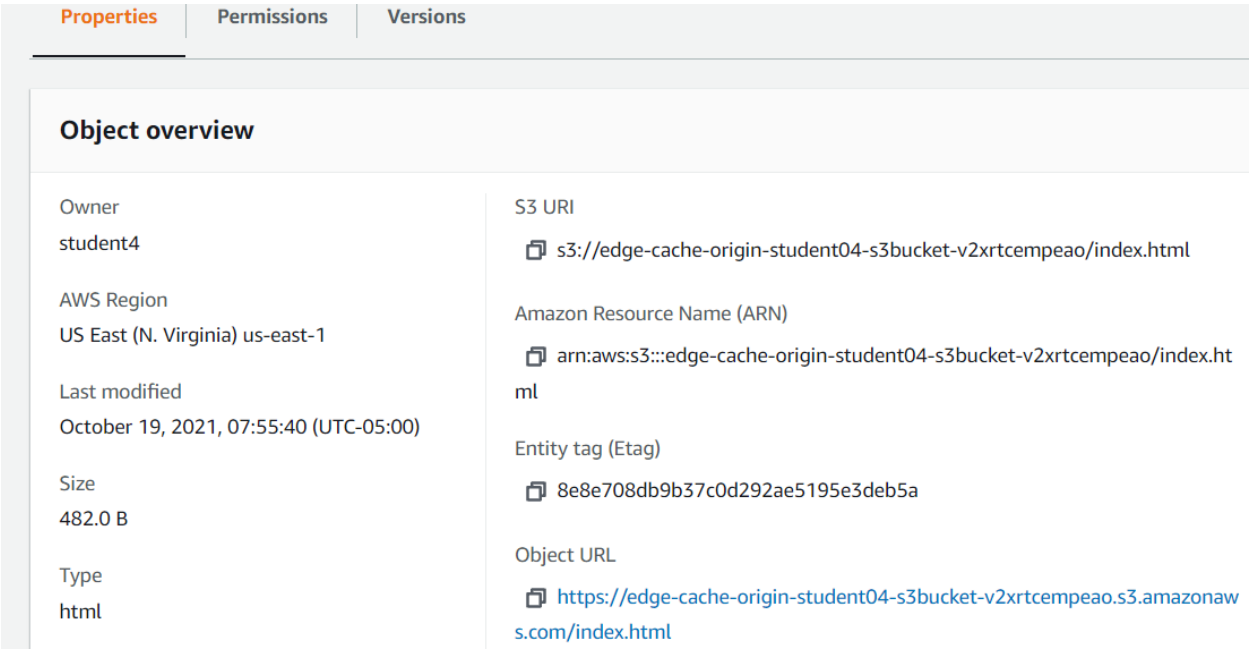
6. Browse to the **index.html** that you created or downloaded for our Edge Origin Content and select that file for upload.
7. Leave the remaining options for permissions, location, and properties as default and select **Upload**

Edge Object Storage Security

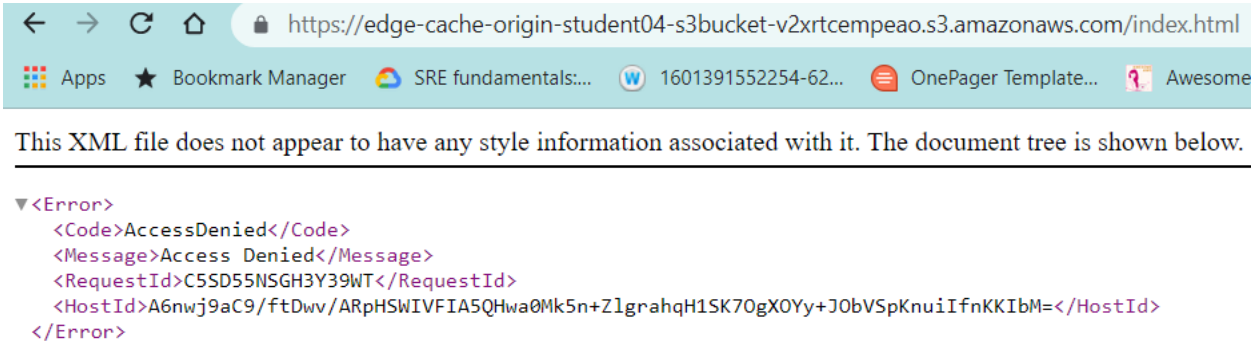
When you try to request index.html using the S3 provided Object URL, the access will be denied since it is not configured as public object.



1. Select the **index.html hyperlinked name** to view the file information



2. Select the **Object URL** from the Object overview in Properties tab to open the index.html that we created and uploaded in a web browser



3. Notice that we are presented with an error page not the index.html Edge Content that we created. We'll correct that security issue later in this experiment

Edge Computing Serverless Lambda

The CloudFormation template has deployed a Node.Js based application that listens to HTTP requests on port 80 of the EC2 instance. Upon receiving a request, the application will send back a JSON response that includes the headers received in the request. It will also inspect the query string info and return some data from the webserver based on the query string value. The application code is below for your reference:

```
const express = require('express')
const app = express()

app.get('/api', function (req, res) {
  console.log(JSON.stringify(req.headers))
  if (req.query.info) {
    require('child_process').exec('cat '+ req.query.info,
      function (err, data) {
        res.send(new Date().toISOString() + '\n' +
JSON.stringify(req.headers)+ '\n'+data)
      });
  } else {
    res.send(new Date().toISOString() + '\n' +
JSON.stringify(req.headers))
  }
});

app.listen(8080, function () {
  console.log('api is up!')
})
```

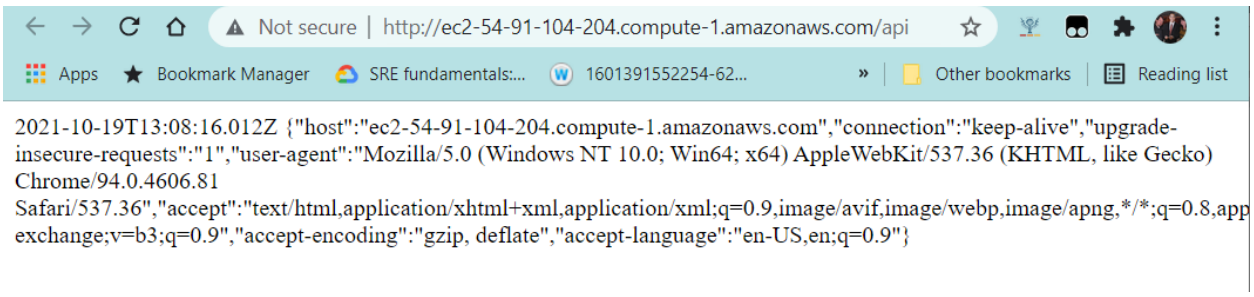
Make sure the application work by entering the following in your browser: [http://[EC2-DNS-name]/api](http://[EC2-DNS-name]/api). The EC2DNSName is listed in the outputs for your CloudFormation

stack (below the name of the S3 bucket we just uploaded our index.html into). It should look similar to `ec2-54-91-104-204.compute-1.amazonaws.com`

The URL should be similar to

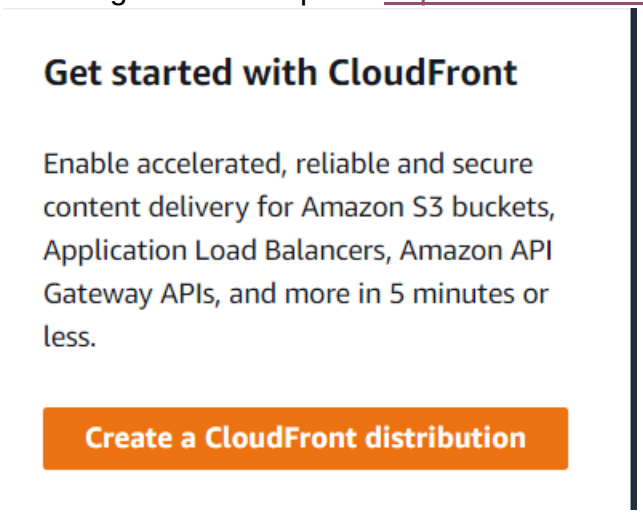
http://ec2-54-91-104-204.compute-1.amazonaws.com/api

You should see a response like the example below.

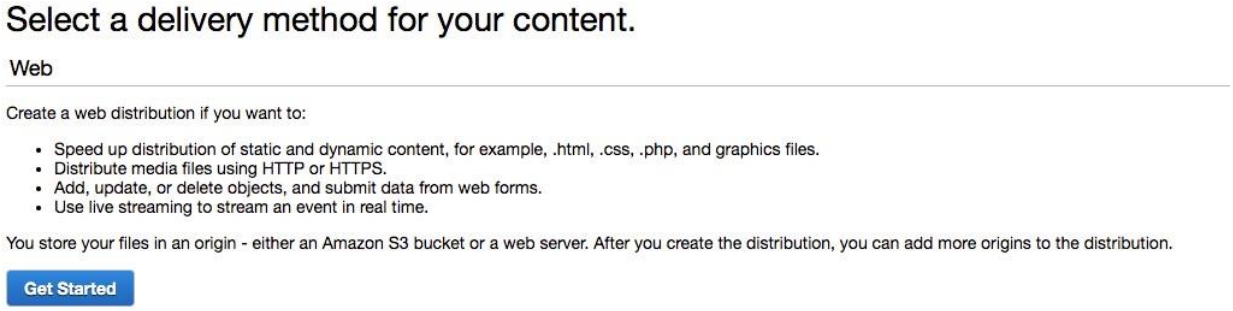


Create Edge Computing Distribution

1. Browse to the CloudFront console via the Service Search dialog or use the following service deep link https://us-east-1.console.aws.amazon.com/cloudfront

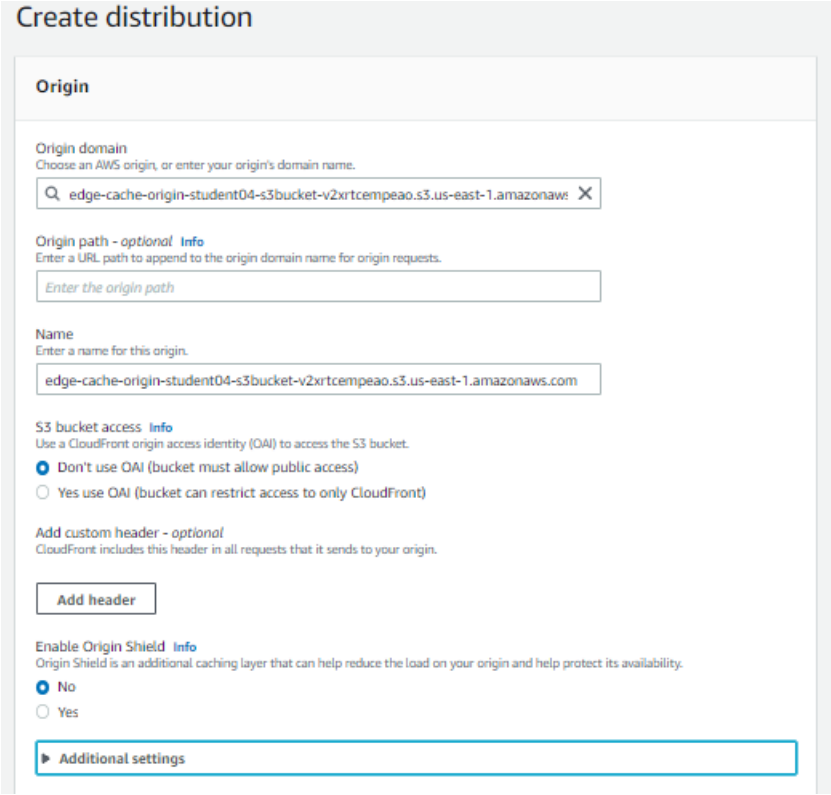


2. Select **Create a CloudFront distribution.**

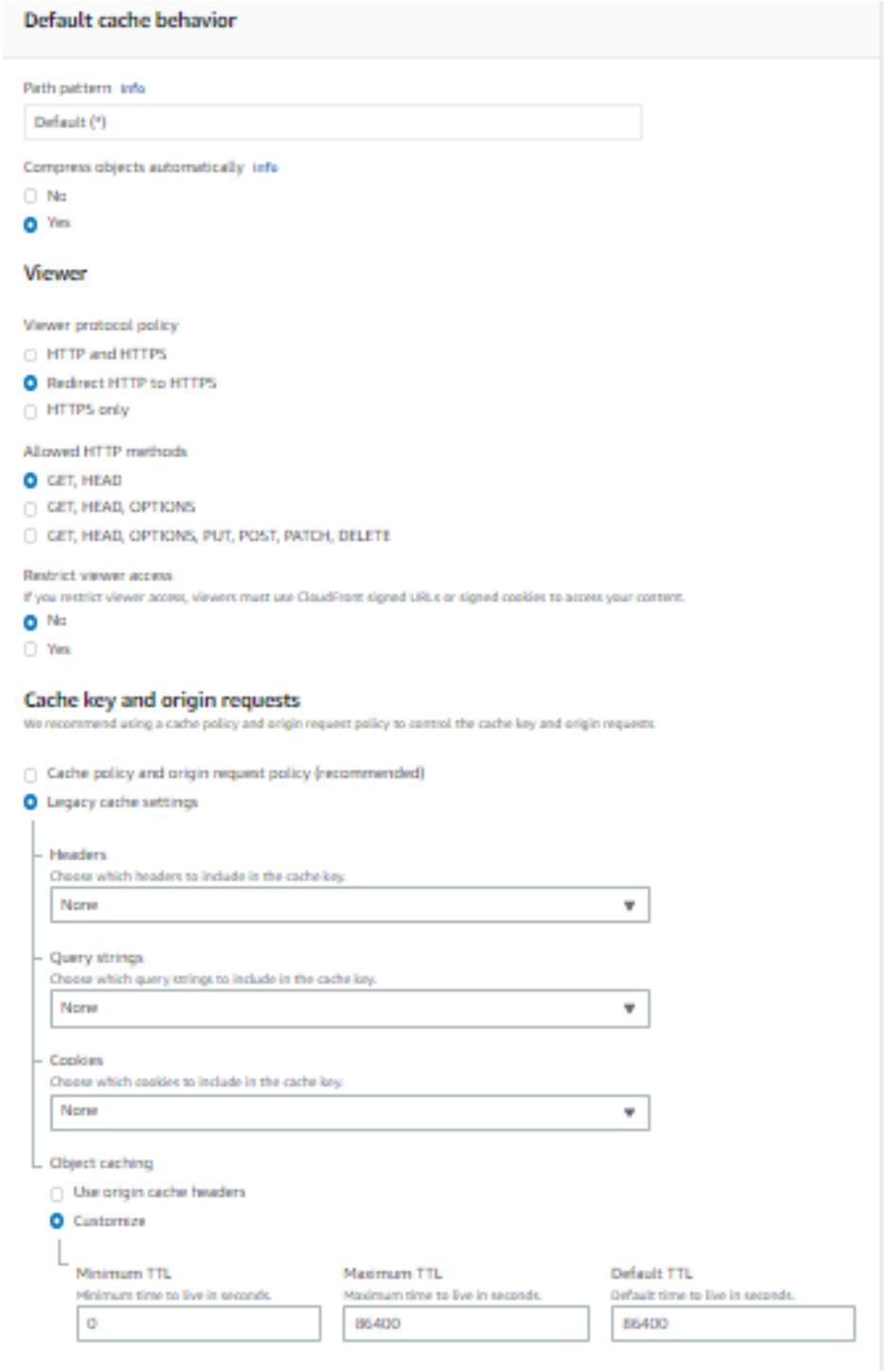


3. Configure the default origin to the previously created S3 bucket and grant CloudFront the permissions to the bucket using Origin Access Identity settings:

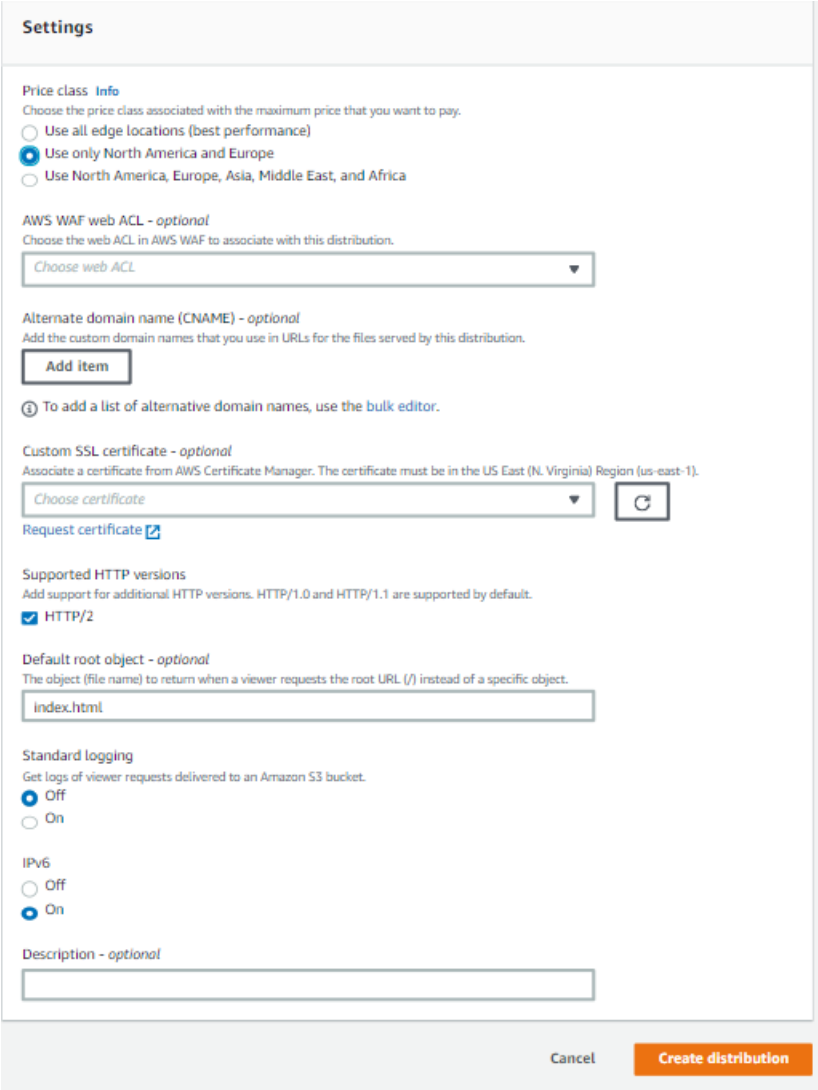
- Restrict Bucket Access: Yes
- Origin Access Identity: Create a new Identity
- Grant Permissions on Bucket: Yes, Update Bucket policy
- Origin Domain Name: Select the S3 bucket, similar to **`edge-cache-origin-student04-s3bucket-v2xrtcempeao`**, that we created in our CloudFormation by clicking in the input box and it should be automatically visible in the list provided. This will automatically populate the Name field, as well. Leave the remaining defaults in the Origin section for the new Edge Distribution



4. Configure the Default Cache Behavior as follows:



- Viewer Protocol Policy: Redirect HTTP to HTTPS
- Cache and origin request settings: Use legacy cache settings
- Object Caching: Customize
- Minimum TTL: 86400



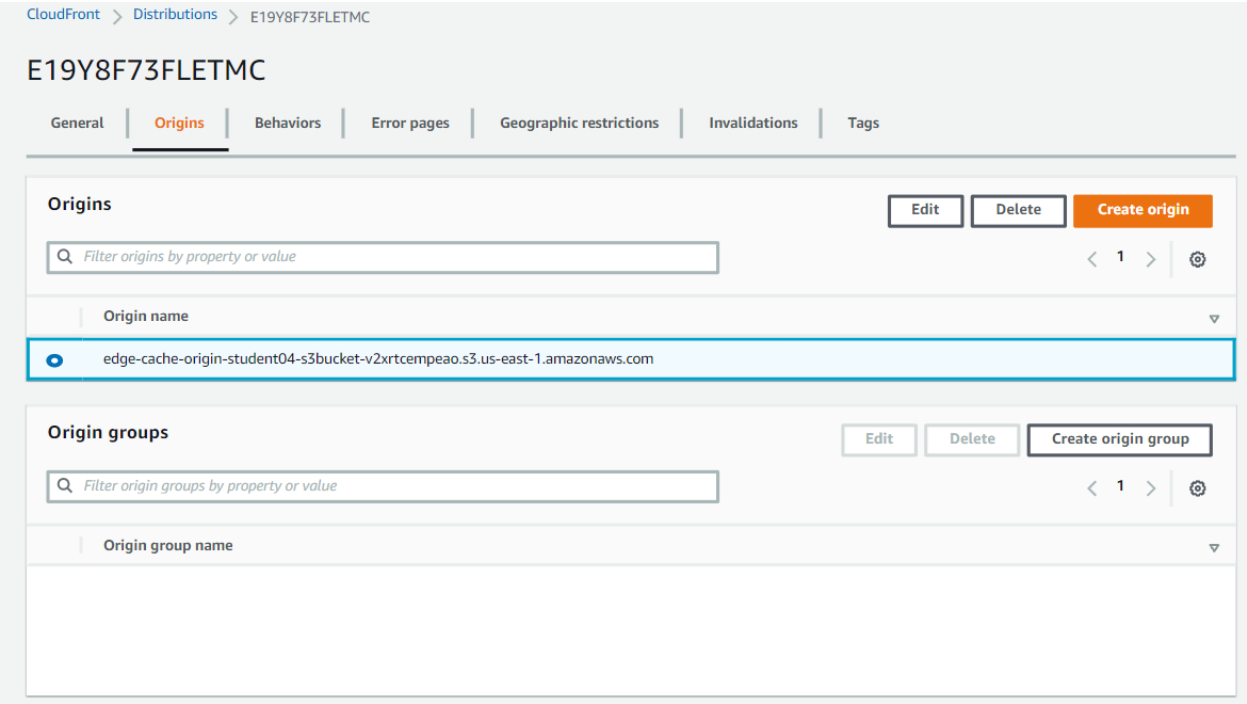
5. In the Distributions Settings section, configure **Default root object** to **index.html** and **Price class** to **Use only North America and Europe**, leave the rest to defaults.

In this lab, you will be using a domain name provided by CloudFront, however, if you want to use your own domain name, you can configure it with Alternate Domain Names (CNAMEs) section.

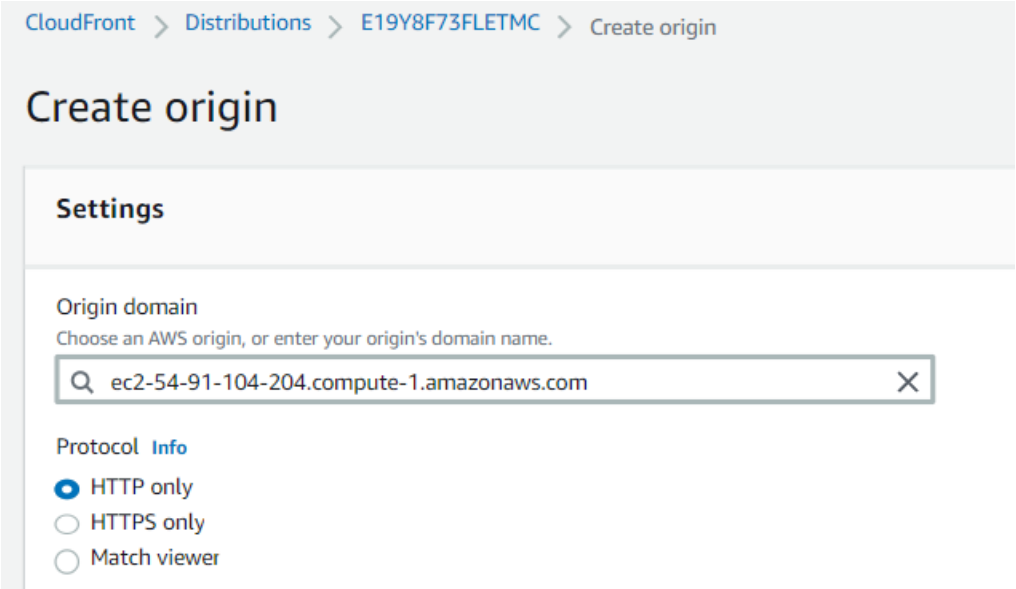
6. Select **Create distribution**. CloudFront will start creating the Edge Computing Cache distribution and normally it takes 5 to 10 minutes to fully propagate. The status of the distribution will be In Progress. To check the status, you can click on the Distribution menu on left pane.

Add Edge Cache API Origin

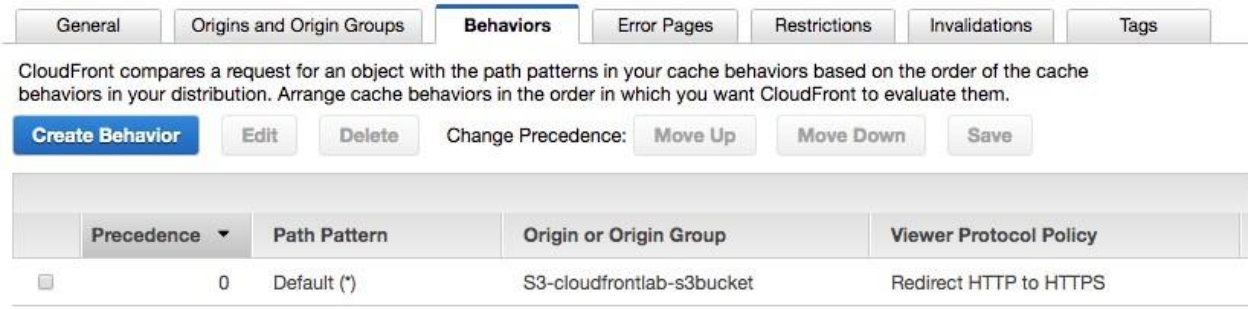
1. In the distributions console, click on your distribution ID, then select the Origins Tab to view the Origins and Origin Groups view.



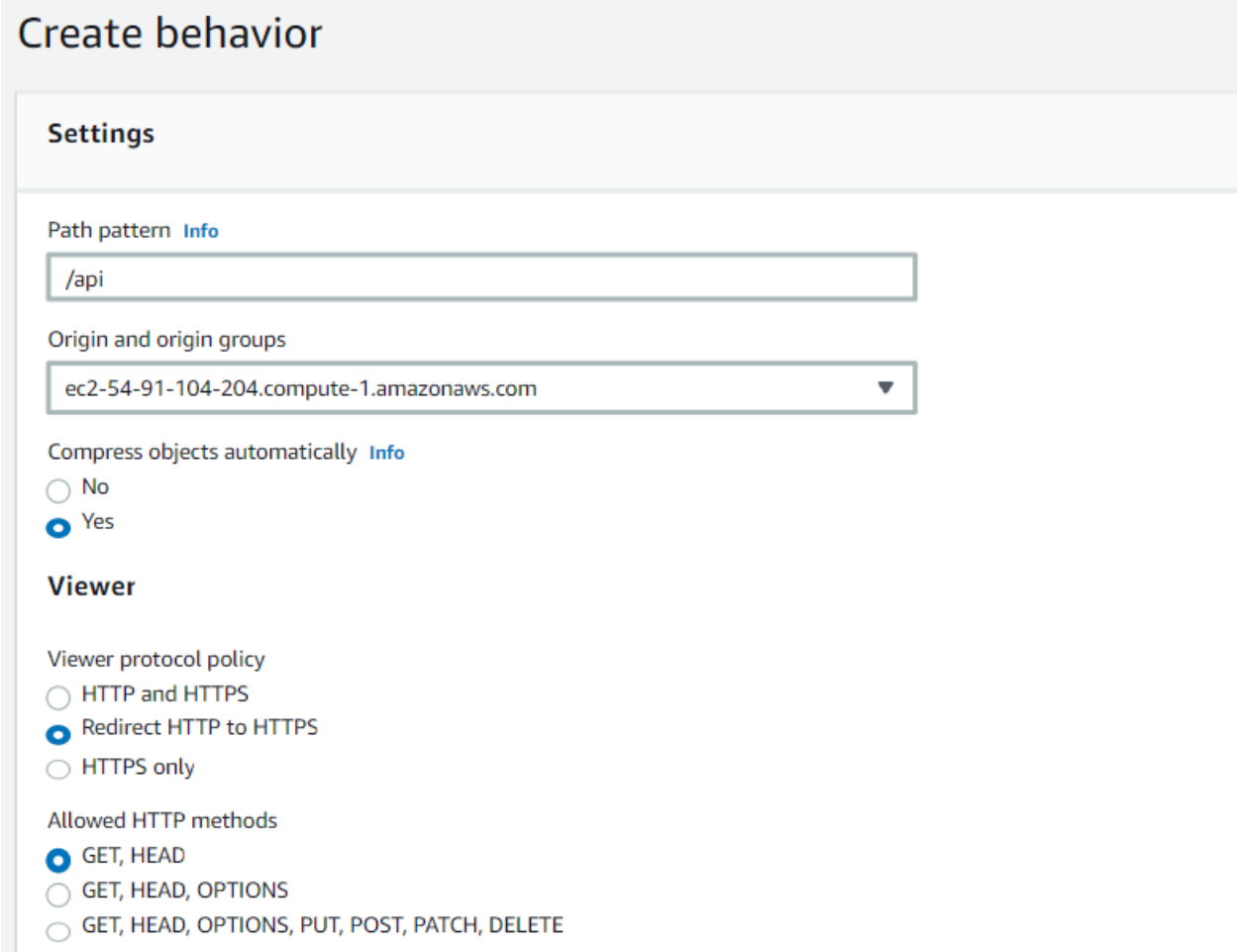
2. Enter the EC2DNSName from our CloudFormation Outputs that will be similar to ec2-54-91-104-204.compute-1.amazonaws.com. Increase the keep alive timeout to 60 seconds. Please note that although we want to serve content on HTTPS to users, we want to keep HTTP connection the origin to reduce the TLS overhead on the origin. This is configured by setting the Origin Protocol Policy to HTTP



3. Select **Create origin** button.
4.
5. Select the Behaviours Tab



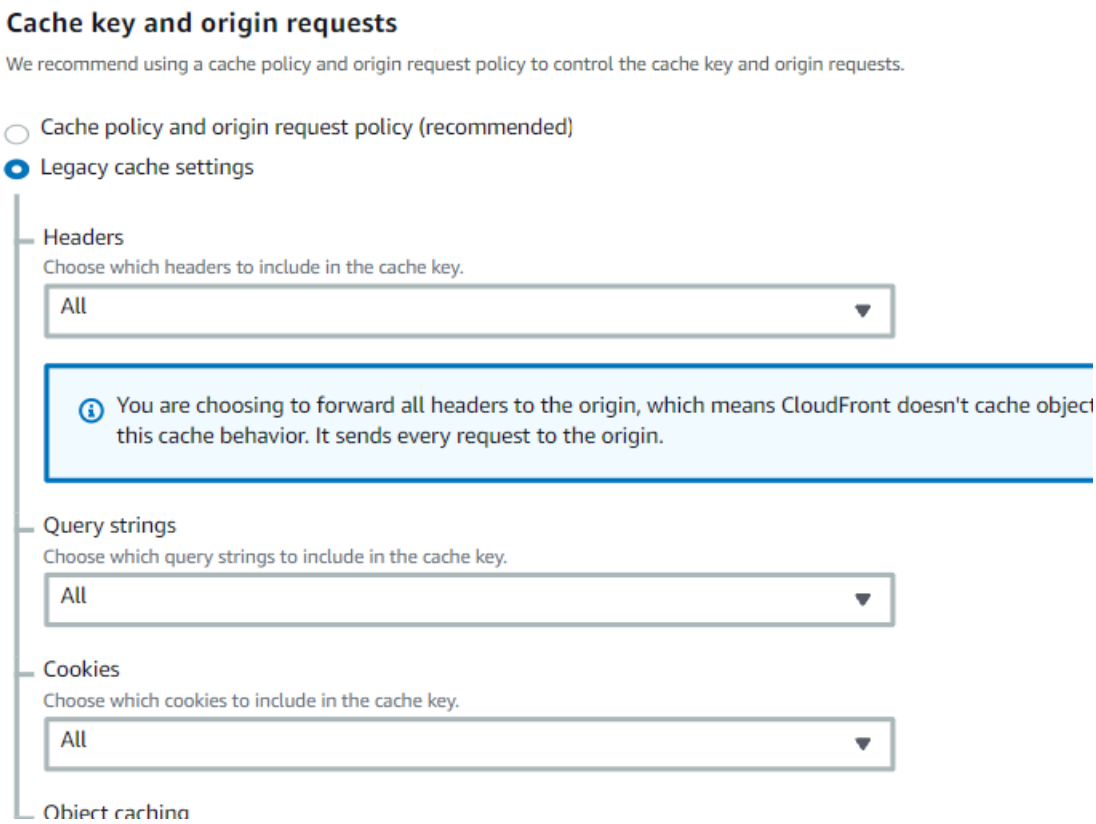
6. Select **Create behavior**



7. Update

    - Path pattern - /api
    - Compress objects – Yes
    - Viewer – Redirect HTTP to HTTPS

8. Update the Cache Key and Origin Request Settings as noted below selecting **Legacy cache settings** option

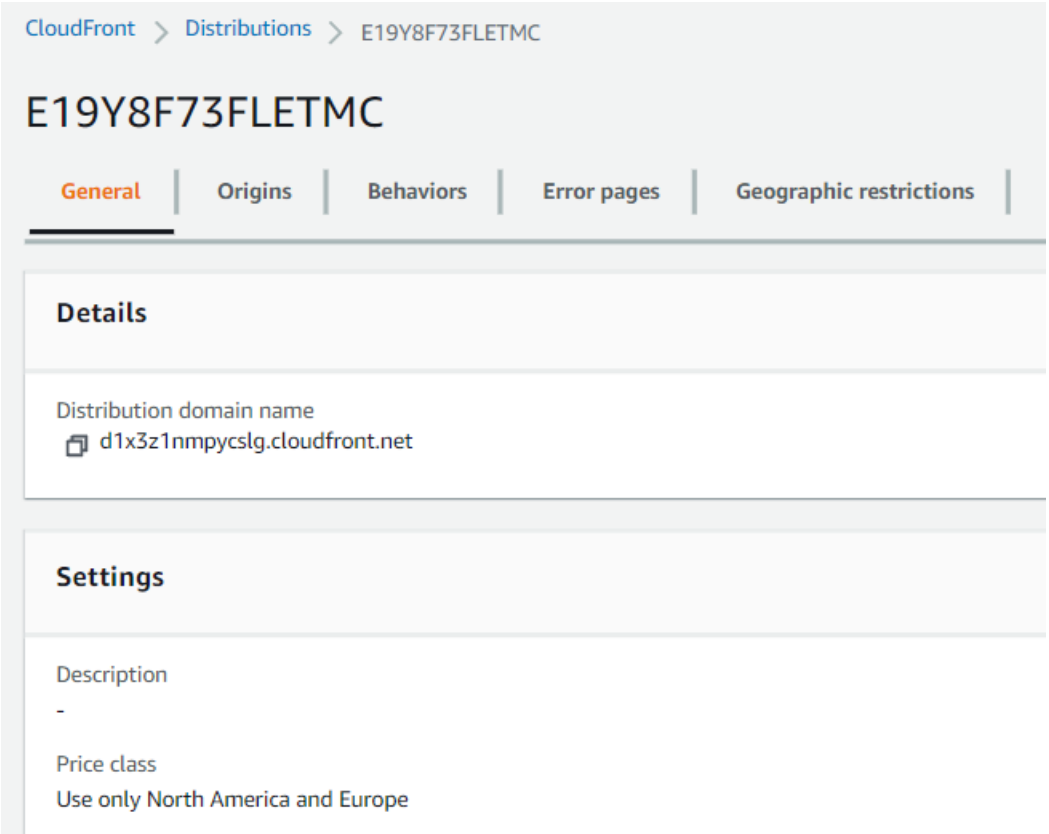


- Headers – All
- Query Strings – All
- Cookies – All

9. Select **Create behavior**

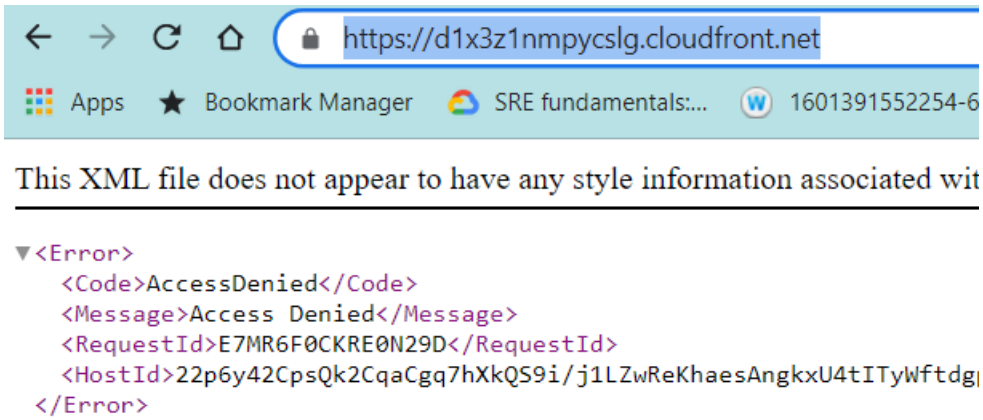
Test The Edge Application On CloudFront

1. Note that when we uploaded our index.html we never made it public so we received an error on viewing. We'll verify that is still occurring. View our CloudFront Edge Cache Distribution

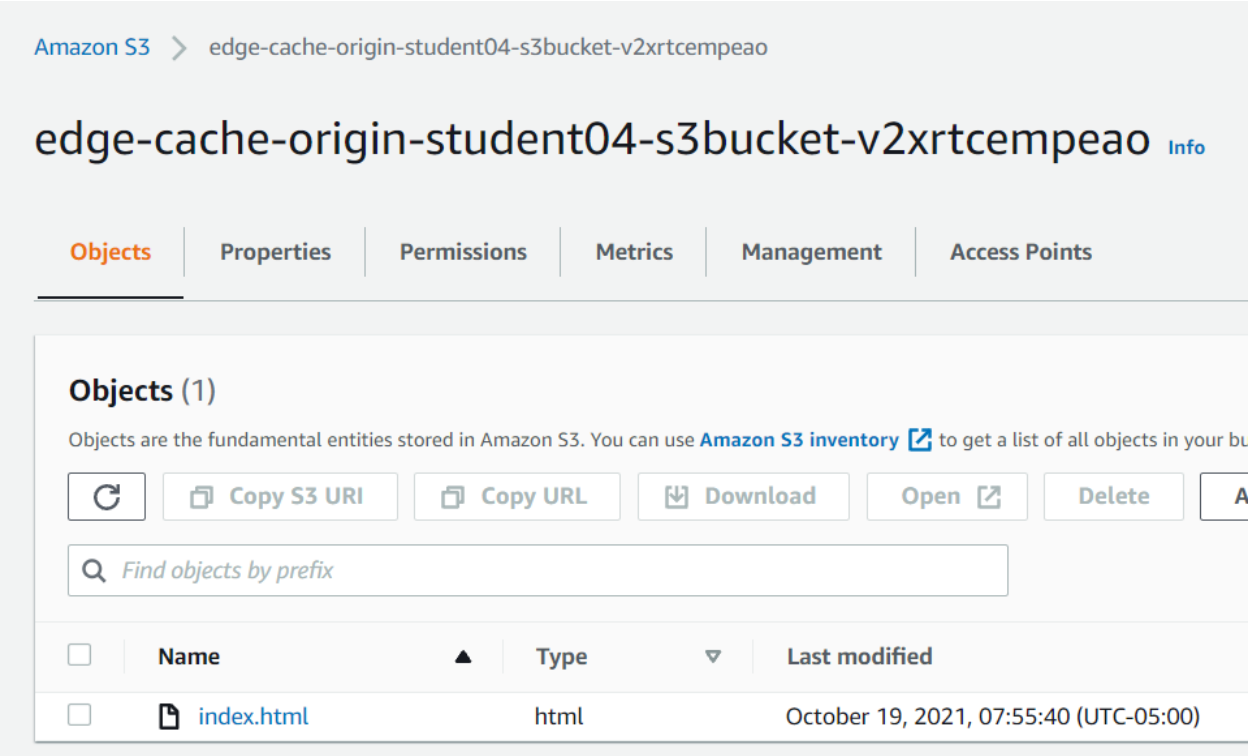


2. Check the **Distribution domain name** that CloudFront has associated to your distribution in the General tab. It should be similar to https://d1x3z1nmpycslg.cloudfront.net/ as seen in the view.

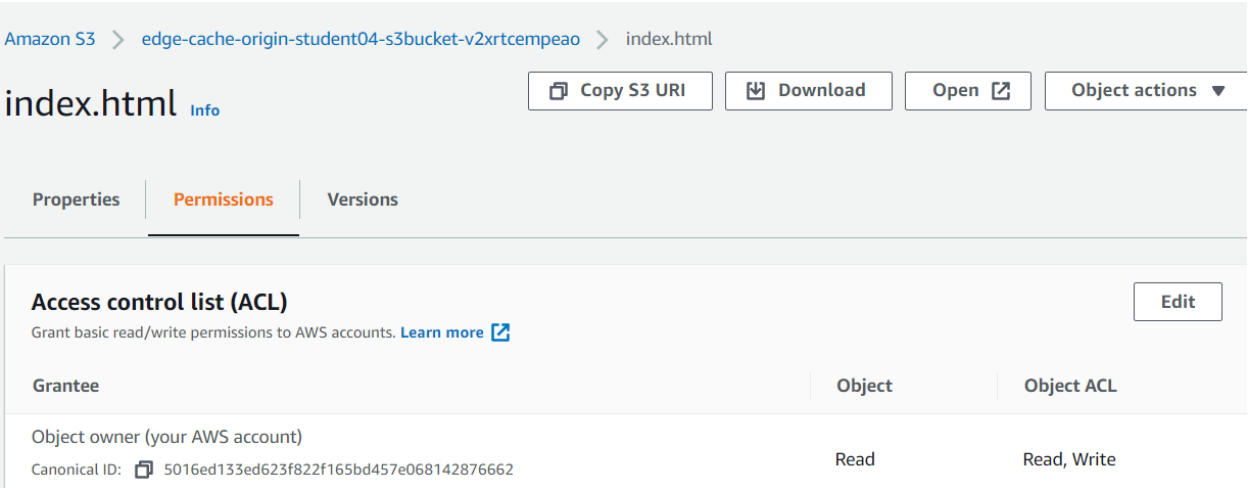
3. View the CloudFront distribution we've created in a browser. CloudFront distributions can be ready to be used locally even if the status is still in progress, since the status will change to deployed when the propagation has reached all 220+ edge locations.



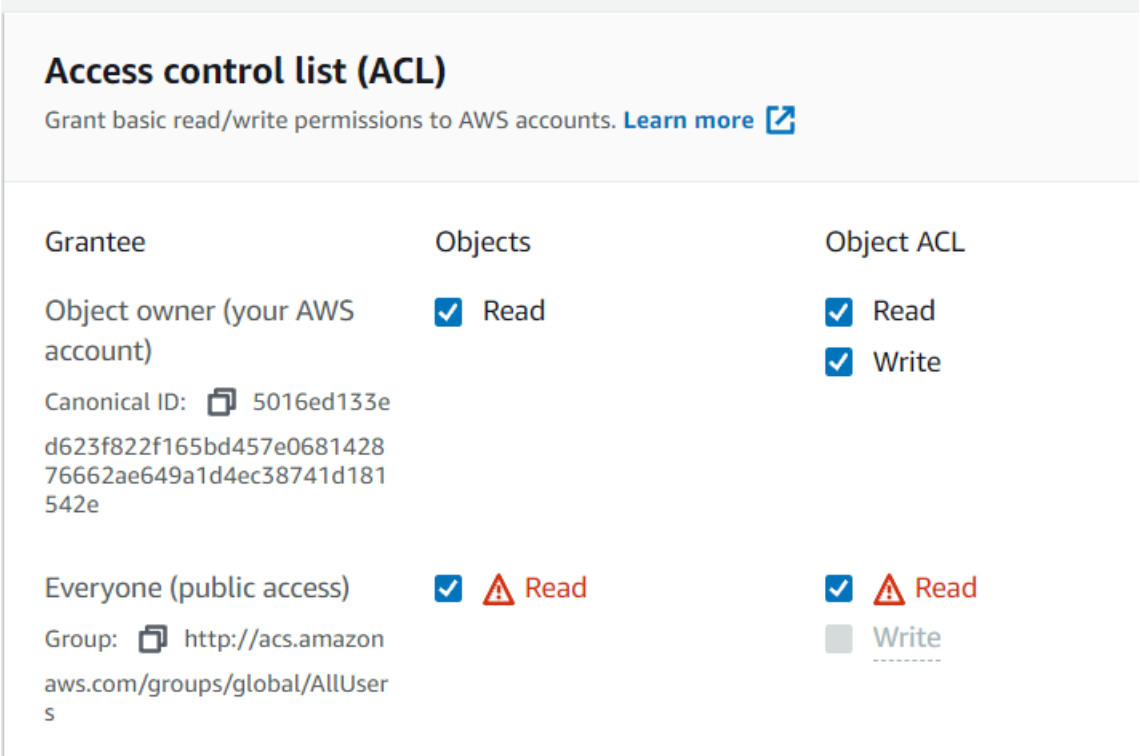
4. Browse to the S3 Console via the Service Search dialog or the following deep link. https://console.aws.amazon.com/s3



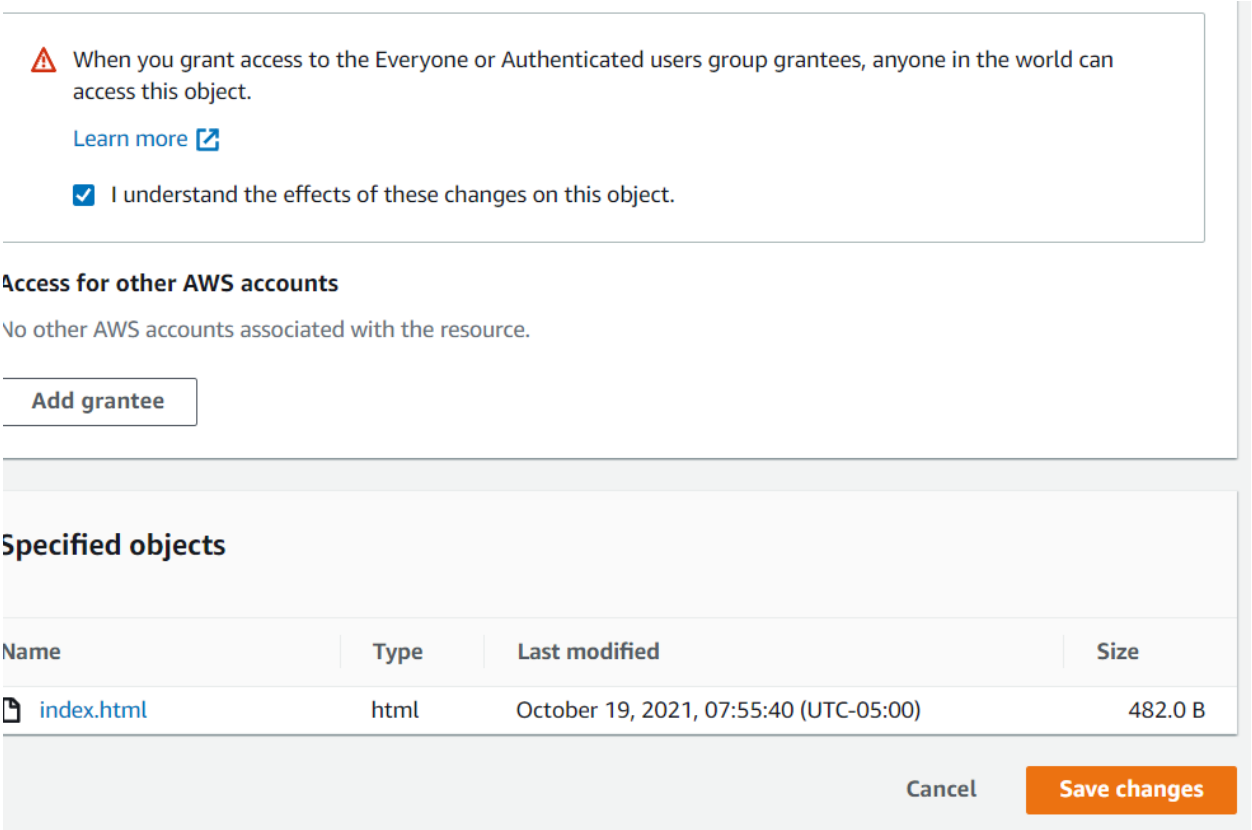
5. Click on the index.html hyperlink to view the properties of the file we've uploaded.



6. Select the Permissions tab and select **Edit**

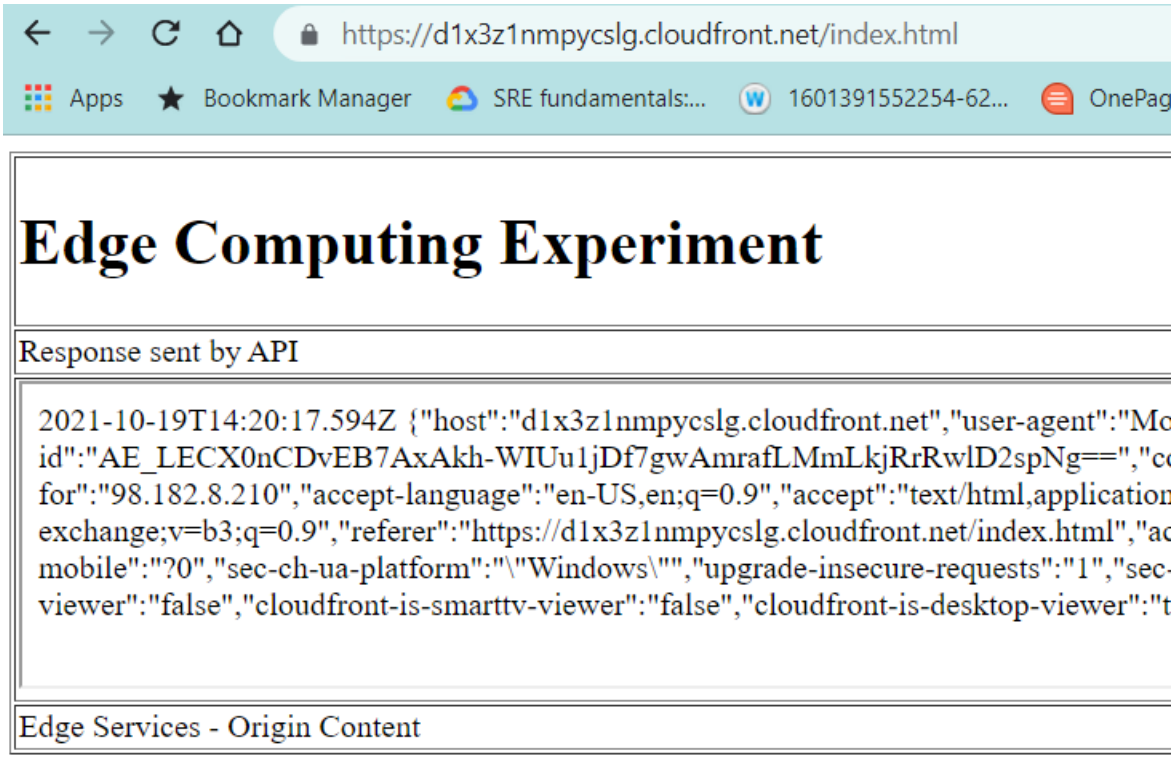


7. Check the box for **Read** on the Objects and Object ACL



8. Check the box for **I understand the effects of these changes …** and select **Save changes**

9. Check the **Distribution domain name** that CloudFront has associated to your distribution in the General tab. It should be similar to https://d1x3z1nmpycslg.cloudfront.net/ as seen in the view.



Test the Edge DNS Resolution

1. To test if the distribution is ready to be used locally, you can lookup its CloudFront domain name in command line by nslookup command. Keep in mind that dxxxx.cloudfront.net name is unique for every distribution, this is why need to use your own distribution value for testing. Note how CloudFront returns multiple IPs for each DNS query to increase application resiliency.

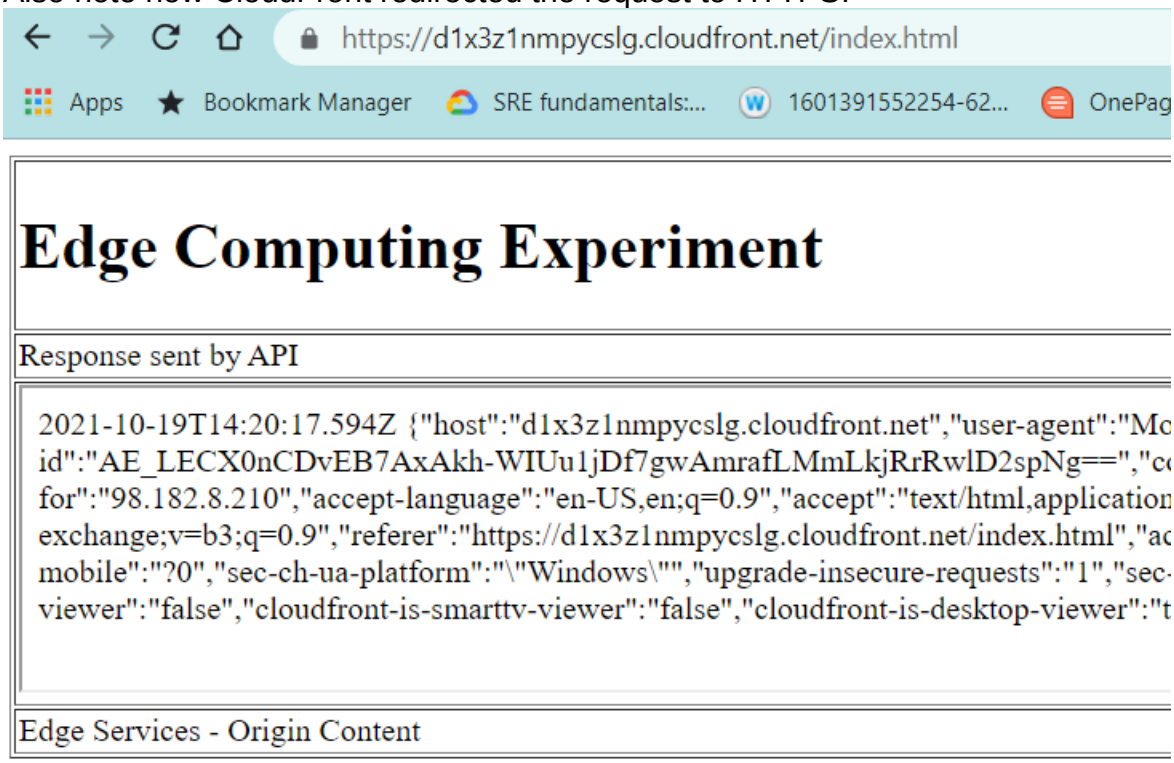
```
~ nslookup d1wj00cdxoisvm.cloudfront.net
Server:         10.4.4.10
Address:        10.4.4.10#53

Non-authoritative answer:
Name:   d1wj00cdxoisvm.cloudfront.net
Address: 52.84.225.127
Name:   d1wj00cdxoisvm.cloudfront.net
Address: 52.84.225.132
Name:   d1wj00cdxoisvm.cloudfront.net
Address: 52.84.225.137
Name:   d1wj00cdxoisvm.cloudfront.net
Address: 52.84.225.183
```

2. When the propagation is complete, you can test the webpage on your browser as served by CloudFront using http://dxxxx.cloudfront.net .
   In the webpage, you can see the different headers that CloudFront has forwarded and appended to your API endpoint:

- **cloudfront-forwarded-proto**: Indicates the protocol used by the viewer to connect to CloudFront
- **cloudfront-is-mobile-viewer**: Indicates the viewer's device type
- **cloudfront-viewer-country**: Indicates the viewer's country
- **x-amz-cf-id**: a unique id for this request provided by CloudFront. IF you refresh the webpage, you will see that how the request id is changing. It's useful to log it on your webserver in general. Additionally, this id will be sent back to every viewer request and sent to CloudFront access logs. If you need to debug any issue you can open a support ticket and provide them with the req id.

Also note how CloudFront redirected the request to HTTPS.



3. If you use the developer tools of your favorite web browser, you can check the response headers sent by CloudFront. Three headers are interesting to check:

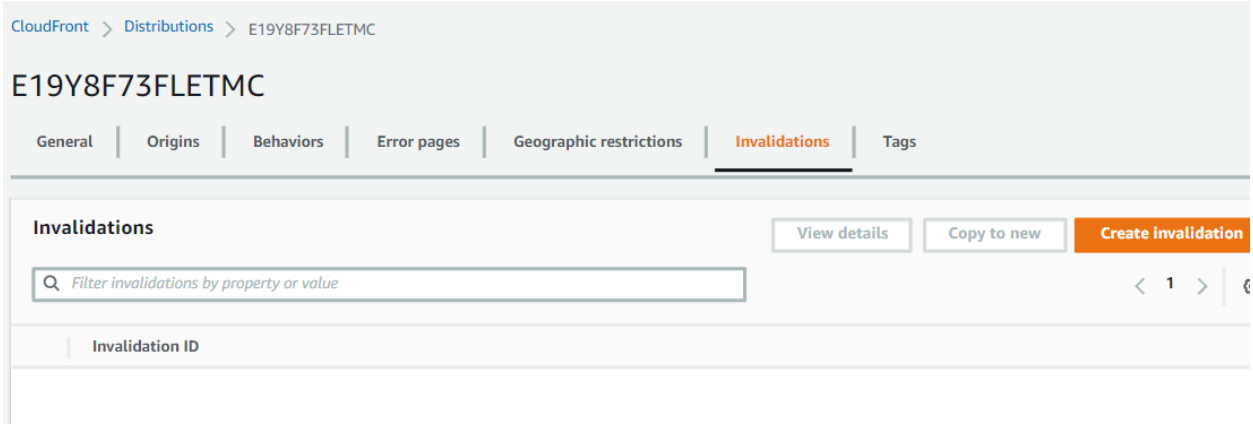
- **x-amz-cf-id** which holds the request id assigned by CloudFront.
- **x-amz-cf-pop** which indicates the CloudFront edge location that served your request. Each edge location is identified by a three-letter code and an arbitrarily assigned number, for example, DFW3. The three-letter code typically corresponds with the International Air Transport Association airport code for an airport near the edge location.

- **x-cache** which indicates whether the request was a cache hit or a cache miss. Normally, for your html file, you will get a 'Hit from Cloudfront' value in the subsequent requests, but always 'Miss from CloudFront' for /api request since caching is disabled for this behavior.

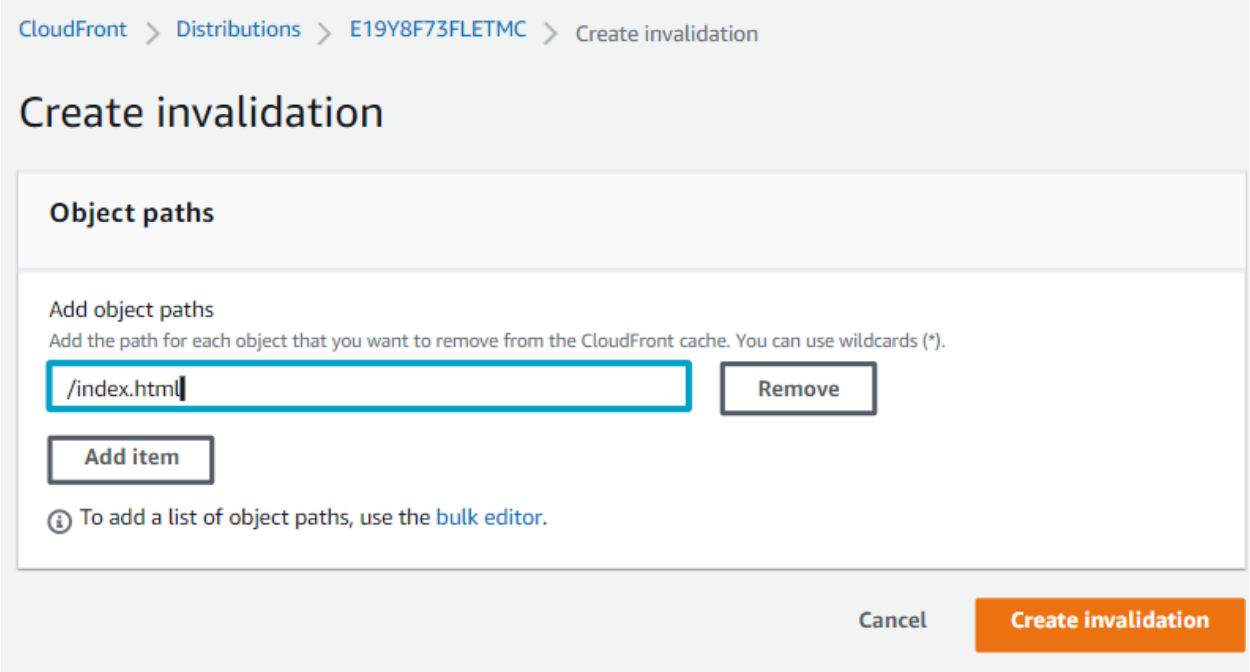
Test Edge Cache Invalidations

As you saw previously, the main index.html page is in cache and resulting in a Hit from CloudFront. Suppose that you have to change the HTML file but you can't change the URL to point to the new version, in this case, you need to invalidate the page.

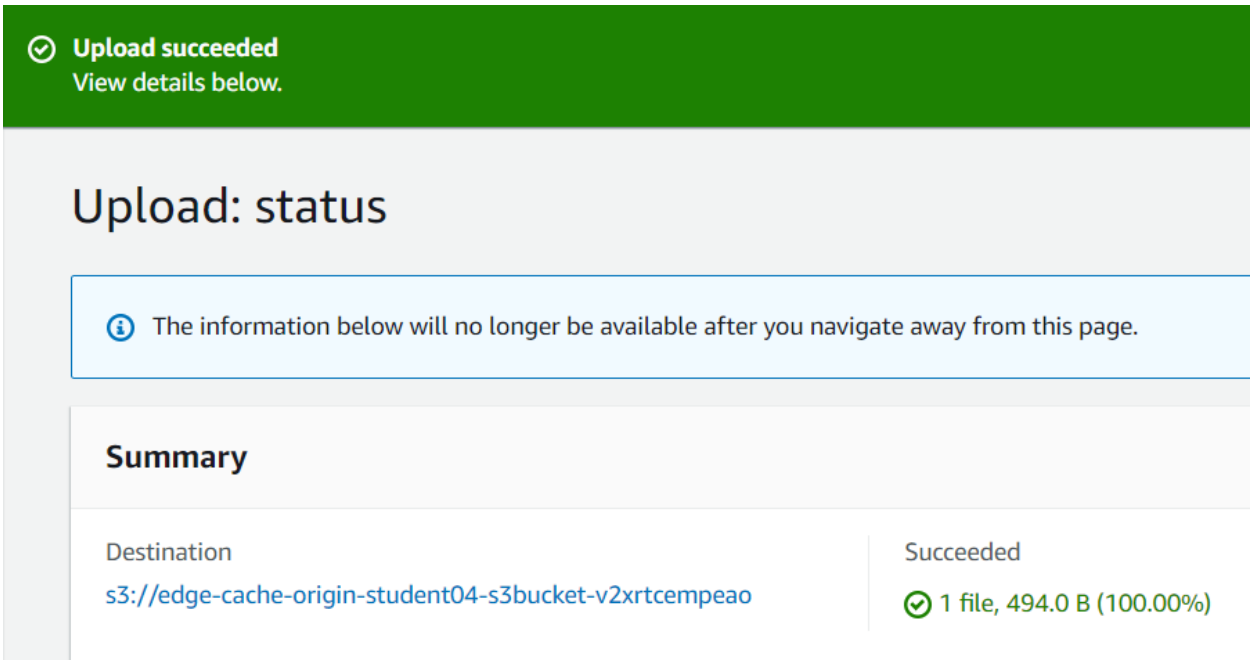
1. Go to the Invalidations Tab on the CloudFront console.



2. Select **Create invalidation**. Create an invalidation for your index.html. You can specify / in the Object paths because we already set index.html as default root object.

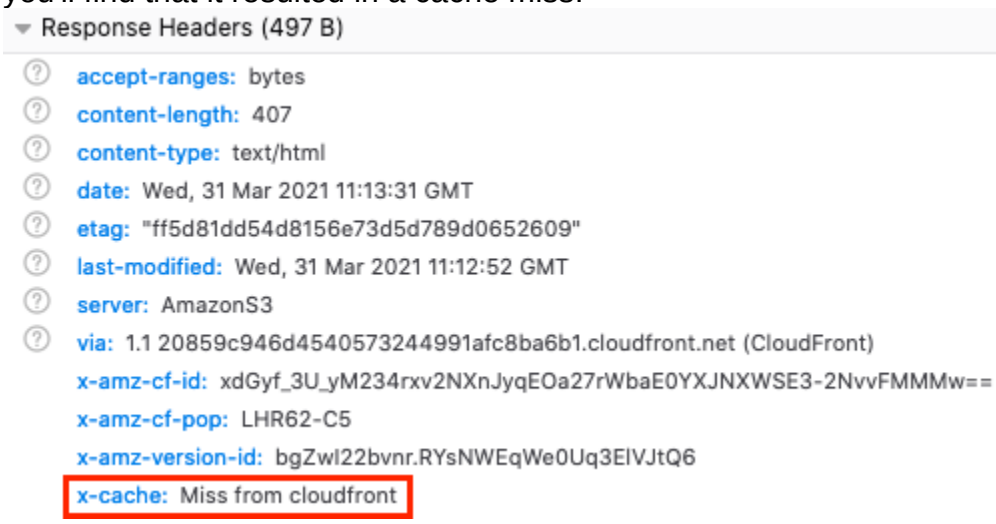


3. Update our invalidation for the root path with the forward slash (/index.html) as noted and select **Create Invalidation**.



4. Update the **index.html** that we created or downloaded in a text editor and upload that again to the S3 bucket that we created for the experiment. Since we're uploading a new object (even with the same name) we'll have to update the Permissions to make it readable again.

5. After a few of seconds, test again the page using the browser developer tool, and you'll find that it resulted in a cache miss.



6. Noted that when we reload the browser view, we should see our modified page

Configure Custom Error Page

1. Test a random URL using your CloudFront domain name and you will get a 403 Forbidden response from S3 behind CloudFront because the file does not exist.

By default, CloudFront caches this response for 5 minutes.

https://d1wj00cdxoisvm.cloudfront.net/random

This XML file does not appear to have any style information associated with it. The document tree is shown below.


```
▼<Error>
  <Code>AccessDenied</Code>
  <Message>Access Denied</Message>
  <RequestId>BE8E155EBE78989B</RequestId>
 ▼<HostId>
   Oym/PnbkouqSzSCypW84yj8dHYM/8n65OTOZLj/YBOhl2wrNE3iDic/JE3behPj7LR6JNGFuzv0=
  </HostId>
</Error>
```


2. Create an **error.html** file on your computer using a text editor with the below HTML content, and upload it to your S3 bucket like you did earlier for index.html.

```
<html lang="en">
  <body>
    <h1>Edge Cache Error</h1>
    Oops, this is a nice error page!
  </body>
</html>
```

Name	Last modified	Size
error.html	Jul 19, 2019 2:54:07 PM GMT+0800	109.0 B
index.html	Jul 19, 2019 1:58:10 PM GMT+0800	463.0 B

3. When uploading update the permissions on the upload or update the security after you've uploaded to make the error.html public

Permissions

Grant public access and access to other AWS accounts.

Access control list (ACL)

Grant basic read/write permissions to other AWS accounts. Learn more

AWS recommends using S3 bucket policies or IAM policies for access control. Learn more

Access control list (ACL)

- (●) Choose from predefined ACLs
- ( ) Specify individual ACL permissions

Predefined ACLs

- ( ) Private (recommended)
  Only the object owner will have read and write access.
- (●) Grant public-read access
  Anyone in the world will be able to access the specified objects. The object owner will have read and write access. Learn r

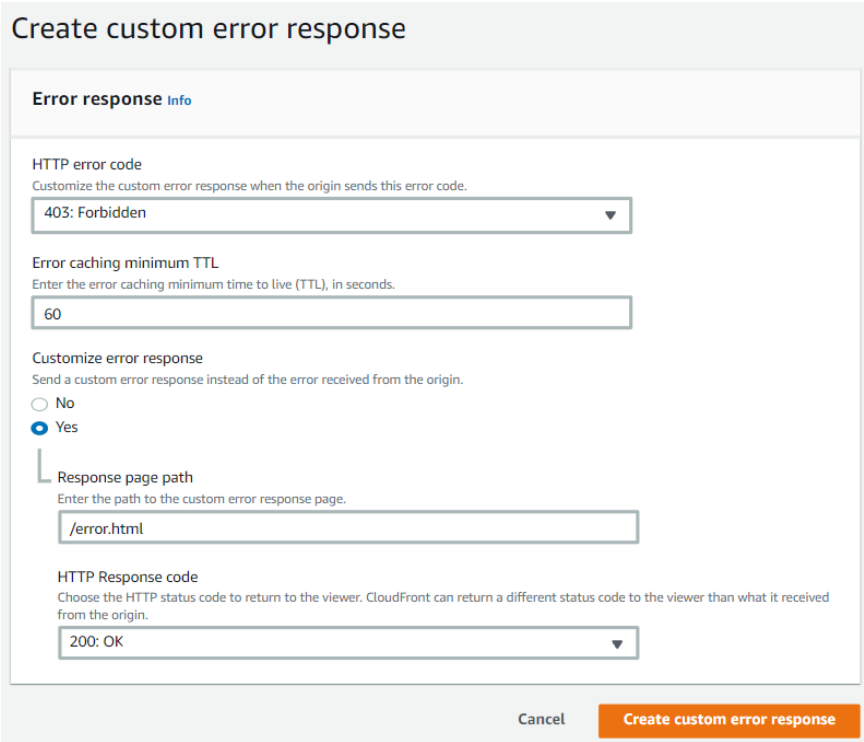
Granting public-read access is not recommended
Anyone in the world will be able to access the specified objects. Learn more

[x] I understand the risk of granting public-read access to the specified objects.

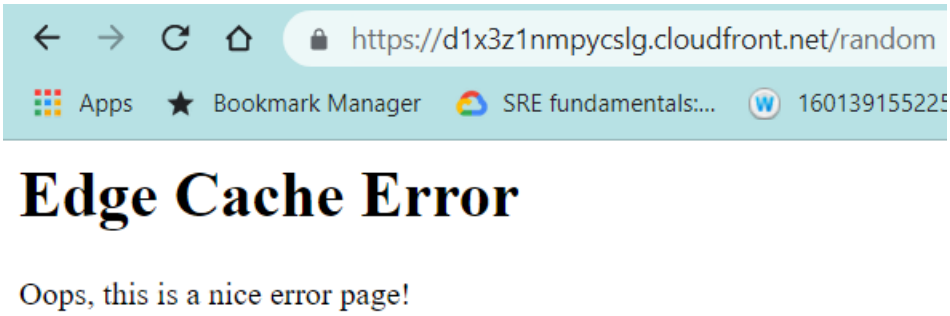
4. On the CloudFront console, within your distribution, go to the Error Pages tab and click the **Create Custom Error Response** button.

5. Configure the custom error response with the following settings.

- HTTP Error Code: 403 Forbidden
- Error Cacching Minimum TTL (seconds) : 60
- Customize Error Response : Yes
- Response Page Path : /error.html
- HTTP Response Code: 200 OK



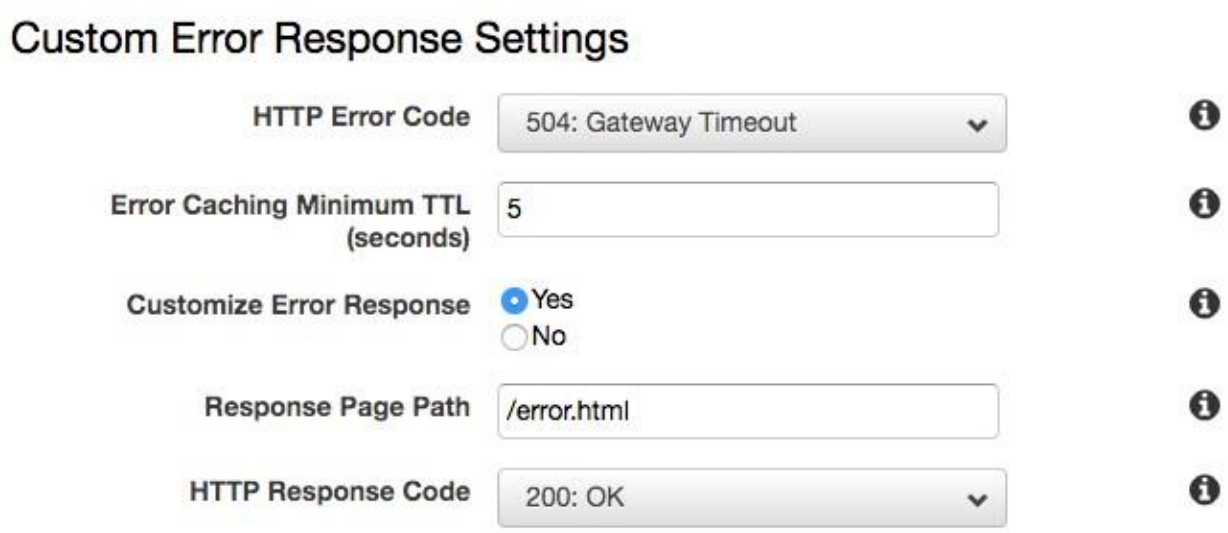
6. Test your custom error page, by requesting a random page from CloudFront. You may need a few minutes to wait for distribution to update and propagate to edge locations. Make sure that you use a different random value from the previous test, otherwise you will get the same cached version if you test within 5 minute.



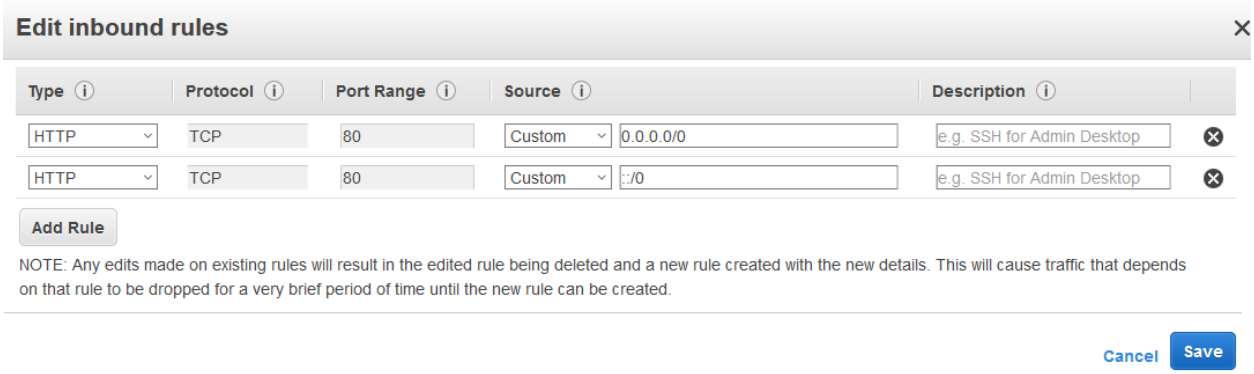
Optional – More Custom Error Exploration

Create another custom error page that will be triggered when the origin is not reachable by CloudFront. Use the following settings:

- HTTP Error Code: 504 Gateway Timeout
- Error Caching Minimum TTL (seconds) : 5
- Customize Error Response : Yes
- Response Page Path: /error.html
- HTTP Response Code: 200 OK



Go to the EC2 console and block inbound traffic to the EC2 instance which is hosting the Nodejs api.

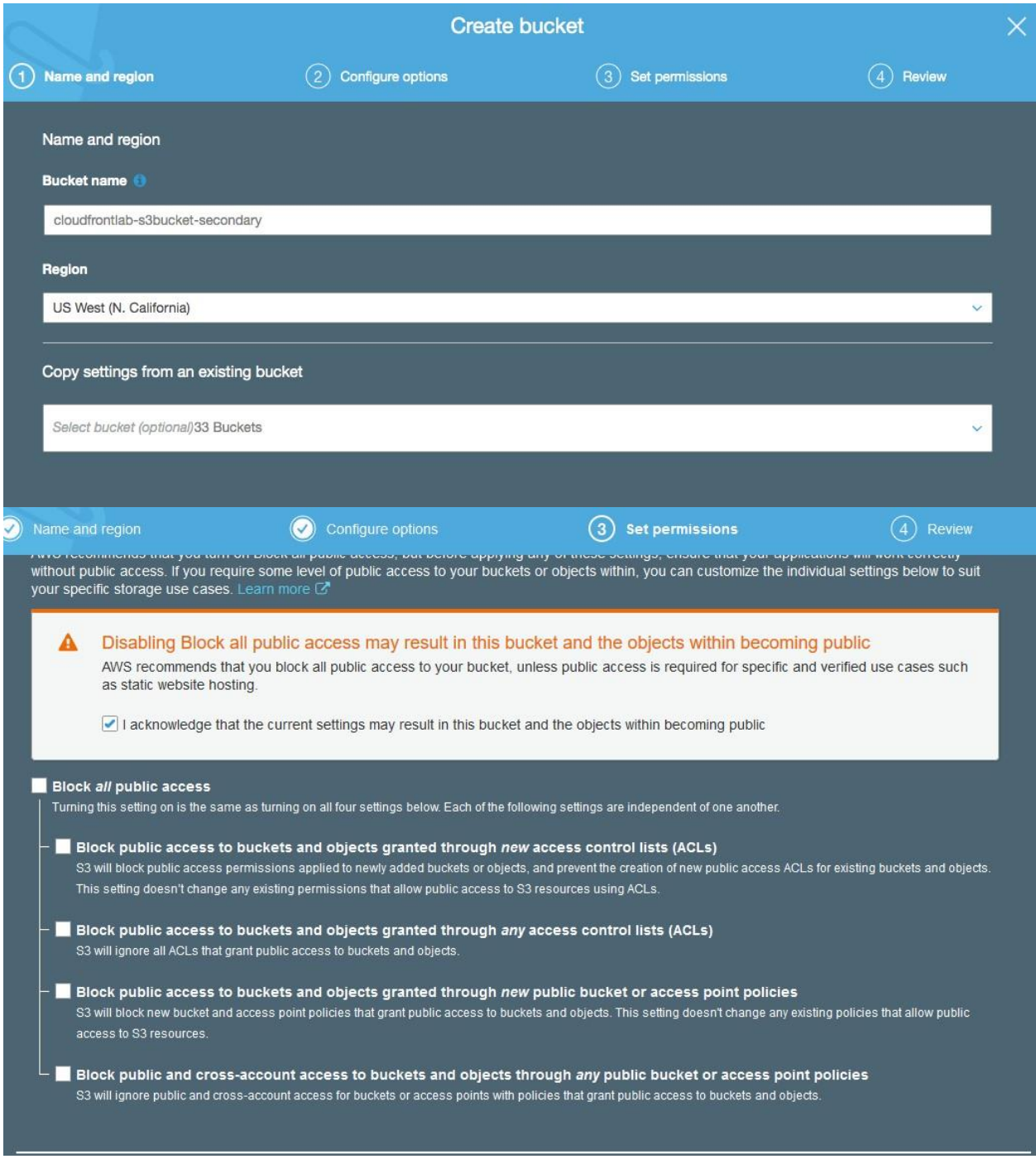


8 - Test your index.html page on the browser. Wait for a few moments until the API call fails gracefully to the custom error page.

Configure Origin Group

In this section, you will configure an origin group to provide rerouting during a failover event. You can associate an origin group with a cache behavior to have requests routed from a primary origin to a secondary origin for failover.

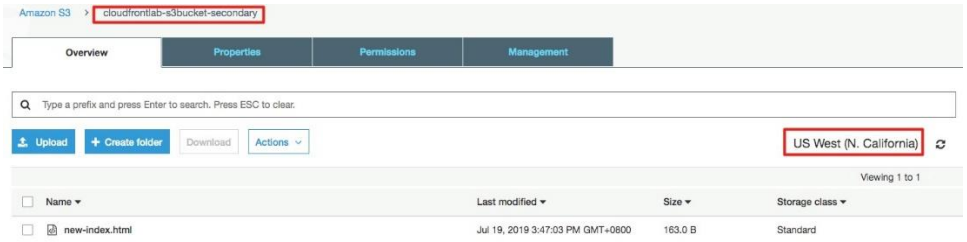
1. In S3 console, create a new S3 bucket in a different region, for example, us-west-1. Give it a unique name, remember S3 bucket names are globally unique, so add a personalized suffix, such as **cloudfrontlab-s3bucket-secondary-**. Uncheck the “block all public access” box.



2. Create a new-index.html file on your computer with the below HTML content and upload it to your new S3 bucket in us-west-1 from the S3 console. Make the new-index.html public by selecting the option from Object actions dropdown menu.

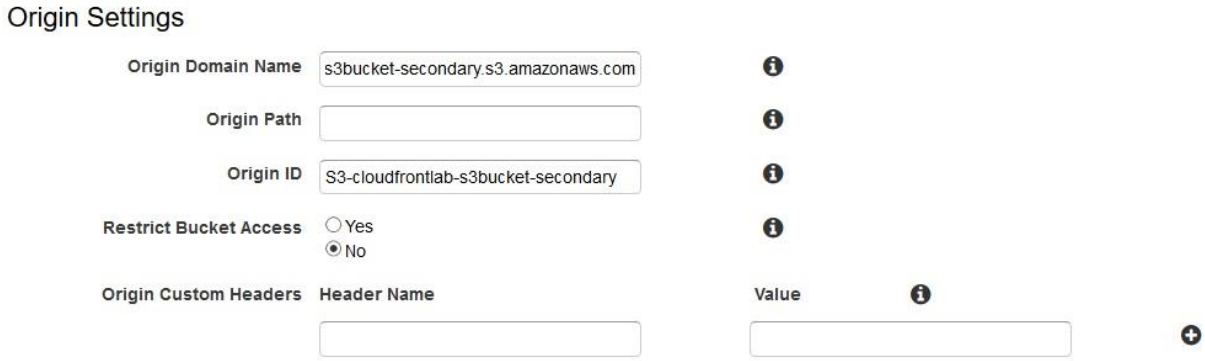
```
<html lang="en">
  <body>
    <h1>CloudFront Lab</h1>
    Hi, this is a page from my secondary Origin! We now support
Origin group and failover!
```

```
  </body>
</html>
```

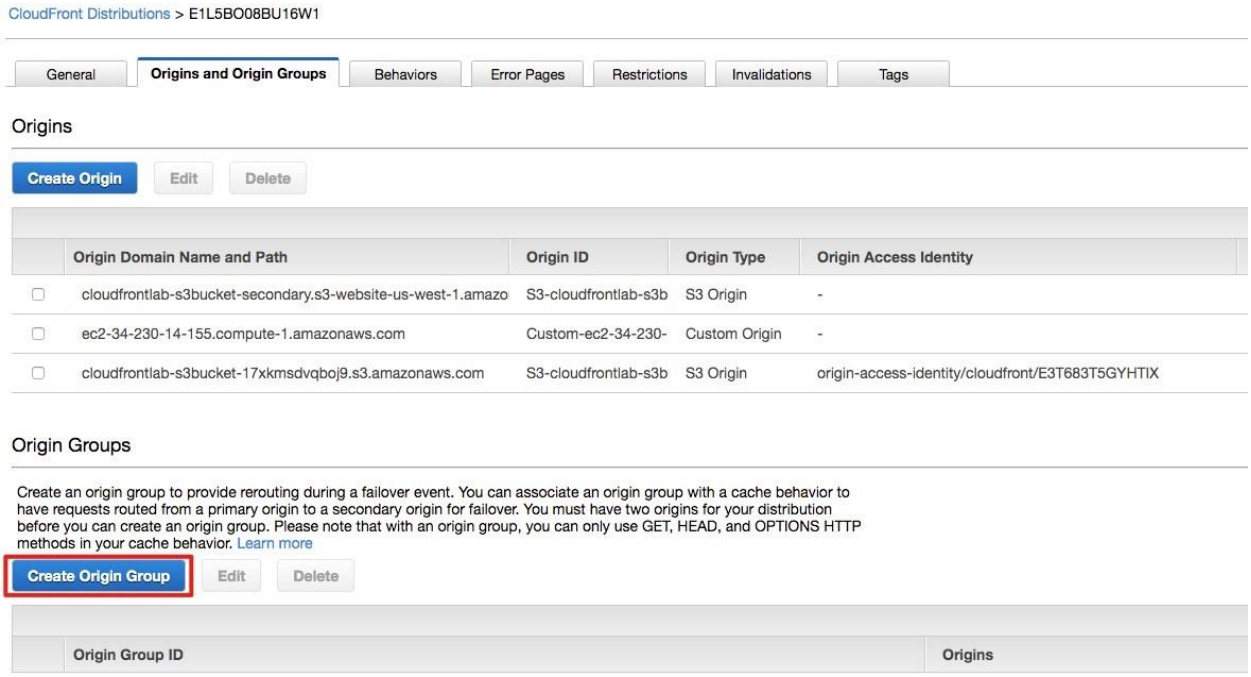


3. Go back to your CloudFront distribution and create a new Origin with the newly created S3 bucket in us-west-1.

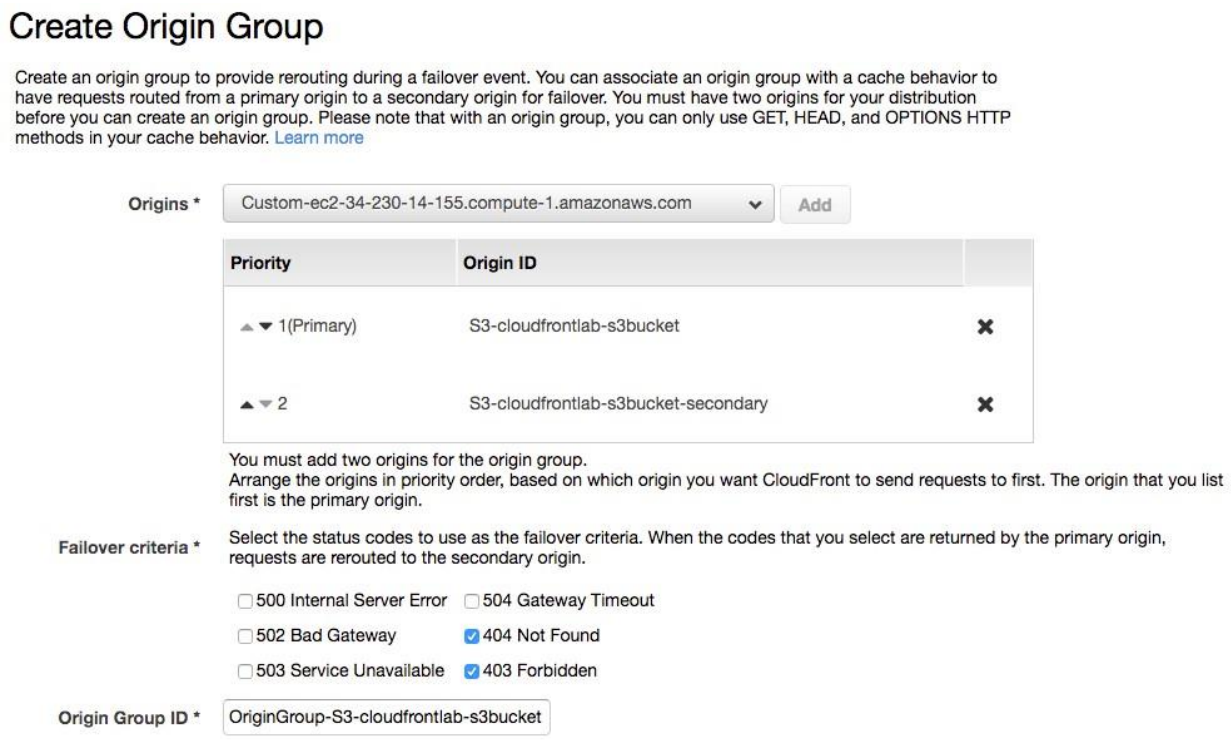
- Origin Domain Name: The website hosting Endpoint of Your New S3 bucket (like http://cloudfrontlab-s3bucket-secondary.s3-website-us-west-1.amazonaws.com)



4. Create an Origin Group with a primary and secondary origin. Go to CloudFront Origins and Origin Groups Tab, click Create Origin Group.

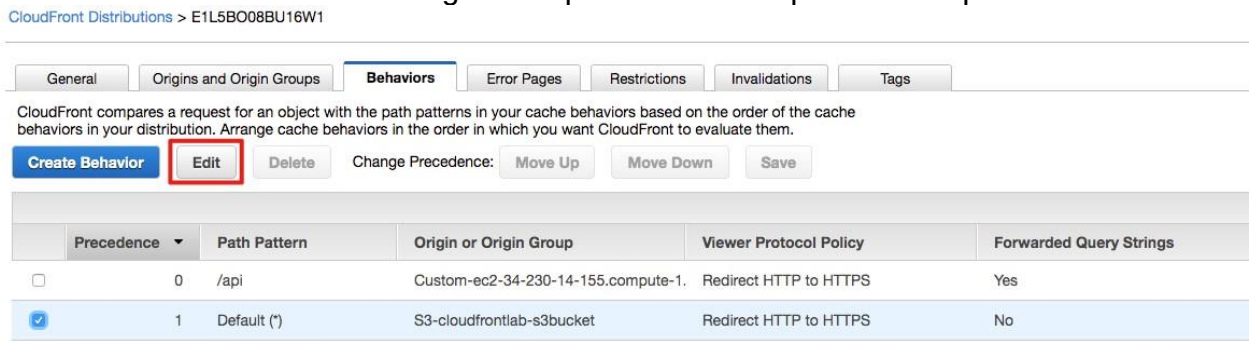


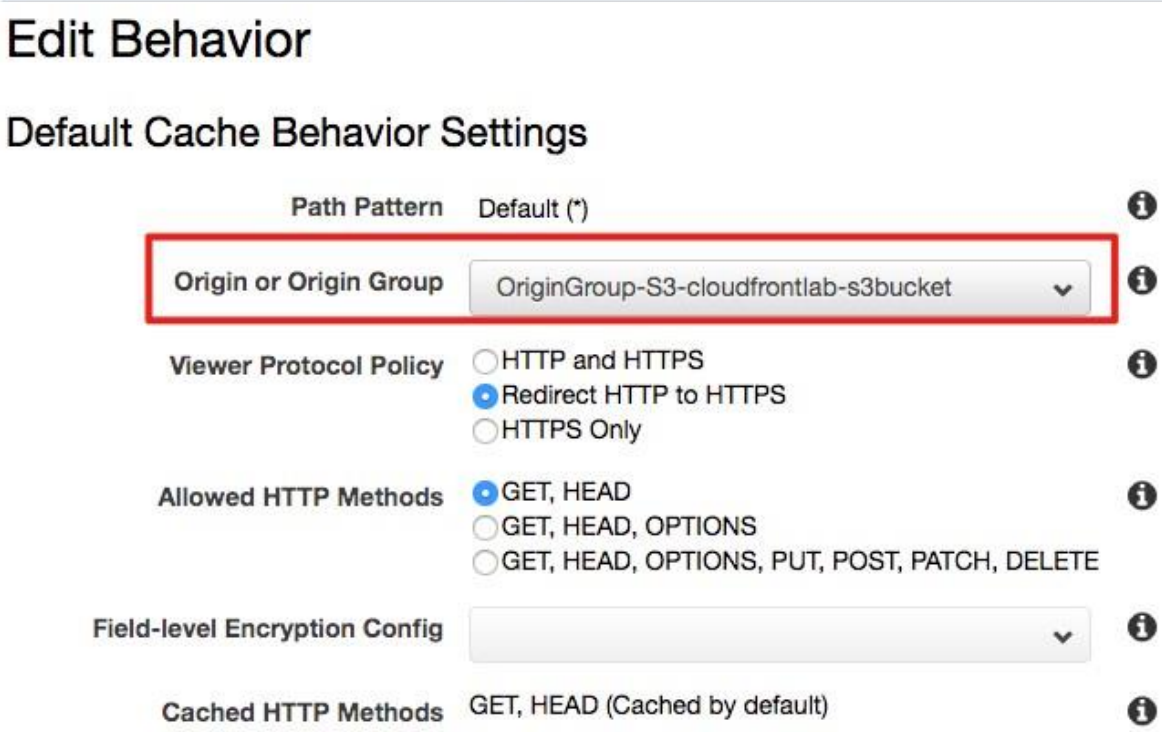
5. Use S3-cloudfrontlab-s3bucket as primary origin, and S3-cloudfrontlab-s3bucket-secondary as secondary origin.
   For failover criteria, choose 404 Not Found and 403 Forbidden.



6. Edit the default behavior of the distribution to use the new Origin Group, so that we can test failover. Go to CloudFront Behaviors Tab, select

Default(*), and click Edit.
Edit the behavior to use the Origin Group we created in previous step.





7. After the distribution status changed to Deployed, request new-index.html page from CloudFront, you can see your secondary S3 bucket origin serve your request correctly.