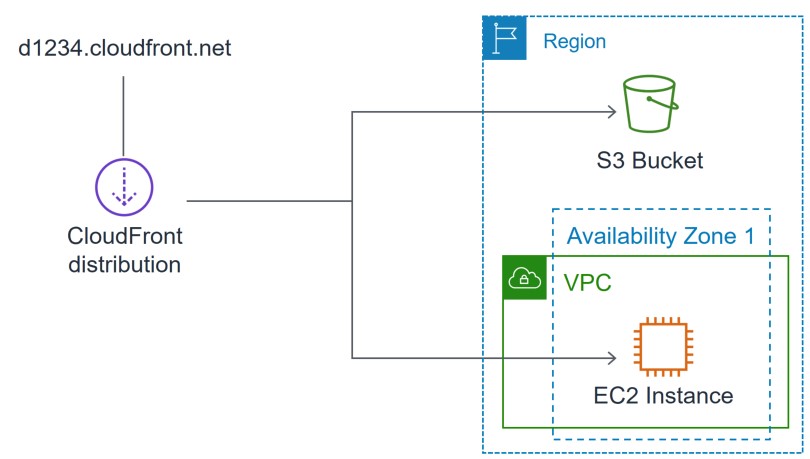
ACCELERATE YOUR CONTENT USING AMAZON CLOUDFRONT

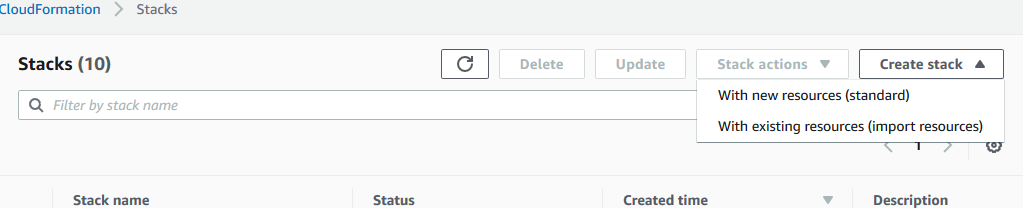
In this Lab, you will learn how to set up a CloudFront distribution to front a simple web application with static and dynamic content hosted on Amazon S3 and on an Amazon EC2 instance respectively, as per the below diagram. You will learn how to test it, and check what are the special headers sent by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

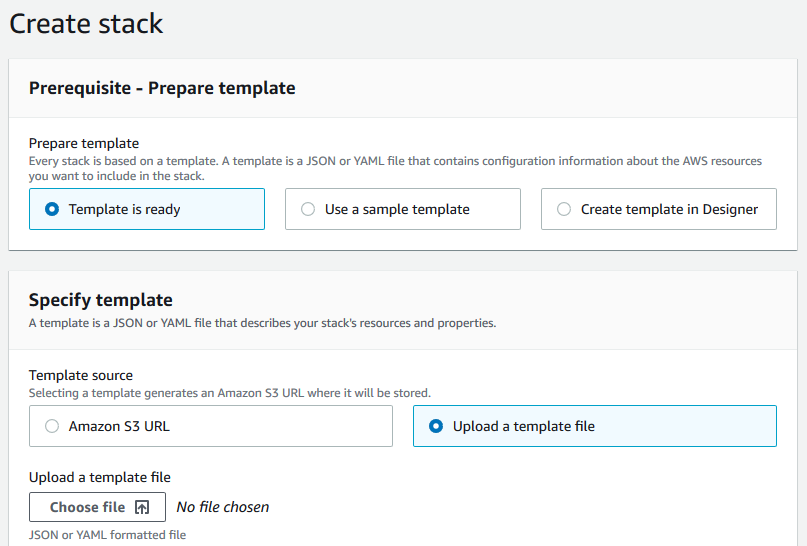
[](https://content-acceleration-cloudfront.workshop.aws/images/index-structure.jpg)

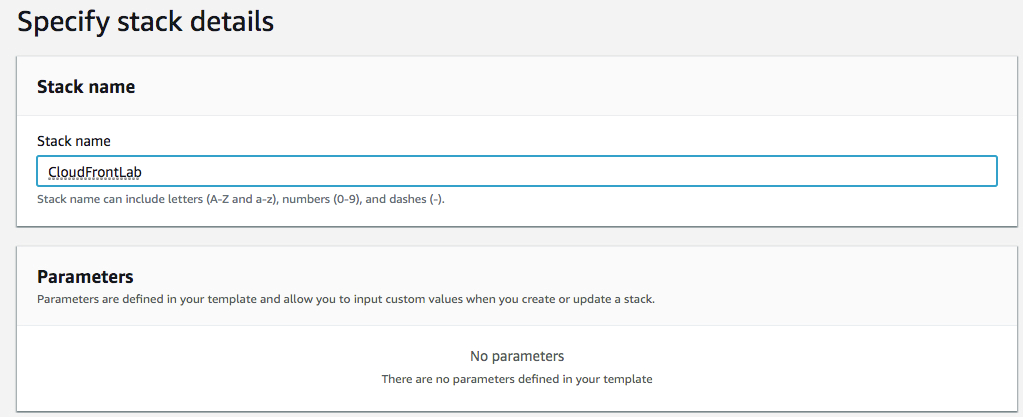
# CREATE EC2 & S3 ORIGINS

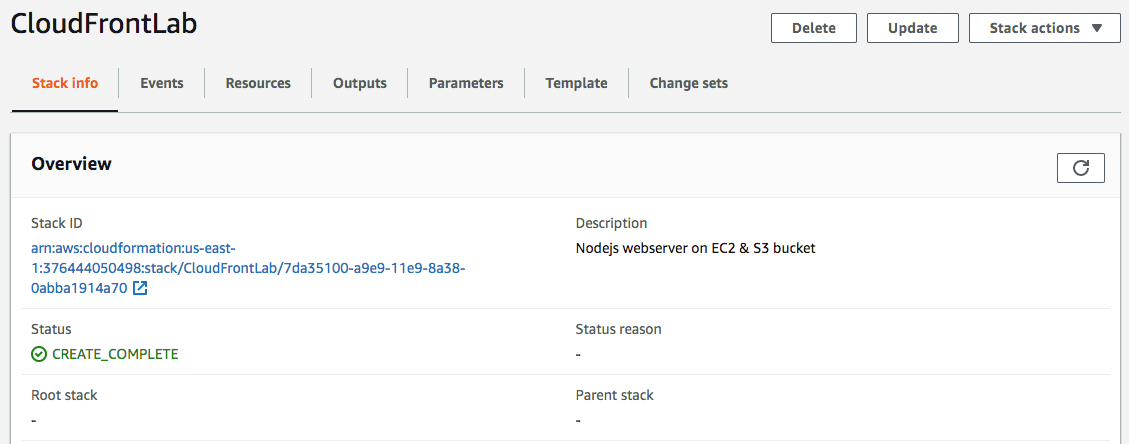
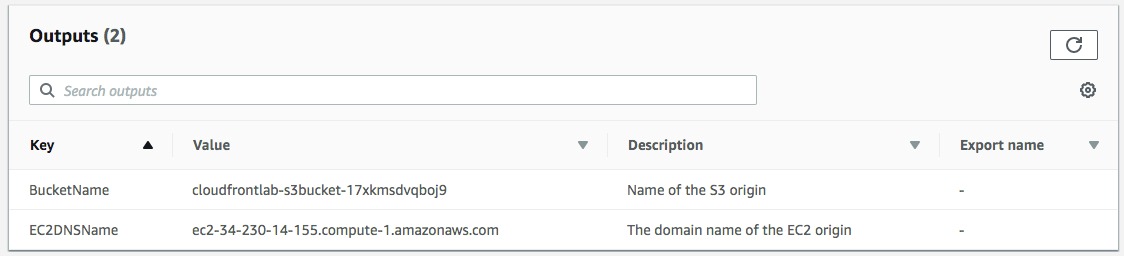
**In this section, you will create both S3 and EC2 origin using a provided CloudFormation template.**

### **Create new CloudFormation stack**

1 – Go to CloudFormation console in North Virginia ***us-east-1*** and select **Create stack with new resources** ( ‘Create stack’ > ‘With new resources (standard)’).[](https://content-acceleration-cloudfront.workshop.aws/images/create-ec2-s3-origin/image1.png)

2 - Choose “Template is ready” and “Upload a template file”, then choose and upload the following template file: [cloudfront-lab.yaml](https://content-acceleration-cloudfront.workshop.aws/cloudfront-lab.yaml)[](https://content-acceleration-cloudfront.workshop.aws/images/create-ec2-s3-origin/image2.png)

3 – Click Next, enter a name for your stack and click on Next twice (Options and Review) , then click Create stack.[](https://content-acceleration-cloudfront.workshop.aws/images/create-ec2-s3-origin/image3.png)

4 – Once the stack status shows CREATE\_COMPLETE, go to 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.[](https://content-acceleration-cloudfront.workshop.aws/images/create-ec2-s3-origin/image4.png)[](https://content-acceleration-cloudfront.workshop.aws/images/create-ec2-s3-origin/image5.png)

### **Create index.html in S3 bucket**

5 - 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.

<!DOCTYPE html>

<html lang="en">

<body>

<table border="1" width="100%">

<thead>

<tr><td><h1>CloudFront Lab</h1></td></tr>

</thead>

<tfoot>

<tr><td>Immersion Days - CloudFront</td></tr>

</tfoot>

<tbody>

<tr><td>Response sent by API</td></tr>

</tbody>

<tbody>

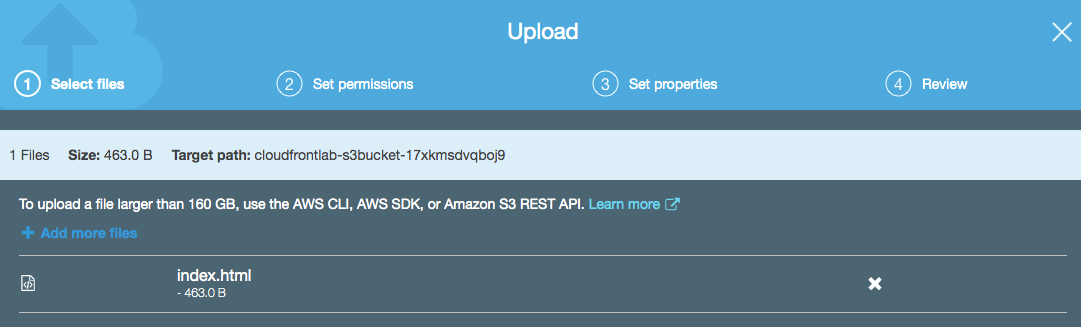
<tr><td> <iframe src='../api' style="width:100%; height:100%;"></iframe></td></tr>

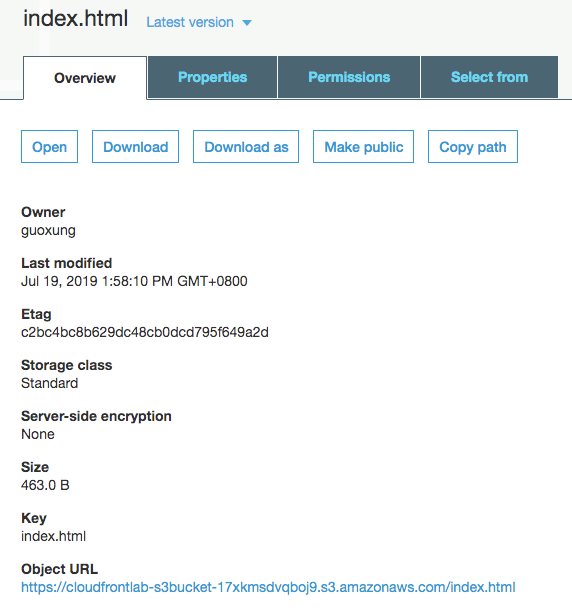
</tbody>

</table>

</body>

</html>

6 – On the AWS console, go to the S3 bucket created by CloudFormation and upload the index.html file. Leave all settings as default.[](https://content-acceleration-cloudfront.workshop.aws/images/create-ec2-s3-origin/image6.png)

7 - 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.[](https://content-acceleration-cloudfront.workshop.aws/images/create-ec2-s3-origin/image7.png)[](https://content-acceleration-cloudfront.workshop.aws/images/create-ec2-s3-origin/image8.png)

### **Verify EC2**

8 - 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))

}

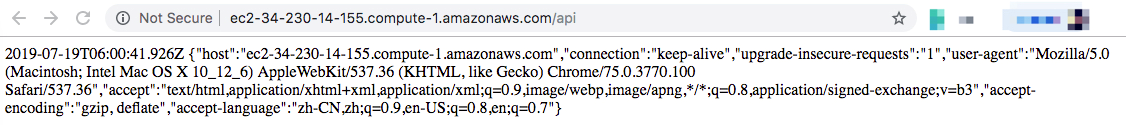
});

app.listen(8080, function () {

console.log('api is up!')

})

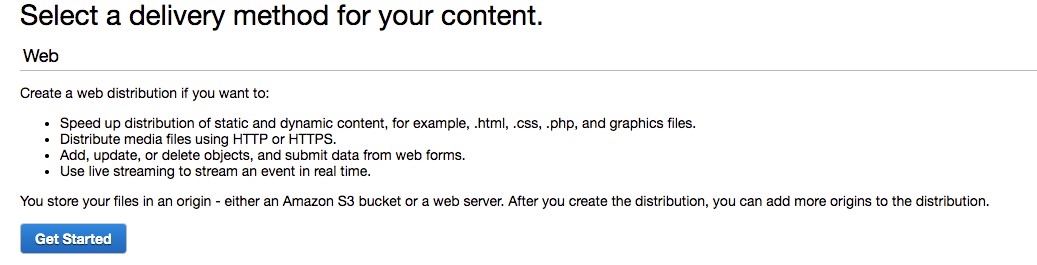
9 - Make sure the application work by entering the following in your browser: http://[EC2-DNS-name]/api.  
You should see a response similar to the example below.

[](https://content-acceleration-cloudfront.workshop.aws/images/create-ec2-s3-origin/image9.png)

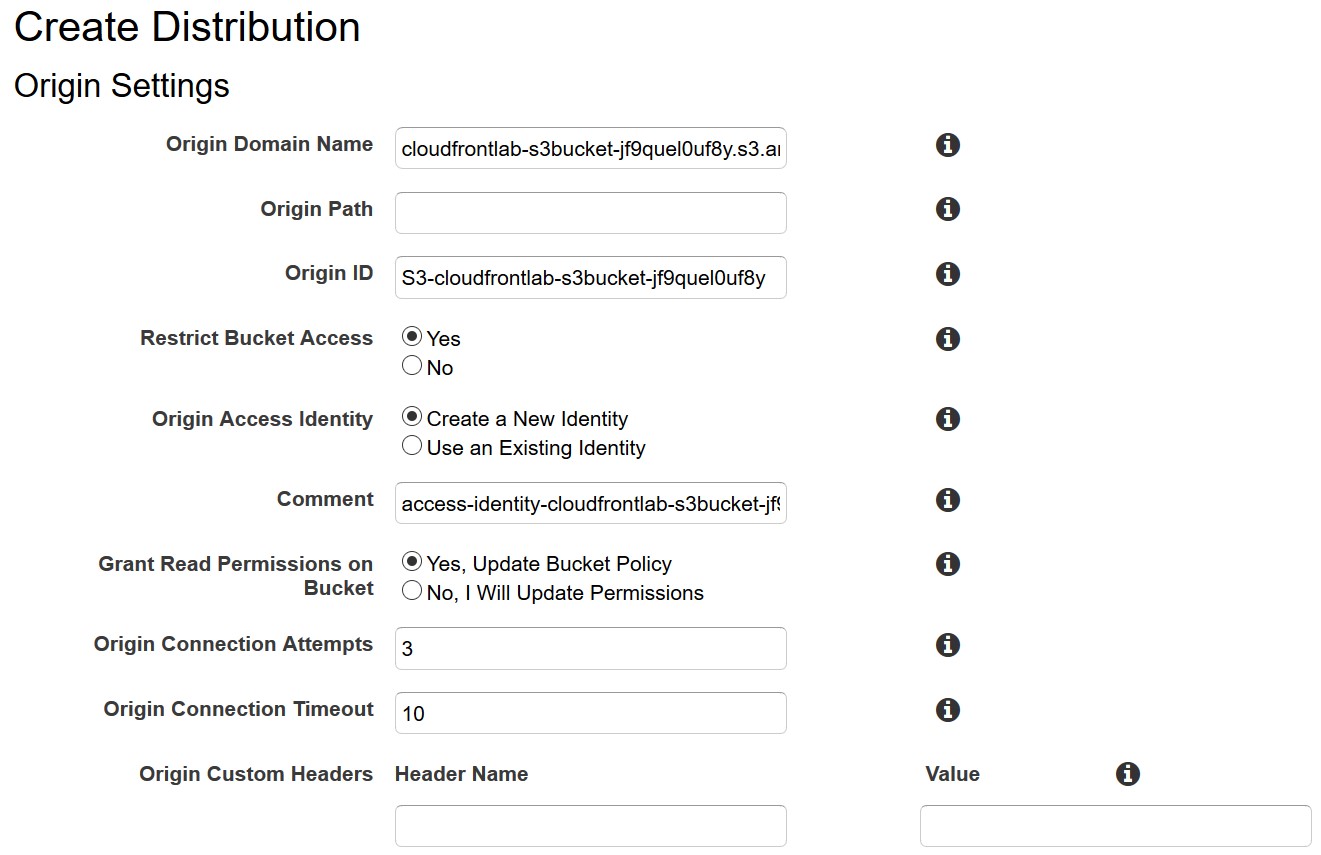
# CREATE A CLOUDFRONT DISTRIBUTION

**In this section we create CloudFront Distribution.**

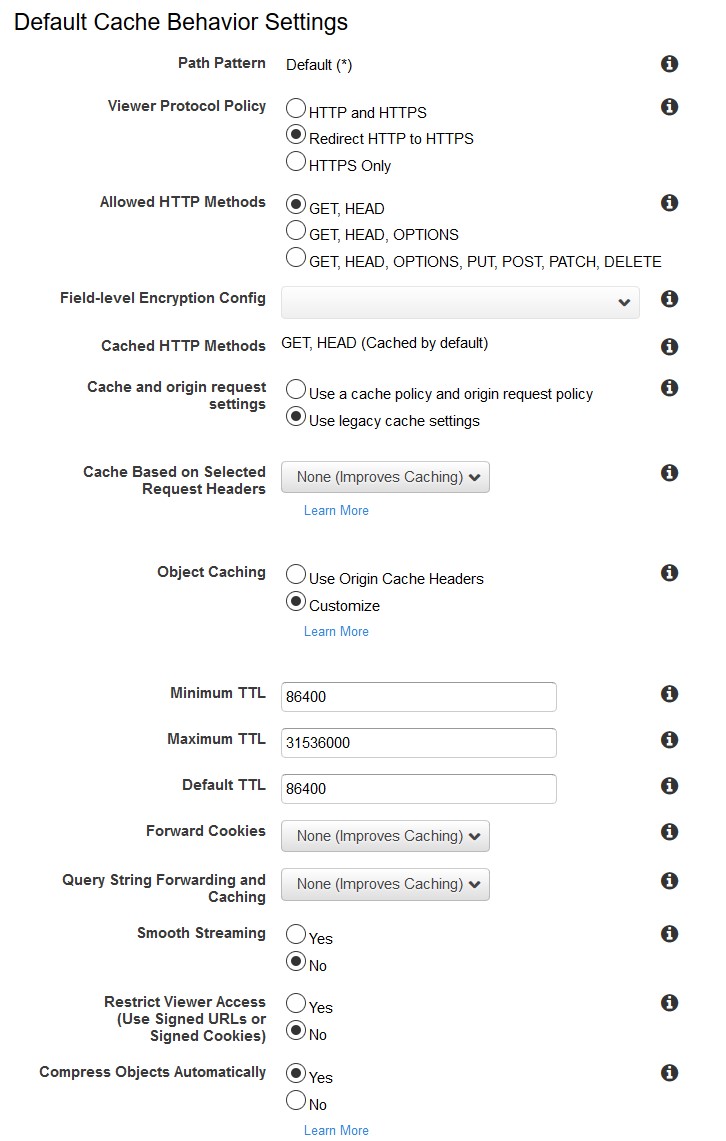
### **Create A CloudFront Distribution**

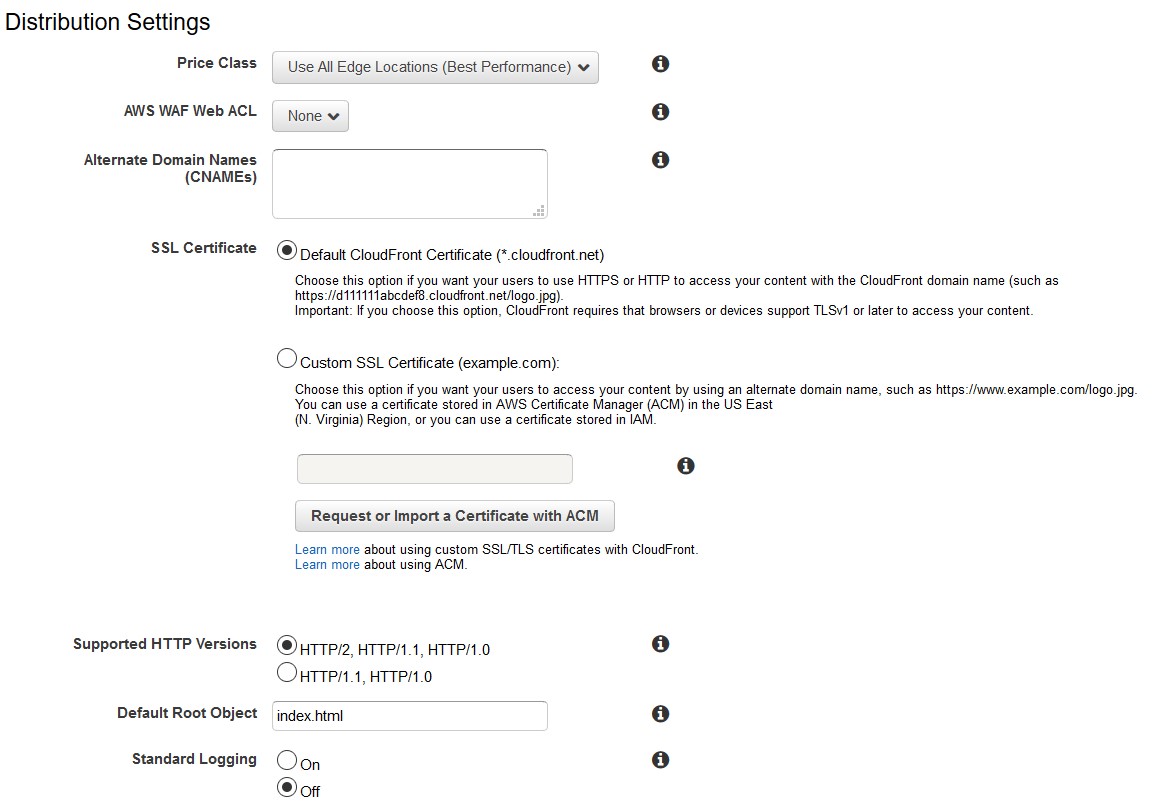
1 - Go to CloudFront on the AWS console and create a new distribution. If this is your first time to use CloudFront, click on Create Distribution in the Getting Started page. Otherwise, you can get started with a Web distribution.[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image1.jpg)

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

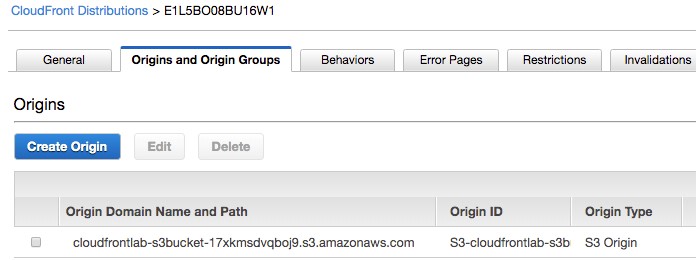
* Origin Domain Name: Your S3 bucket
* Restrict Bucket Access: Yes
* Origin Access Identity: Create a new Identity
* Grant Permissions on Bucket: Yes, Update Bucket policy[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image2.jpg)

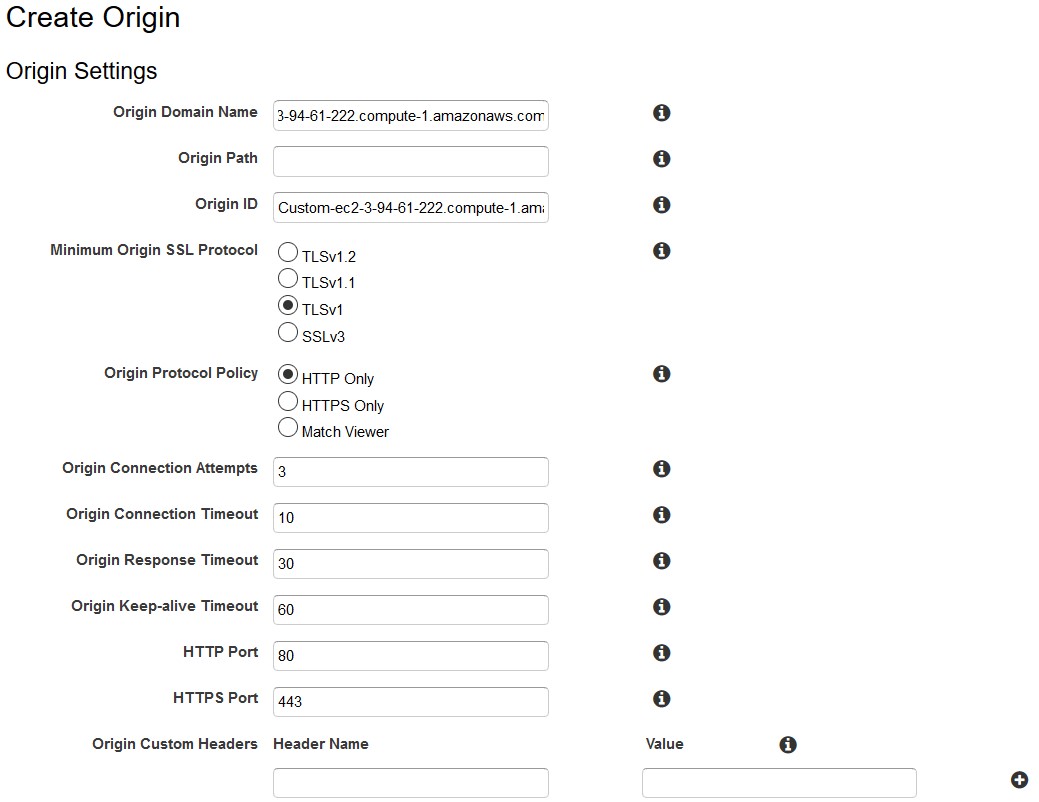
3 - Configure the Default Cache Behavior as follows:

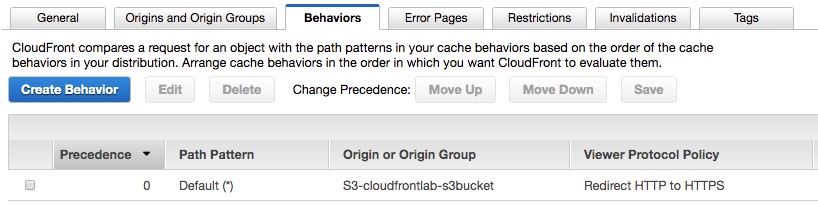
* Viewer Protocol Policy: Redirect HTTP to HTTPS
* Cache and origin request settings: Use legacy cache settings
* Object Caching: Customize
* Minimum TTL: 86400
* Compress Objects Automatically: Ye[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image3.jpg)

4 -In the Distributions Settings section, configure Default Root Object to index.html and 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.  
Click Create distribution. CloudFront start creating the 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.[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image4-1.jpg)[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image4-2.jpg)[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image4-3.jpg)

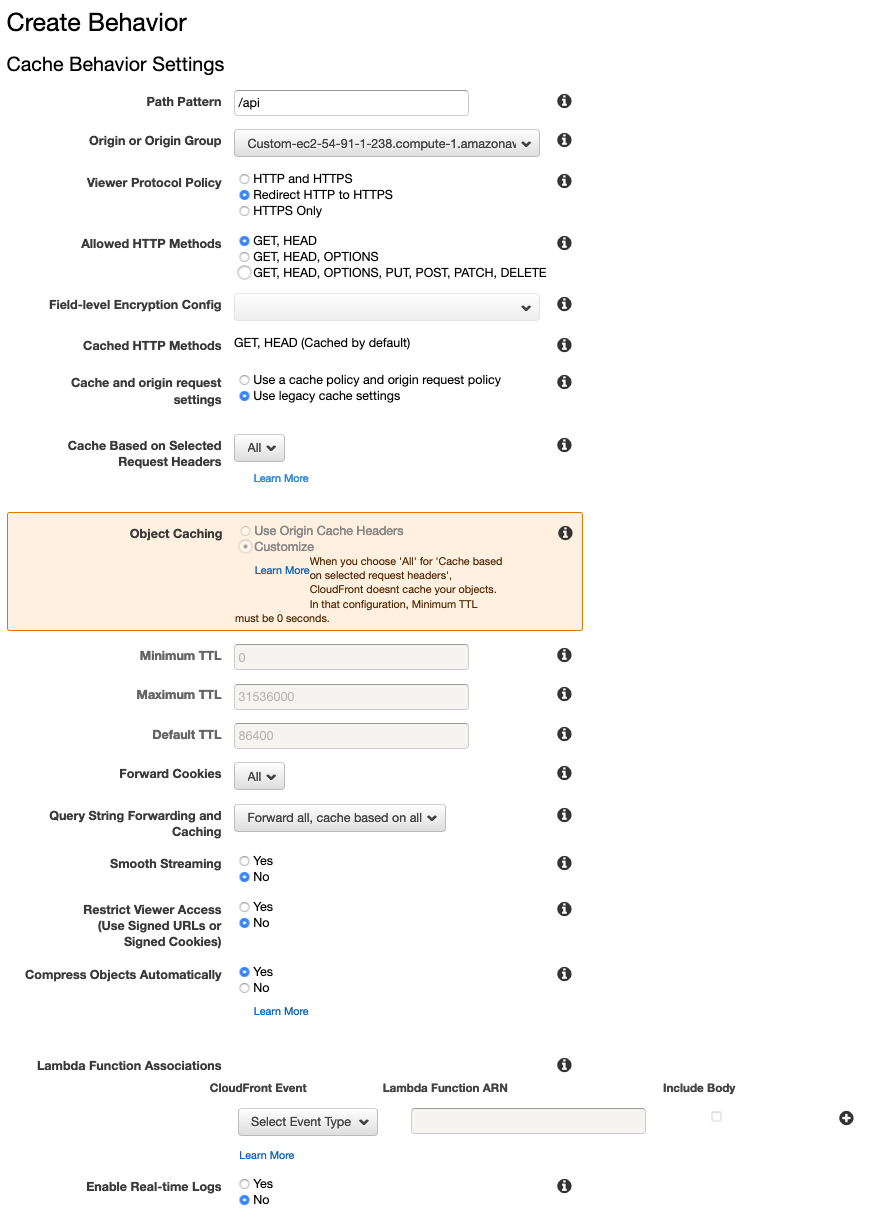
### **Adding EC2 Origin**

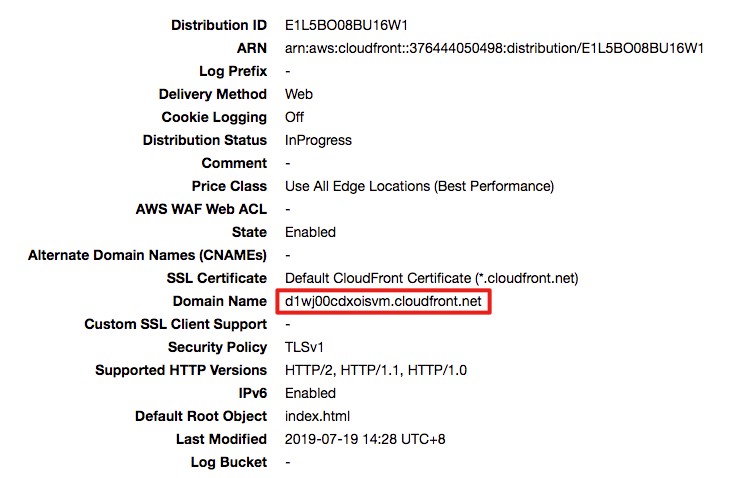
5 - In the distributions console, click on your distribution ID, then go to Origins and Origin Groups Tab to create another origin for the api. Click Origins and Origin Groups tab then click Create Origin button.[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image5.jpg)

6 – Enter the EC2 instance DNS name as the Origin Domain Name, and 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 only.  
[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image6.jpg)

7 - Create a new Behavior for /api. Select the Behaviors tab then click Create Behavior button.[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image7.jpg)

8 - Configure a second cache behavior to use the EC2 origin with following parameters to use CloudFront as proxy and bypass any caching layers.

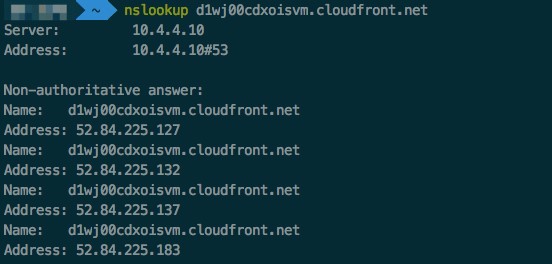
* Path Pattern: /api
* Viewer Protocol Policy: Redirect HTTP to HTTPS
* Cache Based on Selected Headers: All
* Forward Cookies: All
* Query String Forwarding and Caching: Forward all, cache based on all.
* Compress Objects Automatically: Yes[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image8.jpg)

9 - Check the unique domain name that CloudFront has associated to your distribution in the General tab.  
[](https://content-acceleration-cloudfront.workshop.aws/images/create-cloudfront-distribution/image9.jpg)

# TEST THE APPLICATION ON CLOUDFRONT

**In this section, you will test the application on CloudFront**

### **Test The Application On CloudFront**

1 - 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. 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.[](https://content-acceleration-cloudfront.workshop.aws/images/test-application-on-cloudfront/image1.jpg)

2 - When the propagation is complete, you can test the webpage on your browser as served by CloudFront using [http://dxxxx.cloudfront.net](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.[](https://content-acceleration-cloudfront.workshop.aws/images/test-application-on-cloudfront/image2.jpg)

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:

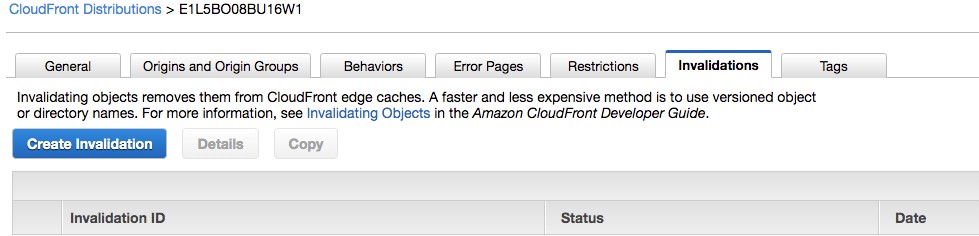
* **x-amz-cf-id** which holds the request id assigned by CloudFront.
* **x-amz-cf-pop** which indicates the CloudFront edge location that served your request. Each edge location is identified by a three-letter code and an arbitrarily assigned number, for example, DFW3. The three-letter code typically 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.  
  [](https://content-acceleration-cloudfront.workshop.aws/images/test-application-on-cloudfront/image3.jpg)

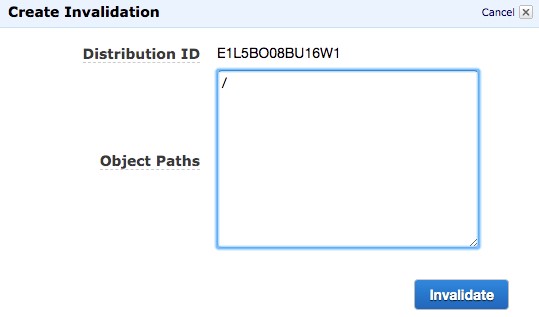
# TEST INVALIDATIONS

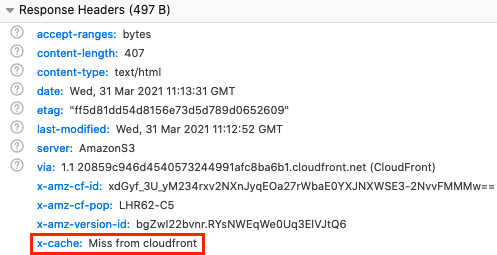
**In this section, you will run invalidation to deploy newly updated content.**

### **Test 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.  
[](https://content-acceleration-cloudfront.workshop.aws/images/test-invalidation/image1.jpg)

2 - Create an invalidation for your index.html. You can specify / in the Object paths because we already set index.html as default root object.[](https://content-acceleration-cloudfront.workshop.aws/images/test-invalidation/image2.jpg)

3 - 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.  
[](https://content-acceleration-cloudfront.workshop.aws/images/test-invalidation/image3.jpg)

# CONFIGURE CUSTOM ERROR PAGE

**In this section, you will configure a custom error page for graceful failover if the requested content is not found.**

### **Create A 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://content-acceleration-cloudfront.workshop.aws/images/custom-error-page/image1.jpg)

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>CloudFront Lab</h1>

Oups, this is a nice error page!

</body>

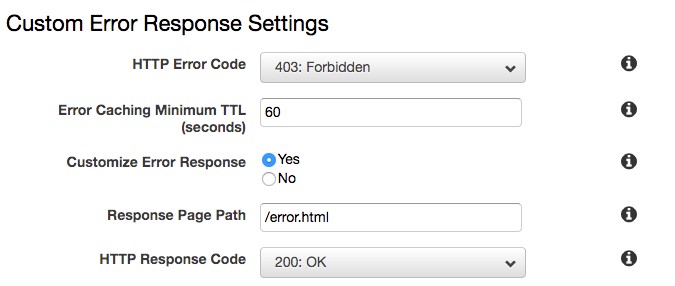
</html>

[](https://content-acceleration-cloudfront.workshop.aws/images/custom-error-page/image2.jpg)

### **Configure The Custom Error Page in CloudFront**

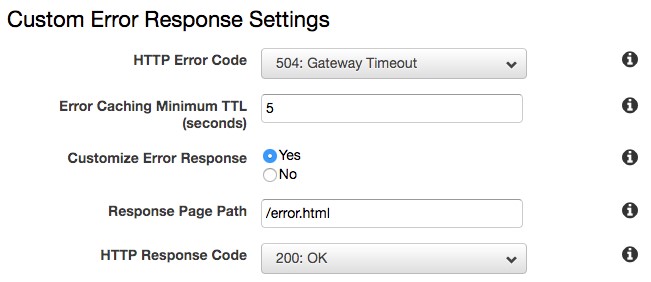
3 – On the CloudFront console, within your distribution, go to the Error Pages tab and click the **Create Custom Error Response** button.

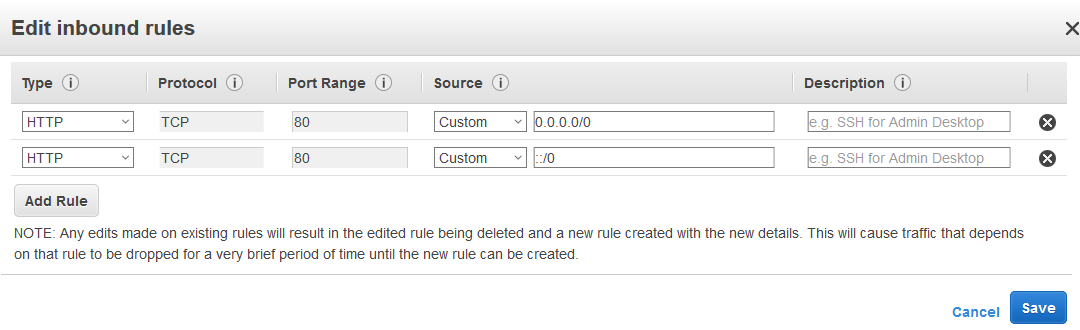
4 – Configure the custom error response with the following settings.

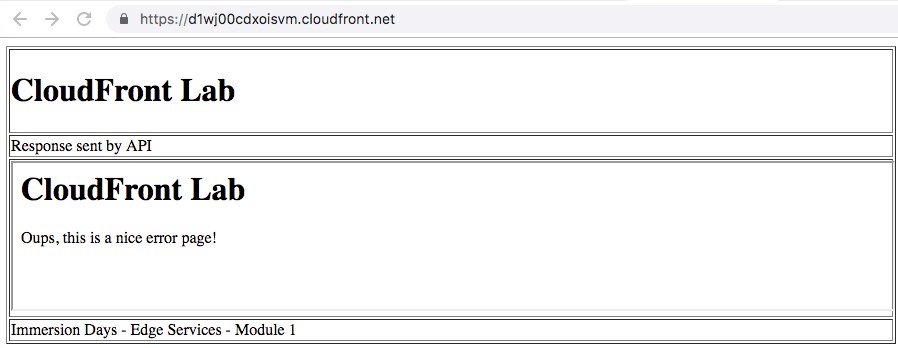
* HTTP Error Code: 403 Forbidden
* Error Cacching Minimum TTL (seconds) : 60
* Customize Error Response : Yes
* Response Page Path : /error.html
* HTTP Response Code: 200 OK[](https://content-acceleration-cloudfront.workshop.aws/images/custom-error-page/image3.jpg)

5 - 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.[](https://content-acceleration-cloudfront.workshop.aws/images/custom-error-page/image4.jpg)

6 - 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[](https://content-acceleration-cloudfront.workshop.aws/images/custom-error-page/image5.jpg)

7 – Go to the EC2 console and block inbound traffic to the EC2 instance which is hosting the Nodejs api.  
[](https://content-acceleration-cloudfront.workshop.aws/images/custom-error-page/image7.png)

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.  
[](https://content-acceleration-cloudfront.workshop.aws/images/custom-error-page/image8.jpg)