Working With Cross-Origin Resource Sharing

In this lesson, you will learn how to work with cross-origin resource sharing when using AWS Lambda.

Cross-origin resource sharing (CORS)

Client code will need to access resources from S3 and API Gateway, which will be on different domains. To prevent online fraud, browsers request special authorisation when a page from one domain wants to access resources on another domain. This is cross-origin resource sharing (CORS). Here's a quick introduction to how CORS works. It should be enough for your needs in this course (for a more in-depth introduction to CORS, check out the *Cross-Origin Resource Sharing* documentation page on the Mozilla Developer Network):

An *origin*, in browser terminology, is a combination of URL protocol, domain and, optionally, network port. So, for example, the origin https://runningserverless.com is distinct from https://gojko.net. Browsers will happily load scripts or images from a different origin during primary web page parsing, but they will not allow background network requests to different origins so willingly. For example, a page from runningserverless.com can include an image from gojko.net in its HTML contents without any special configuration. However, the same page will not be able to read the same image asynchronously using JavaScript unless the CORS settings of gojko.net explicitly allow it.

Pre-flight request

Before executing a network request from JavaScript code, browsers will verify

Before executing a network request from JavaScript code, browsers will verify that the page is actually allowed to access a resource on a different origin.

Browsers do that by sending a *pre-flight* request to the resource URL. The pre-flight request is an HTTP call using the *OPTIONS* method, including the resource it wants to access and the CORS context URL. It's essentially a browser asking the remote server: 'If I were to try making this request for a page from this origin, would you let me?'

The resource server is supposed to reply to the pre-flight request repeating the requested origin and providing a list of HTTP headers and methods it would allow for that resource. Technically, the server response needs to include the policy in the `Access-Control-Allow-Origin`, `Access-Control-Allow-Methods` and `Access-Control-Allow-Headers` HTTP headers.

The browser then compares the allowed methods and headers to the request the JavaScript code wants to send. If everything matches, it proceeds with the full request. Otherwise, it will make the JavaScript code think that there was a network error.

When the resource server responds to the full request (not just the pre-flight request), it also needs to include the *Access-Control-Allow-Origin* header, responding with the same origin as the current page. Without that, the browser will refuse to pass the result back to the JavaScript code.

> CORS security errors are particularly tricky to troubleshoot because they only apply to background browser actions. Running exactly the same request from a command line, for example using `curl`, won't show any problems. You also won't find any logs on the server about such problems, because browsers kill requests before they even reach your API.

When you move web assets to a separate website endpoint and move the client workflows to JavaScript, you'll introduce potential CORS issues into the application.

The first step in the figure, retrieving the web page, is a direct request, so it falls outside CORS.

The second step ('get pre-signed grants') will be a JavaScript call to the API

The second step ('get pre-signed grants') will be a JavaScript call to the API Gateway URL, which will be on a different origin, so it will be restricted by

CORS.

The third step ('upload file') will post a form dynamically to the uploads bucket URL, also on a different origin and restricted by CORS.

The fourth step ('get results') also requires CORS, because you'll be polling the results bucket dynamically to check whether the file is ready.

Configuring S3 buckets for CORS

The second and third CORS issues are easy to address. Because S3 supports basic web serving features, including CORS, you can use CloudFormation to configure CORS policies on S3 buckets. You just need to add the CORS origin (the website endpoint for the static assets) to the upload and thumbnails bucket using the `CorsConfiguration` property.

The upload bucket should only allow `POST` requests across origins, and only from the pages loaded from the assets bucket. The `UploadS3Bucket` resource is changed in your template to match the following listing (the important changes are in lines 8-16).

```yaml
UploadS3Bucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketEncryption:
      ServerSideEncryptionConfiguration:
        - ServerSideEncryptionByDefault:
            SSEAlgorithm: AES256
    CorsConfiguration:
      CorsRules:
        - AllowedHeaders:
            - "*"
          AllowedMethods:
            - POST
          AllowedOrigins:
            - !GetAtt WebAssetsS3Bucket.WebsiteURL
          MaxAge: 3600
```

Line 32 to Line 47 of code/ch11/template.yaml

The results bucket should only allow `GET` requests across origins, from the pages loaded from the web assets bucket. The `ThumbnailsS3Bucket` resource in your application template is changed to match the following listing (the important changes are in lines 8-16).

important changes are in lines 8-16).

```
ThumbnailsS3Bucket:
  Type: AWS::S3::Bucket
  Properties:
    BucketEncryption:
      ServerSideEncryptionConfiguration:
        - ServerSideEncryptionByDefault:
            SSEAlgorithm: AES256
    CorsConfiguration:
      CorsRules:
        - AllowedHeaders:
            - "*"
          AllowedMethods:
            - GET
          AllowedOrigins:
            - !GetAtt WebAssetsS3Bucket.WebsiteURL
          MaxAge: 3600
```

Line 48 to Line 63 of code/ch11/template.yaml

For more information on the `CorsRules` setting and the `WebsiteURL` property of the S3 bucket resource, check out the AWS::S3::Bucket documentation page.

You will learn how to configure the API gateway for CORS in the next lesson!