IP Detector Design Document and Test Report

Application Architecture and Topology

- Topology

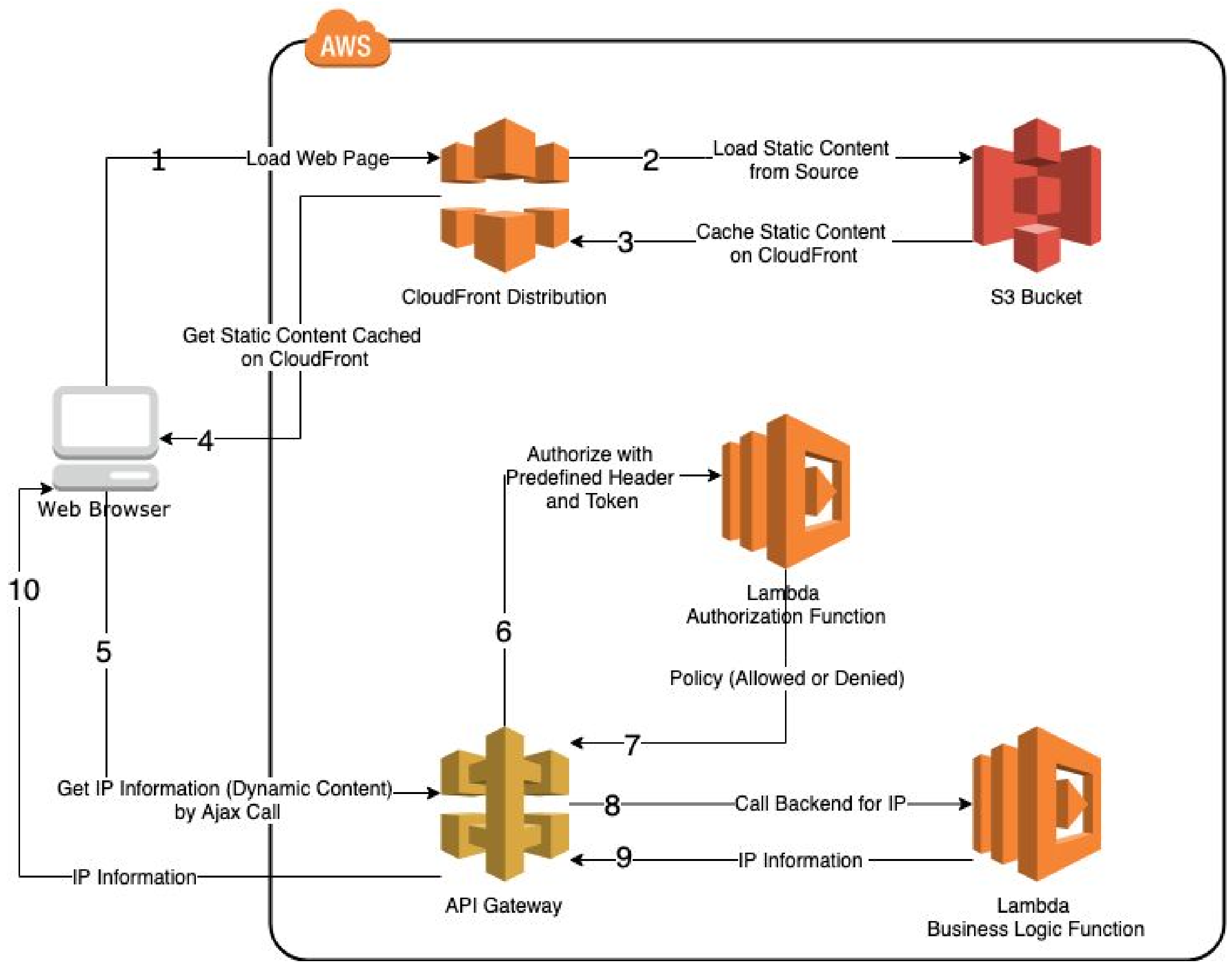

AWS
1
Load Web Page
2
Load Static Content from Source
3
Cache Static Content on CloudFront
CloudFront Distribution
S3 Bucket
Get Static Content Cached on CloudFront
4
Web Browser
Authorize with Predefined Header and Token
Lambda Authorization Function
6
10
5
Policy (Allowed or Denied)
7
Get IP Information (Dynamic Content) by Ajax Call
8
Call Backend for IP
9
IP Information
IP Information
API Gateway
Lambda Business Logic Function

Figure 1

- Application Flow

1. As illustrated in Figure 1, the browser loads the application from it's URL. A user would probably think it's just a website. However, it's the URL to a CloudFront Distribution.
2. CloudFront further checks whether the home page itself and the resource files it requires are cached on CloudFront or not. If they are not cached, it loads files from origin, which is a S3 Bucket.
3. Files loaded from a S3 Bucket are now cached on CloudFront. For further requests to these files, they will be served directly from CloudFront. However, files are not cached on CloudFront forever. It depends on the TTL set on CloudFront or can be invalidated by administrators through AWS console.

4. Static contents are now downloaded from CloudFront.
5. An Ajax call initiated to get the IP information is coded in the Javascript of the downloaded home page. The Ajax call is requesting to a http method hosted on API Gateway.
6. It may not be a good idea for users who get the URL of the API method to get responses due to security reason or better resource control. To achieve that, we put a Lambda authorization function in place to decide whether a request is good to go or not based on the predefined header and token.
7. If the token is authorized by the Lambda function, it issues an allowed policy. Otherwise, it returns the denied policy to API Gateway and lets it returns a 403 forbidden response.
8. Once the request gets authorized, it passes the request to the Lambda business logic function. This Lambda functions analyses headers of the request, retrieves the IP Information and adds it to the response of the request.
9. Lambda returns the IP Information to API Gateway.
10. API Gateway returns IP Information to the Ajax call initiated by the browser and rendered the IP Information by Javascript.

Design Details

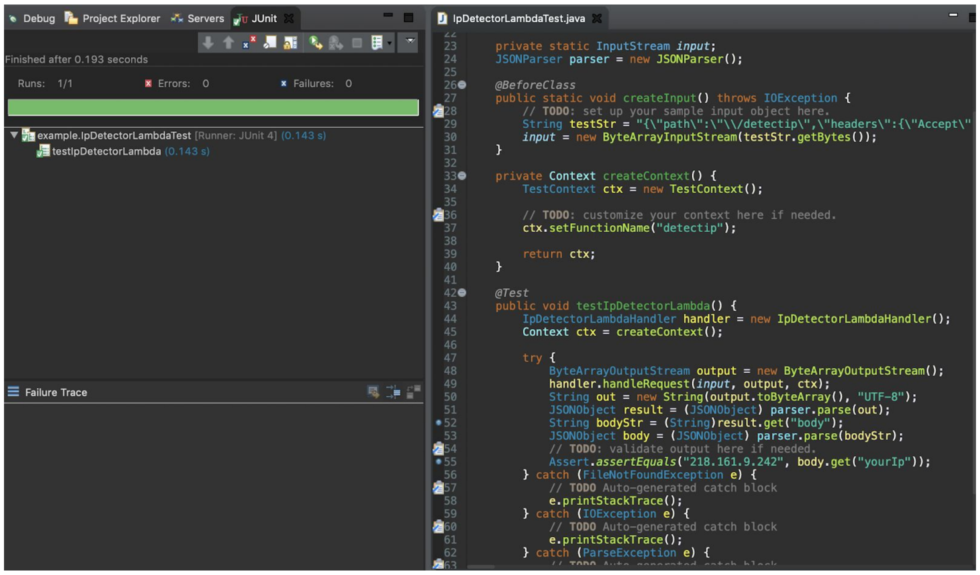
- CDN
    - Domain Name: d3ucfn7czb26to.cloudfront.net
    - We use CloudFront as our CDN solution. The reason why we use CloudFront is that we want to leverage its high performance, high availability, and scalability. Assume that our website is getting popular and lots of users are visiting the website at the same time, the amount of network traffic to the site would increase rapidly. With CloudFront, we don't need to worry about. It would scale up as it grows. Even if there is something wrong with the static origin and dynamic backend, the cached static version on CloudFront still works well. Besides, it always serves the static content from an edge server that is closest to the client to achieve the best performance.
    - Another reason to choose CloudFront is that it integrates with S3 for its OAI (Origin Access Identity). By enabling OAI, there is no other ways to access S3 content but through CloudFront URLs. The OAI I use is origin-access-identity/cloudfront/E1OSCFBXXS0TUU .
- Storage for Static Contents
    - Bucket Name: ip-detector-yichan-ho
    - Again, we use S3 for its integration with CloudFront and OAI. Also, S3 has different pricing models based on the access frequency of objects.
    - In the bucket, I put one folder named js and some files. The js folder contains jQuery library which is required by index.html in the root level. The index.html is the home page of the application. It has a Javascript snippet that triggers Ajax call for IP information. The favicon.ico required by the browser and the aws logo image are also kept in the bucket.
- Microservice API

    - Invoke URL: https://171nk3xfcb.execute-api.ap-northeast-1.amazonaws.com/prod/detectip
    - The API is served by API Gateway and it's a HTTP GET method.
    - I choose Lambda Function for the integration type and uses the Lambda Function named detectIp.
    - The API URL can't be hit directly. It requires a predefined header named Authorization and the value to the header has to be allow to make it through.
- Authorization
    - The Authorization is implemented in a Lambda function with Java 8.
    - Function Name: javaTokenAuthorizer
    - Function ARN: arn:aws:lambda:ap-northeast-1:177683156638:function:javaTokenAuthorizer
    - I use the Java Blueprint obtained from GitHub and customize it to meet the my requirement. It takes the token retrieved from TokenAuthorizerContext and checks if the value is allow. If so, it returns the Allow Policy. Otherwise, a Deny Policy is returned.
    - The Authorizer powered by Lambda has to be configured in API Gateway. The token source of the authorizer must be Authorization to conform with the CORS configuration set in ip-detector-yichan-ho s3 bucket.
    - After the Authorizer is created, we also need to set the authorization of the target method execution to this authorizer. Here, I name it tokenAuthorizer.
- Backend Logic
    - The backend logic is also implemented in a Lambda function with Java 8.
    - Function Name: detectIp
    - Function ARN: arn:aws:lambda:ap-northeast-1:177683156638:function:detectIp
    - Based on the Java template from AWS tutorial, my code is to parse the headers passed in InputStream of the request, look for X-Forwarded-For, get the list stored in it. Ideally, the first IP in the list is the public IP of user. As for remote server IP, there is no relevant header of it. Instead, I use header origin, it is the domain name where the request gets initiated and is also the domain name of the CloudFront distribution. I further translate the domain name into IP by Java's InetAddress.getAllByName() method.
    - It is also import to add Access-Control-Allow-Methods and Access-Control-Allow-Headers as the headers to the response to resolve CORS issue.
- Source Code
    - Static content of the application
        - Filename: static_content.zip
        - Also available at https://s3-ap-northeast-1.amazonaws.com/tam-test-yichan-ho/static_content.zip
        - Extract the zip file. Go into the static_content folder and upload all files and folder to a bitbucket

    - Authorizer implemented in Java
        - Filename: lambda-java-example-token-authorizer.zip
        - Also available at https://s3-ap-northeast-1.amazonaws.com/tam-test-yichan-ho/lambda-java-example-token-authorizer.zip
        - Extract the zip file and follow the README.txt in it.
    - Backend logic implemented in Java
        - Filename: lambda-java-example.zip
        - Also available at https://s3-ap-northeast-1.amazonaws.com/tam-test-yichan-ho/lambda-java-example.zip
        - Extract the zip file and follow the README.txt in it.
- Deployment
    - Basically, I use AWS console to accomplish my deployment.
    - However, I also make a CloudFormation to stimulate my manual deployment.
    - I do not 100% realize the deployment with CloudFormation. What I have achieve is to automatically deploy backend logic lambda, authorizer lambda, and the API gateway for the API method.
    - Template JSON: IpDetectorCloudFormation.json
    - Also available at https://s3-ap-northeast-1.amazonaws.com/tam-test-yichan-ho/IpDetectorCloudFormation.json

Test Report

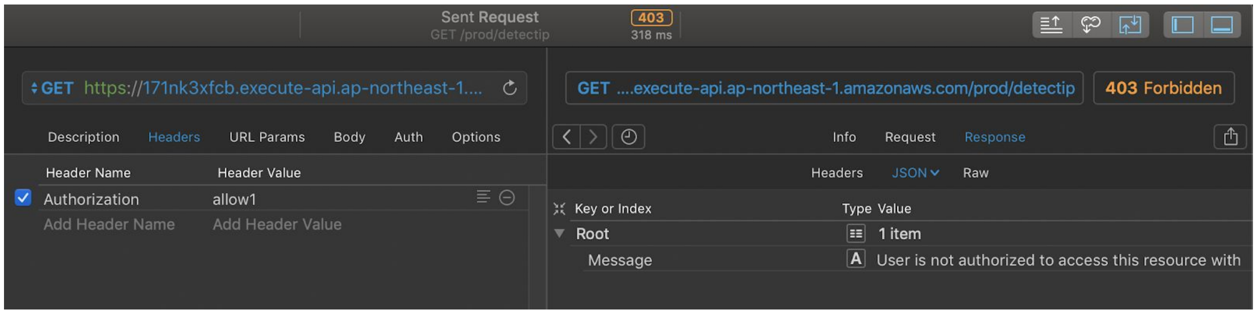
- Sample Application URL https://d3ucfn7czb26to.cloudfront.net/index.html
- Java Unit Test
    - Triggered in Eclipse with Java
    - Given the mock input with 218.161.9.242 in X-Forwarded-For, assert the return IP equals 218.161.9.242
    - Test Result: Pass



```
Finished after 0.193 seconds

Runs:  1/1        Errors:  0        Failures:  0

example.IpDetectorLambdaTest [Runner: JUnit 4] (0.143 s)
    testIpDetectorLambda (0.143 s)
```

```java
    private static InputStream input;
    JSONParser parser = new JSONParser();

    @BeforeClass
    public static void createInput() throws IOException {
        // TODO: set up your sample input object here.
        String testStr = "{\"path\":\"\\/detectip\",\"headers\":{\"Accept\"
        input = new ByteArrayInputStream(testStr.getBytes());
    }

    private Context createContext() {
        TestContext ctx = new TestContext();

        // TODO: customize your context here if needed.
        ctx.setFunctionName("detectip");

        return ctx;
    }

    @Test
    public void testIpDetectorLambda() {
        IpDetectorLambdaHandler handler = new IpDetectorLambdaHandler();
        Context ctx = createContext();

        try {
            ByteArrayOutputStream output = new ByteArrayOutputStream();
            handler.handleRequest(input, output, ctx);
            String out = new String(output.toByteArray(), "UTF-8");
            JSONObject result = (JSONObject) parser.parse(out);
            String bodyStr = (String)result.get("body");
            JSONObject body = (JSONObject) parser.parse(bodyStr);
            // TODO: validate output here if needed.
            Assert.assertEquals("218.161.9.242", body.get("yourIp"));
        } catch (FileNotFoundException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (ParseException e) {
```

- PAW Unit Test
  - Case 1: Enter the Authorization header with illegal value
    - Expected Result: 403 Forbidden and not getting IP Information
    - Test Result: Pass



Sent Request 403
GET /prod/detectip 318 ms

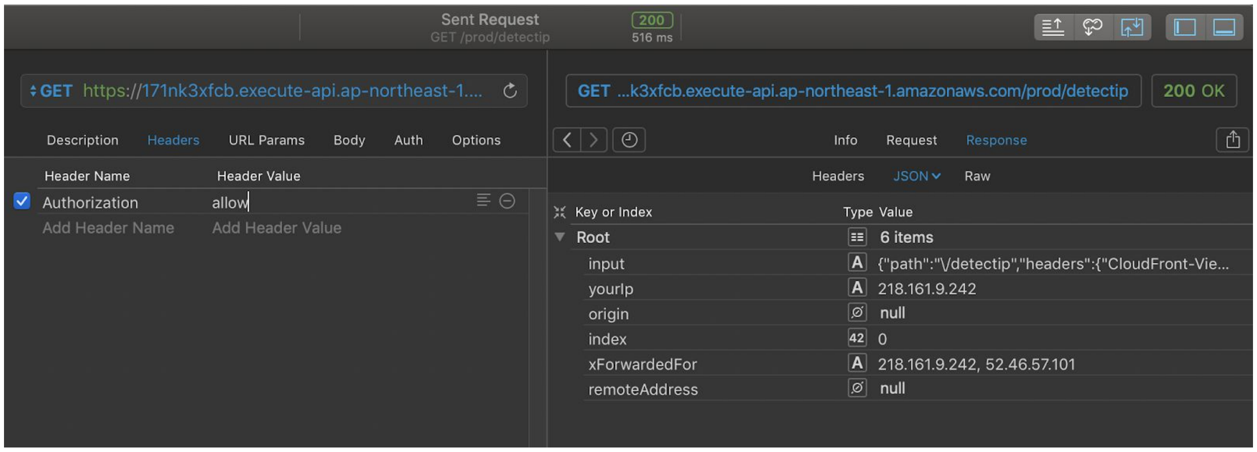
GET https://171nk3xfcb.execute-api.ap-northeast-1....

Header Name	Header Value
Authorization	allow1
Add Header Name	Add Header Value

GET ....execute-api.ap-northeast-1.amazonaws.com/prod/detectip 403 Forbidden

Key or Index	Type Value
Root	1 item
Message	User is not authorized to access this resource with

  - Case 2: Enter the Authorization header with allow
    - Expected Result: 200 Ok and get IP Information
    - Test Result: Pass



Sent Request 200
GET /prod/detectip 516 ms

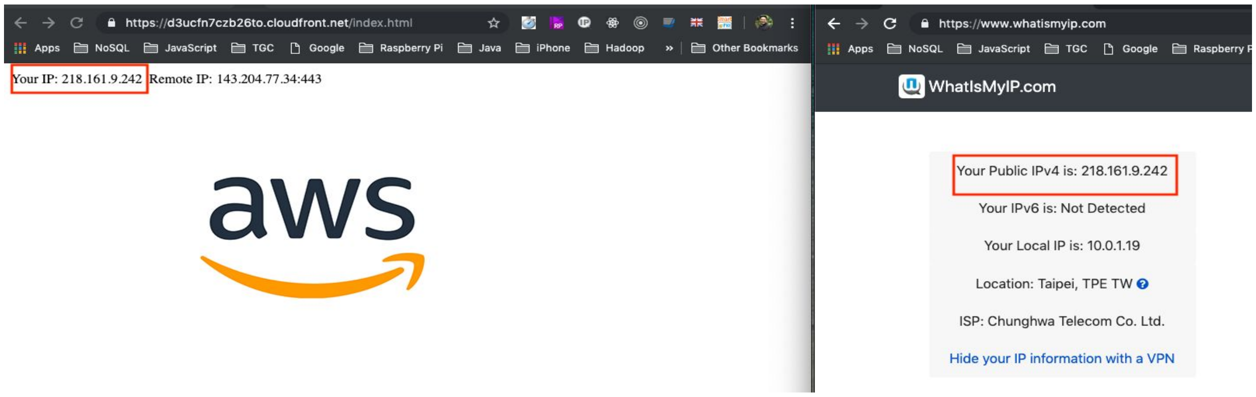
GET https://171nk3xfcb.execute-api.ap-northeast-1....

Header Name	Header Value
Authorization	allow
Add Header Name	Add Header Value

GET ...k3xfcb.execute-api.ap-northeast-1.amazonaws.com/prod/detectip 200 OK

Key or Index	Type Value
Root	6 items
input	{"path":"\/detectip","headers":{"CloudFront-Vie...
yourIp	218.161.9.242
origin	null
index	0
xForwardedFor	218.161.9.242, 52.46.57.101
remoteAddress	null

- Function Test
  - Case 1: Compare Application's IP with 3rd party IP service
    - Expected Result: IPs are the same.
    - Test Result: Pass



https://d3ucfn7czb26to.cloudfront.net/index.html

Your IP: 218.161.9.242 Remote IP: 143.204.77.34:443

https://www.whatismyip.com

WhatIsMyIP.com

Your Public IPv4 is: 218.161.9.242

Your IPv6 is: Not Detected

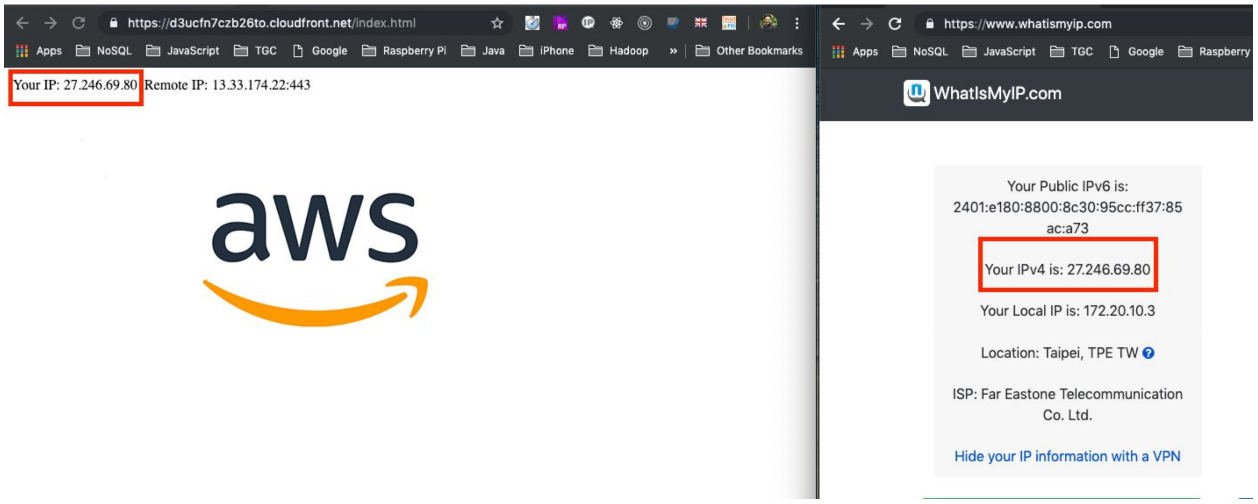
Your Local IP is: 10.0.1.19

Location: Taipei, TPE TW

ISP: Chunghwa Telecom Co. Ltd.

Hide your IP information with a VPN

- Case 2: Switch to 4G network and compare Application's IP with 3rd party IP service
  - Expected Result: IPs are the same.
  - Test Result: Pass



https://d3ucfn7czb26to.cloudfront.net/index.html

Your IP: 27.246.69.80 Remote IP: 13.33.174.22:443

https://www.whatismyip.com

WhatIsMyIP.com

Your Public IPv6 is: 2401:e180:8800:8c30:95cc:ff37:85ac:a73

Your IPv4 is: 27.246.69.80

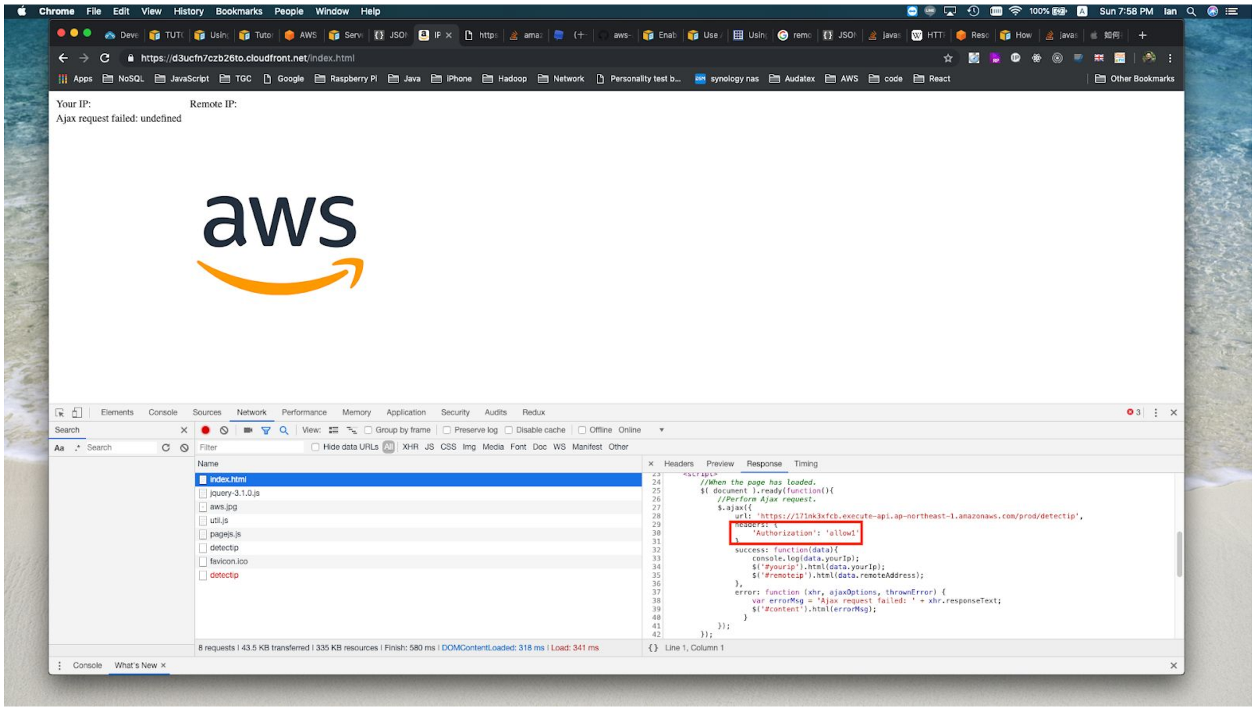
Your Local IP is: 172.20.10.3

Location: Taipei, TPE TW

ISP: Far Eastone Telecommunication Co. Ltd.

Hide your IP information with a VPN

- Case 3: Modify the Javascript of the Ajax call so that it sends invalid token
  - Expected Result: Not getting IP information
  - Test Result: Pass



https://d3ucfn7czb26to.cloudfront.net/index.html

Your IP: Remote IP:
Ajax request failed: undefined

```
<script>
    //When the page has loaded.
    $( document ).ready(function(){
        //Perform Ajax request.
        $.ajax({
            url: 'https://171nk3xfcb.execute-api.ap-northeast-1.amazonaws.com/prod/detectip',
            headers: {
                'Authorization': 'allow1'
            }
            success: function(data){
                console.log(data.yourIp);
                $('#yourip').html(data.yourIp);
                $('#remoteip').html(data.remoteAddress);
            },
            error: function (xhr, ajaxOptions, thrownError) {
                var errorMsg = 'Ajax request failed: ' + xhr.responseText;
                $('#content').html(errorMsg);
             }
        });
    });
```

- Known Issues
    - It is not possible to resolve the domain name to the same IP as a browser (Chrome) does. Even though we can do nslookup in a terminal from the same computer, there is no relevant Javascript capability to do so. Instead, I retrieve the origin header in the backend and apply InetAddress.getAllByName() as the alternative solution. However, the IP obtained on the server side probably won't be the same as what is seen in Chrome since the network topologies for server and browser are different.
    - My CloudFormation template only achieve the deployment of two Lambda functions and one API Gateway.
    - I fail to give API Gateway permission to invoke my Lambda function with CloudFormation template. It needs an additional manual step to make the API accessible through Internet.
    - None of the bonus questions are answered in this delivery.