Attachment authorization with AWS Lambda@Edge

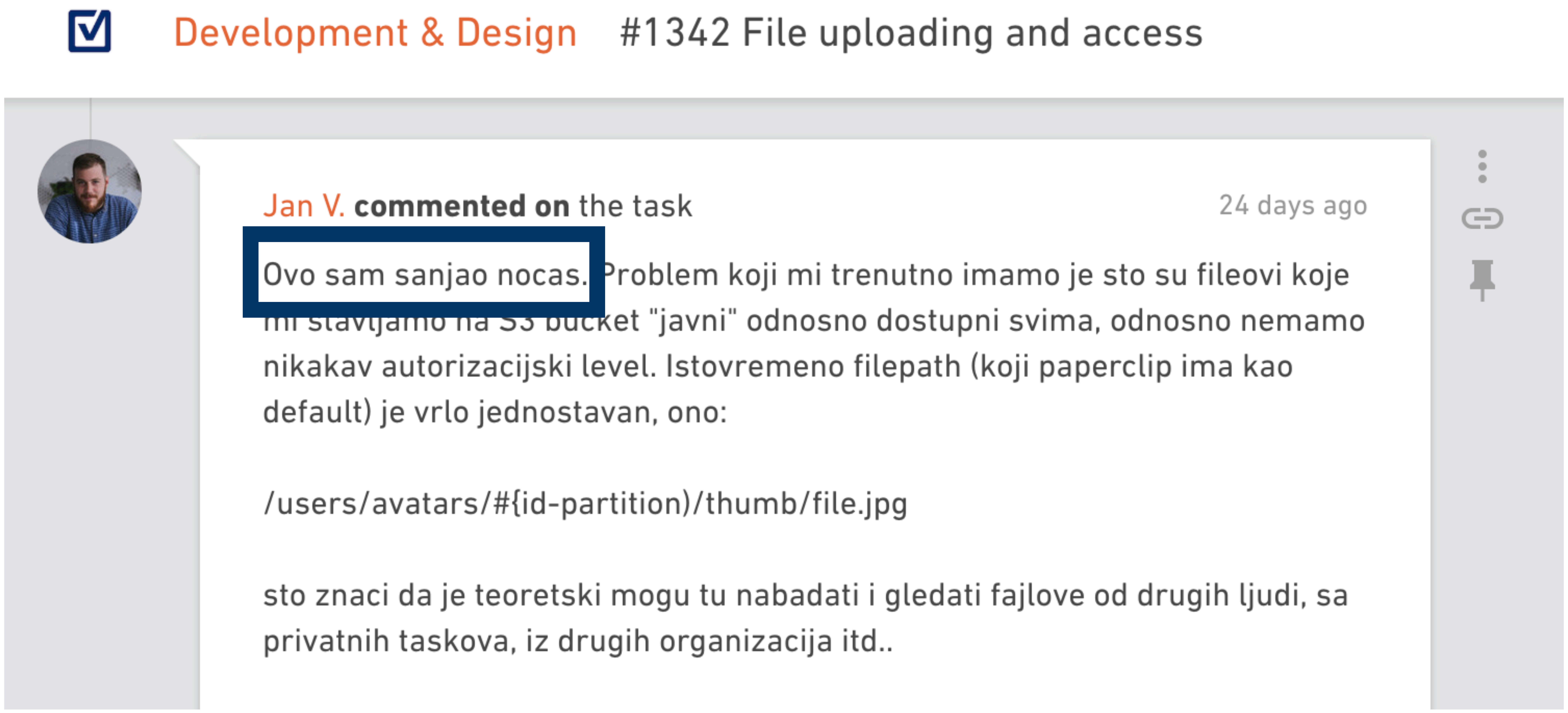
Hrvoje Matić
Ruby on Rails engineer

01

PROBLEM

01

MOTIVATION



Development & Design #1342 File uploading and access

Jan V. **commented on** the task

24 days ago

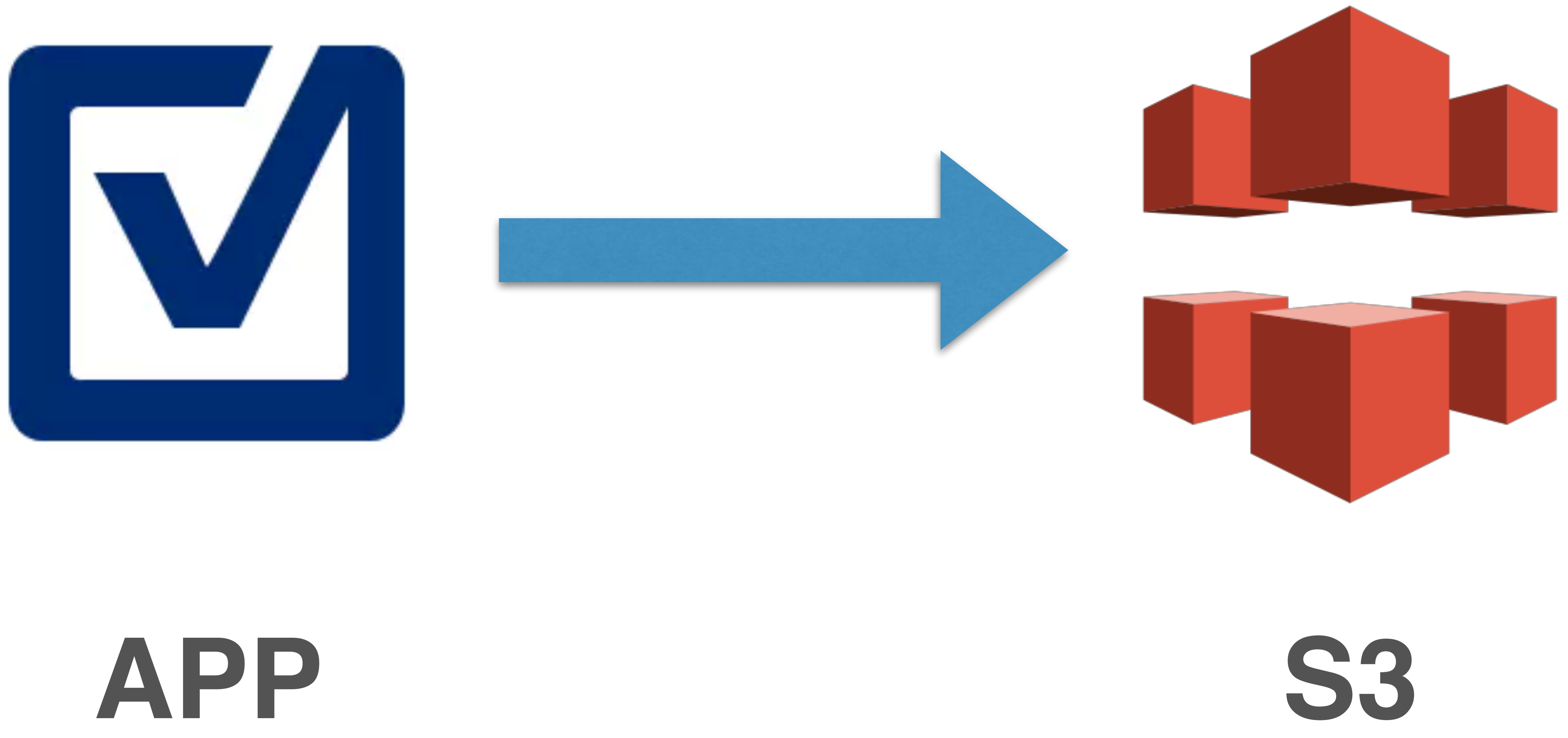
Ovo sam sanjao nocas. Problem koji mi trenutno imamo je sto su fileovi koje mi stavljamo na S3 bucket "javni" odnosno dostupni svima, odnosno nemamo nikakav autorizacijski level. Istovremeno filepath (koji paperclip ima kao default) je vrlo jednostavan, ono:

/users/avatars/#{id-partition)/thumb/file.jpg

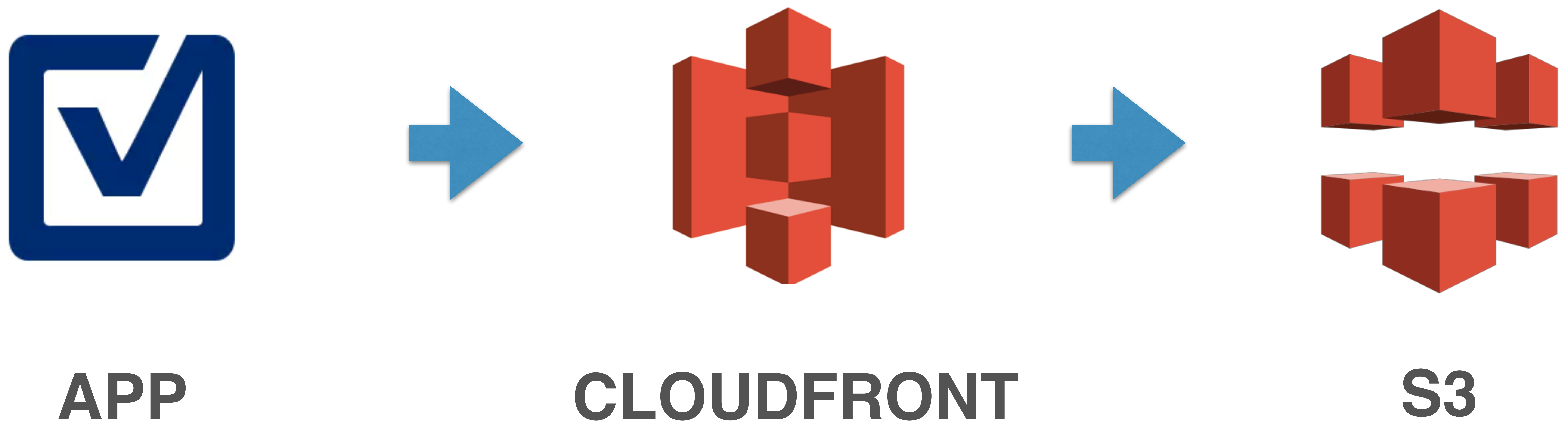
sto znaci da je teoretski mogu tu nabadati i gledati fajlove od drugih ljudi, sa privatnih taskova, iz drugih organizacija itd..

02

CURRENT STATE



https://s3.amazonaws.com/infinum.productive.staging/attachments/files/000/263/443/original/attachment.png



https://files.productive.io/attachments/files/000/263/443/original/attachment.png

03

GOALS

GOALS

- need to secure our attachments from unauthorised access
- users share attachment links, but attachments should be accessible only to Productive users
- user account is deactivated, it is no longer possible for him to see attachment
- we want to authorise only /attachments, we don’t need to authorise other assets because of performance
- unauthenticated access should redirect to login page which redirects back to requested resource after logging in

02

SOLUTION

01

IDEAS

INITIAL IDEAS

Randomize urls?

New method on attachments controller?

New microservice for attachments?

02

AWS LAMBDA

AWS LAMBDA

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. AWS Lambda executes your code only when needed and scales automatically, from a few requests per day to thousands per second.

PROS

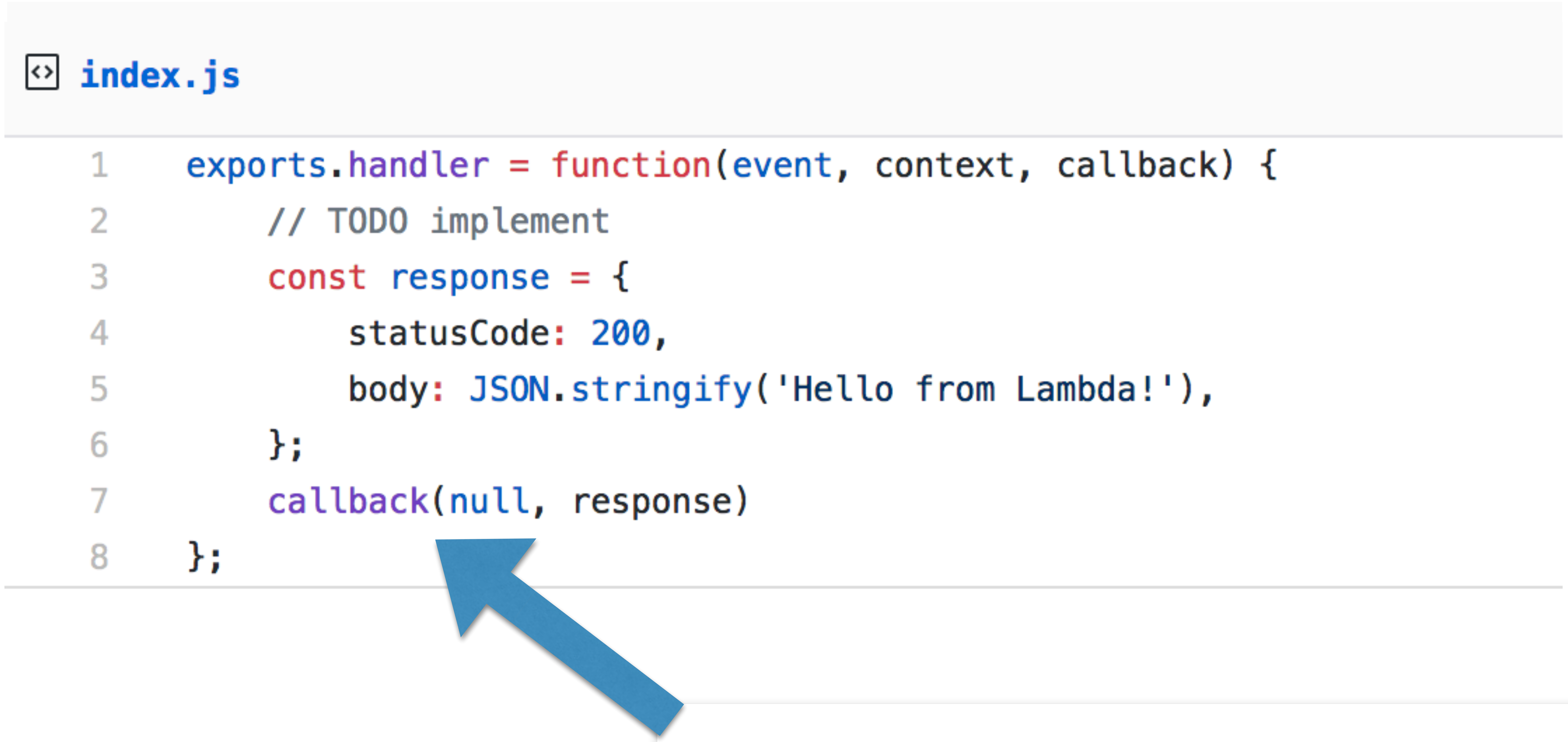
- no server management
- auto-scaling
- pay only what you use
- integration with other Amazon services

CONS

- not all languages are supported
- Lambda environment limitations
- does not support streams

Memory allocation	128 MB to 3008 MB
Timeout	15 minutes
/tmp storage	512 MB
Execution processes/threads	1024

HELLO FROM LAMBDA



index.js

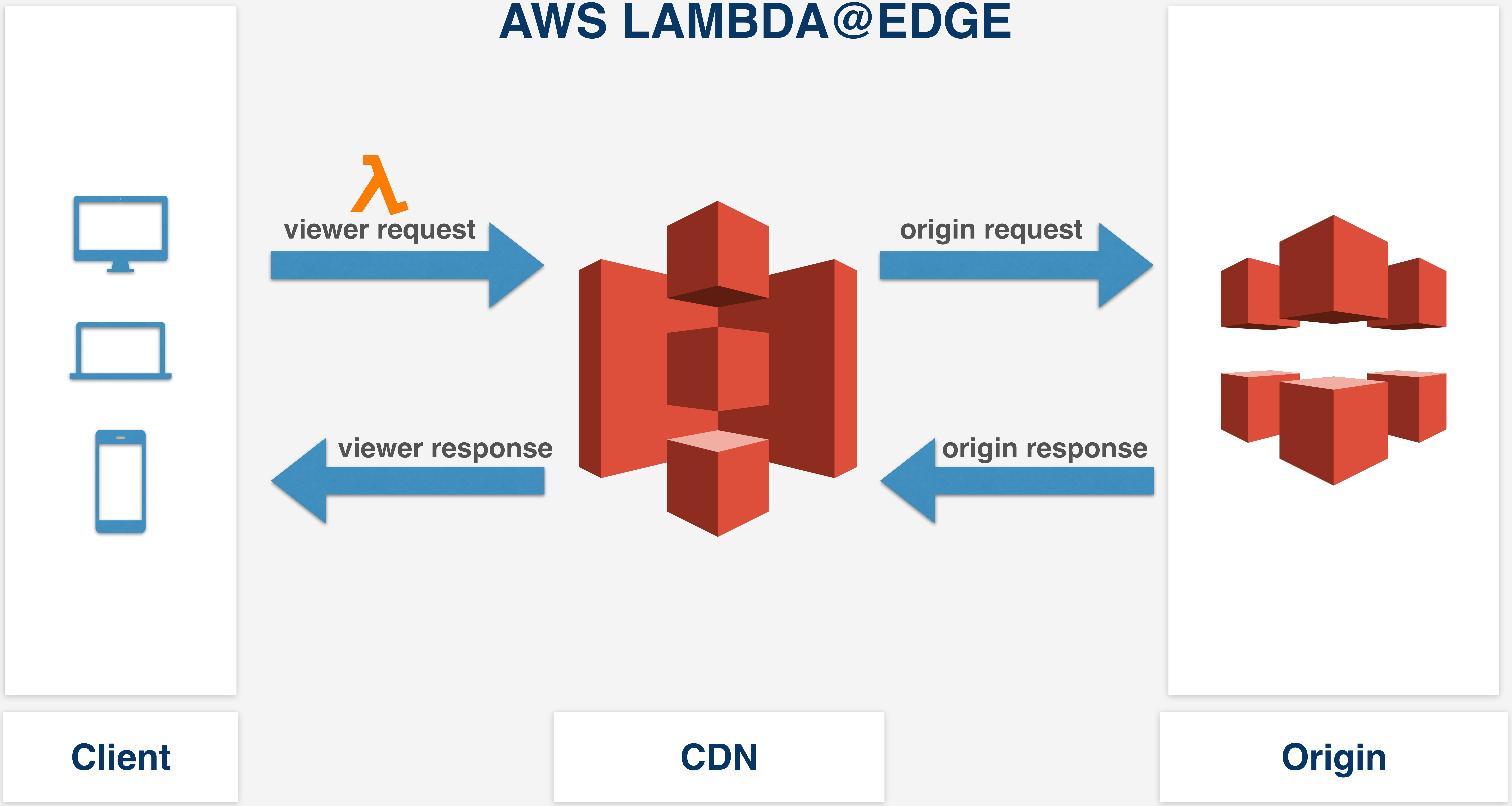
```
exports.handler = function(event, context, callback) {
    // TODO implement
    const response = {
        statusCode: 200,
        body: JSON.stringify('Hello from Lambda!'),
    };
    callback(null, response)
};
```

callback(Error error, Object result);

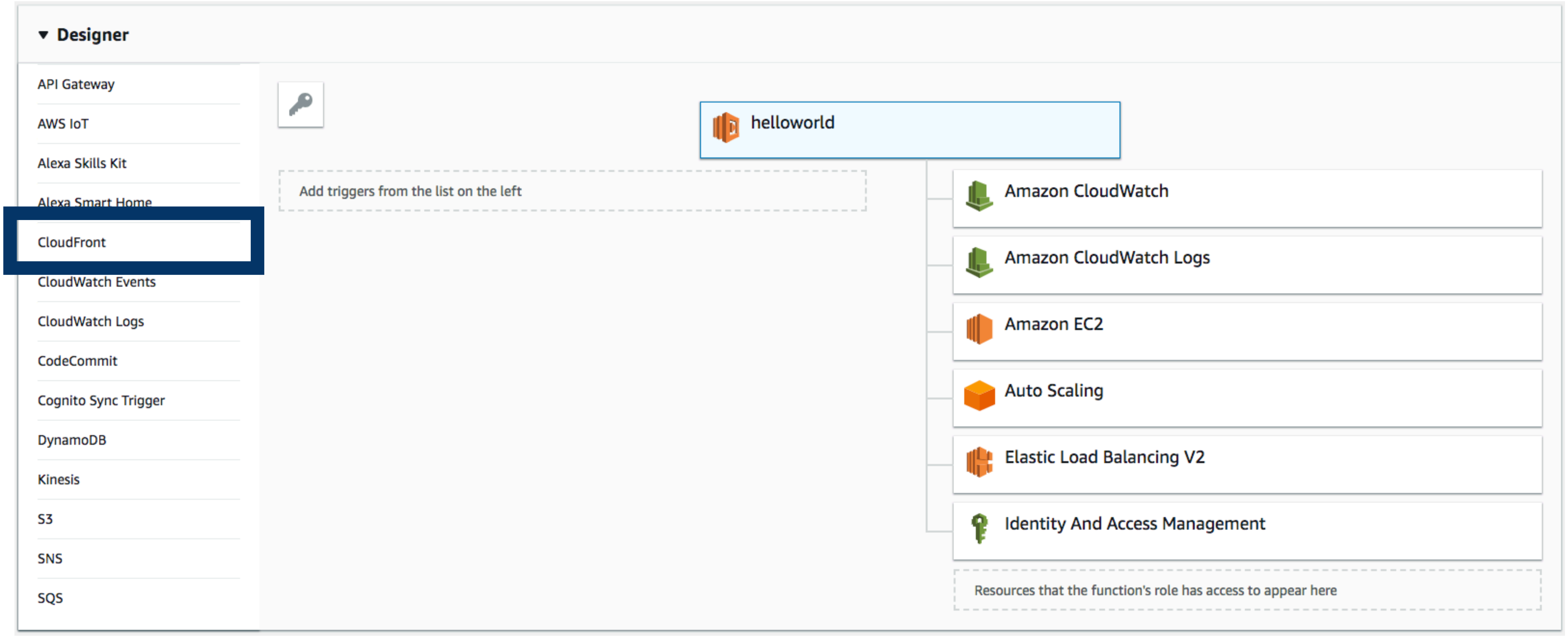
02

BACK TO OUR SOLUTION

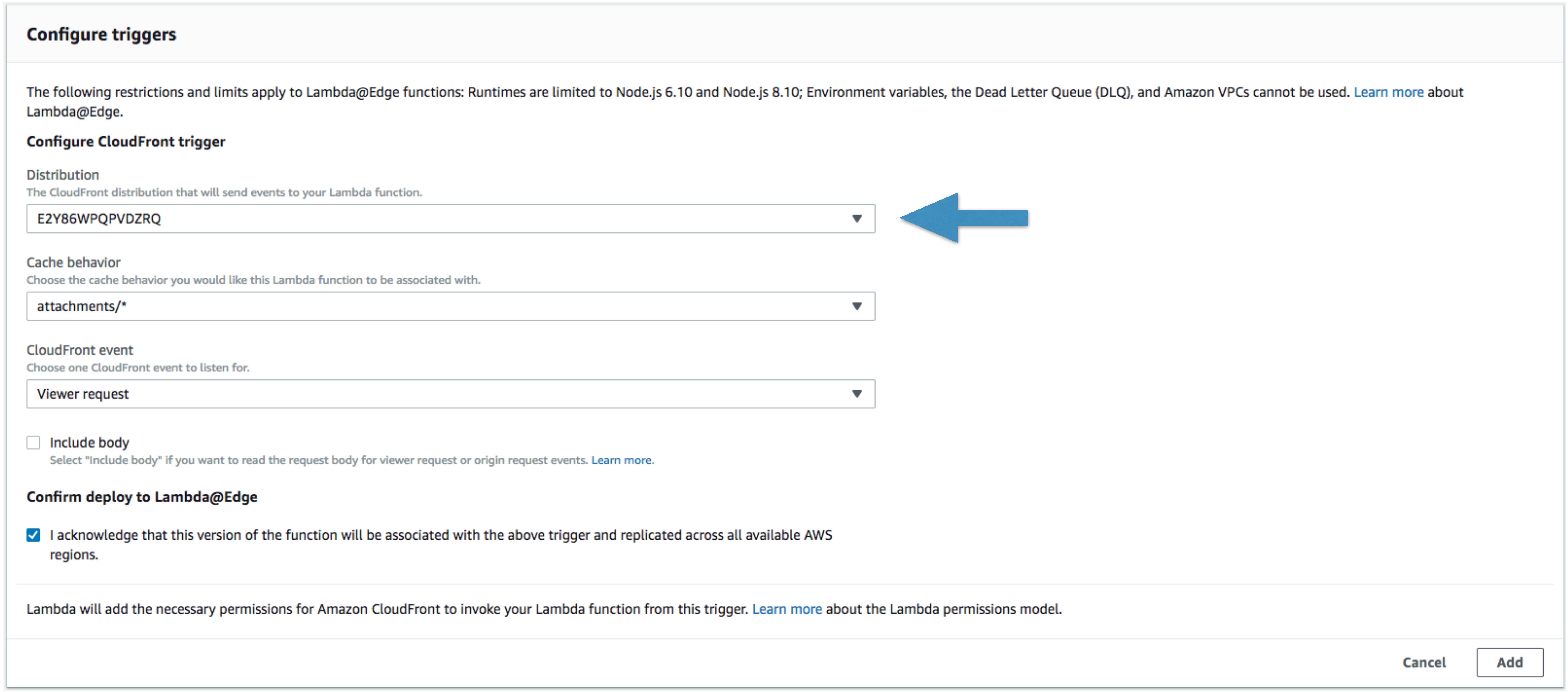
AWS LAMBDA@EDGE



LAMBDA TRIGGERS


▼ Designer

CLOUDFRONT TRIGGER SETUP



Configure triggers

The following restrictions and limits apply to Lambda@Edge functions: Runtimes are limited to Node.js 6.10 and Node.js 8.10; Environment variables, the Dead Letter Queue (DLQ), and Amazon VPCs cannot be used. Learn more about Lambda@Edge.

Configure CloudFront trigger

Distribution
The CloudFront distribution that will send events to your Lambda function.
E2Y86WPQPVDZRQ ▼

Cache behavior
Choose the cache behavior you would like this Lambda function to be associated with.
attachments/* ▼

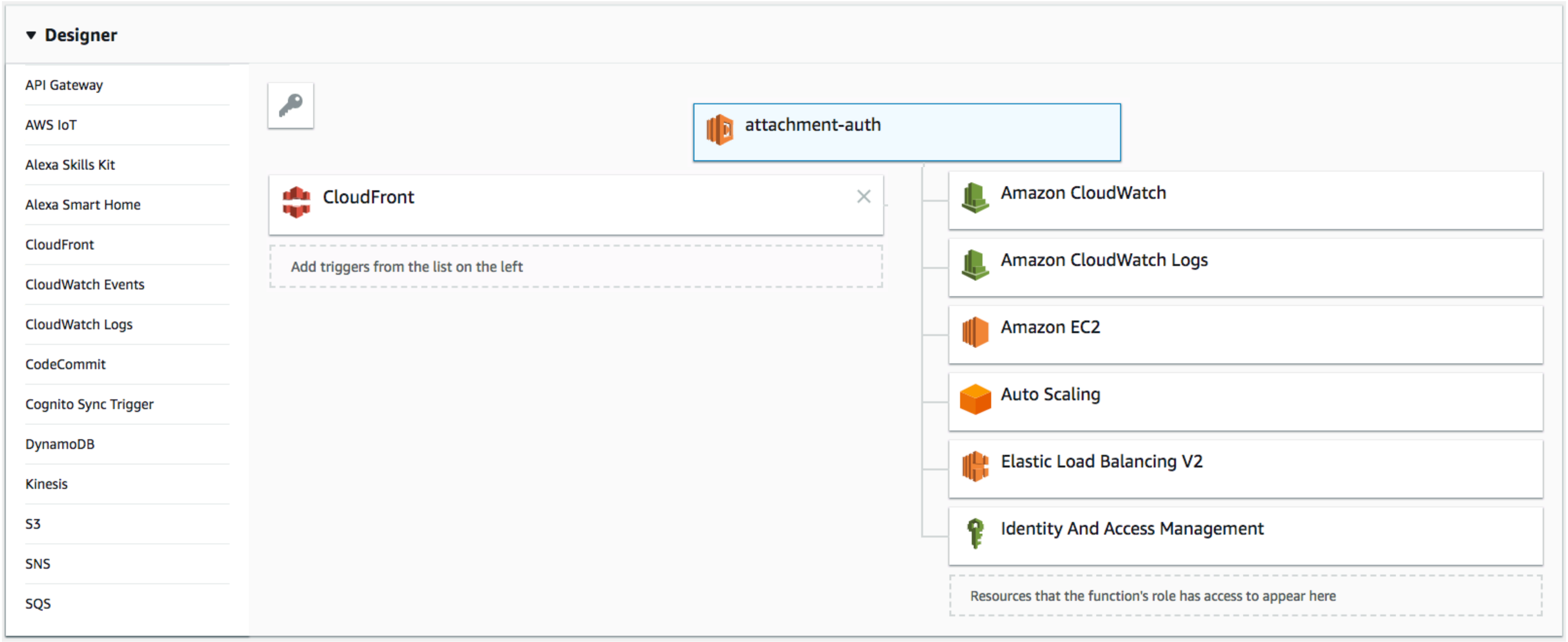
CloudFront event
Choose one CloudFront event to listen for.
Viewer request ▼

☐ Include body
Select "Include body" if you want to read the request body for viewer request or origin request events. Learn more.

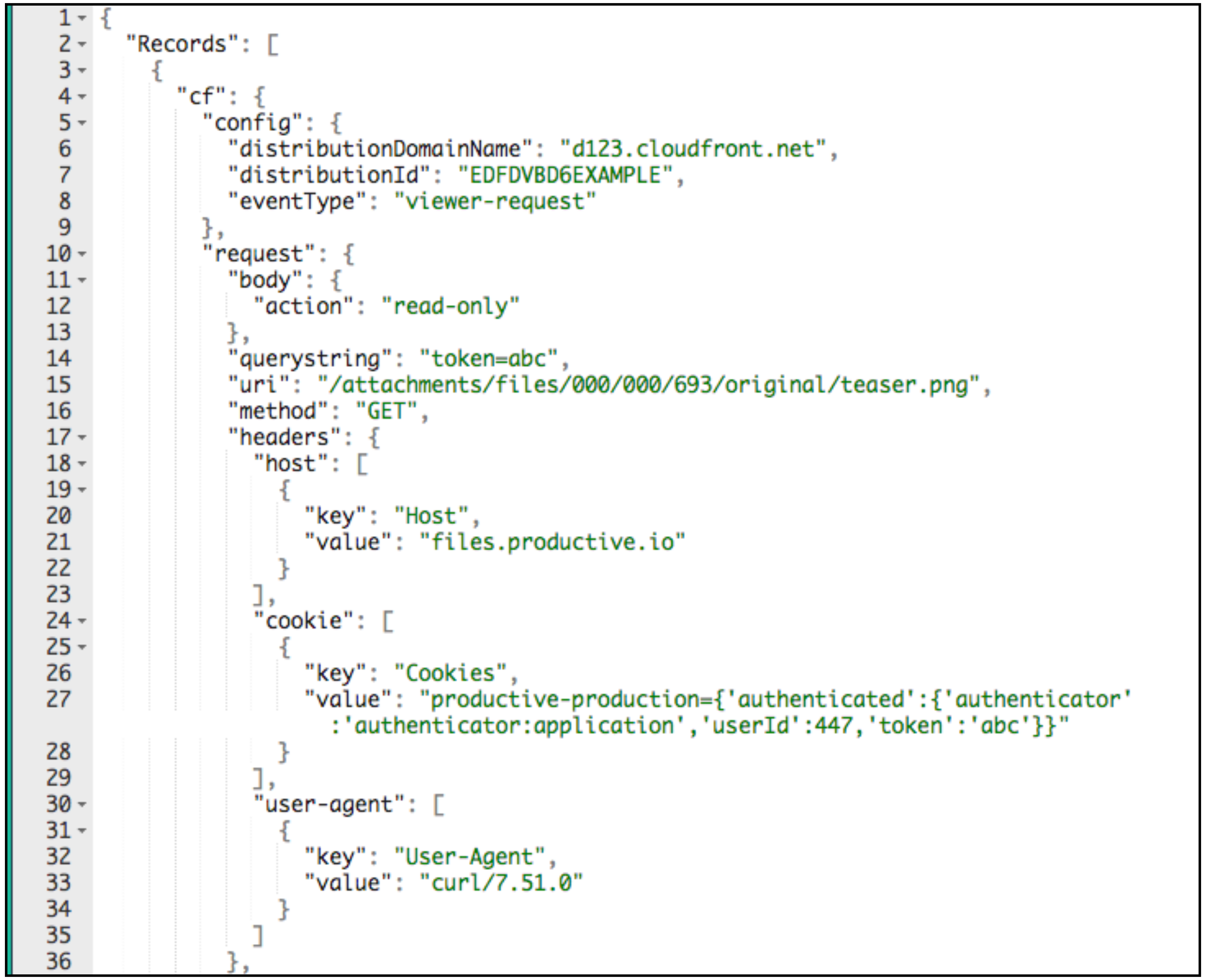
Confirm deploy to Lambda@Edge

☑ I acknowledge that this version of the function will be associated with the above trigger and replicated across all available AWS regions.

Lambda will add the necessary permissions for Amazon CloudFront to invoke your Lambda function from this trigger. Learn more about the Lambda permissions model.

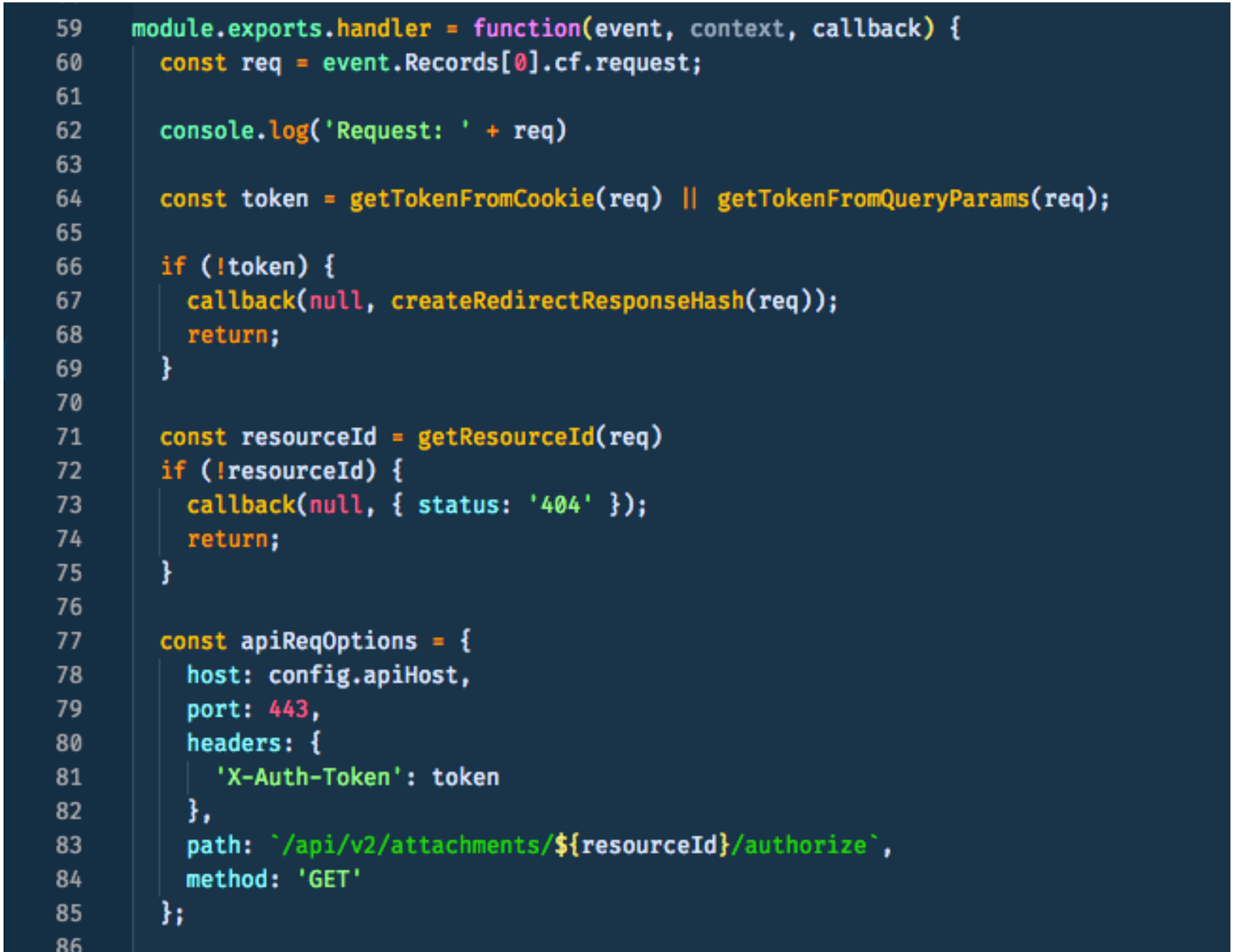

▼ Designer
API Gateway
AWS IoT
Alexa Skills Kit
Alexa Smart Home
CloudFront
CloudWatch Events
CloudWatch Logs
CodeCommit
Cognito Sync Trigger
DynamoDB
Kinesis
S3
SNS
SQS
attachment-auth
CloudFront
Add triggers from the list on the left
Amazon CloudWatch
Amazon CloudWatch Logs
Amazon EC2
Auto Scaling
Elastic Load Balancing V2
Identity And Access Management
Resources that the function's role has access to appear here

EVENT

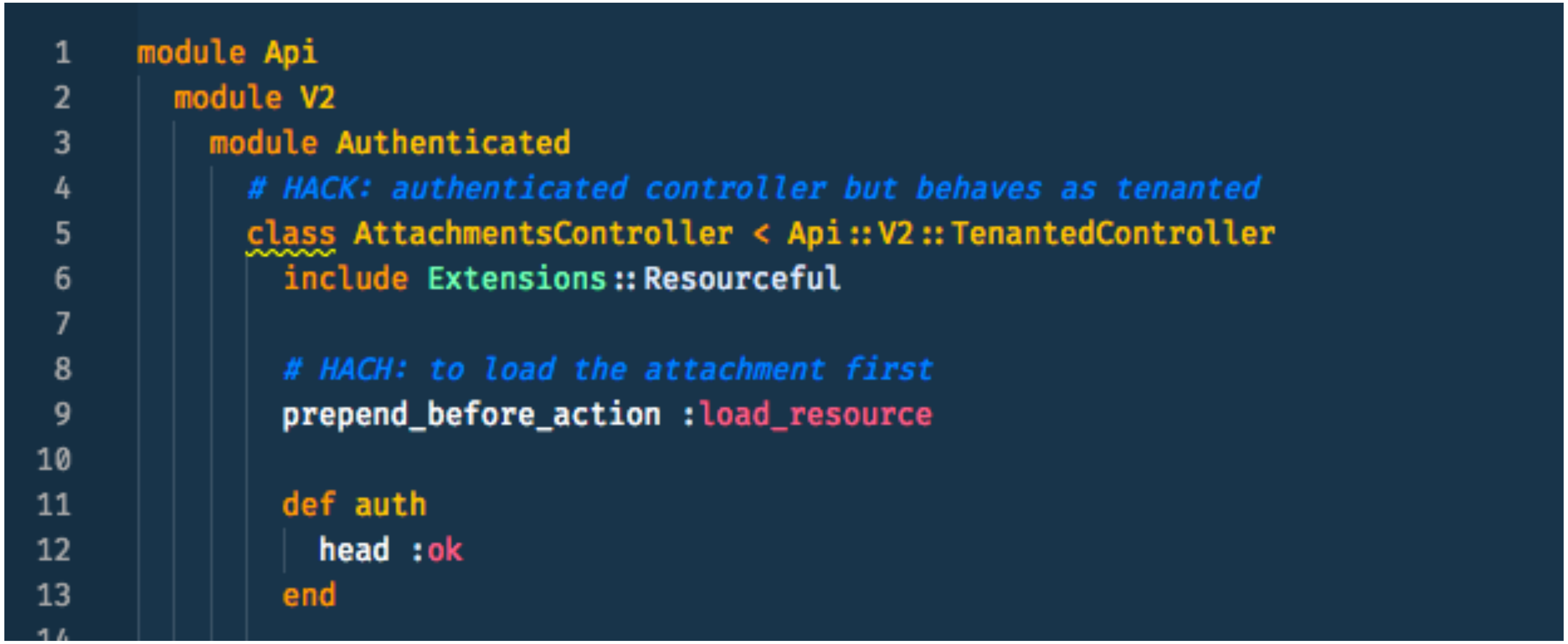


```
{
  "Records": [
    {
      "cf": {
        "config": {
          "distributionDomainName": "d123.cloudfront.net",
          "distributionId": "EDFDVBD6EXAMPLE",
          "eventType": "viewer-request"
        },
        "request": {
          "body": {
            "action": "read-only"
          },
          "querystring": "token=abc",
          "uri": "/attachments/files/000/000/693/original/teaser.png",
          "method": "GET",
          "headers": {
            "host": [
              {
                "key": "Host",
                "value": "files.productive.io"
              }
            ],
            "cookie": [
              {
                "key": "Cookies",
                "value": "productive-production={'authenticated':{'authenticator':'authenticator:application','userId':447,'token':'abc'}}"
              }
            ],
            "user-agent": [
              {
                "key": "User-Agent",
                "value": "curl/7.51.0"
              }
            ]
          },
```

https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/lambda-event-structure.html


```javascript
module.exports.handler = function(event, context, callback) {
  const req = event.Records[0].cf.request;

  console.log('Request: ' + req)

  const token = getTokenFromCookie(req) || getTokenFromQueryParams(req);

  if (!token) {
    callback(null, createRedirectResponseHash(req));
    return;
  }

  const resourceId = getResourceId(req)
  if (!resourceId) {
    callback(null, { status: '404' });
    return;
  }

  const apiReqOptions = {
    host: config.apiHost,
    port: 443,
    headers: {
      'X-Auth-Token': token
    },
    path: `/api/v2/attachments/${resourceId}/authorize`,
    method: 'GET'
  };
```

ATTACHMENTS CONTROLLER

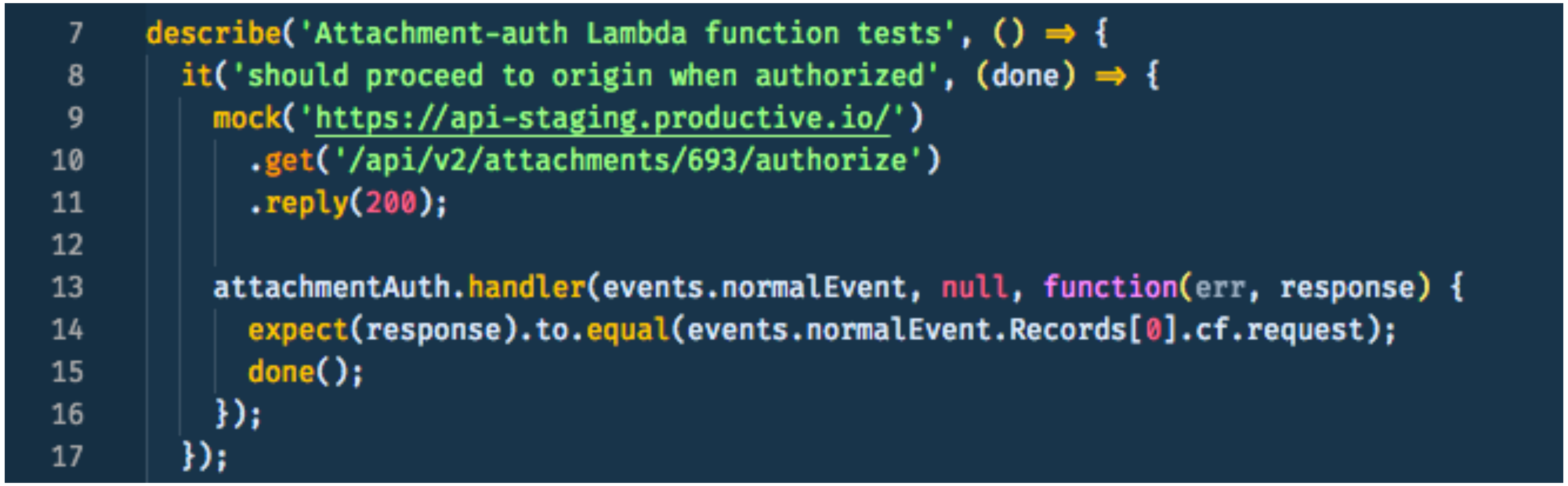


```ruby
module Api
  module V2
    module Authenticated
      # HACK: authenticated controller but behaves as tenanted
      class AttachmentsController < Api::V2::TenantedController
        include Extensions::Resourceful

        # HACH: to load the attachment first
        prepend_before_action :load_resource

        def auth
          head :ok
        end
```

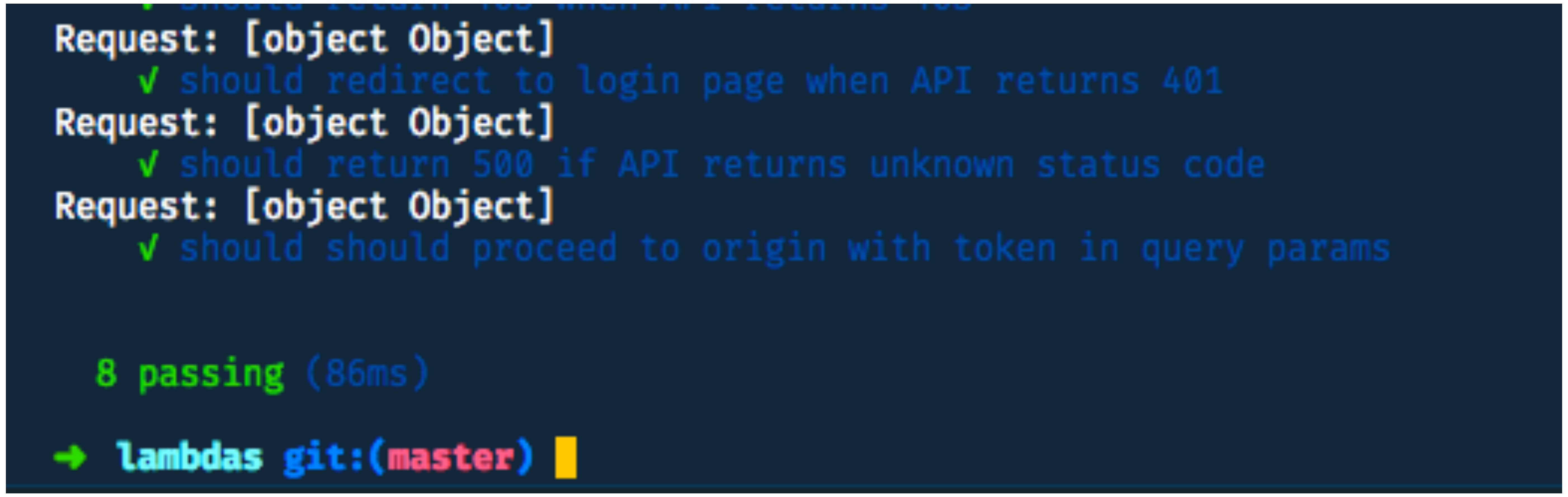
03

TESTING

UNIT TESTS



```javascript
describe('Attachment-auth Lambda function tests', () => {
  it('should proceed to origin when authorized', (done) => {
    mock('https://api-staging.productive.io/')
      .get('/api/v2/attachments/693/authorize')
      .reply(200);

    attachmentAuth.handler(events.normalEvent, null, function(err, response) {
      expect(response).to.equal(events.normalEvent.Records[0].cf.request);
      done();
    });
  });
```



```
Request: [object Object]
    √ should redirect to login page when API returns 401
Request: [object Object]
    √ should return 500 if API returns unknown status code
Request: [object Object]
    √ should should proceed to origin with token in query params


  8 passing (86ms)

➜ lambdas git:(master) █
```

04

CI / CD

USING AWS CLI

```
aws lambda update-function-code --function-name attachment_auth
--zip-file fileb://attachment_auth/index.zip
```

```
aws lambda publish-version --function-name attachment_auth
```

```
aws lambda update-alias --function-name attachment_auth
--name STAGING --version {verzija}
```

OR EVEN BETTER...

serverless

05

OTHER POSSIBLE USECASES

OTHER USECASES:

- processing uploaded s3 objects
- automated backups and every day tasks
- running jobs with CloudWatch Events
- log analysis
- serverless apps
- imagination is limit

06

QUESTIONS?

Thank you!

hrvoje.matic@productive.io
@hmatic

Find me on social media:

hmatic hmatic