SECURING STATIC S3 WEBSITE WITH OAC AND CREATING A CI/CD PIPELINE

Mike Macdonald

CONTENTS

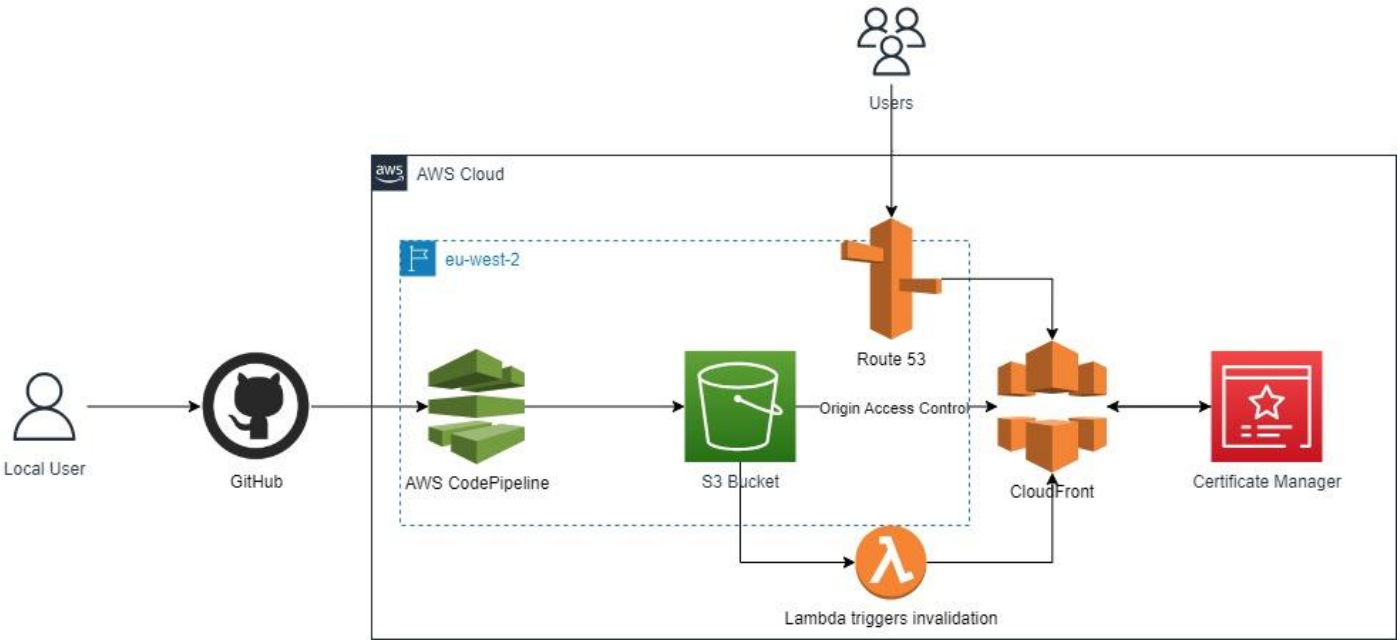
1. BRIEF

Part 1 - Previously I have created a static website using S3, a custom domain name (mike-macdonald.co.uk) and distributed using AWS CloudFront. However, the origin of the CloudFront distribution uses the website endpoint of the S3 bucket meaning that the S3 bucket needs public access for CloudFront to be able to distribute the content. Whilst this method works, from a security point of view it is not ideal to have public access granted (albeit read only) to a bucket if it is not directly required. For security, I am going to change the set-up of the CloudFront distribution to use Origin Access Control (OAC). This is a secure way for CloudFront to access S3 origins.

Part 2 – Currently whenever I want to update the website I have to edit the website locally then either manually upload the updated file or use the CLI. I want to create a CI/CD process using AWS CodePipeline and GitHub so that changes can be pushed automatically to S3.

Part 3 – CloudFront works by caching data at edge locations. This can mean that it serves out of date content if the content has been updated but the cache hasn't refreshed. I have created a lambda function that creates an invalidation on the CloudFront distribution each time a file called index.html is uploaded to the S3 bucket. This ensures CloudFront will always be serving the most up to date website.

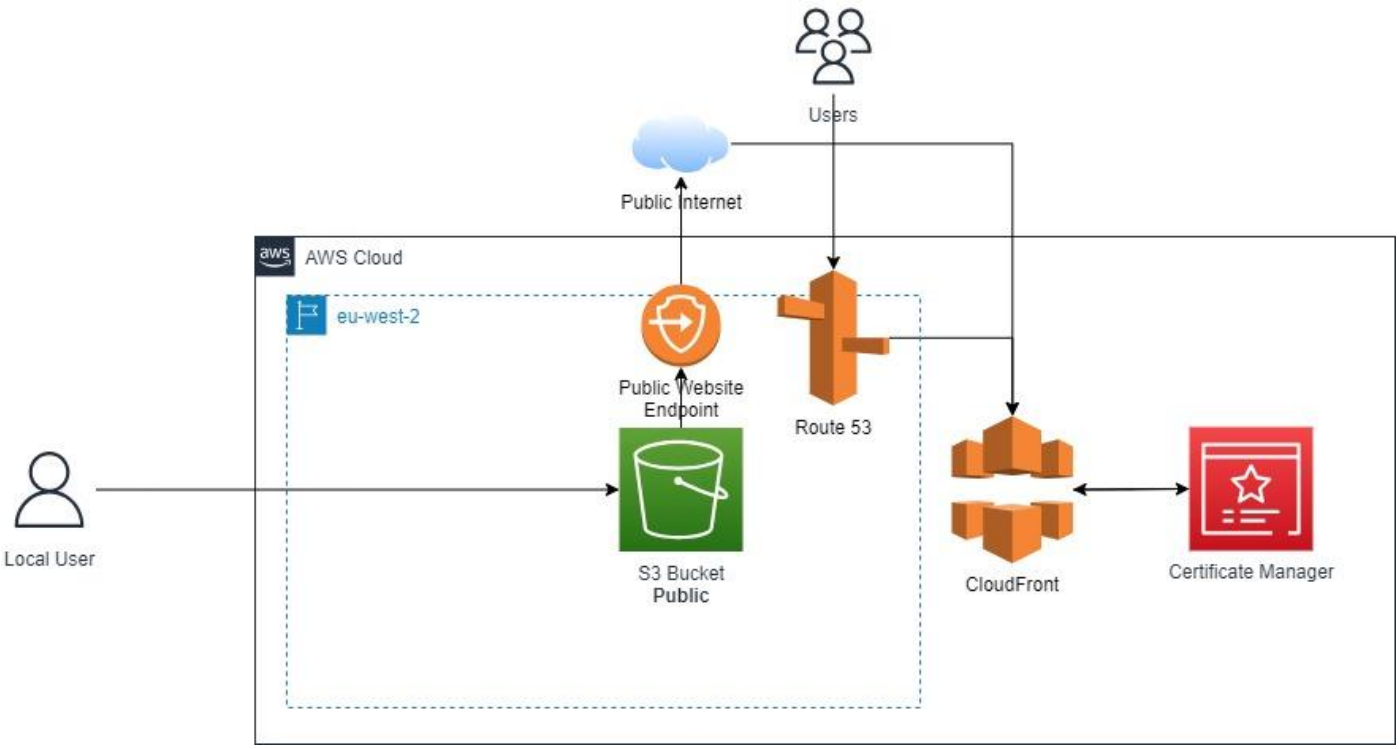
Final architecture:



2. ORIGIN ACCESS CONTROL (OAC)

2.1 CURRENT SETUP

With the current setup Cloudfront uses the public website endpoint of the S3 bucket. This means that cloudformation is accessing the bucket through the public internet.



Currently the bucket that hosts the website is publically accessible.

○	mike-macdonald.co.uk	EU (London) eu-west-2	⚠ Public	May 26, 2023, 16:37:58 (UTC+01:00)

Public access is enabled however for security you still need a bucket policy to decide which actions can be done on the bucket. There is a bucket policy on this bucket just to allow anyone to get any of the objects in the bucket. This prevents any unauthorised users adding or removing objects from the bucket.

Bucket policy

The bucket policy, written in JSON, provides access to the objects stored in the bucket. Bucket policies don't apply to objects owned by other accounts. Learn more

Edit | Delete

Copy

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::mike-macdonald.co.uk/*"
    }
  ]
}
```

Within CloudFront I have a single origin of the website endpoint of the S3 bucket.

Origins

Filter origins by property or value

Origin name	Origin domain	Origin path	Origin type
mike-macdonald.co.uk	mike-macdonald.co.uk.s3-…		S3 static website

Settings

Origin domain

Choose an AWS origin, or enter your origin's domain name.

mike-macdonald.co.uk.s3-website.eu-west-2.amazonaws.com

2.2 CREATING OAC

Origin Access is created under the security heading within CloudFront.

Details

Name

Website-Origin-Access

The name must be unique. Valid characters: letters, numbers and most special characters. Use up to 64 characters.

Description - *optional*

Origin Access Control for mike-macdonald.co.uk website

The description can have up to 256 characters.

Settings Info

Signing behavior

- ○ Do not sign requests
- ◉ Sign requests (recommended)
  - ☐ Do not override authorization header
    Do not sign if incoming request has authorization header.

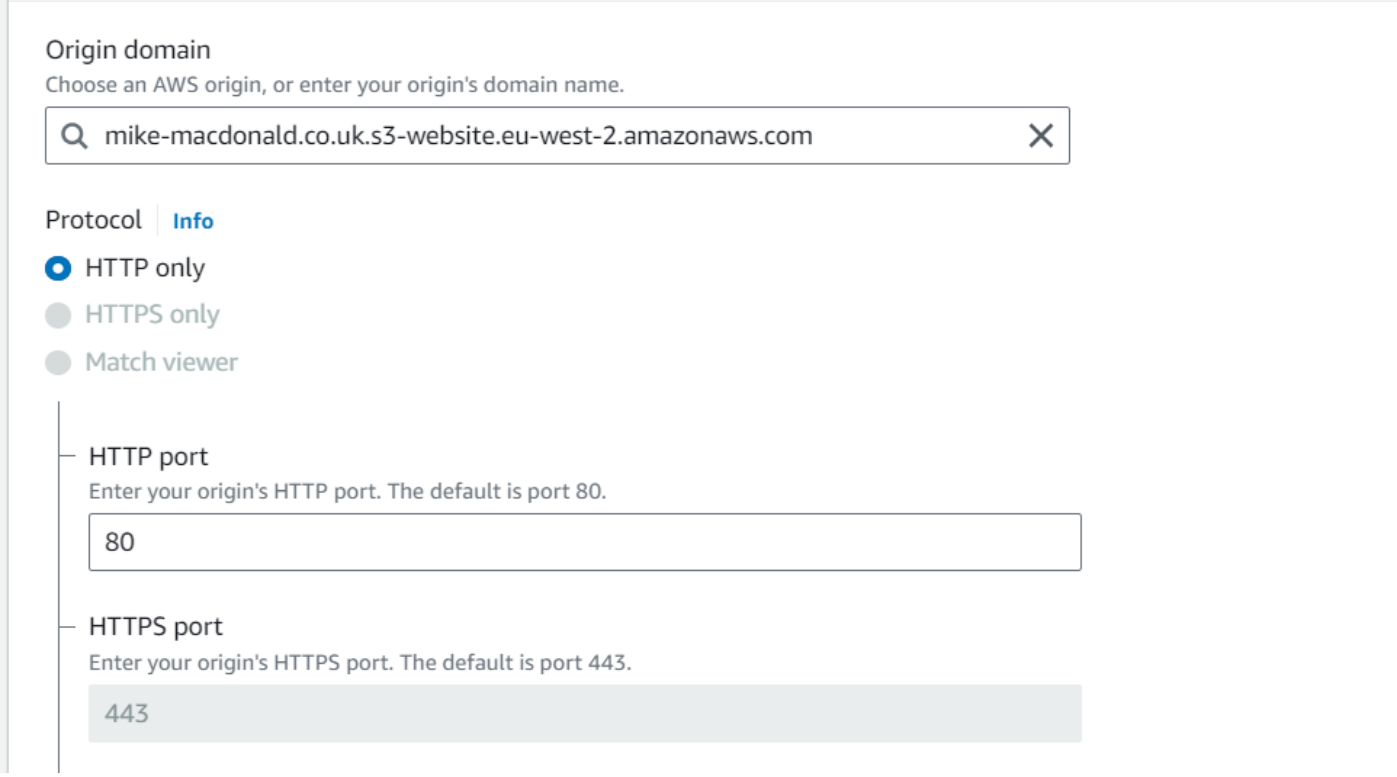
Origin type

S3

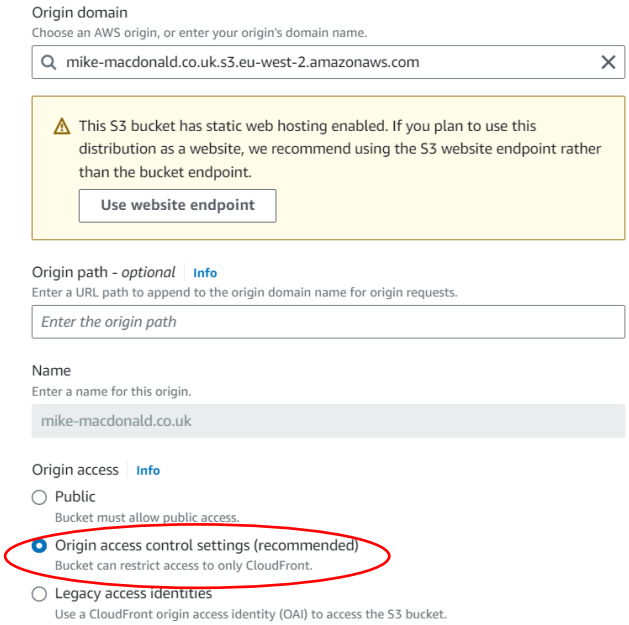
The origin type must be the same type as origin domain.

2.3 ADDING OAC TO CLOUDFRONT DISTRIBUTION

When original set up, the S3 bucket was created to host a static website. When setting up the origin in CloudFront you have the choice to use the *Website Endpoint* instead of the *Bucket Endpoint*. This makes the CloudFront distribution see the bucket only as a website and not as a bucket. Using this method, **OAC cannot be set up**. There is no option below to set up Origin Access Control.



To rectify the situation it's required to change the origin domain from the website endpoint and put it back to the original bucket endpoint. This then does give the option for Origin Access Control.



Here we can then choose the OAC that we created previously.

Origin access Info

- ○ Public
  Bucket must allow public access.
- ● Origin access control settings (recommended)
  Bucket can restrict access to only CloudFront.
- ○ Legacy access identities
  Use a CloudFront origin access identity (OAI) to access the S3 bucket.

Origin access control
Select an existing origin access control (recommended) or create a new configuration.

Website-Origin-Access — Origin type: S3
Origin Access Control for mike-macdonald.co.uk website

Create control setting

Bucket policy
Policy must allow access to CloudFront IAM service principal role.

I will manually update the policy

You must allow access to CloudFront using this policy statement. Learn more about giving CloudFront permission to access the S3 bucket.

Copy policy

Go to S3 bucket permissions

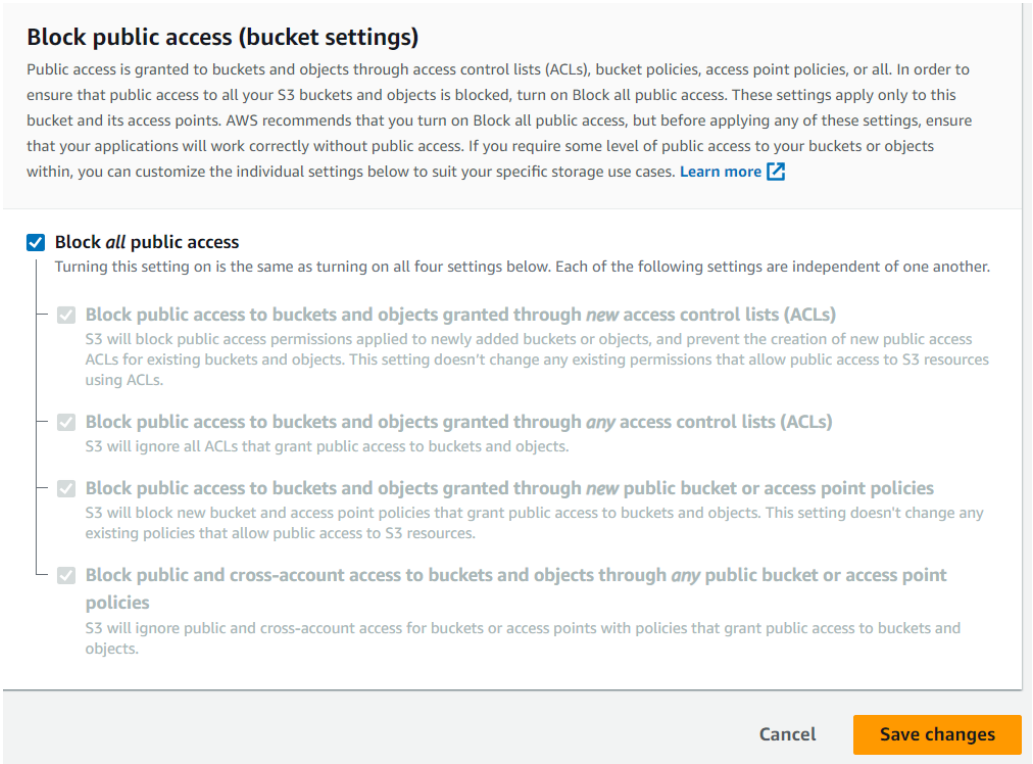
Do notice that using this method you have to change the S3 bucket policy to allow CloudFront permission to access the S3 bucket. Fortunately, it gives an example policy to use that can be copied into the S3 bucket policy directly. (Source ARN anonymised for security).

Policy

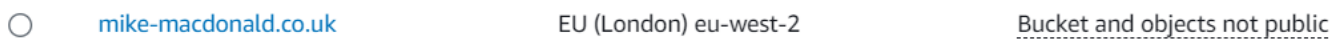
```
1
2  {
3      "Version": "2008-10-17",
4      "Id": "PolicyForCloudFrontPrivateContent",
5      "Statement": [
6          {
7              "Sid": "AllowCloudFrontServicePrincipal",
8              "Effect": "Allow",
9              "Principal": {
10                 "Service": "cloudfront.amazonaws.com"
11             },
12             "Action": "s3:GetObject",
13             "Resource": "arn:aws:s3:::mike-macdonald.co.uk/*",
14             "Condition": {
15                 "StringEquals": {
16                     "AWS:SourceArn": "arn:aws:cloudfront::[redacted]
17                 }
18             }
19         }
20     ]
21 }
```

2.4 BLOCKING PUBLIC ACCESS

After having changed the bucket policy you can block public access within the permissions of the S3 bucket.

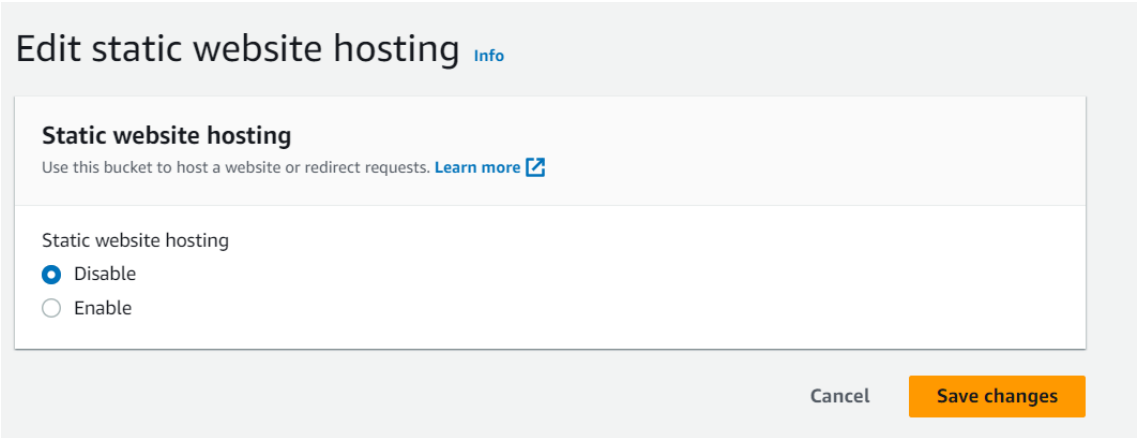


This has now changed the S3 bucket from being publically accessible being more secure.



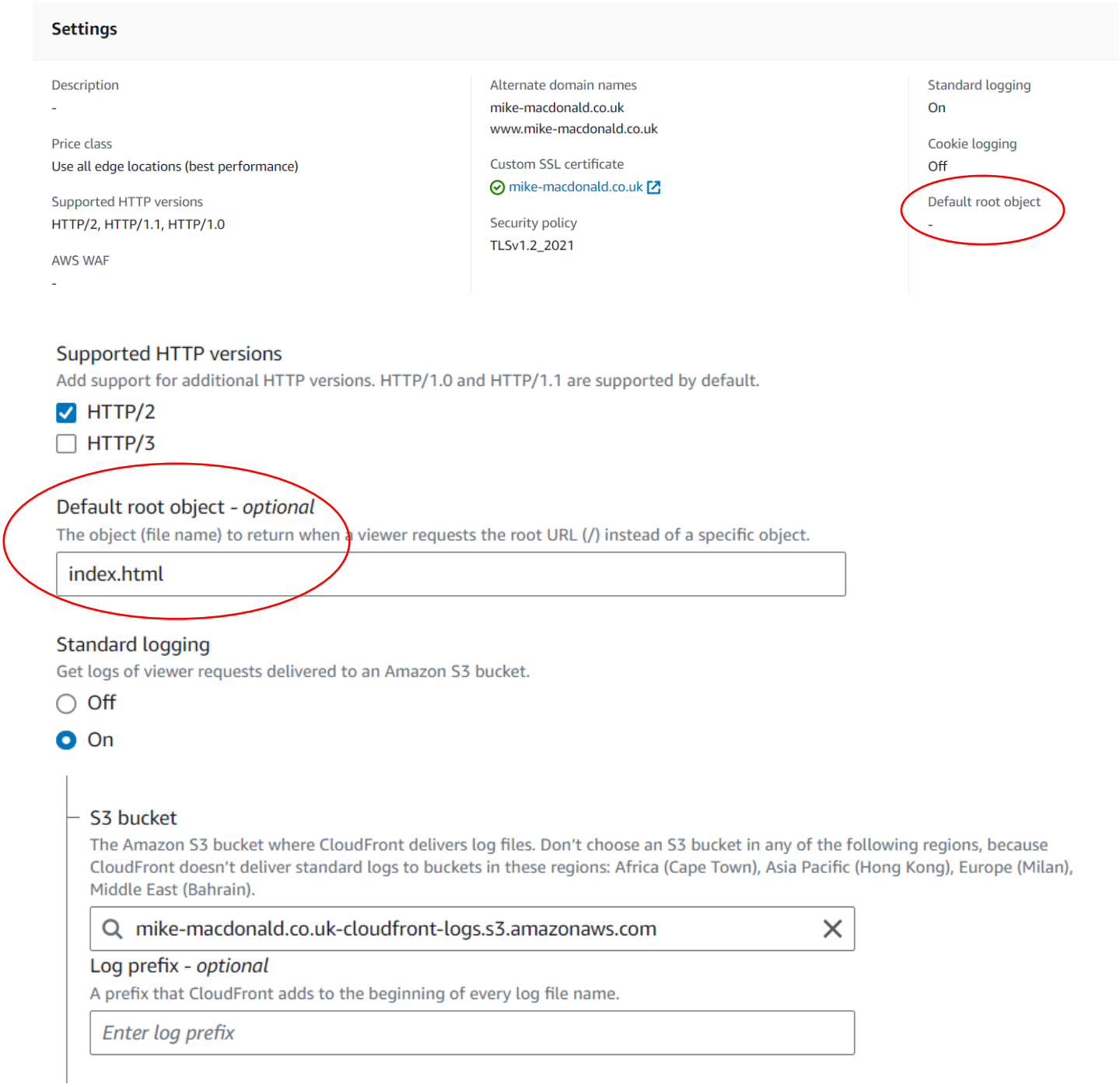
2.5 DISABLE STATIC WEBSITE HOSTING

Within the properties of the S3 bucket we need to change Static Website Hosting to **Disable**.



2.6 ADDING INDEX.HTML AS THE DEFAULT ROOT OBJECT

When you set up a static website using S3 you have to define which file you want to use as your root object (home page). When moving over to using OAC this needs doing but in the settings of the CloudFront distribution rather than within the S3 bucket. My home page is named "index.html" so this is set as the default root object.



The website is now no longer publically accessible through S3 and is only available through the CloudFront distribution, adding more security but still providing high-speed delivery.

3. CREATING A CI/CD PIPELINE WITH GITHUB

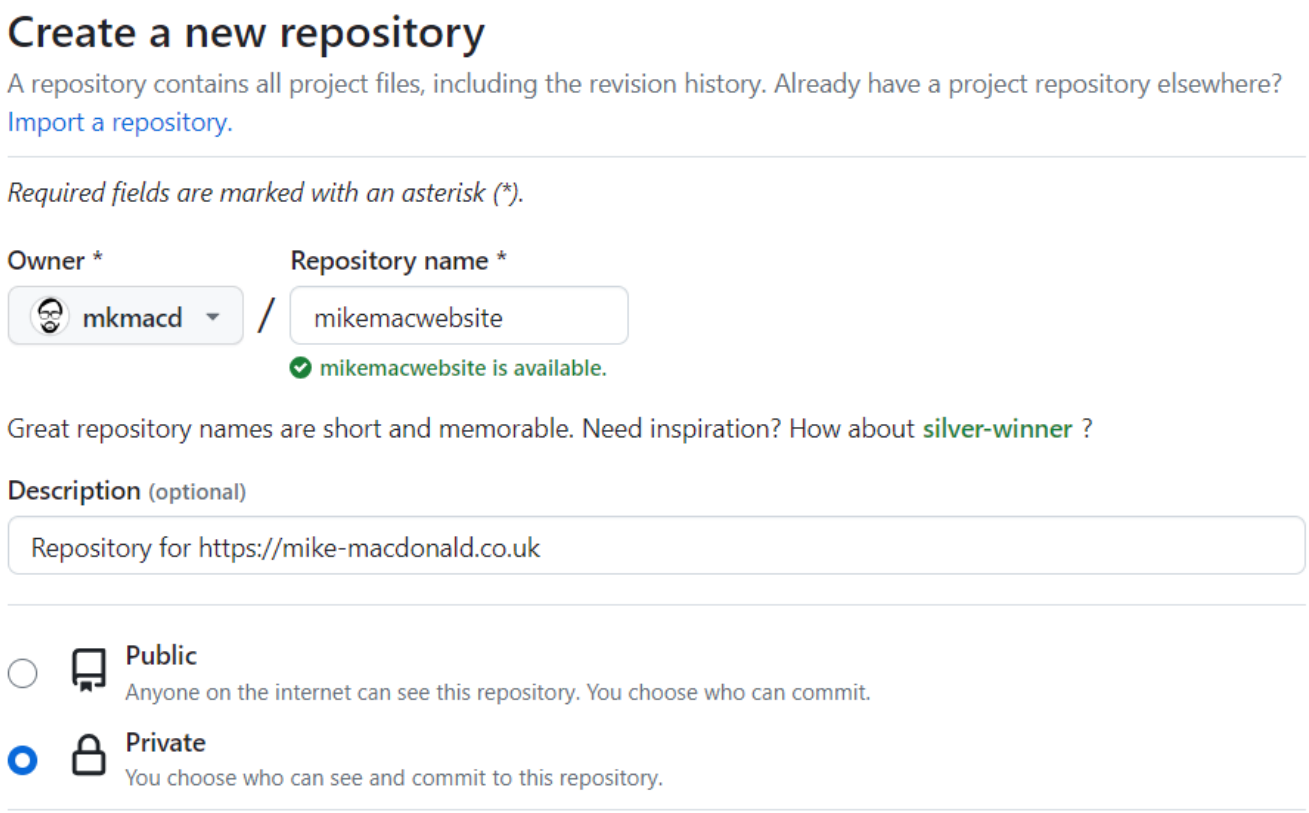
3.1 OVERVIEW

Currently every time I want to update the website, I have to manually go to my S3 bucket, overwrite the index.html file, and add any other files (eg images) to the various folders. This can also be done more efficiently via the CLI, utilising the access key and secret access key. However I wanted to expand my CI/CD skills and so decided to use GitHub in conjunction with AWS CodePipeline.

CI/CD is Continuous Integration/Continuous Deployment. Continuous Integration allows for automated integration of code changes from multiple developers into one repository. Continuous Deployment automatically deploys the validated and tested code changes to production environments. Whilst this isn't technically CI/CD in its truest form because it's not triggering builds or running tests, it is deploying code and using the principals of committing changes, pushing to a repository and having CodePipeline pull from the remote repository and update the S3 file has strong similarities.

3.2 CREATING A GITHUB REPOSITORY

I am going to use GitHub as a remote repository that I will connect with AWS CodePipeline. The first stage of this is to create a GitHub repository.



3.3 INITIALISNG GIT AND CREATING REMOTE WITHIN GITHUB REPOSITORY

Git is already installed locally and I have already set up SSH keys to allow access to my GitHub account. I need to initialise git in the folder where my files are stored using the command *git init*.

```
PS C:\Users\mike.macdonald\Documents\Website> git init
Initialized empty Git repository in C:/Users/mike.macdonald/Documents/Website/.git/
PS C:\Users\mike.macdonald\Documents\Website>
```

Then I rename the default brain to "main" using the command *git branch –M main* (this has no output)

To add a remote within the GitHub repository I use the command *git remote add origin git@github.com:mkmacd/mikemacwebsite* (This also has no output)

3.4 PUSHING FILES TO THE REPOSITORY

Using the command *git status* you can see the files and folders that aren't currently being tracked

```
PS C:\Users\mike.macdonald\Documents\Mike-Macdonald.co.uk Website> git branch -M main
PS C:\Users\mike.macdonald\Documents\Mike-Macdonald.co.uk Website> git status
On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        Mike Macdonald CV.pdf
        README.md
        assets/
        css/
        httpapi.htm
        index.html
        js/
        peering.htm

nothing added to commit but untracked files present (use "git add" to track)
PS C:\Users\mike.macdonald\Documents\Mike-Macdonald.co.uk Website>
```

Using the command *git add .* (including the period) will add all of these files to the tracked list.

```
PS C:\Users\mike.macdonald\Documents\Website> git add .
warning: in the working copy of 'js/scripts.js', LF will be replaced by CRLF the next time Git touches it
PS C:\Users\mike.macdonald\Documents\Website>
```

Running *git status* again shows the tracked files.

```
PS C:\Users\mike.macdonald\Documents\Website> git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   README.md
        new file:   assets/img/DynamoDB.png
        new file:   assets/img/Favicon.ico
        new file:   assets/img/amazon-api-gateway.png
        new file:   assets/img/amazon-rds.png
        new file:   assets/img/aws-cloud-practitioner.png
        new file:   assets/img/aws-cloudquest-cloud-practitioner.png
        new file:   assets/img/aws-cloudquest-solutions-architect.png
        new file:   assets/img/aws-saa.png
        new file:   assets/img/camping.jpg
        new file:   assets/img/corfu.jpg
        new file:   assets/img/dance.jpg
        new file:   assets/img/guitar.jpg
        new file:   assets/img/northface.jpg
        new file:   assets/img/profile.JPG
        new file:   assets/img/qts.jpeg
        new file:   assets/img/skiing.JPG
        new file:   assets/img/twominutetuesdays.png
        new file:   assets/img/vpcpeering.png
        new file:   css/styles.css
        new file:   httpapi.htm
        new file:   index.html
        new file:   js/scripts.js
        new file:   peering.htm
```

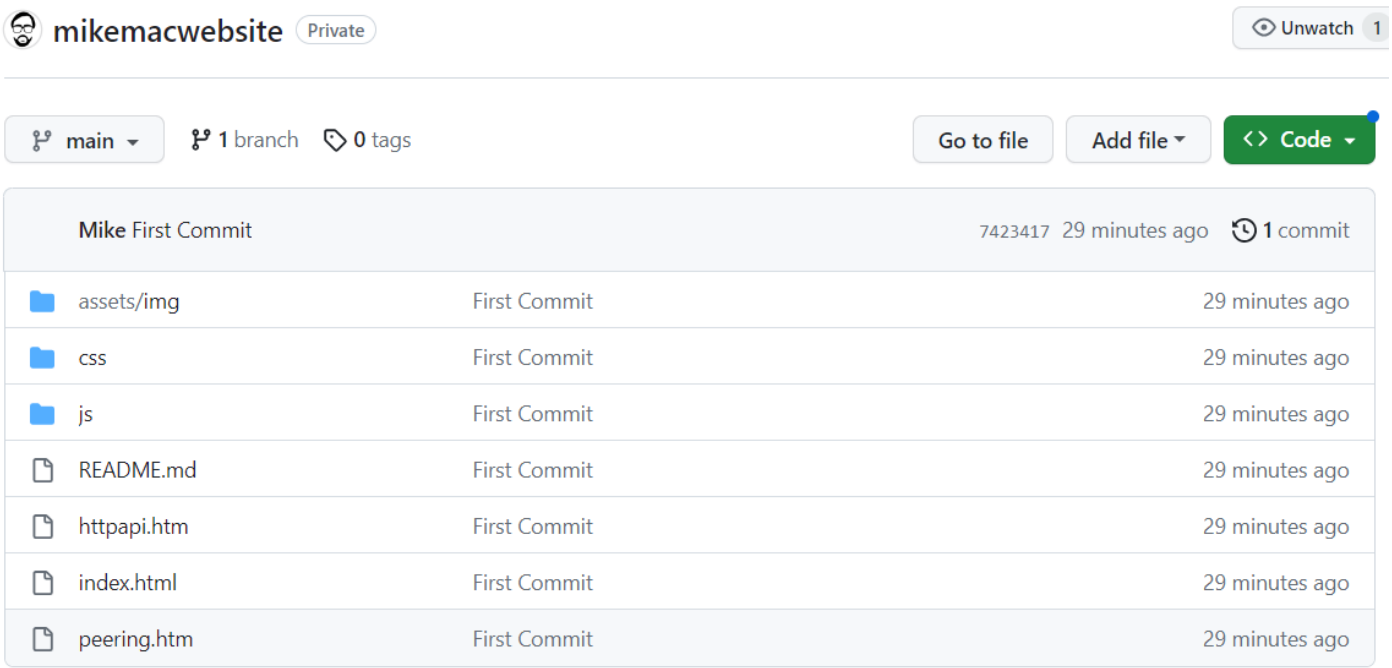
Next we need to commit these files using *git commit -m "First Commit".* This includes a comment for the commit "First Commit" – These comments are use to describe changes.

```
PS C:\Users\mike.macdonald\Documents\Website> git commit -m "First Commit"
[master (root-commit) 7423417] First Commit
 24 files changed, 11511 insertions(+)
 create mode 100644 README.md
 create mode 100644 assets/img/DynamoDB.png
 create mode 100644 assets/img/Favicon.ico
 create mode 100644 assets/img/amazon-api-gateway.png
 create mode 100644 assets/img/amazon-rds.png
 create mode 100644 assets/img/aws-cloud-practitioner.png
 create mode 100644 assets/img/aws-cloudquest-cloud-practitioner.png
 create mode 100644 assets/img/aws-cloudquest-solutions-architect.png
 create mode 100644 assets/img/aws-saa.png
 create mode 100644 assets/img/camping.jpg
 create mode 100644 assets/img/corfu.jpg
 create mode 100644 assets/img/dance.jpg
 create mode 100644 assets/img/guitar.jpg
 create mode 100644 assets/img/northface.jpg
 create mode 100644 assets/img/profile.JPG
 create mode 100644 assets/img/qts.jpeg
 create mode 100644 assets/img/skiing.JPG
 create mode 100644 assets/img/twominutetuesdays.png
 create mode 100644 assets/img/vpcpeering.png
 create mode 100644 css/styles.css
 create mode 100644 httpapi.htm
 create mode 100644 index.html
 create mode 100644 js/scripts.js
 create mode 100644 peering.htm
PS C:\Users\mike.macdonald\Documents\Website>
```

Now this needs pushing to the remote repository on GitHub. This is done with the command *git push –u origin main* (Note: This command with *–u origin main* only needs doing once, from this point on just *git push* can be used. *–u origin main* sets up a tracking relationship between the local *main* branch with the remote *origin/main* branch in the remote repository.)

```
PS C:\Users\mike.macdonald\Documents\Website> git push -u origin main
Enumerating objects: 30, done.
Counting objects: 100% (30/30), done.
Delta compression using up to 8 threads
Compressing objects: 100% (26/26), done.
Writing objects: 100% (30/30), 2.82 MiB | 2.77 MiB/s, done.
Total 30 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To github.com:mkmacd/mikemacwebsite
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
PS C:\Users\mike.macdonald\Documents\Website>
```

This can be seen within the remote repository on GitHub



3.5 MAKING CHANGES TO INDEX.HTML

If I now make changes to index.html file by removing these two unnecessary blank lines:

```
83                      <hr class="m-0 mt-5">
84
85
86                      <div class="subheading mb-3">Two Minute Tuesdays</div>
87                      <div class="row ">
88                          <div class="col-sm-4">
89                              <img class="img-thumbnail " src="assets/img/twominutetuesdays.png" alt="...">
90                          </div>
```

To become:

```
83                      <hr class="m-0 mt-5">
84                      <div class="subheading mb-3">Two Minute Tuesdays</div>
85                      <div class="row ">
86                          <div class="col-sm-4">
87                              <img class="img-thumbnail " src="assets/img/twominutetuesdays.png" alt="...">
88                          </div>
```

Then save the changes and run the command *git status* again this shows there has been a modification to index.html:

```
PS C:\Users\mike.macdonald\Documents\Website> git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   index.html

no changes added to commit (use "git add" and/or "git commit -a")
```

It is possible to view the modification that have been made using *git diff ./index.html*

```
PS C:\Users\mike.macdonald\Documents\Mike-Macdonald.co.uk Website> git diff ./index.html
diff --git a/index.html b/index.html
index 59fa729..15706f9 100644
--- a/index.html
+++ b/index.html
@@ -80,9 +80,7 @@
                                                                <a href="https://github.com/mkmacd/API_to_start_and_stop
_EC2s" target = "_blank">HTTP API GitHub Repository</a>
                                                            </p></div>

-                                   <hr class="m-0 mt-5">
-
-
+                                   <hr class="m-0 mt-5">
                                    <div class="subheading mb-3">Two Minute Tuesdays</div>

                                    <div class="row ">
                                        <div class="col-sm-4">
PS C:\Users\mike.macdonald\Documents\Mike-Macdonald.co.uk Website>
```

This shows the lines that have been removed.

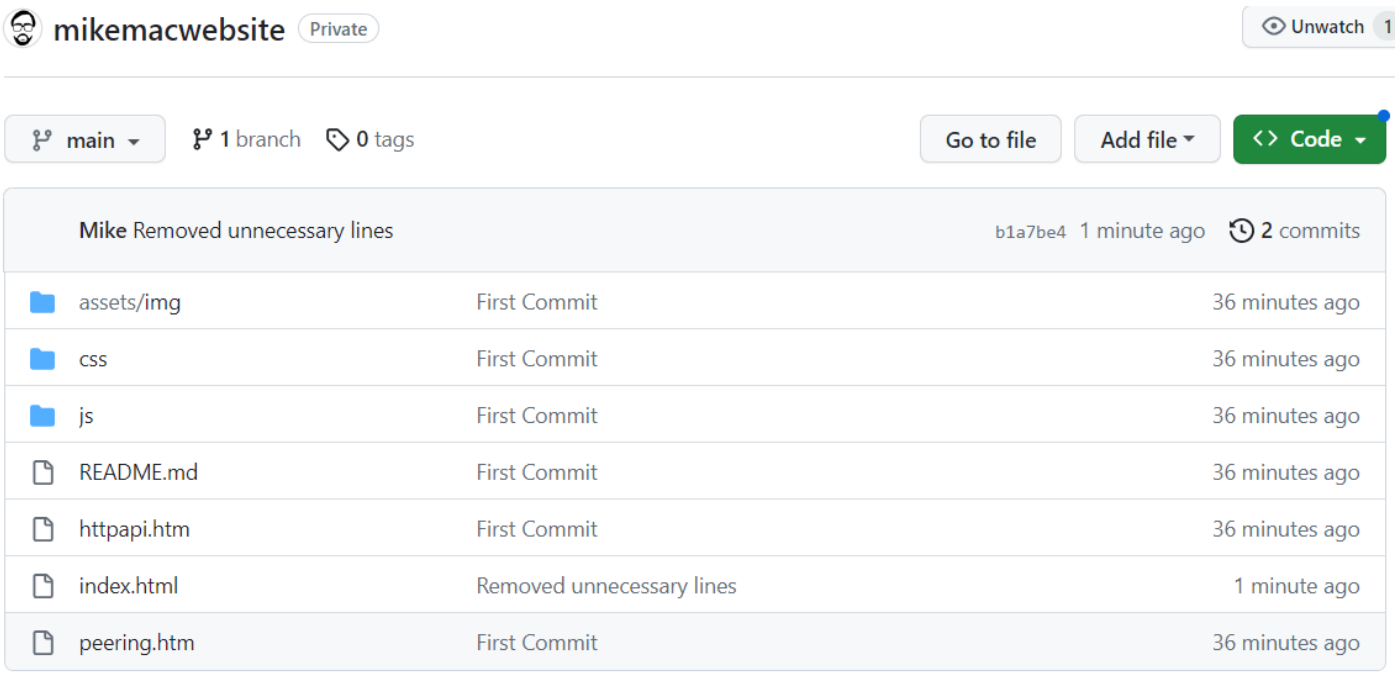
This change hasn't yet been staged, but can be with *git add ./index.html* (This has no output)

This can then be committed with git commit –m "Removed unnecessary lines"

```
PS C:\Users\mike.macdonald\Documents\Website> git commit -m "Removed unnecessary lines"
[main b1a7be4] Removed unnecessary lines
 1 file changed, 1 insertion(+), 3 deletions(-)
PS C:\Users\mike.macdonald\Documents\Website>
```

And finally this commit can be pushed to the remote repository again using *git push*

You can see that in the remote repository that index.html has a more recent commit.



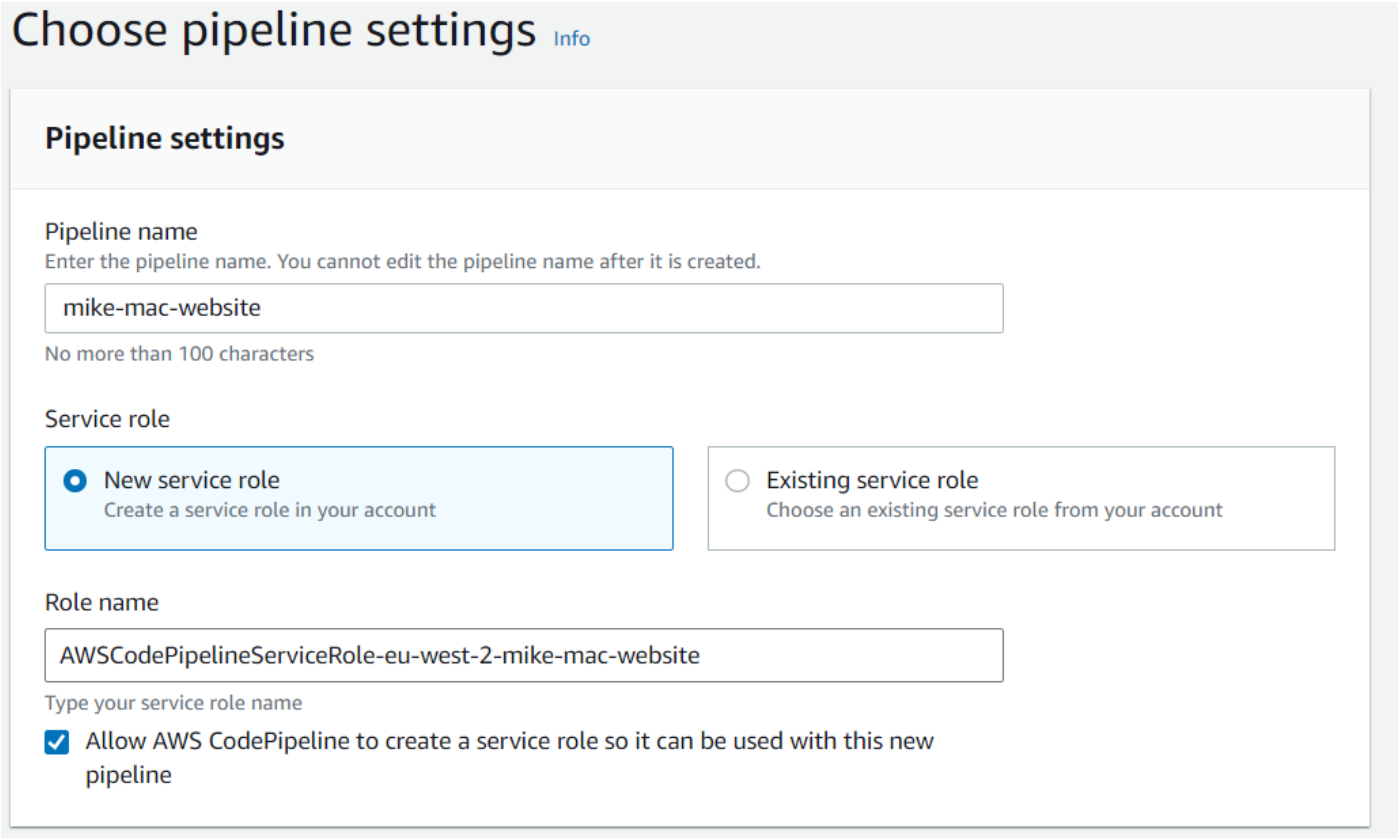
4. INTEGRATION WITH AWS CODEPIPELINE

4.1 OVERVIEW

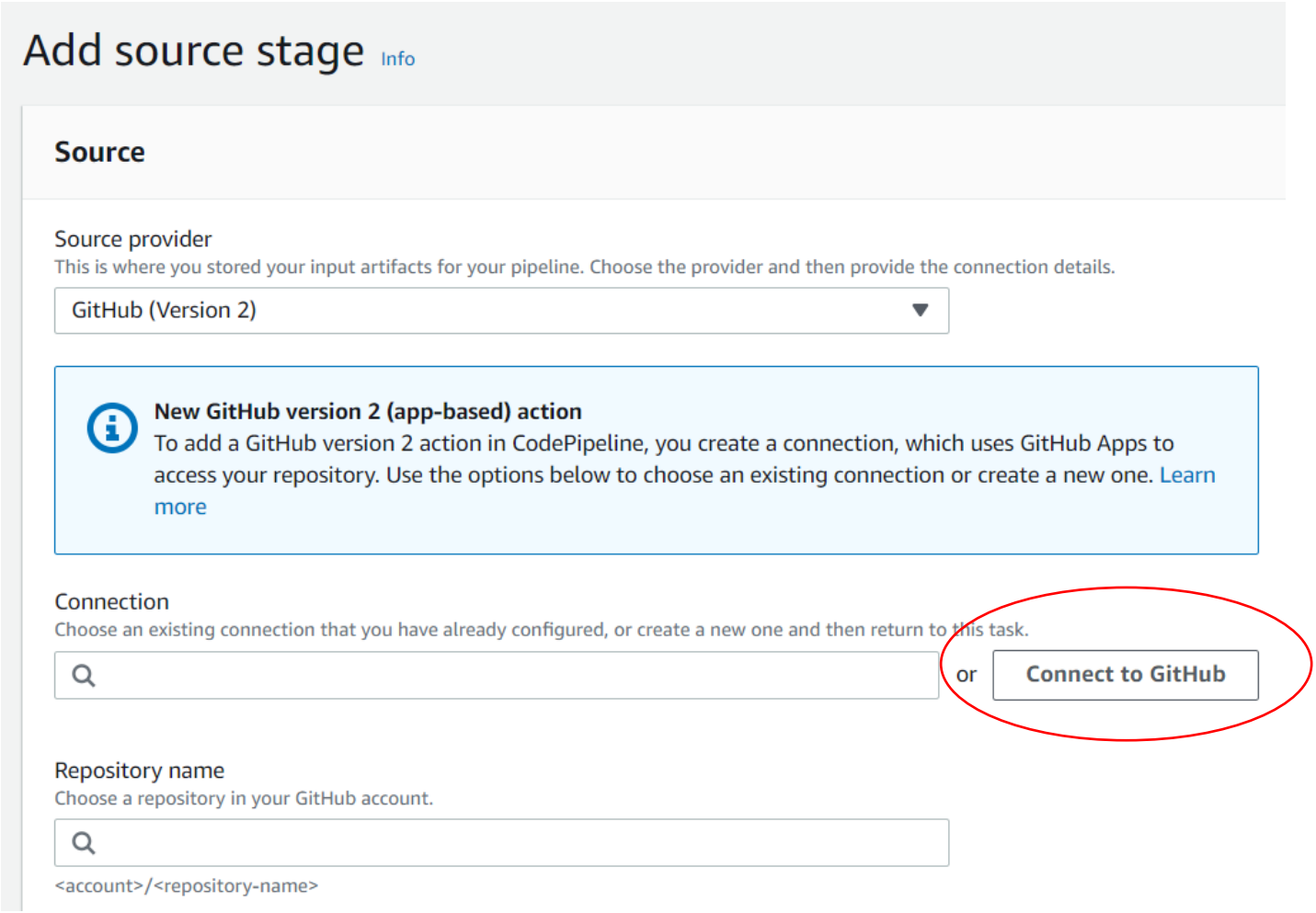
AWS CodePipeline can be configured to pull source code from GitHub (or other online repositories). To automate pipeline executions, webhooks from the GitHub repository can be used which notify CodePipeline when new changes have been pushed to the repository. When these are triggered CodePipeline pulls the changes and passes them to subsequent stages in the pipeline (such as building, testing and deployment).

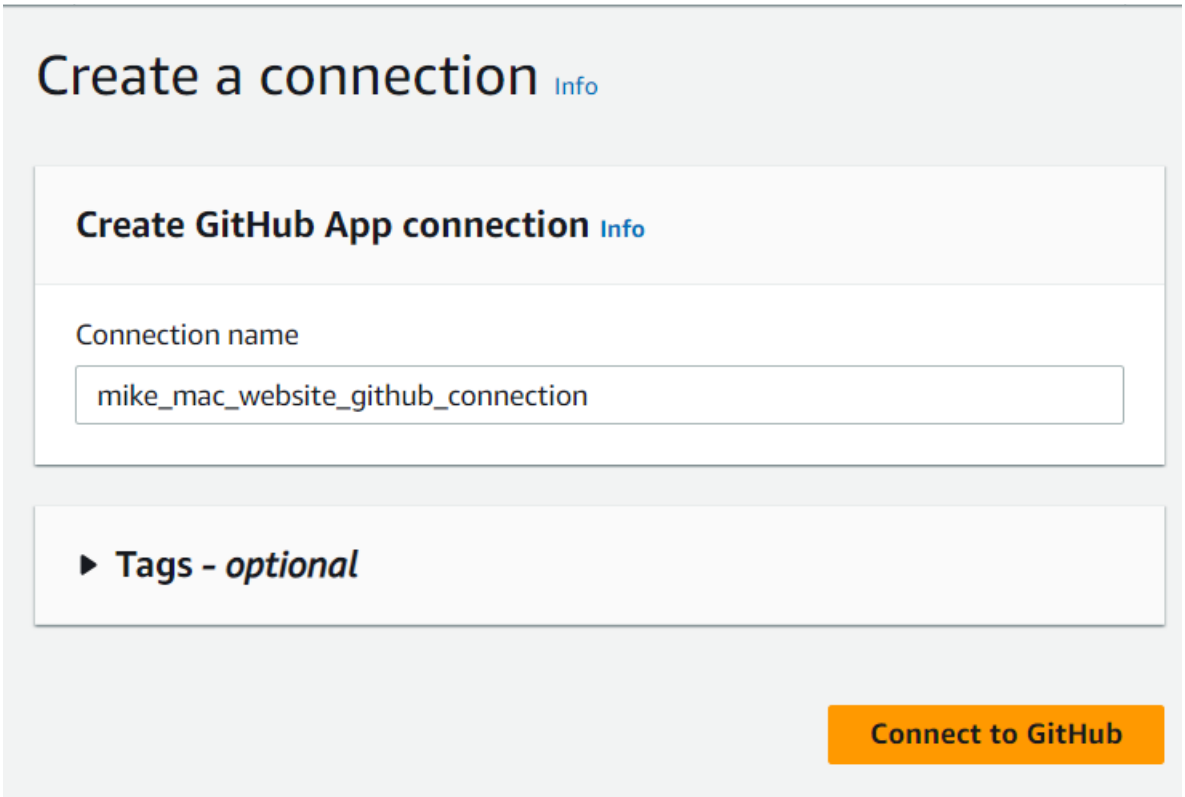
4.2 CONNECTING GITHUB AND AWS CODEPIPELINE

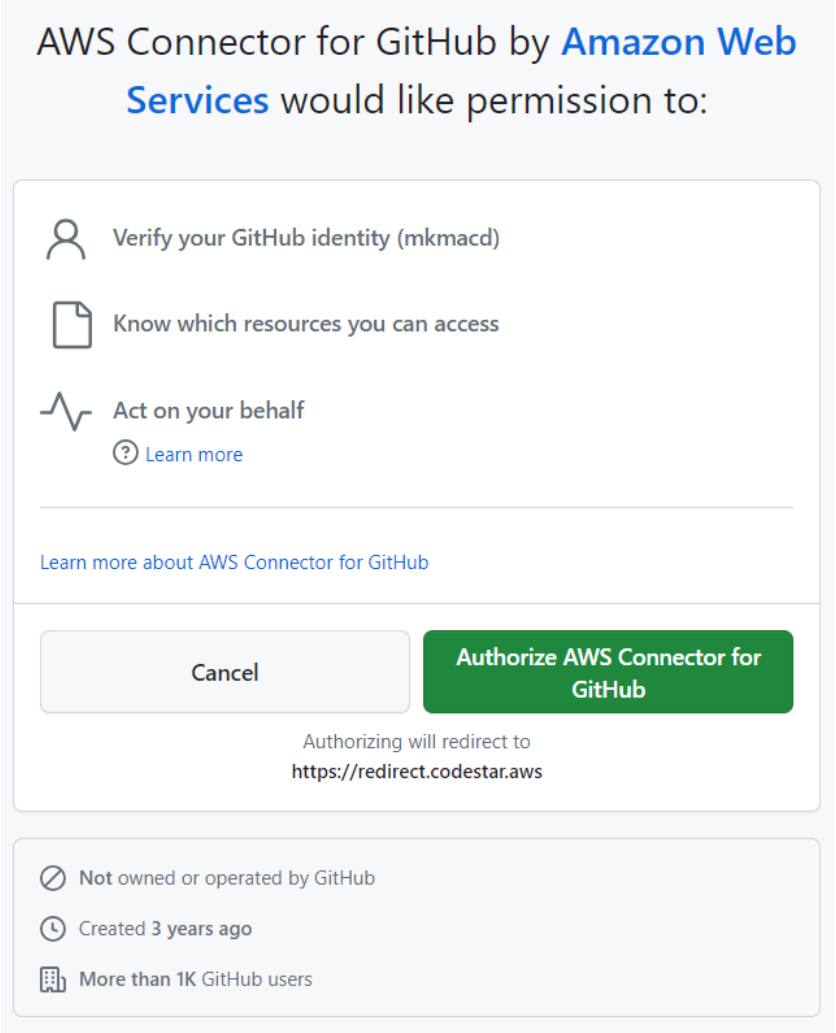
4.2.1 CREATE THE PIPELINE SETTINGS

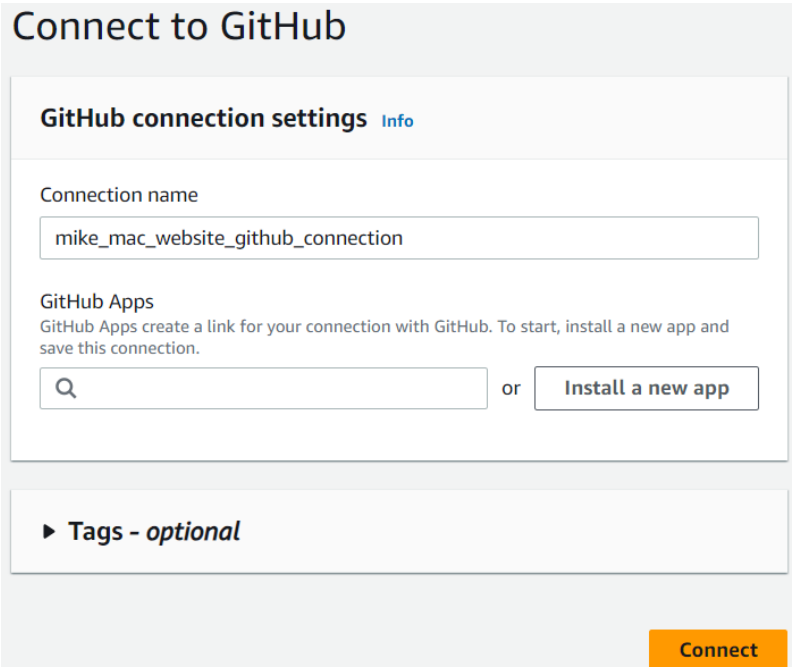


4.2.2 CONNECT GITHUB






AWS Connector for GitHub by Amazon Web Services would like permission to:
Verify your GitHub identity (mkmacd)
Know which resources you can access
Act on your behalf
Learn more
Learn more about AWS Connector for GitHub
Cancel
Authorize AWS Connector for GitHub
Authorizing will redirect to
https://redirect.codestar.aws
Not owned or operated by GitHub
Created 3 years ago
More than 1K GitHub users


Connect to GitHub
GitHub connection settings Info
Connection name
mike_mac_website_github_connection
GitHub Apps
GitHub Apps create a link for your connection with GitHub. To start, install a new app and save this connection.
or
Install a new app
Tags - optional
Connect

At this stage you have to install a new app (assuming you haven't already done this. I've chosen to only allow the single website repository.

Install on your personal account mkmacd

All repositories
This applies to all current *and* future repositories owned by the resource owner.
Also includes public repositories (read-only).

Only select repositories
Select at least one repository.
Also includes public repositories (read-only).

Select repositories

Selected 1 repository.

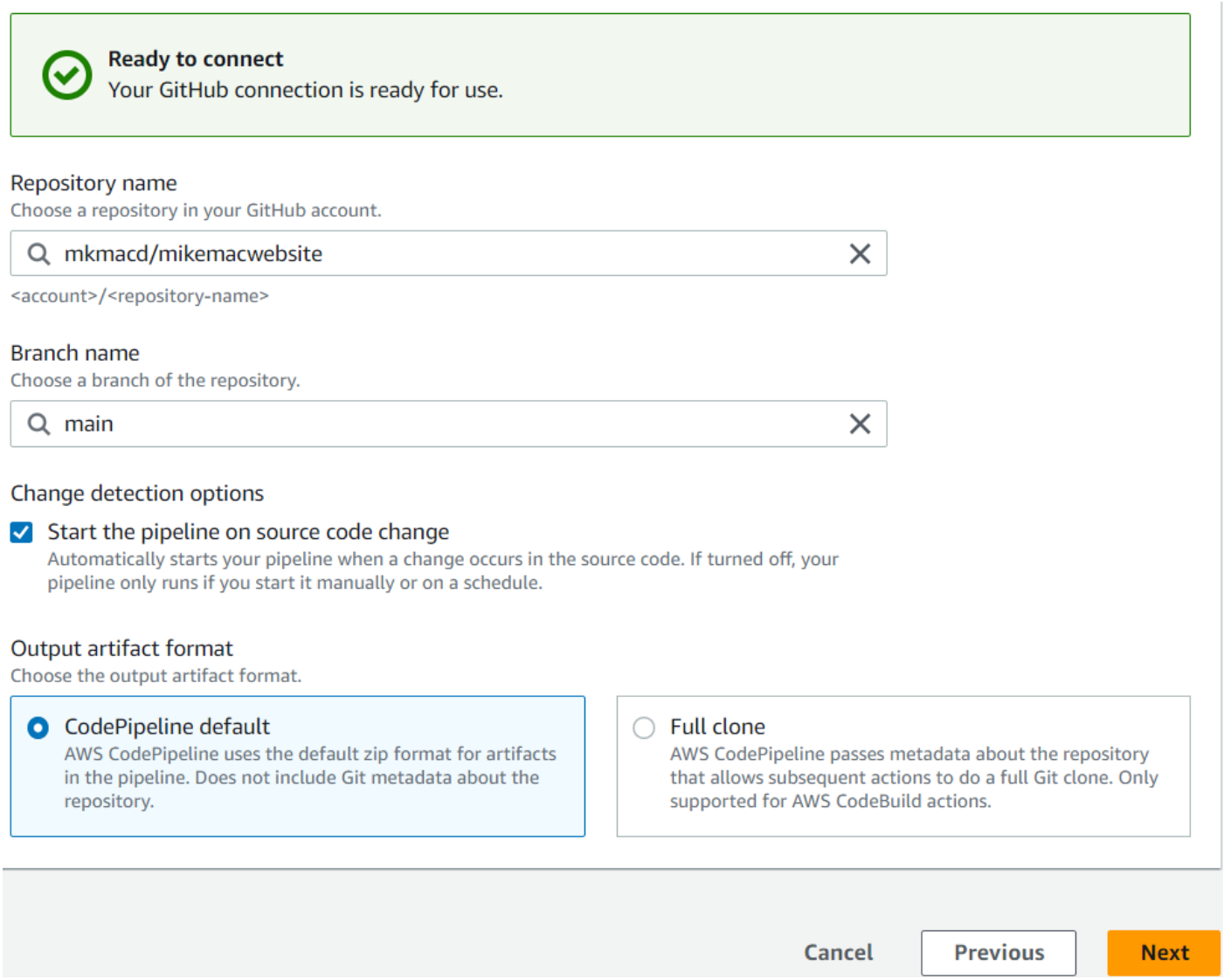
mkmacd/**mikemacwebsite**

with these permissions:

- **Read** access to issues and metadata
- **Read** and **write** access to administration, code, commit statuses, and pull requests

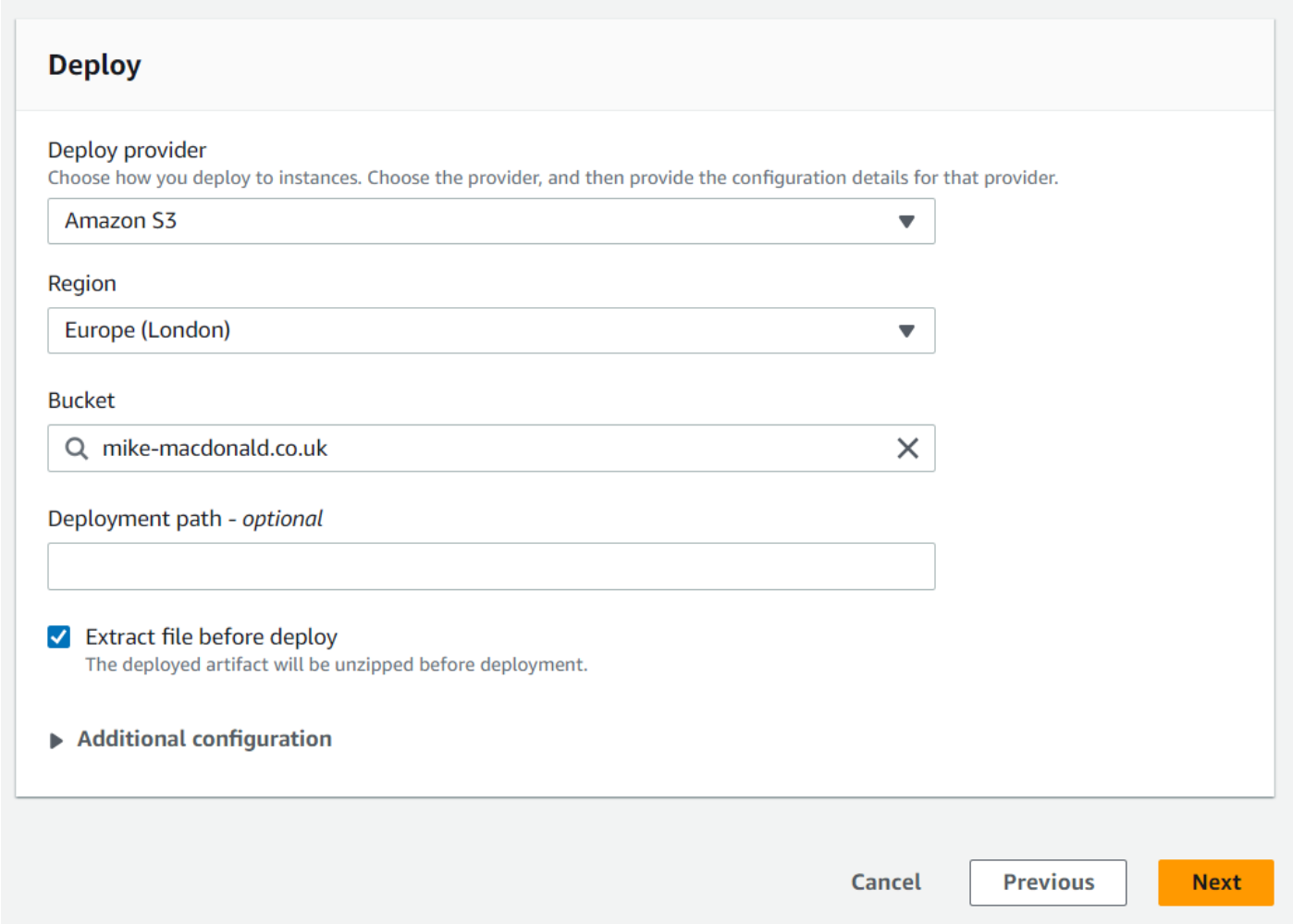
Install Cancel

Next: you'll be directed to the GitHub App's site to complete setup.

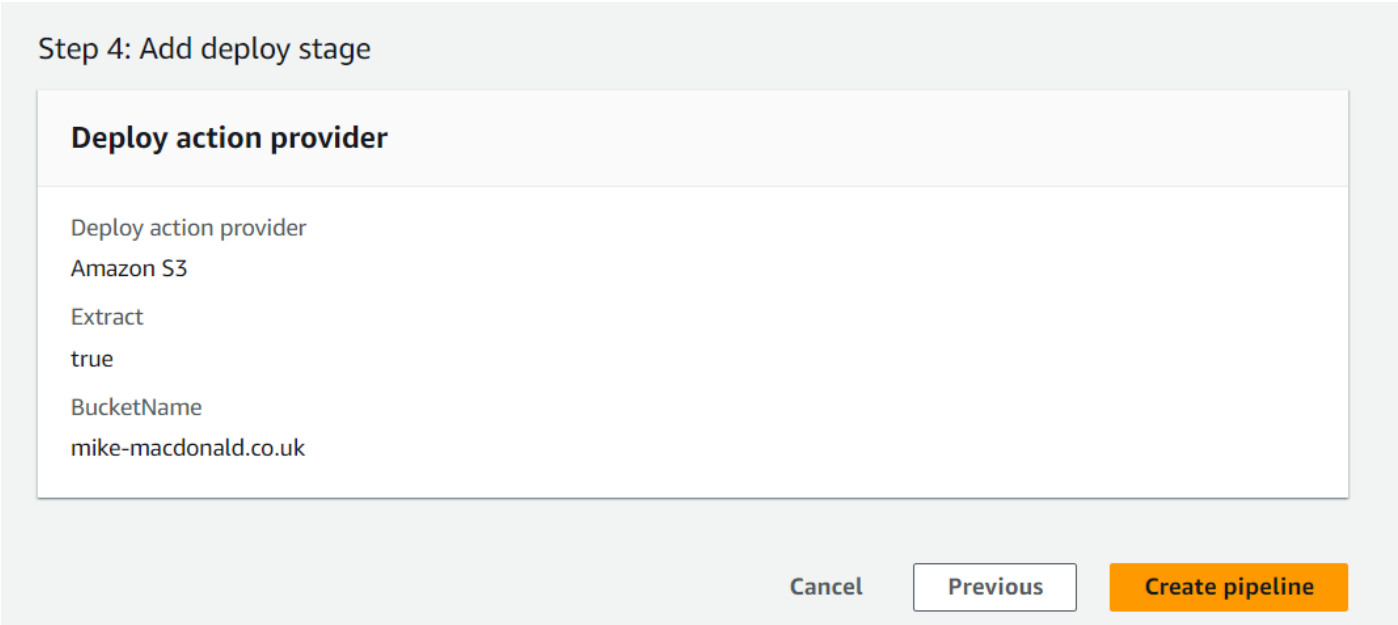


We can then skip the build stage as this isn't relevant to this project.

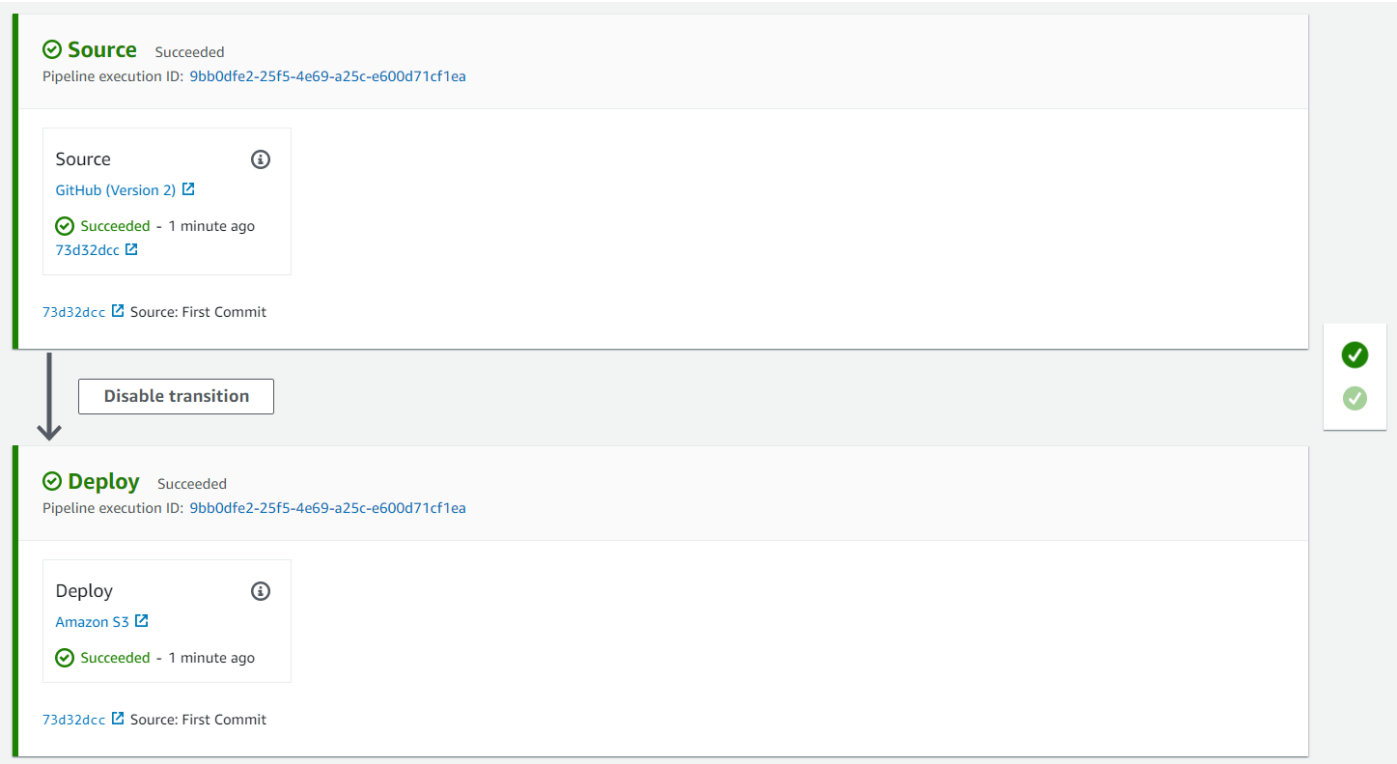
It’s here we chose to deploy to a specific S3 bucket.



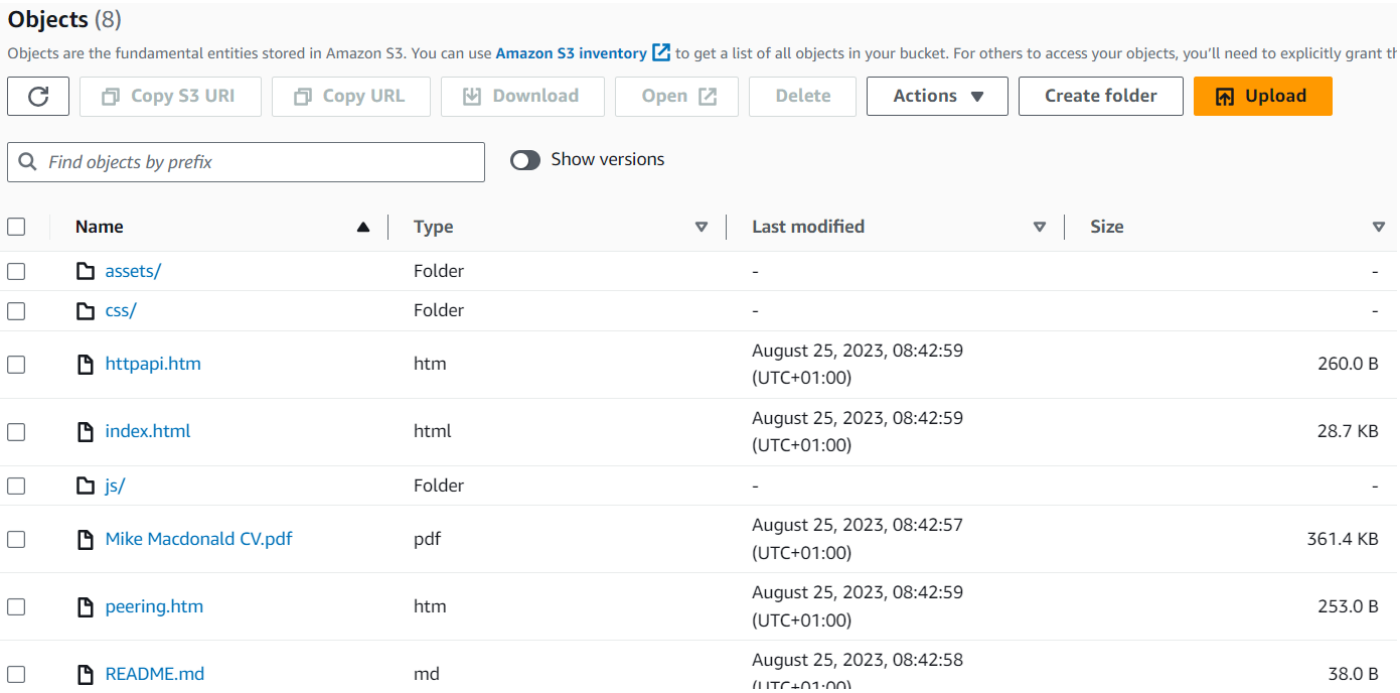
The finally we can create the pipeline.



AWS then confirms that this has worked



CodePipeline is now pulling any changes from GitHub and deploying them to the bucket that the website is hosted in. Looking at the S3 bucket you can see all the files have been updated at the same time. This is because CodePipeline deploys the entire repository rather than just individual changed files.



5. LAMBDA FUNCTION THAT INVALIDATES CLOUDFRONT CACHE

As CloudFront caches content at edge locations for faster delivery, if the cache isn't updated then this can lead to out of date content. By default CloudFront caches content from S3 every 24 hours. However it is possible to manually invalidate a CloudFront distribution to force it to cache a new response.

As I want the website to be as up to date as possible I created a Lambda function which will be triggered every time a file is uploaded to the website S3 bucket (which will happen via CloudPipeline every time new files are pushed to GitHub).

5.1 CREATING THE LAMBDA FUNCTION

The Lambda functions receives the event from the S3 bucket when new objects are uploaded. It then cycles through the names of the objects and if there is an object called "index.html" then it creates an invalidation on the CloudFront distribution.

```
import json
import boto3

s3_client = boto3.client('s3')
cloudfront_client = boto3.client('cloudfront')

def lambda_handler(event, context):
    for record in event['Records']:
        if record['s3']['object']['key'] == "index.html":
            invalidation = cloudfront_client.create_invalidation(
                DistributionId = "E204AHJ2Q4OTIR",
                InvalidationBatch ={
                    'Paths': {
                        "Quantity": 1,
                        "Items": ["/*"]
                    },
                    "CallerReference": str(record['eventTime'])
                }
            )
            return {
                'statusCode': 200,
                'body': json.dumps("Cloudfront invalidation successfully created")
            }
    return{
        'statusCode': 204,
        'body': json.dumps("No file called index.html therefore no invalidation
created")
    }
```

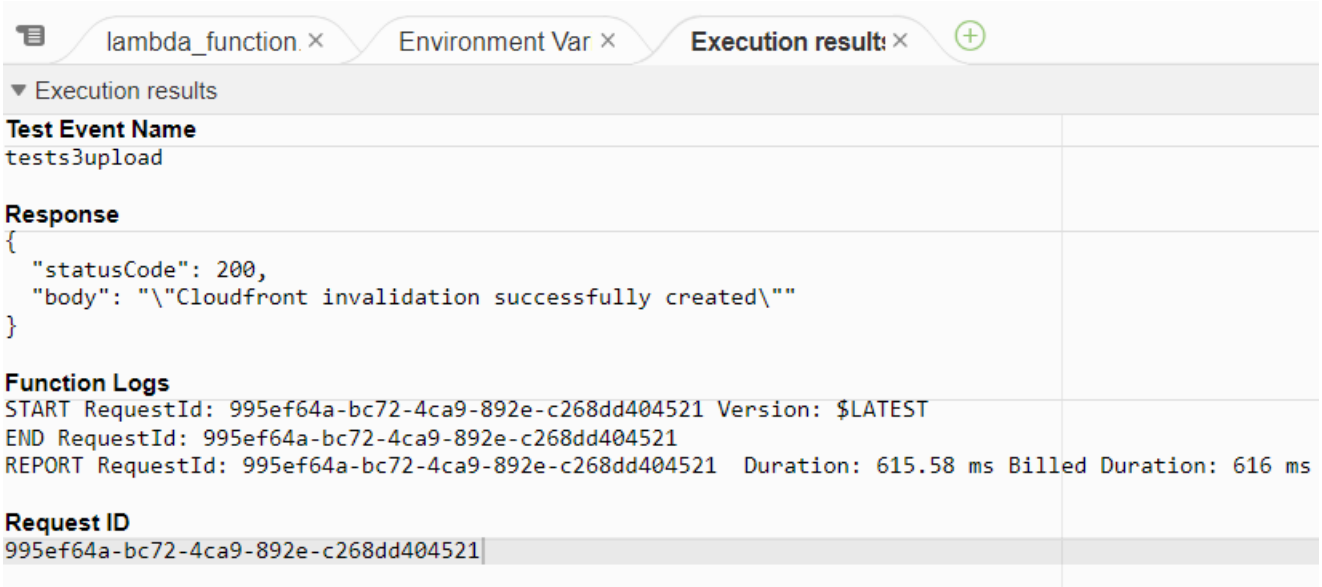
5.2 GIVING LAMBDA CORRECT PERMISSIONS

Lambda needs permissions to be able to create the invalidation so I created an inline policy and attached it to the basic Lambda execution role.

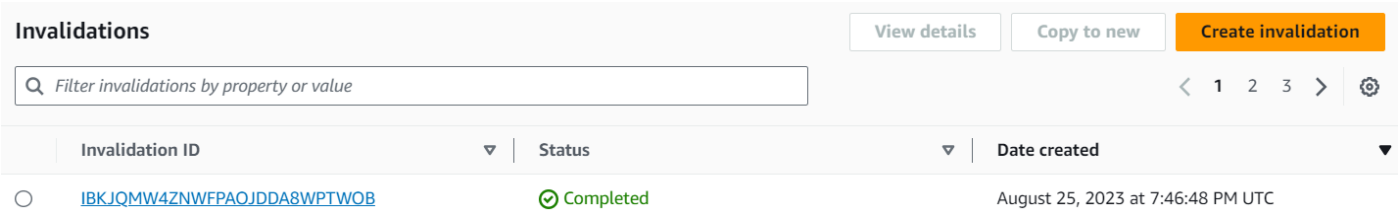
```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "Allow Distribution Invalidation",
         "Effect": "Allow",
         "Action": "cloudfront:CreateInvalidation",
         "Resource": "arn:aws:cloudfront::601403623821:distribution/E204AHJ2Q4OTIR"
      }
   ]
}
```

5.3 TESTING THE LAMBDA FUNCTION

I created a test event within Lambda to check functionality. To do this I used AWS' example event (https://docs.aws.amazon.com/AmazonS3/latest/userguide/notification-content-structure.html). I changed the object key to "index.html" to ensure that test passed.

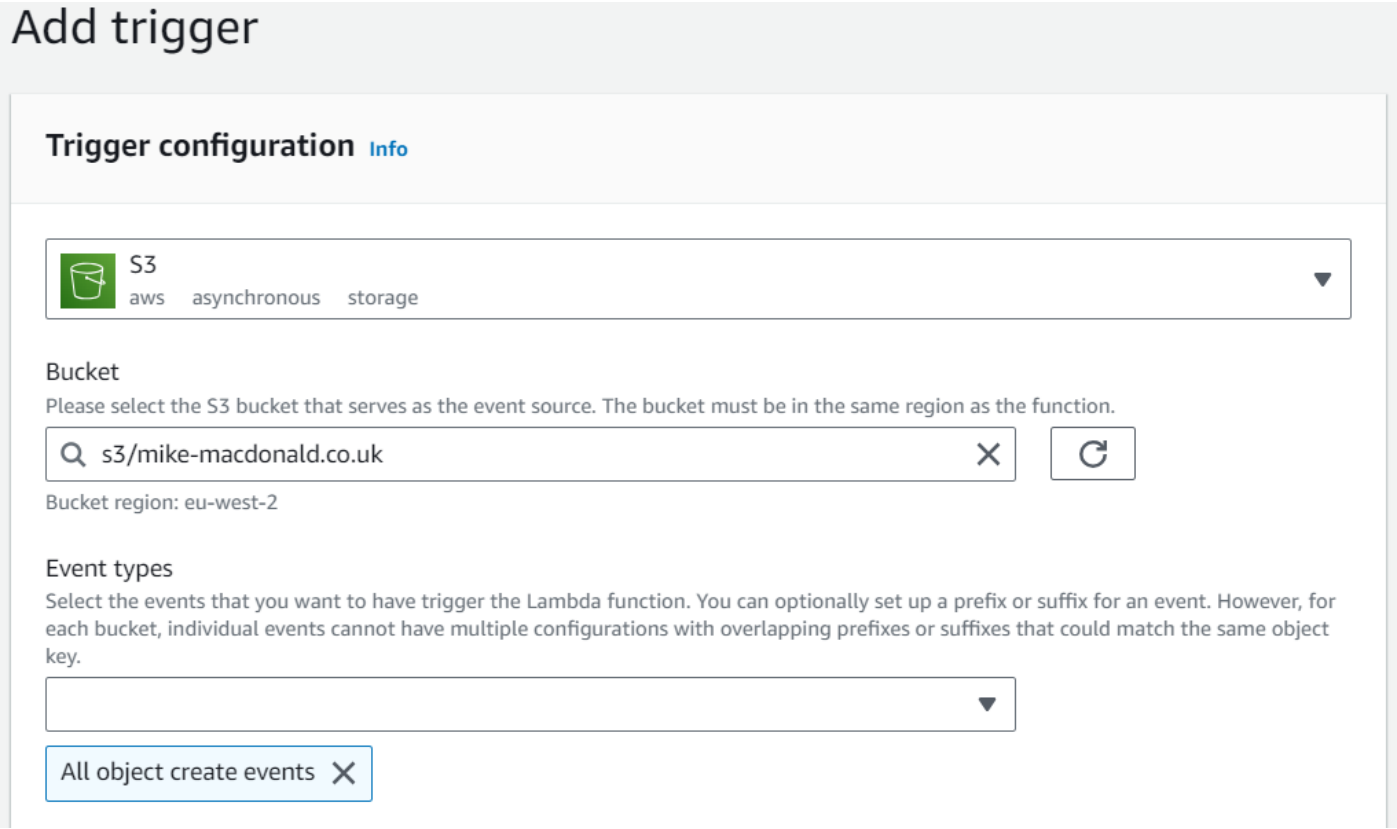


Invalidation is created.



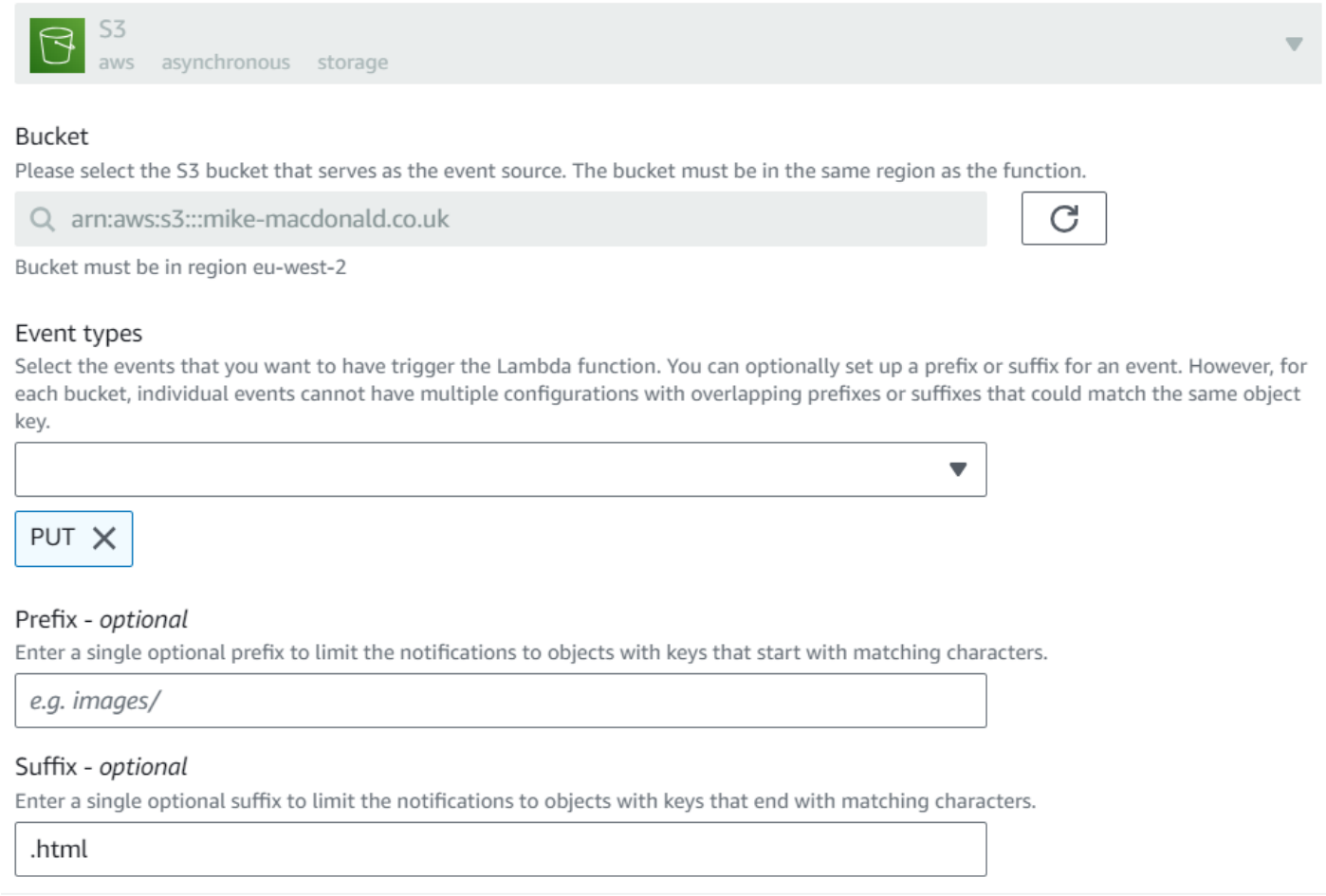
5.4 TRIGGERING LAMBDA WITH S3 EVENT

You can create an S3 trigger within the Lambda function. Specific events within the bucket trigger the lambda function.

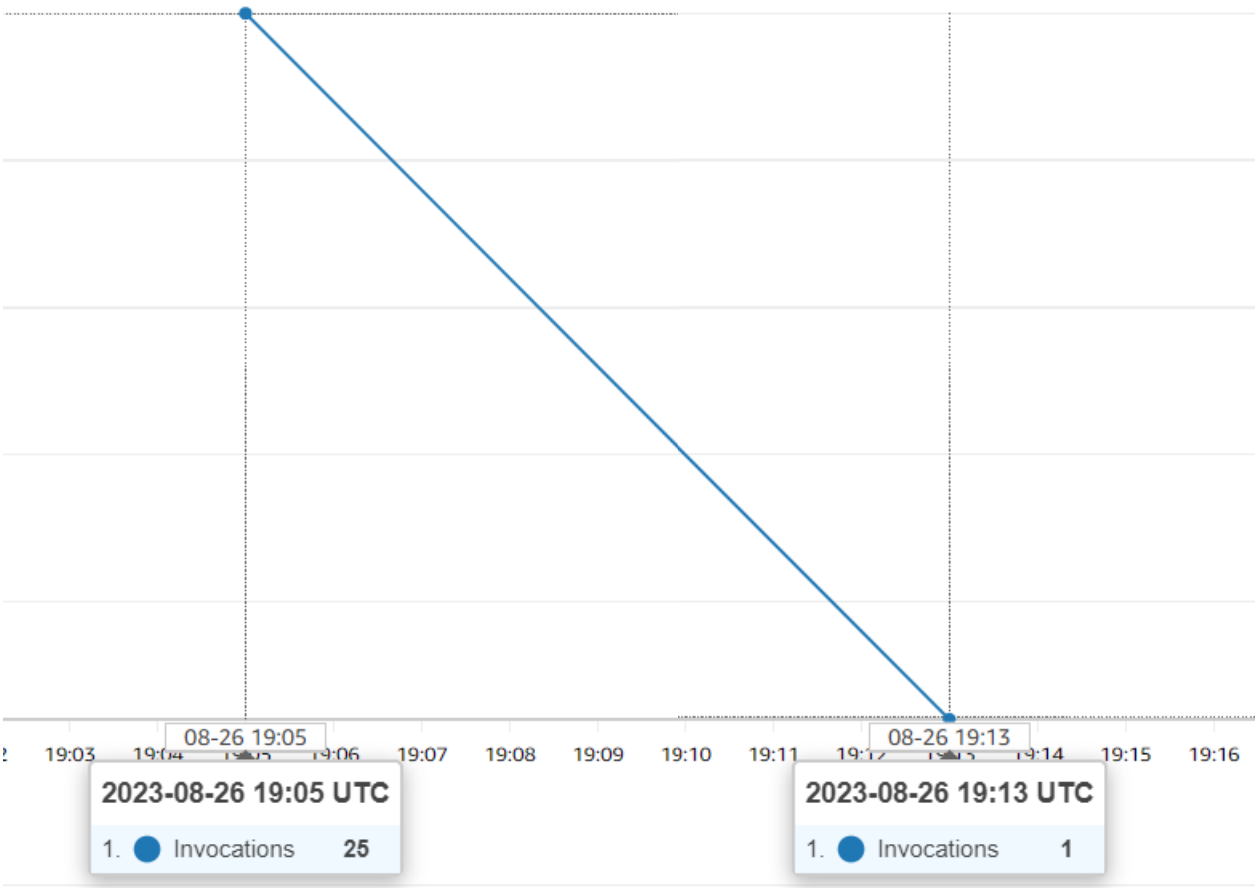


5.5 UPDATING LAMBDA TRIGGER

Whilst the above trigger works, it isn't cost effective. It triggers the lambda function for every file that is uploaded. In this situation that's 25 files and so the lambda function runs 25 times (and would increase if there were more files). The only time I want the lambda function to run when the index.html file is uploaded and so I updated the trigger configuration to only trigger lambda when a PUT event occurs and it has a suffix of .html This should significantly reduce the number of lambda function calls.



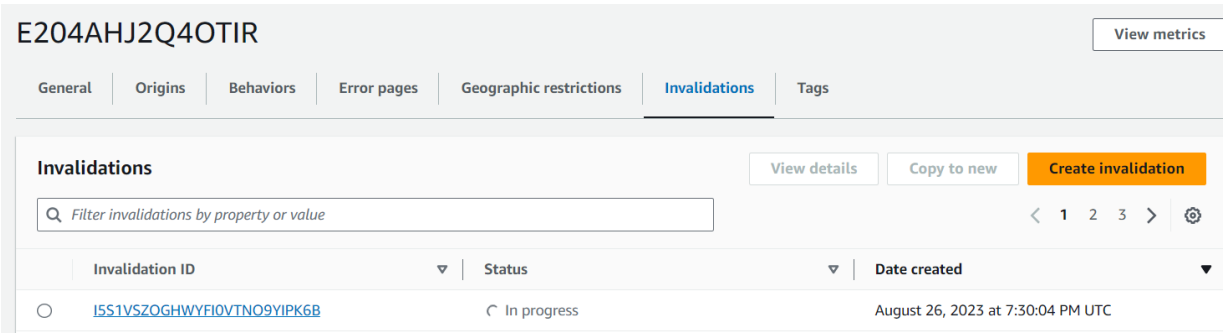
You can see from the CloudWatch metrics, at 19:05, before the change to the trigger, the lambda function was called 25 times but at 19:13 after the change, the lambda function was only called once.



5.6 FINAL TEST

As a final test I changed part of the index.html file, committed the changes and pushed them to the remote GitHub repository to test if the entire process functioned as expected.

The invalidation is created immediately.



The website is instantly updated due to the invalidation.

6. CONCLUSION

Security should always be a major consideration with any architecture. S3 buckets that don't need to be public never should be as there are always websites trawling for public buckets (see https://buckets.grayhatwarfare.com/) and so using methods to secure these are critical. This can include Origin Access Identity but also using interface or gateway endpoints to prevent services needing to cross the public internet to access files within S3 buckets.

CI/CD is a key part of software development. Whilst this is a basic use case of AWS Codepipline, it can be used along with other software deployment services such as AWS Codecommit (AWS' distributed version control service) and AWS Codedeploy which deploys committed code to AWS services such as EC2, on premises servers, Lambda function, ECS etc.

Finally using event driven architecture such as Lambda can simplify workflows but automating tasks. In this case it keeps the website perfectly up to date and removes the need to manually invalidate the CloudFront distribution everytime a change is made.