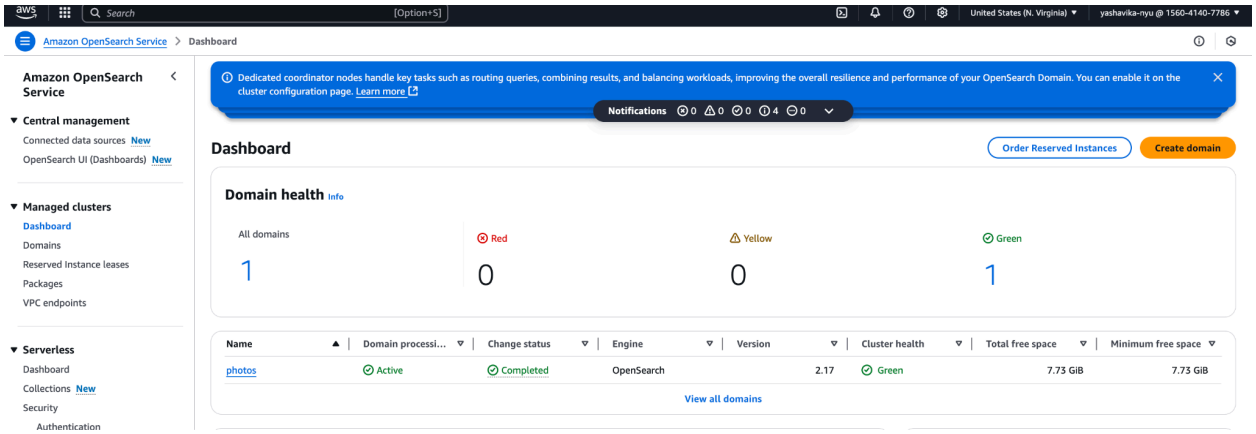
ASSIGNMENT - 3

Sindhu Jyoti Dutta - sd6201
Yashavika Singh - ys6668

1. Launch an ElasticSearch instance **3**
a. Using AWS ElasticSearch service, create a new domain called “photos” 3
2. Upload & index photos **4**
a. Create a S3 bucket (B2) to store the photos. 4
b. Create a Lambda function (LF1) called “index-photos” 4
C. Set up a PUT event trigger4 on the photos S3 bucket (B2), such that whenever a photo gets uploaded to the bucket, it triggers the Lambda function (LF1) to index it. 5
d. Implement the indexing Lambda function (LF1): 6
3. Search **9**
a. Create a Lambda function (LF2) called “search-photos”. **9**
b. Create an Amazon Lex bot to handle search queries. **9**
c. Implement the Search Lambda function (LF2): **10**
4. Build the API layer **13**
a. Build an API using APIGateway. 13
b. The API should have two methods: 14
c. Setup an API key for your two API methods. 17
d. Deploy the API. 18
e. Generate a SDK for the API (SDK1). 19
5. Frontend **20**
a. Build a simple frontend application that allows users to: 20
b. Create a S3 bucket for your frontend (B1). 23
c. Set up the bucket for static website hosting (same as HW1) 23
d. Upload the frontend files to the bucket (B2). 24
e. Integrate the API Gateway-generated SDK (SDK1) into the frontend, to connect your API. 25
6. Deploy your code using AWS CodePipeline **27**
a. Define a pipeline (P1) in AWS CodePipeline that builds and deploys the code for/to all your Lambda functions. 27
b. Define a pipeline (P2) in AWS CodePipeline that builds and deploysyour frontend code to its corresponding S3 bucket. 27
7. Create a AWS CloudFormation12 template for the stack **27**
template.yaml 27
Spinned up resources: 29

1. Launch an ElasticSearch instance

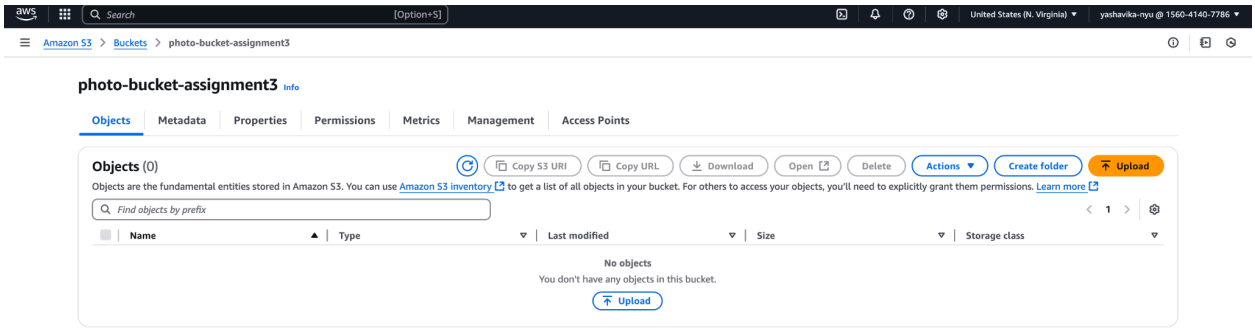
a. Using AWS ElasticSearch service, create a new domain called **“photos”**



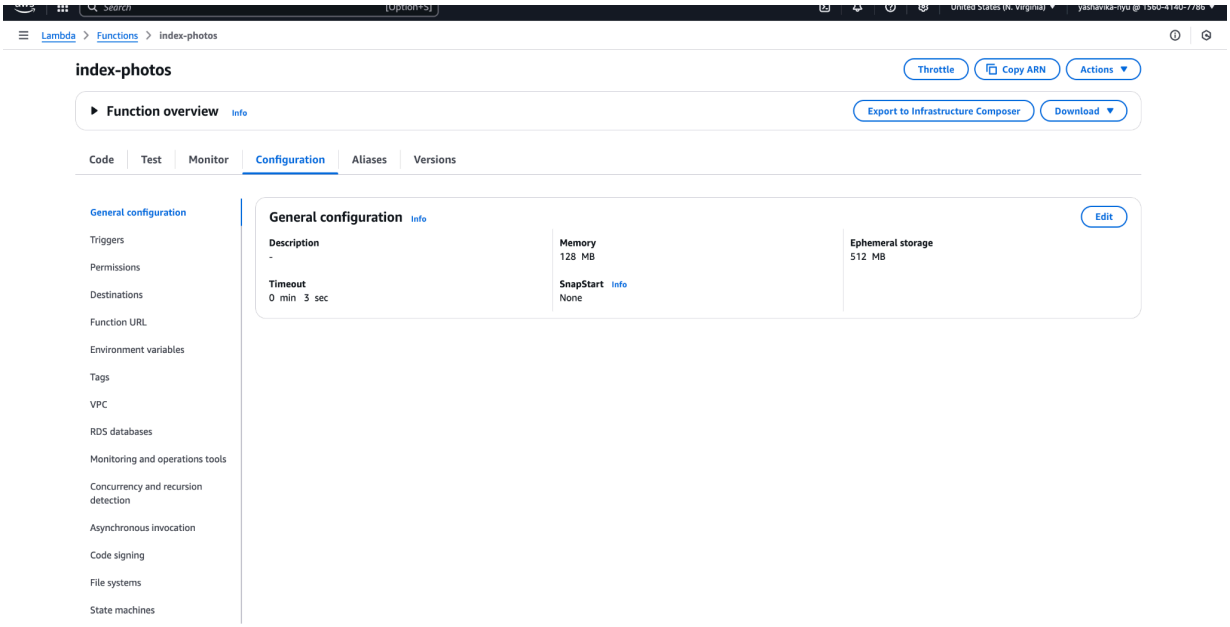
.

2. Upload & index photos

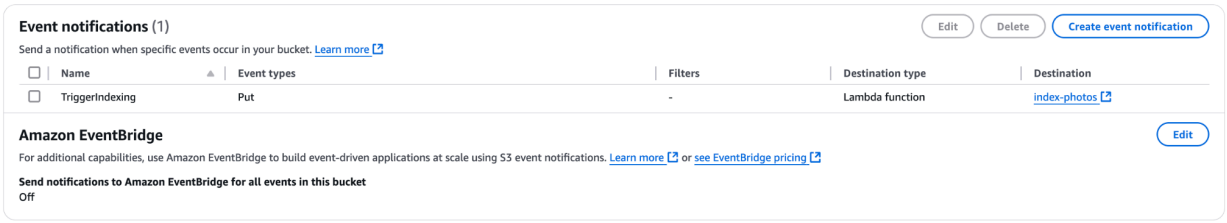
a. Create a S3 bucket (**B2**) to store the photos.



b. Create a Lambda function (**LF1**) called **“index-photos”**

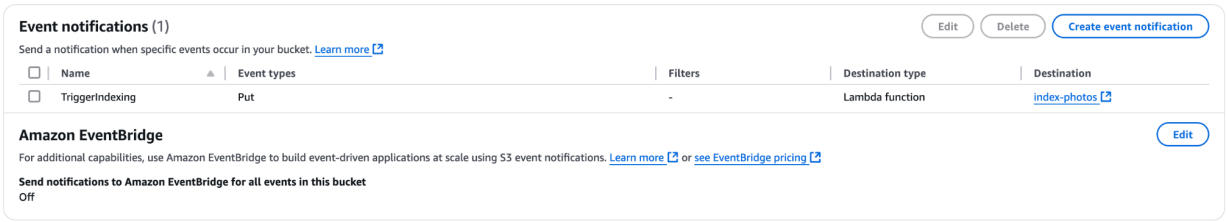


c. Set up a PUT event trigger4 on the photos S3 bucket (**B2**), such that whenever a photo gets uploaded to the bucket, it triggers the Lambda function (**LF1**) to index it.



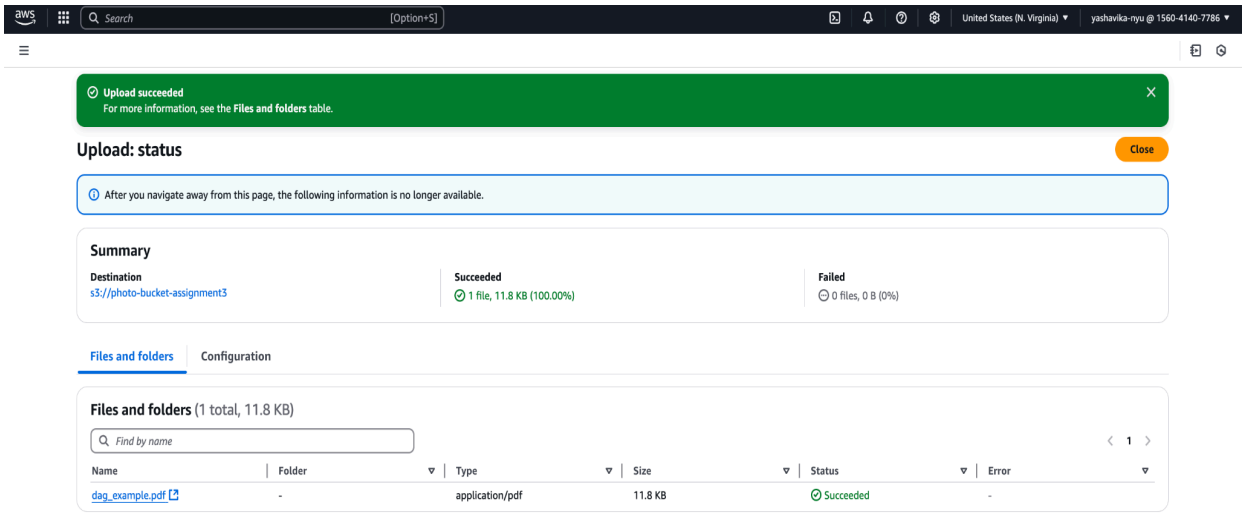
- To test this functionality, upload a file to the photos S3 bucket (**B2**) and check the logs of the indexing Lambda function (**LF1**) to see if it got invoked. If it did, your setup is complete.

  The event trigger on photos bucket:

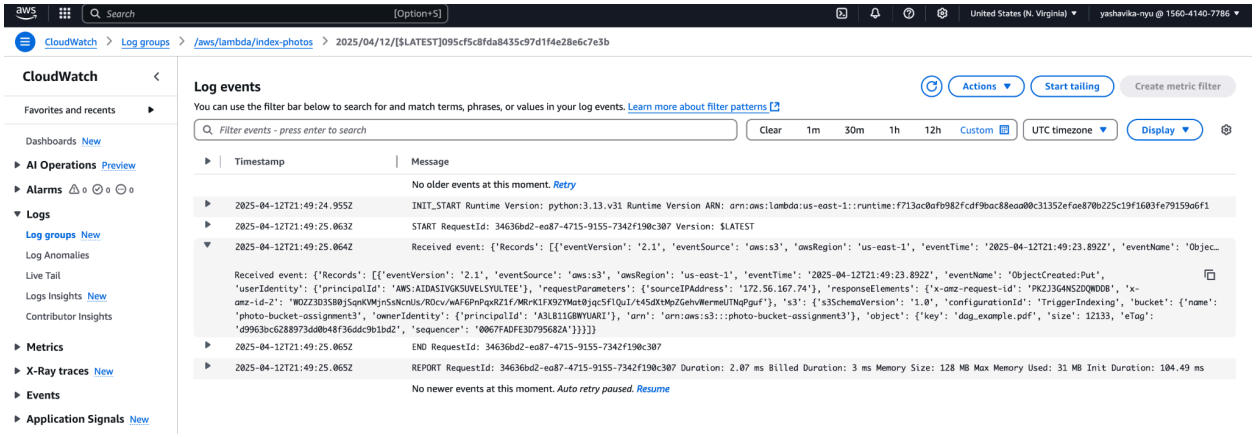


Triggers the index-photos lambda function when an image is uploaded to the photos bucket.

Uploaded a file to the s3 bucket



Cloudwatch logs showing the lambda function being triggered when an object was uploaded to the s3 bucket



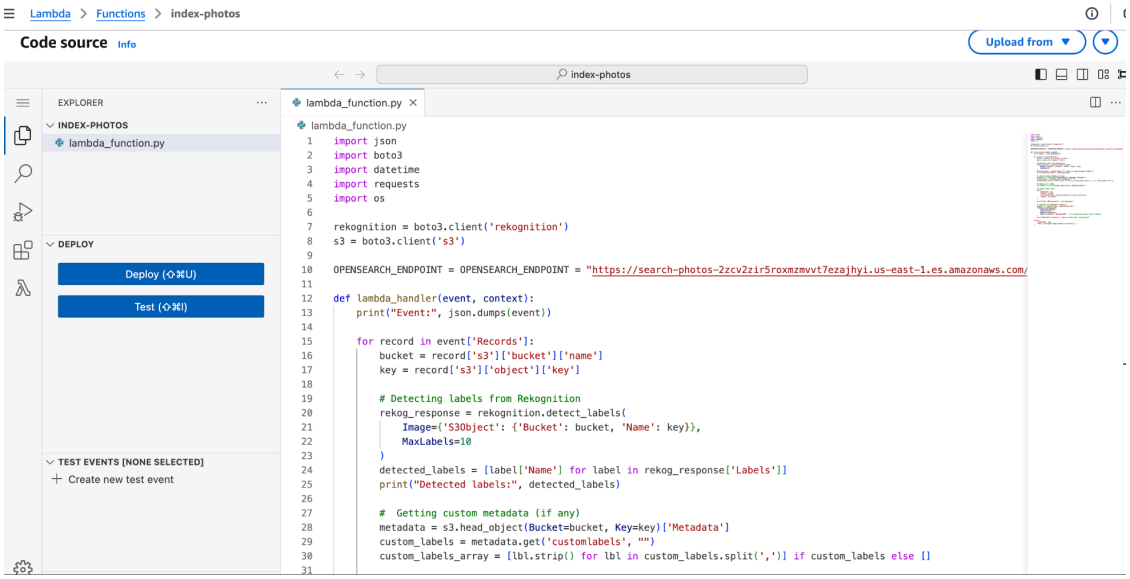
- If the Lambda (**LF1**) did not get invoked, check to see if you set up the correct permissions5for S3 to invoke your Lambda function.

d. Implement the indexing Lambda function (**LF1**):

- Given a S3 PUT event (**E1**) detect labels in the image, using Rekognition6 (“detectLabels” method).
- Use the S3 SDK’s headObject method7to retrieve the S3 metadata created at the object’s upload time. Retrieve the **x-amz meta-customLabels** metadata field, if applicable, and create a JSON array (**A1**) with the labels.
- Store a JSON object in an ElasticSearch index (“photos”) that references the S3 object from the PUT event (**E1**) and append string labels to the labels array (**A1**), one for each label detected by Rekognition.

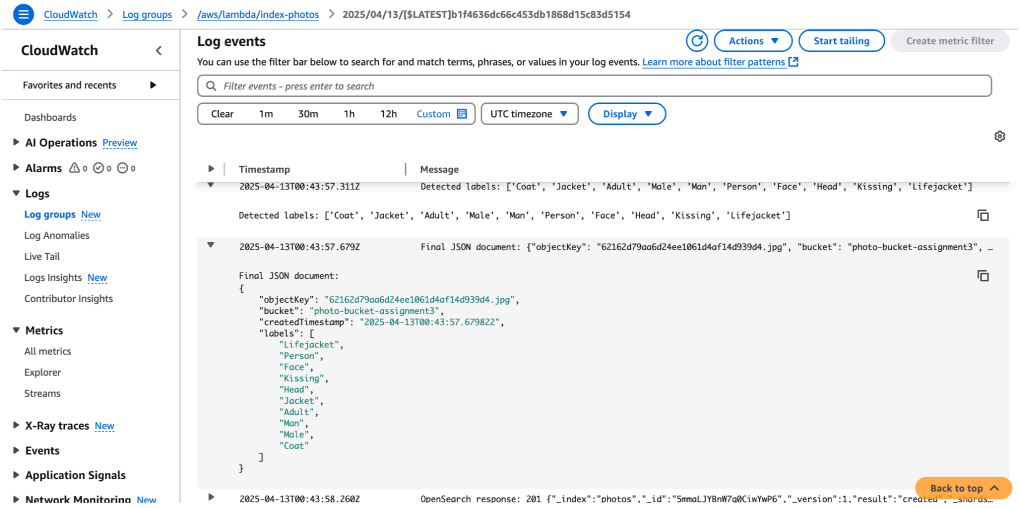
– Implementing indexing

Lambda function



Uploaded an image to s3 bucket

Cloudwatch logs



PUT to opensearch

Tested with curl:

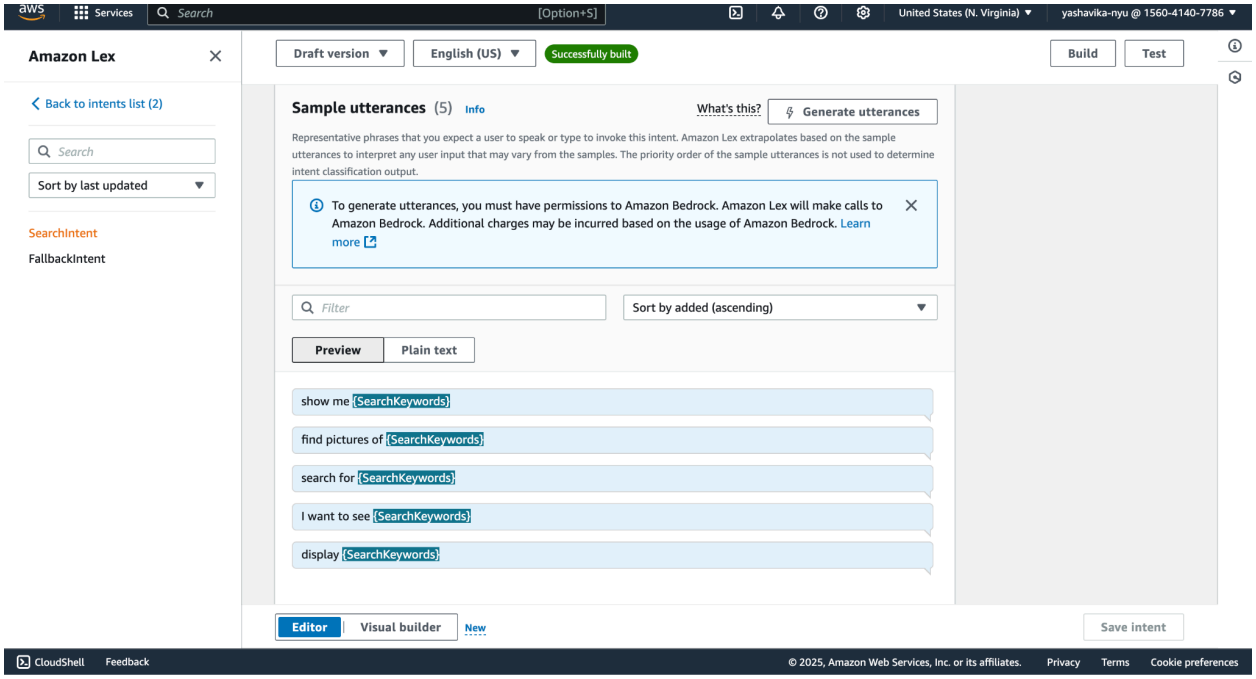
```
(base) yashavikasingh@Yashoos-MacBook-Air ~ % curl -X GET \
  -u 'yashavika:@Alipore1908' \
  'https://search-photos-2zcv2zir5roxmzmvvt7ezajhyi.us-east-1.es.amazonaws.com/photos/_search?pretty'
{
  "took" : 925,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : {
      "value" : 3,
      "relation" : "eq"
    },
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "photos",
        "_id" : "5mmaLJYBnW7q0CiwYwP6",
        "_score" : 1.0,
        "_source" : {
          "objectKey" : "62162d79aa6d24ee1061d4af14d939d4.jpg",
          "bucket" : "photo-bucket-assignment3",
          "createdTimestamp" : "2025-04-13T00:43:57.679822",
          "labels" : [
            "Lifejacket",
            "Person",
            "Face",
            "Kissing",
            "Head",
            "Jacket",
            "Adult",
            "Man",
            "Male",
            "Coat"
          ]
        }
      },
      {
        "_index" : "photos",
        "_id" : "3WmRLJYBnW7q0CiwDANU",
        "_score" : 1.0,
        "_source" : {
          "objectKey" : "me.jpg",
          "bucket" : "photo-bucket-assignment3",
          "createdTimestamp" : "2025-04-13T00:33:37.752575",
          "labels" : [
            "Teen",
            "Shoe",
            "Female",
            "Face",
            "Wheel",
            "Chair",
            "Skirt",
            "Person",
            "Girl",
            "Car"
          ]
```

3. Search

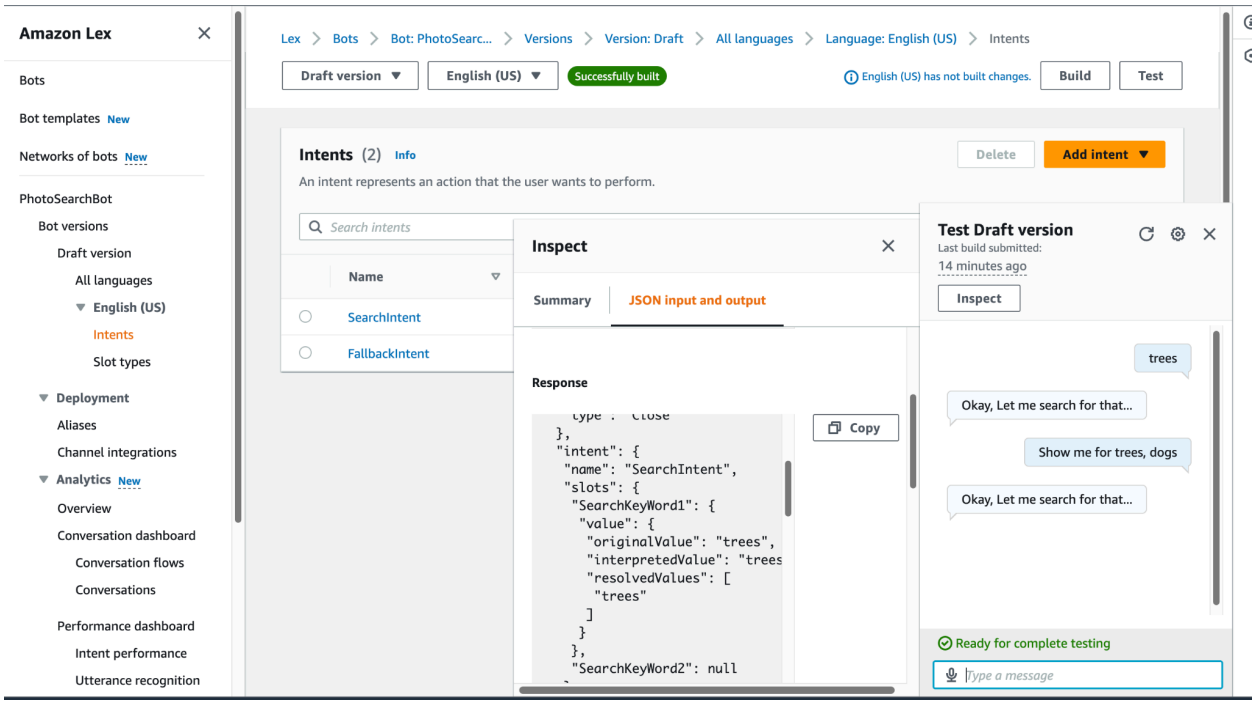
a. Create a Lambda function (**LF2**) called **“search-photos”**.

b. Create an Amazon Lex bot to handle search queries.

i. Create one intent named “SearchIntent”.



ii. Add training utterances to the intent, such that the bot can pick up both keyword searches (“trees”, “birds”), as well as sentence searches (“show me trees”, “show me photos with trees and birds in them”).



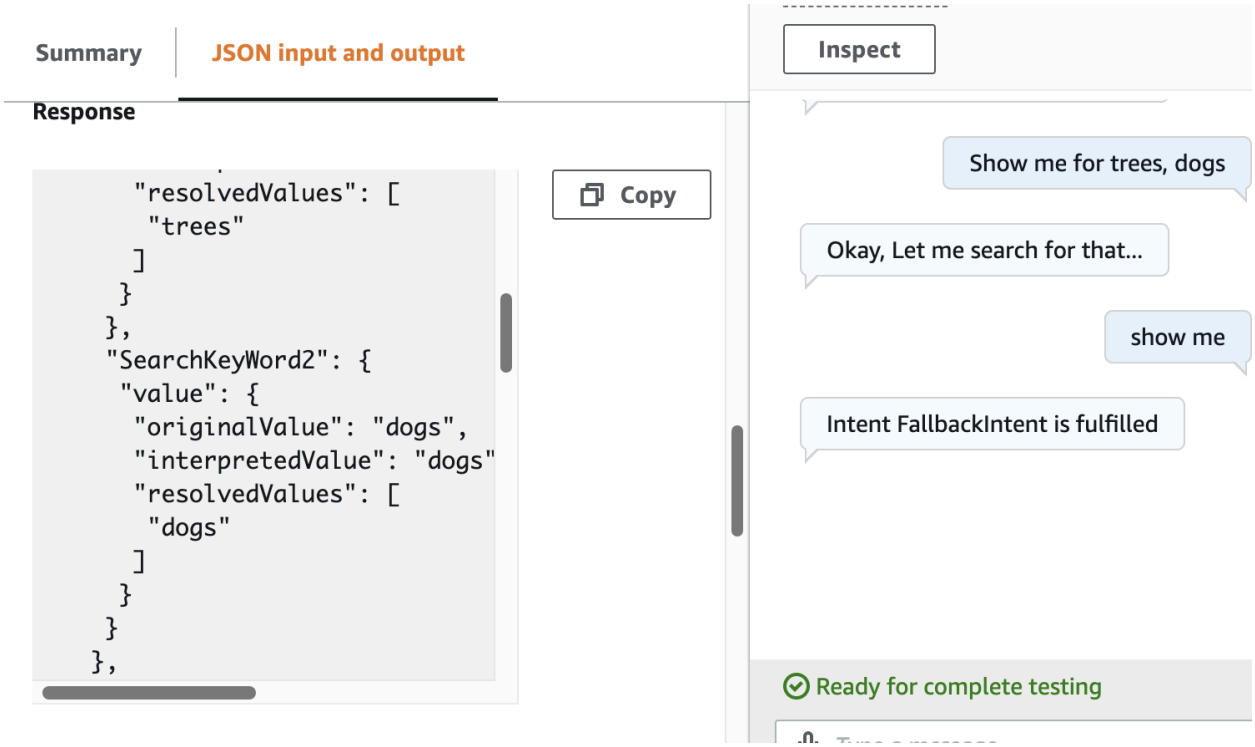
- ***You should be able to handle at least one or two keywords per query.***

c. Implement the Search Lambda function (**LF2**):

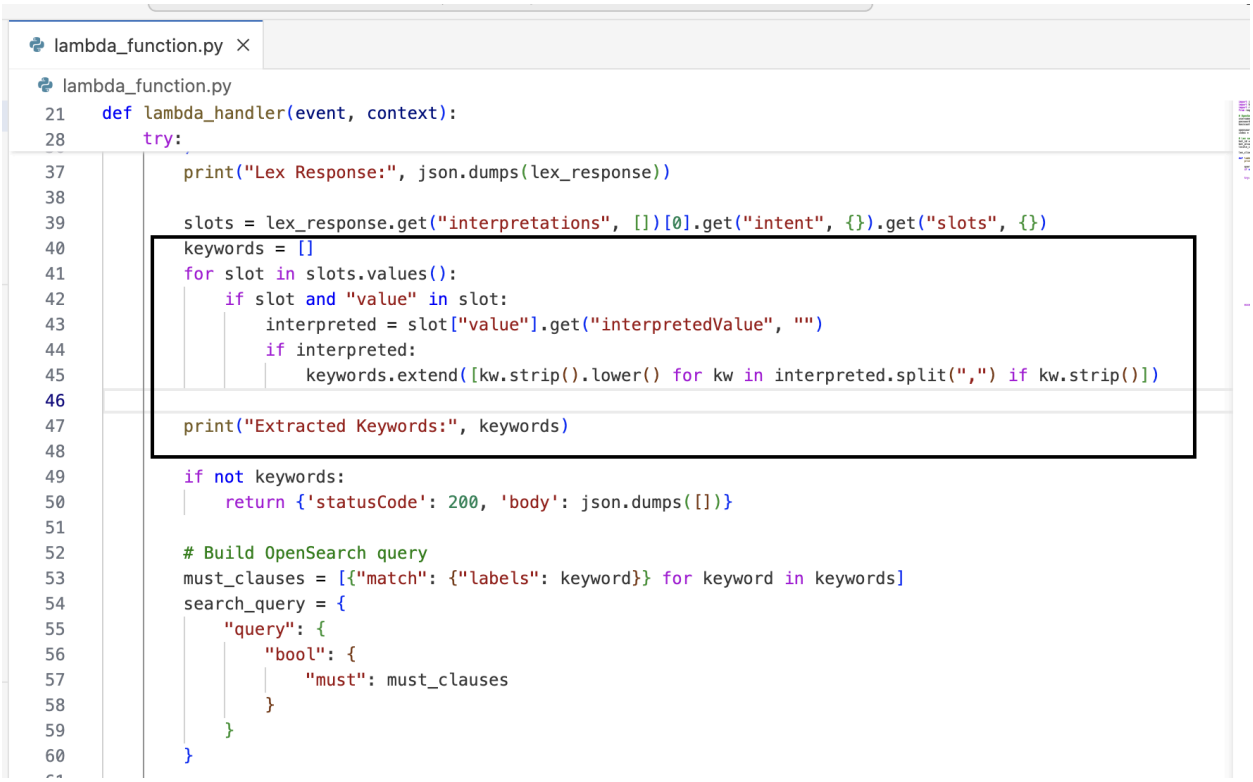
i. Given a search query “q”, disambiguate the query using the Amazon Lex bot.

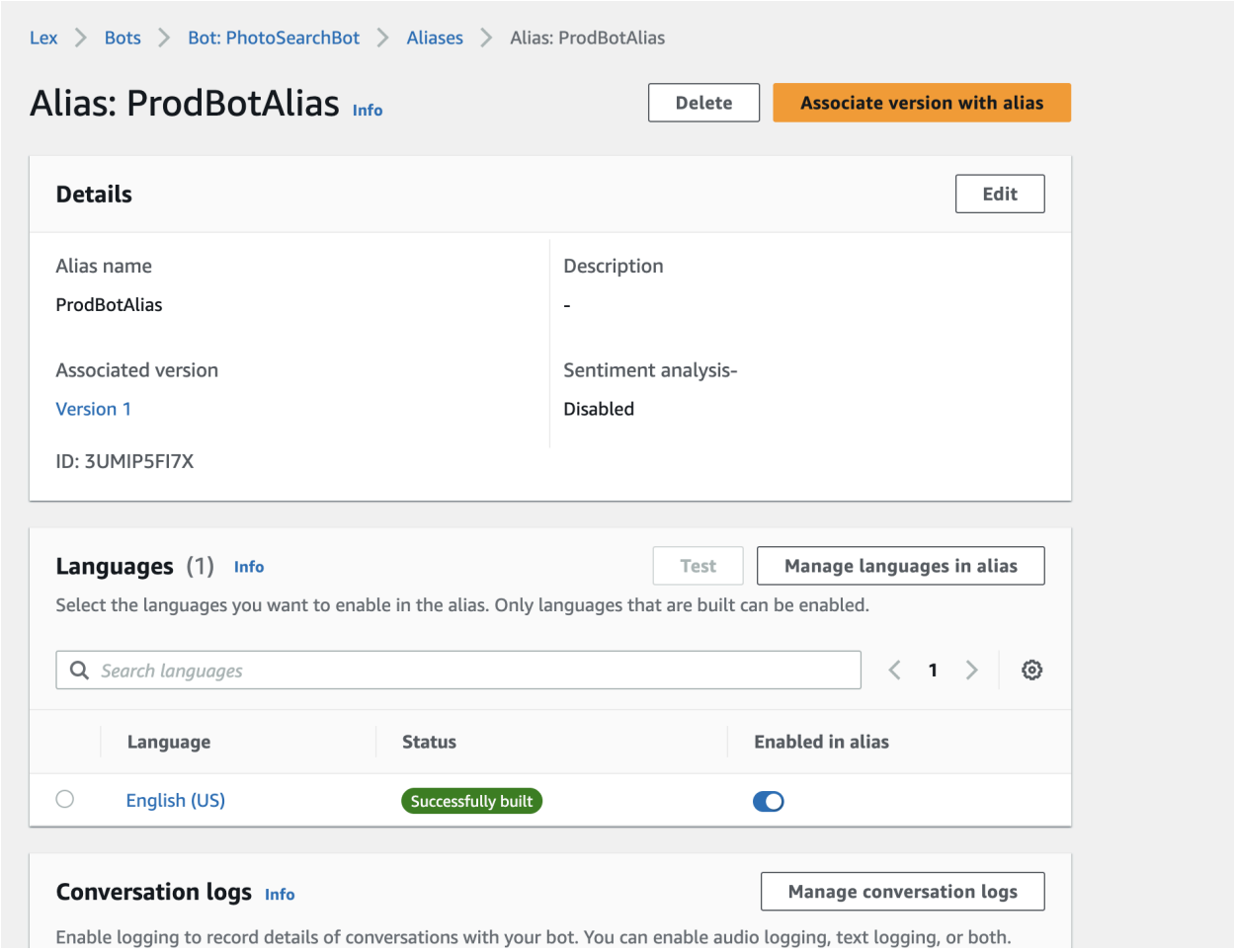

```
{
 "messages": [
  {
   "content": "Okay, Let me search for that...",
   "contentType": "PlainText"
  }
 ],
 "sessionState": {
  "dialogAction": {
   "type": "Close"
  },
  "intent": {
   "name": "SearchIntent",
   "slots": {
    "SearchKeyWord1": {
     "value": {
      "originalValue": "trees",
      "interpretedValue": "trees",
      "resolvedValues": [
       "trees"
      ]
     }
    },
    "SearchKeyWord2": {
     "value": {
      "originalValue": "dogs",
      "interpretedValue": "dogs",
      "resolvedValues": [
       "dogs"
      ]
     }
    }
   },
   "state": "ReadyForFulfillment",
   "confirmationState": "None"
  },
  "sessionAttributes": {},
  "originatingRequestId": "8885276e-aa57-403f-a9fb-07eed0b34985"
 },
 "interpretations": [
  {
   "nluConfidence": {
    "score": 0.94
   },
   "intent": {
    "name": "SearchIntent",
```


ii. If the Lex disambiguation request yields any keywords (K1 ….Kn) search the “photos” ElasticSearch index for results, and return the accordingly (as per the API spec). You should look for ElasticSearch SDK libraries to perform the search.



iii. Otherwise, return an empty array of results (as per the API spec).


lambda_function.py
lambda_function.py
def lambda_handler(event, context):
try:
print("Lex Response:", json.dumps(lex_response))
slots = lex_response.get("interpretations", [])[0].get("intent", {}).get("slots", {})
keywords = []
for slot in slots.values():
if slot and "value" in slot:
interpreted = slot["value"].get("interpretedValue", "")
if interpreted:
keywords.extend([kw.strip().lower() for kw in interpreted.split(",") if kw.strip()])
print("Extracted Keywords:", keywords)
if not keywords:
return {'statusCode': 200, 'body': json.dumps([])}
# Build OpenSearch query
must_clauses = [{"match": {"labels": keyword}} for keyword in keywords]
search_query = {
"query": {
"bool": {
"must": must_clauses
}
}
}

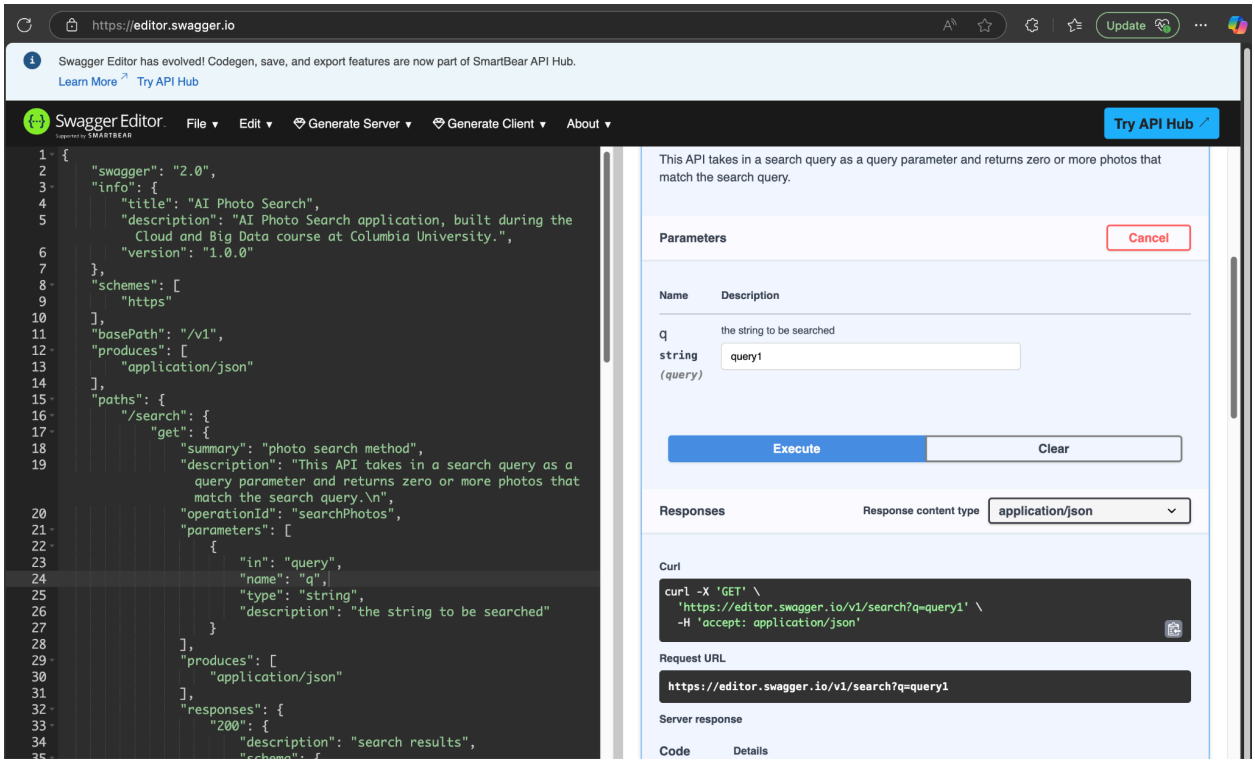


4. Build the API layer

a. Build an API using APIGateway.

i. The Swagger API documentation for the API can be found here: https://github.com/001000001/ai-photo-search-columbia-f2018/blob/master/swagger.yaml

Build is successful for SPECS:

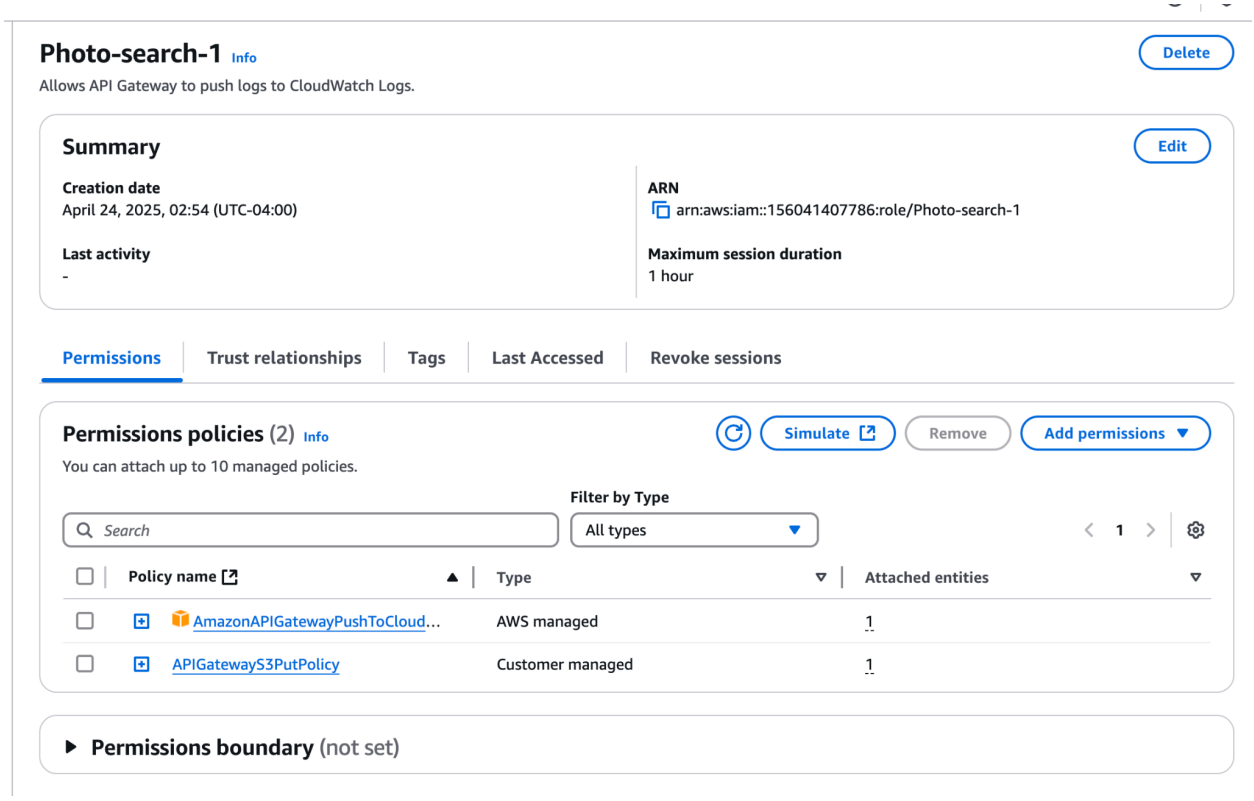


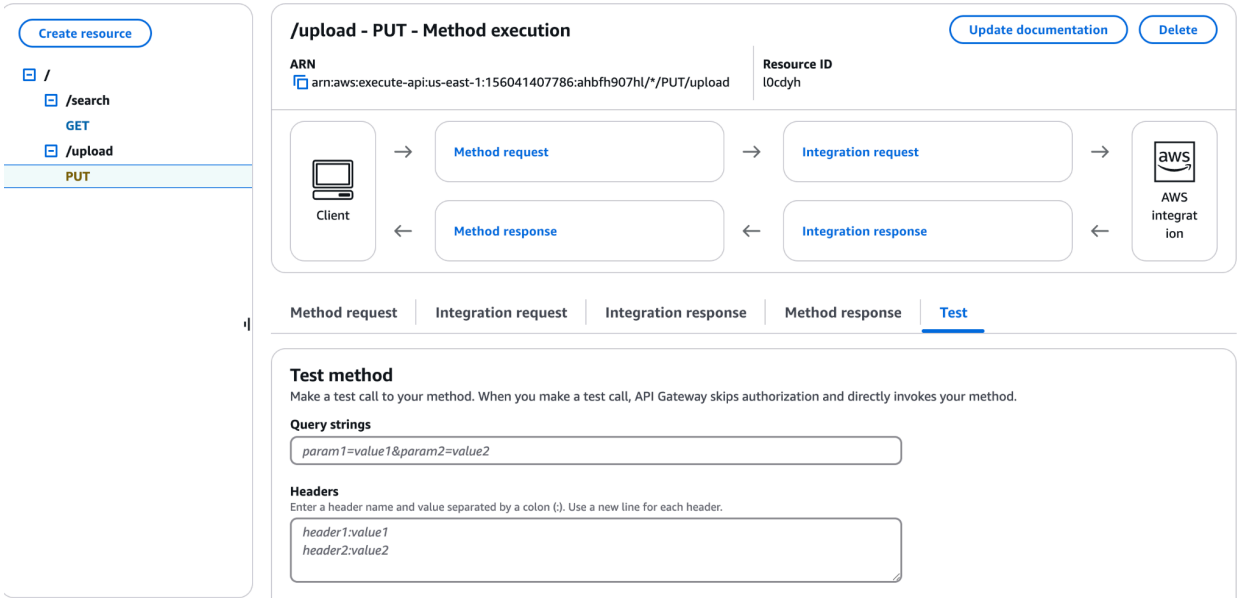
b. The API should have two methods:

i. PUT /photos

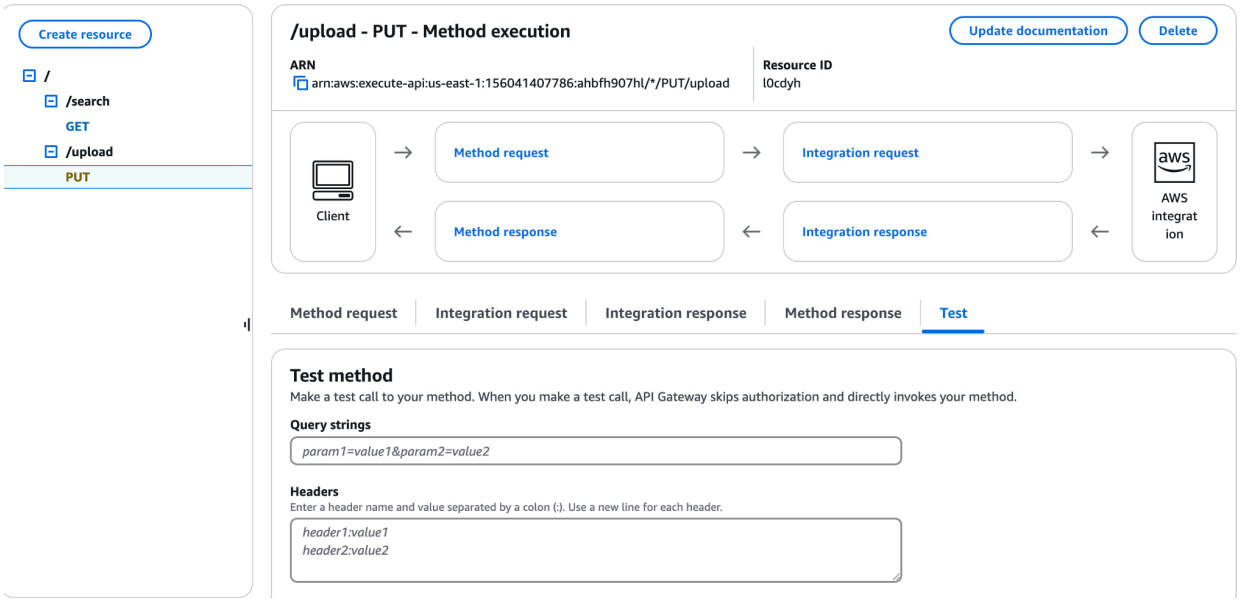
Set up the method as an Amazon S3 Proxy8. This will allow API Gateway to forward your PUT request directly to S3.

Creating the Role and Policy:



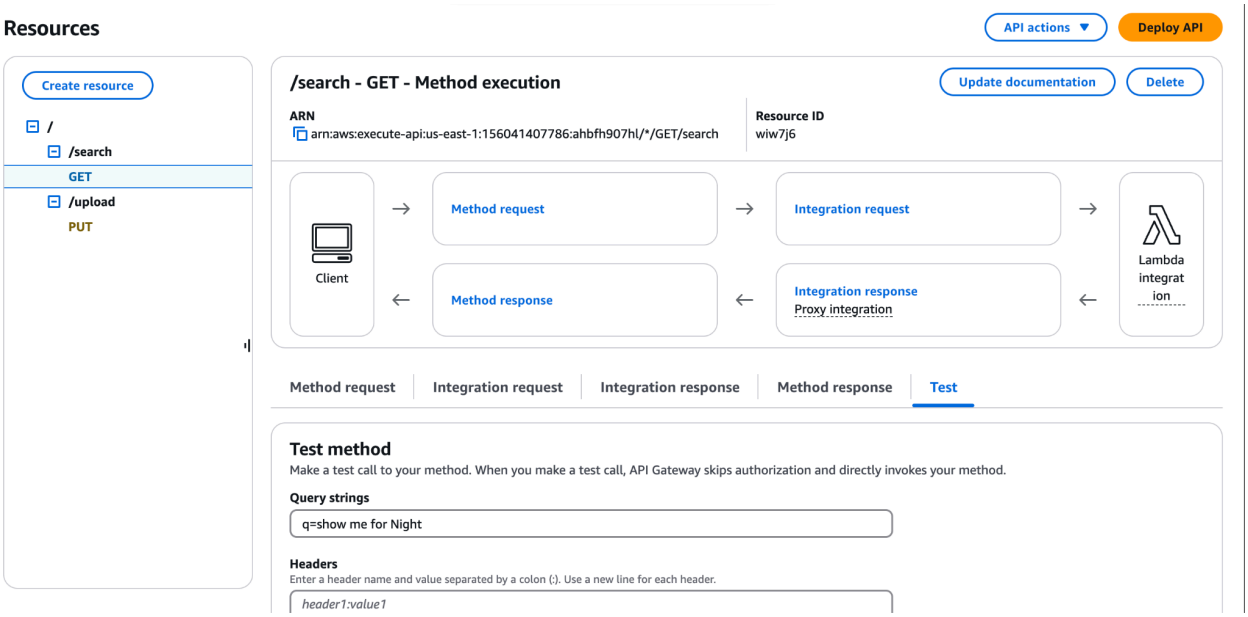


- Use a custom **x-amz-meta-customLabels** HTTP header to include any custom labels the user specifies at upload time.

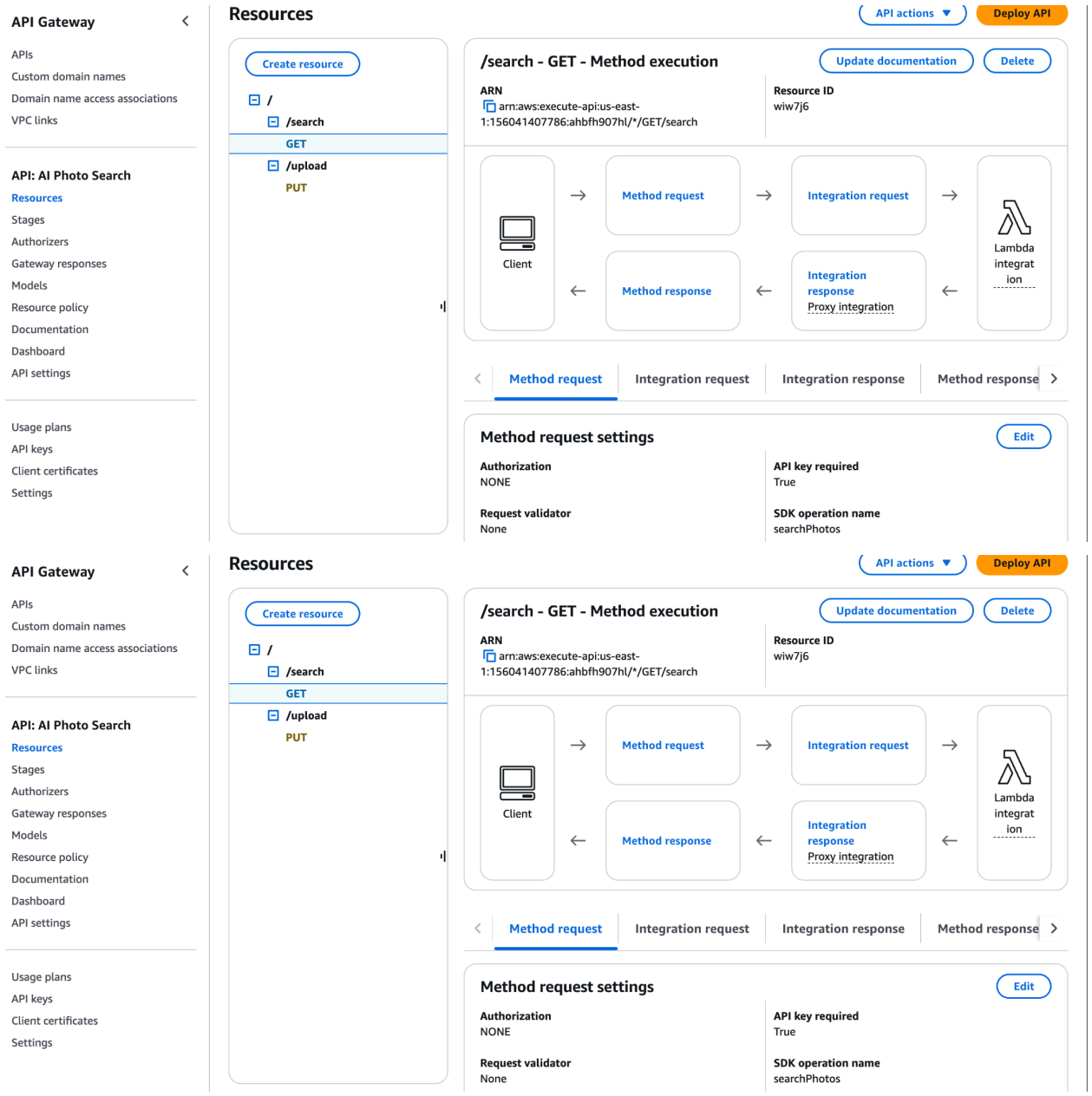


ii. GET /search?q={query text}

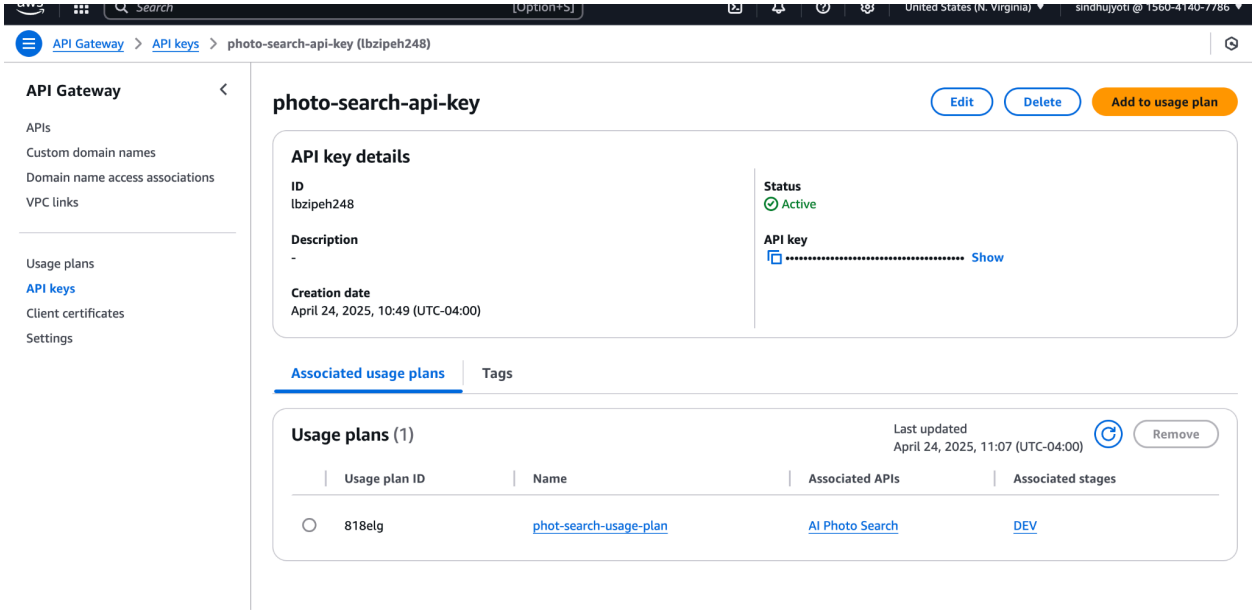
Connect this method to the search Lambda function (**LF2**).



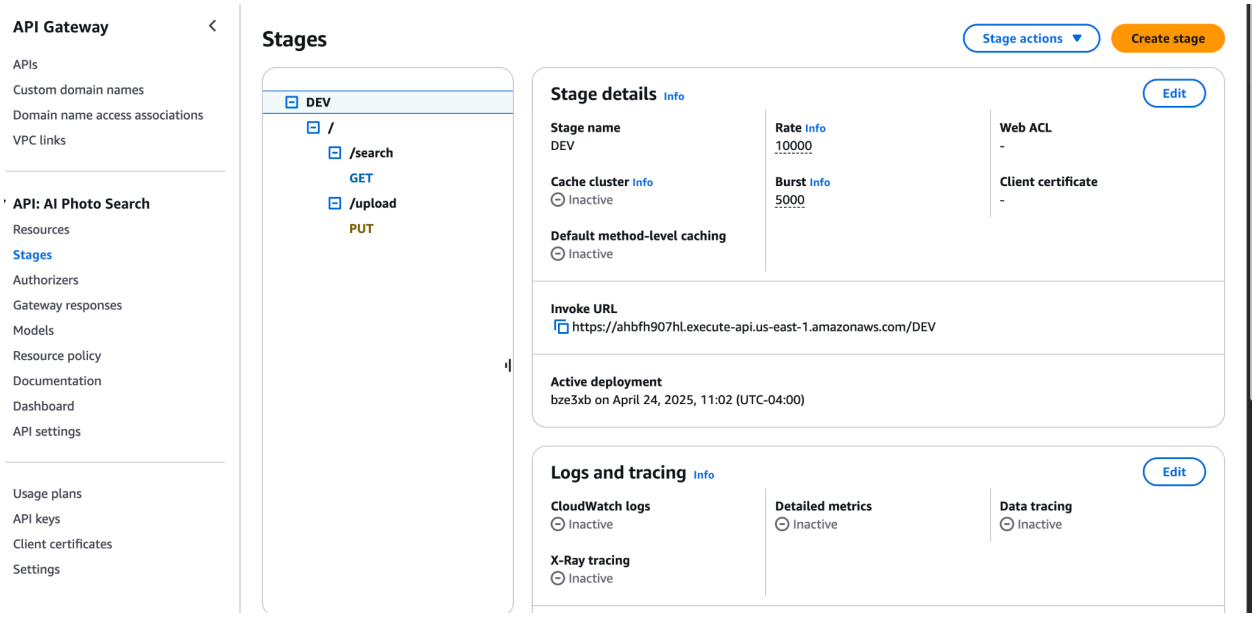
c. Setup an API key for your two API methods.



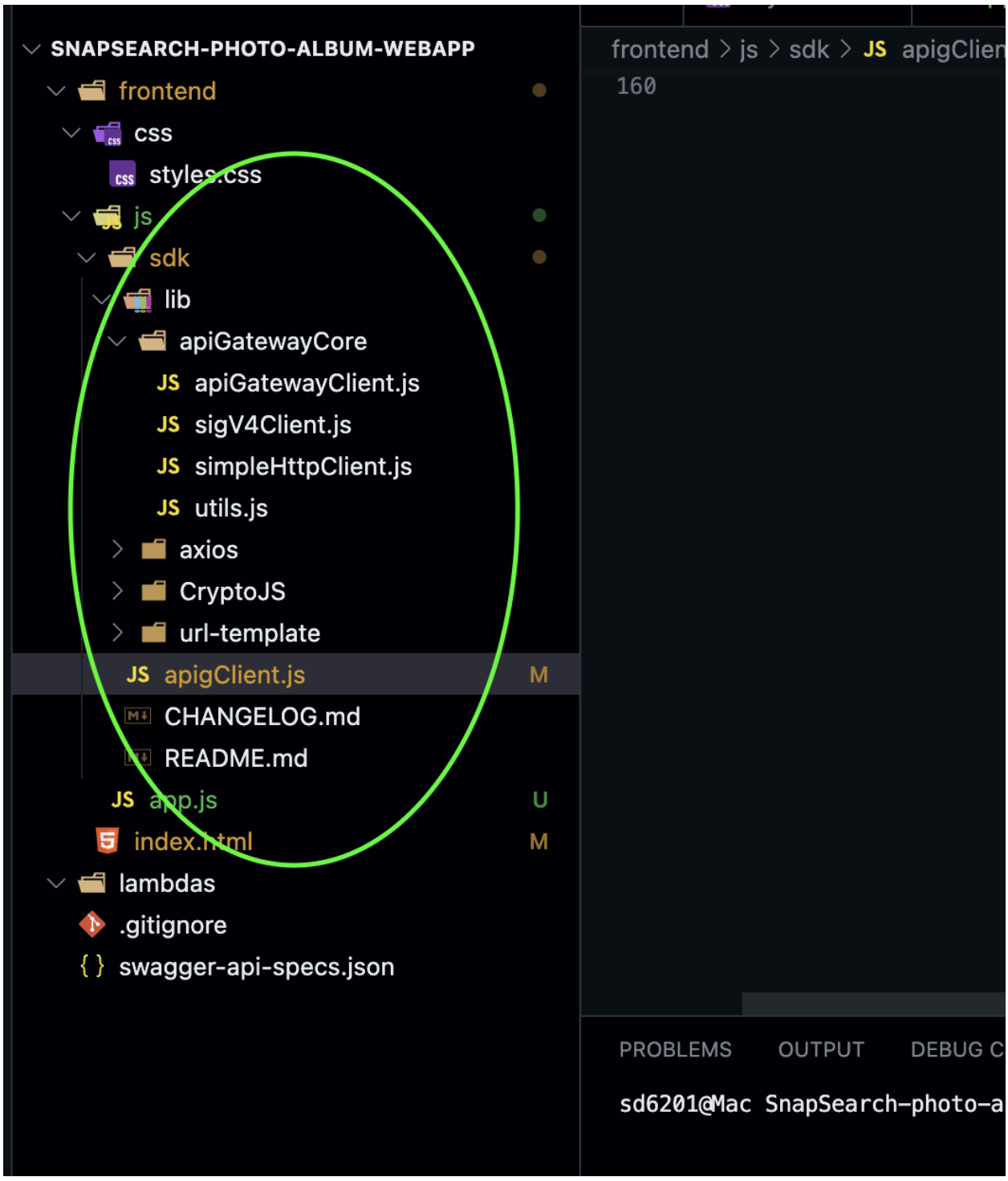
API Gateway
APIs
Custom domain names
Domain name access associations
VPC links
API: AI Photo Search
Resources
Stages
Authorizers
Gateway responses
Models
Resource policy
Documentation
Dashboard
API settings
Usage plans
API keys
Client certificates
Settings
Resources
API actions
Deploy API
Create resource
/
/search
GET
/upload
PUT
/search - GET - Method execution
Update documentation
Delete
ARN
arn:aws:execute-api:us-east-1:156041407786:ahbfh907hl/*/GET/search
Resource ID
wiw7j6
Client
Method request
Integration request
Lambda integration
Method response
Integration response
Proxy integration
Method request
Integration request
Integration response
Method response
Method request settings
Edit
Authorization
NONE
API key required
True
Request validator
None
SDK operation name
searchPhotos



d. Deploy the API.



e. Generate a SDK for the API (**SDK1**).

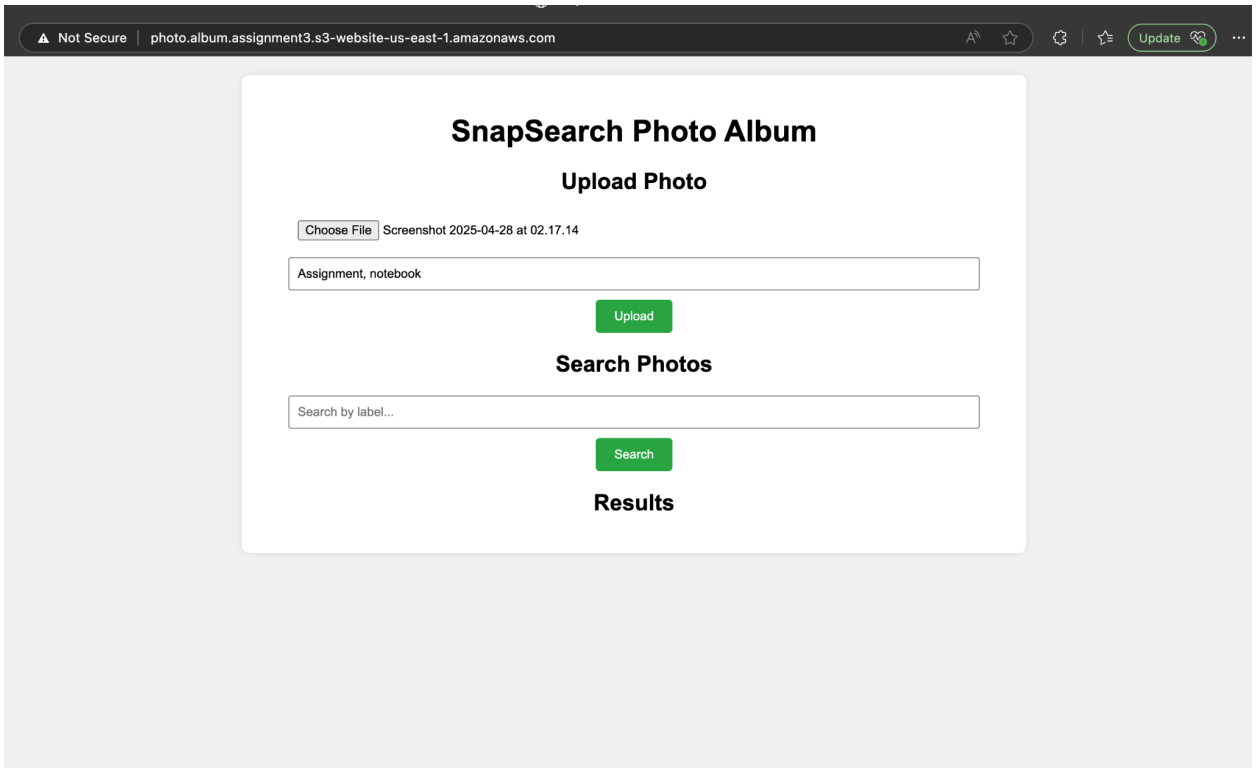


5. Frontend

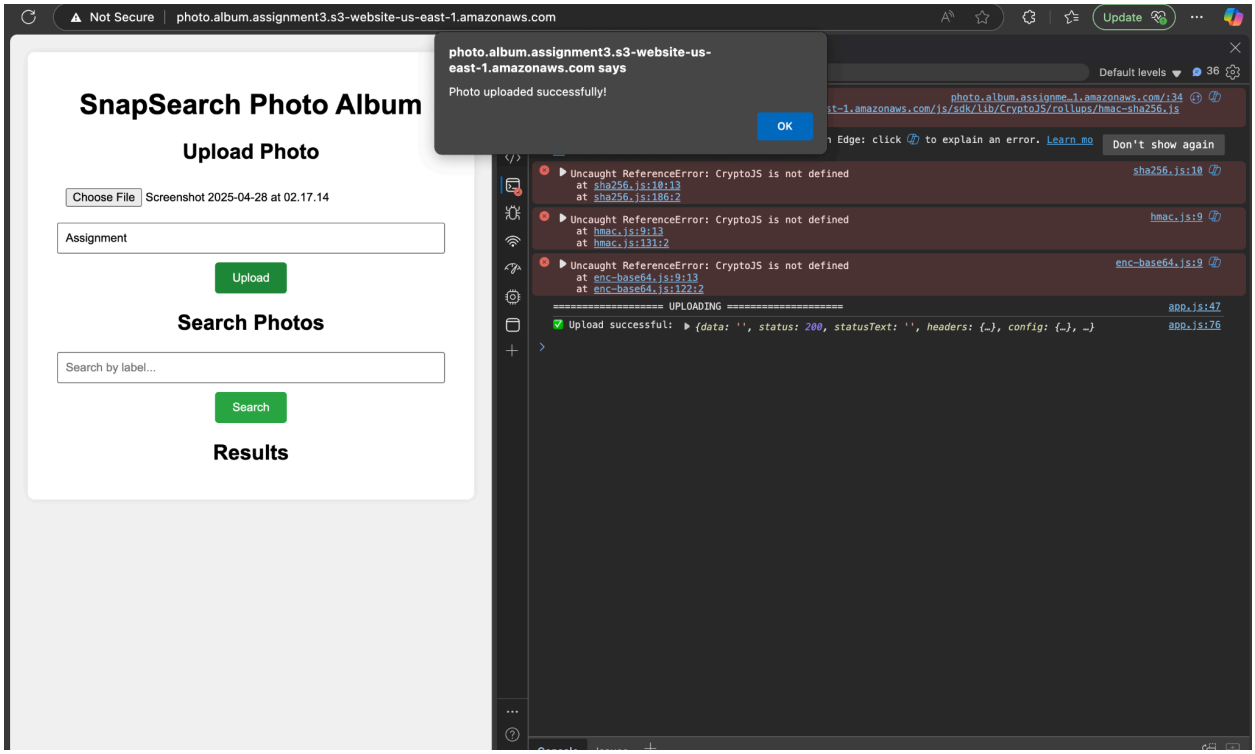
a. Build a simple frontend application that allows users to:

i. Make search requests to the GET /search endpoint
ii. Display the results (photos) resulting from the query
iii. Upload new photos using the PUT /photos

Frontend:



PUT functionality:


Not Secure
photo.album.assignment3.s3-website-us-east-1.amazonaws.com
Update
SnapSearch Photo Album
Upload Photo
Choose File
Screenshot 2025-04-28 at 02.17.14
Assignment
Upload
Search Photos
Search by label...
Search
Results
photo.album.assignment3.s3-website-us-east-1.amazonaws.com says
Photo uploaded successfully!
OK
Default levels
36
Don't show again
Uncaught ReferenceError: CryptoJS is not defined
at sha256.js:10:13
at sha256.js:186:2
sha256.js:10
Uncaught ReferenceError: CryptoJS is not defined
at hmac.js:9:13
at hmac.js:131:2
hmac.js:9
Uncaught ReferenceError: CryptoJS is not defined
at enc-base64.js:9:13
at enc-base64.js:122:2
enc-base64.js:9
UPLOADING
app.js:47
Upload successful:
app.js:76

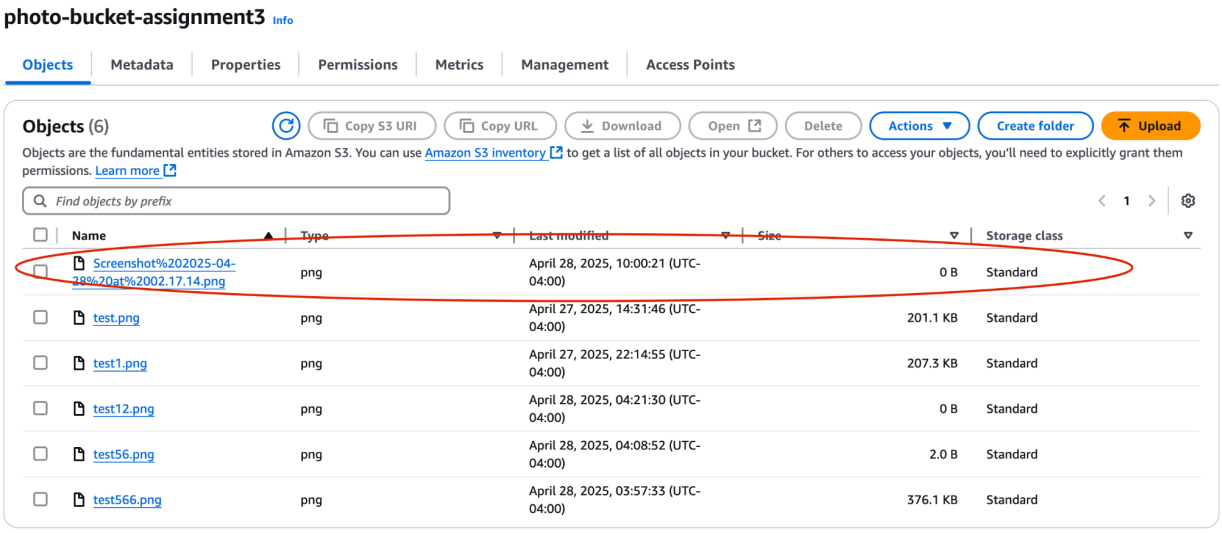
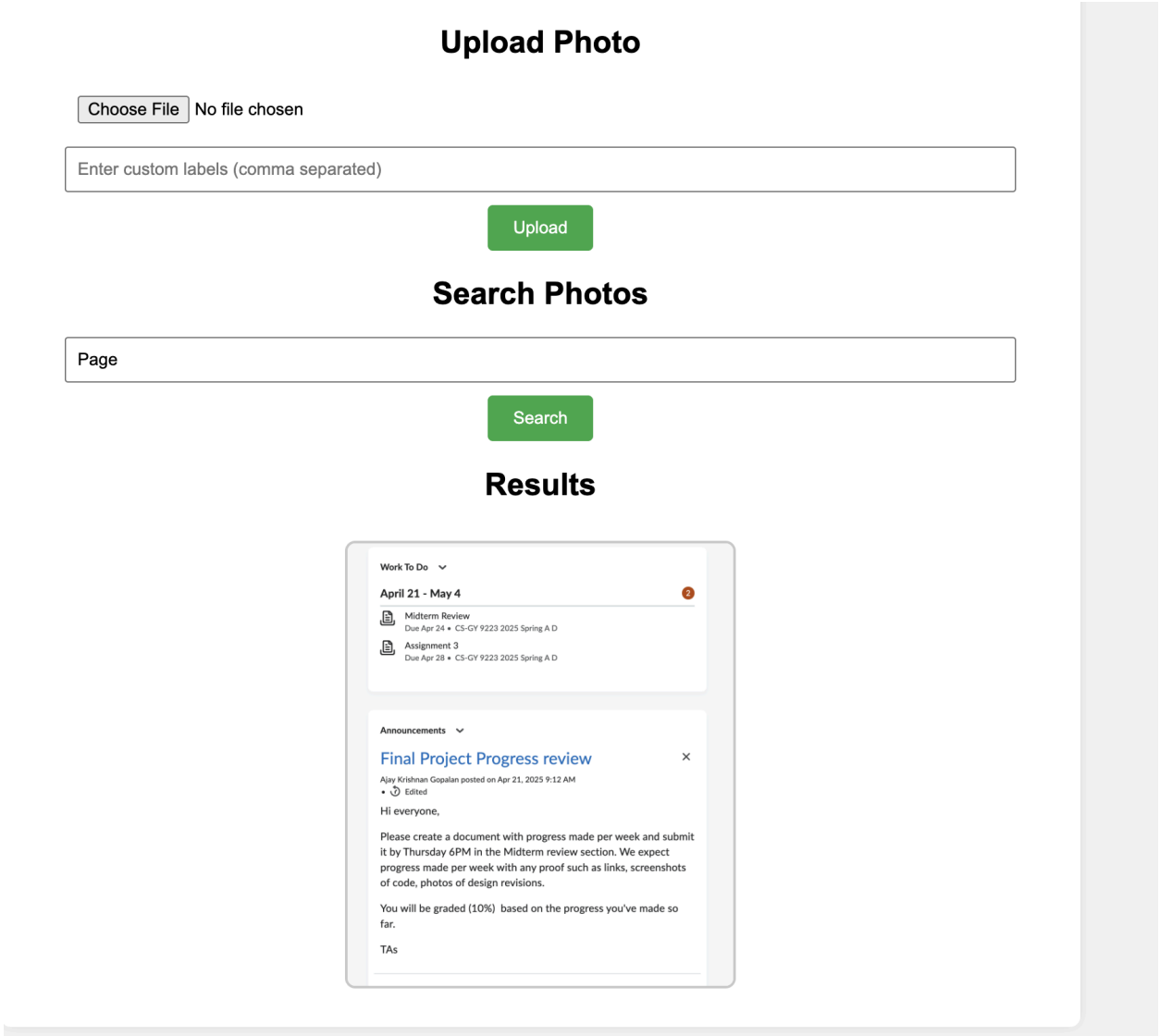


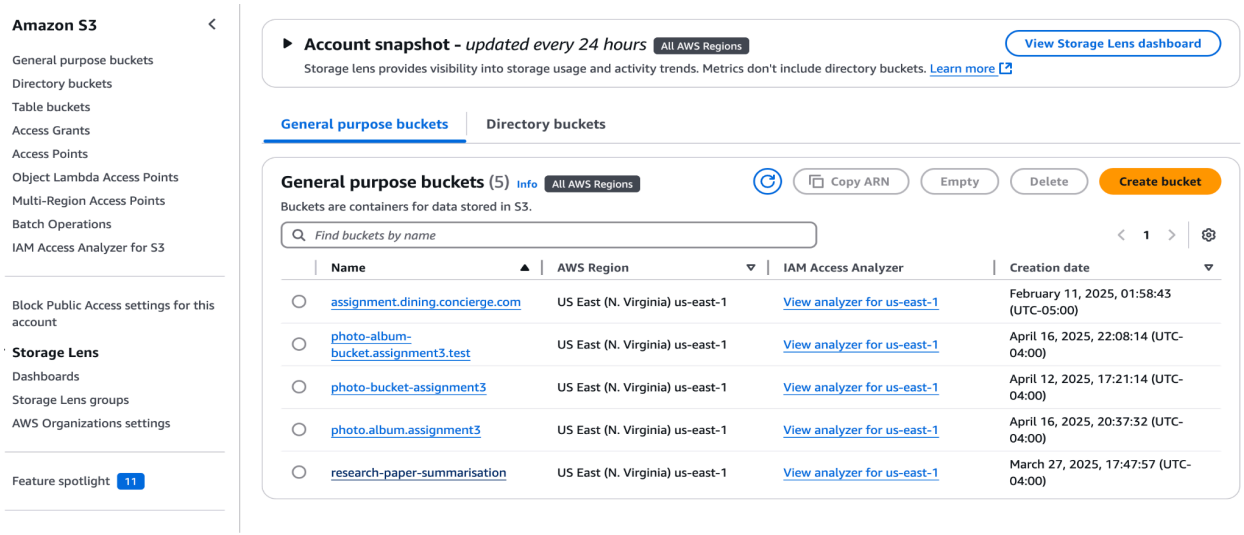
photo-bucket-assignment3
Info
Objects
Metadata
Properties
Permissions
Metrics
Management
Access Points
Objects (6)
Copy S3 URI
Copy URL
Download
Open
Delete
Actions
Create folder
Upload
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more
Find objects by prefix
Name
Type
Last modified
Size
Storage class
Screenshot%202025-04-28%20at%2002.17.14.png
png
April 28, 2025, 10:00:21 (UTC-04:00)
0 B
Standard
test.png
png
April 27, 2025, 14:31:46 (UTC-04:00)
201.1 KB
Standard
test1.png
png
April 27, 2025, 22:14:55 (UTC-04:00)
207.3 KB
Standard
test12.png
png
April 28, 2025, 04:21:30 (UTC-04:00)
0 B
Standard
test56.png
png
April 28, 2025, 04:08:52 (UTC-04:00)
2.0 B
Standard
test566.png
png
April 28, 2025, 03:57:33 (UTC-04:00)
376.1 KB
Standard

Search:

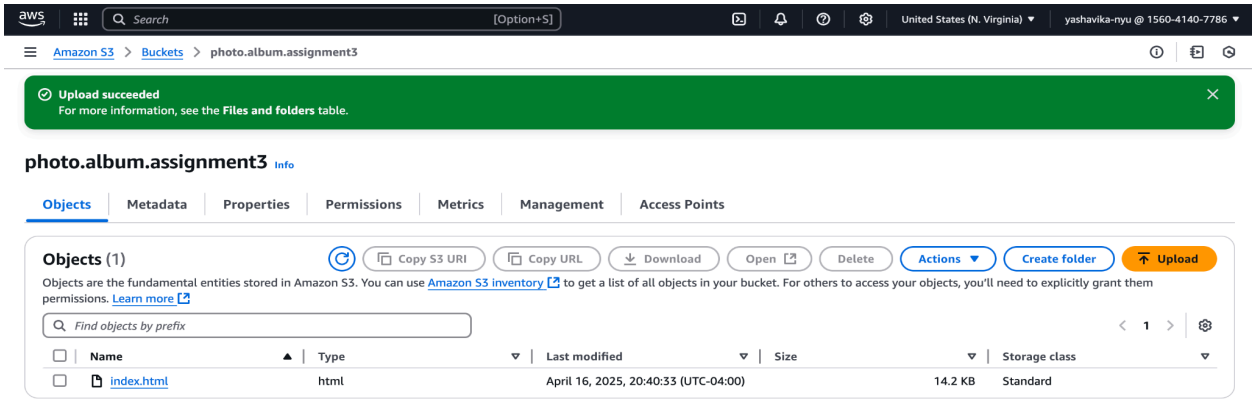


● *In the upload form, allow the user to specify one or more custom labels, that will be appended to the list of labels detected automatically by Rekognition (see 2.d.iii above). These custom labels should be converted to a comma-separated list and uploaded as part of the S3 object's metadata using a x-amz-meta-customLabels metadata HTTP header.*
For instance, if you specify two custom labels at upload time, "Sam" and "Sally", the metadata HTTP header should look like: 'x-amz-meta-customLabels': 'Sam, Sally'
https://docs.aws.amazon.com/apigateway/latest/developerguide/integrating-api-with-aws-services-s3.html

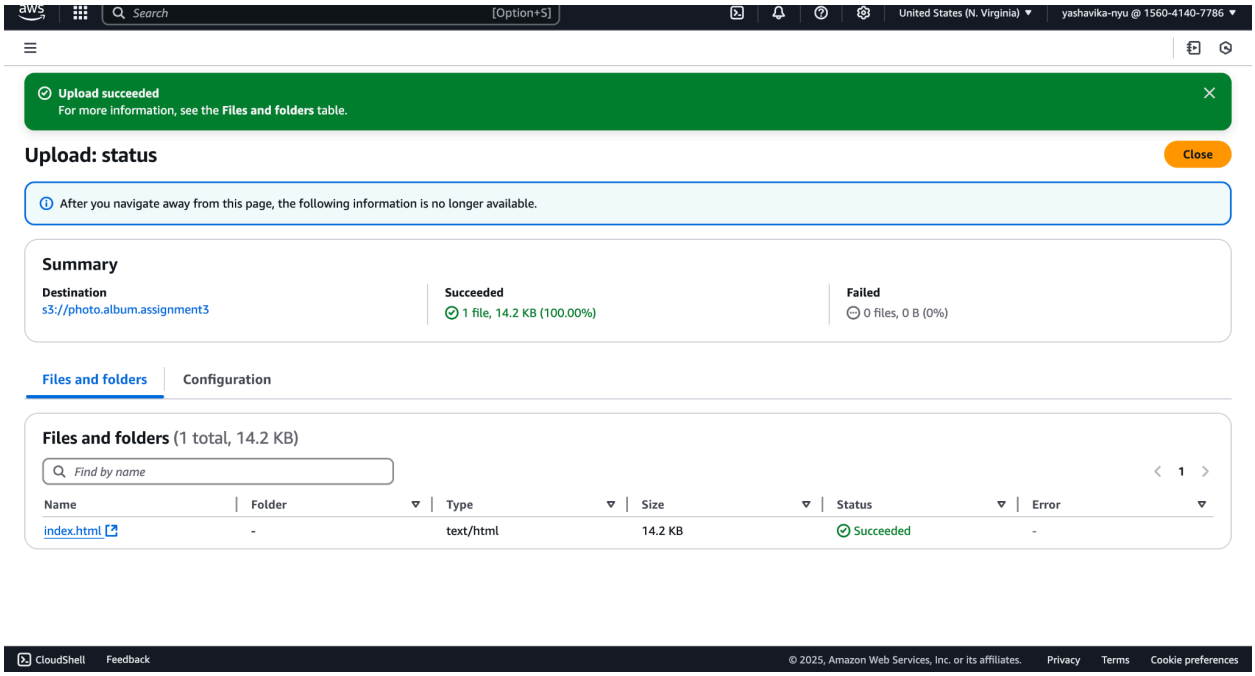
b. Create a S3 bucket for your frontend (**B1**).



c. Set up the bucket for static website hosting (same as HW1)

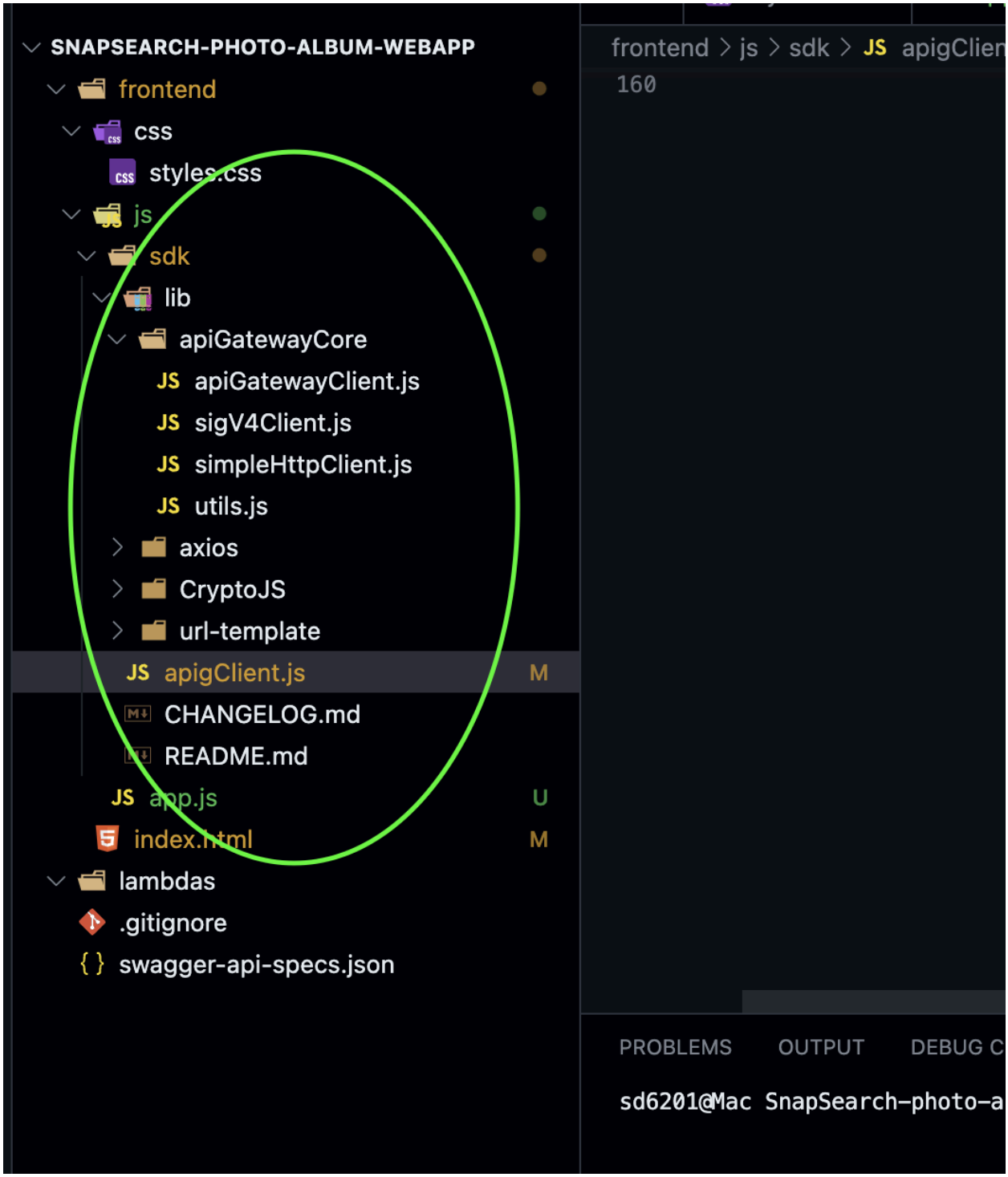


d. Upload the frontend files to the bucket (**B2**).



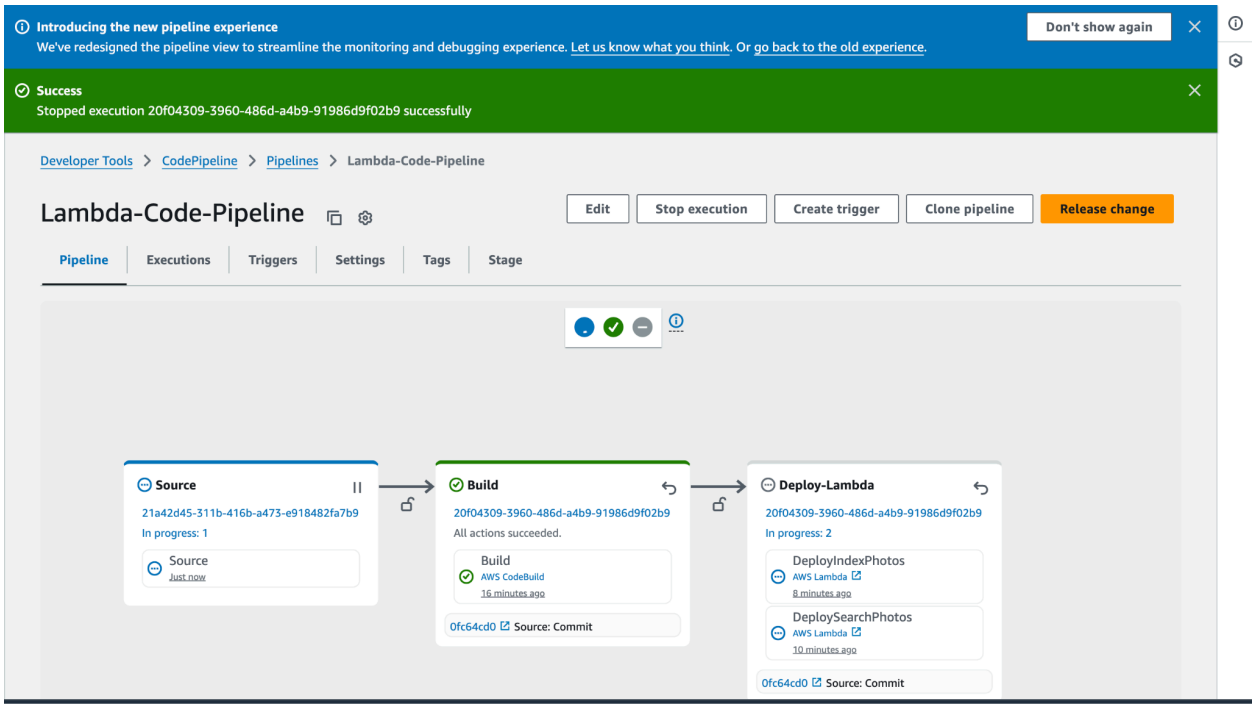
https://s3.us-east-1.amazonaws.com/photo.album.assignment3/index.html

e. Integrate the API Gateway-generated SDK (**SDK1**) into the frontend, to connect your API.



6. Deploy your code using AWS CodePipeline

a. Define a pipeline (P1) in AWS CodePipeline that builds and deploys the code for/to all your Lambda functions.



Developer Tools > CodePipeline > Pipelines > Lambda-Code-Pipeline

Lambda-Code-Pipeline

← Overview

▸ **Pipeline execution details**

Logs | Summary | Input | Output

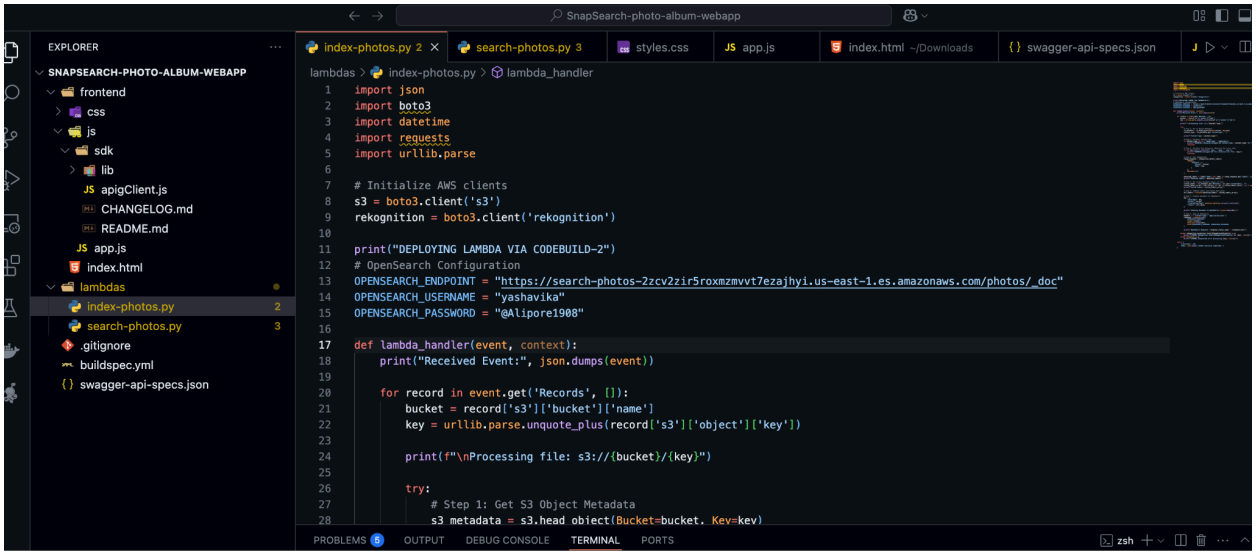
Showing the last 56 lines of the build log. View entire log

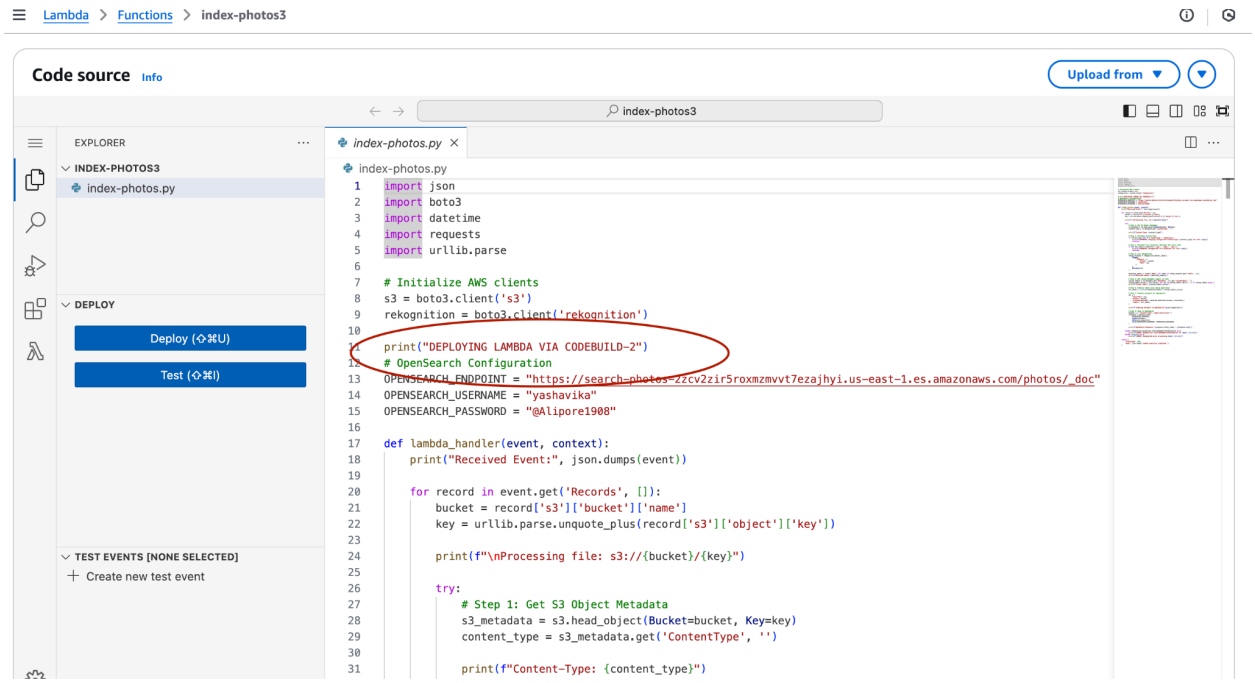
Tail logs

No previous logs

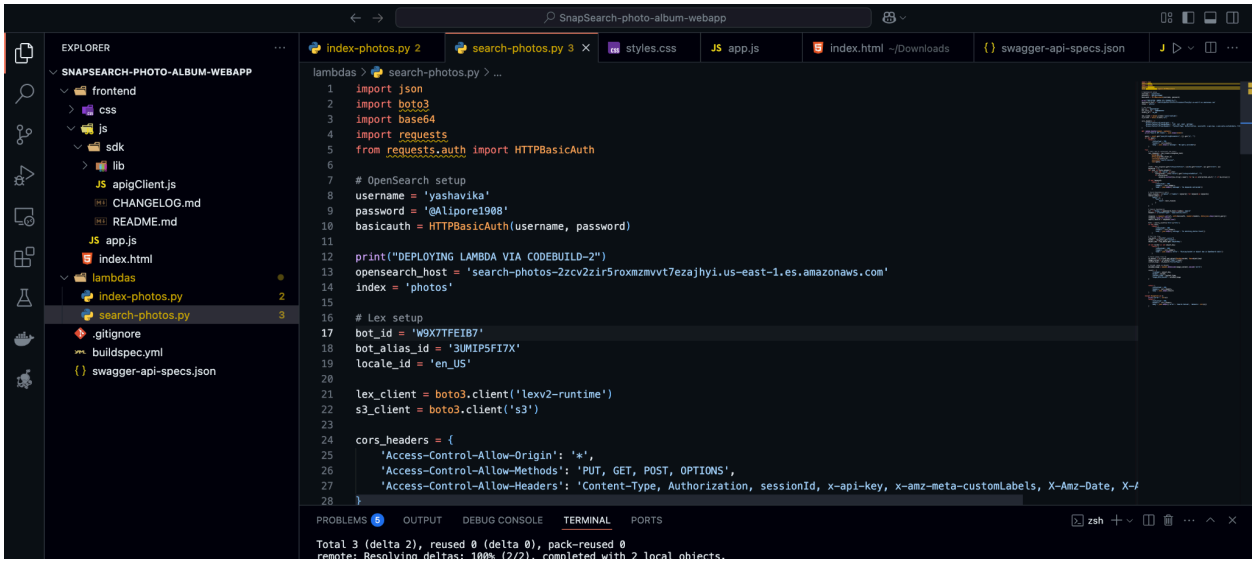
```
1  [Container] 2025/04/28 21:36:11.315264 Running on CodeBuild On-demand
2  [Container] 2025/04/28 21:36:11.315274 Waiting for agent ping
3  [Container] 2025/04/28 21:36:11.516548 Waiting for DOWNLOAD_SOURCE
4  [Container] 2025/04/28 21:36:12.534335 Phase is DOWNLOAD_SOURCE
5  [Container] 2025/04/28 21:36:12.535657 CODEBUILD_SRC_DIR=/codebuild/output/src3498283727/src
6  [Container] 2025/04/28 21:36:12.536273 YAML location is /codebuild/output/src3498283727/src/buildspec.yml
7  [Container] 2025/04/28 21:36:12.538217 Setting HTTP client timeout to higher timeout for S3 source
8  [Container] 2025/04/28 21:36:12.538498 Processing environment variables
9  [Container] 2025/04/28 21:36:12.775867 No runtime version selected in buildspec.
10 [Container] 2025/04/28 21:36:12.791433 Moving to directory /codebuild/output/src3498283727/src
11 [Container] 2025/04/28 21:36:12.791456 Cache is not defined in the buildspec
12 [Container] 2025/04/28 21:36:12.828793 Skip cache due to: no paths specified to be cached
13 [Container] 2025/04/28 21:36:12.829164 Registering with agent
14 [Container] 2025/04/28 21:36:12.864006 Phases found in YAML: 1
15 [Container] 2025/04/28 21:36:12.864024  BUILD: 3 commands
16 [Container] 2025/04/28 21:36:12.864316 Phase complete: DOWNLOAD_SOURCE State: SUCCEEDED
17 [Container] 2025/04/28 21:36:12.864329 Phase context status code:  Message:
18 [Container] 2025/04/28 21:36:12.940652 Entering phase INSTALL
19 [Container] 2025/04/28 21:36:12.978190 Phase complete: INSTALL State: SUCCEEDED
20 [Container] 2025/04/28 21:36:12.978208 Phase context status code:  Message:
```

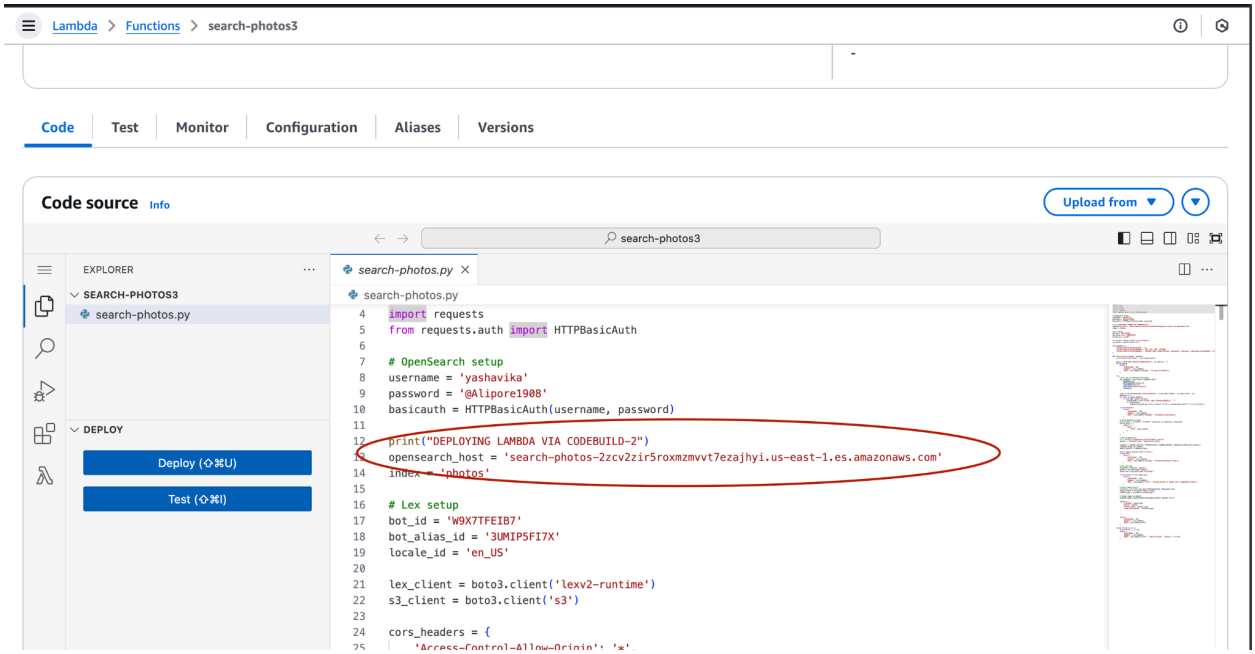
Changes in the index-photos3:



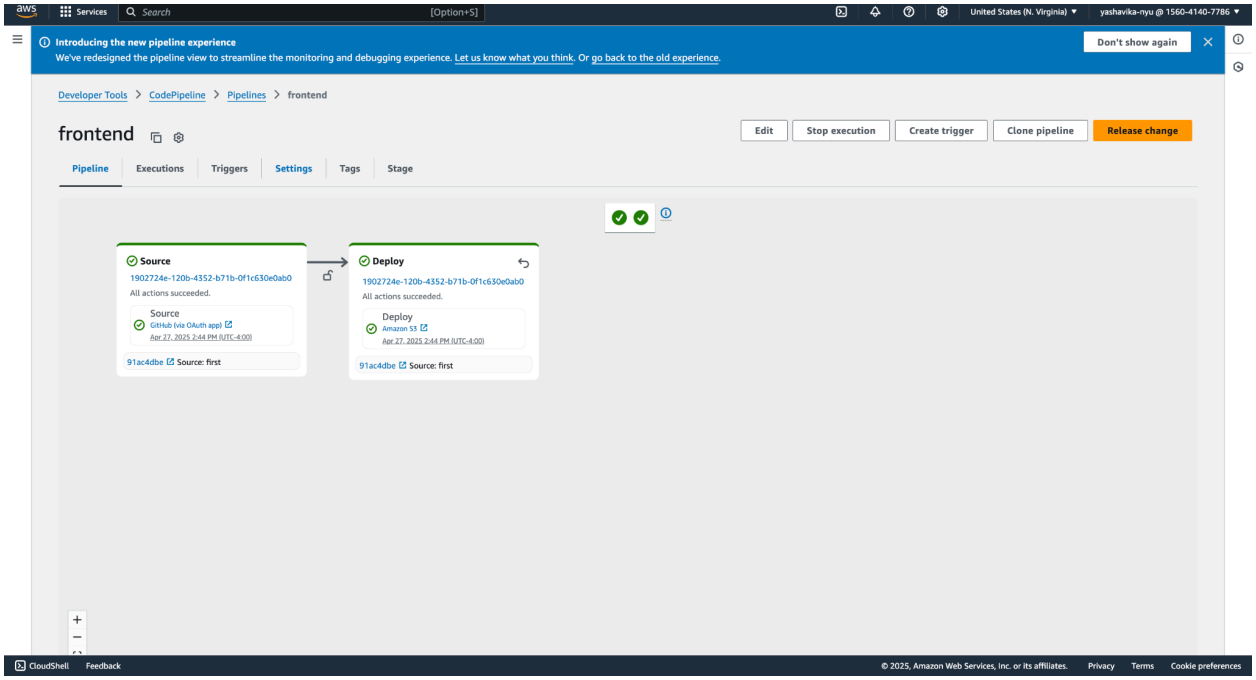


Changes in the search-photos3:



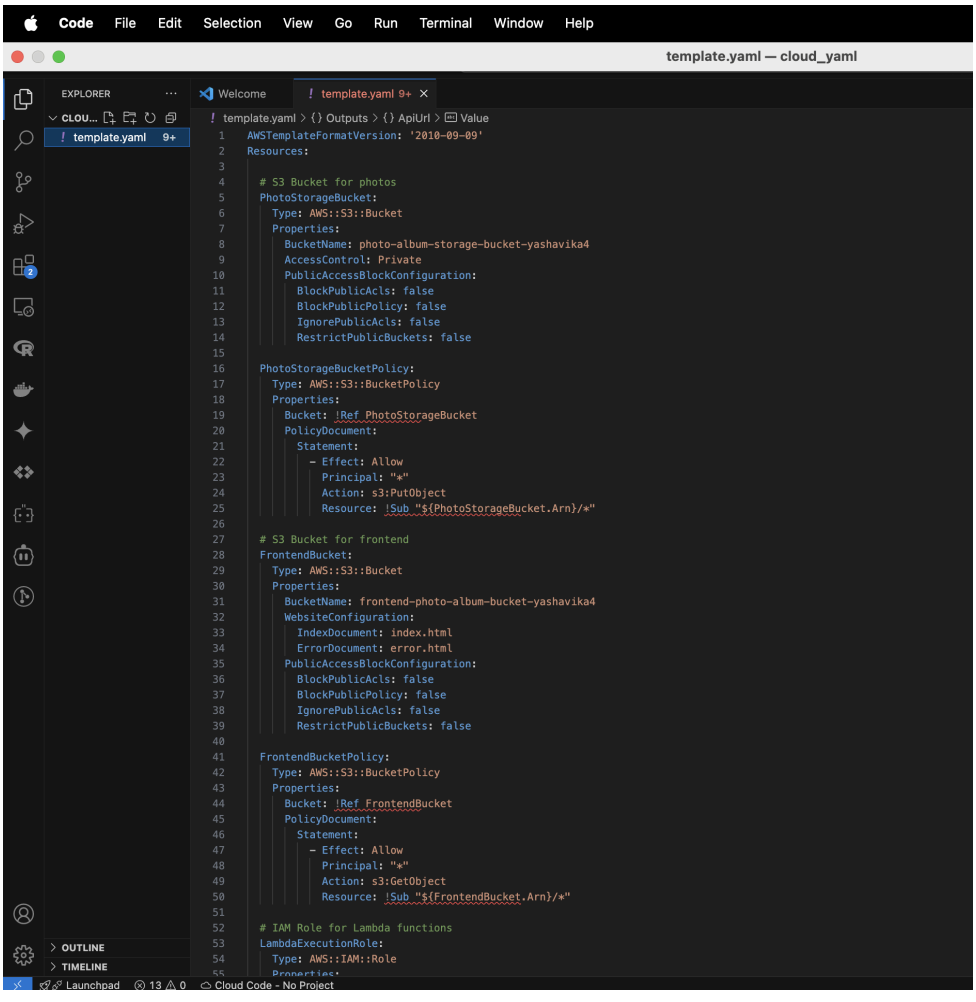


b. Define a pipeline (P2) in AWS CodePipeline that builds and deploysyour frontend code to its corresponding S3 bucket.



7. Create a AWS CloudFormation12 template for the stack

template.yaml



Spinned up resources: