

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 trigger ⁴ 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 deploys your 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”

The screenshot shows the AWS OpenSearch Service dashboard. On the left, there's a sidebar with navigation links for 'Central management', 'Managed clusters', and 'Serverless'. The main area displays 'Domain health' with a summary: 1 Red, 0 Yellow, and 1 Green. Below this, a table lists domains. The 'photos' domain is listed with the following details:

Name	Domain process...	Change status	Engine	Version	Cluster health	Total free space	Minimum free space
photos	Active	Completed	OpenSearch	2.17	Green	7.73 GiB	7.73 GiB

At the bottom of the table, there's a link 'View all domains'.

2. Upload & index photos

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

The screenshot shows the AWS S3 console with the path [Amazon S3 > Buckets > photo-bucket-assignment3](#). The bucket name is **photo-bucket-assignment3**. The main interface displays a table with the heading "Objects (0)". A prominent orange "Upload" button is located at the bottom right of the table area. The top navigation bar includes tabs for Objects, Metadata, Properties, Permissions, Metrics, Management, and Access Points. The status bar at the bottom indicates "No objects" and "You don't have any objects in this bucket".

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

The screenshot shows the AWS Lambda console with the path [Lambda > Functions > index-photos](#). The function name is **index-photos**. The configuration tab is selected. On the left, a sidebar lists various configuration options: Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, RDS databases, Monitoring and operations tools, Concurrency and recursion detection, Asynchronous invocation, Code signing, File systems, and State machines. The main panel displays the "General configuration" section with the following details:

- Description: None
- Memory: 128 MB
- Timeout: 0 min 3 sec
- SnapStart: None
- Ephemeral storage: 512 MB

An "Edit" button is located in the top right corner of this section. Other tabs available include Code, Test, Monitor, Configuration, Aliases, and Versions.

- 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.

The screenshot shows the 'Event notifications' section with one entry. The trigger is named 'TriggerIndexing' and is set to respond to 'Put' events. The destination is a Lambda function named 'index-photos'. There is also a section for 'Amazon EventBridge' which is currently off.

- 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:

This screenshot is identical to the one above, showing the 'Event notifications' section with the 'TriggerIndexing' trigger and its configuration.

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

Uploaded a file to the s3 bucket

The screenshot shows the AWS S3 'Upload: status' page. A green success message indicates 'Upload succeeded'. Below it, the 'Summary' table shows 1 succeeded file (dag_example.pdf) and 0 failed files. The 'Files and folders' tab is selected, showing a single file named 'dag_example.pdf' with a size of 11.8 KB and a status of 'Succeeded'.

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

The screenshot shows the AWS CloudWatch Log Events interface. The left sidebar navigation includes 'CloudWatch' (selected), 'Favorites and recents', 'Dashboards New', 'AI Operations Preview', 'Alarms', 'Logs' (selected), 'Log groups New', 'Log Anomalies', 'Live Tail', 'Logs Insights New', 'Contributor Insights', 'Metrics', 'X-Ray traces New', 'Events', and 'Application Signals New'. The main content area is titled 'Log events' with a search bar and filter options. It displays log entries for an Lambda function execution. One entry shows the function starting and performing an S3 PUT operation to trigger indexing.

```

2025-04-12T21:49:24.955Z INIT_START Runtime Version: python:3.13.v31 Runtime Version ARN: arn:aws:lambda:us-east-1::runtime:f713oc0efbf982fcf9ba88eaad0c31352efoe870b225c19f1683fe791596f1
2025-04-12T21:49:25.063Z START RequestId: 34636d2-e087-4715-9155-7342f190c307 Version: $LATEST
2025-04-12T21:49:25.064Z Received event: {'Records': [{'eventVersion': '2.1', 'eventSource': 'aws:s3', 'awsRegion': 'us-east-1', 'eventTime': '2025-04-12T21:49:23.892Z', 'eventName': 'ObjectCreated:Put', 'userIdentity': {'principalId': 'AWS-AIDASIVGQ5UVELSYLTHEE'}, 'requestParameters': {'sourceIPAddress': '172.56.167.74'}, 'responseElements': {'x-amz-request-id': 'PK2J3GANS2DQN00B', 'x-amz-id-2': 'W02Z23D5BBjSgnMy5sNcnUs/RoCv/wAFcPnPgXZ1fM0rKfX029Net0jq5f1Qo1/4542dkp2GehVlernmUHqPufF'}, 's3': {'s3SchemaVersion': '1.0', 'configurationId': 'TriggerIndexing', 'bucket': {'name': 'photo-bucket-assignment3'}, 'object': {'key': 'dog.example.pdf', 'size': 12133, 'eTag': 'd9963bc6288973d00b487364dc91bd2', 'sequencer': '0067FA0FE30795682A'}}]}
2025-04-12T21:49:25.065Z REPORT RequestId: 34636d2-e087-4715-9155-7342f190c307 Duration: 2.07 ms Billed Duration: 3 ms Memory Size: 128 MB Max Memory Used: 31 MB Init Duration: 104.49 ms
2025-04-12T21:49:25.065Z END RequestId: 34636d2-e087-4715-9155-7342f190c307

```

- If the Lambda (**LF1**) did not get invoked, check to see if you set up the correct permissions for 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 Rekognition ("detectLabels" method).
- Use the S3 SDK's headObject method to 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

The screenshot shows the AWS Lambda Function Editor. The left sidebar shows 'Lambda' > 'Functions' > 'index-photos'. The main area has tabs for 'Code source' (selected) and 'Info'. The code editor contains Python code for the 'lambda_function.py' file:

```

EXPLORER
INDEX-PHOTOS
lambda_function.py

lambda_function.py
1 import json
2 import boto3
3 import datetime
4 import requests
5 import os
6
7 rekognition = boto3.client('rekognition')
8 s3 = boto3.client('s3')
9
10 OPENSEARCH_ENDPOINT = OPENSEARCH_ENDPOINT = "https://search-photos-2zcvc2zir5roxzmvt7ezajhyi.us-east-1.es.amazonaws.com"
11
12 def lambda_handler(event, context):
13     print("Event:", json.dumps(event))
14
15     for record in event['Records']:
16         bucket = record['s3']['bucket']['name']
17         key = record['s3']['object']['key']
18
19         # Detecting labels from Rekognition
20         rekog_response = rekognition.detect_labels(
21             Image={'S3Object': {'Bucket': bucket, 'Name': key}},
22             MaxLabels=10
23         )
24         detected_labels = [label['Name'] for label in rekog_response['Labels']]
25         print("Detected labels:", detected_labels)
26
27         # Getting custom metadata (if any)
28         metadata = s3.head_object(Bucket=bucket, Key=key)['Metadata']
29         custom_labels = metadata.get('customlabels', '')
30         custom_labels_array = [lbl.strip() for lbl in custom_labels.split(',') if custom_labels else []]
31
32         # Creating a new document in Elasticsearch
33         es_data = {
34             "id": key,
35             "file_name": key,
36             "file_size": os.path.getsize(key),
37             "file_type": "image",
38             "labels": detected_labels + custom_labels_array
39         }
40
41         # Indexing the document
42         es_index = client.index(index=OPENSEARCH_ENDPOINT, id=key, body=es_data)
43
44     return {"statusCode": 200, "body": "Success"}

```

Uploaded an image to s3 bucket



Cloudwatch logs

CloudWatch < Log groups > /aws/lambda/index-photos > 2025/04/13/[SLATEST]b1f4636dc66c453db1868d15c83d5154

Log events

You can use the filter bar below to search for and match terms, phrases, or values in your log events. [Learn more about filter patterns](#)

Filter events - press enter to search

Clear 1m 30m 1h 12h Custom UTC timezone Display

Timestamp	Message
2025-04-13T00:43:57.311Z	Detected labels: ['Coat', 'Jacket', 'Adult', 'Male', 'Man', 'Person', 'Face', 'Head', 'Kissing', 'Lifejacket'] Detected labels: ['Coat', 'Jacket', 'Adult', 'Male', 'Man', 'Person', 'Face', 'Head', 'Kissing', 'Lifejacket']
2025-04-13T00:43:57.679Z	Final JSON document: {"objectKey": "62162d79aa6d24ee1061d4af14d939d4.jpg", "bucket": "photo-bucket-assignment3", ... Final JSON document: { "objectKey": "62162d79aa6d24ee1061d4af14d939d4.jpg", "bucket": "photo-bucket-assignment3", "creationTimestamp": "2025-04-13T00:43:57.679822", "labels": ["Lifejacket", "Person", "Face", "Kissing", "Male", "Jacket", "Adult", "Man", "Male", "Coat"] } OpenSearch response: 201 {"_index": "photos", "_id": "5mmaLJYBnW7q0CiwYwP6", "_version": 1, "result": "created", "_shards... Back to top
2025-04-13T00:43:58.260Z	OpenSearch response: 201 {"_index": "photos", "_id": "5mmaLJYBnW7q0CiwYwP6", "_version": 1, "result": "created", "_shards...

PUT to opensearch

2025-04-13T00:43:58.260Z OpenSearch response: 201 {"_index": "photos", "_id": "5mmaLJYBnW7q0CiwYwP6", "_version": 1, "result": "created", "_shards...

OpenSearch response: 201 {"_index": "photos", "_id": "5mmaLJYBnW7q0CiwYwP6", "_version": 1, "result": "created", "_shards": { "total": 2, "successful": 1, "failed": 0 }, "_seq_no": 0, "_primary_term": 1 }

Tested with curl:

```
(base) yashavikasingh@Yashoos-MacBook-Air ~ % curl -X GET \
-u 'yashavika:@alipore1908' \
[ 'https://search-photos-2zcw2zir5roxzmvt7ezajhyi.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" : "6mmalJYBnW7q0CiwYwP6",
        "_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

- Create a Lambda function (**LF2**) called “**search-photos**”.
- Create an Amazon Lex bot to handle search queries.
 - Create one intent named “**SearchIntent**”.

The screenshot shows the AWS Lambda function configuration interface. The left sidebar lists the function name as "SearchIntent". The main area displays the function code in a text editor, which includes imports for AWS services like CloudWatch Metrics and CloudWatch Logs, and a handler function named "lambdaHandler". The function uses the "Node.js 18.x" runtime. The configuration tab is selected, showing environment variables and triggers. The "Code" tab shows the actual Lambda code.

- 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”).

The screenshot shows the AWS Amazon Lex console. The left sidebar lists the bot name as "PhotoSearchBot". The main area shows the "Intents" section with two intents: "SearchIntent" and "FallbackIntent". The "SearchIntent" card is open, showing its configuration details. A test window on the right shows a conversation where the user types "trees" and the bot responds with "Okay, Let me search for that...". Another message "Show me for trees, dogs" is shown, followed by a response "Okay, Let me search for that...". The status bar at the bottom indicates "Ready for complete testing".

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

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

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

```

1  [
2   "messages": [
3     {
4       "content": "Okay, Let me search for that...",
5       "contentType": "PlainText"
6     }
7   ],
8   "sessionState": {
9     "dialogAction": {
10      "type": "Close"
11    },
12    "intent": {
13      "name": "SearchIntent",
14      "slots": {
15        "SearchKeyWord1": {
16          "value": {
17            "originalValue": "trees",
18            "interpretedValue": "trees",
19            "resolvedValues": [
20              "trees"
21            ]
22          }
23        },
24        "SearchKeyWord2": {
25          "value": {
26            "originalValue": "dogs",
27            "interpretedValue": "dogs",
28            "resolvedValues": [
29              "dogs"
30            ]
31          }
32        }
33      },
34      "state": "ReadyForFulfillment",
35      "confirmationState": "None"
36    },
37    "sessionAttributes": {},
38    "originatingRequestId": "8885276e-aa57-403f-a9fb-07eed0b34985"
39  },
40  "interpretations": [
41    {
42      "nluConfidence": {
43        "score": 0.94
44      },
45      "intent": {
46        "name": "SearchIntent",

```

- If the Lex disambiguation request yields any keywords (K1Kn) 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.

Request 73274018-b506-4e90-ad15-ce0965f3476b

```
{  
  "botAliasId": "TSTALIASID",  
  "botId": "W9X7TFEIB7",  
  "localeId": "en_US",  
  "text": "Show me for trees, dog",  
  "sessionId": "156041407786639"  
}
```

 Copy

Response

Summary

JSON input and output

Response

```
.  
  "resolvedValues": [  
    "trees"  
  ]  
},  
  "SearchKeyWord2": {  
    "value": {  
      "originalValue": "dogs",  
      "interpretedValue": "dogs"  
    }  
    "resolvedValues": [  
      "dogs"  
    ]  
  }  
},  
},
```

 Copy

Inspect

Show me for trees, dogs

Okay, Let me search for that...

show me

Intent FallbackIntent is fulfilled

 Ready for complete testing

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

```
lambda_function.py X
lambda_function.py
21 def lambda_handler(event, context):
22     try:
23
24         print("Lex Response:", json.dumps(lex_response))
25
26         slots = lex_response.get("interpretations", [])[0].get("intent", {}).get("slots", {})
27         keywords = []
28         for slot in slots.values():
29             if slot and "value" in slot:
30                 interpreted = slot["value"].get("interpretedValue", "")
31                 if interpreted:
32                     keywords.extend([kw.strip().lower() for kw in interpreted.split(",") if kw.strip()])
33
34         print("Extracted Keywords:", keywords)
35
36         if not keywords:
37             return {'statusCode': 200, 'body': json.dumps([])}
38
39         # Build OpenSearch query
40         must_clauses = [{"match": {"labels": keyword}} for keyword in keywords]
41         search_query = {
42             "query": {
43                 "bool": {
44                     "must": must_clauses
45                 }
46             }
47         }
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
69
```

The screenshot shows the AWS Lambda Alias configuration page for an alias named 'ProdBotAlias'. The top navigation bar includes links for Lex, Bots, Bot: PhotoSearchBot, Aliases, and Alias: ProdBotAlias. Below the navigation, there are two buttons: 'Delete' and 'Associate version with alias' (which is highlighted in orange). The main section is titled 'Details' and contains the following information:

Alias name	ProdBotAlias	Description
Associated version	Version 1	Sentiment analysis- Disabled
ID:	3UMIP5FI7X	

Below the details, there is a section titled 'Languages (1)' with an 'Info' link. It includes a 'Test' button and a 'Manage languages in alias' button. A note says 'Select the languages you want to enable in the alias. Only languages that are built can be enabled.' A search bar labeled 'Search languages' is present. The language table shows one entry:

Language	Status	Enabled in alias
English (US)	Successfully built	<input checked="" type="checkbox"/>

Finally, there is a 'Conversation logs' section with an 'Info' link and a 'Manage conversation logs' button. A note says 'Enable logging to record details of conversations with your bot. You can enable audio logging, text logging, or both.'

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:

The screenshot shows the Swagger Editor interface at <https://editor.swagger.io>. The left panel displays the generated Swagger JSON code for an AI Photo Search API. The right panel shows the API details, parameters, responses, and execution tools.

API Details: This API takes in a search query as a query parameter and returns zero or more photos that match the search query.

Parameters:

Name	Description
q	the string to be searched string (query)

Responses: Response content type: application/json

Curl: curl -X 'GET' \n 'https://editor.swagger.io/v1/search?q=query1' \n-H 'accept: application/json'

Request URL: <https://editor.swagger.io/v1/search?query1>

Server response: (Code, Details)

```
1: { "swagger": "2.0", "info": { "title": "AI Photo Search", "description": "AI Photo Search application, built during the Cloud and Big Data course at Columbia University.", "version": "1.0.0" }, "schemes": [ "https" ], "basePath": "/v1", "produces": [ "application/json" ], "paths": { "/search": { "get": { "summary": "photo search method", "description": "This API takes in a search query as a query parameter and returns zero or more photos that match the search query.", "operationId": "searchPhotos", "parameters": [ { "in": "query", "name": "q", "type": "string", "description": "the string to be searched" } ], "produces": [ "application/json" ], "responses": { "200": { "description": "search results", "schema": { "type": "array", "items": { "type": "object", "properties": { "id": { "type": "string" }, "url": { "type": "string" } } } } } } } } }
```

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:

Photo-search-1 [Info](#)

Allows API Gateway to push logs to CloudWatch Logs.

Summary

Creation date April 24, 2025, 02:54 (UTC-04:00)	ARN arn:aws:iam::156041407786:role/Photo-search-1
Last activity -	Maximum session duration 1 hour

Permissions [Edit](#)

Permissions policies (2) [Info](#)

You can attach up to 10 managed policies.

Filter by Type		
Search	All types	
<input type="checkbox"/> Policy name AmazonAPIGatewayPushToCloud...	Type	AWS managed
<input type="checkbox"/> APIGatewayS3PutPolicy	Customer managed	1

Permissions boundary (not set)

[Create resource](#)

- /
- /search
 - GET
- /upload
 - PUT

/upload - PUT - Method execution

ARN [arn:aws:execute-api:us-east-1:156041407786:ahbfh907hl/*/PUT/upload](#) Resource ID l0cdyh

Method request → Integration request → Integration response ← Method response ← Integration response ← Method response ← Client → AWS integration

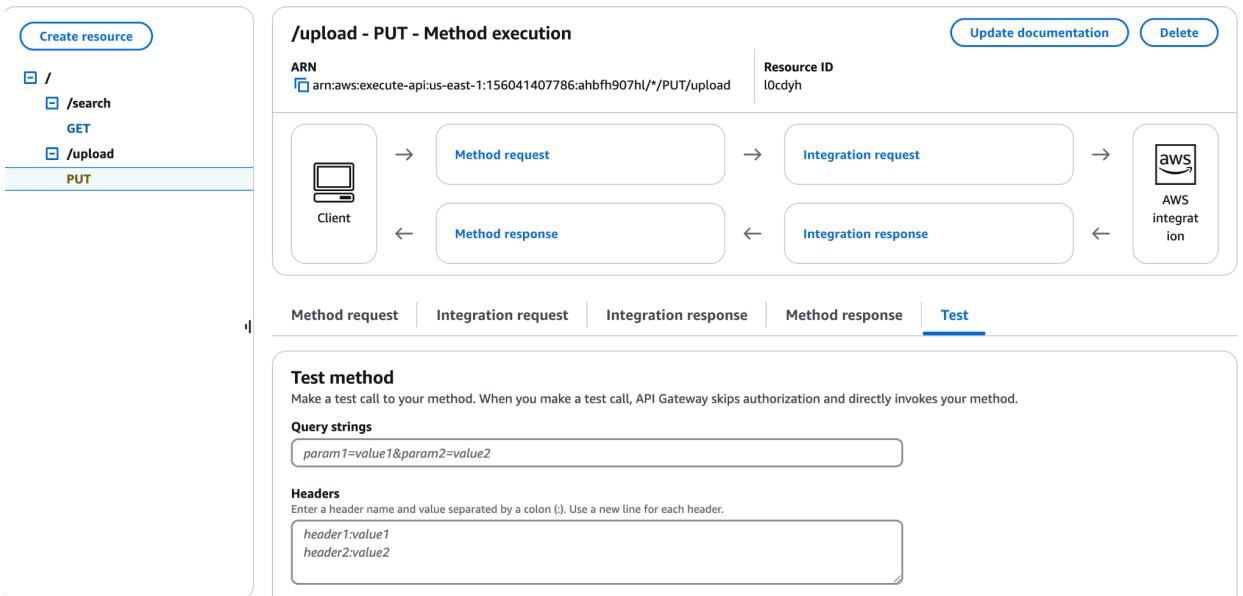
[Update documentation](#) [Delete](#)

Test method
Make a test call to your method. When you make a test call, API Gateway skips authorization and directly invokes your method.

Query strings
`param1=value1¶m2=value2`

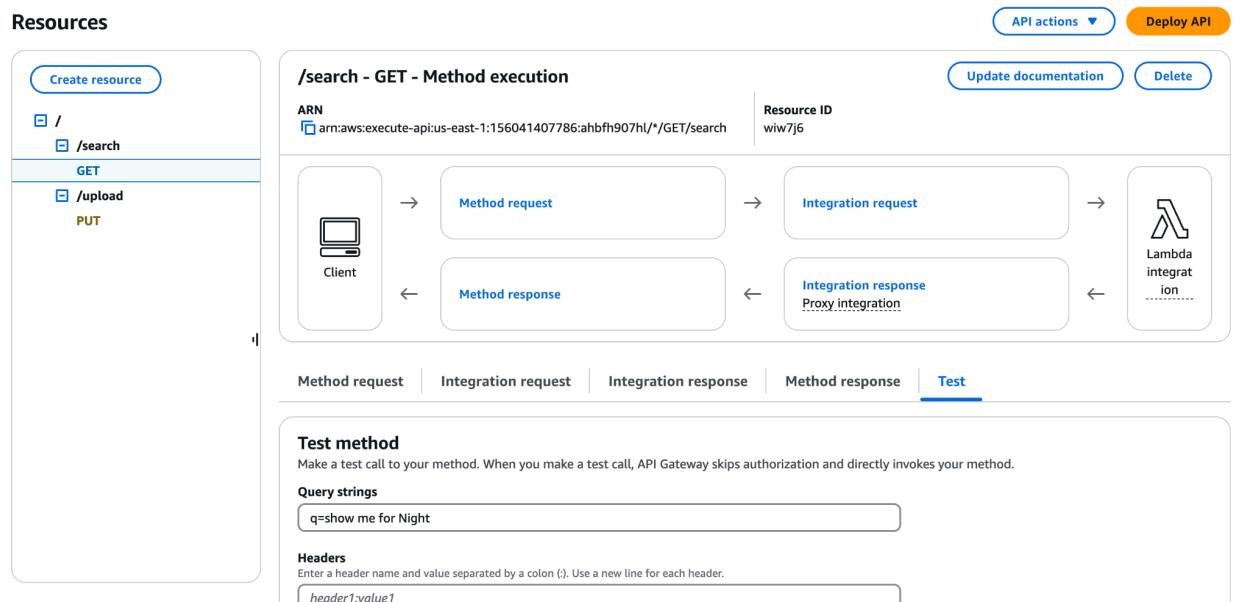
Headers
Enter a header name and value separated by a colon (:). Use a new line for each header.
`header1:value1
header2:value2`

- 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

Create resource

- /
- /search
- GET
- /upload
- PUT

/search - GET - Method execution

ARN: arn:aws:execute-api:us-east-1:156041407786:ahbfh907hl/*GET/search

Resource ID: wiw7j6

Method request → Integration request → Lambda integration

← Method response ← Integration response ← Proxy integration

Method request settings

Authorization: NONE

Request validator: None

API key required: True

SDK operation name: searchPhotos

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

Create resource

- /
- /search
- GET
- /upload
- PUT

/search - GET - Method execution

ARN: arn:aws:execute-api:us-east-1:156041407786:ahbfh907hl/*GET/search

Resource ID: wiw7j6

Method request → Integration request → Lambda integration

← Method response ← Integration response ← Proxy integration

Method request settings

Authorization: NONE

Request validator: None

API key required: True

SDK operation name: searchPhotos

The screenshot shows the AWS API Gateway interface. On the left, a sidebar lists 'APIs', 'Custom domain names', 'Domain name access associations', 'VPC links', 'Usage plans', 'API keys' (which is selected), 'Client certificates', and 'Settings'. The main content area is titled 'photo-search-api-key'. It displays 'API key details' including ID (lbzipeh248), Status (Active), Description (-), Creation date (April 24, 2025, 10:49 (UTC-04:00)), and an API key value. Below this, tabs for 'Associated usage plans' and 'Tags' are shown. Under 'Associated usage plans', there is one entry: 'Usage plans (1)' with a table showing a single row for 'phot-search-usage-plan' associated with 'AI Photo Search' under 'Associated APIs' and 'DEV' under 'Associated stages'. A 'Last updated' timestamp is also present.

d. Deploy the API.

The screenshot shows the AWS API Gateway interface. The left sidebar includes 'APIs', 'Custom domain names', 'Domain name access associations', 'VPC links', 'API: AI Photo Search' (selected), 'Resources', 'Stages' (selected), 'Authorizers', 'Gateway responses', 'Models', 'Resource policy', 'Documentation', 'Dashboard', and 'API settings'. The main content area is titled 'Stages' and shows a tree view of deployed stages: 'DEV' (selected) containing '/search' (with 'GET' and 'PUT' methods) and '/upload'. To the right, the 'Stage details' section shows 'Stage name: DEV', 'Rate info: 10000', 'Cache cluster info: Inactive', 'Default method-level caching: Inactive', 'Invoke URL: https://ahbfh907hl.execute-api.us-east-1.amazonaws.com/DEV', and an 'Active deployment' log entry. Below this, the 'Logs and tracing' section includes 'CloudWatch logs: Inactive', 'X-Ray tracing: Inactive', and 'Detailed metrics: Inactive'.

```
Last login: Fri Apr 18 11:50:40 on ttys008
sd6201Mac ~ % curl -X GET "https://ahbfh907hl.execute-api.us-east-1.amazonaws.com/DEV?q=Night" \
-H "x-api-key: ZA050y0qWC8c04d0bybw6qSIAp5DE98qeJwGN110"
{"message": "Missing Authentication Token"}%
sd6201Mac ~ % curl -X GET "https://ahbfh907hl.execute-api.us-east-1.amazonaws.com/DEV/search?q=Night" \
-H "x-api-key: ZA050y0qWC8c04d0bybw6qSIAp5DE98qeJwGN110"
{"message": "Forbidden"}%
sd6201Mac ~ % curl -X GET "https://ahbfh907hl.execute-api.us-east-1.amazonaws.com/DEV/search?q=Night" \
-H "x-api-key: ZA050y0qWC8c04d0bybw6qSIAp5DE98qeJwGN110"
{"message": "Forbidden"}%
sd6201Mac ~ % curl -X GET "https://ahbfh907hl.execute-api.us-east-1.amazonaws.com/DEV/search?q=Night" \
-H "x-api-key: ZA050y0qWC8c04d0bybw6qSIAp5DE98qeJwGN110"
[{"objectKey": "image1.png", "bucket": "photo-bucket-assignment3", "createdTimestamp": "2025-04-23T04:17:54.818576", "labels": ["Sky", "Light", "Flare", "Night", "Outdoors", "Texture", "Nature", "Rainbow", "Lighting"]}, {"objectKey": "image1.png", "bucket": "photo-bucket-assignment3", "createdTimestamp": "2025-04-23T04:19:08.210422", "labels": ["Light", "Sky", "Texture", "Rainbow", "Lighting", "Flare", "Nature", "Night", "Outdoors"]}]

sd6201Mac ~ %
```

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

frontend > js > sdk > **JS** apigClient.js
160

PROBLEMS OUTPUT DEBUG C
sd6201@Mac SnapSearch-photo-a

```
SNAPSEARCH-PHOTO-ALBUM-WEBAPP
└── frontend
    ├── css
    │   └── styles.css
    ├── js
    │   └── sdk
    │       ├── lib
    │       │   └── apiGatewayCore
    │       │       ├── apiGatewayClient.js
    │       │       ├── sigV4Client.js
    │       │       ├── simpleHttpClient.js
    │       │       └── utils.js
    │       ├── axios
    │       ├── CryptoJS
    │       ├── url-template
    │       └── apigClient.js
    ├── CHANGELOG.md
    ├── README.md
    ├── app.js
    └── index.html
```

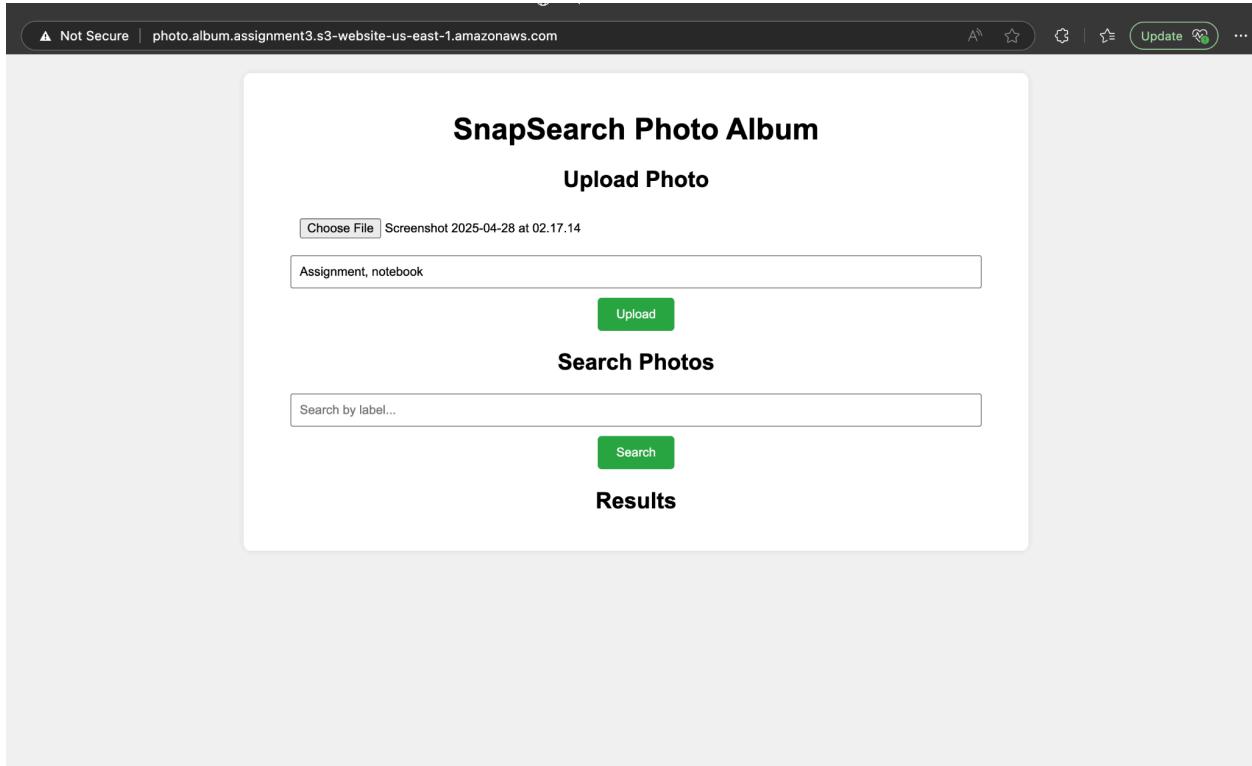
.gitignore

swagger-api-specs.json

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:

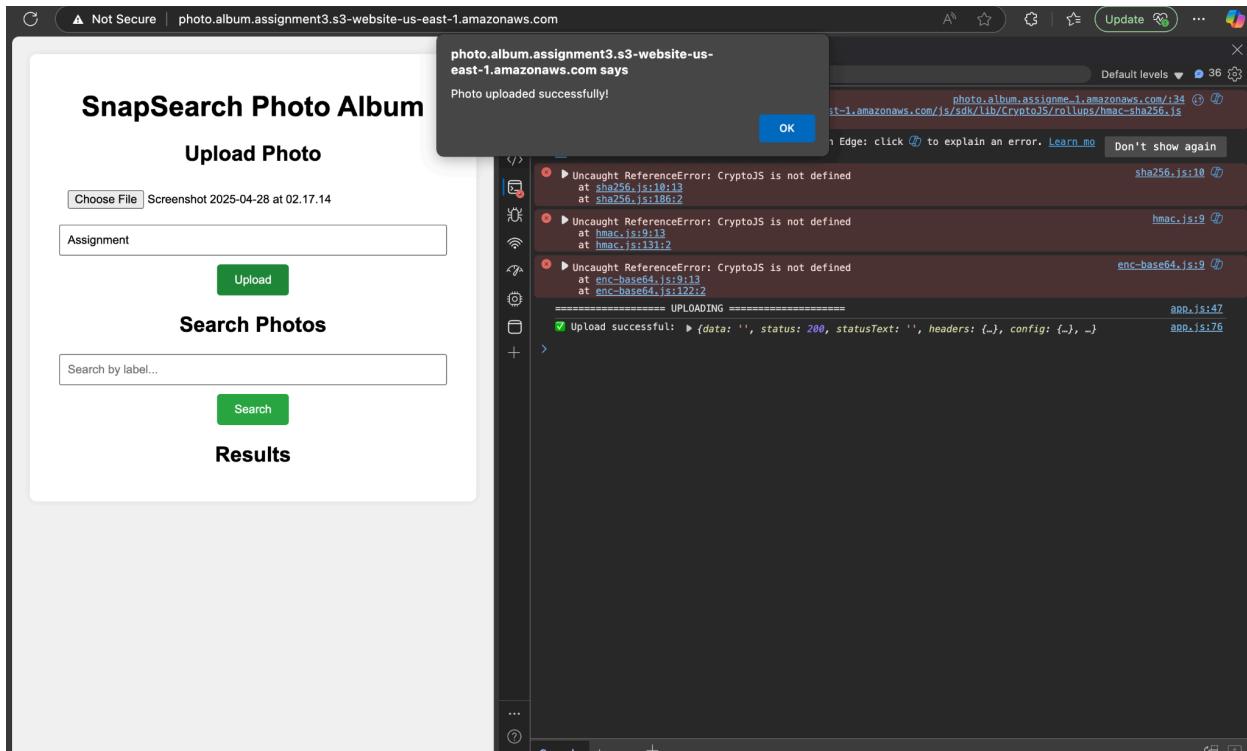


photo-bucket-assignment3 [Info](#)

Objects							
		Metadata	Properties	Permissions	Metrics	Management	Access Points
Objects (6)							
<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class		
<input type="checkbox"/>	Screenshot%202025-04-29%20at%2002.17.14.png	png	April 28, 2025, 10:00:21 (UTC-04:00)	0 B	Standard		
<input type="checkbox"/>	test.png	png	April 27, 2025, 14:31:46 (UTC-04:00)	201.1 KB	Standard		
<input type="checkbox"/>	test1.png	png	April 27, 2025, 22:14:55 (UTC-04:00)	207.3 KB	Standard		
<input type="checkbox"/>	test12.png	png	April 28, 2025, 04:21:30 (UTC-04:00)	0 B	Standard		
<input type="checkbox"/>	test56.png	png	April 28, 2025, 04:08:52 (UTC-04:00)	2.0 B	Standard		
<input type="checkbox"/>	test566.png	png	April 28, 2025, 03:57:33 (UTC-04:00)	376.1 KB	Standard		

Search:

Upload Photo

No file chosen

Enter custom labels (comma separated)

Search Photos

Results

The screenshot shows a mobile application interface. At the top, there is a header bar with a back arrow icon and the text "Work To Do". Below this, there is a section titled "April 21 - May 4" which contains two items: "Midterm Review" and "Assignment 3". Underneath this is another section titled "Announcements" with a dropdown arrow. Below this, there is a post titled "Final Project Progress review" by "Ajay Krishnan Gopalan" posted on "Apr 21, 2025 9:12 AM". The post content includes a message to everyone, instructions for creating a progress document, and a note about grading. It also mentions TAs.

- 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**).

The screenshot shows the AWS S3 console with the 'General purpose buckets' tab selected. At the top, there's an 'Account snapshot - updated every 24 hours' section with a link to 'View Storage Lens dashboard'. Below it, a note says 'Storage lens provides visibility into storage usage and activity trends. Metrics don't include directory buckets.' with a 'Learn more' link. A search bar labeled 'Find buckets by name' is present. To the right are buttons for 'Copy ARN', 'Empty', 'Delete', and 'Create bucket'. The main area displays a table of five existing buckets:

Name	AWS Region	IAM Access Analyzer	Creation date
assignment.dining.concierge.com	US East (N. Virginia) us-east-1	View analyzer for us-east-1	February 11, 2025, 01:58:43 (UTC-05:00)
photo-album-bucket.assignment3.test	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 16, 2025, 22:08:14 (UTC-04:00)
photo-bucket-assignment3	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 12, 2025, 17:21:14 (UTC-04:00)
photo.album.assignment3	US East (N. Virginia) us-east-1	View analyzer for us-east-1	April 16, 2025, 20:37:32 (UTC-04:00)
research-paper-summarisation	US East (N. Virginia) us-east-1	View analyzer for us-east-1	March 27, 2025, 17:47:57 (UTC-04:00)

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

The screenshot shows the AWS S3 console interface. At the top, there's a navigation bar with the AWS logo, a search bar, and account information: United States (N. Virginia) and yashavika-nyu @ 1560-4140-7786. Below the navigation bar, a green banner displays a success message: "Upload succeeded" with a link to the "Files and folders table". The main area is titled "photo.album.assignments3" with a "Info" link. A navigation menu below the title includes "Objects", "Metadata", "Properties", "Permissions", "Metrics", "Management", and "Access Points". The "Objects" tab is selected. A sub-header "Objects (1)" is shown above a table. The table has columns for Name, Type, Last modified, Size, and Storage class. One item, "index.html", is listed with a size of 14.2 KB and a storage class of Standard. Action buttons like Copy S3 URI, Copy URL, Download, Open, Delete, Actions, Create folder, and Upload are available at the top of the object list.

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

This screenshot shows the AWS S3 console after an upload. The top navigation bar and account info are identical to the previous screenshot. A green banner at the top indicates "Upload succeeded" with a link to the "Files and folders table". Below it, a section titled "Upload: status" is displayed. It includes a note: "After you navigate away from this page, the following information is no longer available." A "Close" button is in the top right of this section. The "Summary" section shows the destination as "s3://photo.album.assignment3". It lists "Succeeded" with "1 file, 14.2 KB (100.00%)" and "Failed" with "0 files, 0 B (0%)". Below this, tabs for "Files and folders" and "Configuration" are visible. The "Files and folders" tab is selected, showing a table with one item: "index.html" (text/html, 14.2 KB, Status: Succeeded). The table has columns for Name, Folder, Type, Size, Status, and Error.

<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.

The screenshot shows a file explorer interface with the following project structure:

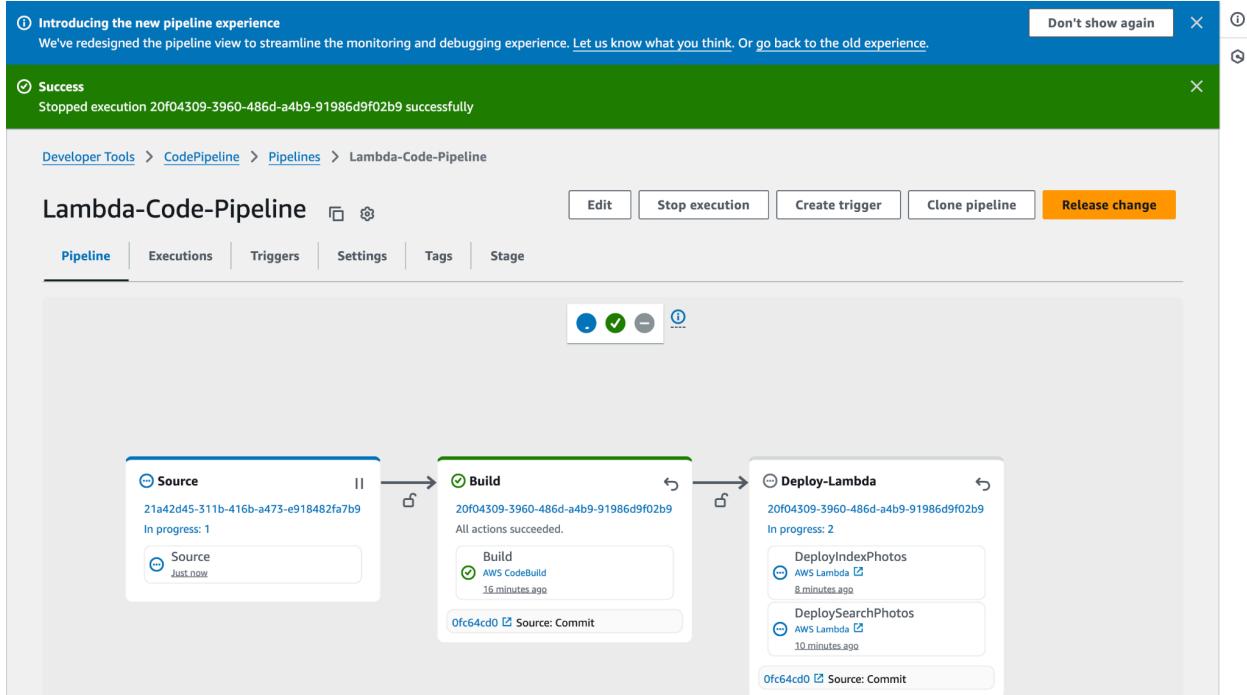
- SNAPSEARCH-PHOTO-ALBUM-WEBAPP
 - frontend
 - css (styles.css)
 - js
 - sdk
 - lib
 - apiGatewayCore
 - JS apiGatewayClient.js
 - JS sigV4Client.js
 - JS simpleHttpClient.js
 - JS utils.js
 - > axios
 - > CryptoJS
 - > url-template
 - JS apigClient.js
 - CHANGELOG.md
 - README.md
 - JS app.js
 - index.html
 - lambdas
 - .gitignore
 - { } swagger-api-specs.json

On the right side of the interface, there is a status bar with the following information:

 - frontend > js > sdk > JS apigClient
 - 160
 - M
 - U
 - M
 - PROBLEMS
 - OUTPUT
 - DEBUG C
 - sd6201@Mac SnapSearch-photo-a

6. Deploy your code using AWS CodePipeline

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



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 21:36:12.864406 Phases found in YAML: 1
15 [Container] 2025/04/28 21:36:12.864406 Phases found in YAML: 1
16 [Container] 2025/04/28 21:36:12.864416 Phase complete: DOWNLOAD_SOURCE State: SUCCEEDED
17 [Container] 2025/04/28 21:36:12.884329 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:

```

EXPLORER
SNAPSEARCH-PHOTO-ALBUM-WEBAPP
  frontend
    css
    js
    sdk
      lib
        apigClient.js
        CHANGELOG.md
        README.md
      app.js
      index.html
  lambdas
    index-photos.py 2
    search-photos.py 3
    .gitignore
    buildspec.yml
    swagger-api-specs.json

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

```

index-photos.py 2 x search-photos.py 3 styles.css app.js index.html ~/Downloads {} swagger-api-specs.json J D □
lambdas > index-photos.py @ lambda_handler
1 import json
2 import boto3
3 import datetime
4 import requests
5 import urllib.parse
6
7 # Initialize AWS clients
8 s3 = boto3.client('s3')
9 rekognition = boto3.client('rekognition')
10
11 print("DEPLOYING LAMBDA VIA CODEBUILD-2")
12 # OpenSearch Configuration
13 OPENSEARCH_ENDPOINT = "https://search-photos-2zcv2zir5roxmzvvt7ezajhy1.us-east-1.es.amazonaws.com/photos/_doc"
14 OPENSEARCH_USERNAME = "yashavika"
15 OPENSEARCH_PASSWORD = "@Alipore1908"
16
17 def lambda_handler(event, context):
18     print("Received Event:", json.dumps(event))
19
20     for record in event.get('Records', []):
21         bucket = record['s3']['bucket']['name']
22         key = urllib.parse.unquote_plus(record['s3']['object']['key'])
23
24         print(f"\nProcessing file: s3://{bucket}/{key}")
25
26         try:
27             # Step 1: Get S3 Object Metadata
28             s3_metadata = s3.head_object(Bucket=bucket, Key=key)

```

Lambda > Functions > index-photos3

Code source [Info](#)

[Upload from](#) [Download](#)

EXPLORER

- INDEX-PHOTOS3
 - index-photos.py

DEPLOY

- [Deploy \(Δ\)](#)
- [Test \(Δ\)](#)

TEST EVENTS [NONE SELECTED]

- + Create new test event

index-photos.py

```

1 import json
2 import boto3
3 import datetime
4 import requests
5 import urllib.parse
6
7 # Initialize AWS clients
8 s3 = boto3.client('s3')
9 rekognition = boto3.client('rekognition')
10
11 print("DEPLOYING LAMBDA VIA CODEBUILD-2")
12 # OpenSearch Configuration
13 OPENSEARCH_ENDPOINT = "https://search-photos-2zcv2zir5roxzmzmv7ezajhyi.us-east-1.es.amazonaws.com/photos/_doc"
14 OPENSEARCH_USERNAME = "yashavika"
15 OPENSEARCH_PASSWORD = "@Alipore1908"
16
17 def lambda_handler(event, context):
18     print("Received Event:", json.dumps(event))
19
20     for record in event.get("Records", []):
21         bucket = record['s3']['bucket']['name']
22         key = urllib.parse.unquote_plus(record['s3']['object']['key'])
23
24         print(f"\nProcessing file: s3://{bucket}/{key}")
25
26         try:
27             # Step 1: Get S3 Object Metadata
28             s3_metadata = s3.head_object(Bucket=bucket, Key=key)
29             content_type = s3_metadata.get('ContentType', '')
30
31             print(f"Content-Type: {content_type}")

```

Changes in the search-photos3:

SnapSearch-photo-album-webapp

EXPLORER

- SNAPSEARCH-PHOTO-ALBUM-WEBAPP
 - frontend
 - css
 - js
 - sdk
 - lib
 - apiclient.js
 - CHANGELOG.md
 - README.md
 - app.js
 - index.html
 - lambda
 - index-photos.py
 - search-photos.py
 - .gitignore
 - buildspec.yml
 - swagger-api-specs.json

index-photos.py

```

1 import json
2 import boto3
3 import base64
4 import requests
5 from requests.auth import HTTPBasicAuth
6
7 # OpenSearch setup
8 username = 'yashavika'
9 password = '@Alipore1908'
10 basicauth = HTTPBasicAuth(username, password)
11
12 print("DEPLOYING LAMBDA VIA CODEBUILD-2")
13 opensearch_host = 'search-photos-2zcv2zir5roxzmzmv7ezajhyi.us-east-1.es.amazonaws.com'
14 index = 'photos'
15
16 # Lex setup
17 bot_id = 'W9X7TTEIB7'
18 bot_alias_id = '3UMIP5FI7X'
19 locale_id = 'en_US'
20
21 lex_client = boto3.client('lexv2-runtime')
22 s3_client = boto3.client('s3')
23
24 cors_headers = {
25     'Access-Control-Allow-Origin': '*',
26     'Access-Control-Allow-Methods': 'PUT, GET, POST, OPTIONS',
27     'Access-Control-Allow-Headers': 'Content-Type, Authorization, sessionId, x-api-key, x-amz-meta-customLabels, X-Amz-Date, X-'
28 }

```

search-photos.py

```

1 import json
2 import boto3
3 import base64
4 import requests
5 from requests.auth import HTTPBasicAuth
6
7 # OpenSearch setup
8 username = 'yashavika'
9 password = '@Alipore1908'
10 basicauth = HTTPBasicAuth(username, password)
11
12 print("DEPLOYING LAMBDA VIA CODEBUILD-2")
13 opensearch_host = 'search-photos-2zcv2zir5roxzmzmv7ezajhyi.us-east-1.es.amazonaws.com'
14 index = 'photos'
15
16 # Lex setup
17 bot_id = 'W9X7TTEIB7'
18 bot_alias_id = '3UMIP5FI7X'
19 locale_id = 'en_US'
20
21 lex_client = boto3.client('lexv2-runtime')
22 s3_client = boto3.client('s3')
23
24 cors_headers = {
25     'Access-Control-Allow-Origin': '*',
26     'Access-Control-Allow-Methods': 'PUT, GET, POST, OPTIONS',
27     'Access-Control-Allow-Headers': 'Content-Type, Authorization, sessionId, x-api-key, x-amz-meta-customLabels, X-Amz-Date, X-'
28 }

```

PROBLEMS **OUTPUT** **DEBUG CONSOLE** **TERMINAL** **PORTS**

Total 3 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2) completed with 2 local objects.

Code source Info

Upload from ▼ ▼

EXPLORER ...

SEARCH-PHOTOS ...

search-photos.py

DEPLOY

Deploy (F5)

Test (F6)

```
4 import requests
5 from requests.auth import HTTPBasicAuth
6
7 # OpenSearch setup
8 username = 'yashavika'
9 password = '@Alipore1908'
10 basicauth = HTTPBasicAuth(username, password)
11
12 print("DEPLOYING LAMBDA VIA CODEBUILD-2")
13 opensearch_host = 'search-photos-2zcv2zir5roxzmvt7ezajhyi.us-east-1.es.amazonaws.com'
14 index = 'photos'
15
16 # Lex setup
17 bot_id = 'W9X7TFEIB7'
18 bot_alias_id = '3UMIP5FI7X'
19 locale_id = 'en_US'
20
21 lex_client = boto3.client('lexv2-runtime')
22 s3_client = boto3.client('s3')
23
24 cors_headers = [
    'Access-Control-Allow-Origin': '*'
```

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

The screenshot shows the AWS CodePipeline console with a pipeline named "frontend". The pipeline has two stages: "Source" and "Deploy". The "Source" stage is triggered by GitHub via OAuth app and has one action named "first". The "Deploy" stage is to Amazon S3 and also has one action named "first". Both stages show green checkmarks indicating success. The pipeline status is "All actions succeeded".

7. Create a AWS CloudFormation template for the stack

```

template.yaml -- cloud_yaml

template.yaml
! template.yaml > {} Outputs > {} ApiUrl > Value
 1 AWSTemplateFormatVersion: '2010-09-09'
 2 Resources:
 3
 4 # S3 Bucket for photos
 5 PhotoStorageBucket:
 6   Type: AWS::S3::Bucket
 7   Properties:
 8     BucketName: photo-album-storage-bucket-yashavika4
 9     AccessControl: Private
10     PublicAccessBlockConfiguration:
11       BlockPublicAcls: false
12       BlockPublicPolicy: false
13       IgnorePublicAcls: false
14       RestrictPublicBuckets: false
15
16 PhotoStorageBucketPolicy:
17   Type: AWS::S3::BucketPolicy
18   Properties:
19     Bucket: !Ref PhotoStorageBucket
20     PolicyDocument:
21       Statement:
22         - Effect: Allow
23           Principal: "*"
24           Action: s3:PutObject
25           Resource: !Sub "${PhotoStorageBucket.Arn}/*"
26
27 # S3 Bucket for frontend
28 FrontendBucket:
29   Type: AWS::S3::Bucket
30   Properties:
31     BucketName: frontend-photo-album-bucket-yashavika4
32     WebsiteConfiguration:
33       IndexDocument: index.html
34       ErrorDocument: error.html
35     PublicAccessBlockConfiguration:
36       BlockPublicAcls: false
37       BlockPublicPolicy: false
38       IgnorePublicAcls: false
39       RestrictPublicBuckets: false
40
41 FrontendBucketPolicy:
42   Type: AWS::S3::BucketPolicy
43   Properties:
44     Bucket: !Ref FrontendBucket
45     PolicyDocument:
46       Statement:
47         - Effect: Allow
48           Principal: "*"
49           Action: s3:GetObject
50           Resource: !Sub "${FrontendBucket.Arn}/*"
51
52 # IAM Role for Lambda functions
53 LambdaExecutionRole:
54   Type: AWS::IAM::Role
55   Properties:

```

The screenshot shows the AWS CloudFormation template file "template.yaml" in the Cloud9 IDE. The template defines two S3 buckets: "PhotoStorageBucket" and "FrontendBucket", each with specific properties like BucketName, AccessControl, and BucketPolicy. It also defines an IAM role for Lambda functions.

Spinned up resources:

The screenshot shows the AWS CloudFormation console interface. On the left, there's a sidebar with navigation links like 'CloudFormation', 'Stacks', 'Drifts', 'StackSets', 'Exports', 'Infrastructure Composer', 'Hooks overview', 'Registry', 'Feedback', and 'Spotlight'. The main area displays the 'Stacks (12)' section, with 'cloud12' selected. The 'Resources' tab is active, showing a table of 10 resources with their logical IDs, physical IDs, types, and statuses. Most resources are in 'CREATE_COMPLETE' status, except for 'cloud11' and 'cloud10' which are in 'ROLLBACK_COMPLETE' status.

Logical ID	Physical ID	Type	Status	Module
FrontendBucket	frontend-photo-album-bucket-yashavika4	AWS::S3::Bucket	CREATE_COMPLETE	-
FrontendBucketPolicy	frontend-photo-album-bucket-yashavika4	AWS::S3::BucketPolicy	CREATE_COMPLETE	-
IndexPhotosLambda	index-photos3	AWS::Lambda::Function	CREATE_COMPLETE	-
LambdaExecutionRole	cloud12-LambdaExecutionRole-rDUnDjaswh	AWS::IAM::Role	CREATE_COMPLETE	-
PhotoAlbumApi	kpxqdkbBr0	AWS::ApiGateway::RestApi	CREATE_COMPLETE	-
PhotosResource	62ngtk	AWS::ApiGateway::Resource	CREATE_COMPLETE	-
PhotoStorageBucket	photo-album-storage-bucket-yashavika4	AWS::S3::Bucket	CREATE_COMPLETE	-
PhotoStorageBucketPolicy	photo-album-storage-bucket-yashavika4	AWS::S3::BucketPolicy	CREATE_COMPLETE	-
SearchPhotosLambda	search-photos3	AWS::Lambda::Function	CREATE_COMPLETE	-
SearchResource	l1xowl	AWS::ApiGateway::Resource	CREATE_COMPLETE	-