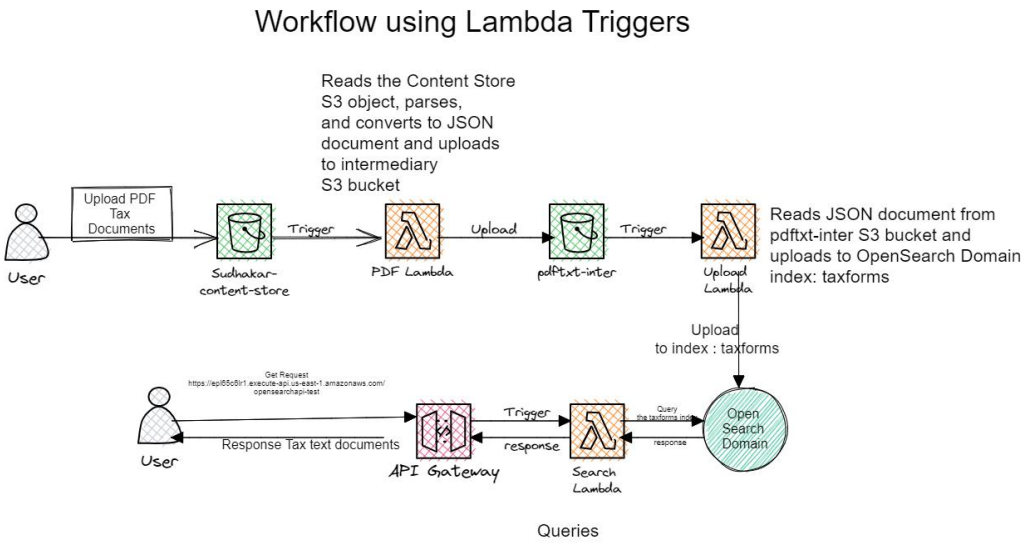
Project 3 – Deployment of AWS Open Search Domain to search Tax Forms

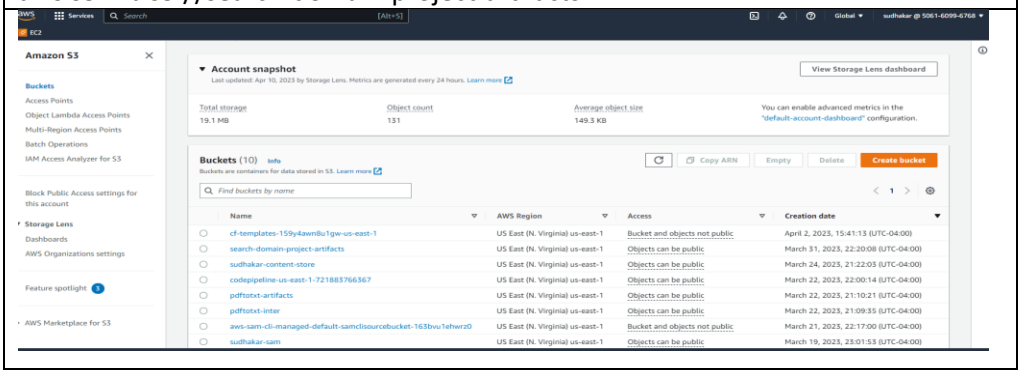
This project demonstrates the use of Lambda functions to parse and load the PDF Tax forms to AWS Open Search Engine and query the tax forms based on the title, description, and keywords. This also demonstrates the use of SAM CloudFormation templates to deploy Lambdas using the AWS CodePipelines. The following workflow shows how Lambdas are triggered by S3 buckets and API Gateway.



The AWS Cloud9 environment has been used to implement this project.

1.0 Create the Content-Store, Intermediary-Stage, Artifact buckets.

Screenshots of S3 buckets. Can be created using the CLI commands.
aws s3 mb s3://sudhakar-content-store
aws s3 mb s3://pdftotext-inter
aws s3 mb s3://sudhakar-content-store
aws s3 mb s3://search-domain-project-artifacts

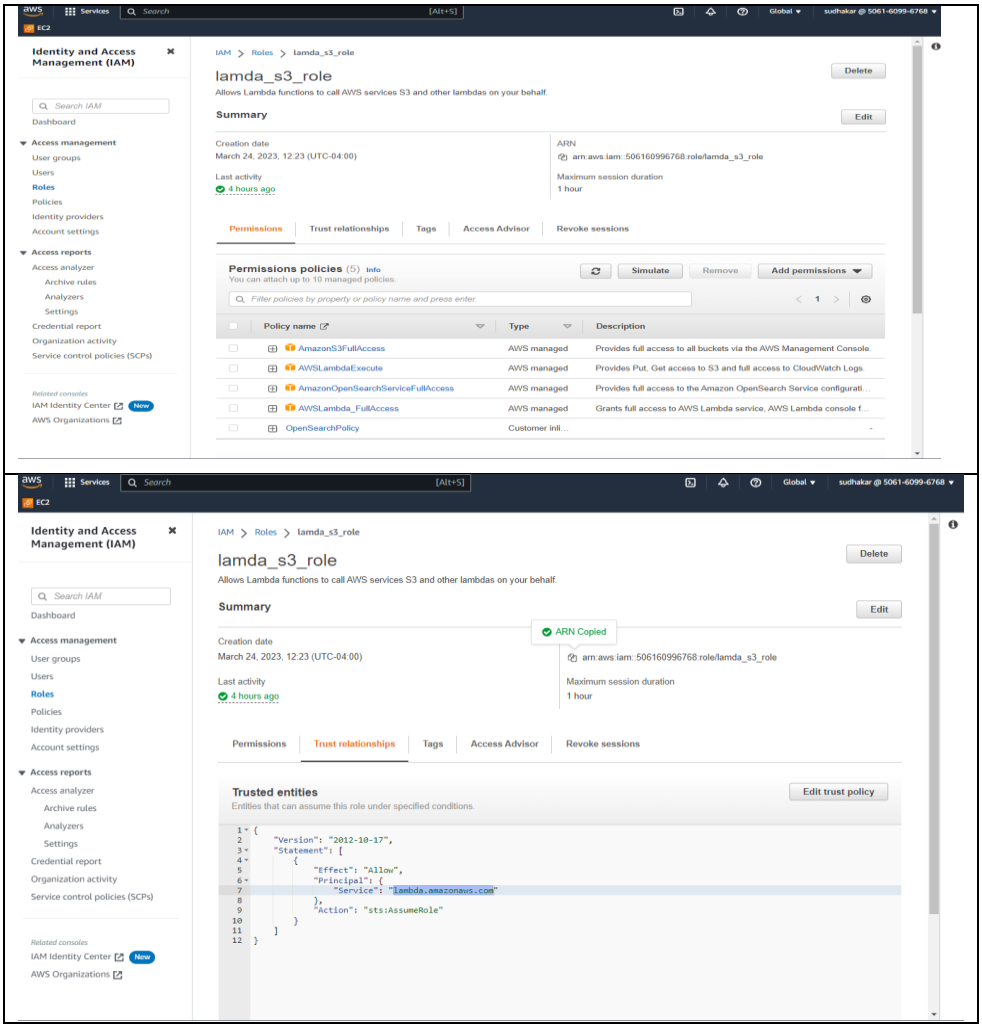


2.0 Create an IAM Role for Lambdas (arn:aws:iam::AWS_ACCTNO:role/lamda_s3_role)

For the workflow the PDF & Upload Lambdas require access to S3 buckets and Search Lambda requires access to OpenSearch Domain. We need to create an IAM role (s3_lambda_role) to grant permissions to

lambda.amazonaws.com to access all S3 buckets, Open Search Domain, CloudWatch and other resources. (AmazonS3FullAccess, OpenSearchServiceFullAccess)

Screenshot

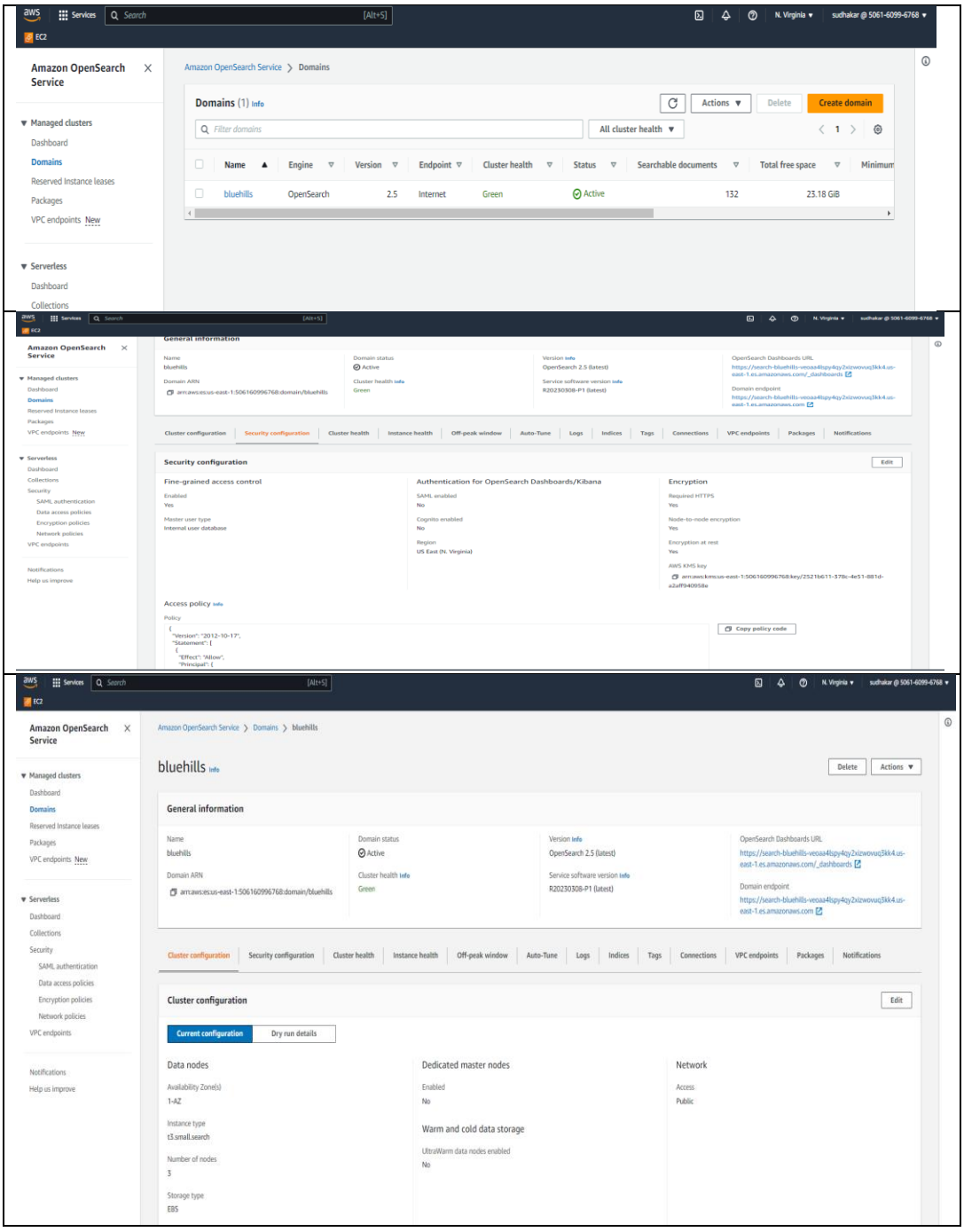


3.0 Create an OpenSearch Domain using the console.

(https://search-bluehills-veoaa4lspy4qy2xizwovuq3kk4.us-east-1.es.amazonaws.com/)

An OpenSearch domain (bluehills) has been created with the configuration parameters (t3.small.search, Number of Nodes 3, EBS Storage type, public network access, fine-grained access control, master-user, master-password).

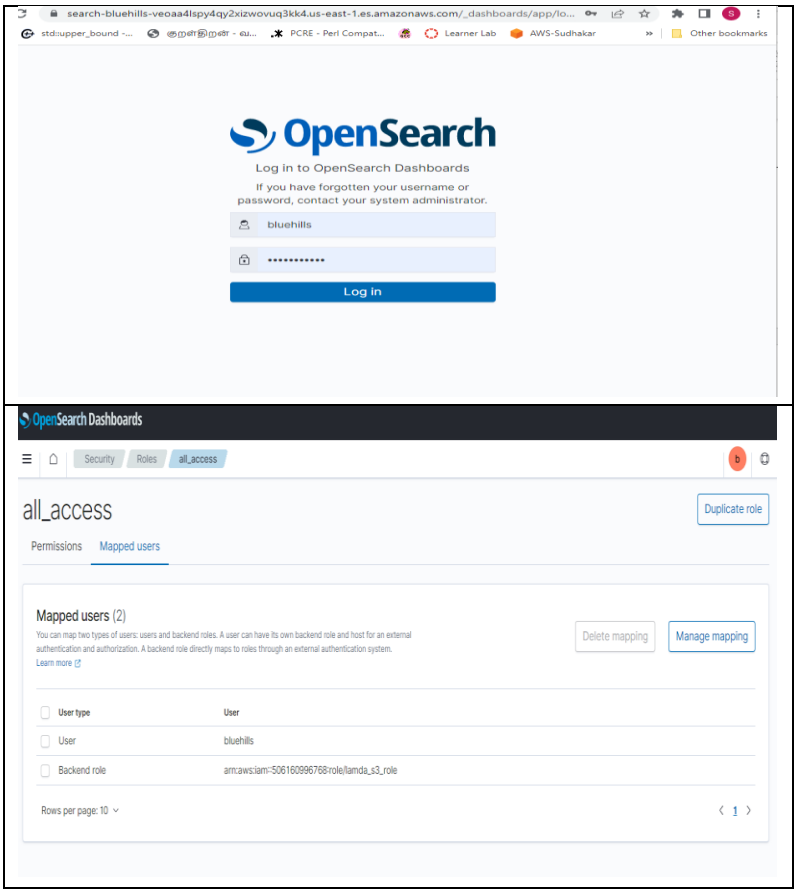
3.1 Screenshot



3.2 login to OpenSearch Domain and assign the OpenSearch ‘all-access’ role to Lambda role

Go to Security/Roles and select ‘all_access’ role. Enter the backend role as (arn:aws:iam::AWS_ACCTNO:role/lamda_s3_role) in ‘Mapped Users’ panel. This will enable the Lambda function to upload and query the OpenSearch domain.

Screenshot

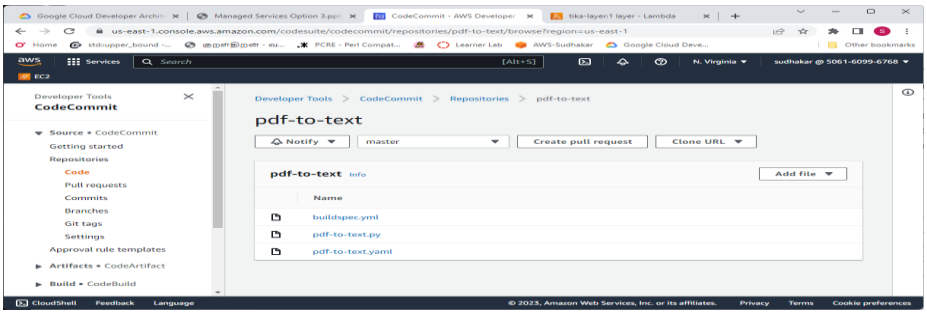


4.0 Implementation of CodeCommit Repositories for the Lambda functions and the SAM templates

4.1 PDF-to-Text Lambda (arn:aws:codecommit:us-east-1:AWS_ACCTNO/pdf-to-text)

S3 Create-Events on the s3://sudhakar-content-store bucket trigger the python script, Pdf-to-text.py to be executed. This reads the PDF document from the S3 bucket, parses the metadata and contents using the tika python library and creates a JSON document in the following scheme: {title: XXXX, date_created: XXXX, description: XXXX, author: xxx, contents: XXXX}. The JSON document is uploaded to the s3://pdftxt-inter bucket. This Lambda is deployed using the pdf-to-text.yaml SAM template file. The source files have been checked in into the arn:aws:codecommit:us-east-1:AWS_ACCTNO/pdf-to-text CodeCommit repository.

4.1.1 Screenshot



4.1.2 Contents

File	Description	Contents
Buildspec.yml	Used by CodeBuild project to generate the deployment package from the template file and store the package in the artifact S3 bucket s3://pdftotxt-artifacts	version: 0.2 phases: install: runtime-versions: python: 3.9 commands: - aws cloudformation package --template-file pdf-to-text.yaml --s3-bucket pdftotxt-artifacts --output-template-file pdf-to-text-output.yaml artifacts: files: - pdf-to-text.yaml - pdf-to-text-output.yaml
Pdf-to-text.yaml	This SAM template file defines the handler function and the IAM role and runtime dependencies (tika-layer) for the Lambda. Also defines the environment variables (TARGET_BUCKET) used by Lambda	AWSTemplateFormatVersion: '2010-09-09' Transform: 'AWS::Serverless-2016-10-31' Description: to install a Lambda function to convert pdf to JSON. Resources: pdftotextstack: Type: 'AWS::Serverless::Function' Properties: Handler: pdf-to-text.lambda_handler Runtime: python3.7 CodeUri: . Description: '' MemorySize: 512 Timeout: 900 Role: 'arn:aws:iam::AWS_ACCTNO:role/lamda_s3_role' Environment: Variables: TARGET_BUCKET: pdftotxt-inter Layers: - 'arn:aws:lambda:us-east-1:AWS_ACCTNO:layer:tika-layer:1'
Pdf-to-text.py	This reads the PDF document, parses the metadata and contents using the tika library, converts it into JSON document and uploads it to the s3://pdftotxt-inter bucket	import boto3 import json import os import re import logging logger = logging.getLogger() logger.setLevel(logging.INFO) def lambda_handler(event, context): logger.info('********** Environment and Event variables are **********') logger.info(os.environ) logger.info(event) extract_content(event) return { 'statusCode': 200, 'body': json.dumps('Execution is now complete') } def convert_pdf_to_json(pdffile): """ Parses and Extracts the metadata (author, title, date-created, description) and contents converts them to JSON string """ import tika from tika import parser tika.initVM() tika_output = parser.from_buffer(pdffile) metadata = tika_output.get('metadata',{}) title = metadata.get("dc:title","No title") created = metadata.get("pdf:docinfo:created","1900/01/01") description = metadata.get("dc:description","No Description") creator = metadata.get("dc:creator","Unknown Author") contents = tika_output.get("content","No Contents") #remove duplicate newlines contents = re.sub('[\n]+', '\n', contents) output = { "title":title, "date_created":created, "description":description, "author":creator, "contents" : contents } return json.dumps(output,indent=2) def extract_content(event): #Read the target bucket from the lambda environment variable targetBucket = os.environ.get('TARGET_BUCKET',"NO_BUCKET") print('Target bucket is', targetBucket) bucket = event['Records'][0]['s3']['bucket']['name'] key = event['Records'][0]['s3']['object']['key'] print('The s3 bucket is', bucket, 'and the file name is', key) s3client = boto3.client('s3') response = s3client.get_object(Bucket=bucket, Key=key) pdffile = response["Body"] print('The binary pdf file type is', type(pdffile))

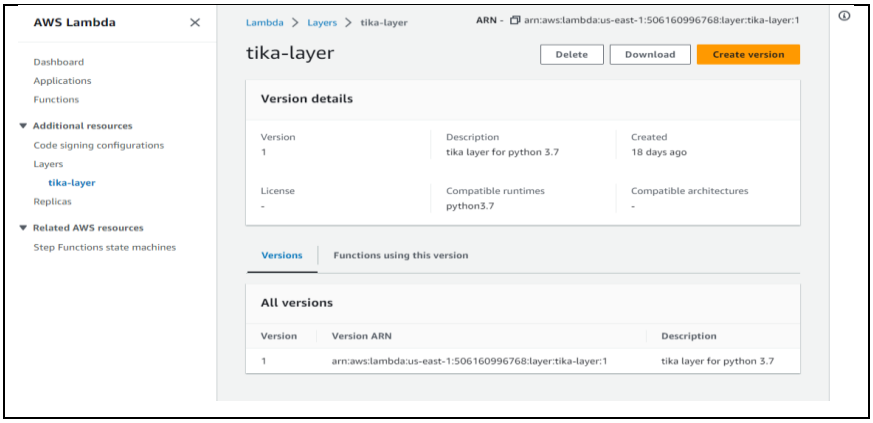
		json_output = convert_pdf_to_json(pdffile) s3client.put_object(Bucket=targetBucket, Key=key+".json", Body=json_output) print('All done, returning from Extract content method')

4.1.3 Creation of the tika Lambda layer *(arn:aws:lambda:us-east-1:AWS_ACCTNO:layer:tika-layer:1)*

The SAM template specifies the tika Python library as a runtime dependency using Lambda Layer. This lambda layer can be generated using the 'prepare-tika-layer' script.

Description	Contents of prepare-tika-layer
This installs tika python library using pip in Python 3.7 virtual environment and packages tika and its dependencies in 'python.zip' file. This file is published to AWS Lambda layers using the AWS CLI command. (aws lambda publish-layer-version)	#!/bin/bash #Use python version 3.7 ver=$(python3 --version \|grep 3.7) if [["$ver" == ""]]; then echo "Python 3.7 needs to be used" exit fi python3 -m venv .venv source .venv/bin/activate pip install tika #prepare requirements.txt for tika echo tika > requirements.txt pip install -r requirements.txt -t python zip -r python.zip python/ aws lambda publish-layer-version --layer-name tika-layer --description "tika layer for python 3.7" --compatible-runtimes python3.7 --zip-file fileb://python.zip

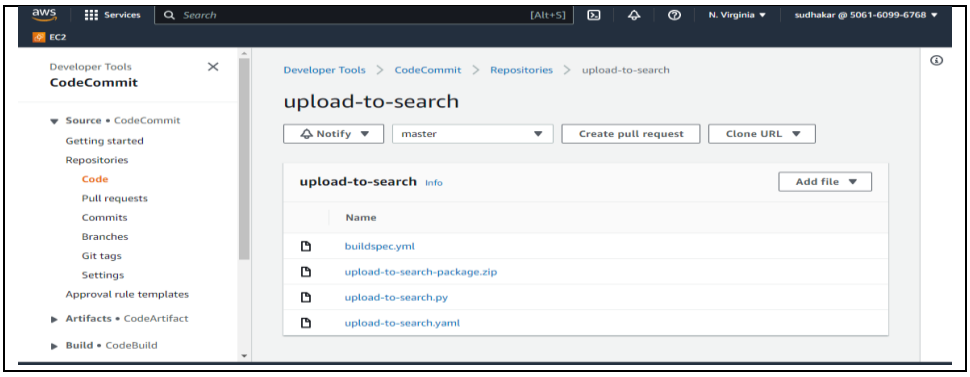
Screenshot



4.2 Upload-to-Search-Domain Lambda (arn:aws:codecommit:us-east-1:AWS_ACCTNO/upload-to-search)

S3 Create-Events on the s3://pdftotxt-inter bucket causes the upload-to-search Lambda function to be executed. This reads the JSON document from this S3 bucket and uploads it to the 'taxforms' index in the OpenSearch Domain defined in the environment variable : SEARCH_DOMAIN.

4.2.1 Screenshot



4.2.2 Repo Contents

File	Description	Contents
buidspec.yml	Used by CodeBuild project to generate a deployment package from the SAM template 'upload-to-search.yaml' and the artifacts are stored in s3://search-domain-project-artifacts	version: 0.2 phases: install: runtime-versions: python: 3.9 commands: - aws cloudformation package --template-file upload-to-search.yaml --s3-bucket search-domain-project-artifacts --output-template-file upload-to-search-output.yaml artifacts: type: zip files: - upload-to-search.yaml - upload-to-search-output.yaml
upload-to-search.yaml	This SAM template defines the Lambda handler function (upload-to-search.handler). The CodeUri refers to a zip file archive which contains the python script and its dependencies. The following section explains the preparation of zip file archive. Also defines the IAM role for the Lambda.	AWSTemplateFormatVersion: '2010-09-09' Transform: 'AWS::Serverless-2016-10-31' Description: Lambda function to upload the documents from s3 bucket to Search Domain # Resources Resources: uploadtosearch: Type: 'AWS::Serverless::Function' Properties: Handler: upload-to-search.handler Runtime: python3.9 CodeUri: './upload-to-search-package.zip' Description: '' MemorySize: 512 Timeout: 900 Role: 'arn:aws:iam::AWS_ACCTNO:role/lamda_s3_role'
upload-to-search-package-zip	This zip file archive contains the python script and its dependencies. Preparation of this archive is explained in the following section.	
upload-to-search.py	This reads the JSON document from the s3://pdftotxt-inter bucket and uploads it to the 'taxforms' index in OpenSearch Domain defined in the environment variable SEARCH_DOMAIN. This uses requests_aws4auth library to generate an auth key to access OpenSearch domain. Requests library is used to post the documents.	import boto3 import re import os import requests import json from requests_aws4auth import AWS4Auth region = 'us-east-1' service = 'es' credentials = boto3.Session().get_credentials() awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service, session_token=credentials.token) print("Credentials access key:", credentials.access_key) print("Credentials secret key:", credentials.secret_key) host = os.environ.get('SEARCH_DOMAIN',None) if not host: print("Environment variable: SEARCH_DOMAIN not defined") index = 'taxforms' datatype = '_doc' url = host + '/' + index + '/' + datatype headers = { "Content-Type": "application/json" } s3 = boto3.client('s3')

		# Lambda execution starts here def handler(event, context): for record in event['Records']: # Get the bucket name and key for the new file bucket = record['s3']['bucket']['name'] key = record['s3']['object']['key'] # Get, read, and split the file into lines obj = s3.get_object(Bucket=bucket, Key=key) body = obj['Body'].read() document = json.loads(body) print("Document:",document) r = requests.post(url, auth=awsauth, json=document, headers=headers) print("Response:", r.text)

4.2.3 Creation of zip file Archive for Upload-to-Search Lambda dependencies

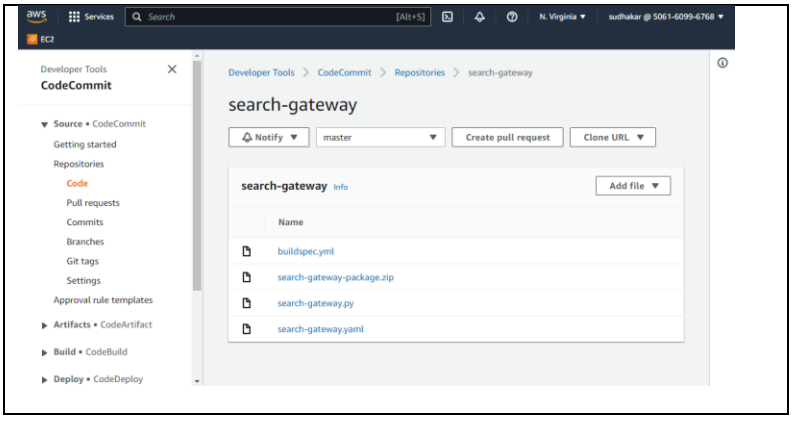
Since the upload-to-search python script uses requests and requests_aws4auth libraries, these dependencies need to be packaged in a zip file archive as part of the deployment. The link below provides the steps needed to prepare the zip file archive. https://docs.aws.amazon.com/lambda/latest/dg/python-package.html

The archive needs to be generated in python 3.9 environment because of the runtime requirements. The following commands can be used to generate the zip file archive.

Description	Commands to generate zip archive file for requests & requests_aws4auth libraries
Make sure python3.9 is installed in your environment. If not , Dockerfile with python:3.9 can be used pull the python3.9 image and run the container based on it. Install the requests and requests_aws4auth libraries in a 'package' folder and zip the package folder recursively. Also add the python script to the zip package. This zip file archive is checked in into the CodeCommit repository.	#go to repo folder where upload-to-search.py is stored cd ~/environments/repos/upload-to-search.py #use the virtual environment source .venv/source/bin/activate pip install –target ./package requests pip install –target ./package requests_aws4auth cd package zip -r ../upload-to-search-package.zip . cd .. zip upload-to-search-package.zip upload-to-search.py

4.3 Search-Gateway Lambda (arn:aws:codecommit:us-east-1:AWS_ACCTNO/search-gateway)

4.3.1 Screenshot



4.3.2 Repo Contents

File	Description	Contents
buildspec.yml	Used by CodeBuild Project to generate a deployment package	version: 0.2 phases: install: runtime-versions:

	from the search-gateway.yaml template file and store the artifacts in s3://search-domain-project-artifacts.	python: 3.9 commands: - aws cloudformation package --template-file search-gateway.yaml --s3-bucket search-domain-project-artifacts --output-template-file search-gateway-output.yaml artifacts: type: zip files: - search-gateway.yaml - search-gateway-output.yaml
search-gateway.yaml	This SAM cloudformation template specified the handler function(search-gateway.lambda_handler) for the Lambda. The CodeUri refers to a zip file archive which contains the python script and its dependencies. The following section explains the preparation of zip file archive. Also defines the IAM role for the Lambda.	AWSTemplateFormatVersion: '2010-09-09' Transform: 'AWS::Serverless-2016-10-31' Description: Serverless template to install lambda to query the search gateway Resources: searchgateway: Type: 'AWS::Serverless::Function' Properties: Handler: search-gateway.lambda_handler Runtime: python3.9 CodeUri: './search-gateway-package.zip' Description: '' MemorySize: 128 Timeout: 300 Role: 'arn:aws:iam::AWS_ACCTNO:role/lamda_s3_role'
search-gateway-package.zip	This zip file archive contains the python script and its dependencies. Preparation of this archive is explained in the following section.	
search-gateway.py	This script is triggered by get-request from the API Gateway. This parses the get-request ,extracts the q argument, queries the OpenSearch Doman based on the q parameter, returns the results to the API gateway.	import boto3 import json import os import requests from requests_aws4auth import AWS4Auth region = 'us-east-1' service = 'es' credentials = boto3.Session().get_credentials() awsauth = AWS4Auth(credentials.access_key, credentials.secret_key, region, service,session_token=credentials.token) host = os.environ.get('SEARCH_DOMAIN','NO_SEARCH_DOMAIN') index = 'taxforms' url = host + '/' + index + '/_search' # Lambda execution starts here def lambda_handler(event, context): print('Event',event) query = { "size": 25, "query": { "multi_match": { "query": event['queryStringParameters']['q'], "fields": ["author", "date_created","title","description"] } } } headers = { "Content-Type": "application/json" } r = requests.get(url, auth=awsauth, headers=headers, data=json.dumps(query)) response = { "statusCode": 200, "headers": { "Access-Control-Allow-Origin": '*' }, "isBase64Encoded": False } response['body'] = r.text return response

4.3.3 Creation of zip file Archive for Upload-to-Search Lambda dependencies

Since the search-gateway python script uses requests and requests_aws4auth libraries, these dependencies need to be packaged in a zip file archive as part of the deployment. The link below provides the steps needed to prepare the zip file archive. https://docs.aws.amazon.com/lambda/latest/dg/python-package.html

The archive needs to be generated in python 3.9 environment because of the runtime requirements. The following commands are like those given in section 1.2.3 except for the Lambda Handler script.

Description	Commands to generate zip archive file for requests & requests_aws4auth libraries
Make sure python3.9 is installed in your environment. If not , Dockerfile with python:3.9 can be used pull the python3.9 image and run the container based on it. Install the requests and requests_aws4auth libraries in a 'package' folder and zip the package folder recursively. Also add the python script to the zip package. This zip file archive is checked in into the CodeCommit repository.	#go to repo folder where upload-to-search.py is stored cd ~/environments/repos/upload-to-search.py #use the virtual environment source .venv/source/bin/activate pip install –target ./package requests pip install –target ./package requests_aws4auth cd package zip -r ../search-gateway-package.zip . cd .. zip search-gateway-package.zip search-gateway.py

5.0 Implementation of CodeBuild Projects for PDF, Upload, and Search Lambdas

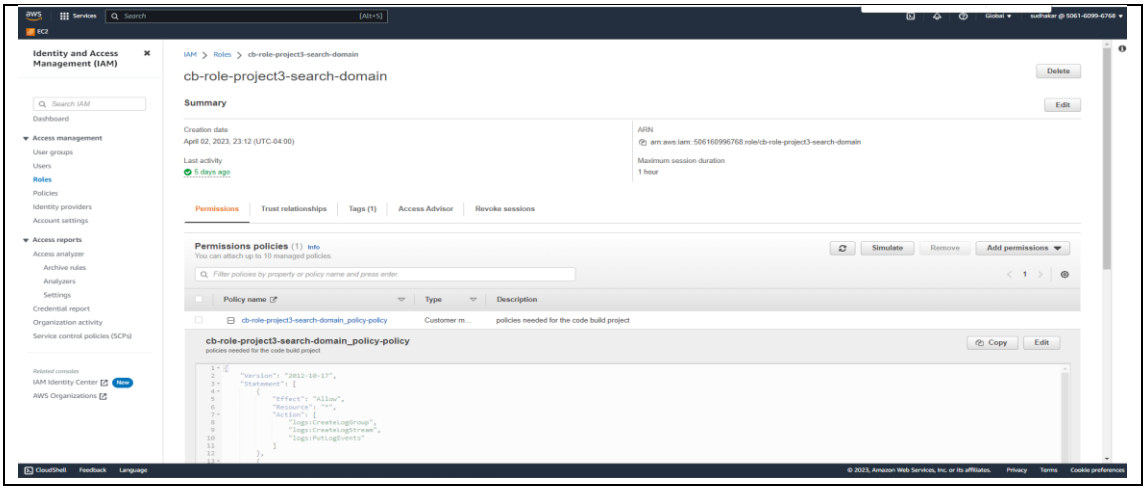
The CodeBuild projects need to pull the source files from CodeCommit repos, build the artifacts based on the template files, store them in artifact S3 buckets, and log the build events in CloudWatch. An IAM role needs to be implemented to grant permissions to CodeCommit, S3 Buckets, CloudWatch Log Groups.

5.1 Create an IAM role for the CodeBuild Projects

A boto3 python script has been implemented to create the IAM role.

Description	Contents of 'create-role-cb.py'
This script a. Creates a policy (cb-role-project3-search-domain-policy) to access CodeCommit, S3, report groups, Cloud watch. b. Creates a role (cb-role-project3-search-domain) with CodeProject as trusted entity. c. Attaches the role to the policy. The entire contents of the scripts are not shown. Please see the attached script file 'create-role-cb.py'	#!/usr/bin/env python3 import boto3 import json iam = boto3.client('iam') my_region = boto3.session.Session().region_name my_account = boto3.client('sts').get_caller_identity()['Account'] def get_assume_role_policy(principal : str): … def get_cloudwatch_policy_statement(): …. def get_s3_artifact_policy_statement() : … def get_codecommit_policy_statement(): … def get_report_group_policy_statement(): … def create_codebuild_policy(policy_name : str,tags : list): … def create_codebuild_role(role_name : str, tags : list): #Create a role to be used/assumed by codebuild role = iam.create_role(RoleName=role_name, AssumeRolePolicyDocument=json.dumps(get_assume_role_policy('codebuild')), Tags=tags) #Create necessary resource access policies and attach it to the role policy = create_codebuild_policy(policy_name=f'{role_name}_policy',tags=tags) iam.attach_role_policy(PolicyArn=policy['Policy']['Arn'], RoleName=role_name) return role cb_role = create_codebuild_role(role_name='cb-role-project3-search-domain', tags=[{'Key': 'project-name','Value': 'project3-search-domain'}]) print(cb_role)

Screenshot



5.2 Create CodeBuild Projects

A boto3 python script has been developed to create the following CodeBuild projects for PDF, Update, and Search Lambdas.

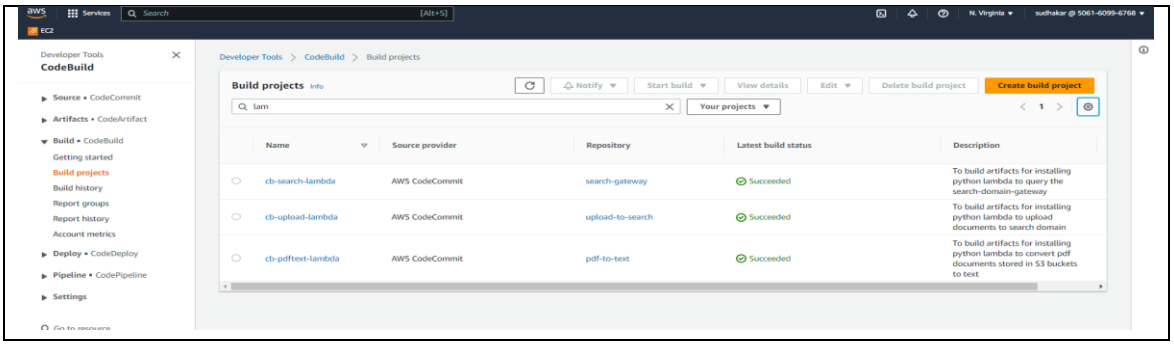
CodeBuild Project	CodeCommit Repository
cb-pdftext-lambda	pdf-to-text
Cb-upload-lambda	Upload-to-search
Cb-search-lambda	Search-gateway

The table below lists the python script.

Description	Contents of 'create-cb.py'
This script creates the following CodeBuild projects	#!/usr/bin/env python3 import boto3 client_codebuild = boto3.client('codebuild') def create_codebuild_project(name : str, desc : str, repo_url : str , artifact_bucket : str, service_role : str, tags : list): ''' name: CodeBulid Project repo_url: Source Git Repository artifact_bucket: S3 bucket to store the Codebuild artifacts service_role: to grant necessary permissions to codebuild service ''' source={ 'type': 'CODECOMMIT', 'location': repo_url, 'gitCloneDepth': 0, 'gitSubmodulesConfig': { 'fetchSubmodules': True }, 'buildspec': '' } artifacts={ 'type': 'S3', 'location': artifact_bucket, 'path': '', 'namespaceType': 'NONE', 'name': '', 'packaging': 'NONE', 'overrideArtifactName': True, 'encryptionDisabled': True, 'artifactIdentifier': '', 'bucketOwnerAccess': 'NONE' } environment={

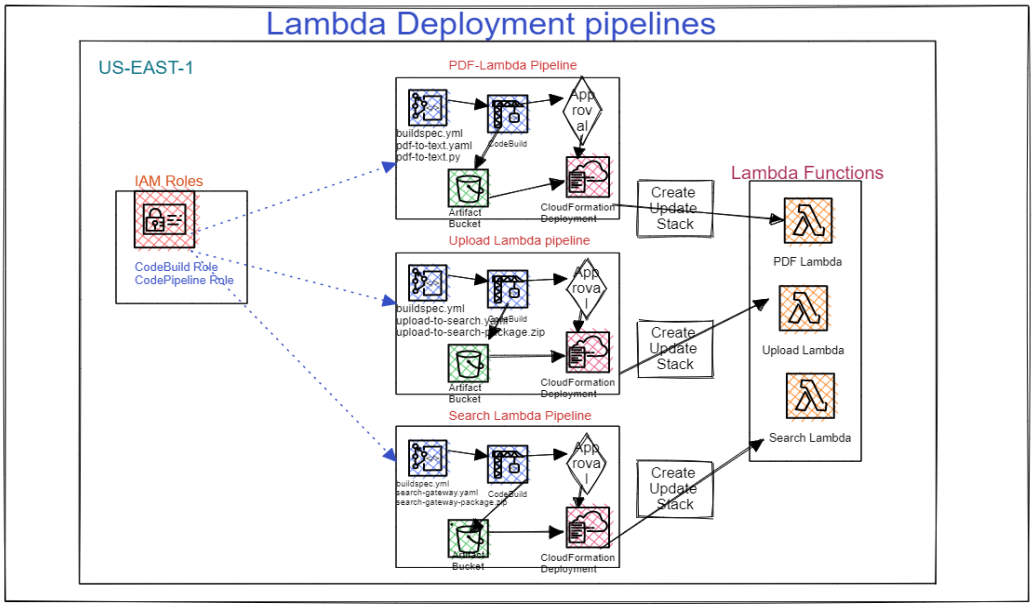
```
            'type': 'LINUX_CONTAINER',
            'image': 'aws/codebuild/standard:4.0',
            'computeType': 'BUILD_GENERAL1_SMALL',
        }

    return client_codebuild.create_project(
        name=name,
        description=desc,
        source=source,
        environment=environment,
        artifacts=artifacts,
        serviceRole=service_role,
        tags=tags)


#S3 bucket to store all codebuild project artifacts
artifact_bucket = 'search-domain-project-artifacts'
#Service role for the codebuild project to allow S3 /CodeCommit/LogGroup/Report actions
service_role = 'arn:aws:iam::AWS_ACCTNO:role/cb-role-project3-search-domain'
tags = [{'key': 'project-name','value': 'search-domain'}]


def create_codebuild_project_pdftext_lambda():
    return create_codebuild_project(name = 'cb-pdftext-lambda',
        desc='To build artifacts for installing python lambda to convert pdf documents stored in S3 buckets to text',
        repo_url='https://git-codecommit.us-east-1.amazonaws.com/v1/repos/pdf-to-text',
        artifact_bucket=artifact_bucket,
        service_role=service_role,
        tags=tags)

def create_codebuild_project_upload_lambda():
    return create_codebuild_project(name = 'cb-upload-lambda',
        desc='To build artifacts for installing python lambda to upload documents to search domain',
        repo_url='https://git-codecommit.us-east-1.amazonaws.com/v1/repos/upload-to-search',
        artifact_bucket=artifact_bucket,
        service_role=service_role,
        tags=tags)

def create_codebuild_project_search_lambda():
    return create_codebuild_project(name = 'cb-search-lambda',
        desc='To build artifacts for installing python lambda to query the search-domain-gateway',
        repo_url='https://git-codecommit.us-east-1.amazonaws.com/v1/repos/search-gateway',
        artifact_bucket=artifact_bucket,
        service_role=service_role,
        tags=tags)

print(create_codebuild_project_pdftext_lambda())
print(create_codebuild_project_upload_lambda())
print(create_codebuild_project_search_lambda())
```

Screenshot



6.0 Create CodePipelines for deploying PDF, Upload, and Search Lambdas

The diagram below shows the pipelines used to deploy the PDF, Upload, and Search Lambdas. Each pipeline has 4 stages (Source, Build, Approval, and Deploy). Boto3 Python scripts have been implemented to create CodePipelines and the IAM roles required by them.



The table below lists the script used for creating the IAM role: *arn:aws:iam::AWS_ACCTNO:role/role-codepipeline-project3-search-domain*

Description	Contents of 'create-role-pipeline.py
This script creates 'role-codepipeline-project3-search-domain' (arn:aws:iam::AWS_ACCTNO:role/role-codepipeline-project3-search-domain) to permt CodePipeline and Cloudformation to access S3 buckets, CodeCommit repositories, CodeDeploy functions and Lambda functions. The entire contents of the script are not shown here. Please refer to the attached file 'create-role-pipeline.py'	#!/usr/bin/env python3 import boto3 import json iam = boto3.client('iam') my_region = boto3.session.Session().region_name my_account = boto3.client('sts').get_caller_identity()['Account'] def get_assume_role_policy(principals : list): def get_statement(principal : str): return { "Sid": "", "Effect": "Allow", "Principal": { "Service": [f"{principal}.amazonaws.com"] }, "Action": "sts:AssumeRole" } return { "Version": "2012-10-17", "Statement": [get_statement(ppl) for ppl in principals] } codepipeline_policy_statements = [.....] def create_codepipeline_policy(policy_name : str,tags : list): code_policy_doc = json.dumps({ "Version": "2012-10-17", "Statement": codepipeline_policy_statements }) return iam.create_policy(PolicyName=f'{policy_name}-policy', Description=f'policies needed for the codepipeline project ', PolicyDocument=code_policy_doc, Tags=tags) def create_codepipeline_role(role_name : str, tags : list): #Create a role to be used/assumed by codepipeline role = iam.create_role(RoleName=role_name, AssumeRolePolicyDocument=json.dumps(get_assume_role_policy(['codepipeline','cloudformation'])), Tags=tags)

	#Create necessary resource access policies and attach it to the role policy = create_codepipeline_policy(policy_name=f'{role_name}_policy',tags=tags) iam.attach_role_policy(PolicyArn=policy['Policy']['Arn'], RoleName=role_name) iam.attach_role_policy(PolicyArn='arn:aws:iam::aws:policy/AWSLambda_FullAccess', RoleName=role_name) return role codepipeline_role = create_codepipeline_role(role_name='role-codepipeline-project3-search-domain', tags=[{'Key': 'project-name','Value': 'project3-search-domain'}]) print(codepipeline_role)

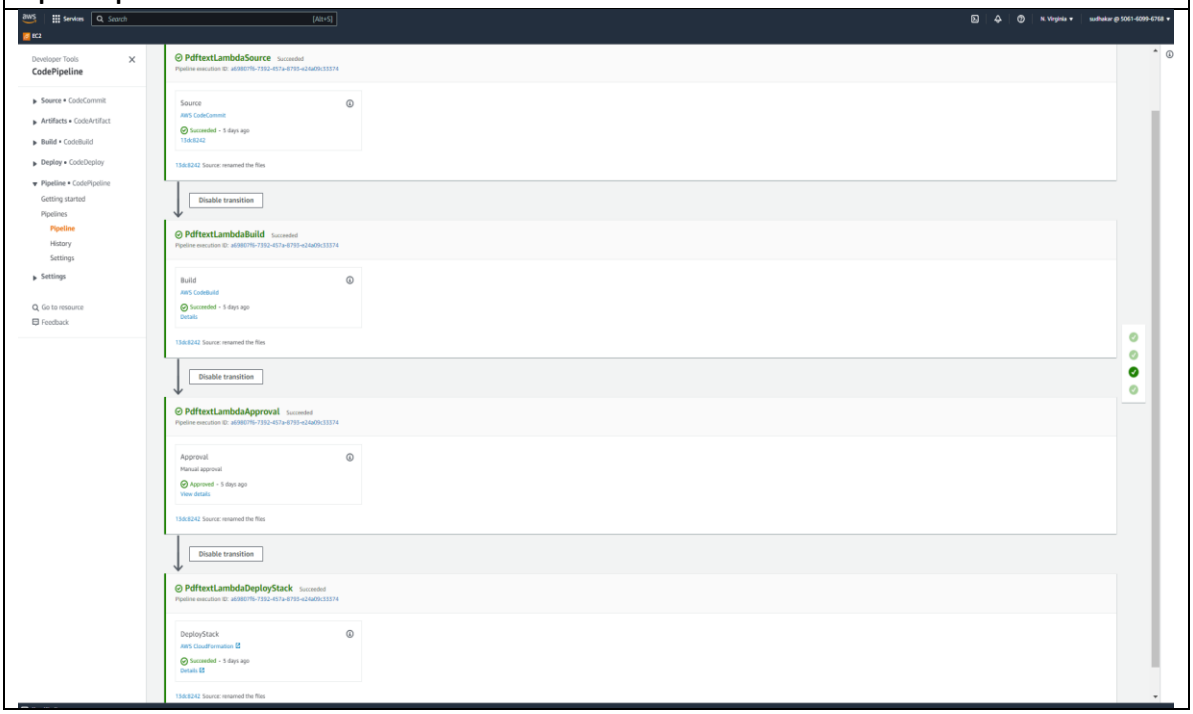
The table below lists the Boto3 python script used for creating the pipelines shown in the above diagram.

Description	Contents of 'create-pipeline.py'
This script creates the following 4-Stage pipelines. Pipeline-pdf-lambda Pipeline-upload-lambda Pipeline-search-lambda S3://search-domain-project-artifacts bucket is used for storing the pipeline artifacts. The entire contents of the script are not shown here. Please refer to the attached file 'create--pipeline.py'	#!/usr/bin/env python3 import boto3 codepipeline_client = boto3.client('codepipeline') codecommit_client = boto3.client('codecommit') codebuild_client = boto3.client('codebuild') region = boto3.session.Session().region_name account = boto3.client('sts').get_caller_identity()['Account'] def create_4_stage_pipeline(prefix : str, #prefix for stage names pipeline_name : str,pipeline_role : str,pipeline_artifact : str, #pipeline details source_repo_name : str, #codecommit -- Source stage codebuild_project_name : str, #codebuild - Build stage sns_arn : str, #sns for approval stage stack_name : str, stack_template_file : str,#stack name and template file for cloudformation deployment tags : list): #tags if not does_codebuildproject_exist(codebuild_project_name): raise RuntimeError(f'Code bulid project {codebuild_project_name} does not exist') if not does_repo_exist(source_repo_name): raise RuntimeError(f'CodeCommit Repo {source_repo_name} does not exist') source_stage = {... } build_stage = { } approval_stage = { } deploy_stage = { } pipeline = { "name": pipeline_name, "roleArn": pipeline_role, "artifactStore": { "type": "S3", "location": pipeline_artifact }, "stages": [source_stage , build_stage , approval_stage, deploy_stage,], "version": 1 } if does_pipeline_exist(pipeline_name): return codepipeline_client.update_pipeline(pipeline=pipeline) return codepipeline_client.create_pipeline(pipeline=pipeline,tags=tags) pipeline_role = "arn:aws:iam::AWS_ACCTNO:role/role-codepipeline-project3-search-domain" pipeline_artifact = "search-domain-project-artifacts" sns_arn = "arn:aws:sns:us-east-1:AWS_ACCTNO:BuildStatus" tags=[{'key': 'project-name','value': 'project3-search-domain'}] def create_pipeline_search_lambda():

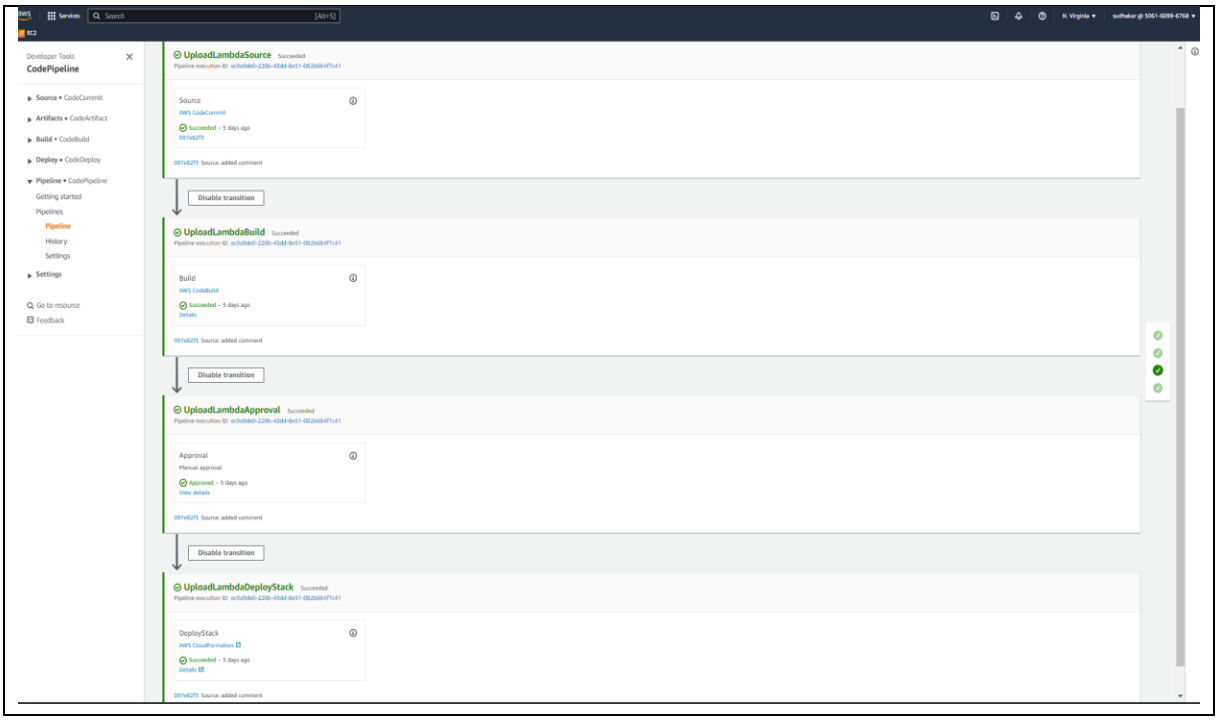
```
    return create_4_stage_pipeline(prefix="SearchLambda",
        pipeline_name='pipeline-search-lambda',
        pipeline_role=pipeline_role,
        pipeline_artifact=pipeline_artifact,
        source_repo_name="search-gateway",
        codebuild_project_name="cb-search-lambda",
        sns_arn=sns_arn,
        stack_name="stack-search-lambda",
        stack_template_file=f"search-gateway-output.yaml",
        tags=tags)

def create_pipeline_upload_lambda():
    return create_4_stage_pipeline(prefix="UploadLambda",
        pipeline_name='pipeline-upload-lambda',
        pipeline_role=pipeline_role,
        pipeline_artifact=pipeline_artifact,
        source_repo_name="upload-to-search",
        codebuild_project_name="cb-upload-lambda",
        sns_arn=sns_arn,
        stack_name="stack-upload-lamda",
        stack_template_file="upload-to-search-output.yaml",
        tags=tags)

def create_pipeline_pdftext_lambda():
    return create_4_stage_pipeline(prefix="PdftextLambda",
        pipeline_name='pipeline-pdftext-lambda',
        pipeline_role=pipeline_role,
        pipeline_artifact=pipeline_artifact,
        source_repo_name="pdf-to-text",
        codebuild_project_name="cb-pdftext-lambda",
        sns_arn=sns_arn,
        stack_name="stack-pdftext-lamda",
        stack_template_file="pdf-to-text-output.yaml",
        tags=tags)

print(create_pipeline_pdftext_lambda())
print(create_pipeline_upload_lambda())
print(create_pipeline_search_lambda())
```

Screenshots of pipelines

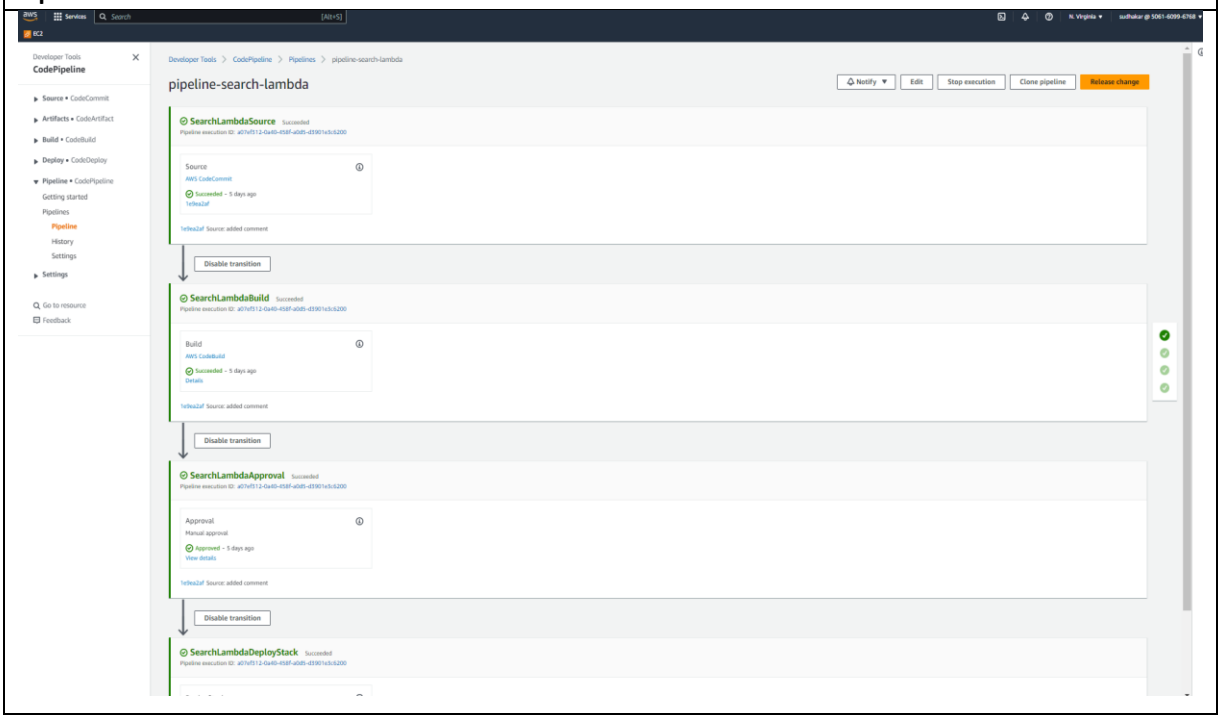
Pipeline-pdf-lambda



Pipeline-upload-lambda

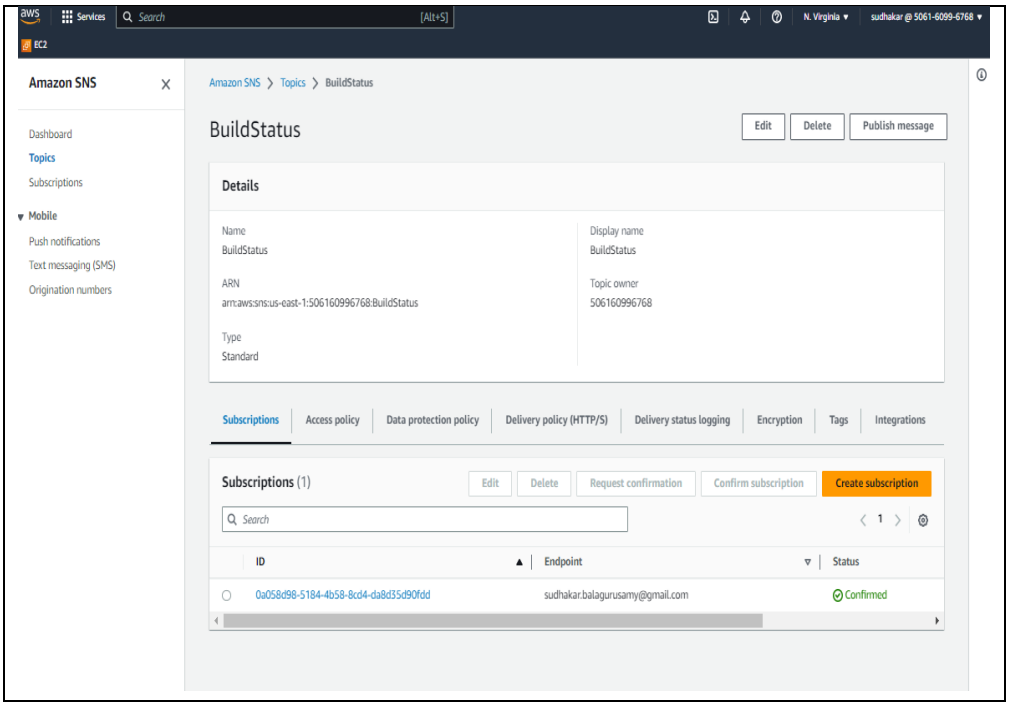


Pipeline-Search-Lambda

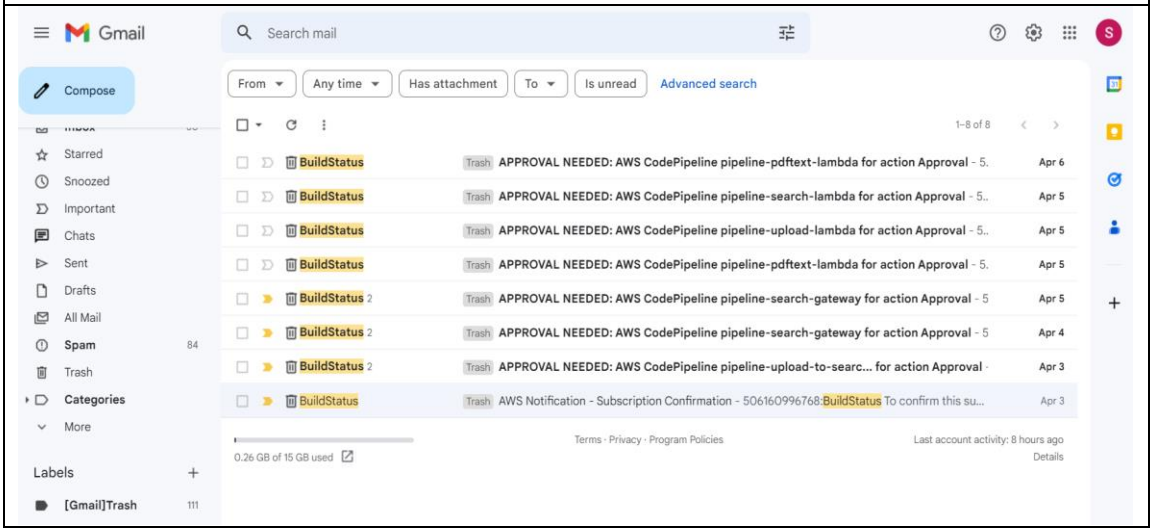


Approval stage uses the following SNS BuildStatus notification to email YOUREMAIL@gmail.com

SNS BuildStatus Topic screenshot



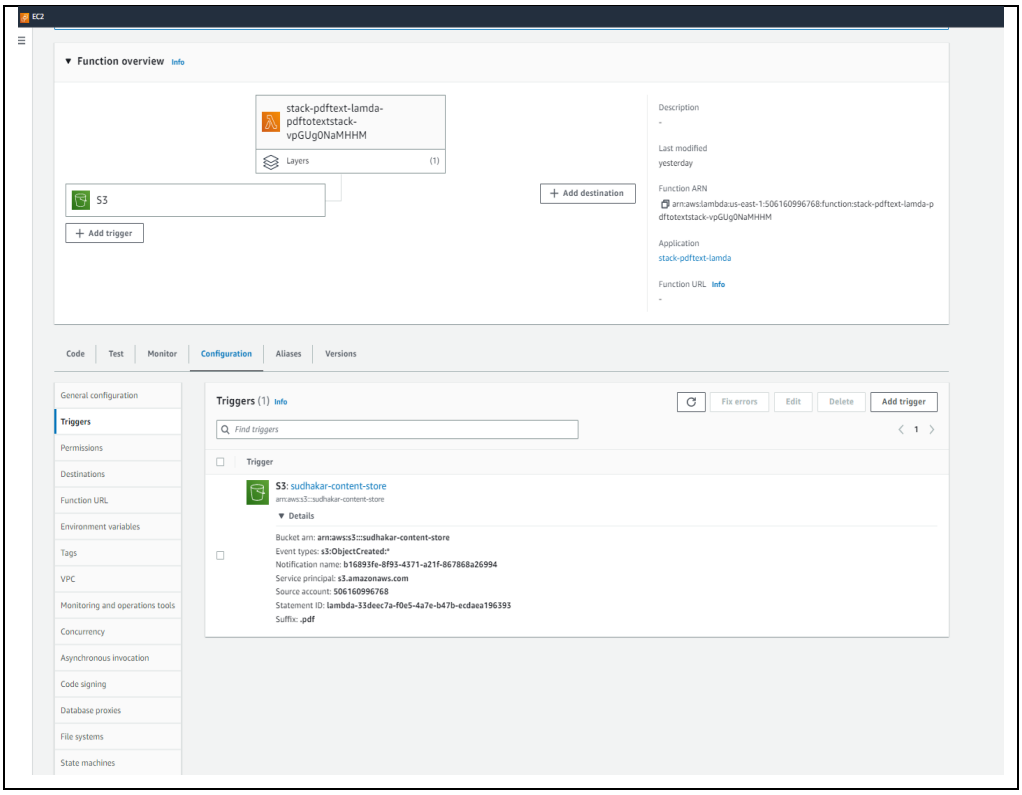
Emails from AWS BuildStatus SNS notification



7.0 S3 bucket Triggers for PDF and Upload Lambdas

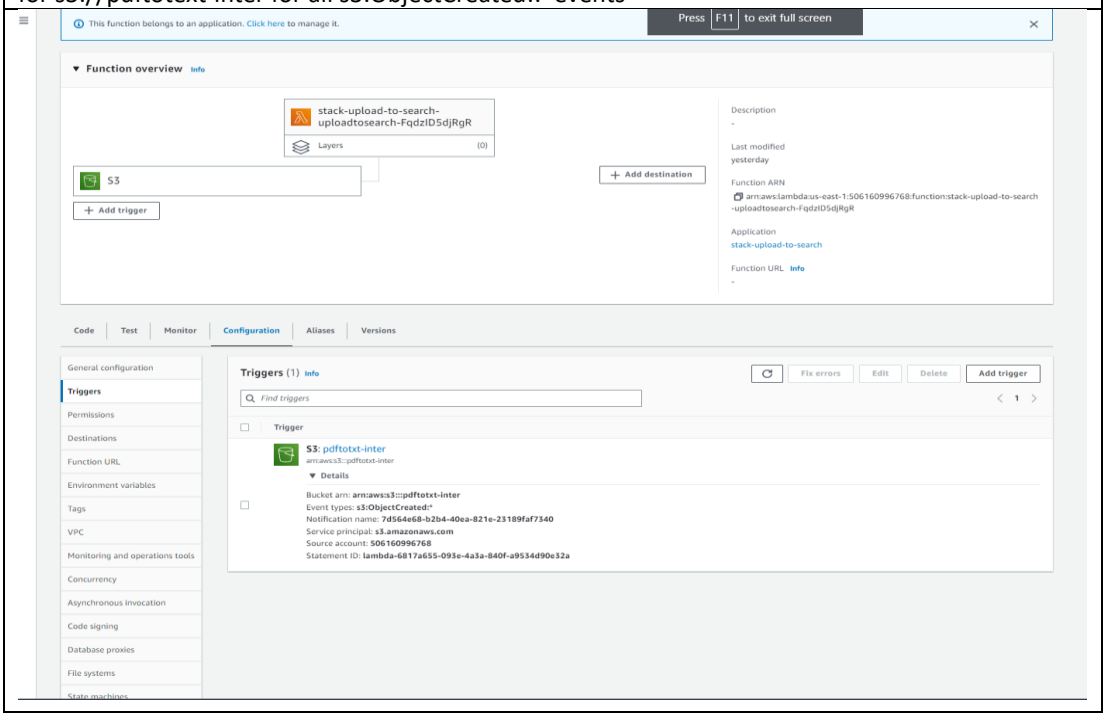
7.1 Trigger for arn:aws:lambda:us-east-1:AWS_ACCTNO:function:stack-pdftext-lamda-pdftotextstack-vpGUg0NaMHHM

Locate the trigger corresponding to the PDF Lambda and add an S3 trigger for s3://sudhakar-content-store for all s3:ObjectCreated::*events



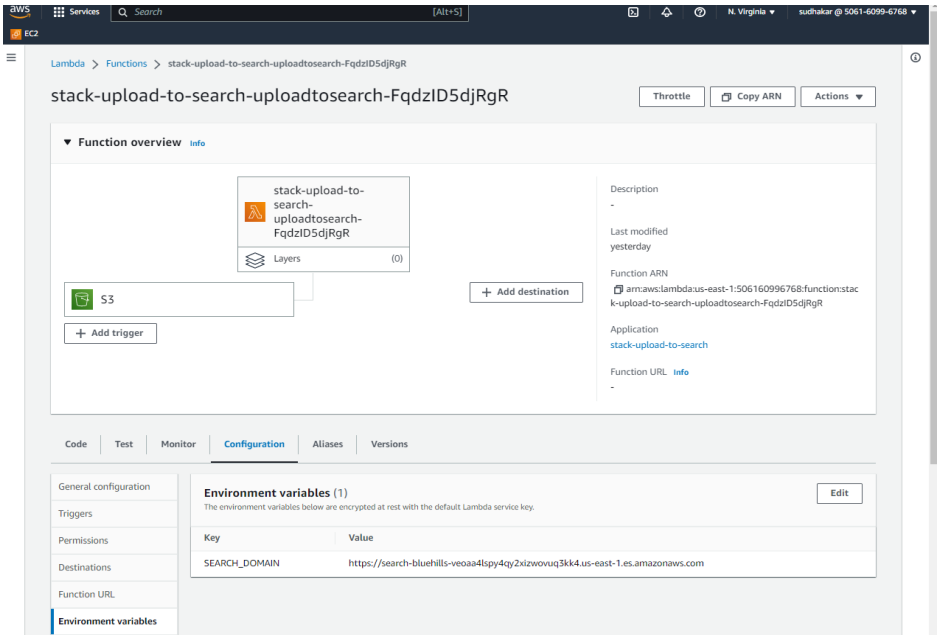
7.2 Trigger for arn:aws:lambda:us-east-1:AWS_ACCTNO:function:stack-upload-to-search-uploadtosearch-FqdzID5djRgR

Locate the trigger corresponding to the PDF Lambda and add an S3 trigger for s3://pdftotext-inter for all s3:ObjectCreated::*events



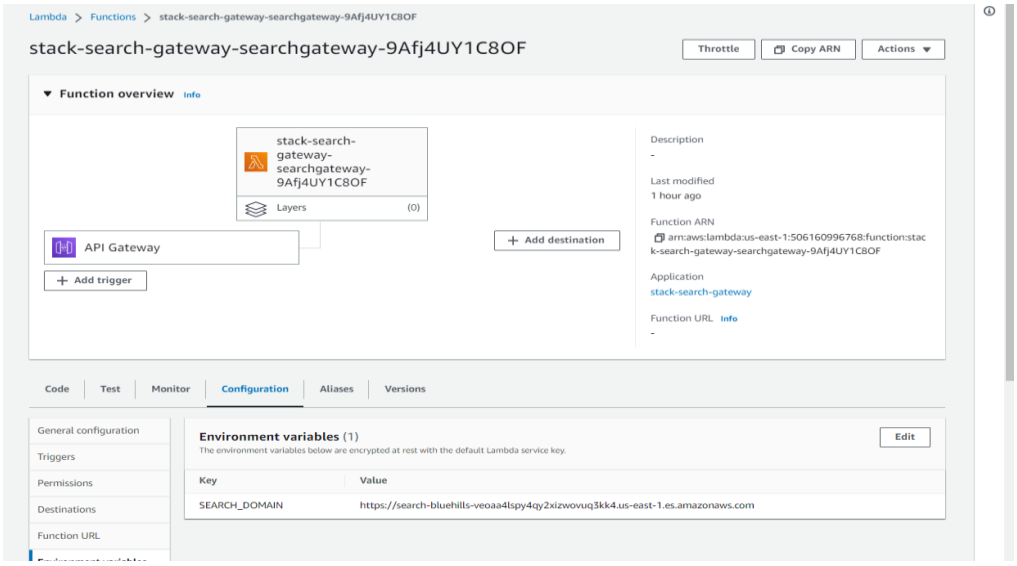
7.3 Modify the Update Lambda configuration.

Set the environment variable SEARCH_DOMAIN to https://search-bluehills-veoaa4lspy4qy2xizwovuq3kk4.us-east-1.es.amazonaws.com



7.3 Modify the Search Lambda configuration.

Set the environment variable SEARCH_DOMAIN to https://search-bluehills-veoaa4lspy4qy2xizwovuq3kk4.us-east-1.es.amazonaws.com



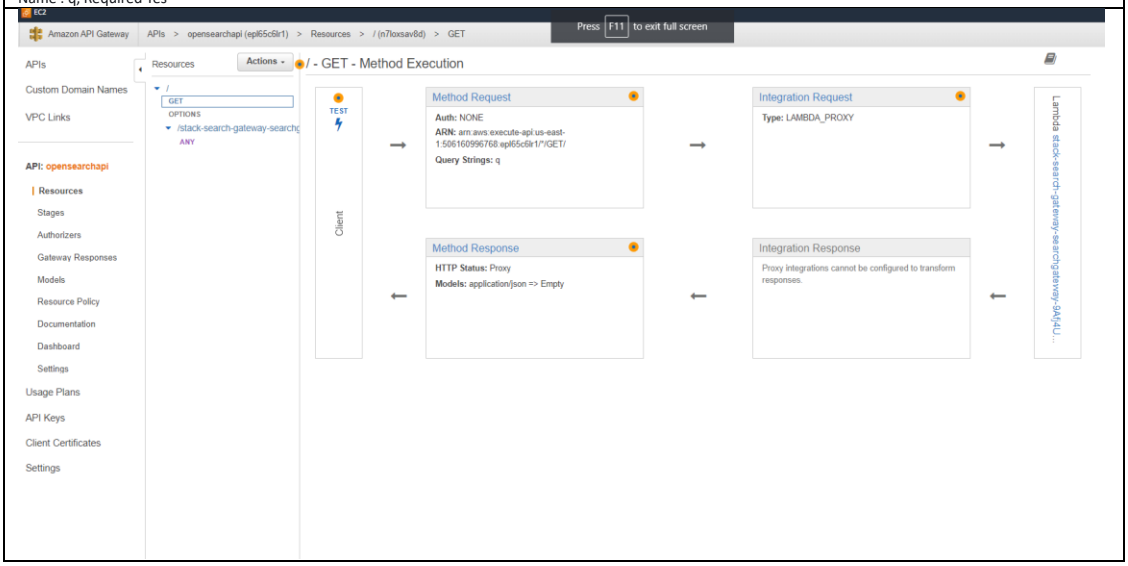
8.0 Create an API using API Gateway to query the OpenSearch Domain using the Search Lambda

The API was created based on the following link.

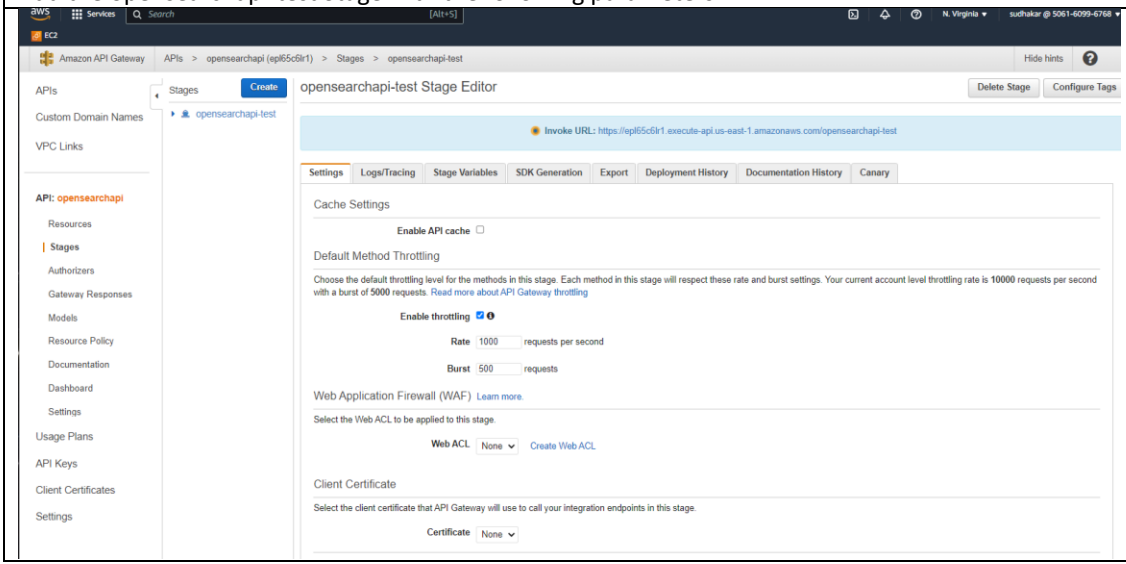
https://docs.aws.amazon.com/opensearch-service/latest/developerguide/search-example.html

8.1 Create opensearchapi using the API Gateway Console

Create a REST API (opensearchapi) using the Build button in API Gateway Console
Add GET -Method Execution from “Actions” Use the following settings.
Integration Type: Lambda function,
Lambda Proxy integration,
Lambda Function: stack-search-gateway-searchgateway-9Afj4UY1C8OF
Configure the Method Request, Validate Query String parameters and headers
Under URL Query String Parameters, choose Add query string
Name : q, Required Yes



Add the opensearchapi-test Stage with the following parameters

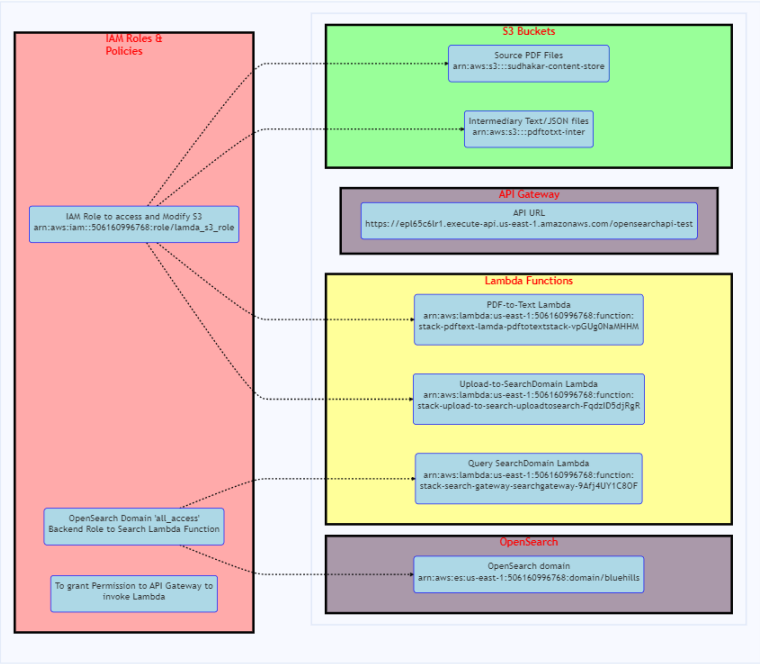


Note the URL https://epl65c6lr1.execute-api.us-east-1.amazonaws.com/opensearchapi-test

8.2 Grant permission to apigateway.amazonaws.com to invoke the Search Lambda function.

This CLI command grants permission to gateway API to invoke the lambda function	aws lambda add-permission --function-name "arn:aws:lambda:us-east-1:AWS_ACCTNO:function:stack-search-gateway-searchgateway-9Afj4UY1C8OF" --action lambda:InvokeFunction --statement-id gatewaystmt --principal apigateway.amazonaws.com

8.3 Summary of IAM Roles



9.0 Setup a web page to test the API (https://epl65c6lr1.execute-api.us-east-1.amazonaws.com/opensearchapi-test)

9.1 Modify the search.js to change the Gateway API endpoint and display the contents in textarea

Modify the endpoint

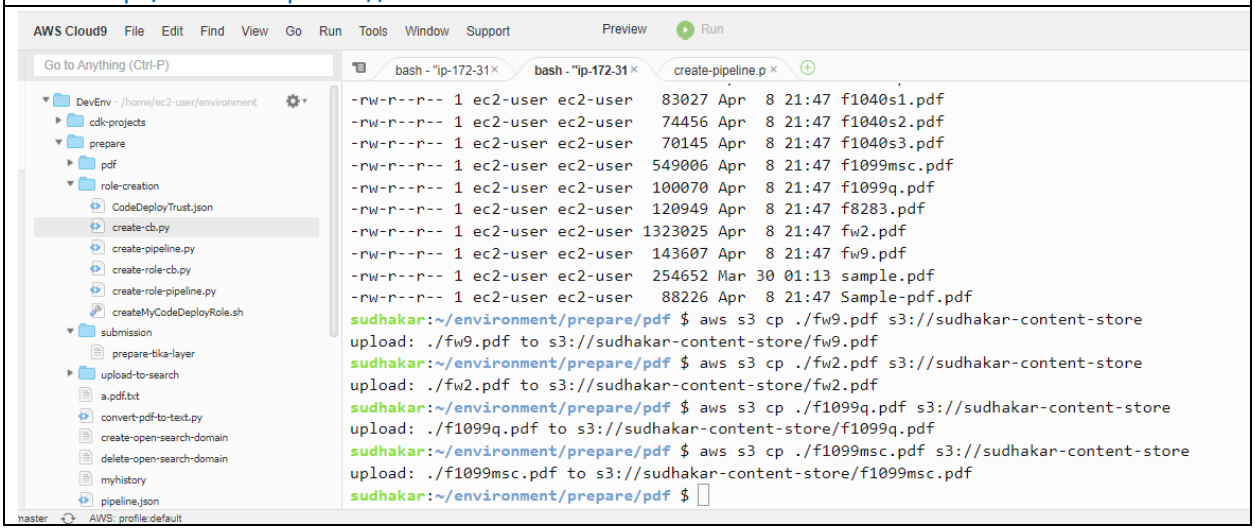
```
// Update this variable to point to your domain.
var apigatewayendpoint = 'https://epl65c6lr1.execute-api.us-east-1.amazonaws.com/opensearchapi-test'
var loadingdiv = $('#loading');
var noresults = $('#noresults');
var resultdiv = $('#results');
var searchbox = $('input#search');
var timer = 0;

// Executes the search function 250 milliseconds after user stops typing
searchbox.keyup(function () {
  clearTimeout(timer);
  timer = setTimeout(search, 250);
});

async function search() {
  // Clear results before searching
  noresults.hide();
  resultdiv.empty();
  loadingdiv.show();
  // Get the query from the user
  let query = searchbox.val();
  // Only run a query if the string contains at least three characters
  if (query.length > 2) {
    // Make the HTTP request with the query as a parameter and wait for the JSON results
    let response = await $.get(apigatewayendpoint, { q: query, size: 25 }, 'json');
                    console.log(response)

    let results = response['hits']['hits'];
    if (results.length > 0) {
      loadingdiv.hide();
      // Iterate through the results and write them to HTML
      resultdiv.append('<p>Found ' + results.length + ' results.</p>');
      for (var item in results) {
        let description = results[item]._source.description;
        let title = results[item]._source.title;
        let contents = results[item]._source.contents;
        let date_created = results[item]._source.date_created;
                                let author = results[item]._source.author;
        // Construct the full HTML string that we want to append to the div
        resultdiv.append('<div class="result">' +
        '<div><h2>' + title + '</h2><p>' + description + ' &mdash; ' + date_created + '</p></div>' +
```

```
                                        '<textarea  rows="40" cols="90">'+ contents +'</textarea>'+
                                        '</div>');
    }
   } else {
     noresults.show();
   }
  }
  loadingdiv.hide();
}

// Tiny function to catch images that fail to load and replace them
function imageError(image) {
  image.src = 'images/no-image.png';
}
```

10.0 Upload a few PDF Tax forms to s3://sudhakar-content-store

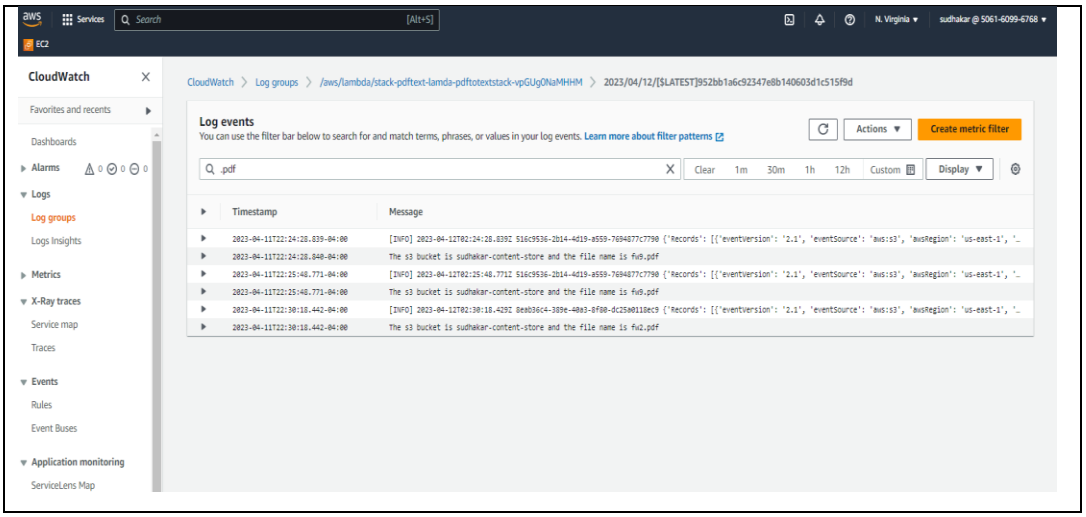
Upload W-2, W-9 , 1099msc, 1099q forms to S3://sudhakar-content-store using the following AWS CLI commands

aws s3 cp ./fw9.pdf s3://sudhakar-content-store
aws s3 cp ./fw2.pdf s3://sudhakar-content-store
aws s3 cp ./f1099q.pdf s3://sudhakar-content-store
aws s3 cp ./f1099msc.pdf s3://sudhakar-content-store



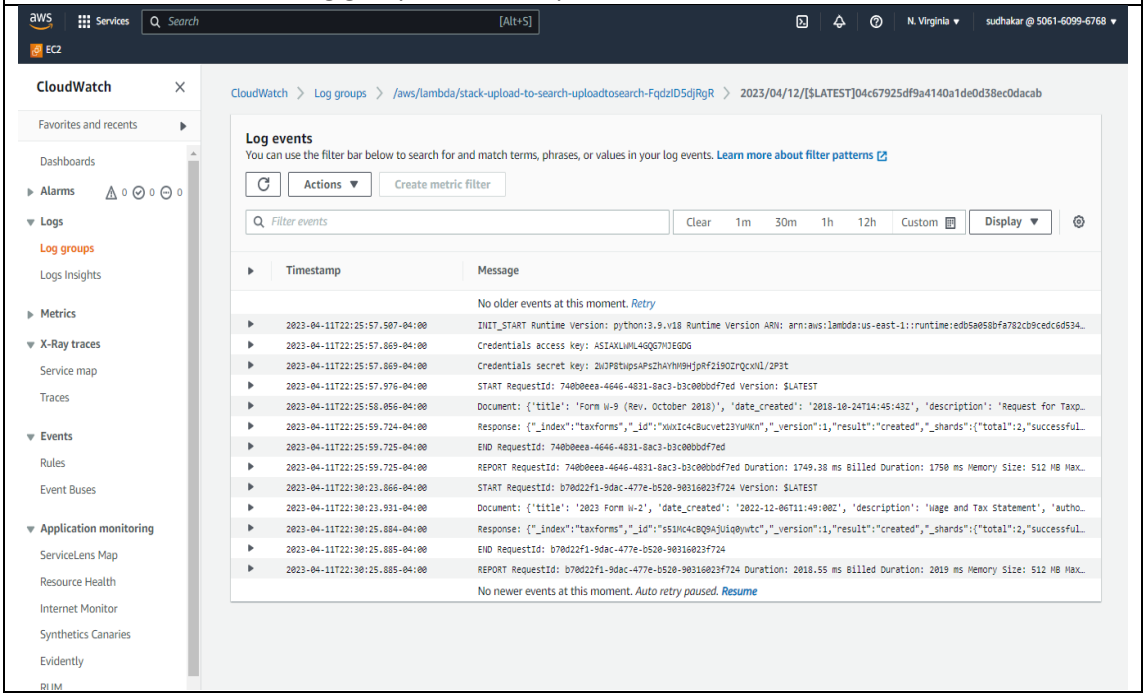
10.1 Check if the PDF lambda processed the fw9 and fw2 forms from the CloudWatch logs.

Locate the CloudWatch logs for the stack-pdftext-lambda and search for *pdf files



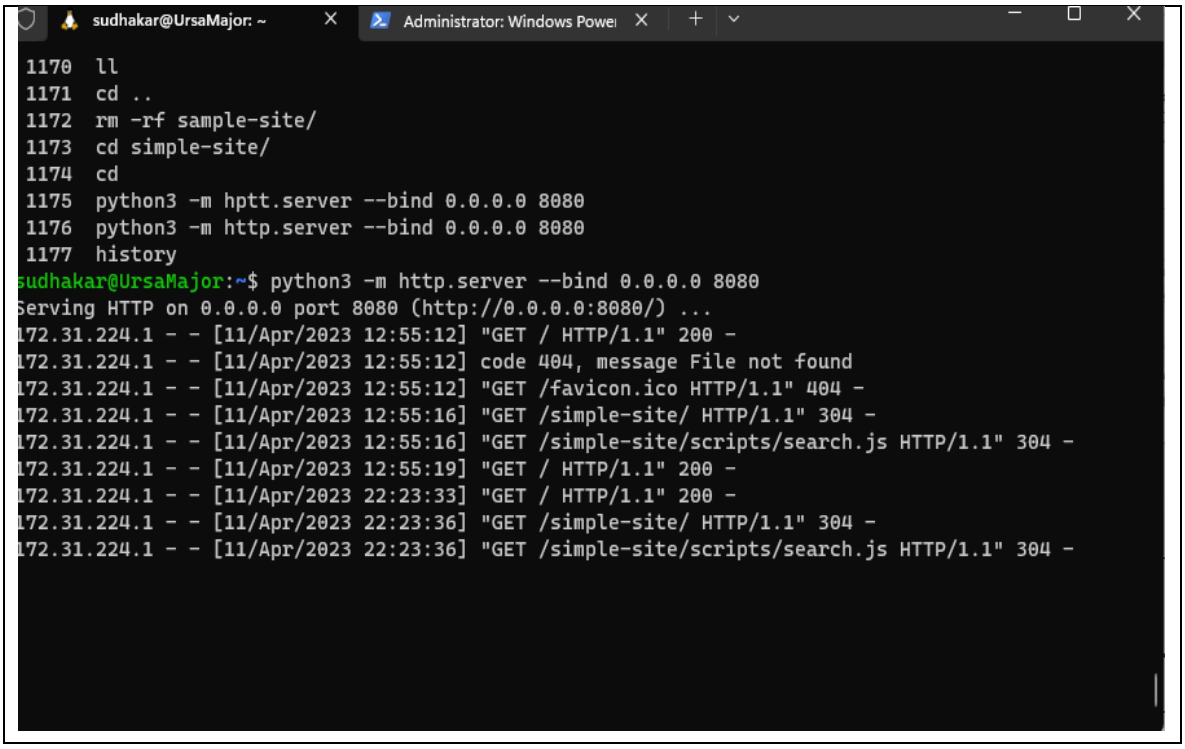
10.2 Check if the Upload lambda uploaded the JSON documents from the CloudWatch logs.

Locate the CloudWatch log groups for stack-upload-to-search and search for w9 and w2

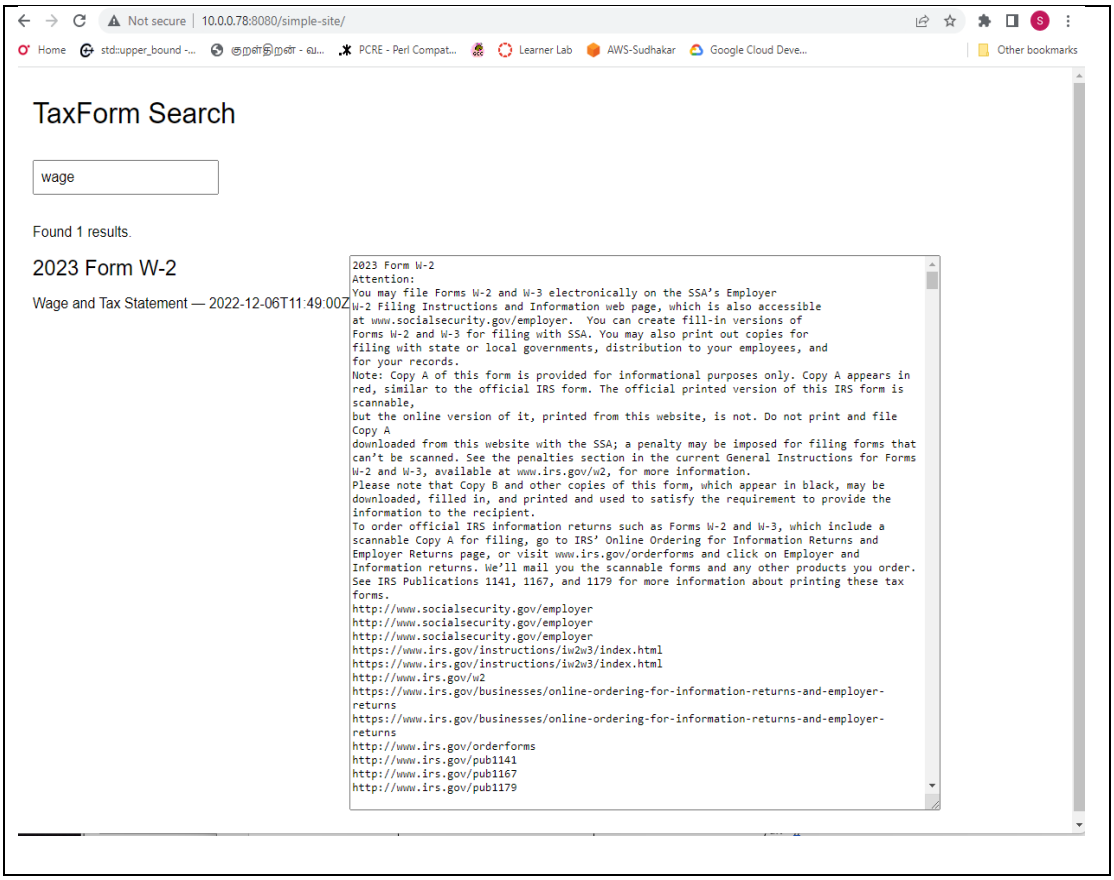


11. Use the Simple-site webpage to search for the tax forms

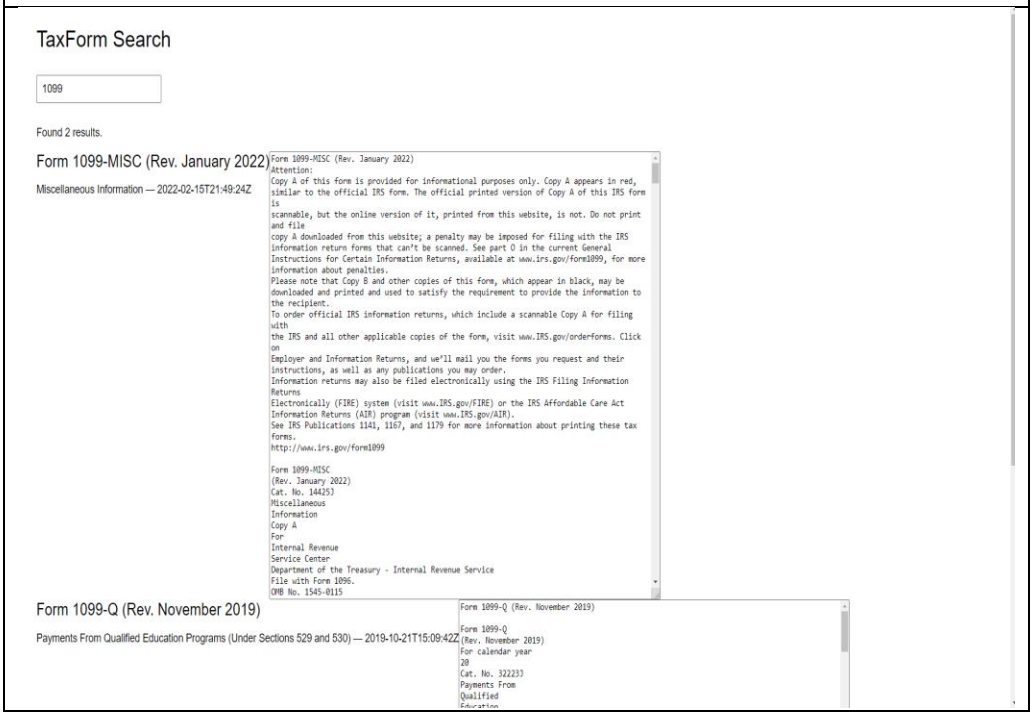
11.1 Run a simple python web server locally and open the simple-site web page.



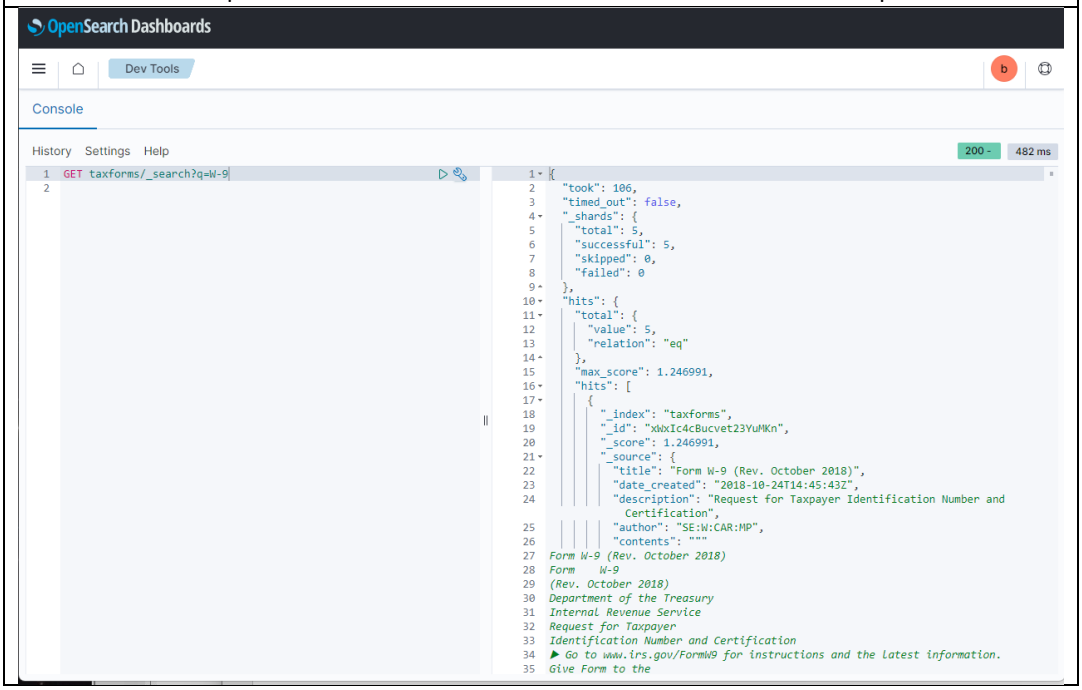
11.2 Search for ‘wage’ this will return W-2 Tax form



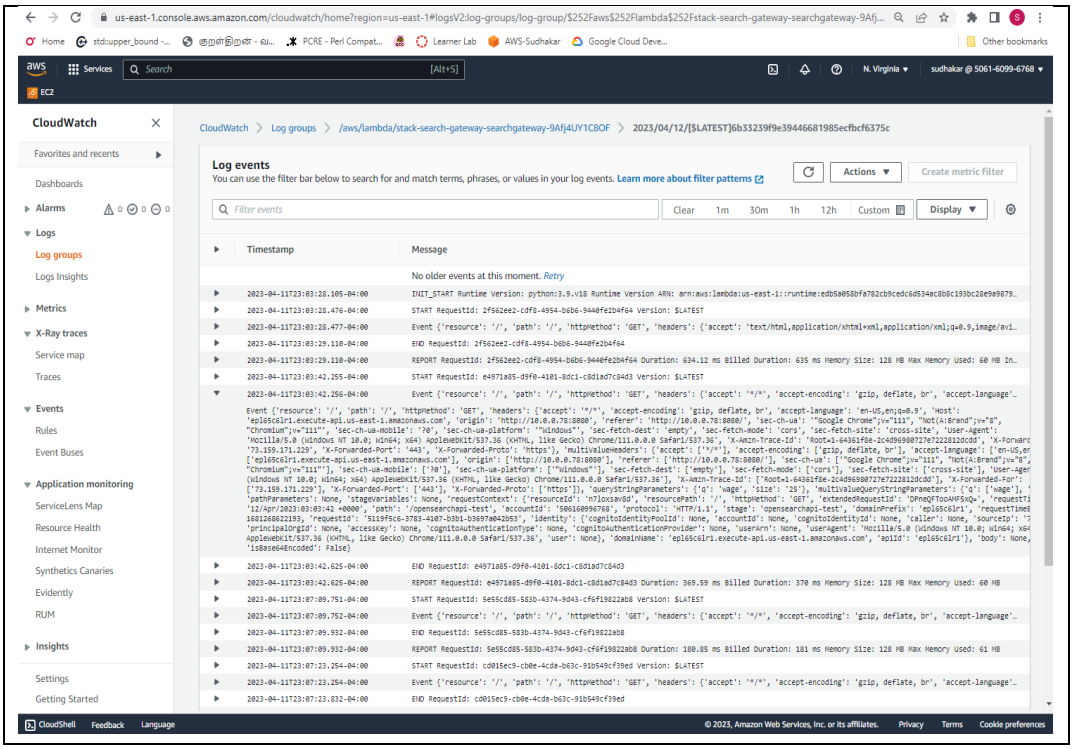
11.3 After uploading f1099q and f1099msc forms, search for 1099. This returns 2 forms



11.4 Also use the OpenSearch Dashboards to check if the tax forms have been uploaded.



11.5 Check the stack-search-gateway Lambda CloudWatch logs



12.0 Cleanup the Resources

Run the following commands to cleanup the resources

Delete the OpenSearch Domain	**aws opensearch delete-domain --domain-name bluehills**
Delete the stacks deployed by Cloud formation	**aws cloudformation delete-stack –stack-name stack-pdftext-lamda** **aws cloudformation delete-stack –stack-name stack-search-lambda** **aws cloudformation delete-stack –stack-name stack-upload-lamda**
Delete the S3 buckets	**aws s3 rb –force s3://sudhakar-content-store** **aws s3 rb –force s3://pdftotxt-inter** **aws s3 rb –force s3://search-domain-project-artifacts**