# PGPCC | Project

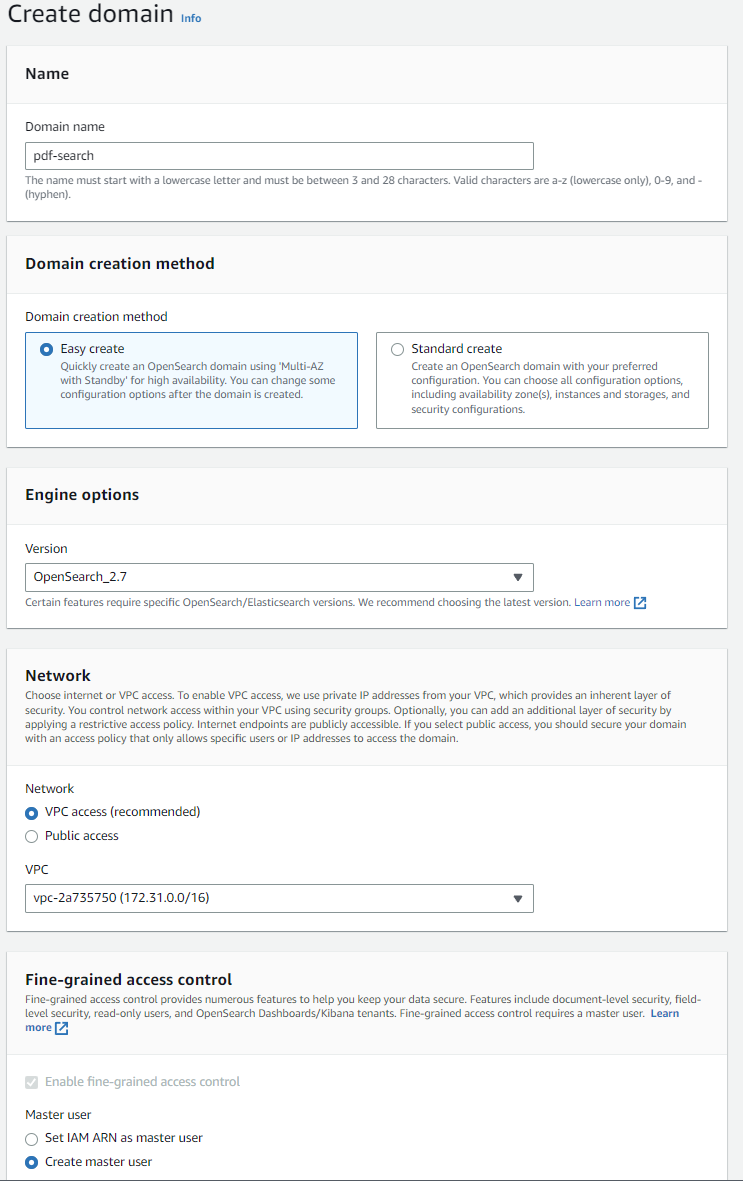
## Deploying a search engine using AWS Managed Services

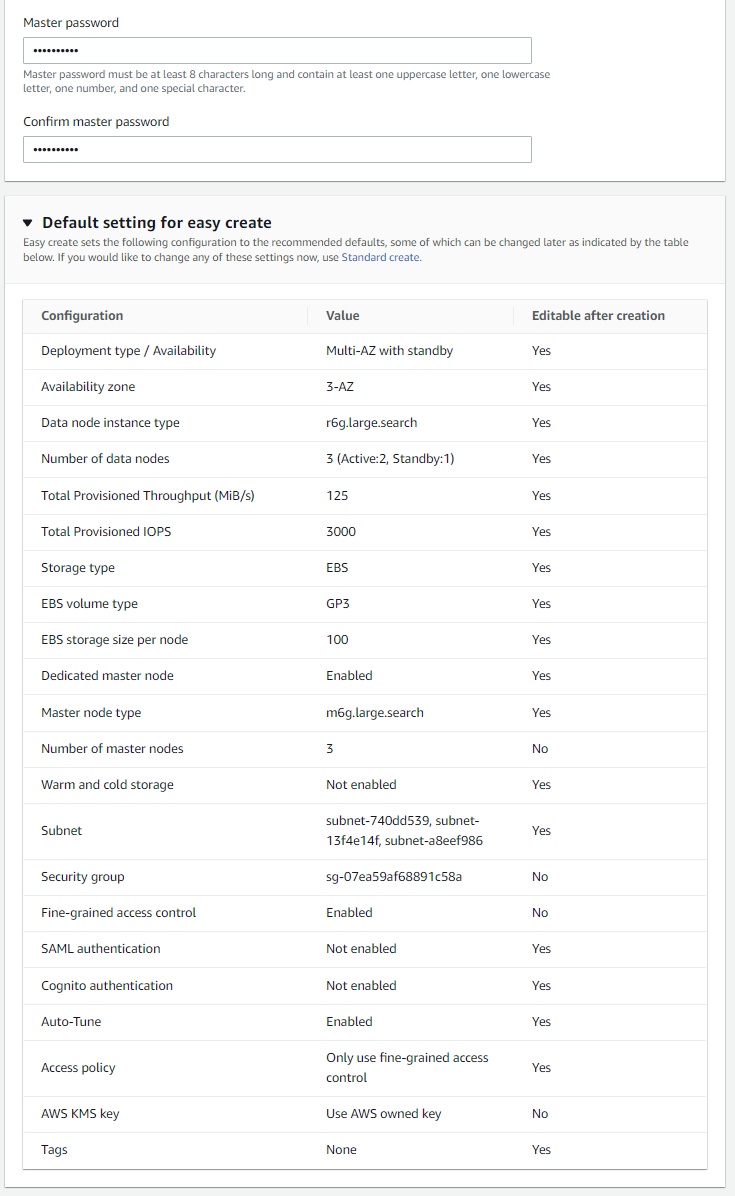
Steps for implementation:

1. Create an AWS Opensearch domain

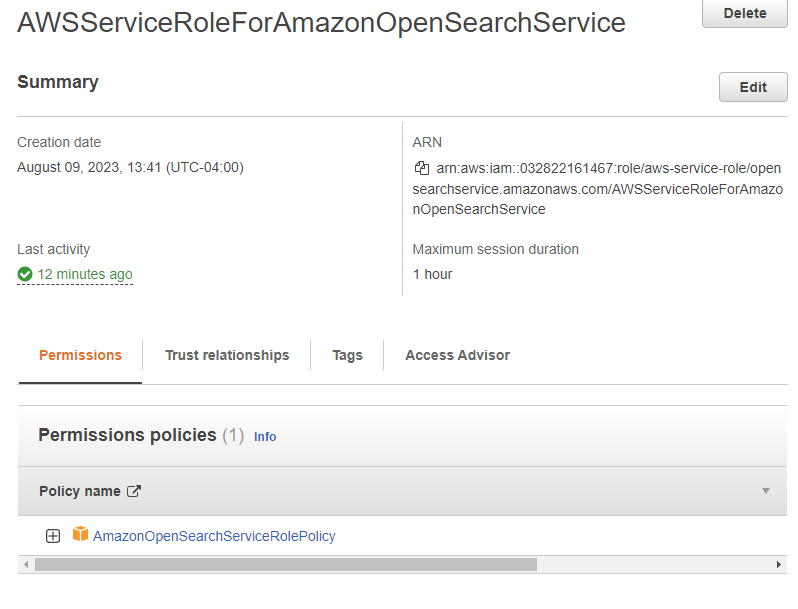
Screen shot of creating the domain. The instructions said to use “VPC access” but that led to a lot of problems for a lot of students. So we got the okay to make it public. But I don’t want to take short cuts. I want to do as much as possible to do it the way we would do it on the job. Public ElasticSearch/Opensearch isn’t how we would do it. So I stuck with “VPC Access”

I did everything in the default VPC.



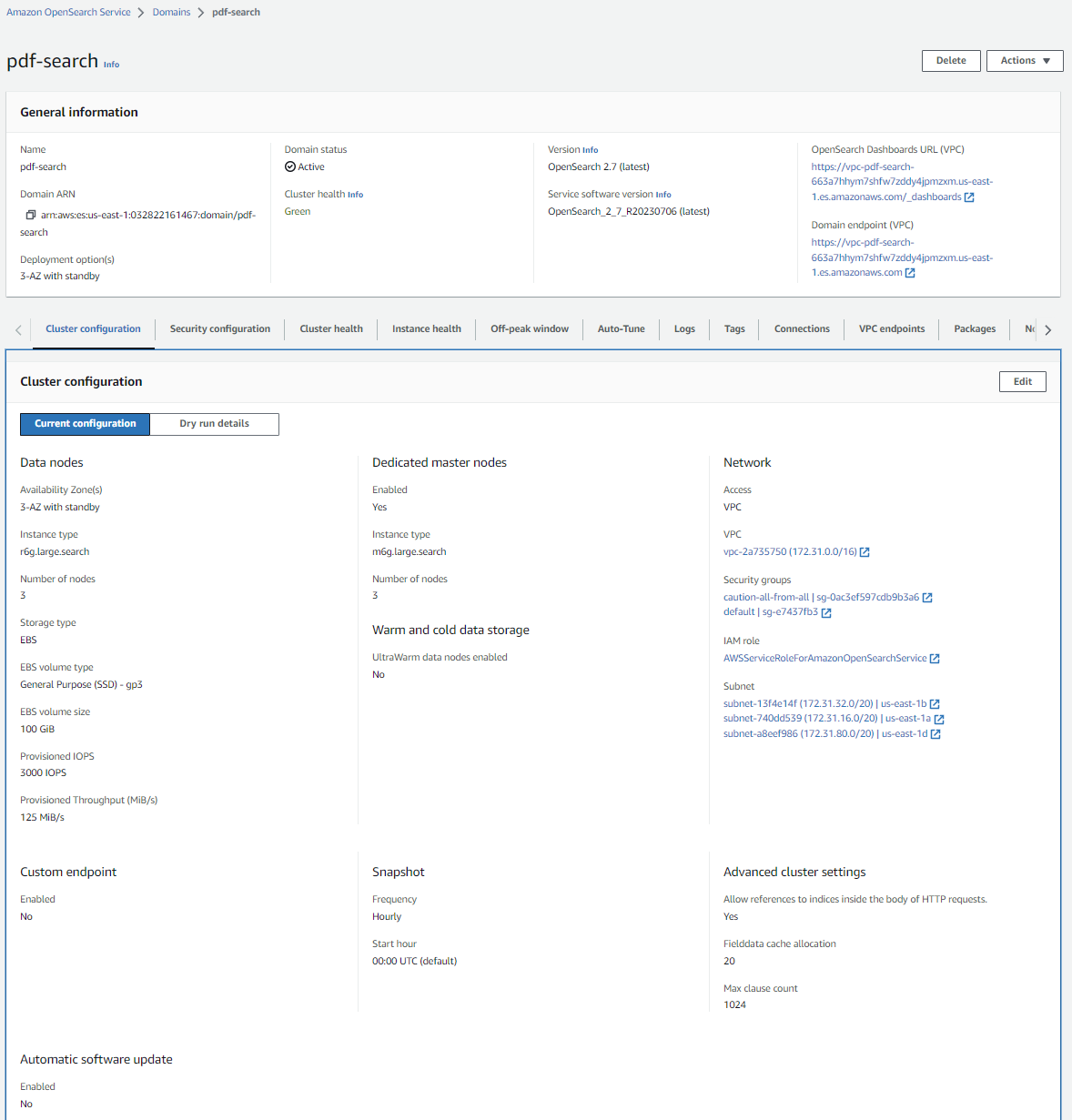


This IAM role was created by the previous step



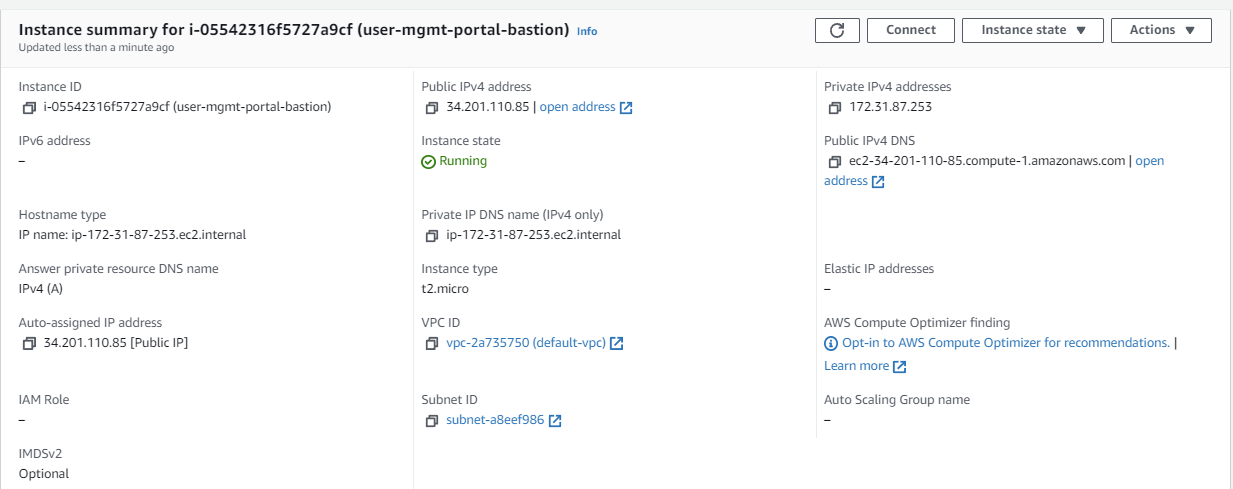
The Create Domain steps seemed to randomly pick the subnets to use and the security groups. It did not use the VPC default security group as one might expect. It grabbed my tomcat security group. But I changed it to the default VPC security group. I was struggling getting this to work in a VPC so for testing purposes I also included a security group that opened up all ports for everyone. Obviously before going into production this would have to be removed.

I took note of what subnets were used to make sure all lambda functions were on these subnets.

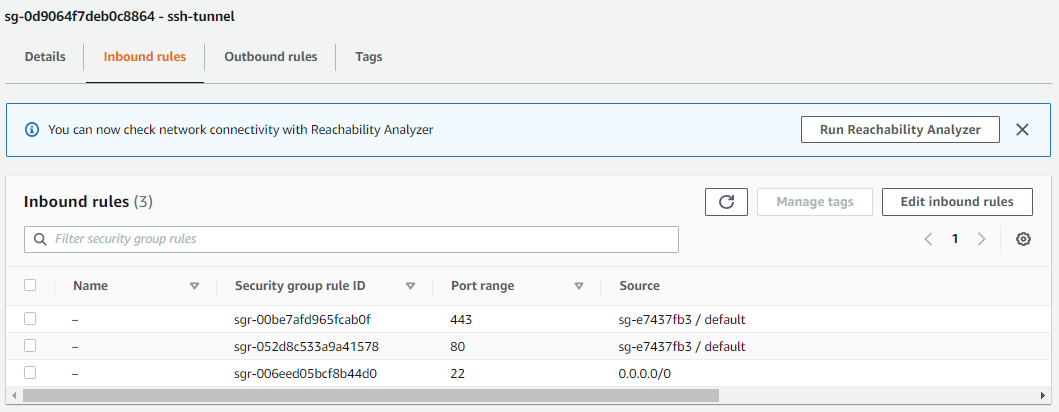


The next step is to add a master user.

The necessary configurations were not in place so the open search domain was not accessible from outside the VPC. I setup a SSH tunnel using an existing bastion host I have on my default VPC.



I created a “ssh-tunnel” security group and assigned it to the instance.



Then I established the SSH tunnel.

> ssh -i .\pemfiles\glkey-us-east-1.pem ubuntu@34.201.110.85 -N -L 9200:vpc-pdf-search-663a7hhym7shfw7zddy4jpmzxm.us-east-1.es.amazonaws.com:443

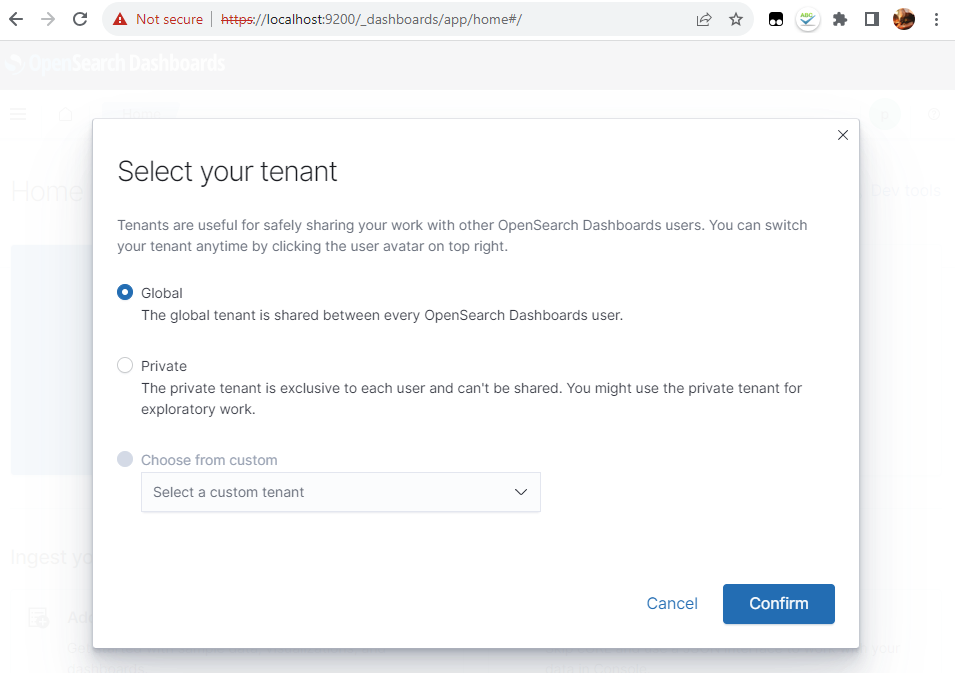
The authenticity of host '34.201.110.85 (34.201.110.85)' can't be established.

ECDSA key fingerprint is SHA256:bta/5dEolG5Wl5biaMahMLU5l7R9bKJYruWdbtfoDs8.

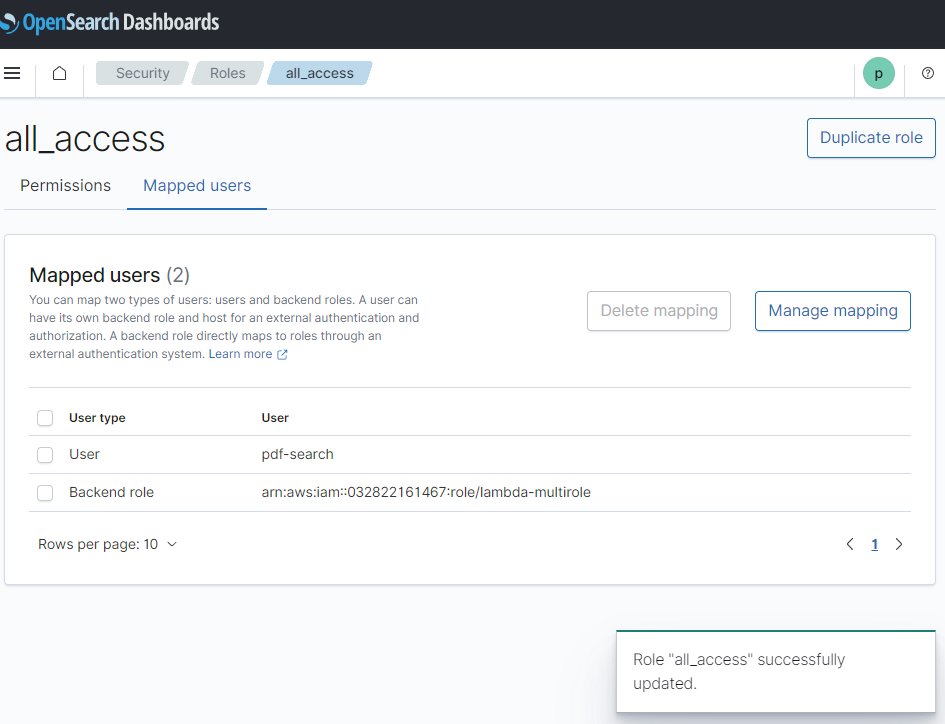
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes

Warning: Permanently added '34.201.110.85' (ECDSA) to the list of known hosts.

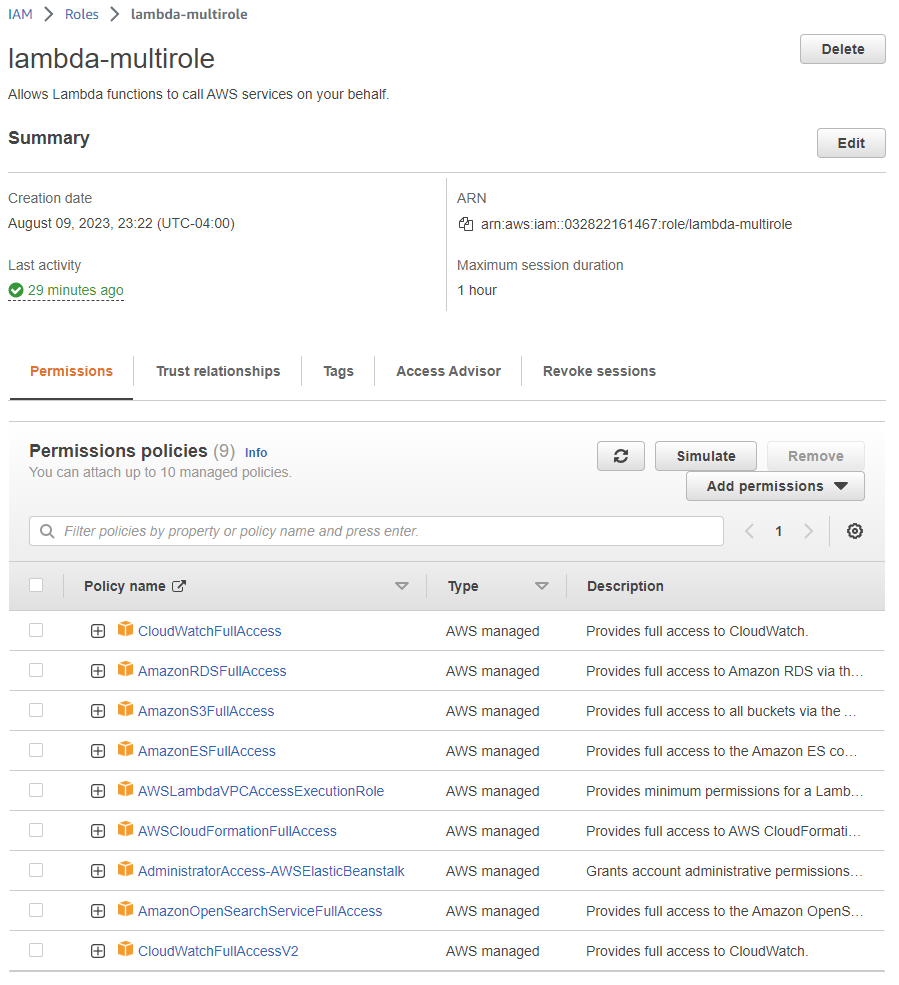
I was then able to go to <https://localhost:9200> to access the Opensearch dashboard



I created the backend role using my lambda-multirole IAM Role. In production I would create a role specify for this task and lock it down with minimal policies needed.

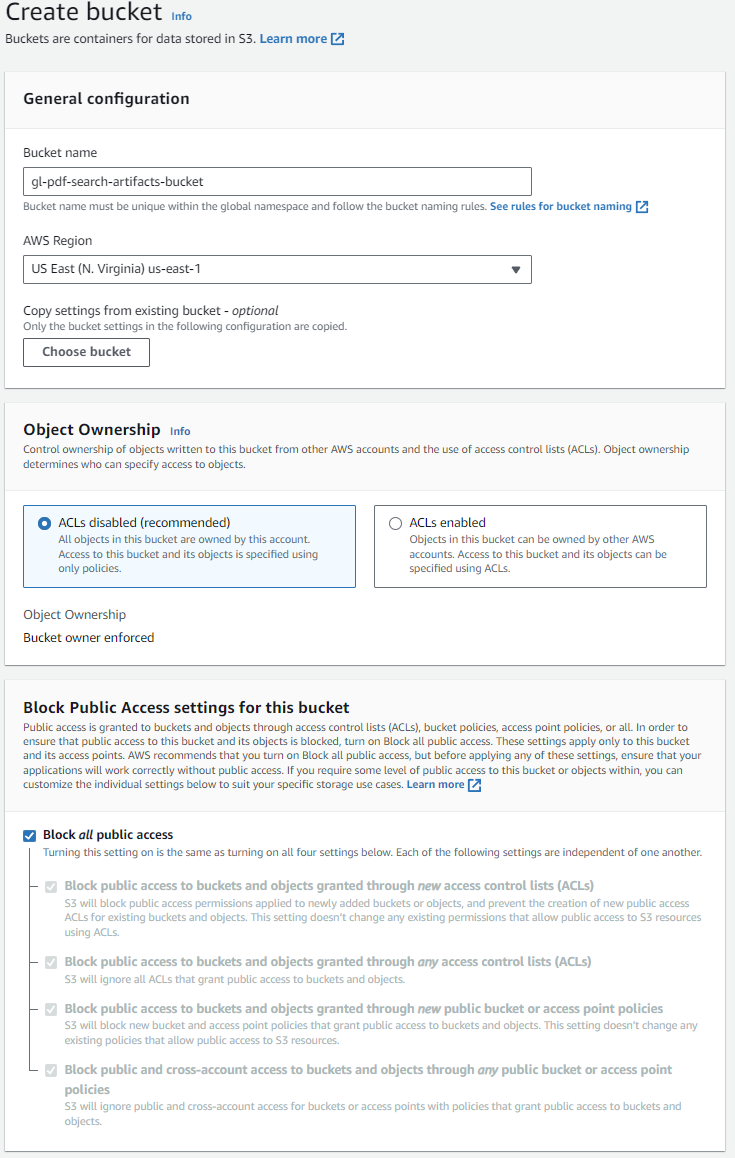


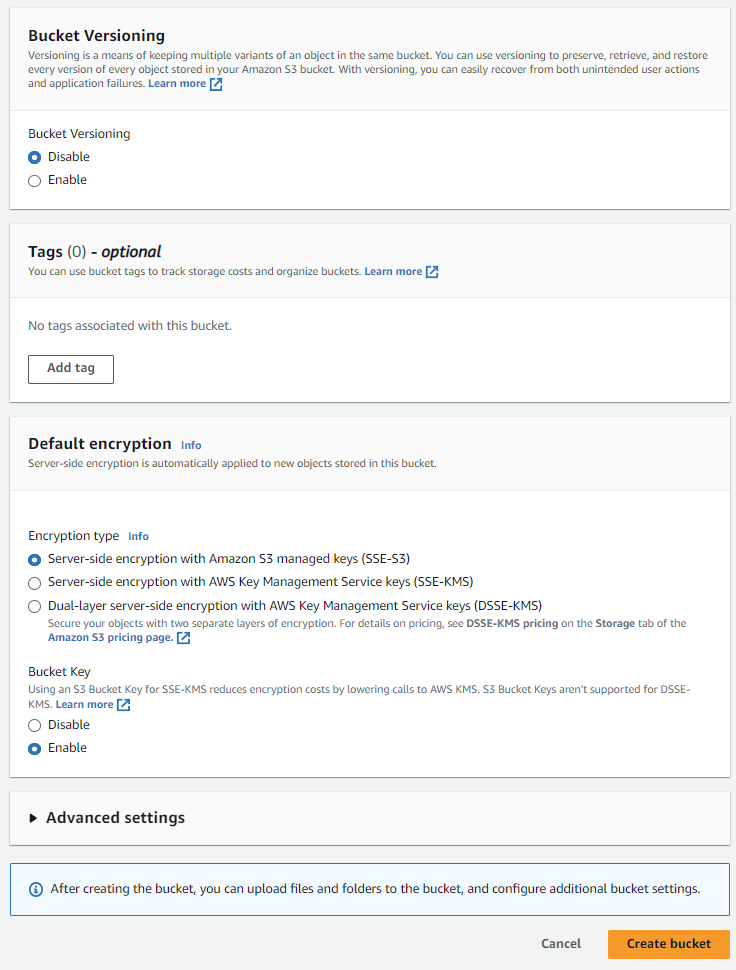
I used lambda-multirole for all lambda IAM Roles. So it has a lot of policies assigned to it. Elastic Beanstalk is assigned for a side project I have running. If this was production I would have dedicated roles with more restrictive policies



Steps B, C and D) Download deployment package files, create pipelines to deploy and setup event triggers

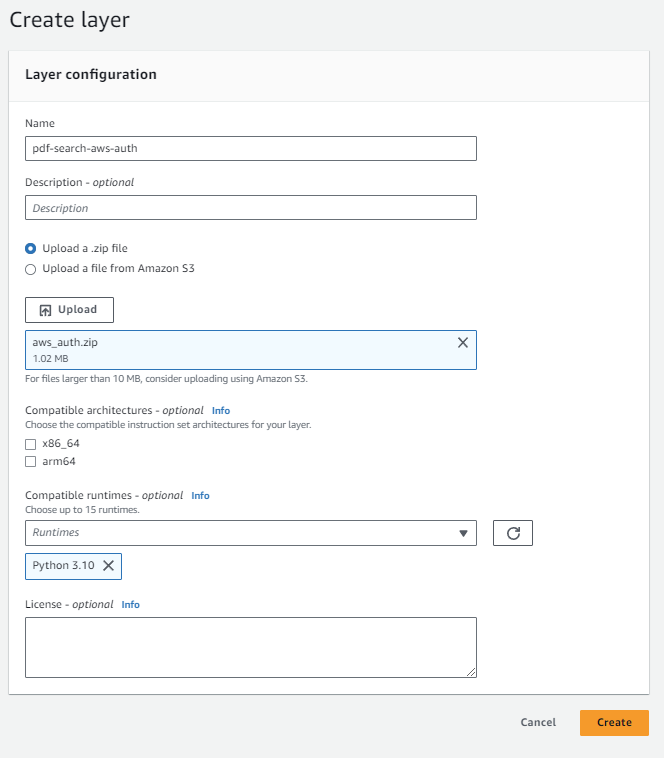
The original architecture for this project for each of the lambda functions would call `aws cloudformation package` in the buildspec.yml file for the CodeBuild projects in the CodePipeline. This command packages up all the artifacts needed for cloud formation and stores them in S3 and produces a new template file with the S3 artifact in the templates CodeUri for the CloudFormation Templates. To do this `aws cloudformation package` needs an S3 bucket to store the packages. I created an S3 bucket `gl-pdf-search-artifacts-bucket` for all the buildspec.yml files to use.

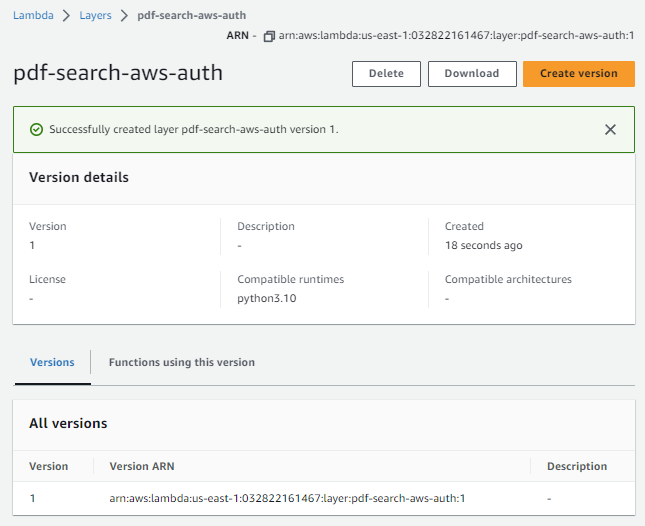




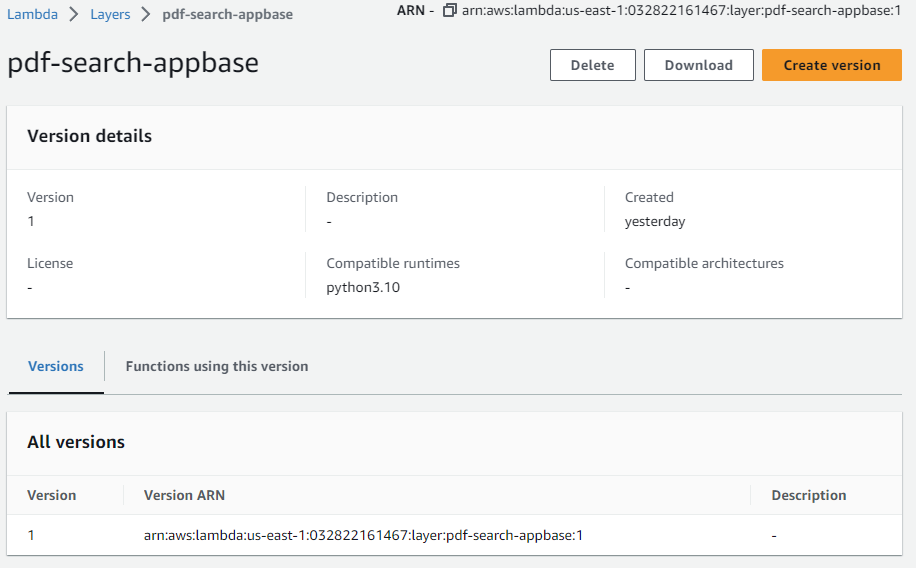
The next step was to create the pipelines. But changes needed to be made to the source files first. Before I could do that, I needed to create Lambda Layers to share dependencies.

pdf-search-aws-auth was the dependencies provided with the project files. I create the following lambda layer with it.





The provided project source uses the pypdf module. The dependencies were not included in the source files for the project. But I created a lambda layer for it in a previous project. So I added it as another lambda layer.



Then I was able to update the source code and create my pipelines.

PDFtoTXT source files: This lambda function is triggered when PDFs are uploaded to a S3 bucket to extract text from the PDF and store the text in another S3 bucket.

buildspec.yml: This file will be used by the CodeBuild project in the CodePipeline to create a Lambda function to export text from PDFs uploaded to an S3 bucket.

version: 0.2

phases:

  install:

    runtime-versions:

      python: 3.10

  build:

    commands:

      - aws cloudformation package --template-file pdftotxt.yaml --s3-bucket gl-pdf-search-artifacts-bucket --output-template-file lambdaoutput.yaml

artifacts:

  type: zip

  files:

    - lambdaoutput.yaml

pdftotxt.yaml: My intention was to create a CloudFormation template that would do the following

* Create the PdfToText lambda function
* Create the S3 bucket for documents to be uploaded to
* Create the S3 bucket intermediary storage
* Setup permissions to enable the S3 bucket for the document uploads to invoke the PdfToText lambda function
* Configure a trigger for the S3 bucket to invoke the function whenever a PDF is uploaded.

I wasted too much time on this. I eventually learned the last item on that list could not be done in the cloudformation template without the stack being created at least once without it due to circular dependency problems. I found three solutions to this problem.

1. Create the CloudFormation template without the even trigger and manually add it.
2. Create the CloudFormation stack without it and then update the template to include the event. With this approach the event trigger is added after the stack has deployed the Lambda function and S3 bucket, thus removing the circular dependency.
3. Create a CloudFormation script that would deploy the Lambda function and S3 bucket and then invoke another lambda function after they are created to create the event.

#3 felt like a really bad hack as the example had the code in the template. #1 is prone to error, because it you delete the stack without manually removing the trigger, CloudFormation gets stuck. So I went with option #2. This does present a maintenance problem as it is not intuitive. If someone deletes the stack and attempts to recreate it they will have to know they have to perform this manual step first.

If this was going into production I might look at something like an AWS Event Bridge. I don’t know if it would make sense to go with something like that, but it’s worth looking into a more intuitive solution.

This is the only CloudFormation template that includes both Lambda layers.

AWSTemplateFormatVersion: 2010-09-09

Transform: 'AWS::Serverless-2016-10-31'

Description: >-

  Defining the PDF To Text Lambda function and the S3 buckets for the PDF

  documents and the text files

Resources:

  glPdfSearchPdfToTxtLambda:

    Type: 'AWS::Serverless::Function'

    Properties:

      FunctionName: gl-pdf-search-pdf-to-txt-lambda

      Handler: lambda\_function.lambda\_handler

      Runtime: python3.7

      Description: ''

      CodeUri: .

      MemorySize: 512

      Timeout: 900

      Role: 'arn:aws:iam::032822161467:role/lambda-multirole'

      Environment:

        Variables:

          TARGET\_BUCKET: gl-pdf-search-inter-store-bucket

      Layers:

        - 'arn:aws:lambda:us-east-1:032822161467:layer:pdf-search-aws-auth:1'

        - 'arn:aws:lambda:us-east-1:032822161467:layer:pdf-search-appbase:1'

      VpcConfig:

        SecurityGroupIds:

          - 'sg-0ac3ef597cdb9b3a6'

          - 'sg-e7437fb3'

        SubnetIds:

          - 'subnet-a8eef986'

          - 'subnet-740dd539'

          - 'subnet-13f4e14f'

  glPdfSearchDocumentStoreBucket:

    Type: 'AWS::S3::Bucket'

    Properties:

      BucketName: gl-pdf-search-document-store-bucket

      # The following NotificationConfiguration is used to trigger the

      # lambda function. But it cannot be used when the stack is created

      # for the first time. After the stack is created, the following

      # NotificationConfiguration can be added and the stack can be

      # updated. This will trigger the Lambda function when a new PDF

      # document is uploaded to the bucket

      # NotificationConfiguration:

      #   LambdaConfigurations:

      #     - Event: 's3:ObjectCreated:\*'

      #       Function: !GetAtt glPdfSearchPdfToTxtLambda.Arn

  glPdfSearchDocumentStoreEventPdfToText:

    Type: 'AWS::Lambda::Permission'

    Properties:

      FunctionName: !Ref glPdfSearchPdfToTxtLambda

      Action: 'lambda:invokeFunction'

      Principal: s3.amazonaws.com

      SourceArn: !GetAtt glPdfSearchDocumentStoreBucket.Arn

buildspec.yml: This file is used by CodeBuild in the CodePipeline to package up the CloudFormation script that is deployed in the CodeDeploy step in the pipeline.

version: 0.2

phases:

  install:

    runtime-versions:

      python: 3.10

  build:

    commands:

      - aws cloudformation package --template-file pdftotxt.yaml --s3-bucket gl-pdf-search-artifacts-bucket --output-template-file lambdaoutput.yaml

artifacts:

  type: zip

  files:

    - lambdaoutput.yaml

recreate\_repository.sh: I used this script to build my code repository.

#!/bin/bash

repository\_name="gl-pdf-search-pdf-to-text-repository"

repository\_description="GL Managed Services Project - PDFtoText"

# Clean up by removing the local repository

rm -rf .git

# Check if the repository already exists and delete it

aws codecommit get-repository --repository-name "$repository\_name" > /dev/null 2>&1

if [ $? -eq 0 ]; then

    aws codecommit delete-repository --repository-name "$repository\_name"

fi

# Create a new repository

clone\_url\_http=$(aws codecommit create-repository --repository-name "$repository\_name" --repository-description "$repository\_description" --output text --query 'repositoryMetadata.cloneUrlHttp')

# Initialize a local Git repository

git init

git add buildspec.yml lambda\_function.py pdftotxt.yaml

git commit -m "initial commit"

# Add the remote repository and push the code

git remote add origin "$clone\_url\_http"

git push -u origin master

Output:

$ ./recreate\_repository.sh

{

"repositoryId": "5f16bd60-2df0-4e0c-a8c5-ca3d9282750f"

}

Initialized empty Git repository in C:/Users/demay/OneDrive/Desktop/courses/gl-cloud/projects/managed services/PGPCC Project - Deploying a search engine using AWS Managed Services/PDFtoTXT/.git/

warning: in the working copy of 'buildspec.yml', LF will be replaced by CRLF the next time Git touches it

warning: in the working copy of 'lambda\_function.py', LF will be replaced by CRLF the next time Git touches it

warning: in the working copy of 'pdftotxt.yaml', LF will be replaced by CRLF the next time Git touches it

[master (root-commit) 6b79e96] initial commit

3 files changed, 120 insertions(+)

create mode 100644 buildspec.yml

create mode 100644 lambda\_function.py

create mode 100644 pdftotxt.yaml

Enumerating objects: 5, done.

Counting objects: 100% (5/5), done.

Delta compression using up to 8 threads

Compressing objects: 100% (5/5), done.

Writing objects: 100% (5/5), 1.86 KiB | 238.00 KiB/s, done.

Total 5 (delta 0), reused 0 (delta 0), pack-reused 0

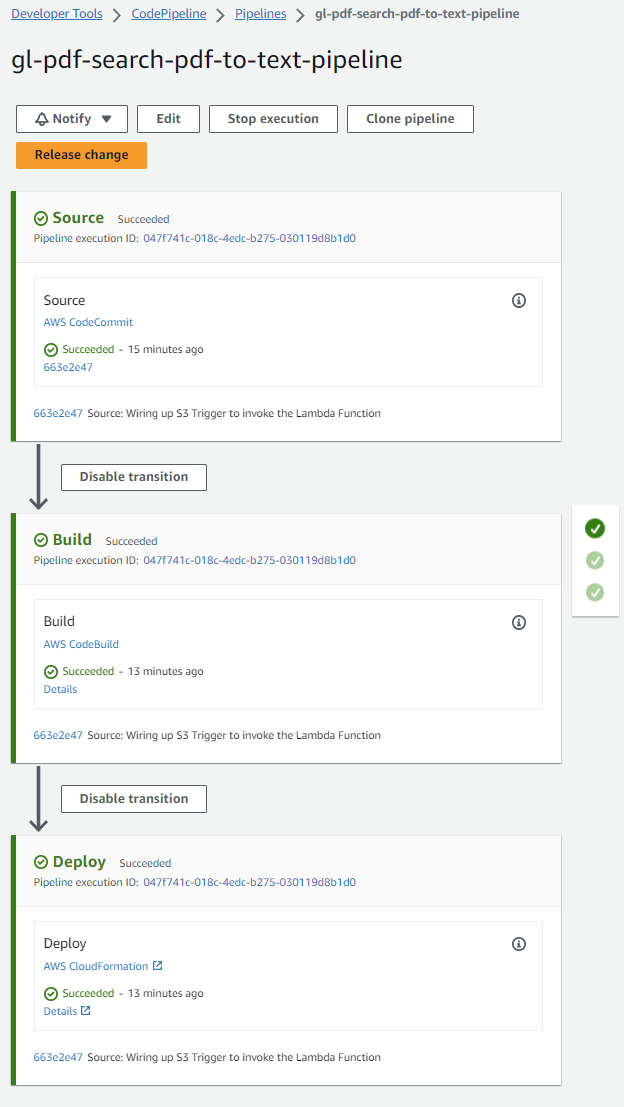
remote: Validating objects: 100%

To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/gl-pdf-search-pdf-to-text-repository

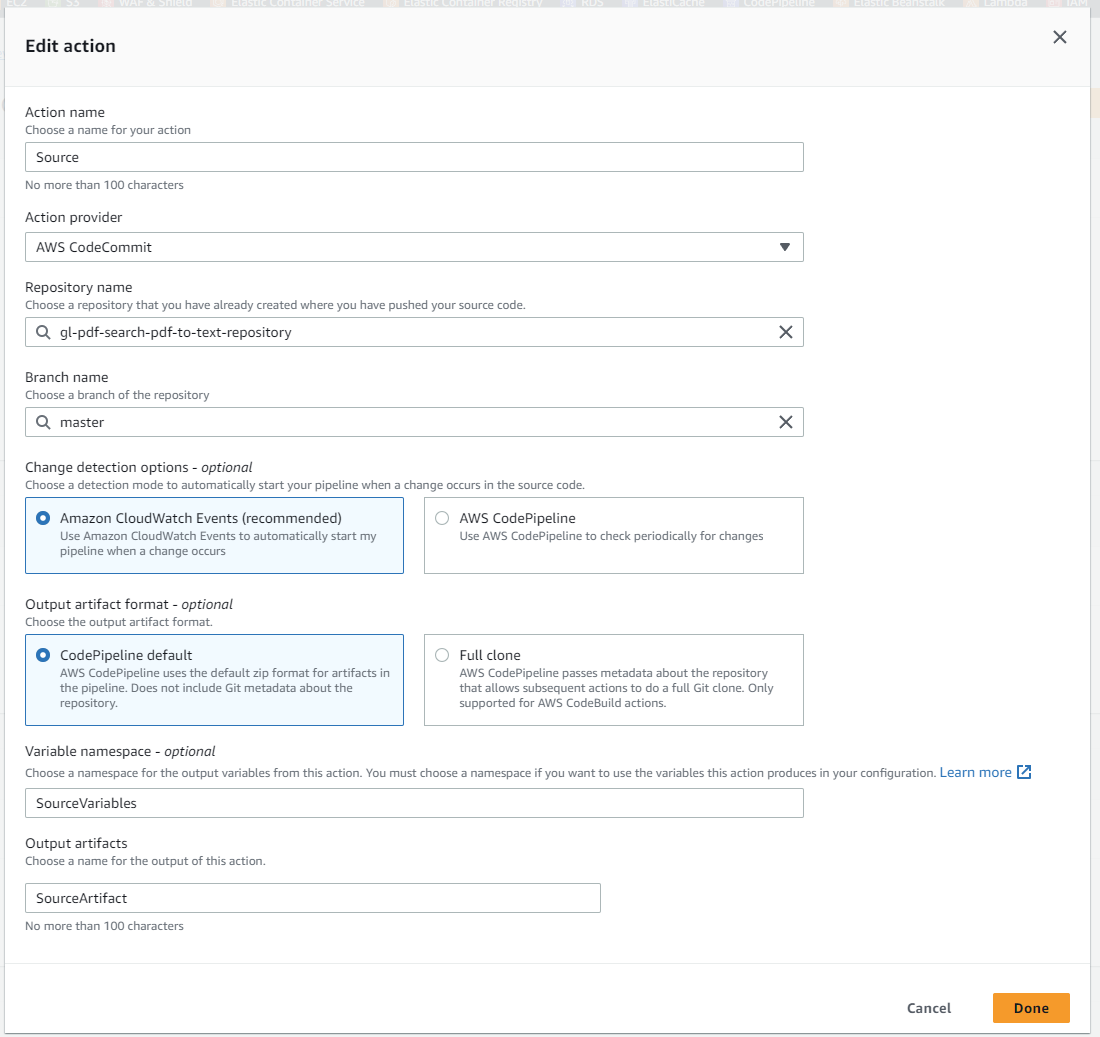
\* [new branch] master -> master

branch 'master' set up to track 'origin/master'.

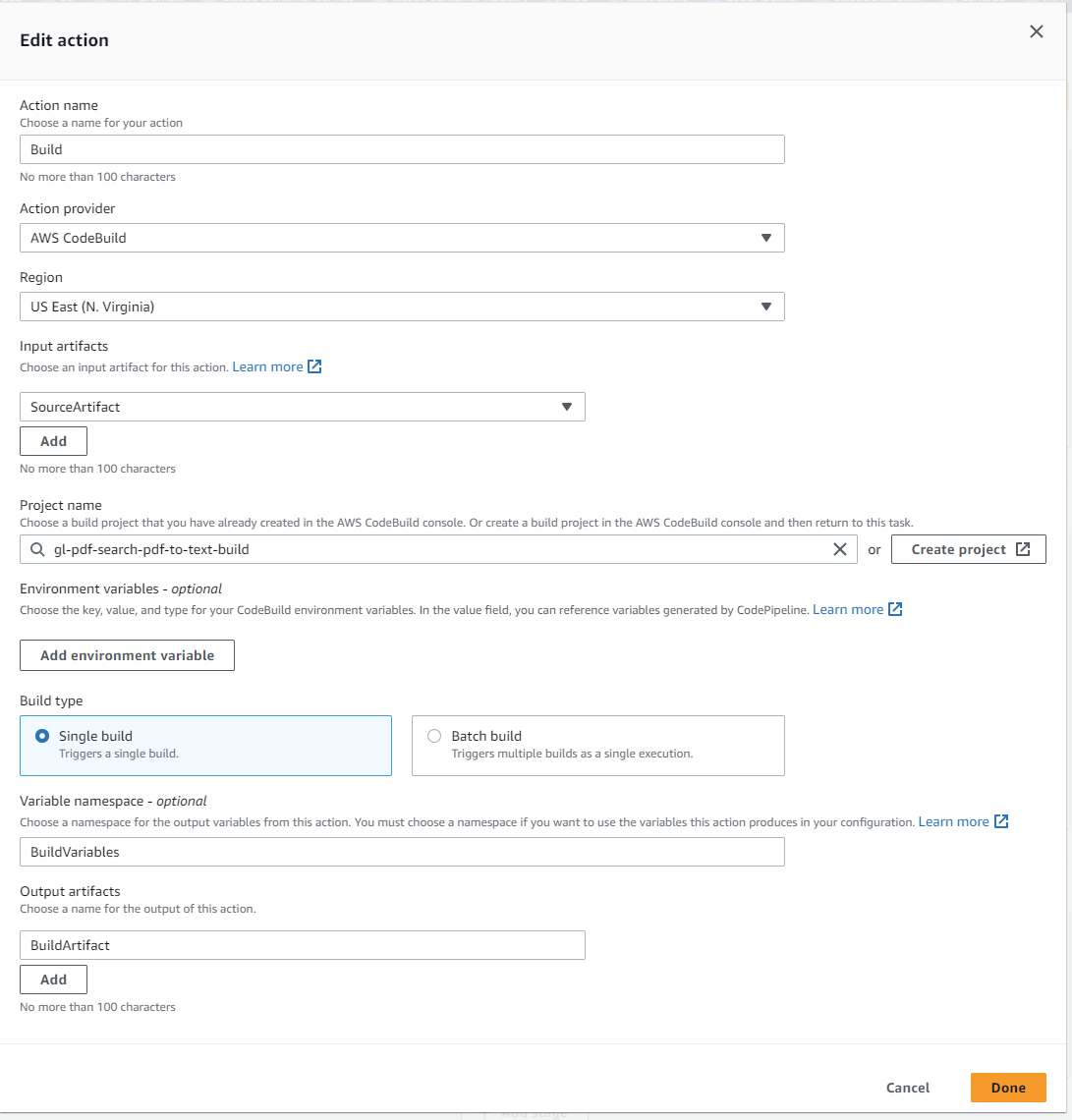
Next step. Setup the code pipeline for PDFtoText



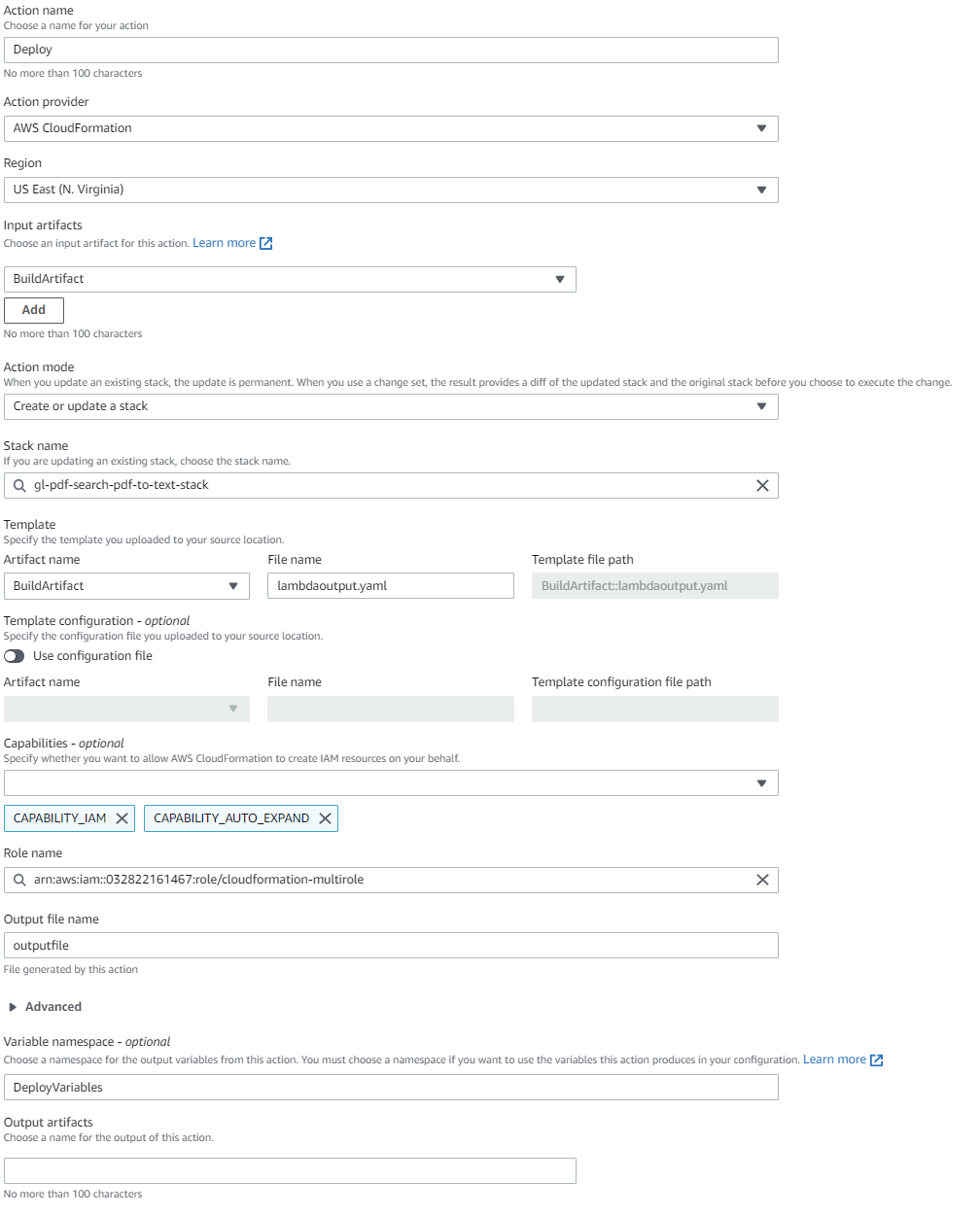
Source step: This is triggered whenever a new commit is made to `gl-pdf-search-pdf-to-text-repository`



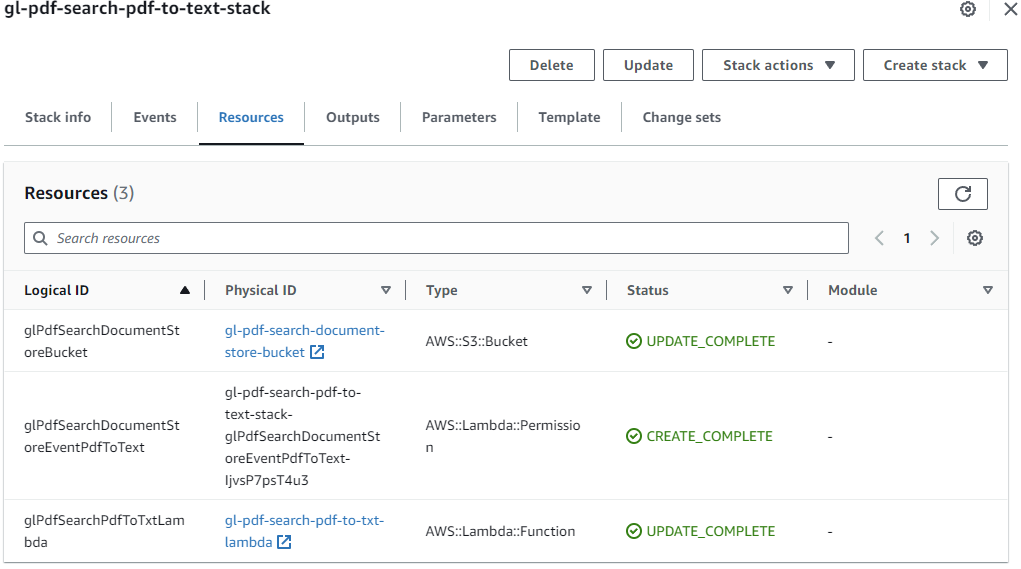
Build step: This calls `aws cloudformation package` to package everything up to deploy to CloudFormation.



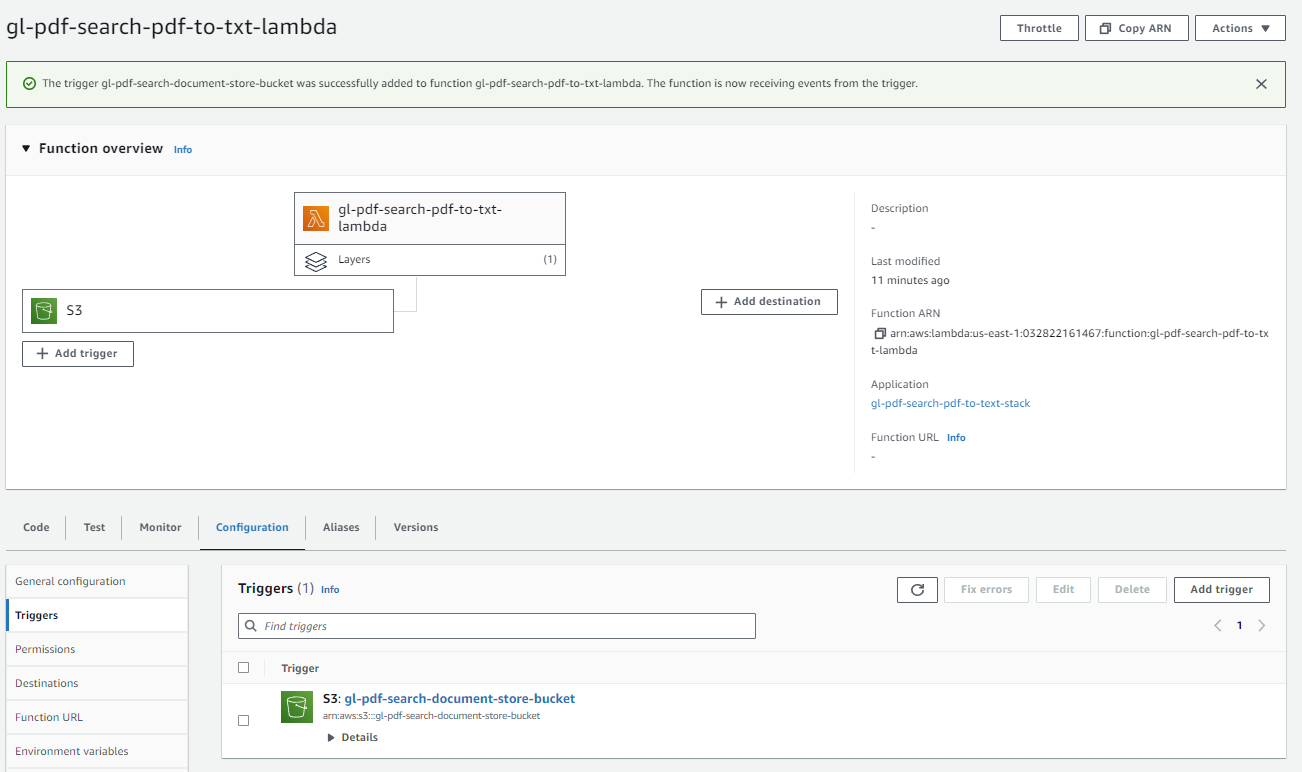
Deploy Stage: After everything is built, the package is pushed out to cloudformation.



These are the resources that are created the second time code is pushed through the pipeline (event has to be added after stack initial creation)



The lambda function is created by the CloudFormation stack.



UploadToSearch: The following lambda function is used to take the extracted PDF text from PDFToText lambda function and upload it to AWS OpenSearch.

Basically everything done for PDFToText was repeated for UploadToSearch with the following exceptions.

* The pypdf lambda layer is not needed
* A VPC Endpoint needed to be added so the lambda function can reach AWS Opensearch domain

buildspec.yml

version: 0.2

phases:

  install:

    runtime-versions:

      python: 3.10

  build:

    commands:

      - aws cloudformation package --template-file upload-to-search.yaml --s3-bucket gl-pdf-search-artifacts-bucket --output-template-file upload.yaml

artifacts:

  type: zip

  files:

    - upload.yaml

upload-to-search.yaml:

AWSTemplateFormatVersion: '2010-09-09'

Transform: 'AWS::Serverless-2016-10-31'

Description: An AWS Serverless Specification template describing your function.

Resources:

  glPdfSearchUploadToSearchLambda:

    Type: 'AWS::Serverless::Function'

    Properties:

      FunctionName: gl-pdf-search-upload-to-search-lambda

      Handler: lambda\_function.handler

      Runtime: python3.9

      CodeUri: .

      Description: ''

      MemorySize: 512

      Timeout: 900

      Role: 'arn:aws:iam::032822161467:role/lambda-multirole'

      Layers:

        - 'arn:aws:lambda:us-east-1:032822161467:layer:pdf-search-aws-auth:1'

      VpcConfig:

        SecurityGroupIds:

          - 'sg-0ac3ef597cdb9b3a6'

          - 'sg-e7437fb3'

        SubnetIds:

          - 'subnet-a8eef986'

          - 'subnet-740dd539'

          - 'subnet-13f4e14f'

  glPdfSearchInterStoreBucket:

    Type: 'AWS::S3::Bucket'

    Properties:

      BucketName: gl-pdf-search-inter-store-bucket

      # The following NotificationConfiguration is used to trigger the

      # lambda function. But it cannot be used when the stack is created

      # for the first time. After the stack is created, the following

      # NotificationConfiguration can be added and the stack can be

      # updated. This will trigger the Lambda function when a new PDF

      # document is uploaded to the bucket

      NotificationConfiguration:

        LambdaConfigurations:

          - Event: 's3:ObjectCreated:\*'

            Function: !GetAtt glPdfSearchUploadToSearchLambda.Arn

  glPdfSearchUploadToSearchVpcEndpoint:

    Type: 'AWS::EC2::VPCEndpoint'

    Properties:

      ServiceName: com.amazonaws.us-east-1.s3

      VpcEndpointType: Gateway

      VpcId: "vpc-2a735750"

      RouteTableIds: ["rtb-ecf65592"]

      PolicyDocument:

        Version: "2008-10-17"

        Statement:

          - Effect: "Allow"

            Principal: "\*"

            Action: "\*"

            Resource: "\*"

  glPdfSearchInterStoreEventUploadToSearch:

    Type: 'AWS::Lambda::Permission'

    Properties:

      FunctionName: !Ref glPdfSearchUploadToSearchLambda

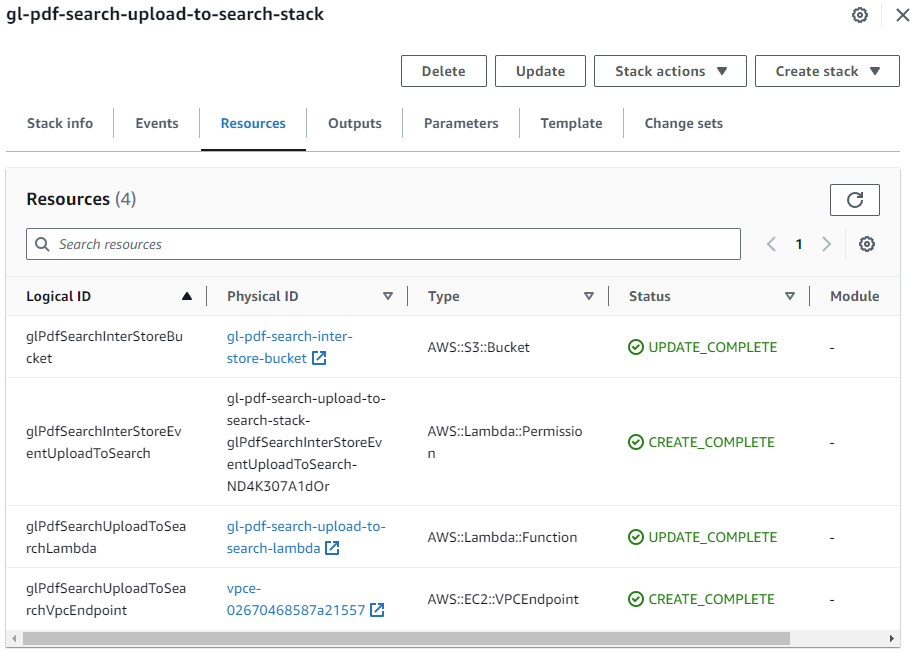
      Action: 'lambda:invokeFunction'

      Principal: s3.amazonaws.com

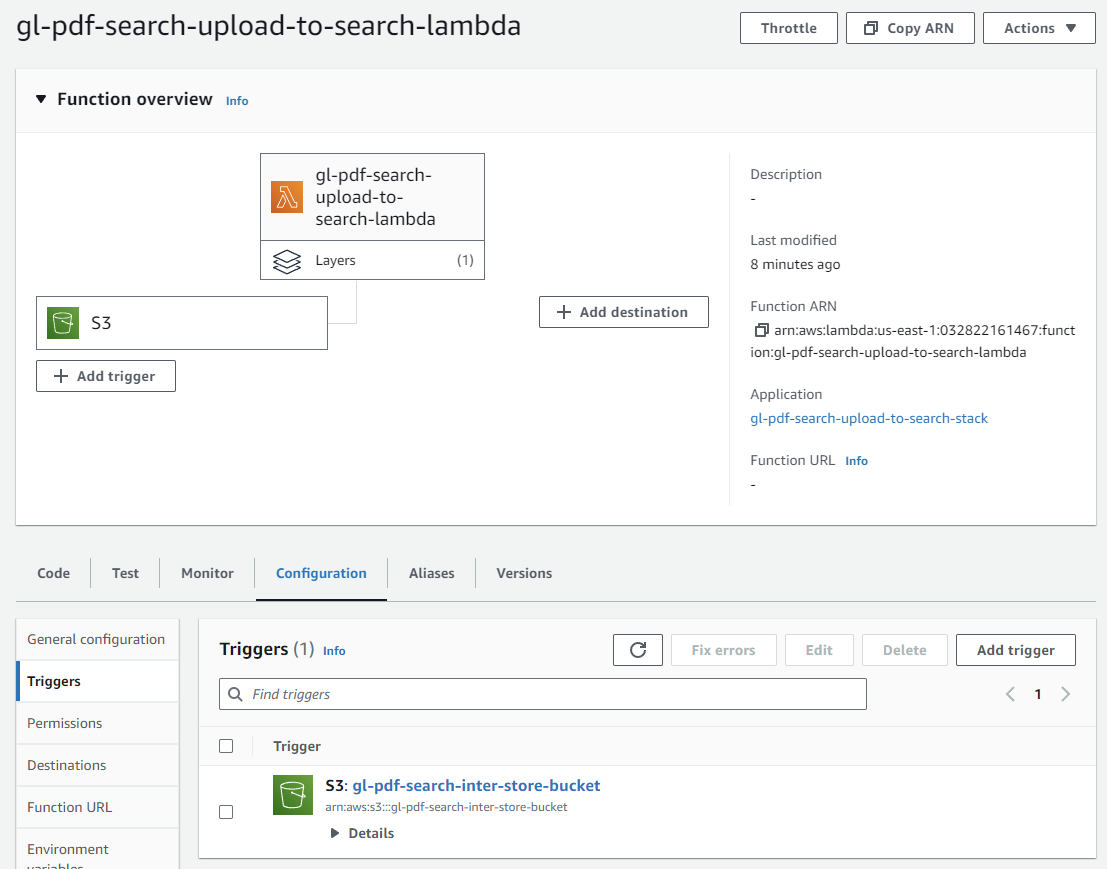
      SourceArn: arn:aws:s3:::gl-pdf-search-inter-store-bucket

The repository and code pipeline was created in the same manner as PDFToText. Pretty much identical except for the resource names. The screen shots have been omitted for brevity.

These are the resources created by the CloudFormation stack

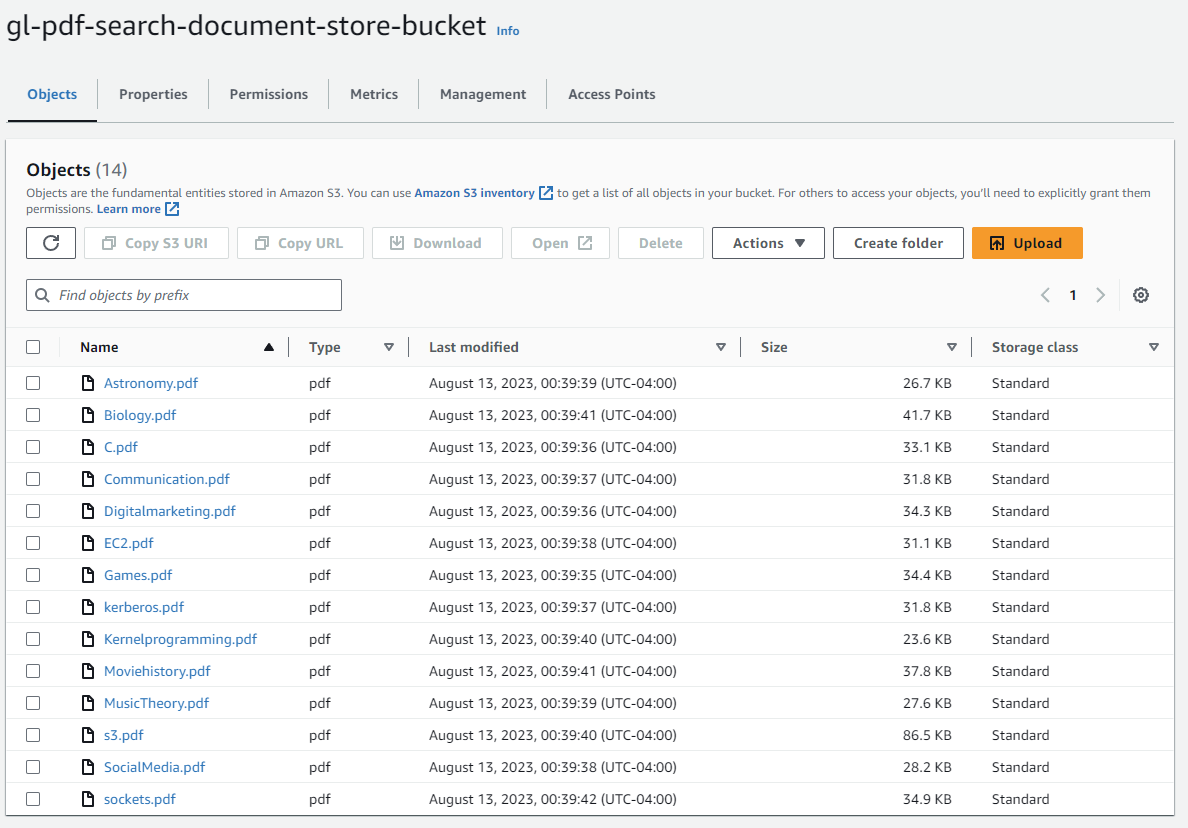


The lambda function created by the CloudFormation stack:

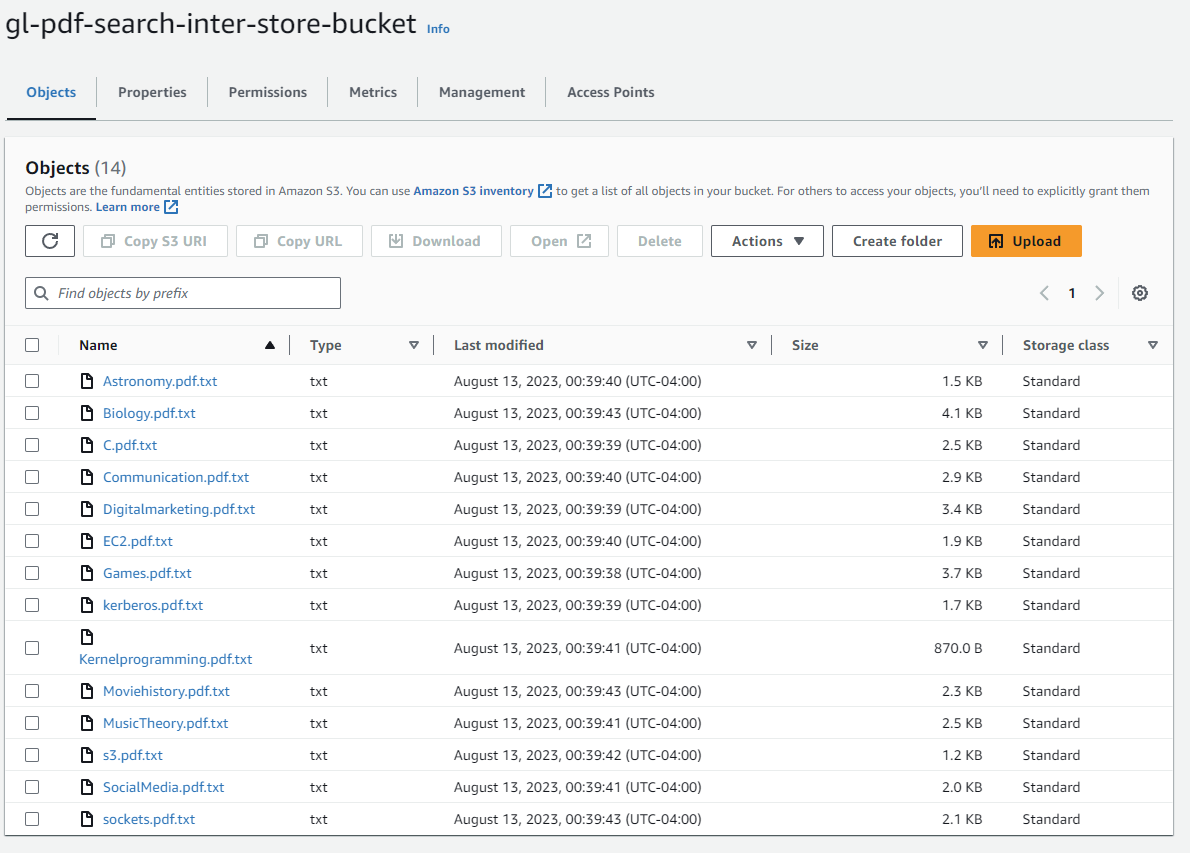


Executing the first two lambda functions:

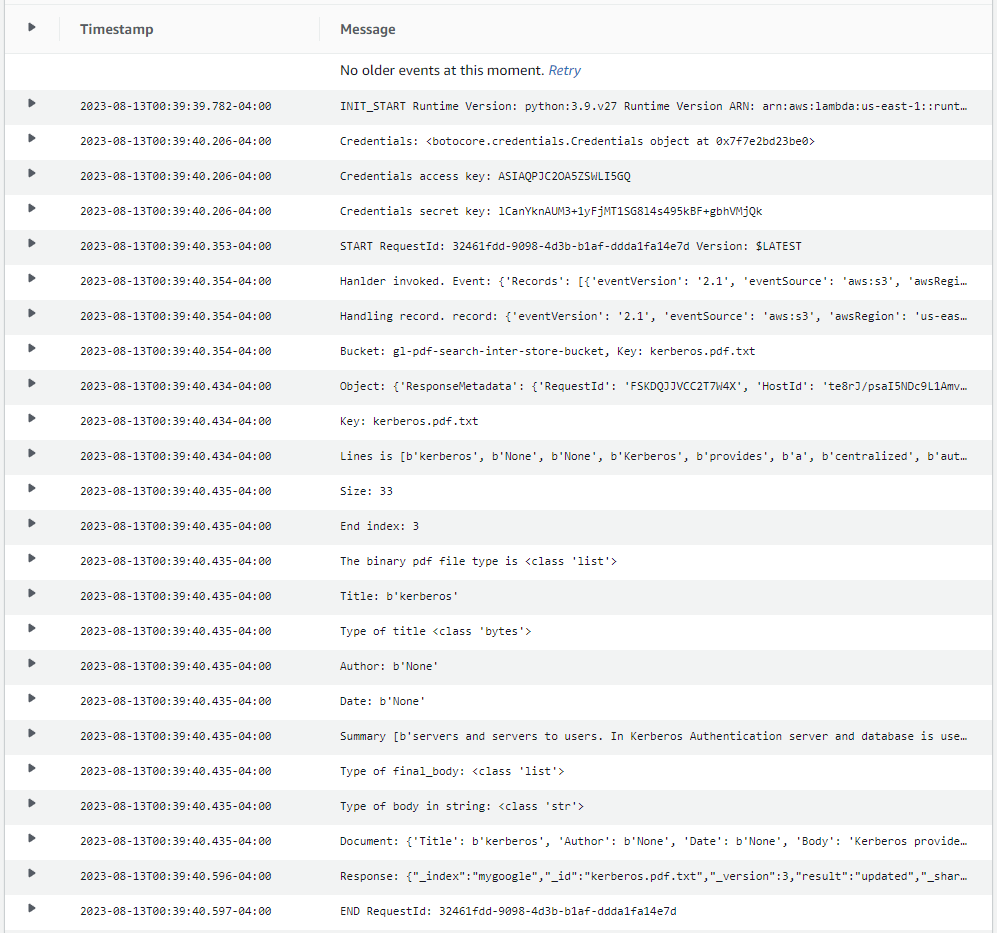
Uploading PDFs for the document store bucket.



Text extracted from the PDF files by the PDFtoText lambda function and uploaded to intermediary storage bucket



CloudWatch logs showing the output from the UploadToSearch lambda function.



Now that we have the text extracted from the PDFs and Uploaded to the Opensearch domain, the next step is to create the user experience.

SearchGateway: This lambda function serves the “Search Page” to the user.

buildspec.yml

version: 0.2

phases:

  install:

    runtime-versions:

      python: 3.10

  build:

    commands:

      - aws cloudformation package --template-file search-gateway.yaml --s3-bucket gl-pdf-search-artifacts-bucket --output-template-file search.yaml

artifacts:

  type: zip

  files:

    - search.yaml

search-gateway.yaml. The CloudFormation template that creates the lambda function. No S3 buckets or event triggers are needed for this. An event trigger from an API HTTP Gateway is added later.

AWSTemplateFormatVersion: '2010-09-09'

Transform: 'AWS::Serverless-2016-10-31'

Description: >-

  Process the incoming HTTP requests from the API Gateway to run search

  queries on the Opensearch domain

Resources:

  searchgateway:

    Type: 'AWS::Serverless::Function'

    Properties:

      FunctionName: gl-pdf-search-search-gateway-lambda

      Handler: lambda\_function.lambda\_handler

      Runtime: python3.9

      CodeUri: .

      Description: ''

      MemorySize: 128

      Timeout: 300

      Role: 'arn:aws:iam::032822161467:role/lambda-multirole'

      Layers:

        - 'arn:aws:lambda:us-east-1:032822161467:layer:pdf-search-aws-auth:1'

      VpcConfig:

        SecurityGroupIds:

          - 'sg-0ac3ef597cdb9b3a6'

          - 'sg-e7437fb3'

        SubnetIds:

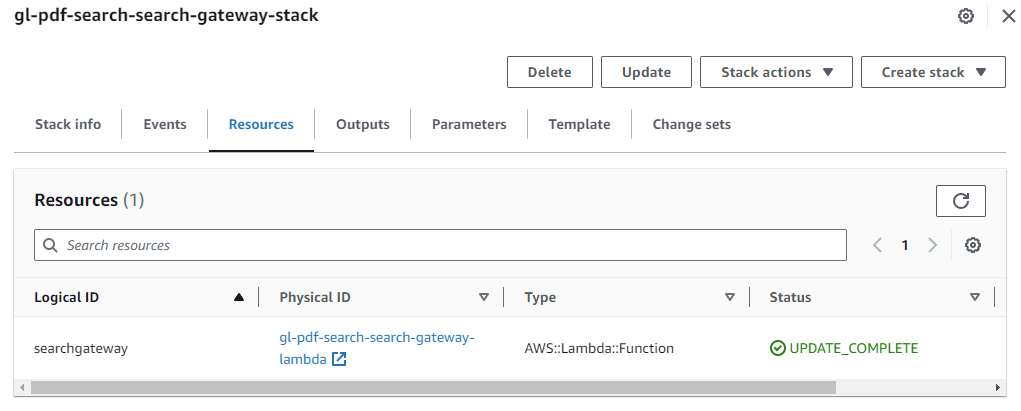
          - 'subnet-a8eef986'

          - 'subnet-740dd539'

          - 'subnet-13f4e14f'

The repository and code pipeline was created in the same manner as the previous two lambda functions. Everything is identical except for the resource names. The screen shots have been omitted for brevity.

The resources created in the CloudFormation stack



SearchFunction: This lambda function handles requests submitted from the “Search Page” implemented in the SearchGateway lambda function.

buildspec.yml

version: 0.2

phases:

  install:

    runtime-versions:

      python: 3.10

  build:

    commands:

      - aws cloudformation package --template-file search-function.yaml --s3-bucket gl-pdf-search-artifacts-bucket --output-template-file function.yaml

artifacts:

  type: zip

  files:

    - function.yaml

search-function.yaml: The CloudFormation template that creates the lambda function. No S3 buckets or event triggers are needed here for this. An event trigger from an API HTTP gateway is added later.

AWSTemplateFormatVersion: '2010-09-09'

Transform: 'AWS::Serverless-2016-10-31'

Description: An AWS Serverless Specification template describing your function.

Resources:

  searchgateway:

    Type: 'AWS::Serverless::Function'

    Properties:

      FunctionName: gl-pdf-search-search-function-lambda

      Handler: lambda\_function.lambda\_handler

      Runtime: python3.9

      CodeUri: .

      Description: ''

      MemorySize: 128

      Timeout: 300

      Role: 'arn:aws:iam::032822161467:role/lambda-multirole'

      Layers:

        - 'arn:aws:lambda:us-east-1:032822161467:layer:pdf-search-aws-auth:1'

      VpcConfig:

        SecurityGroupIds:

          - 'sg-0ac3ef597cdb9b3a6'

          - 'sg-e7437fb3'

        SubnetIds:

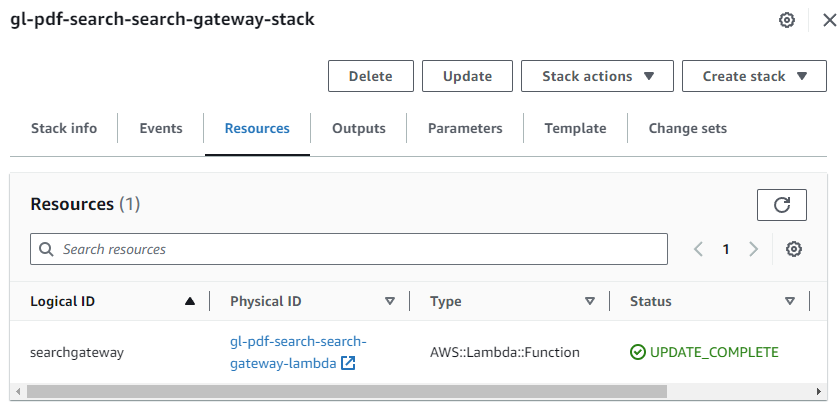
          - 'subnet-a8eef986'

          - 'subnet-740dd539'

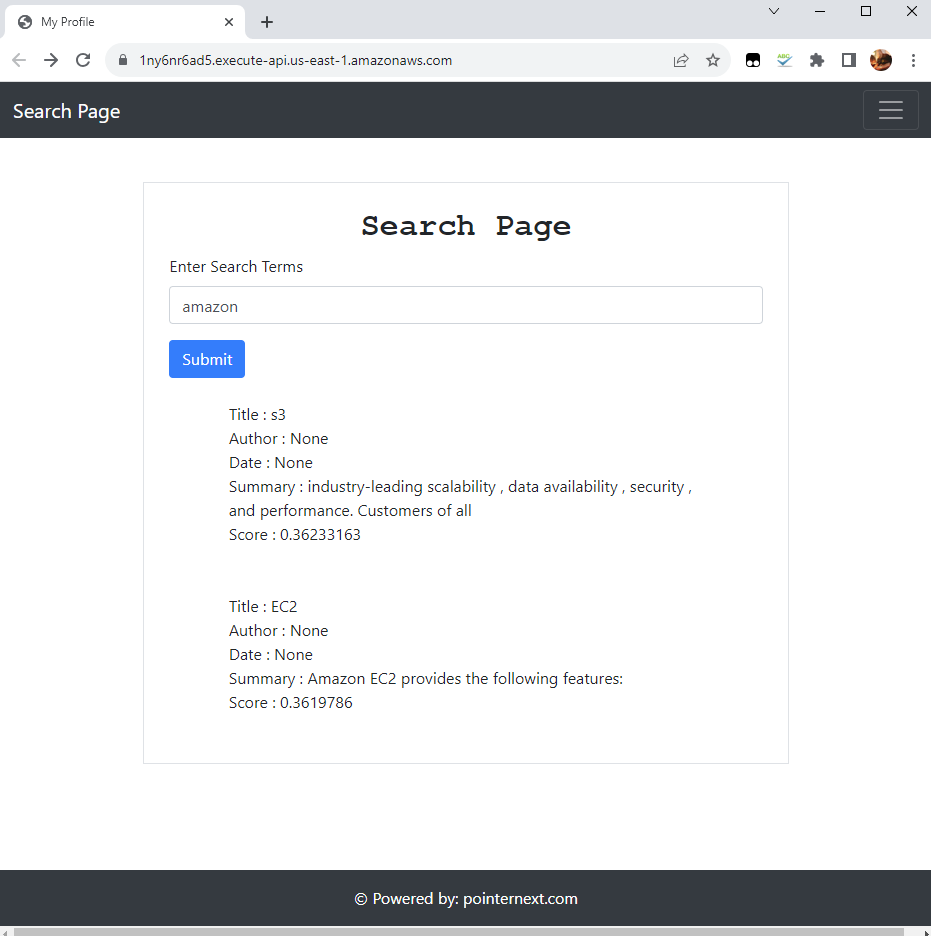
          - 'subnet-13f4e14f'

The repository and code pipeline was created in the same manner as the previous three lambda functions. Everything is identical except for the resource names. The screen shots have been omitted for brevity.

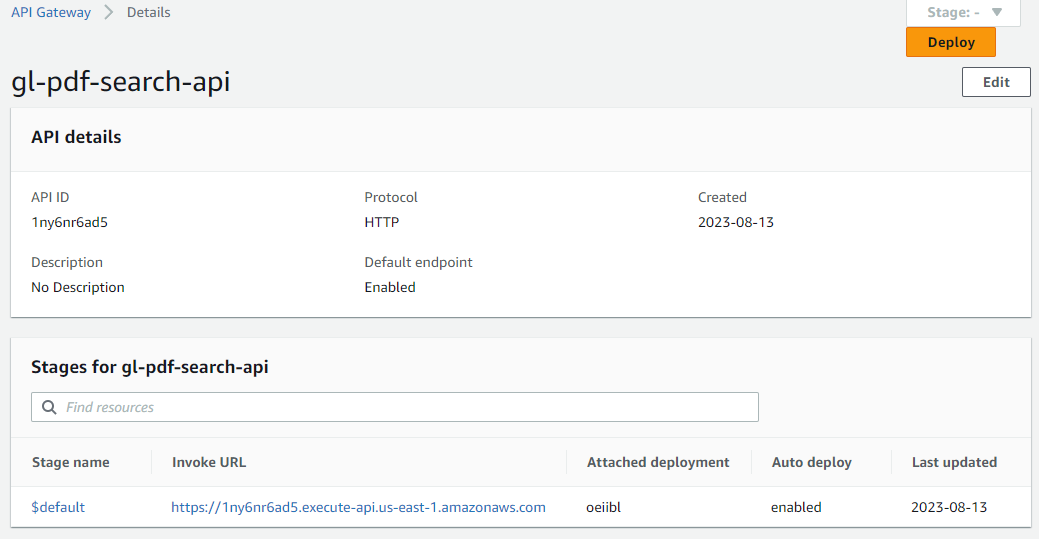
The resources created in the CloudFormation stack



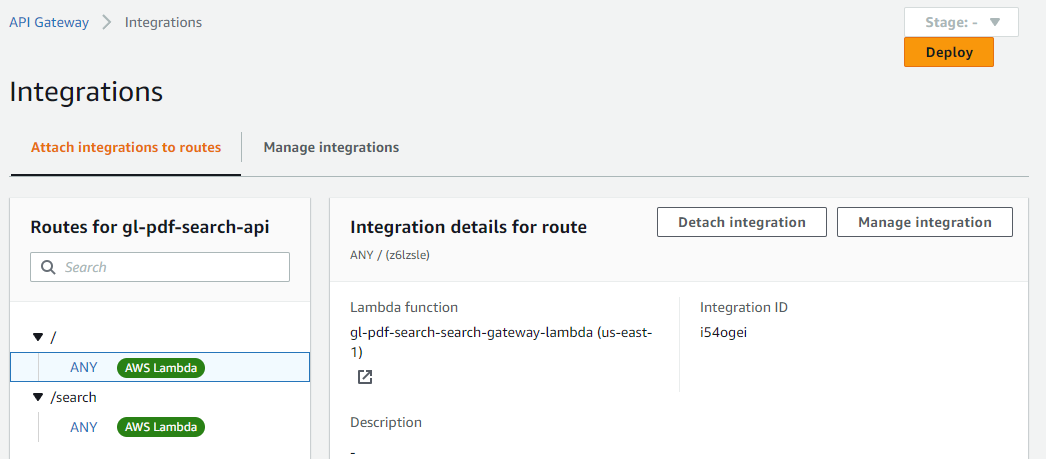
API HTTP Gateway: The API Gateway routes HTTP traffic to the lambda functions. The root route “/” is handled by SearchGateway lambda function to serve the “Search Page”. The route “/search” is handled when an HTTP Post is submitted from the search page to serve the results.



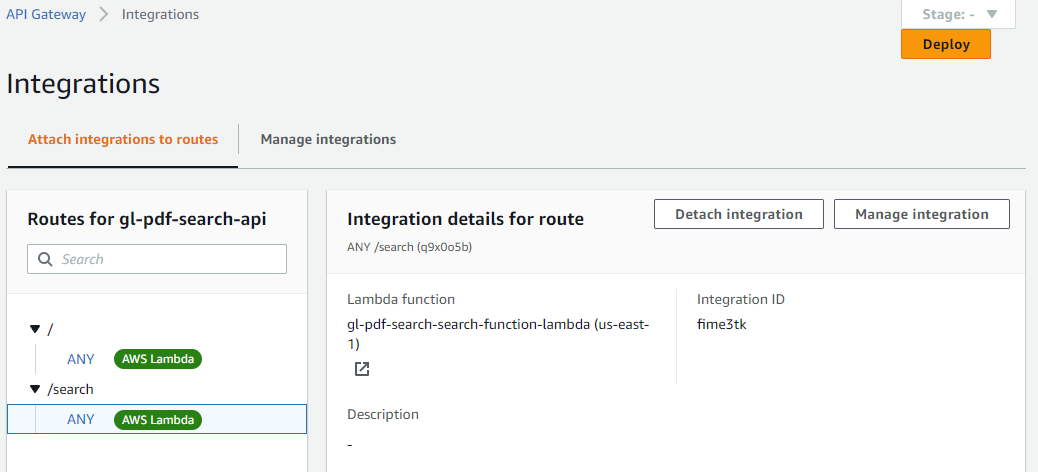
gl-pdf-search-api definition with the invoke URL.



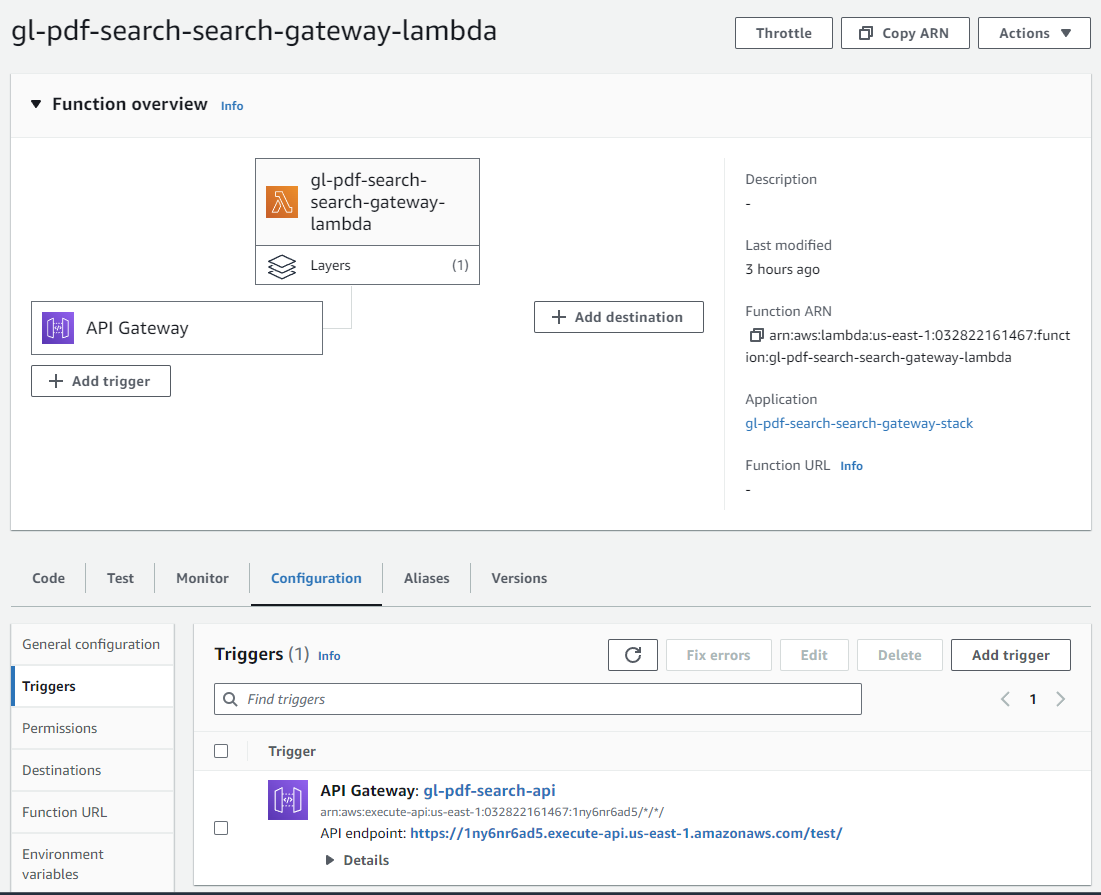
Integration for route “/”. Handles any HTTP call. Probably should be simplified to just handle GET http calls.



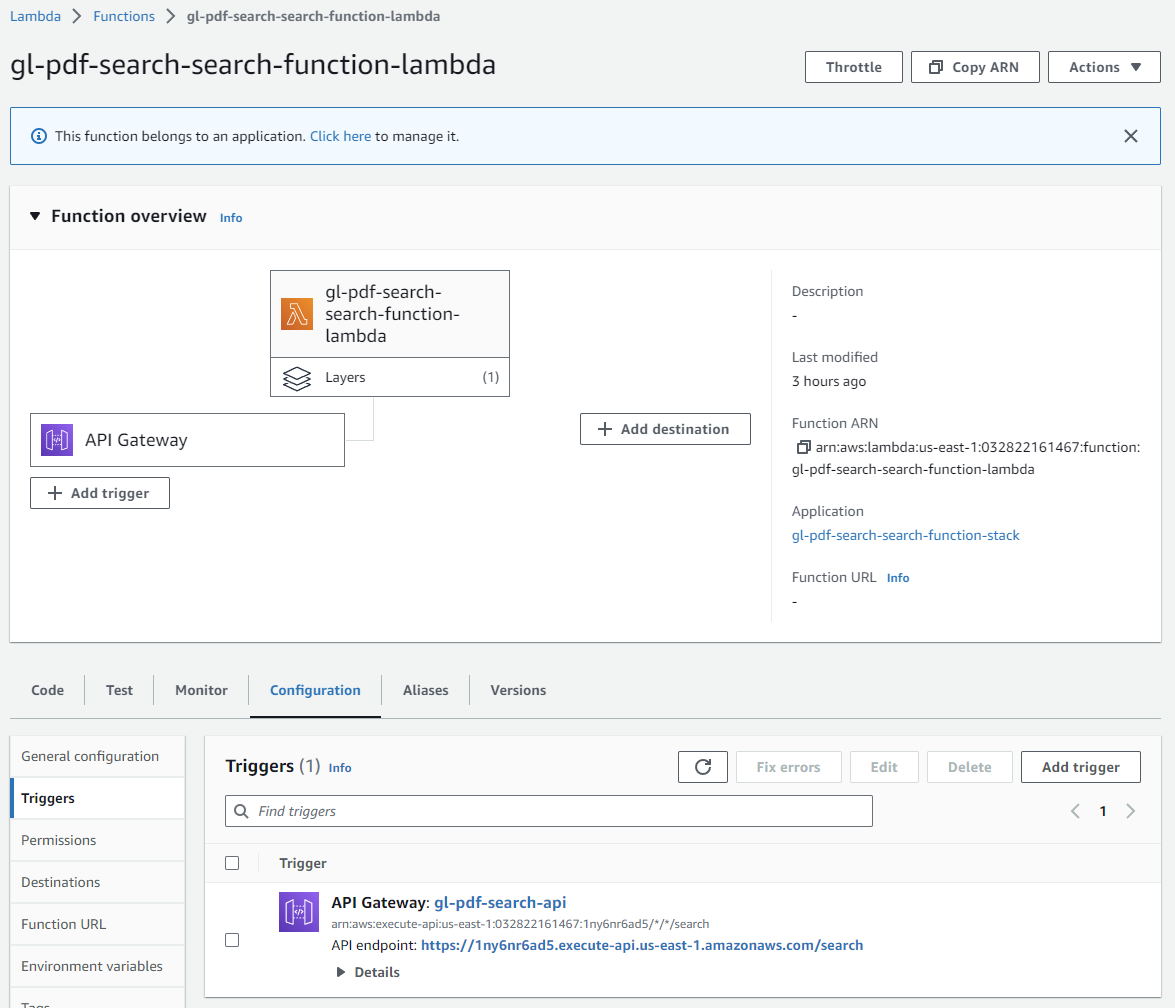
Integration for route “/search”. Handles any HTTP call. An alternative method would be to handle POST at “/” route. No need for “/search”.



SearchGateway lambda function with the API Gateway trigger



SearchFunction lambda function with the API Gateway trigger



Lessons & Observations:

This is one approach that could be taken to lower costs. The lambda functions are only billed by number of requests and execution times. This may be a little slow at times, especially if the lambda function has to be reloaded. But Concurrency Scaling and Availability Scaling can be used to improve performance.

It would probably make sense to keep the PDFToText and UploadToSearch lambda functions as they are as they are less likely to be called frequently and can run asynchronously. So latency isn’t an issue.

A monolithic solution using EC2 Instances or perhaps one or more docker images hosted in EKS can be used for the front end piece to make them more responsive. AutoScaling and LoadBalancing can be used to improve performance.

Security isn’t really taken into account when implementing this project. If this was a real production environment I would setup a dedicated VPC with private subnets and more restrictive security groups and IAM roles would be used.

cleanup.sh:

#!/bin/bash

# Delete HTTP API Gateway

apiId=$(aws apigatewayv2 get-apis --query "Items[?starts\_with(Name, 'gl-pdf-search')].ApiId" --output text)

aws apigatewayv2 delete-api --api-id $apiId

# Empty S3 buckets

BUCKETS=$(aws s3api list-buckets --query "Buckets[?starts\_with(Name, 'gl-pdf-search')].Name" --output text)

for BUCKET in $BUCKETS; do

    aws s3 rm s3://$BUCKET --recursive

done

# Delete CloudFormation stacks

STACKS=$(aws cloudformation list-stacks --query "StackSummaries[?starts\_with(StackName, 'gl-pdf-search') && StackStatus != 'DELETE\_COMPLETE'].StackName" --output text)

for STACK in $STACKS; do

    aws cloudformation delete-stack --stack-name $STACK

done

# Delete CodePipelines

PIPELINES=$(aws codepipeline list-pipelines --query "pipelines[?starts\_with(name, 'gl-pdf-search')].name" --output text)

for PIPELINE in $PIPELINES; do

    aws codepipeline delete-pipeline --name $PIPELINE

done

# Delete CodeBuilds

PROJECTS=$(aws codebuild list-projects --query "projects[?starts\_with(@, 'gl-pdf-search')]" --output text)

for PROJECT in $PROJECTS; do

    aws codebuild delete-project --name $PROJECT

done

# Delete CodeCommit repositories

REPOSITORIES=$(aws codecommit list-repositories --query "repositories[?starts\_with(repositoryName, 'gl-pdf-search')].repositoryName" --output text)

for REPOSITORY in $REPOSITORIES; do

    aws codecommit delete-repository --repository-name $REPOSITORY

done

# Delete Lambda layers

LAYERS=$(aws lambda list-layers --query "Layers[?starts\_with(LayerName, 'pdf-search')].LayerName" --output text)

for LAYER in $LAYERS; do

    VERSIONS=$(aws lambda list-layer-versions --layer-name $LAYER --query "LayerVersions[].Version" --output text)

    for VERSION in $VERSIONS; do

        aws lambda delete-layer-version --layer-name $LAYER --version-number $VERSION

    done

done

# Force delete S3 buckets

BUCKETS=$(aws s3api list-buckets --query "Buckets[?starts\_with(Name, 'gl-pdf-search')].Name" --output text)

for BUCKET in $BUCKETS; do

    aws s3 rb s3://$BUCKET --force

done

# Delete OpenSearch Service domain

aws opensearch delete-domain --domain-name pdf-search

# Stop bastion instance

INSTANCE\_ID=$(aws ec2 describe-instances --filters "Name=tag:Name,Values=user-mgmt-portal-bastion" --query "Reservations[].Instances[].InstanceId" --output text)

aws ec2 stop-instances --instance-ids $INSTANCE\_ID