VP - File Type Detection using AWS SAM

Background

The File Type Detection Product leverages azure serverless functions, with an AWS gateway api redirecting requests and limiting usage. The purpose of this exercise was to convert the azure components to AWS counterpart Lambda's.

Solution Overview

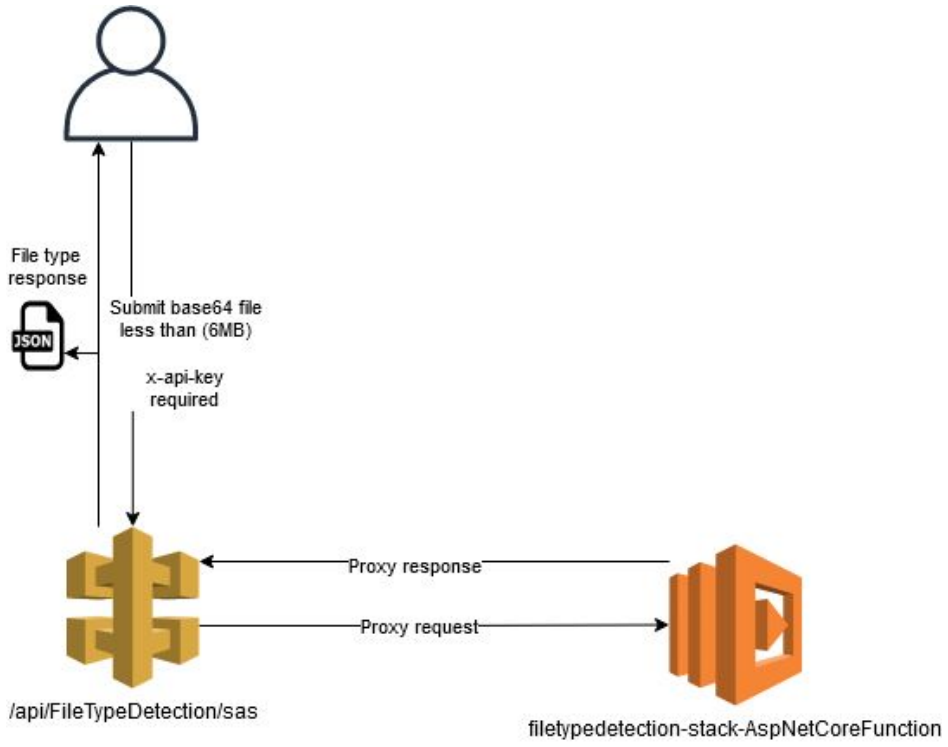
The Azure Function relies on a multipart binary file being in the body of the request, however this does not work as expected on a deployed Lambda, since decoding of the incoming lambda request seems to lose the content when received by the code. Two endpoints have been produced to allow for alternative mechanisms for file type detection:

The following endpoints can be reused by other products, provided they have a way of converting a file to base64 or providing a URL to a file. A good way of using the API would be by leveraging the s3 file upload product. Refer to the S3 File Upload Product documentation to find out how to generate read SAS tokens that can be used on this API

Endpoints

1. /api/FileTypeDetection/base64

Example Workflow Diagram:



This endpoint allows a client service to send a base64 encoding of a file in the body. A limitation of this however, is that you cannot send a file larger than lambda's maximum request size of 6MB.

Example Request Generated by Postman:

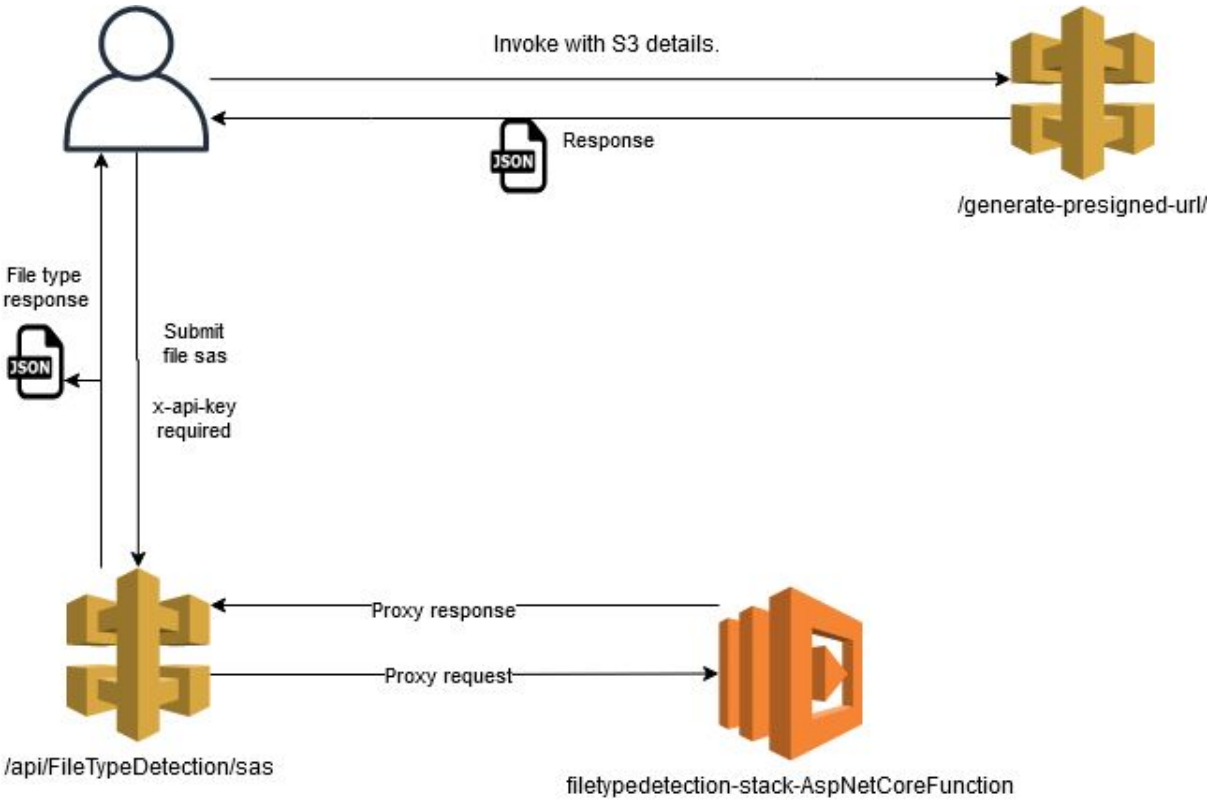
```
curl --location --request POST '[API GATEWAY URL]/api/FileTypeDetection/base64' \
--header 'Content-Type: application/json' \
--data-raw '{
    "Base64": “[YOUR BASE64 GOES HERE]”
}’
```

This endpoint does the following:

- Decodes the base64 file
    - If it is unsuccessful it returns BAD REQUEST
    - If it is successful then it will send the file to the core engine.
- The core engine will then try and determine the file type

- If successful the file type will be returned in an OK response in JSON format
- Otherwise Unknown will be returned instead.

2. /api/FileTypeDetection/sas



This endpoint allows a client to send a SAS url to a file to be downloaded and have its file type detected. An added benefit of this is the benefit of being able to upload files over the 6MB size threshold.

Example Request Generated by Postman:

```
curl --location --request POST '[API GATEWAY URL]/api/FileTypeDetection/sas' \
--header 'Content-Type: application/json' \
--data-raw '{
   "SasUrl": "[YUR SAS URL GOES HERE]"
}'
```

This endpoint does the following:

- Downloads the file found at SasUrl using a GET request.
  - If it is unsuccessful it returns BAD REQUEST
  - If it is successful then it will send the file to the core engine.
- The core engine will then try and determine the file type
  - If successful the file type will be returned in an OK response in JSON format
  - Otherwise Unknown will be returned instead.

Deployment Option 1 - Visual studio

Pre-requisites

1. Visual studio 2017 or 2019 installed.
2. AWS Toolkit extension for visual studio
3. AWS SAM CLI
4. Credentials configured for AWS
5. Permissions to access repo: cloud-serverless-sdk

Steps:

1. Clone repo https://github.com/filetrust/cloud-serverless-sdk
2. Open .sln file Glasswall.CloudSdk.AWS
3. Right click on project Glasswall.Cloudsdk.AWS.FileTypeDetection
4. Click publish to AWS lambda
5. For options:
    a. Account profile: Choose one that has permissions to create a cloud formation.
    b. Region: Where the lambda will run
    c. Stack name: This will be what the name of the stack the cloud formation will use, resources will use this in their name.
    d. S3 bucket: The bucket where the source code will be zipped and uploaded too

Deployment Option - 2: Manually create cloud formation

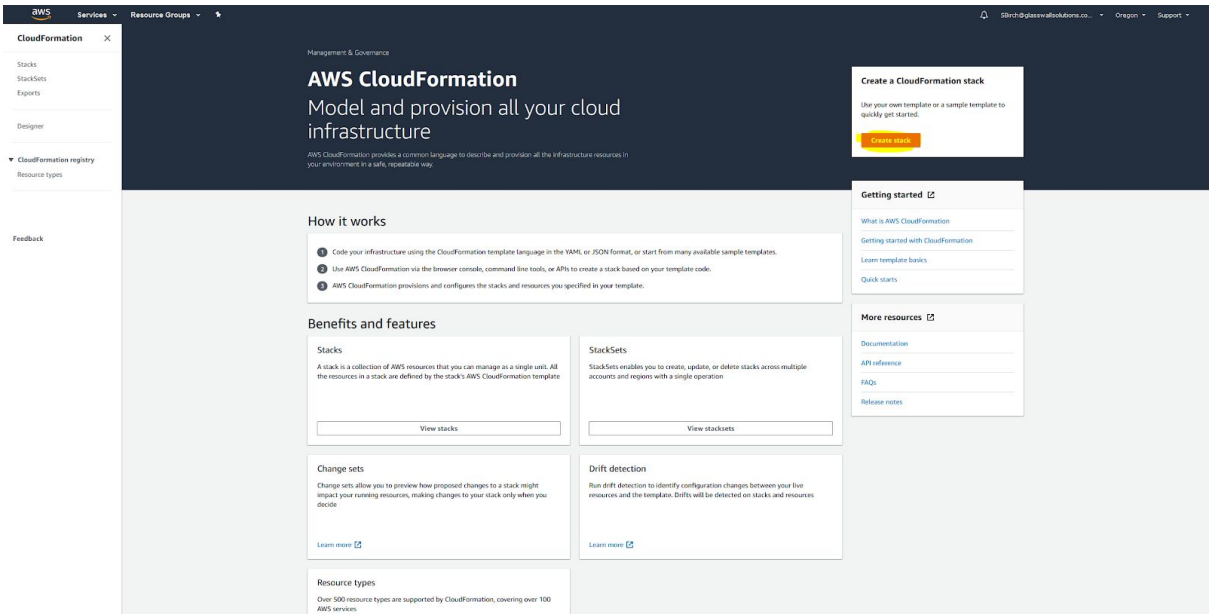
Pre-requisites

1. An aws account
2. Permissions to create cloud formations

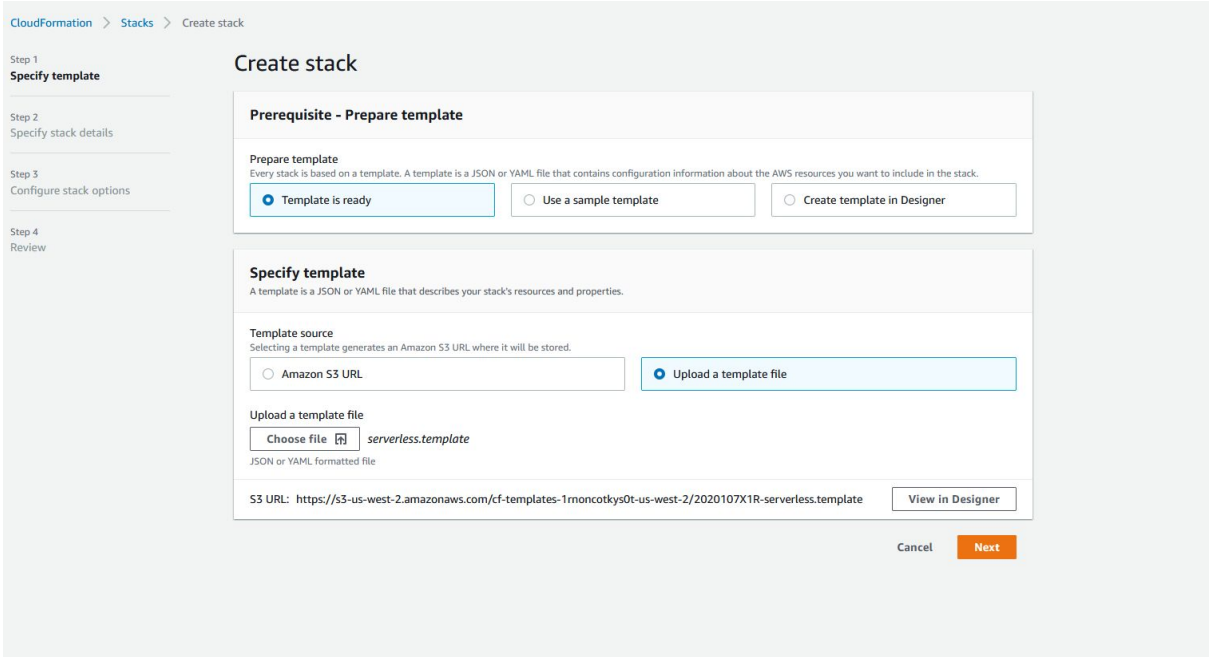
Steps

1. Create a bucket for the deployed code.
2. Upload Deployment zip into bucket, take note of the name
3. Edit serverless.template file, update the codeuri field to reflect the location of the deployment.
    a. Format example: s3://nameofbucket/nameoffolder/nameof.zip
4. Log on to aws
5. Navigate to AWS CloudFormation

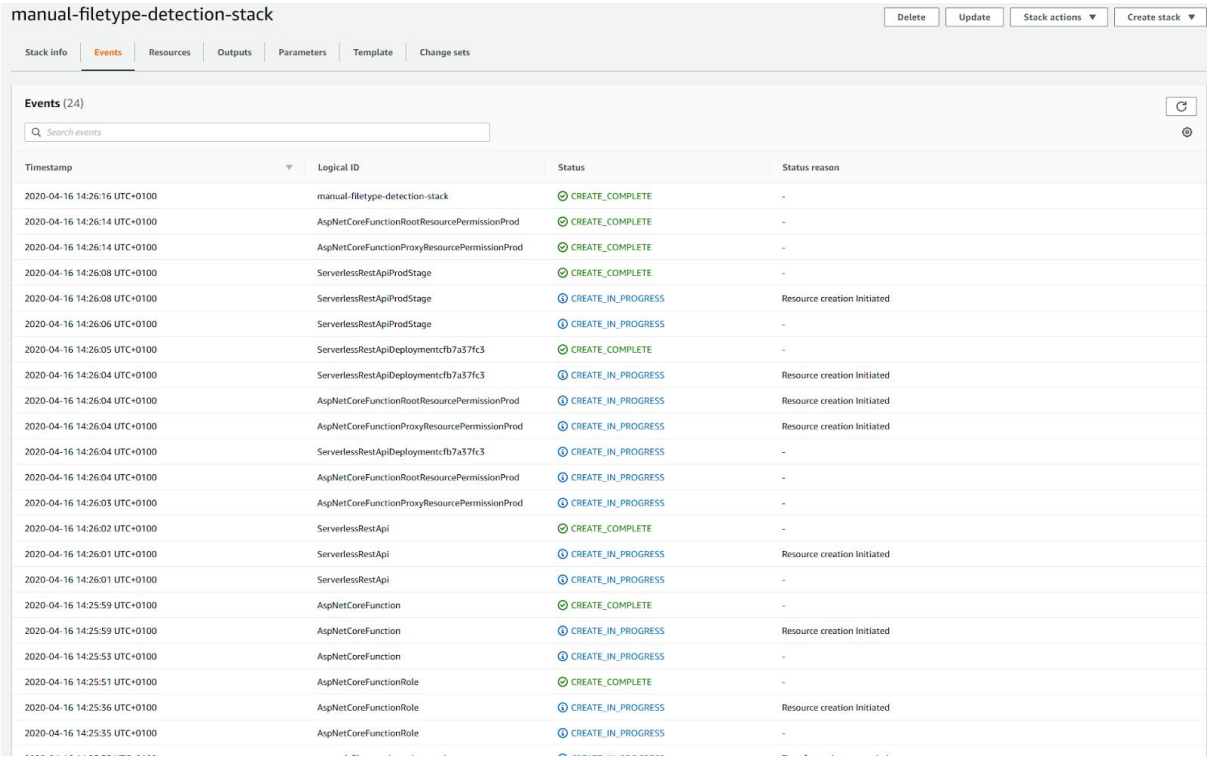
6. Create a stack



7. Select serverless.template from disk



8. Enter a name for the stack
9. Click next until end and finish
10. Success should look like this:


manual-filetype-detection-stack
Delete
Update
Stack actions
Create stack
Stack info
Events
Resources
Outputs
Parameters
Template
Change sets
Events (24)
Search events
Timestamp
Logical ID
Status
Status reason
2020-04-16 14:26:16 UTC+0100 manual-filetype-detection-stack CREATE_COMPLETE -
2020-04-16 14:26:14 UTC+0100 AspNetCoreFunctionRootResourcePermissionProd CREATE_COMPLETE -
2020-04-16 14:26:14 UTC+0100 AspNetCoreFunctionProxyResourcePermissionProd CREATE_COMPLETE -
2020-04-16 14:26:08 UTC+0100 ServerlessRestApiProdStage CREATE_COMPLETE -
2020-04-16 14:26:08 UTC+0100 ServerlessRestApiProdStage CREATE_IN_PROGRESS Resource creation Initiated
2020-04-16 14:26:06 UTC+0100 ServerlessRestApiProdStage CREATE_IN_PROGRESS -
2020-04-16 14:26:05 UTC+0100 ServerlessRestApiDeploymentcfb7a37fc3 CREATE_COMPLETE -
2020-04-16 14:26:04 UTC+0100 ServerlessRestApiDeploymentcfb7a37fc3 CREATE_IN_PROGRESS Resource creation Initiated
2020-04-16 14:26:04 UTC+0100 AspNetCoreFunctionRootResourcePermissionProd CREATE_IN_PROGRESS Resource creation Initiated
2020-04-16 14:26:04 UTC+0100 AspNetCoreFunctionProxyResourcePermissionProd CREATE_IN_PROGRESS Resource creation Initiated
2020-04-16 14:26:04 UTC+0100 ServerlessRestApiDeploymentcfb7a37fc3 CREATE_IN_PROGRESS -
2020-04-16 14:26:04 UTC+0100 AspNetCoreFunctionRootResourcePermissionProd CREATE_IN_PROGRESS -
2020-04-16 14:26:03 UTC+0100 AspNetCoreFunctionProxyResourcePermissionProd CREATE_IN_PROGRESS -
2020-04-16 14:26:02 UTC+0100 ServerlessRestApi CREATE_COMPLETE -
2020-04-16 14:26:01 UTC+0100 ServerlessRestApi CREATE_IN_PROGRESS Resource creation Initiated
2020-04-16 14:26:01 UTC+0100 ServerlessRestApi CREATE_IN_PROGRESS -
2020-04-16 14:25:59 UTC+0100 AspNetCoreFunction CREATE_COMPLETE -
2020-04-16 14:25:59 UTC+0100 AspNetCoreFunction CREATE_IN_PROGRESS Resource creation Initiated
2020-04-16 14:25:53 UTC+0100 AspNetCoreFunction CREATE_IN_PROGRESS -
2020-04-16 14:25:51 UTC+0100 AspNetCoreFunctionRole CREATE_COMPLETE -
2020-04-16 14:25:36 UTC+0100 AspNetCoreFunctionRole CREATE_IN_PROGRESS Resource creation Initiated
2020-04-16 14:25:35 UTC+0100 AspNetCoreFunctionRole CREATE_IN_PROGRESS -