NUM Project: Web Scraping using Companies House and FCA API

MSIN0166: Data Engineering Group Coursework

Abstract

Throughout this project we aim to collect data which can be integrated into NUM Technology Ltd’s DNS-based way of retrieving and storing data. To do this, we created a database of company information not yet collected by NUM with individual companies’ domain name as the key.

To obtain our dataset, we matched a regularly updated csv file containing all the UK Government’s Companies House data with data obtained from the Financial Conduct Authority (FCA) via an API. We found the most efficient way to extract information from the FCA API was to collect reference numbers from FCA website’s download section and run these through the API. The reference numbers available for download on the FCA website constituted 38 thousand of the total 59 thousand listed FCA entities.

As a result, we successfully obtained over 8 thousand data points. Each point contains the entity's domain name, as well as the FCA number and company number which were both previously uncollected by NUM.

Project Background/Motivation

Num Technology Ltd is a DNS-based alternative to the World Wide Web and is used for storing and retrieving small, structured data. NUM was founded to tackle the inefficiencies that arise when searching for precise information, such as contact information and bank details, on the web. These efficiencies are achieved because NUM is built on top of the DNS, which allows for the domain names to be converted to the relevant information that is important to end users. The stored data is currently being scraped from individual websites. Therefore, we need to look at other sources of data to increase the amount and type of data offered to customers

Project Plan/Goal

The goal of the project is to collect relevant information that can be compiled to add value to NUM’s business. The two main sources of data for this project are the Companies House and Financial Conduct Authority (FCA) databases. The aim is to merge data from both sources so that NUM can provide a more complete account of company information to their customers. We plan to use Airflow combined with Lambda functions to execute the tasks of scraping the website, transforming the data into a suitable format and finally storing it in a DynamoDB database.

Data Sources

Companies House csv: http://download.companieshouse.gov.uk/en_output.html

FCA collection of files: https://register.fca.org.uk/s/resources#Downloads

Link to github: https://github.com/Lukebotbol/Data_Engineering_pipeline

Business Value for NUM

NUM currently scrapes information directly from company websites and stores it in the DNS. Some of the information successfully being scraped includes company names, straplines and social media handles. Other items such as addresses, company logo, and telephone number labels are scraped with partial success.

We intend to add value to NUM by expanding the information currently being offered, as well as introducing new data sources for NUM to use. Public datasets such as Companies House and FCA offer large and reliable information about companies that are very useful to NUM and potential end users. After careful evaluation of potential information that would be useful to NUM, we decided to collect the following:

1.) Domain Name

2.) Company Name

3.) Company Number

4.) FCA Number

5.) Company Category

6.) Address

7.) Address Type

8.) SIC code

9.) Phone number

10.) Fax Number

11.) Country of Origin

Obtaining the Dataset

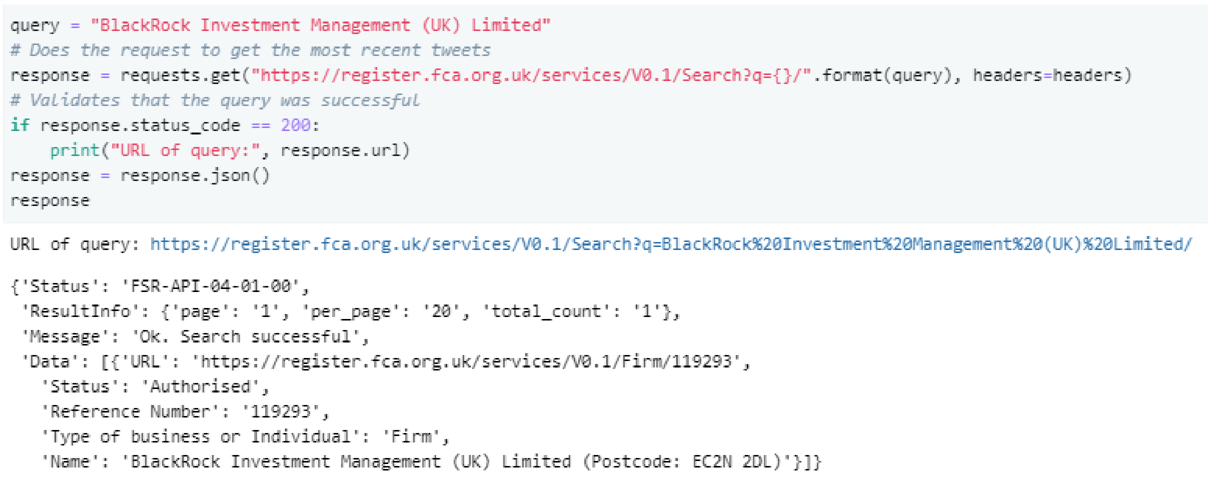
Before starting with our search, NUM highlighted how we were free to find the data from anywhere we wanted but stressed two constraints. First, we were not to collect data from platforms such as Google and Facebook as this goes against their businesses' philosophy of bypassing these companies. Second, we were not to collect data from companies which sell data as a business model. These constraints would cause us a lot of trouble but introduced us to the reality of collecting data in a commercial setting.

We started by searching through various company databases including Open Corporate, Crunchbase, Doogal and Yell. Even though these databases are valuable in their own right, we found that the majority of them did not allow web scraping for commercial purposes or involved costs which we did not want to incur. However, two databases, the FCA and

Companies House met the requirements and both had APIs. Therefore, we looked into these databases to find the data wanted by NUM. Companies House had company numbers, addresses, phone numbers and company type. The FCA had FCA numbers and most importantly, domain names.

Making our lives easier, we found a csv of all the Companies House data which is updated on a monthly basis online. Therefore, we needed to find a way to match the Companies House information with the FCA information. We realised that we could use the FCA Search API to search by company names to extract company names and reference numbers. The idea was to select all the company names from the Companies House csv and run them through the FCA Search API to get the reference numbers. However, the Companies House file had around 6 million instances and there are only 59 thousand companies registered by the FCA. Running all instances through the FCA Search API would be computationally expensive and time consuming.

The first approach to tackle this problem was to filter through the SIC codes, to match those of companies typically registered by the FCA. Examples include financial advisers, banks, insurancers and brokerages. We then used the company names from the Companies House csv to run through the FCA Search API which returned the FCA reference number. This number was needed to collect further information from the FCA.

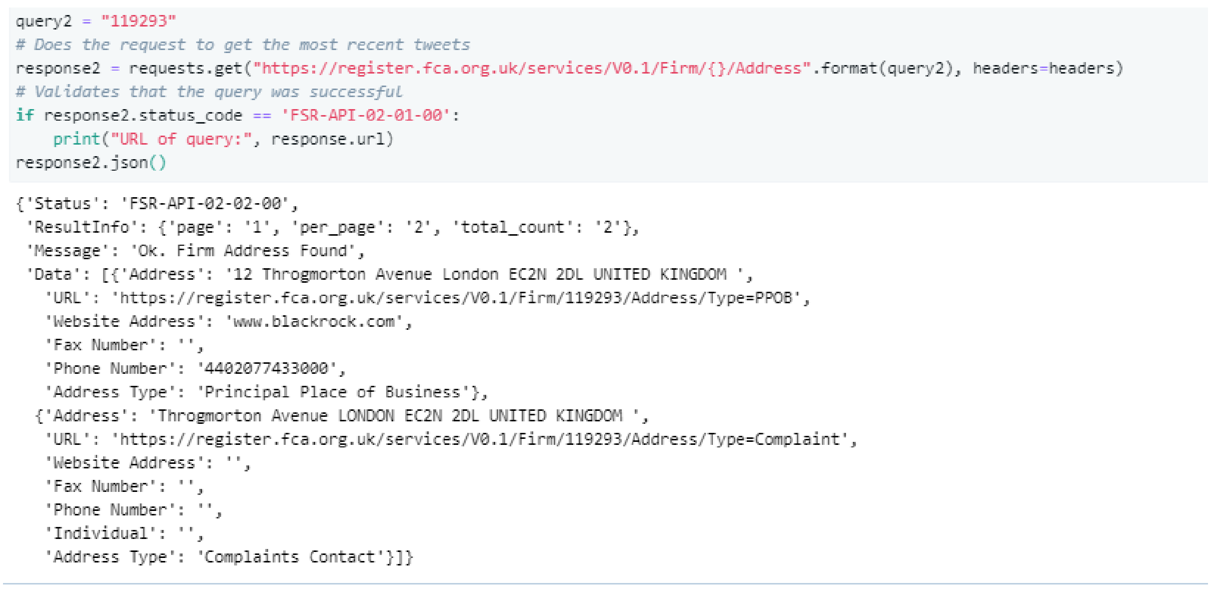


```python
query = "BlackRock Investment Management (UK) Limited"
# Does the request to get the most recent tweets
response = requests.get("https://register.fca.org.uk/services/V0.1/Search?q={}/".format(query), headers=headers)
# Validates that the query was successful
if response.status_code == 200:
    print("URL of query:", response.url)
response = response.json()
response
```

```
URL of query: https://register.fca.org.uk/services/V0.1/Search?q=BlackRock%20Investment%20Management%20(UK)%20Limited/

{'Status': 'FSR-API-04-01-00',
 'ResultInfo': {'page': '1', 'per_page': '20', 'total_count': '1'},
 'Message': 'Ok. Search successful',
 'Data': [{'URL': 'https://register.fca.org.uk/services/V0.1/Firm/119293',
   'Status': 'Authorised',
   'Reference Number': '119293',
   'Type of business or Individual': 'Firm',
   'Name': 'BlackRock Investment Management (UK) Limited (Postcode: EC2N 2DL)'}]}
```

Figure 1. Example use of the FCA Search API

Then we used the collected reference numbers and ran them through the FCA Address API which returned the needed website address and merged it with the FCA Search data collected in the previous step.



```
query2 = "119293"
# Does the request to get the most recent tweets
response2 = requests.get("https://register.fca.org.uk/services/V0.1/Firm/{}/Address".format(query2), headers=headers)
# Validates that the query was successful
if response2.status_code == 'FSR-API-02-01-00':
    print("URL of query:", response.url)
response2.json()
```

```
{'Status': 'FSR-API-02-02-00',
 'ResultInfo': {'page': '1', 'per_page': '2', 'total_count': '2'},
 'Message': 'Ok. Firm Address Found',
 'Data': [{'Address': '12 Throgmorton Avenue London EC2N 2DL UNITED KINGDOM ',
   'URL': 'https://register.fca.org.uk/services/V0.1/Firm/119293/Address/Type=PPOB',
   'Website Address': 'www.blackrock.com',
   'Fax Number': '',
   'Phone Number': '4402077433000',
   'Address Type': 'Principal Place of Business'},
  {'Address': 'Throgmorton Avenue LONDON EC2N 2DL UNITED KINGDOM ',
   'URL': 'https://register.fca.org.uk/services/V0.1/Firm/119293/Address/Type=Complaint',
   'Website Address': '',
   'Fax Number': '',
   'Phone Number': '',
   'Individual': '',
   'Address Type': 'Complaints Contact'}]}
```

Figure 2. Example use of the FCA Address API

The next step was to merge this FCA data with the Companies House data. Unfortunately, neither dataset had a common reference number. We tried merging on postcodes but realised multiple companies can share the same postcode if, for example, they are situated in the same office. So we decided to standardise the company names in both sets and tried merging on company names.

An issue with this approach was that, even after filtering by SIC codes, we would need to run over a million searches through the FCA API which was not an efficient approach. Furthermore, if one runs a search for a company not included in the FCA, the API would return information of 20 companies similar to the search result. This meant our searches resulted in many duplicates.

After discussing with NUM, we decided to send the FCA an email requesting a list of company names to no avail. However, we found a collection of csv files on the FCA website which subgrouped certain listed companies which brought us to our second approach.

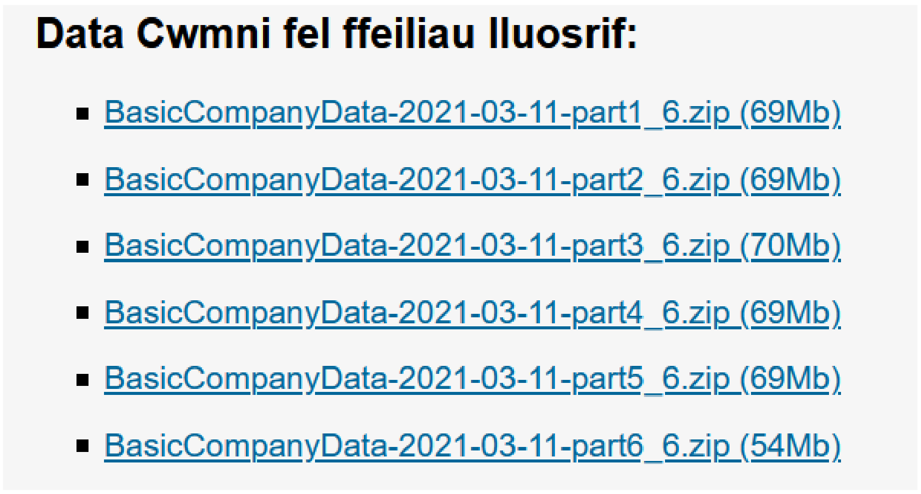
Our second and final approach skipped running company names from the Companies House csv through the Search API. Instead we took FCA reference numbers directly from csv files on the FCA website and ran those. The various csv files concatenated only contained 38 thousand of the 59 thousand total listed companies. However, this approach was more efficient as we only needed to conduct one API search and knew that each search would be successful.

The main issue we faced in the data collection process was attaining domain names. The FCA website was the only suitable database that had websites listed. However, even here only around 15% of companies had their domain names listed. A further issue was that a single company could have multiple FCA entities all with the same website address listed. This meant our approach had a limit to the amount of useful data it could collect. After a further meeting with NUM, different approaches to attaining website addresses were discussed such as crawling the web for a potential follow-up project.

Data Storage and Processing

In order to create a data pipeline to store and process the data, we used a variety of tools. After gaining an understanding of which dataset to use and how to process it, we had to find a way to create a data pipeline that could run every month.

First, we built a web scraper within a Lambda function to retrieve all the csv files from Companies House and to store them in a S3 Bucket called 'data-eng-group'. We decided to use a scraper as the files are updated monthly.



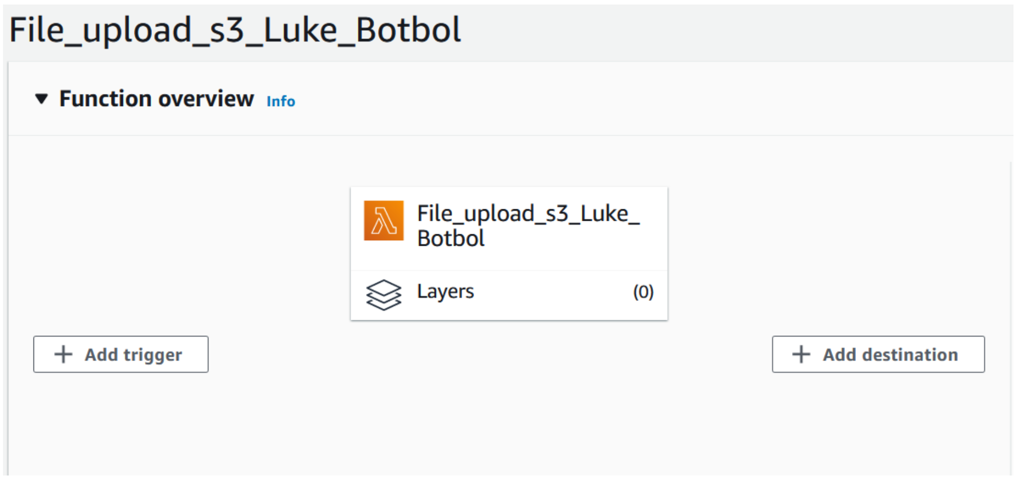
Data Cwmni fel ffeiliau lluosrif:

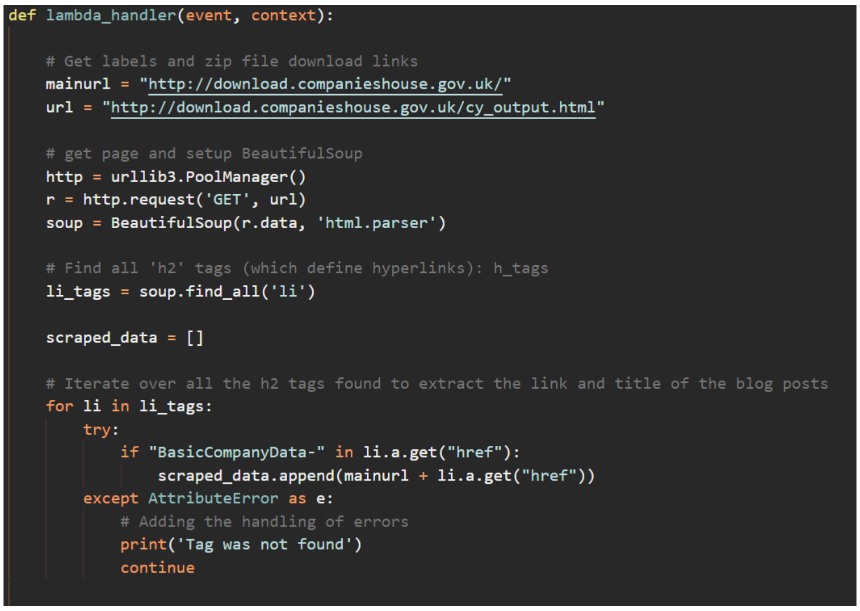
- BasicCompanyData-2021-03-11-part1_6.zip (69Mb)
- BasicCompanyData-2021-03-11-part2_6.zip (69Mb)
- BasicCompanyData-2021-03-11-part3_6.zip (70Mb)
- BasicCompanyData-2021-03-11-part4_6.zip (69Mb)
- BasicCompanyData-2021-03-11-part5_6.zip (69Mb)
- BasicCompanyData-2021-03-11-part6_6.zip (54Mb)

Figure 3. Companies House data in segments

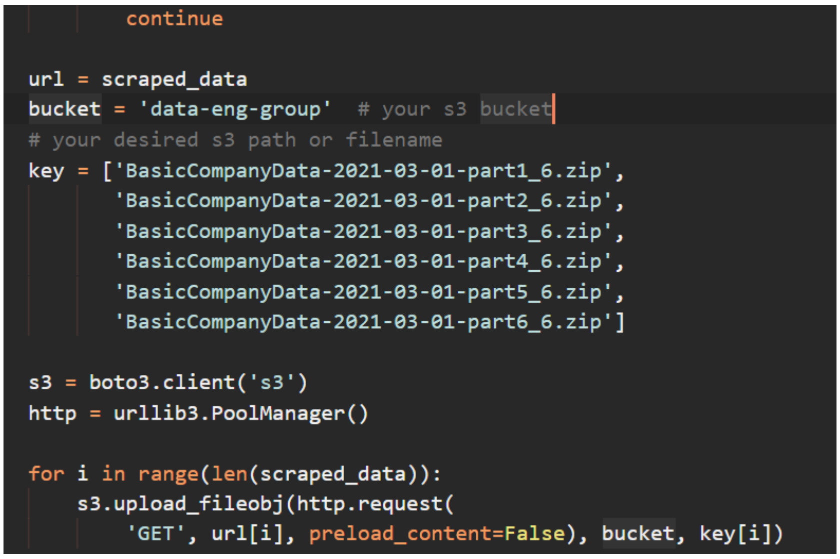
Lambda functions were chosen to run the tasks due to their serverless approach. Indeed, we could schedule the task within Airflow without the need to start and stop an EC2 instance which would be very expensive. Furthermore, Lambda provides an easy way to store the data to S3 bucket.
One of the main challenges of this task was integrating Beautiful Soup within Lambda. After some research, we managed to find a package containing all modules necessary for the scraping which has been added into our GitHub repository.

After collecting the required packages, we needed to write a Lambda function which called to scrape the data:


File_upload_s3_Luke_Botbol



```python
def lambda_handler(event, context):

    # Get labels and zip file download links
    mainurl = "http://download.companieshouse.gov.uk/"
    url = "http://download.companieshouse.gov.uk/cy_output.html"

    # get page and setup BeautifulSoup
    http = urllib3.PoolManager()
    r = http.request('GET', url)
    soup = BeautifulSoup(r.data, 'html.parser')

    # Find all 'h2' tags (which define hyperlinks): h_tags
    li_tags = soup.find_all('li')

    scraped_data = []

    # Iterate over all the h2 tags found to extract the link and title of the blog posts
    for li in li_tags:
        try:
            if "BasicCompanyData-" in li.a.get("href"):
                scraped_data.append(mainurl + li.a.get("href"))
        except AttributeError as e:
            # Adding the handling of errors
            print('Tag was not found')
            continue
```



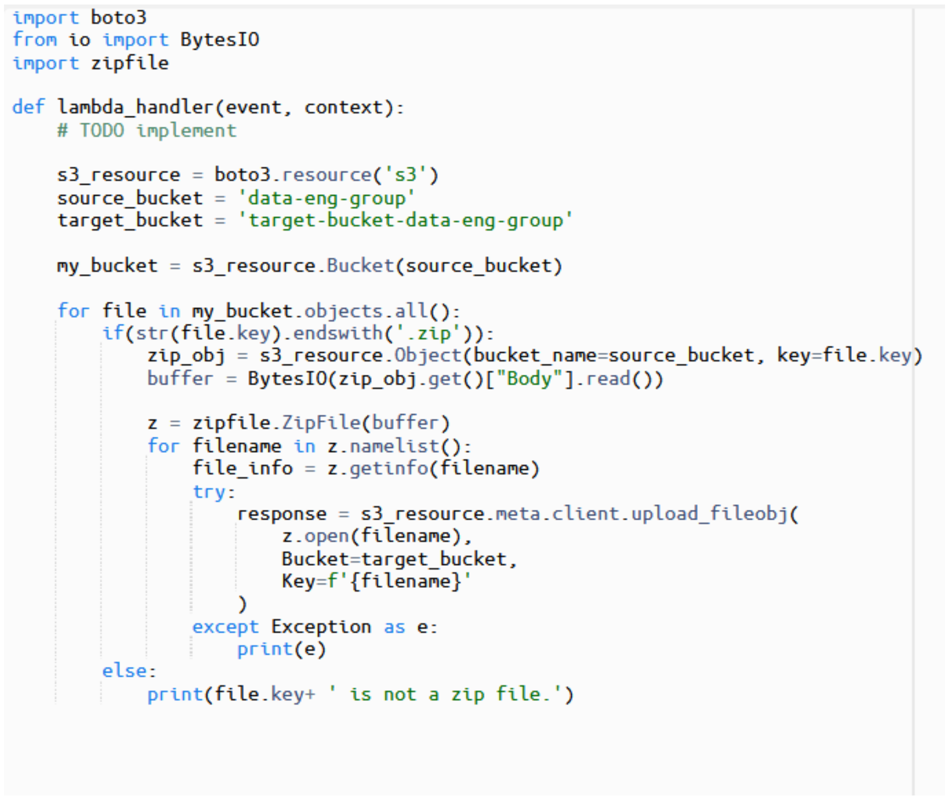
```python
            continue

    url = scraped_data
    bucket = 'data-eng-group'  # your s3 bucket
    # your desired s3 path or filename
    key = ['BasicCompanyData-2021-03-01-part1_6.zip',
           'BasicCompanyData-2021-03-01-part2_6.zip',
           'BasicCompanyData-2021-03-01-part3_6.zip',
           'BasicCompanyData-2021-03-01-part4_6.zip',
           'BasicCompanyData-2021-03-01-part5_6.zip',
           'BasicCompanyData-2021-03-01-part6_6.zip']

    s3 = boto3.client('s3')
    http = urllib3.PoolManager()

    for i in range(len(scraped_data)):
        s3.upload_fileobj(http.request(
            'GET', url[i], preload_content=False), bucket, key[i])
```

Figure 4. Lambda Function Overview

The function selects all the files starting with ‘BasicCompanyData-’ in HTML having the tag ‘li’.

The next issue we faced was figuring out a way to unzip the scrapped files. To solve this problem, we built a second Lambda function which unzipped the files and stored them in a second S3 bucket called ‘target-bucket-data-eng’. We chose Lambda to execute this task for the same reasons as above, it provided a serverless approach that could be triggered within Airflow. We decided not to implement an automatic trigger for this function as we believed it was easier to control within Airflow.

The function to unzip the scrapped file is called ‘unzip-target-bucket-Luke-Botbol’, the code can also be found in our GitHub repository.



```python
import boto3
from io import BytesIO
import zipfile

def lambda_handler(event, context):
    # TODO implement

    s3_resource = boto3.resource('s3')
    source_bucket = 'data-eng-group'
    target_bucket = 'target-bucket-data-eng-group'

    my_bucket = s3_resource.Bucket(source_bucket)

    for file in my_bucket.objects.all():
        if(str(file.key).endswith('.zip')):
            zip_obj = s3_resource.Object(bucket_name=source_bucket, key=file.key)
            buffer = BytesIO(zip_obj.get()["Body"].read())

            z = zipfile.ZipFile(buffer)
            for filename in z.namelist():
                file_info = z.getinfo(filename)
                try:
                    response = s3_resource.meta.client.upload_fileobj(
                        z.open(filename),
                        Bucket=target_bucket,
                        Key=f'{filename}'
                    )
                except Exception as e:
                    print(e)
        else:
            print(file.key+ ' is not a zip file.')
```

Figure 5. Unzip target with Lambda

With these two functions we were able to retrieve our data in a csv format and add it in a S3 bucket in a serverless approach.

Furthermore, we tried implementing the same strategy to retrieve the csv files from the FCA website in order to pursue the second approach as detailed in the *Obtaining the Dataset* section above. However, we faced some difficulties as the FCA did not authorise web-scraping. We therefore had to download the files locally and upload them into our target-bucket-data-eng on S3, which is a limitation of our data pipeline. Nevertheless, the FCA provides no information whether those files are updated. Therefore, there is no incentive in integrating a web-scrape every month as it would only increase costs and provide no value.

Having all the datasets in an S3 bucket, we could start implementing our scheduling using Airflow. Airflow makes it easy to monitor the state of a pipeline in their UI, and to build DAGs with complex fan-in and fan-out relationships between tasks.

The first step in the pipeline retrieves all the data from the FCA API. In order to implement this into the DAG, we first loaded the csv files downloaded from the FCA into S3. To retrieve those files, we create a connection to S3 on Airflow and create 6 variables for the 6 files with the name of the file as key. We loaded the files as Bytes objects and read them using pandas. After retrieving the reference numbers, we collect the data using the FCA API. Since task instances of the same DAG can be executed on different workers, we decided to read these files from S3 and use the FCA API in the same task.

The DAG's next task is to trigger the Lambda function set-up to scrape the Companies House files. To trigger this function we used a Lambda hook and a python operator. For the following task, which is to activate the second Lambda function to unzip those files, we again used the Lambda hook. However, with this task we faced another limitation. The function worked successfully when testing on AWS Lambda; however, the task was marked as failed on Airflow. After some research, we discovered that this was due to Airflow's time limit. We solved this problem by adding a 'time.sleep' function for both Lambda triggers. Furthermore, we also added a 'delay-python-task' in order for Lambda to have enough time to execute and unzip the files before going to the next task which will use those files.

The final step of the DAG is to merge the results with the Companies House data in order to populate our findings. This step was a major challenge in our project. Indeed, to merge those files we first tried to read the Companies House data files from S3 on Airflow. However, the total size of the files was above 2.3GB. As Airflow has different workers, the task was probably executed by different workers which made it impossible to retrieve a dataframe. To overcome this challenge we first decided to convert all the Companies House files into parquet format for its efficient data-compression. However, this approach didn't work as we realised that the issue was more related to the conversion into a dataframe which required a lot of in-memory storage.

Discussing the issue with the TAs we decided to solve the problem using a Lambda function to read and merge both datasets. As the task is relatively simple, it can be triggered with Airflow. Therefore Lambda is less expensive than other options such as Spark or SageMaker.

In order to use Lambda we had to add a new task into our DAG to save our result from the FCA API into an S3 bucket called 'numdata'. We passed the result of the API request onto the next task by converting the data frame into a dictionary using xcom. We then wrote a function to save the file to the S3 bucket.

Having the FCA result and Companies House files in S3, we could then implement our function. Yet again, we encountered several issues using the Lambda function. Indeed, we needed to add a Pandas layer which wasn't included in Lambda's environment. After some research, we managed to find a GitHub repo with a Pandas layer (https://github.com/GozdeKocabas/regnum-address). However, we still faced a problem while reading the file into a Dataframe as Pandas relied on the module fsspec to read files which was also not included in the environment. After several attempts we couldn't manage to install this module into the environment. Therefore, we solved this issue by downloading the 'awswrangler' layer which doesn't rely on fsspec. The layer was found on a github repo (https://github.com/awslabs/aws-data-wrangler/releases) .

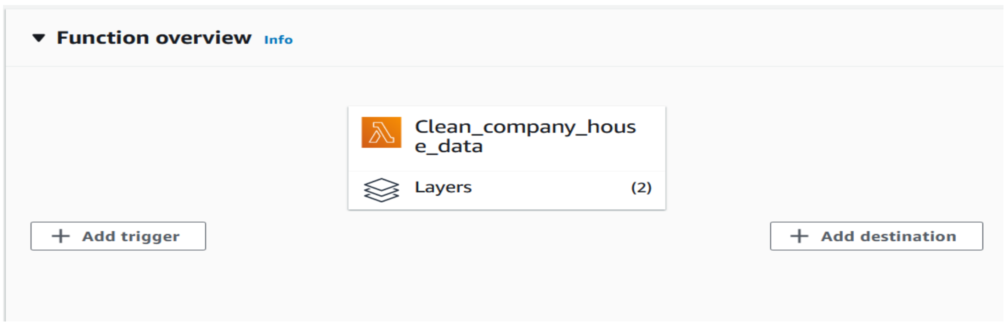
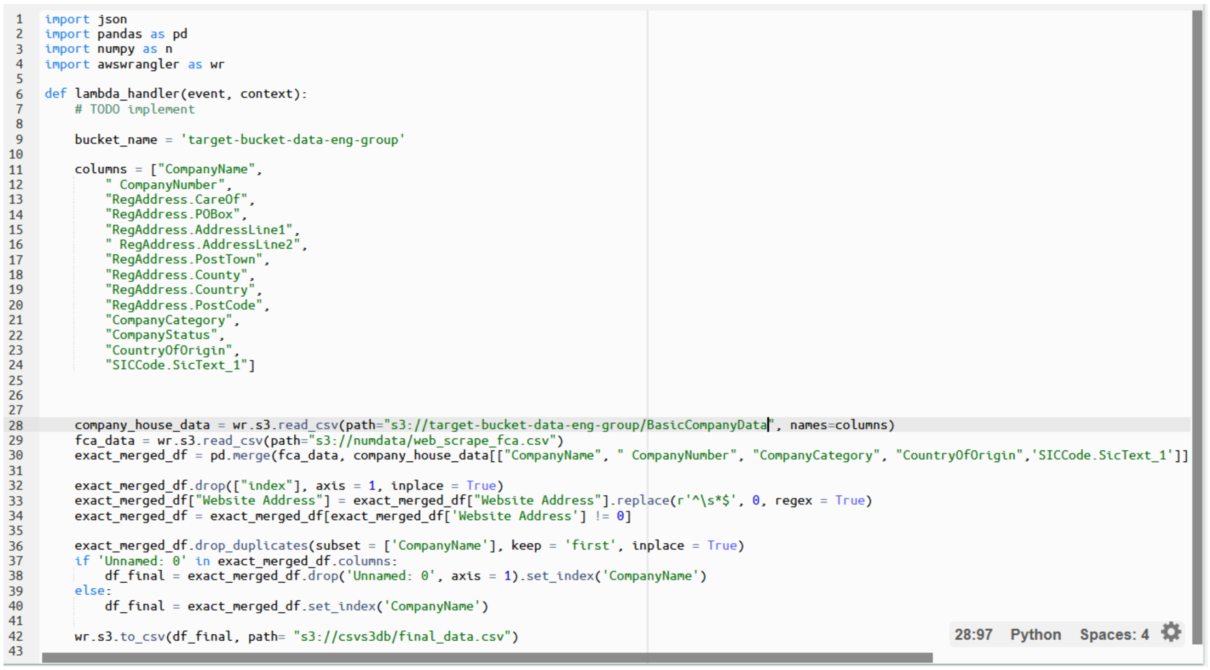


Figure 6. Merge Company House Lambda Function

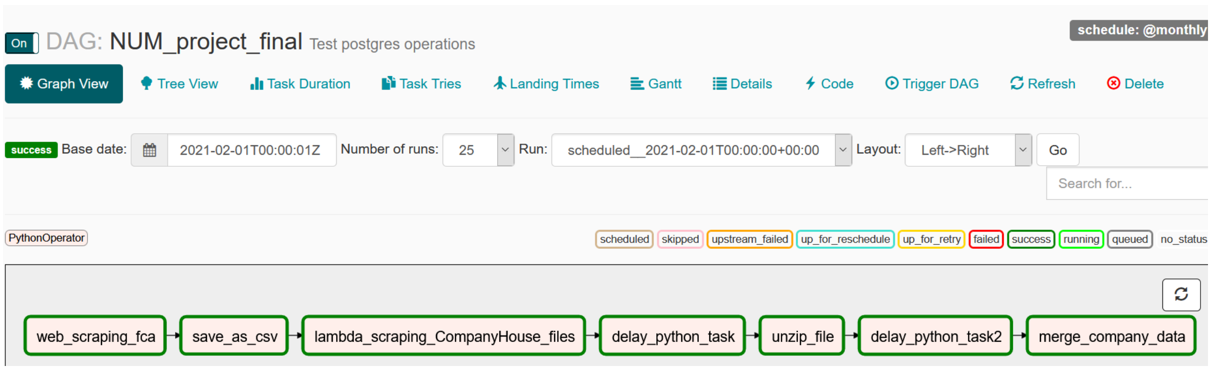
Having solved all the issues related to the environment, we finally could run the Lambda function called ‘Clean_company_house_data’, which would merge the two files and save the result into an S3 bucket called ‘csvs3db’. We wanted to save the file in a ‘csv’ format in order for NUM to have structured data with ‘website address’ as the first column.



```python
import json
import pandas as pd
import numpy as n
import awswrangler as wr

def lambda_handler(event, context):
    # TODO implement

    bucket_name = 'target-bucket-data-eng-group'

    columns = ["CompanyName",
        " CompanyNumber",
        "RegAddress.CareOf",
        "RegAddress.POBox",
        "RegAddress.AddressLine1",
        " RegAddress.AddressLine2",
        "RegAddress.PostTown",
        "RegAddress.County",
        "RegAddress.Country",
        "RegAddress.PostCode",
        "CompanyCategory",
        "CompanyStatus",
        "CountryOfOrigin",
        "SICCode.SicText_1"]



    company_house_data = wr.s3.read_csv(path="s3://target-bucket-data-eng-group/BasicCompanyData", names=columns)
    fca_data = wr.s3.read_csv(path="s3://numdata/web_scrape_fca.csv")
    exact_merged_df = pd.merge(fca_data, company_house_data[["CompanyName", " CompanyNumber", "CompanyCategory", "CountryOfOrigin",'SICCode.SicText_1']]

    exact_merged_df.drop(["index"], axis = 1, inplace = True)
    exact_merged_df["Website Address"] = exact_merged_df["Website Address"].replace(r'^\s*$', 0, regex = True)
    exact_merged_df = exact_merged_df[exact_merged_df['Website Address'] != 0]

    exact_merged_df.drop_duplicates(subset = ['CompanyName'], keep = 'first', inplace = True)
    if 'Unnamed: 0' in exact_merged_df.columns:
        df_final = exact_merged_df.drop('Unnamed: 0', axis = 1).set_index('CompanyName')
    else:
        df_final = exact_merged_df.set_index('CompanyName')

    wr.s3.to_csv(df_final, path= "s3://csvs3db/final_data.csv")

```

Figure 7. Lambda Function Code

The final step of the DAG was to trigger the Lambda function using the Lambda Hook.



On DAG: NUM_project_final Test postgres operations

schedule: @monthly

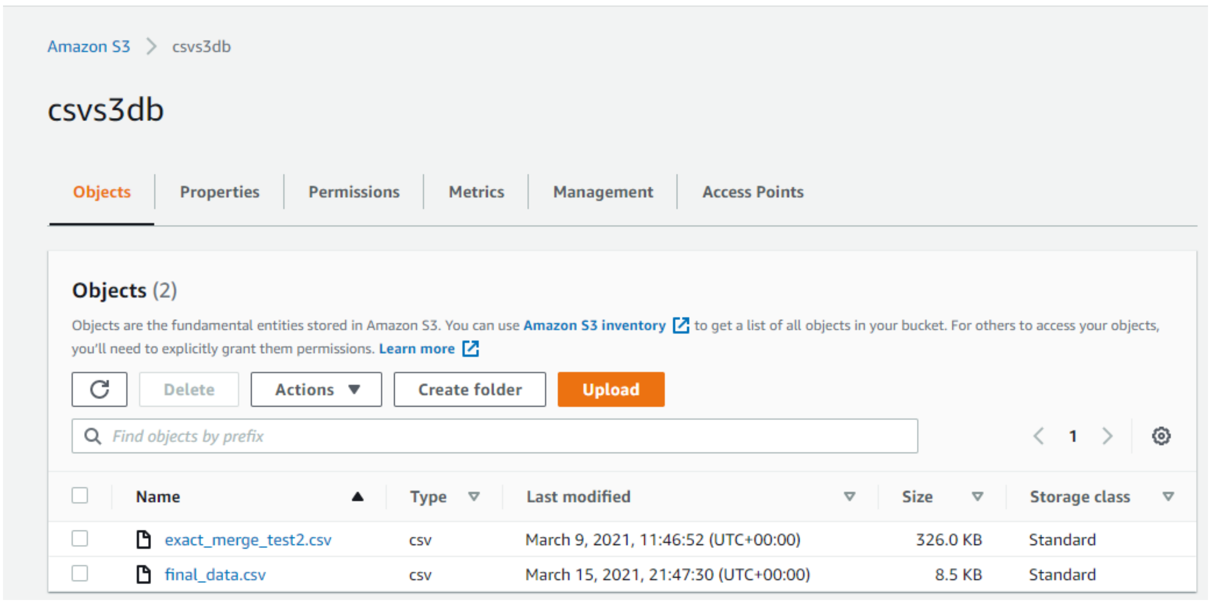
success Base date: 2021-02-01T00:00:01Z Number of runs: 25 Run: scheduled__2021-02-01T00:00:00+00:00 Layout: Left->Right

Figure 8. Complete DAG

The full code of the DAG can be found in our github repo along with all the Lambda functions.

Lambda Function: Processing Data from S3 to DynamoDb

After the DAG stored the scraped data in csv format in the S3-bucket called “csvs3db”, we created a Lambda function titled “csvs3db” to process the data from S3 and store it in DynamoDB in a table called “NUM_data”.



Objects (2)

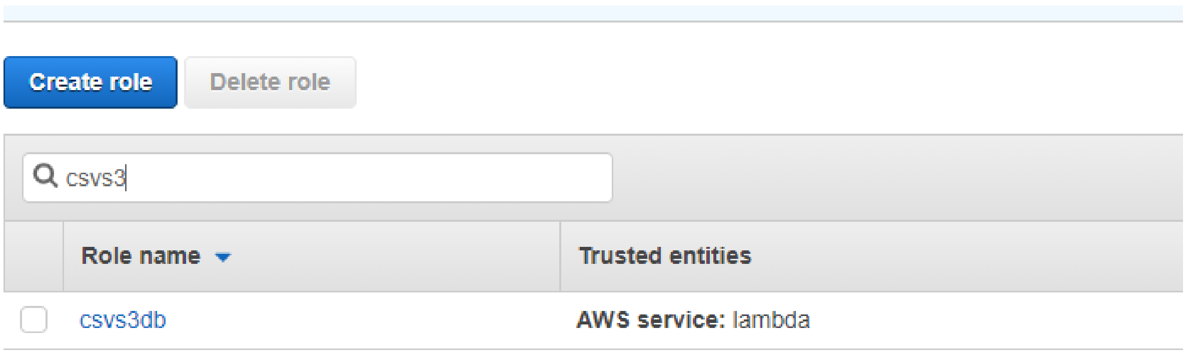
Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
exact_merge_test2.csv	csv	March 9, 2021, 11:46:52 (UTC+00:00)	326.0 KB	Standard
final_data.csv	csv	March 15, 2021, 21:47:30 (UTC+00:00)	8.5 KB	Standard

Figure 9. Store File in a S3 Bucket

We chose to use Lambda for this task in order to automate and easen the process. In fact, Lambda can manage the transfer of data from S3 to Dynamodb with just a predefined IAM role and trigger. It also allows the process to run continuously whereby each time a new file is added to S3 bucket, Lambda function will be triggered on its own to perform the processing.

Before setting up the Lambda function, an IAM role was created with pre-defined connections to three main policies that give full access to S3, DynamoDB, and AWS CloudWatch Logs.



Role name	Trusted entities
csvs3db	AWS service: lambda

Figure 10. IAM role

Then, we created the Lambda function and we linked it to the previous mentioned IAM role. We also added a S3 trigger for the function, with a suffix (.csv) which ensures that each time a csv file is added to the csvs3db bucket, the function will be triggered to process data automatically from the bucket to the Dynamo database.

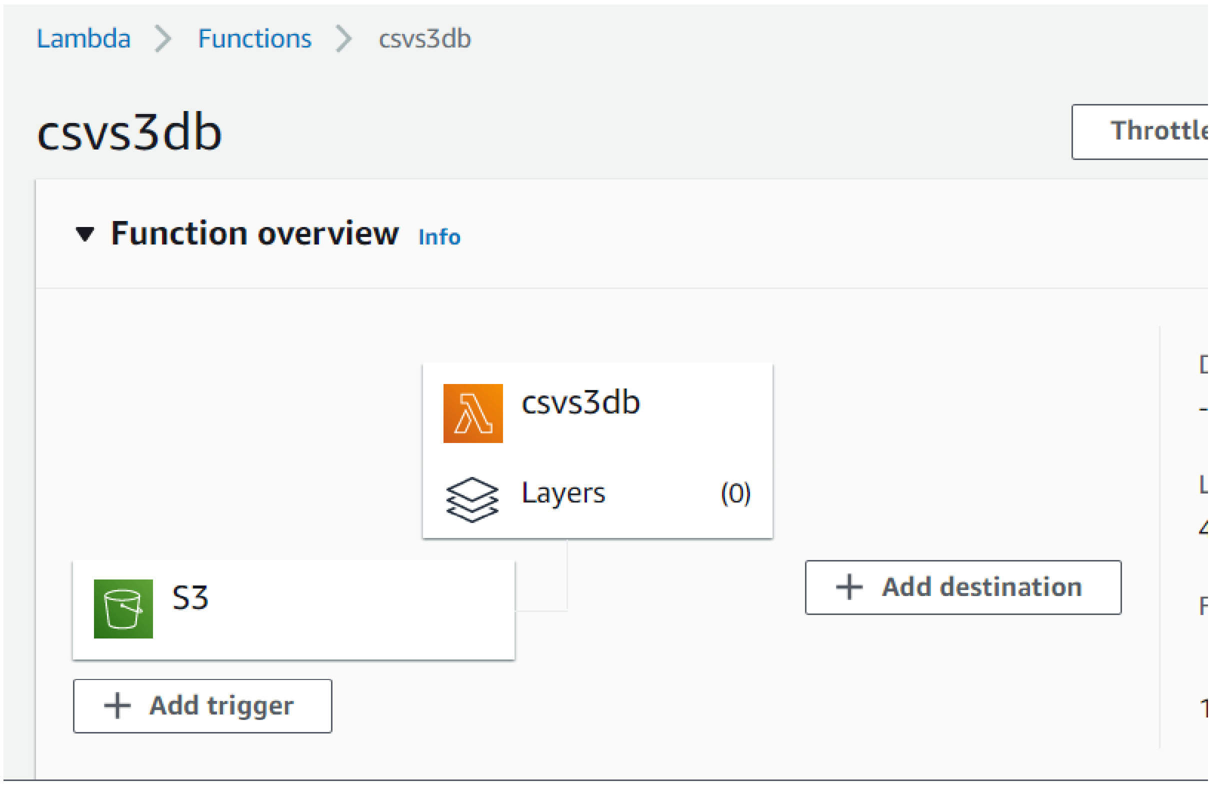
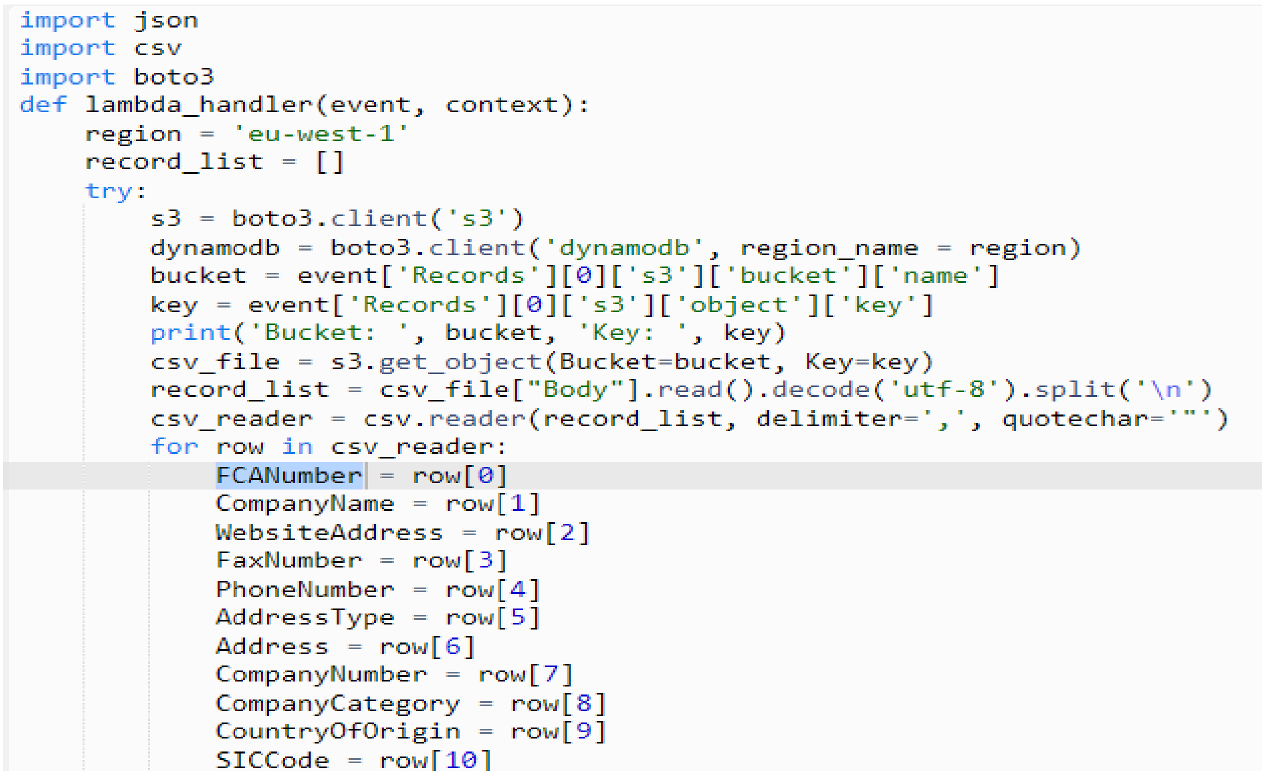
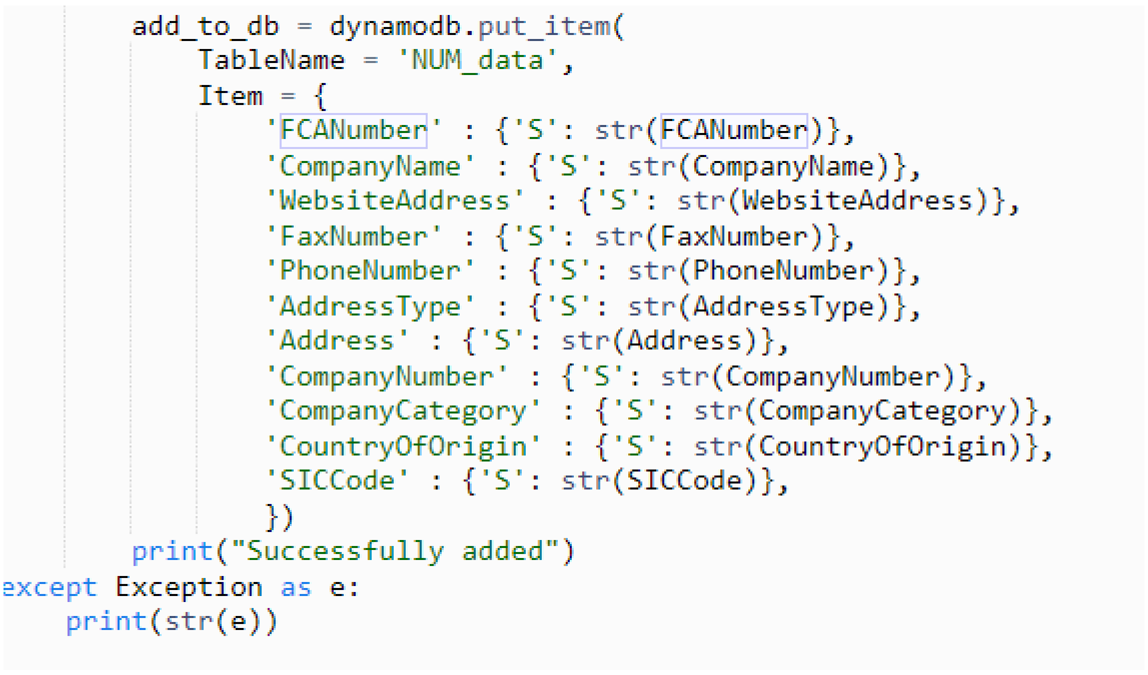


Figure 11. Lambda Function with S3 Trigger

We wrote up the function’s code which decodes the file and splits the rows. The csv-reader method is used to read the csv file and eliminate the commas between data points. Then, rows are iterated over in a for loop to specify the column of each attribute, in order to be added later to the DynamoDB using the add_to_db method. The code for this function is available at our github (https://github.com/Lukebotbol/Data_Engineering_pipeline).



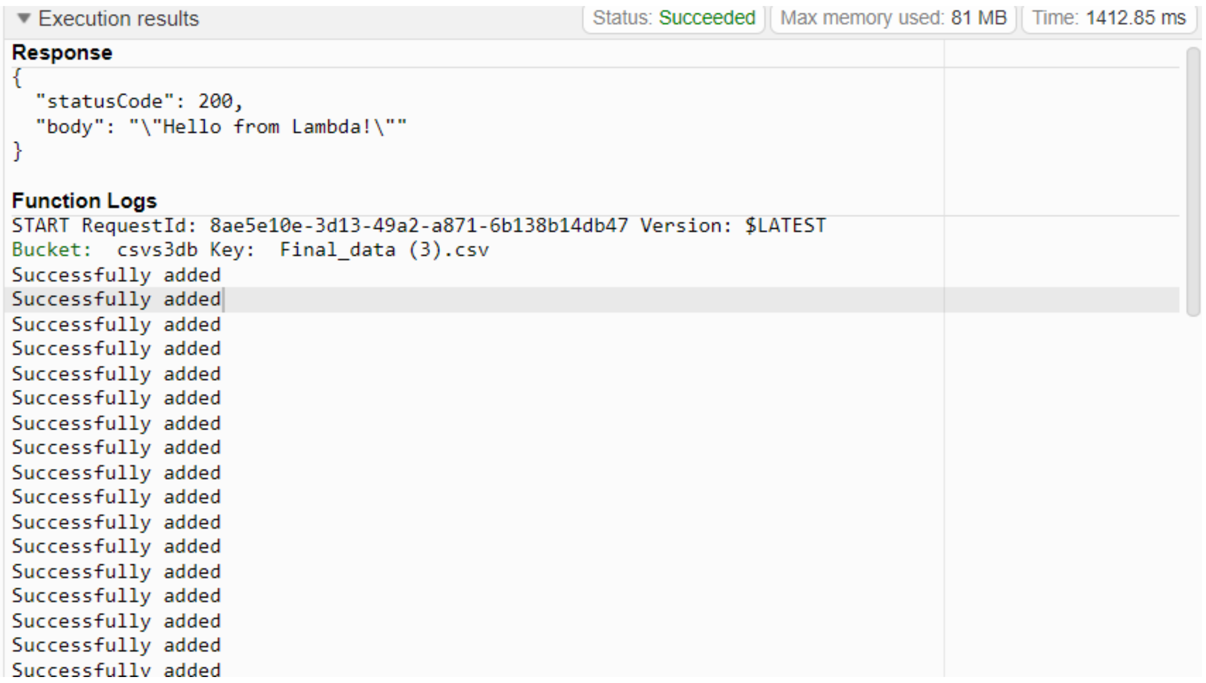
```
import json
import csv
import boto3
def lambda_handler(event, context):
    region = 'eu-west-1'
    record_list = []
    try:
        s3 = boto3.client('s3')
        dynamodb = boto3.client('dynamodb', region_name = region)
        bucket = event['Records'][0]['s3']['bucket']['name']
        key = event['Records'][0]['s3']['object']['key']
        print('Bucket: ', bucket, 'Key: ', key)
        csv_file = s3.get_object(Bucket=bucket, Key=key)
        record_list = csv_file["Body"].read().decode('utf-8').split('\n')
        csv_reader = csv.reader(record_list, delimiter=',', quotechar='"')
        for row in csv_reader:
            FCANumber = row[0]
            CompanyName = row[1]
            WebsiteAddress = row[2]
            FaxNumber = row[3]
            PhoneNumber = row[4]
            AddressType = row[5]
            Address = row[6]
            CompanyNumber = row[7]
            CompanyCategory = row[8]
            CountryOfOrigin = row[9]
            SICCode = row[10]
```



```
        add_to_db = dynamodb.put_item(
            TableName = 'NUM_data',
            Item = {
                'FCANumber' : {'S': str(FCANumber)},
                'CompanyName' : {'S': str(CompanyName)},
                'WebsiteAddress' : {'S': str(WebsiteAddress)},
                'FaxNumber' : {'S': str(FaxNumber)},
                'PhoneNumber' : {'S': str(PhoneNumber)},
                'AddressType' : {'S': str(AddressType)},
                'Address' : {'S': str(Address)},
                'CompanyNumber' : {'S': str(CompanyNumber)},
                'CompanyCategory' : {'S': str(CompanyCategory)},
                'CountryOfOrigin' : {'S': str(CountryOfOrigin)},
                'SICCode' : {'S': str(SICCode)},
                })
        print("Successfully added")
except Exception as e:
    print(str(e))
```

Figure 12. Lambda Function Code

Additionally, we tested the function to investigate execution errors. The function was deployed and data was successfully processed.

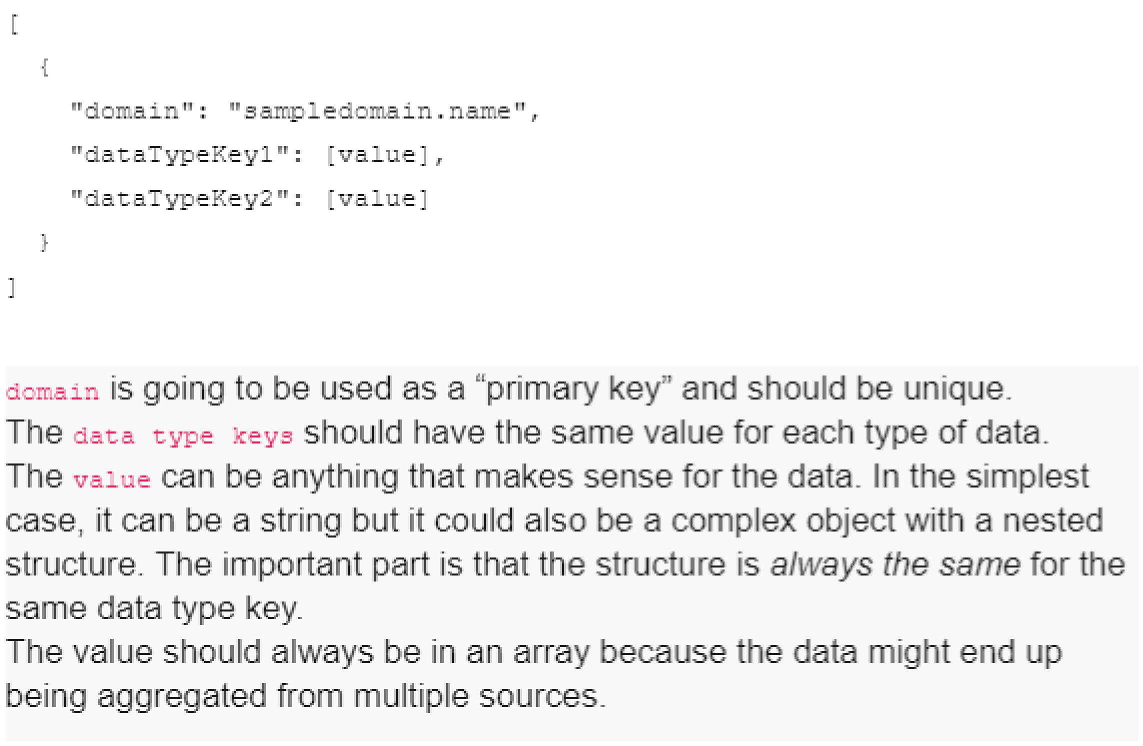


```
{
  "statusCode": 200,
  "body": "\"Hello from Lambda!\""
}
```

```
START RequestId: 8ae5e10e-3d13-49a2-a871-6b138b14db47 Version: $LATEST
Bucket:  csvs3db Key:  Final_data (3).csv
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
Successfully added
```

Figure 14. Execution Results

We chose DynamoDB due to its key-value setup. This setup matches the required schema outlined by NUM where the primary key is the domain name and secondary keys are other data values.



```
[
  {
    "domain": "sampledomain.name",
    "dataTypeKey1": [value],
    "dataTypeKey2": [value]
  }
]
```

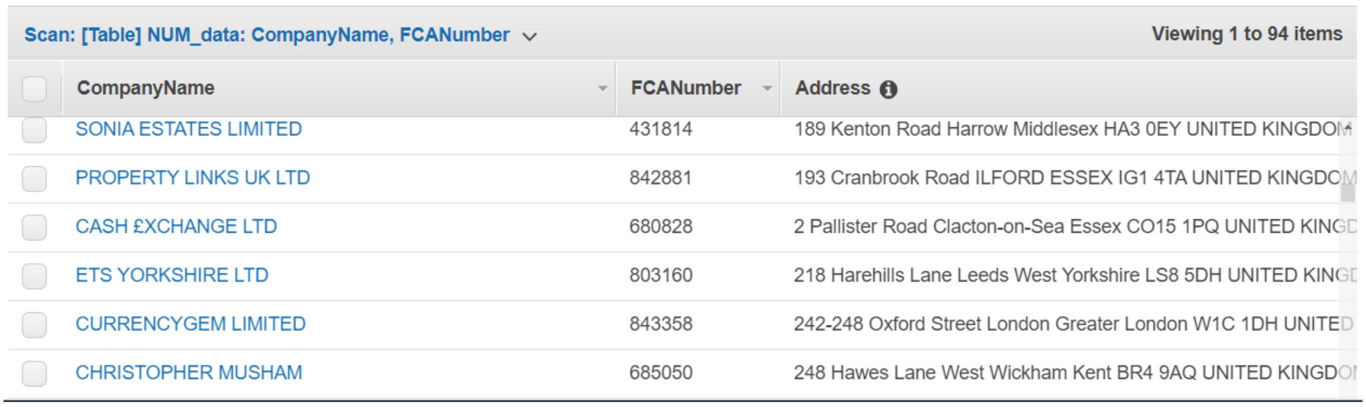
`domain` is going to be used as a “primary key” and should be unique.
The `data type keys` should have the same value for each type of data.
The `value` can be anything that makes sense for the data. In the simplest case, it can be a string but it could also be a complex object with a nested structure. The important part is that the structure is *always the same* for the same data type key.
The value should always be in an array because the data might end up being aggregated from multiple sources.

Figure 15. Data Format - NUM Company

However, we weren't able to utilise the domain name as a primary key because it is not unique (this will be discussed further below).

Another reason behind selecting DynamoDB is because we have a non-relational data type which requires NoSQL database tools. Also, DynamoDB is an AWS service which can be easily integrated with the other AWS tools used in the pipeline.

As a result, the data was successfully loaded to the NUM_data table in DynamoDB. The table was created with a primary key: "CompanyName", and a secondary key: "FCANumber". The table has 11 columns which display all the data attributes we have in the csv file.

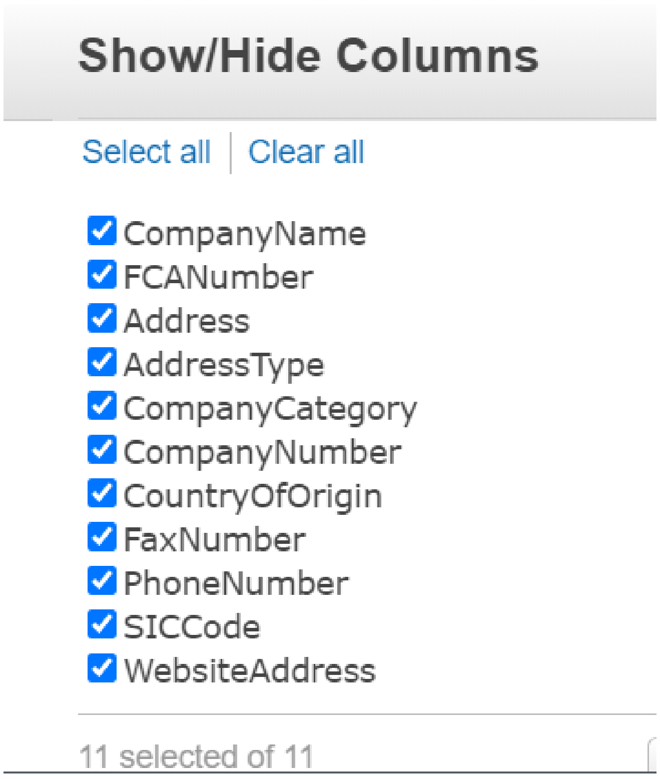


Scan: [Table] NUM_data: CompanyName, FCANumber

Viewing 1 to 94 items

CompanyName	FCANumber	Address
SONIA ESTATES LIMITED	431814	189 Kenton Road Harrow Middlesex HA3 0EY UNITED KINGDOM
PROPERTY LINKS UK LTD	842881	193 Cranbrook Road ILFORD ESSEX IG1 4TA UNITED KINGDOM
CASH £XCHANGE LTD	680828	2 Pallister Road Clacton-on-Sea Essex CO15 1PQ UNITED KING
ETS YORKSHIRE LTD	803160	218 Harehills Lane Leeds West Yorkshire LS8 5DH UNITED KING
CURRENCYGEM LIMITED	843358	242-248 Oxford Street London Greater London W1C 1DH UNITED
CHRISTOPHER MUSHAM	685050	248 Hawes Lane West Wickham Kent BR4 9AQ UNITED KINGDO

Figure 16. Data in NUM_data Table - DynamoDB



Show/Hide Columns

Select all | Clear all

- ✓ CompanyName
- ✓ FCANumber
- ✓ Address
- ✓ AddressType
- ✓ CompanyCategory
- ✓ CompanyNumber
- ✓ CountryOfOrigin
- ✓ FaxNumber
- ✓ PhoneNumber
- ✓ SICCode
- ✓ WebsiteAddress

11 selected of 11

Figure 17. Columns- NUM _data Table - DynamoDB

Issues and Limitations

1- Data Format: Before loading the data as a csv in the S3 bucket, we tried via json format. However, the Lambda function didn't deploy successfully and we encountered decoding and validation exception errors in the log streams of the Lambda function (s3_json-dynamodb_SA) in the CloudWatch section.

2- Unique Key: The company advised that data should be presented in a schema with a domain name as a primary key alongside the other attributes. However, this wasn't applicable in DynamoDB because the domain name wasn't unique among the data points. In other words, many companies have the same domain name. We managed to process the data through other unique keys, namely company number as a primary key and company name as a sort key.

Data Size/Complexity

Throughout the project, we faced several constraints due to the data size of the Companies House file. In order to provide a scalable solution to process the data we used Lambda functions. We managed to process the entire Companies House file with Lambda, meaning there couldn't be any scalability issues in the future. Furthermore, we used Airflow to extract data from the FCA API. For the purpose of the project we only extracted a small sample with Airflow. However, it would be possible to scale with a much larger sample using Celery as it provides a way to scale out the number of workers.

Evaluation of the project Methodology

After a project review meeting with NUM, we agreed that our final methodology of web scraping using the FCA API was acceptable given the restricted timeline. Throughout the process, we were able to understand the reality behind data collection in a commercial environment. We were initially optimistic in web scraping 6 million records from the Companies House csv. However, we realised that only 59 thousand companies were registered on the FCA website. Moreover, amongst the data within the FCA, only around 15% had domain names listed.

Although the journey to our final dataset may not have been optimal, this project has shown some of the hidden challenges that data engineers face in everyday tasks. Issues such as scalability, accessibility and reproducibility were just some of the challenges faced during this project.

A further discussion point with NUM was whether we could obtain FCA numbers via domain names. Since NUM has a pool of existing domain names, NUM could provide this resource for us to web scrape on individual websites. This could prove more effective as most FCA registered companies would have their FCA number presented at the bottom of the web page. However, due to time restriction of the project, we believed that this was not realistically feasible.

Other potential tools which could be utilized in this project are Docker and Apache Spark to build containers, process the big data across multiple workloads using PySpark, perform batch processing, and conduct SQL interactive queries. This exceeds the scope of this assignment but is a potential starting point for further analysis.

Project Management

Throughout the project, our group held weekly zoom meetings internally as well as with the NUM team. During our internal group meetings, we would discuss individual progress and decide on how to best split up tasks. During meetings with NUM, the team would help us with any decisions and guide us in the direction they deemed best for the final product. Further details regarding individual meetings both internally and with NUM are documented in the Appendix.

Conclusion

In the end, we successfully generated about 8 thousand data points that could be used by NUM's end users. Although our results may seem small when compared to NUM's existing data size, we as well as NUM believe this project has set a strong foundation for NUM to continue to diversify their range of data sources and data types they provide to customers and clients.

Appendix

Internal Meetings

8 Feb

- First meeting together as the group, discussing potential ideas from the proposal list
- Short listing number of projects, we decided to take on the NUM project from the proposal idea list
- Initiate conversation with Niall and NUM to begin the process

12/13 Feb

- After the first meeting with NUM, we discuss what particular area of data collection that we are interested in.
- NUM has an established system which takes contact details of the companies
- Whilst, we brainstorm potential areas to web scrape. We awaits for NUM's emails to get a further understanding of which particular area of data collection is successful and not successful

16 Feb

- After the meeting with NUM, we choose to gather company information as our project, trying to scrap domain names, FCA number and Companies House number
- Since we decided on Web scraping, we decide to split up amongst the group to familiarise with the web scraping
- Luke will set up an Faculty Environment
- Jason and Rosie - Company House web scraping
- Freddie, Luke and Sultan - FCA web scraping

25 Feb

- Day before meeting with NUM, we solidify our methodology
- We will use Companies House API to scrape information from the FCA API
- Still need to find the best way to connect companies house data with FCA data
- Template of the web scraping is located on Faculty, ready to showcase NUM

28 Feb

- Communication with NUM about the common crawl data, arranging a meeting with the Data Scientist at NUM
- Our best solution to connect Companies House data with FCA API would be using a CSV file provided by Companies House and directly use that information to feed into the FCA API. Freddie and Jason are in charge of the web scraping and also formatting the output
- Action required: begin to research Airflow and Lambda function to the data pipeline. Luke and Rosie are in charge of the Airflow whilst Sultan will focus on the Lambda Function and setting up DynamoDB

8 March

- Issues with the FCA API with too many companies not registered on FCA

- Only 59k is only registered with the FCA and we would need to scrap from 5 million to obtain a 59k data
- From the 5 million of data, when use SIC code (business category) to filter down businesses who are related to the financial industry. This reduces the dataset down to 1 million. Jason will be handling the filtering of the dataset
- Another Issues with FCA API is that whenever a research that has no result, the API will automatically recommend some similar reason, this creates a lot of duplicates. Freddie will be taking charge of this, whilst Jason will help out.
- Action required: to find a new way of scraping the FCA API and not use the 5 million data from the Companies House CSV
- Lambda function is now up and running to cover data into DynamoDB
- Action required: Setting up Airflow requires a lot of debugging, we therefore arranged a call with the TAs. Everyone needs to part take in the debugging.
- Action required: initial drafting and structure of the report and set up a google drive. Jason is responsible for structuring of the report

10 March

- New solution: using a CSV from the FCA website to feed into the FCA API as it has a more precise FCA number instead of search through by Companies' name
- We need to continue to Web scraping to obtain the final 8k of data from the FCA csv input into the API
- Further debugging require with the TA's with regards to the Airflow DAGs
- Luke had some issues with the DAG's, since we have done with our other parts, we all focused on debugging
- Continuation of the report writing

15 March

- Finally meeting with NUM
- Evaluation of the entire project
- Potential project scoping for the individual assignment

17 March

- Finalise the report
- Checking for mistakes

Meetings with NUM

12 Feb

- First meeting with NUM, joined by Niall and Walter. Introduced the NUM team
- We introduced ourselves, highlighted what we expect from the project,
- Elliot explained in detail what problem NUM are trying to solve
- Decided that we would write an email with the best possible meeting times
- Discussed different ways to collect data, whether from open source websites, crawling the web, web scraping websites, use GPS coordinates
- Niall discussed various approaches we could take, including using Snorkel to test how whether collected data was correct

- Elliot mentioned common crawl as possible solution

16 Feb

- Received an email with what data NUM are looking to collect
- Settled on the approach of using APIs from open source databases such as Open Corporate and Companies House instead of web scraping
- Issue with many databases is that they cost something
- Asked for a template of how they want their data as an end format
- Settled on using Snorkel to test collected data

26 Feb

- Explained our approach - to use the FCA API combined with the Companies House API and merge data
- Elliot asked us to write an email to FCA to request information, might say yes
- Discussed where else we could get domain names from
- Decided to look into common crawl data

4 March

- Explained new approach of using a Companies House csv instead of using its API
- Explained our trouble of merging the two datasets without common feature
- Showed Quentin possible Javascript on github as a solution, was skeptical
- Discussed approach which was too inefficient
- Elliot suggested using FCA download data instead of Companies House names

8 March

- Discussed out extension and possible work on individual assignments
- Discussed how to get our data into json format with domain name as the only key - issue with multiple companies having same domain name

15 March

- Presented our final csv
- Discussed what we liked and disliked about working with NUM
- Left with agreeing to reach out when we have more information regarding the individual assignment