ADVANCES IN DATA SCIENCE/ARCHITECTURE

ASSIGNMENT-1

REPORT

Submitted By:

MEGHA SINGH

SNEHA MALSHETTI

Table of Contents

Overview **3**

Some of the Installations and softwares used for creating the project 4

Objective Flow of the Project **5**

Flowchart of Processing of Data Analysis 6

Data Ingestion **6**

Flow for the Data Ingestion: 7

Descriptive Steps and script for data Ingestion 8

Exploratory Data Analysis **11**

Steps for Data cleansing and EDA 15

Data Wrangling **19**

Steps for Evaluation and data Analysis 20

Automation of dataset on cloud **24**

Docker image 24

Celery pipeline using RabbitMQ 26

Batch scheduling on Amazon AWS with steps 28

Citations **4**

1. Overview

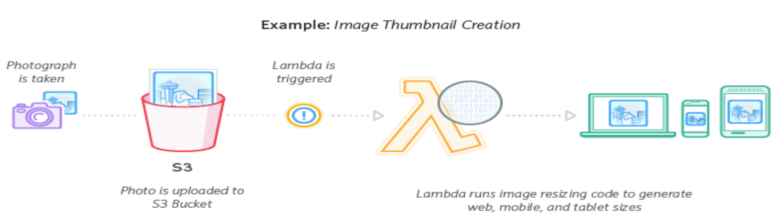
There is a huge pool of data around the world. Tera bytes and even picobytes of data are collected everyday by companies who need this data to process it and analyse it to make future predictions on this data. Not all the data that comes in is properly formatted. In fact, some data is so poorly formatted it becomes more of a challenge to read the data into a csv file or any other system-readable or human-readable format.

This data needs to be pulled in from more than one systems in scenarios where there aren't many API's supporting this import. Hence we use languages like R and Python to convert this data pushed from various systems using standard file formats like JSON and csv to read and manipulate data the way we want to visualize it.

The data that we spend so much space, time and money for, is then used to visualize the trends using various visualization graphs and table format. But for this the data needs to be cleaned first. The processes involved are Feature Selections and Data Wrangling. Feature Selection involves selecting what features you want to keep in the data set you have, with a justifiable reasoning of why this set of data has to be kept and why the other set has to be discard. The reasoning depends completely on the kind of analytics you want to perform. After Feature Selection we perform Data Wrangling which keeps only what is required and manipulates the rest of the data as desired.

And then comes data visualization to see if we missed and missing values or outliers in the data wrangling and maybe perform it again and then comes the part where we automate this entire process using pipelines like (Make, Celery, Airflow) so we can scale this process for larger data sets and finally schedule it so it doesn't need any human intervention to start the process everyday, unless there is an error.

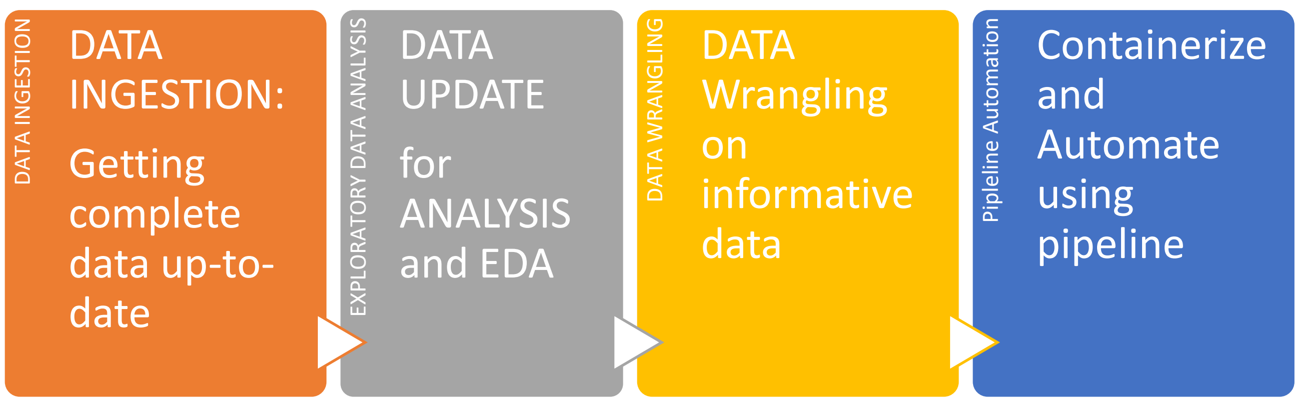
1.1. Some of the Installations and softwares used for creating the project:



2. Objective Flow of the Project

1. Language Used: Python
2. Data Ingestion: Jupyter NoteBook, Boto3 connection, Amazon S3 bucket
3. EDA: Jupyter Notebook
4. Data Wrangling:
5. Pipeline: Celery, RabbitMQ server, Amazon Batch, Job Functions.
6. AWS BATCH: Batch, Lambda function, SNS

2.1 Flowchart of Processing of Data Analysis



3. Data Ingestion

Flow for the Data Ingestion:

Install all Packages Anaconda, Python, Jupyter Notebook, Shell, AWS account setup, Amazon S3.

Code to Fetch the Real-time data from API using the config files.

- Python script to read a config.json file and source the entire dataset from the FTP link.

Code to Downlaod the data set on local as well On Amazon S3 bucket.

- Python script using boto3 connection create a Bucket on Amazon S3 called Team<No><State>Assignment1” and checks if the bucket already exists.

Update the data set for everyday everytime the code is invoked, on both S3 and local

- Run the code and should fetch the CSV from Api and upload directly to the S3 bucket by creating a new file with specific required name
- Also, If run another day, then it should create a new file. For example, if I run this code tomorrow,it should download the new file and create NJ_12062017_WBAN_5902.csv.

Add the updated data in the CSV based on Station ID and date using the config json file.

- If the code is run after few days it will fetch all the data set and take the existing csv(old) and the new csv and merge into one single data set as a whole,
- Also code shouldn’t create another file for the same day.

Create the log for Everytime the Bucket is created, file is downloaded, and error for already Existing file

- Script to creat a log file in format **ddmmyyMMSS.log** everytime the file is uploaded or every instance occurred on S3 bucket, includint the error message of already existing file or already downloaded file.

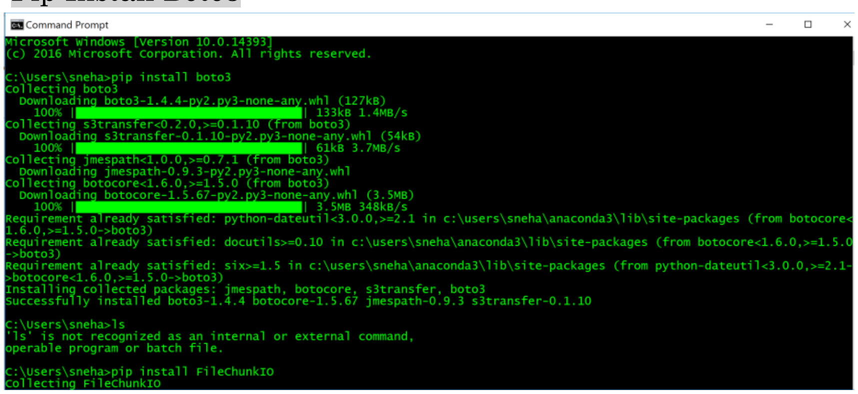
Create a docker image for this complete script as: DATA INGESTION which can be invoked on running image by using run.sh shell script.

- Create a Docker file with setting the entrypoint for run.sh which will invoke the dataingestion.py script file and execute the whole process, only by running the docker image

1. For Data Ingestion, we had to first of all install Anaconda and run the code in jupyter notebook environment which supports Python 3.Python 3 howerver is not compatible with many tools like docker and Xamp.
2. The latest and most compatible versions of Python are 2.6.x nd 2.7.x. This was one issue we faced during out assignment.
3. Another compatibility issue was with python shell and python notebook and ipython. All3 have a few syntax changes and hence this added to the version compatibility issues.

   However, having said that, the most seemingly easy to work with tool was the jupyter notebook because it had most of the libraries installed in the environment so we didn't have to install pandas or numpy libraries or even matplotlib kind of libraries.

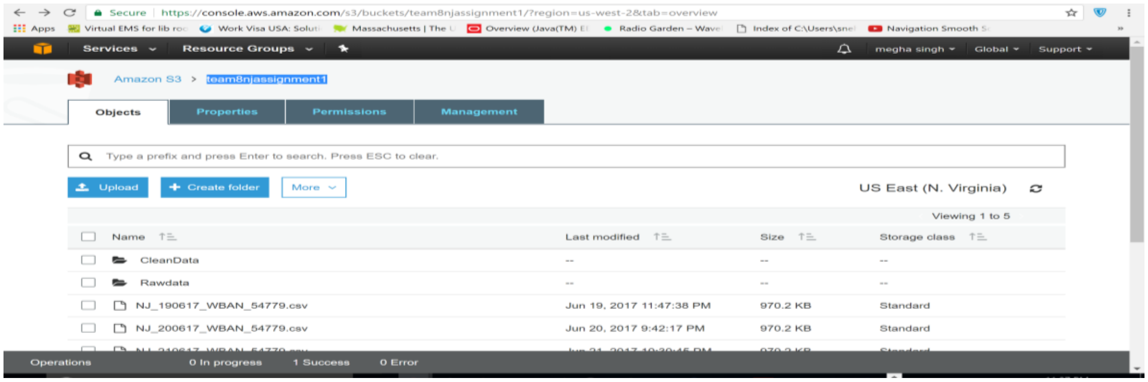
4. To connect with S3 we had to install Boto and Boto3 since boto doesn't work in python3 invironment and hence a lot of issues were faced.
   For creating the connection with S3 bucket dynamically using python script we used Boto3
5. By installing in the environment path Boto Installation :
   Pip Install Boto3



```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\sneha>pip install boto3
Collecting boto3
  Downloading boto3-1.4.4-py2.py3-none-any.whl (127kB)
    100% |████████████████████████████████| 133kB 1.4MB/s
Collecting s3transfer<0.2.0,>=0.1.10 (from boto3)
  Downloading s3transfer-0.1.10-py2.py3-none-any.whl (54kB)
    100% |████████████████████████████████| 61kB 3.7MB/s
Collecting jmespath<1.0.0,>=0.7.1 (from boto3)
  Downloading jmespath-0.9.3-py2.py3-none-any.whl
Collecting botocore<1.6.0,>=1.5.0 (from boto3)
  Downloading botocore-1.5.67-py2.py3-none-any.whl (3.5MB)
    100% |████████████████████████████████| 3.5MB 348kB/s
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in c:\users\sneha\anaconda3\lib\site-packages (from botocore<
1.6.0,>=1.5.0->boto3)
Requirement already satisfied: docutils>=0.10 in c:\users\sneha\anaconda3\lib\site-packages (from botocore<1.6.0,>=1.5.0
->boto3)
Requirement already satisfied: six>=1.5 in c:\users\sneha\anaconda3\lib\site-packages (from python-dateutil<3.0.0,>=2.1-
>botocore<1.6.0,>=1.5.0->boto3)
Installing collected packages: jmespath, botocore, s3transfer, boto3
Successfully installed boto3-1.4.4 botocore-1.5.67 jmespath-0.9.3 s3transfer-0.1.10

C:\Users\sneha>ls
'ls' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\sneha>pip install FileChunkIO
Collecting FileChunkIO
```

6. A lot of times the S3 bucket removes access from buckets and hence there is a lot of trouble with that as well. It seems to change permissions without notice, which is not easy to catch and the code doesn't run since all the data that comes in is from the AWS S3 Bucket. In S3 Bucket we cannot duplicate names so incase there is a name already on S3, we need to
   - create a different name this is to keep all bucket names on the cloud unique.

7. The S3 Bucket we created : team8njassignment1
   Also, check if the bucket exist it will not be created on S3.
   S3Bucket Created :



8. Once the bucket is created using the script, Now we have to fetch the Real time data set from LCD weather API and Upload it on the S3 bucke. For this we will will have to first dynamically create a file name pattern using the config json file and provide the Api Link in it for fetching the up-todate data everytime the script is invoked.
9. For fetching the data we have used two Cofig json files:
   a. Initconfig.json : this will fetch all the data set from the very beginning till the 2017 (since complete data set isn't allowed by the API to be fetched at once)
   b. Config.json : this will fetch the all the remaining data set after the previous one till today.
10. Using the config files now we have fetched the data from the api and uploaded on the s3:
    Here is the small piece of code explaining the same:
    Using the initconfig.json
    a) We are pulling the starting point data set from the FTP link from 2005 to 2014(10 years)
    b) And then putting this fetched dataset into a DataFrame using pandas.

```
if count ==0:
    #init File merge
    initfile1 = pd.read_csv(linkpt1)
    print("Shape of 1st file is :",initfile1.shape)

    initfile2 = pd.read_csv(linkpt2)
    print("Shape of 1st file is :",initfile2.shape)

    initfullmerge=pd.concat([initfile1,initfile2], axis=0).drop_duplicates().reset_index(drop=True)
    print("Shape of merged file is :",initfullmerge.shape)
    initfullmerge2=initfullmerge.drop_duplicates(['DATE'], keep='first')
    print("Shape of merged n duplicated removed file is :",initfullmerge2.shape)

#download on local directory init data
    initialfilename='initfile.csv'
    initfullmerge2.to_csv(initialfilename,sep=',', index=False)
    # log file save event
    my_logger.info("A csv file named 'initfile' was saved in the local repository at:"
+time.strftime("%d%m%Y%H%M%S"))
```

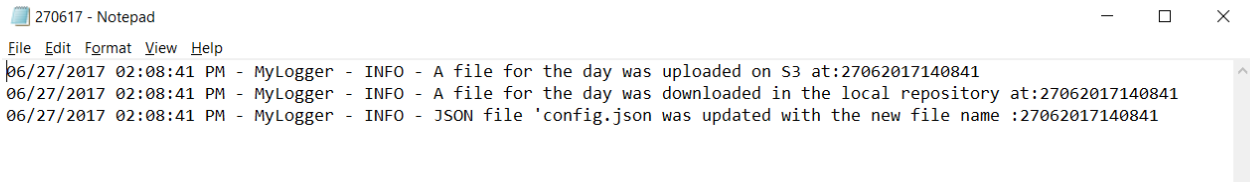
This initfile is created on the local user directory, so everytime the code is invoked it gets updated or being checked the last downloaded date ,then download the further updates and dynamically used to merge with the named as

```
270617.log
<class 'boto.s3.bucket.Bucket'>
b'CleanData/nj_230617_WBAN_54779_clean.csv'
b'NJ_190 17_WBAN_54779.csv'
b'NJ_200617_WBAN_54779.csv'
b'NJ_210617_WBAN_54779.csv'
b'Rawdata/NJ_230617_WBAN_54779.csv'
b'Rawdata/NJ_240617_WBAN_54779.csv'
b'Rawdata/NJ_270617_WBAN_54779.csv'
Number of files in S3 bucket  7
NJ_270617_WBAN_54779.csv
possible None
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:2717: DtypeWarning: Columns (9,10,11,12,13,14,15,1
6,20,23,24,25,26,27,28,29,33,34,44,45,46,47,71,72,73,84,85,86,87) have mixed types. Specify dtype option on import or set low_m
emory=False.
  interactivity=interactivity, compiler=compiler, result=result)
Shape of 1st file is : (90739, 90)
initfile.csv
```

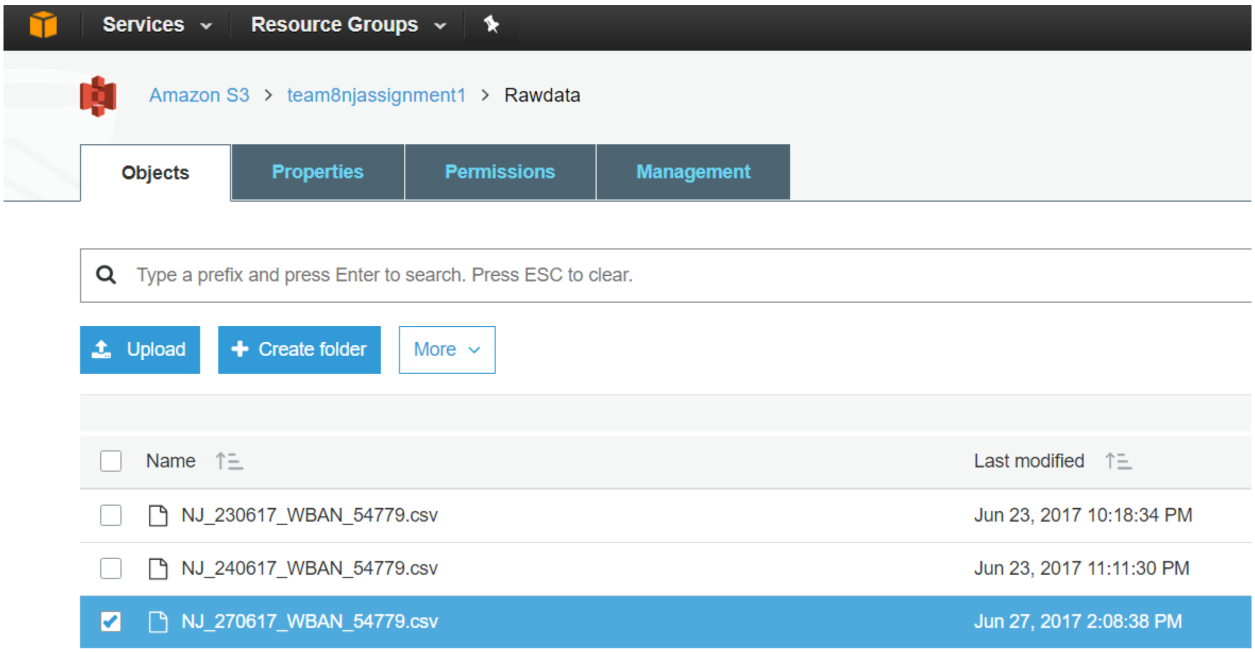
Script merge the old data and new data and upload to S3 bucket and record the event in Log file:

```
#Merge previous and today

        dailymerge=pd.concat([prevdata,todaylink],
axis=0).drop_duplicates().reset_index(drop=True)
        print("Shape of merged file is :",dailymerge.shape)
        dailymerge2=dailymerge.drop_duplicates(['DATE'], keep='first')
        print("Shape of merged n duplicated removed file is :",dailymerge2.shape)

    #upload the file to S3

        k = Key(b)
        k.key = "Rawdata/"+fname
        k.content_type = r.headers['content-type']
        k.set_contents_from_string(r.content)
        print('successfully uploaded to s3')
    #log upload event
        my_logger.info("A file for the day was uploaded on S3 at:"
+time.strftime("%d%m%Y%H%M%S"))
 #download current day file on local as well.
        dailymerge2.to_csv(fname,sep=',')
  #log upload event
        my_logger.info("A file for the day was downloaded in the local repository at:"
+time.strftime("%d%m%Y%H%M%S"))
```

Also the log generated for the same:



This file is merged and complete data set is uploaded to the RAWDATA folder inside the bucket on S3 and the event is recorded on the log file



4. Exploratory Data Analysis :

The following are a few plots that we thought would be useful in summarizing the entire year's Temperature or humidity and other factors. This also helps in Data Analysis. And to make future predictions based on models. The visualization of data also gives us an insight of whether the data has gone bad with a few outliers and missing values or strings in the numeric fields.

Hence it is necessary that we carry out data cleaning activities in the raw data with techniques like data wrangling using R or Python.

EDA involves the cleansing of the data and changing into a meaningful values so we can explore data for perdition or fetch analytical values from it.

Remove redundant or no value columns: For same, First we need to select only the valuable columns and remove the columns which have no values or data lesser than 5%.

Change the datatype to meaningful for analysis After changing and filling the missing data change the data type of column to float, integer or object or Boolean etc.

Fill missing values Nan with mean or Flag: Then we replace all the values to Nan and covert the NaN columns in dataframe into the either flag or mean value of column so it doesn't changes overall values.

For the Exploratory Data Analysis we used Jupyter Notebook:

1. Create a Jupyter notebook to perform EDA on the downloaded data(<state_ddmmyy_stationid>.csv)
2. Using the config json, making the dataframe of the csv dataset
   rawdata= pd.read_csv("NJ_270617_WBAN_54779.csv", parse_dates=['DATE'])
3. Selecting the coloumns with only meaningful data columns

Selecting Important Data columns from File

```
2]:
df= rawdata[['DATE','HOURLYDRYBULBTEMPF',
                                   'HOURLYDRYBULBTEMPC',
                                   'HOURLYWETBULBTEMPF',
                                   'HOURLYWETBULBTEMPC',
                                   'HOURLYDewPointTempF',
                                   'HOURLYDewPointTempC',
                                   'HOURLYRelativeHumidity',
                                   'HOURLYWindSpeed',
                                   'HOURLYWindDirection',
                                   'HOURLYWindGustSpeed',
                                   'HOURLYStationPressure',
                                   'HOURLYPressureTendency',
                                   'HOURLYSeaLevelPressure',
                                   'HOURLYPrecip',
                                   'HOURLYAltimeterSetting',
                                   'REPORTTPYE',
                                   'DAILYSunrise',
                                   'DAILYSunset']]
```

4. Removing the Nan Values and Selecting the dataframe into nonNan values

```
noNaNdf= df.fillna(scipy.mean(df))
print (noNaNdf.head(3))
print(noNaNdf.dtypes)

noNaNdf.head(5)
```

```
                 DATE  HOURLYDRYBULBTEMPF  HOURLYDRYBULBTEMPC  \
0 2017-01-01 00:54:00                  39                 3.9
1 2017-01-01 01:54:00                  39                 3.9
2 2017-01-01 02:54:00                  38                 3.3

   HOURLYWETBULBTEMPF  HOURLYWETBULBTEMPC  HOURLYDewPointTempF  \
0                34.0                 1.0                 24.0
1                34.0                 1.0                 24.0
2                33.0                 0.6                 24.0

   HOURLYDewPointTempC  HOURLYRelativeHumidity  HOURLYWindSpeed
0                 -4.4                    55.0              5.0
```

5. Now Seggregating Data into months date and year by creating new columns
And relplacing Nan Values with the mean of the column so it it not effect the overall values.
#Removing the NaN values
noNaNdf= df.fillna(scipy.mean(df))

```
HOURLYDewPointTempF           float64
HOURLYDewPointTempC           float64
HOURLYRelativeHumidity        float64
HOURLYWindSpeed               float64
HOURLYWindDirection            object
HOURLYWindGustSpeed           float64
HOURLYStationPressure          object
HOURLYPressureTendency        float64
HOURLYSeaLevelPressure         object
HOURLYPrecip                   object
HOURLYAltimeterSetting         object
REPORTTPYE                     object
DAILYSunrise                    int64
DAILYSunset                     int64
new_date                       object
new_time                       object
year                            int64
month                           int64
day                             int64
dtype: object
```

6. Now by using the matplotlib we analyse the Maximum and minimum temperature over the years
First define a data frame for anaysis of temperature for this we select year vs Temp(hourly dry bulb temp) to provide actual temperature range over the years
#define a dataframe
dfYearMaxMinTemp= noNaNdf[["HOURLYDRYBULBTEMPC", "year"]]
7. Now replace the nonNAn values by mean values

#Replace NaN by mean Value
noNaNdfYearMaxMinTemp= dfYearMaxMinTemp.fillna(scipy.mean(dfYearMaxMinTemp))
print (noNaNdfYearMaxMinTemp.head(3))

8. For comparison change the dataset into integer values and put it a new dataframe
noNaNdfYearMaxMinTemp["year"]=noNaNdfYearMaxMinTemp["year"].astype(int)
print(noNaNdfYearMaxMinTemp["HOURLYDRYBULBTEMPC"].dtype)

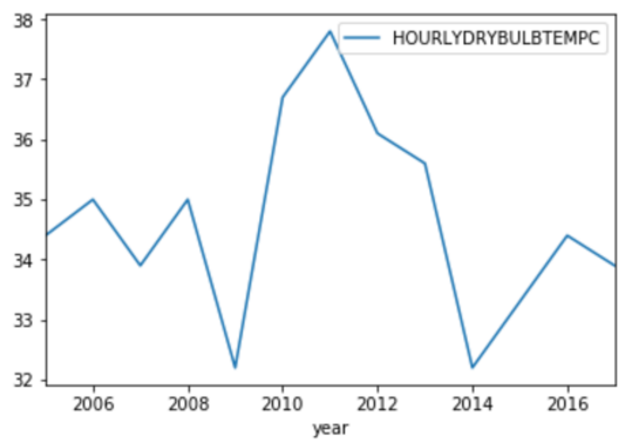
9. Change the data frame into numeric drybulb temperature
noNaNdfYearMaxMinTemp["HOURLYDRYBULBTEMPC"]=noNaNdfYearMaxMinTemp["HOURLYDRYBULBTEMPC"].convert_objects(convert_numeric=True)
print (noNaNdfYearMaxMinTemp["year"].dtype)

10. Now getting the maximun and minimum values for the dataset over the years

For maximum temperature:

maxtempdata=
noNaNdfYearMaxMinTemp.groupby('year')['HOURLYDRYBULBTEMPC'].max().reset_index()
print(maxtempdata)

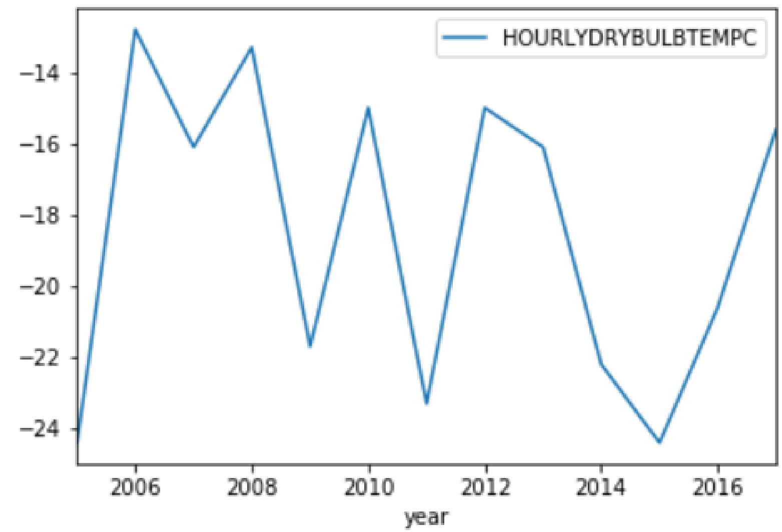
```
    year  HOURLYDRYBULBTEMPC
0   2005                34.4
1   2006                35.0
2   2007                33.9
3   2008                35.0
4   2009                32.2
5   2010                36.7
6   2011                37.8
7   2012                36.1
8   2013                35.6
9   2014                32.2
10  2015                33.3
11  2016                34.4
12  2017                33.9
```



Minimum temperature dataframe over years:

mintempdata=
(noNaNdfYearMaxMinTemp.groupby('year')['HOURLYDRYBULBTEMPC']).min().reset_index()
print(mintempdata)

```
    year  HOURLYDRYBULBTEMPC
0   2005               -24.4
1   2006               -12.8
2   2007               -16.1
3   2008               -13.3
4   2009               -21.7
5   2010               -15.0
6   2011               -23.3
7   2012               -15.0
8   2013               -16.1
9   2014               -22.2
10  2015               -24.4
11  2016               -20.6
12  2017               -15.6
```

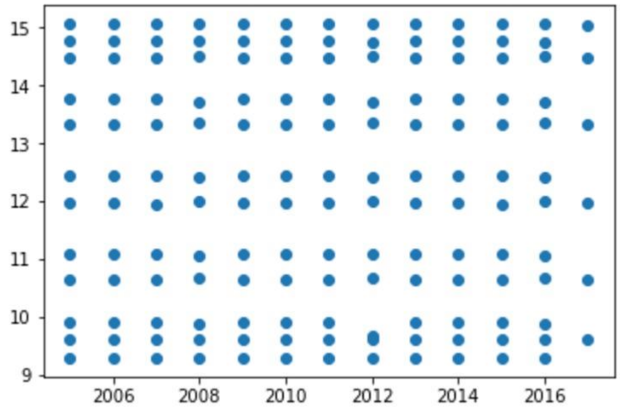


11. Now we can create a new analyitical informative column from the given data set:
    By using the sunrise time and sun set time for everyday we can create a new column for daylength and analyses how the length of the day is varied over an year and when is the daylight savings can be started in which month of the year.
    For this we used Lambda funtion

Plot for length of day

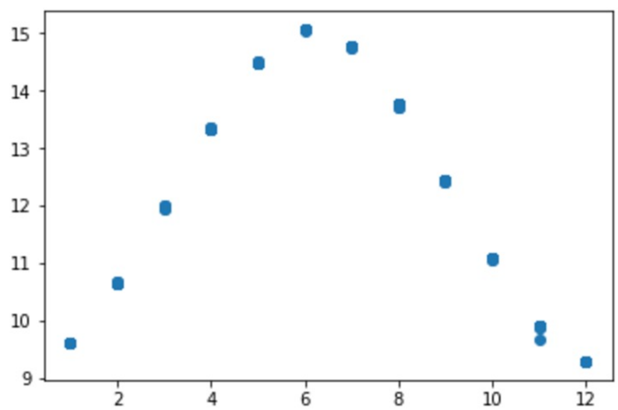
```
In [7]: time_func = lambda x: pd.Timestamp(pd.to_datetime(x, format = '%H%M'))

dataforsunrise=noNaNdf['DAILYSunrise'].apply(time_func)
dataforsunset=noNaNdf['DAILYSunset'].apply(time_func)
daylenght=(dataforsunset-dataforsunrise).astype('timedelta64[m]')/60

print(dataforsunset.head(3))
print(dataforsunrise.head(3))
print(daylenght.head(3))

# daylen1= (daylenght.groupby('year', 'month')['hour']).mean().reset_index()
# print(daylen1)
daylendf=noNaNdf[['year', 'month']]
daylendf['daylight']=daylenght.abs()
print (daylendf.head(5))


mintempdata= (daylendf.groupby(['year', 'month'])['daylight']).mean().reset_index()
print(mintempdata)



plt.scatter(mintempdata['year'], mintempdata['daylight'])
plt.show()
```

12. From this plot we can analysis that over the years the length of the day From January to December

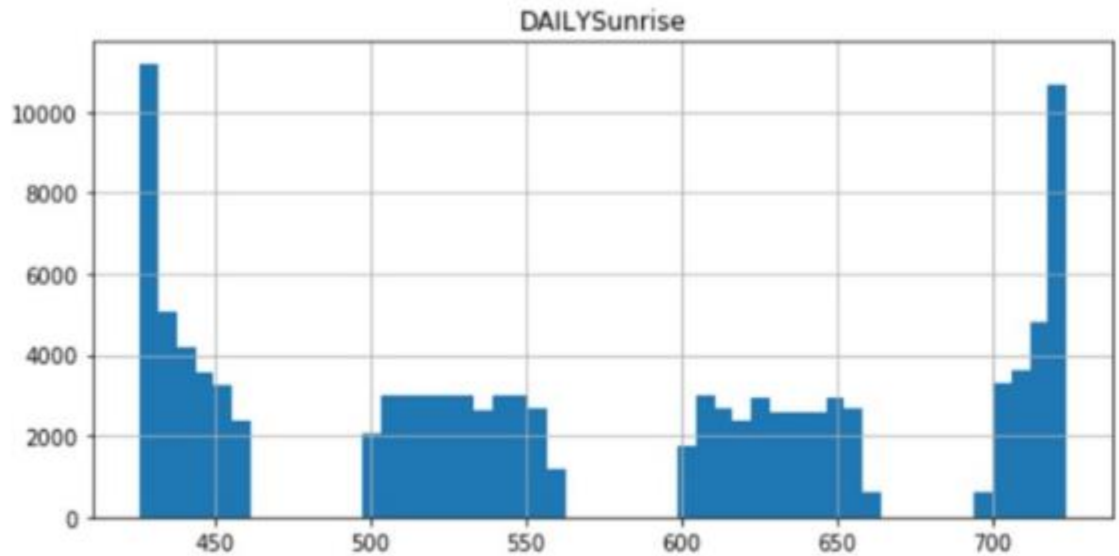


13. For analysis of the maximum and minum day length and we can analyse that the months around lesser or more than 11hrs of day length either increase or decrease the length of the day in next month. So we can start the and stop the day light saving from those month here October and february
And the Maximum day length is month of june(6)
With least day length lies around month of December (12)

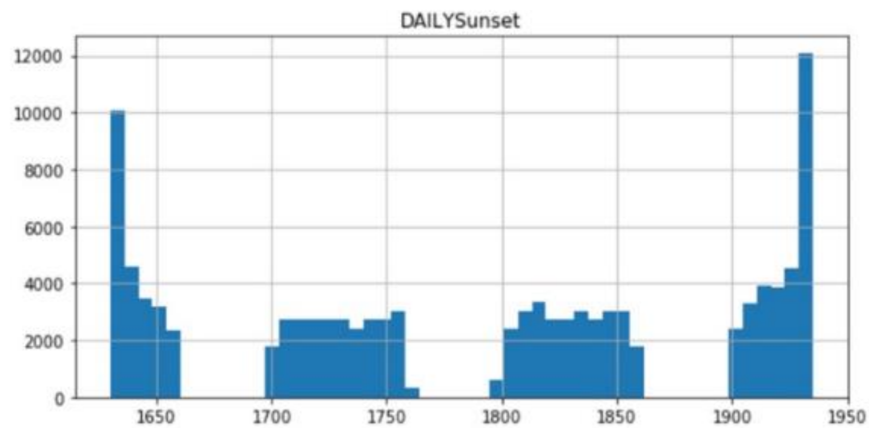
```
plt.scatter(mintempdata['month'], mintempdata['daylight'])
plt.show()
```



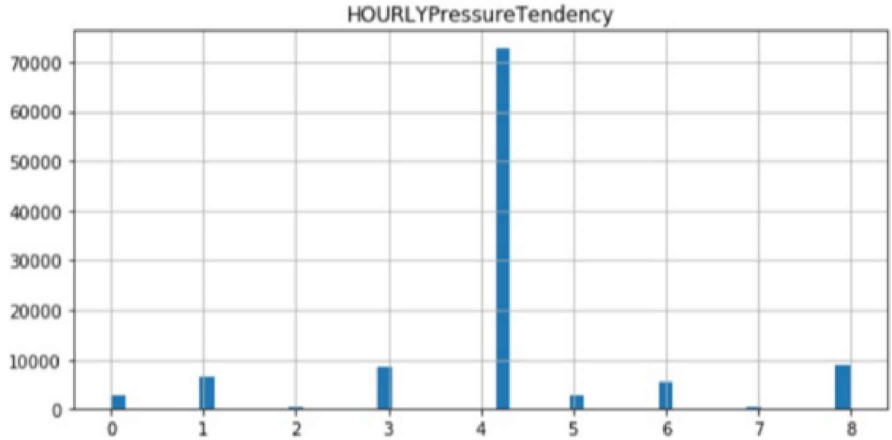
14. Similarly Using the Daily sunrise we can analysie the mostly the sunrise
    In colder months or lateror starting months of the years is late around 6:50-7:00 AM in morning
    While in hotter months of the year of the year its around 4:50-5:00AM
    And the months which lie about in middle of the year 500AM- 6:50 AM



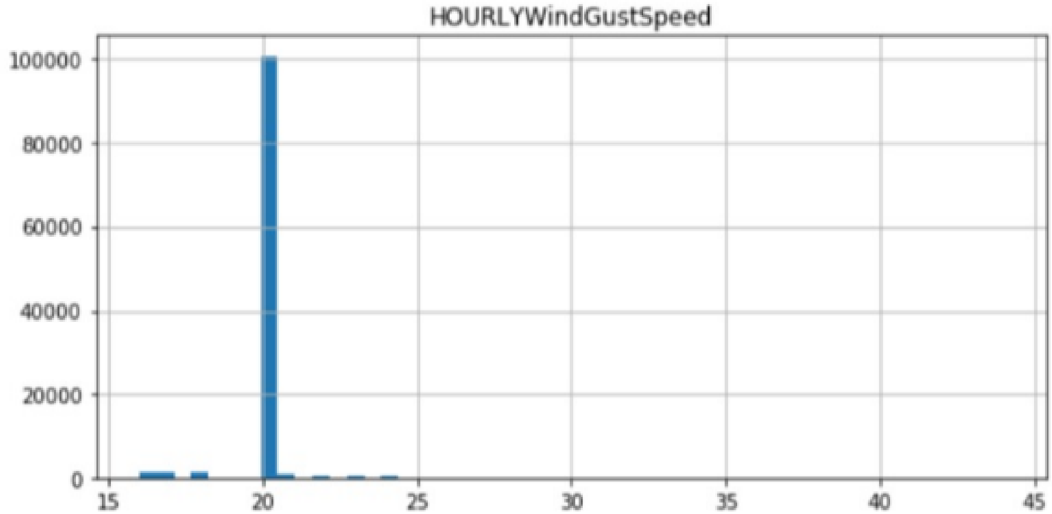
15. Similarly the sunset throughout the year analysis can be made
    Longer days of the year (in month near around june from previous analysis) the sunset is around 6:50PM- 7:50PM
    While the shorter days of years the sunset is around 4:50-5:50PM



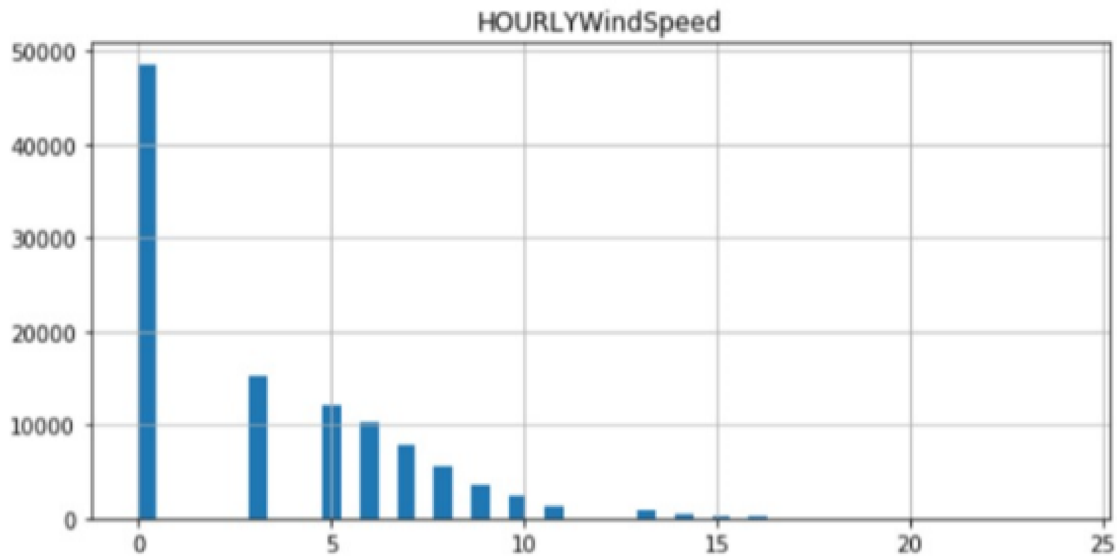
16. Similarly we can check the hourly pressure changes throughout a day
    We can see that the maximum pressure in increases around the May and
    Most varied around February and September thrught the year.



17. Similary the windgust analysis throughout the the year

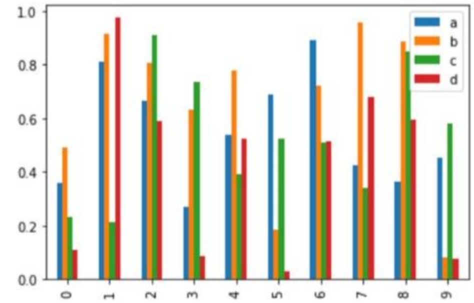


18. Here by plotting the hourly wind speed for the day we can analyse around the day the wind speed values



Now with some analysis some different parameters in one graph over the years.

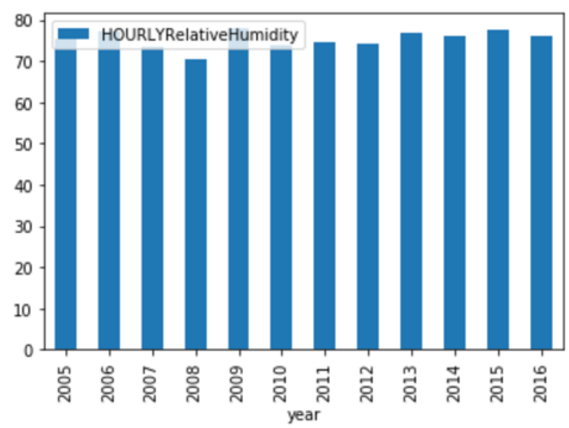
```
import matplotlib.pyplot as plt
noNaNdf[["DAILYSunrise", "DAILYSunset", "HOURLYPressureTendency", "HOURLYWindGustSpeed", "HOURLYWindSpeed"]].hist(bins=50,figsiz
plt.show()
```



19. Also Calculate humidity trend in August over the years, we can see in last 5 years from 2011-2016 it kept on increasing without a fall.
For theis first we selected the hourly humidity for everday of the month and calculates the mean for each August every month and creating Bar graph

	year	HOURLYRelativeHumidity
0	2005	75.203230
1	2006	77.254032
2	2007	73.373656
3	2008	70.323263
4	2009	77.879032
5	2010	73.858871
6	2011	74.732527
7	2012	74.354839
8	2013	76.952957
9	2014	76.237903
10	2015	77.575269
11	2016	76.096774

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launc
-type specific converters pd.to_datetime, pd.to_timedelta
  """
```



5. Data Wrangling :

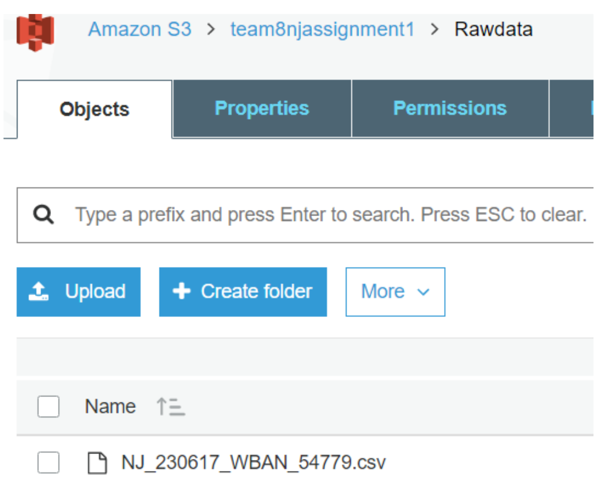
Data wrangling is the process of manipulating data to get reasonable values. This helps in data visualization as well because wrangled data is good for data visualization since it has less anomalies than that when working with Raw data.

For data wrangling process we use the: **configwrangle.json file and S3 bucket raw** data fetch which was created in data Ingestion step.

6.1 Steps For Data Wrangling

Step 1. First establish the connection with S3 using boto3 and config json and fetch the RawData CSV from S3 bucket for wrangling

```
# Loading the config.json file and get merged Csv

with open('configWrangle.json') as data_file:
        data = json.load(data_file)

#Extracting Data From last File Created.

rawdatafile= data["rawData"]
print(rawdatafile)

#Read from S3

        #data to be extracted from link
rawdata1 = pd.read_csv(rawdatafile)
```



Step 2. Convert all the datatype into numeric or float for easy exploration

#converting all int and float rows to Numeric datatype

rawdata.apply(pd.to_numeric, errors='ignore')

```
Shape of second file is : (4070, 90)
STATION                              object
STATION_NAME                         object
ELEVATION                           float64
LATITUDE                            float64
LONGITUDE                           float64
DATE                                 object
REPORTTPYE                           object
HOURLYSKYCONDITIONS                 float64
HOURLYVISIBILITY                    float64
HOURLYPRSENTWEATHERTYPE             float64
HOURLYDRYBULBTEMPF                   object
HOURLYDRYBULBTEMPC                   object
HOURLYWETBULBTEMPF                  float64
HOURLYWETBULBTEMPC                  float64
HOURLYDewPointTempF                 float64
HOURLYDewPointTempC                 float64
```

Step 3. Calculating the threshold value of for the maximum number of NaN values that can be present in the column

```
rowCount=len(rawdata.index)

rowCountpercent=rowCount*5/100

print (rowCountpercent)

treshold=rowCount=rowCountpercent

print (treshold)
```

Step 4. Deleting the columns which exceed the threshold value of NaN present in a column i.e.95%

```
rawdata=rawdata.dropna(thresh=len(rawdata) - treshold, axis=1)
```

```
deleted columns exceed the threshold value of NaN present in a column i.e.95%
(4070, 23)
STATION                              object
STATION_NAME                         object
ELEVATION                           float64
LATITUDE                            float64
LONGITUDE                           float64
DATE                                 object
REPORTTPYE                           object
```

Step 5. Replacing the NAN values with zero

```
replacezerorawdata=rawdata.replace('NaN',0)
print(replacezerorawdata.head(5))
```

```
replaced NaN with zero
STATION                       0
STATION_NAME                  0
ELEVATION                     0
LATITUDE                      0
LONGITUDE                     0
DATE                          0
REPORTTPYE                    0
HOURLYDRYBULBTEMPF            0
HOURLYDRYBULBTEMPC            0
HOURLYWETBULBTEMPF            0
HOURLYWETBULBTEMPC           13
HOURLYDewPointTempF          12
HOURLYDewPointTempC         115
HOURLYRelativeHumidity        0
HOURLYStationPressure         0
HOURLYSeaLevelPressure        0
HOURLYAltimeterSetting        0
```

Step 6. Again check if the zeros are more than 95% of the threshold valuesthen remove the column. Also change the datatype to numeric, float or integers

```
datasummary=(rawdata == 0).sum(axis=0)
print (datasummary)
rawdata = replacezerorawdata.loc[:, (replacezerorawdata != 0).any(axis=0)]
print (rawdata.shape)
print (rawdata.head(5))
rawdata.dropna(thresh=len(rawdata) - treshold, axis=1)

print (rawdata.shape)
rawdata = rawdata[rawdata.REPORTTPYE != 'SOD']
print (rawdata.shape)

print (rawdata.dtypes)
rawdata.head(5)
```

```
(4070, 23)
(3912, 23)
STATION                         object
STATION_NAME                    object
ELEVATION                      float64
LATITUDE                       float64
LONGITUDE                      float64
DATE                            object
REPORTTPYE                      object
HOURLYDRYBULBTEMPF              object
HOURLYDRYBULBTEMPC              object
HOURLYWETBULBTEMPF             float64
HOURLYWETBULBTEMPC             float64
HOURLYDewPointTempF            float64
HOURLYDewPointTempC            float64
HOURLYRelativeHumidity         float64
HOURLYStationPressure           object
HOURLYSeaLevelPressure          object
HOURLYAltimeterSetting          object
```

Step 7. We can calculate the number of zero values in each column and check which column is having the most Zero values or no data at all still.

datasummary=(rawdata == 0).sum(axis=0)
print ("Number of zeroes in a column",datasummary)

```
Number of zeroes in a column STATION
STATION_NAME                    0
ELEVATION                       0
LATITUDE                        0
LONGITUDE                       0
DATE                            0
REPORTTPYE                      0
HOURLYDRYBULBTEMPF              0
HOURLYDRYBULBTEMPC              0
HOURLYWETBULBTEMPF              0
HOURLYWETBULBTEMPC             13
HOURLYDewPointTempF            12
HOURLYDewPointTempC           115
HOURLYRelativeHumidity          0
HOURLYStationPressure           0
HOURLYSeaLevelPressure          0
HOURLYAltimeterSetting          0
```

Step 8. Creating a new informative column for analysis

Converting the daily time into the hourly datatype and calculating the day length.

```
time_func = lambda x: pd.Timestamp(pd.to_datetime(x, format = '%H%M'))

dataforsunrise=rawdata['DAILYSunrise'].apply(time_func)
dataforsunset=rawdata['DAILYSunset'].apply(time_func)
daylenght=(dataforsunset-dataforsunrise).astype('timedelta64[m]')/60

print(dataforsunset.head(3))
print(dataforsunrise.head(3))
print(daylenght.head(3))
```

Step 9. The new column named as daylength is added to the data set.

rawdata['LENGTHOFDAY']=daylenght.abs()
print (rawdata.head(5))

```
   MonthlyMinSeaLevelPressureDate  MonthlyMinSeaLevelPressureTime  LENGTHOFDAY
0                           -9999                           -9999     9.283333
1                           -9999                           -9999     9.283333
2                           -9999                           -9999     9.283333
3                           -9999                           -9999     9.283333
4                           -9999                           -9999     9.283333

[5 rows x 24 columns]
```

The data is cleaned and changed into the informative dataset and all the redundant values are removed. In the final dataset there are 24 meaningful columns are left with non zero values.

Final step 10. Now upload the cleaned data to S3 bucked and check if the file for the day already exist then do not upload and create the log error for the same and record success events in log file.

```
if   fileexists==0:

    k = Key(b)
    k.key = "CleanData/"+fname
    k.content_type = r.headers['content-type']
    k.set_contents_from_string(r.content)
# url = k.generate_url(expires_in=0, query_auth=False)
# print (url)

    print('successfully uploaded to s3')
#update json

 #log upload event
    my_logger.info("A clean file for the day was uploaded on S3 at:" +time.strftime("%d%m%Y%H%M%S"))
    cleanfilelink="https://s3.amazonaws.com/team8njassignment1/CleanData/"+fname+"_clean.csv"

#congif.Json file daily update last changed file.
# print (fname)
|    data["cleanData"]= cleanfilelink
    filename= 'configWrangle.json'
    with open('configWrangle.json', 'r') as f:
        data = json.load(f)
    data['cleanData'] = cleanfilelink # <--- add `id` value.

    os.remove(filename)
    with open(filename, 'w') as f:
        json.dump(data, f, indent=4)
        #log json file update event
```

```
[5 rows x 24 columns]
File Exists
```

6. Automation using Pipelines and Cloud:

The entire process of data extraction/retrieval and processing and analysing can be automated by using pipelines. This can also be scheduled to avoid human intervention and human errors. This helps scale the process for any amount of data and any type of data coming in. The other way is to automate it on the cloud and automating it on cloud is much easier since we have a management console on AWS and we do not have to type in commands and it runs on both Mac and Windows environment.

First Creat the docker image using the docker file:

7.1 DOCKERIZE (Image)

A Docker *image* is a read-only template used to create and launch a Docker *container*.

Steps to create docker file :

Docker File

1. Create a working directory" team8njassignment1" for docker image and then create a docker file inside it.

```
megha@DESKTOP-GOC6T0R MINGW64 ~
$ cd team8njassignment1
```

```
megha@DESKTOP-GOC6T0R MINGW64 ~/team8njassignment1
$ pwd
/c/Users/megha/team8njassignment1
```

2. Create a docker File
    - touch Dockerfile

```
megha@DESKTOP-GOC6T0R MINGW64 ~/team8njassignment1
$ touch Dockerfile
```

Docker file created

3. Now Configure you dockerfile to run python script file and shell (dataingestion.py and)
   Edit the Dockerfile using command
    - Cat Dockerfile

```
megha@DESKTOP-GOC6T0R MINGW64 ~/team8njassignment1
$ cat Dockerfile
```

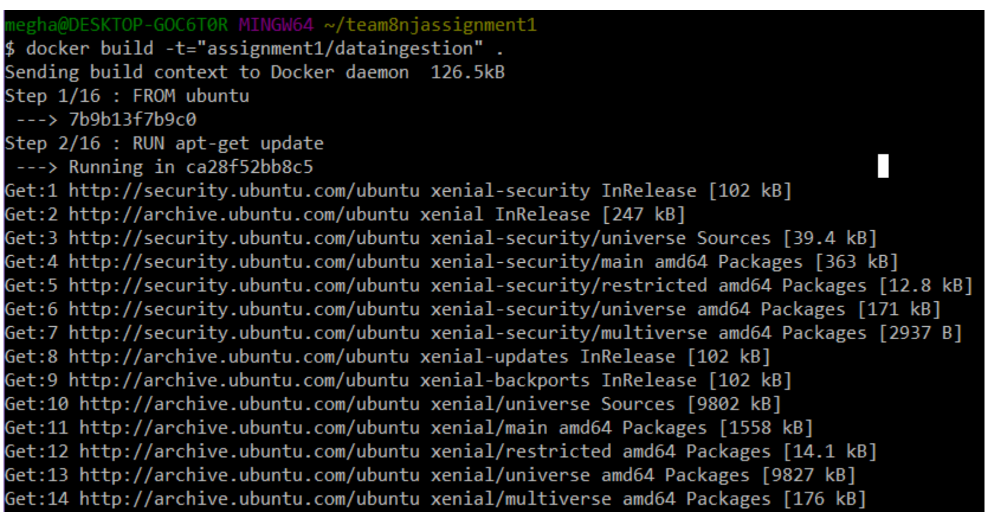
4. For dockerfile
   FROM ubuntu
   RUN apt-get update
   RUN apt-get update update apt-get install php5
   # Install Python.
   FROM python: 3

```
RUN mkdir -p /usr/src/team8njassignment1
WORKDIR /usr/src/team8njassignment1

COPY *.py *.json *.sh /usr/src/team8njassignment1
RUN pip install jupyter notebook
RUN pip install boto3
RUN pip install python-louvain
RUN pip install numpy
RUN pip install matplotlib
RUN pip install pandas
RUN pip install ipython
ADD run.sh /
RUN chmod +x /run.sh
#ENTRYPOINT ["python", "/usr/src/team8njassignment1/dataingestion.ny"]
```

Commands :
COPY
This instruction is similar to the COPY instruction with few added features like remote URL support in the source field and local-only tar extraction. But if you don't need a extra features, it is suggested to use COPY as it is more readable.

ENTRYPOINT
You can use this instruction to set the primary command for the image.
For example, if you have installed only one application in your image and want it to run whenever the image is executed, ENTRYPOINT is the instruction for you.
Also, all the elements specified using CMD will be overridden, except the arguments. They will be passed to the command specified in ENTRYPOINT.
CREATE a docker image using dockerfile

- docker build -t =”assignment1/dataingestion” .



```
megha@DESKTOP-GOC6T0R MINGW64 ~/team8njassignment1
$ docker build -t="assignment1/dataingestion" .
Sending build context to Docker daemon  126.5kB
Step 1/16 : FROM ubuntu
 ---> 7b9b13f7b9c0
Step 2/16 : RUN apt-get update
 ---> Running in ca28f52bb8c5
Get:1 http://security.ubuntu.com/ubuntu xenial-security InRelease [102 kB]
Get:2 http://archive.ubuntu.com/ubuntu xenial InRelease [247 kB]
Get:3 http://security.ubuntu.com/ubuntu xenial-security/universe Sources [39.4 kB]
Get:4 http://security.ubuntu.com/ubuntu xenial-security/main amd64 Packages [363 kB]
Get:5 http://security.ubuntu.com/ubuntu xenial-security/restricted amd64 Packages [12.8 kB]
Get:6 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 Packages [171 kB]
Get:7 http://security.ubuntu.com/ubuntu xenial-security/multiverse amd64 Packages [2937 B]
Get:8 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [102 kB]
Get:9 http://archive.ubuntu.com/ubuntu xenial-backports InRelease [102 kB]
Get:10 http://archive.ubuntu.com/ubuntu xenial/universe Sources [9802 kB]
Get:11 http://archive.ubuntu.com/ubuntu xenial/main amd64 Packages [1558 kB]
Get:12 http://archive.ubuntu.com/ubuntu xenial/restricted amd64 Packages [14.1 kB]
Get:13 http://archive.ubuntu.com/ubuntu xenial/universe amd64 Packages [9827 kB]
Get:14 http://archive.ubuntu.com/ubuntu xenial/multiverse amd64 Packages [176 kB]
```

Docker image is created.

7.2 Celery Task Scheduling with RabbitMQ

Celery task scheduling

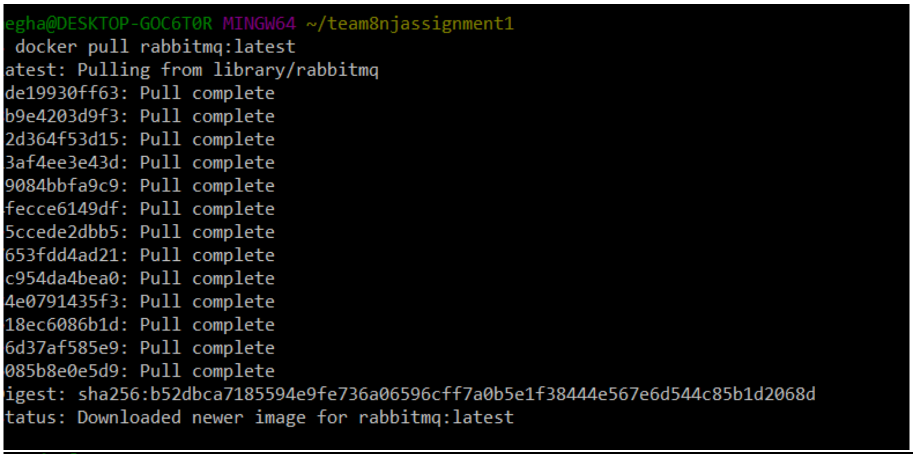
Celery is an asynchronous task queue/job queue based on distributed message passing. It is focused on real-time operation, but supports scheduling as well. The execution units, called tasks, are executed concurrently on a single or more worker servers using multiprocessing, Eventlet, or gevent. Tasks can execute asynchronously (in the background) or synchronously (wait until ready). Celery is used in production systems to process millions of tasks a day.

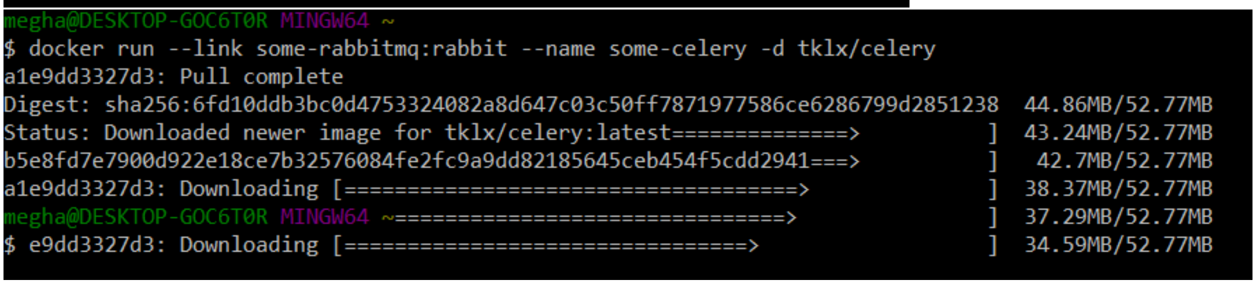
Celery, a python library which sits on top of RabbitMQ and provides workers to execute tasks.

Start a celery worker utilizing RabbitMQ as broker

```
$ docker run --name some-rabbitmq -d tklx/rabbitmq
$ docker run --link some-rabbitmq:rabbit --name some-celery -d tklx/celery
```

```
megha@DESKTOP-GOC6T0R MINGW64 ~/team8njassignment1
$ docker run --name some-rabbitmq -d tklx/rabbitmq
Unable to find image 'tklx/rabbitmq:latest' locally
latest: Pulling from tklx/rabbitmq
36ad7e70e3a7: Pull complete
47b22d221ef2: Pull complete
afe64b3372a9: Pull complete
e265a005d854: Pull complete
b4adffba2190: Pull complete
32da77bd4f40: Pull complete
Digest: sha256:37de8b06c6f07166fa3798bf634f91510b7c63e0be40a807e6b9813cbc81cddb
Status: Downloaded newer image for tklx/rabbitmq:latest
c602e062f2d9202e41e615300aa08a121409daaec63b3f540918c062d8a53d3d
```





A task is just a Python function. You can think of scheduling a task as a time-delayed call to the function. For example, you might ask Celery to call your function task1 with arguments (1, 3,3) after five minutes. Or you could have your function batchjob called every night at midnight.

When a task is ready to be run, Celery puts it on a queue, a list of tasks that are ready to be run. You can have many queues, but we'll assume a single queue here for simplicity.

Putting a task on a queue just adds it to a to-do list, so to speak. In order for the task to be executed, some other process, called a *worker*, has to be watching that queue for tasks. When it sees tasks on the queue, it'll pull off the first and execute it, then go back to wait for more. You can have many workers, possibly on many different servers, but we'll assume a single worker for now.

7.3 AWS LAMBDA FUCNTION:

The code run on AWS lambda is called lambda function

After you create your lambda function it is always ready to run as as soon as it is triggered, similar to a formula in a spreadsheet.

After you upload your code to lambda function or fetch docker image on lambda function to run the docker image , you can associate this function with specific AWS resources such as SNS notification

AWS Batch Job Queuing

What Is AWS Batch?

AWS Batch enables you to run batch computing workloads on the AWS Cloud. Batch computing is a common way for developers, scientists, and engineers to access large amounts of compute resources, and AWS Batch removes the undifferentiated heavy lifting of configuring and managing the required infrastructure. AWS Batch is similar to traditional batch computing software. This service can efficiently provision resources in response to jobs submitted in order to eliminate capacity constraints, reduce compute costs, and deliver results quickly.

As a fully managed service, AWS Batch enables developers, scientists, and engineers to run batch computing workloads of any scale. AWS Batch automatically provisions compute resources and optimizes the workload distribution based on the quantity and scale of the workloads. With AWS Batch, there is no need to install or manage batch computing software, which allows you to focus on analyzing results and solving problems. AWS Batch reduces operational complexities, saves time, and reduces costs, which makes it easy for developers, scientists, and engineers to run their batch jobs in the AWS Cloud.

Jobs

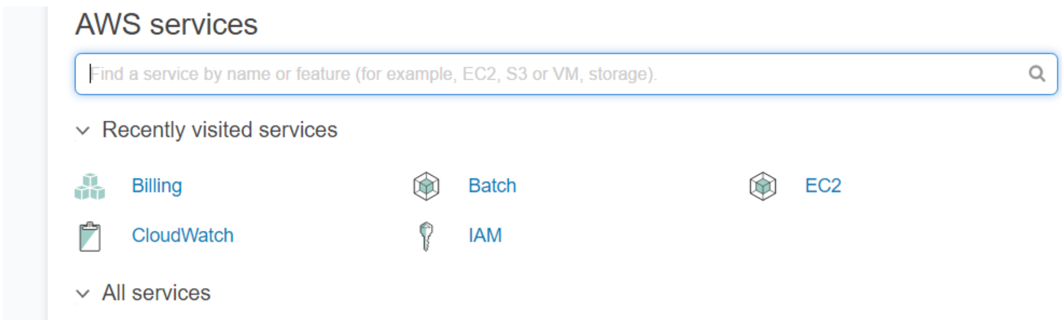
Jobs are the unit of work executed by AWS Batch. Jobs can be executed as containerized applications running on Amazon ECS container instances in an ECS cluster.Containerized jobs can reference a container image, command, and parameters

Job Scheduling

The AWS Batch scheduler evaluates when, where, and how to run jobs that have been submitted to a job queue. Jobs run in approximately the order in which they are submitted as long as all dependencies on other jobs have been met.

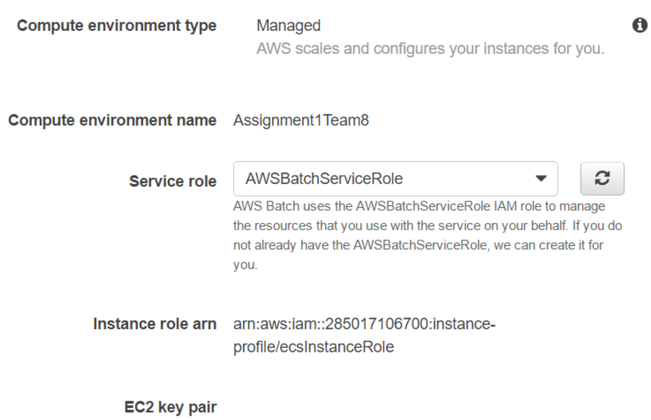
Steps to start a Batch for Amazon AWS S3:

1. Go to AWS Batch service and then go to create a Job



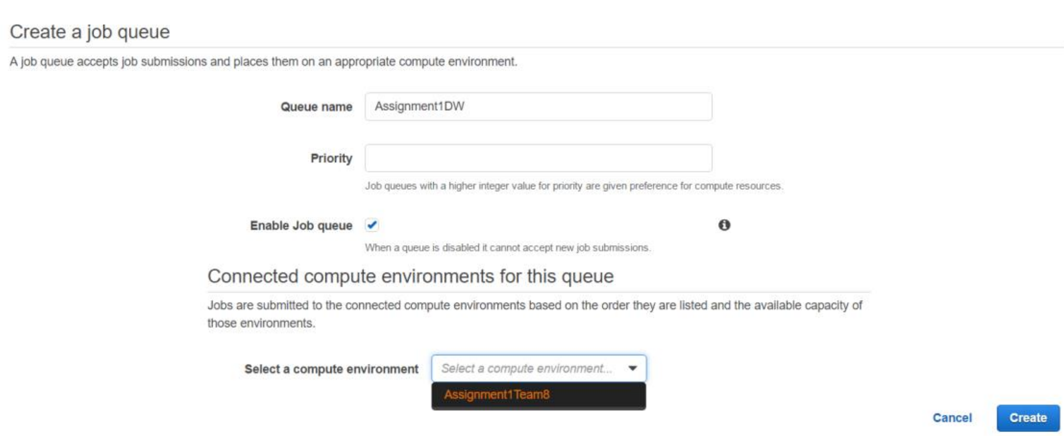
2. Create a Job Environment

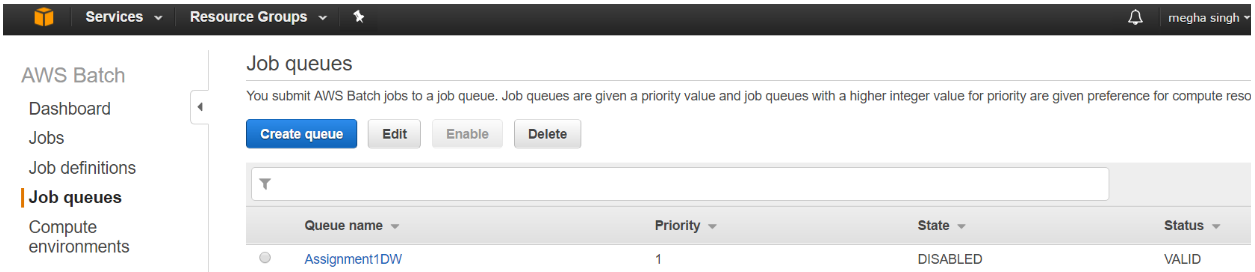
Use the managed configuration for EC2 instance



3. Now create a 1 queue which will hold your Jobs/task to run according to their priority/definition.

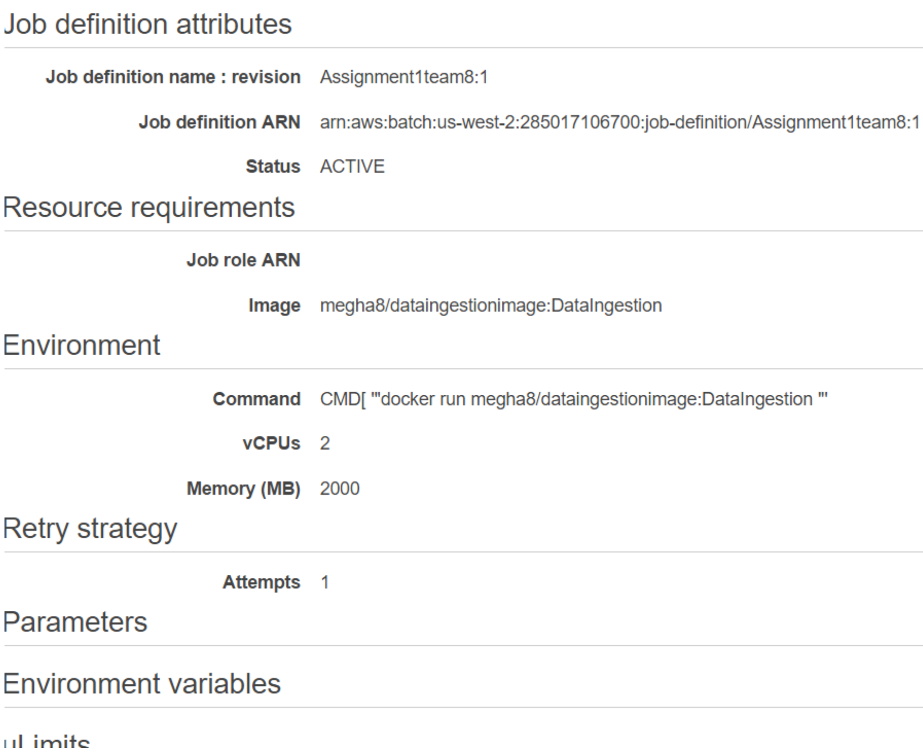
By using the pre-created Environment configuration(done in previous step.)





4. Now create the job definition: which will create the job you need to perform inside the queue (created in precvious step)

In command fetch the Docker image from dockerhub by giving the command CMD:



Similary, Create the 2nd job inside the same environment and same job queue

Job definition attributes

Job definition name : revision	DatawranglingAssignment1:1
Job definition ARN	arn:aws:batch:us-west-2:285017106700:job-definition/DatawranglingAssignment1:1
Status	ACTIVE

Resource requirements

Job role ARN	
Image	megha8/team8njassignment1:wrangling

Environment

Command	CMD["'docker run megha8/team8njassignment1'"]
vCPUs	2
Memory (MB)	2000

Retry strategy

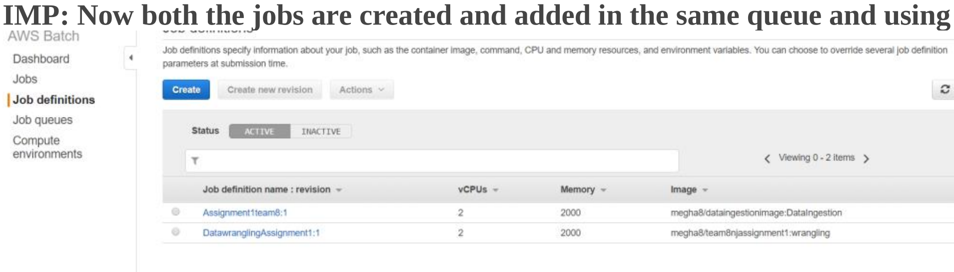
Attempts	1

Parameters

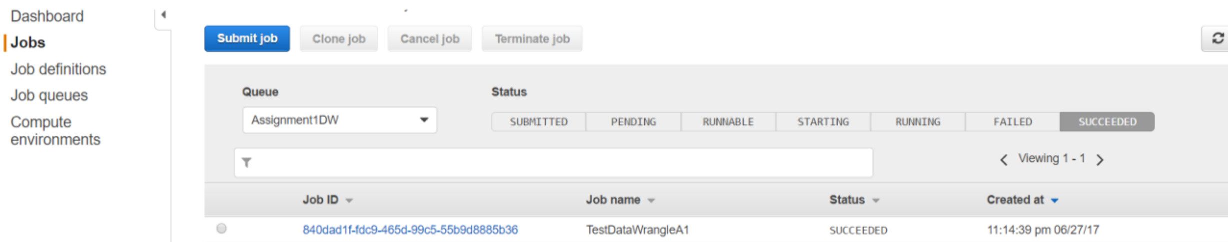
Environment variables

name	DatawranglingAssignment1

IMP: Now both the jobs are created and added in the same queue and using the same environment.



After running the job queue We can check the status of the Jobs being performed inside the queue



AWS Batch Scheduling for job is created with S3 bucket

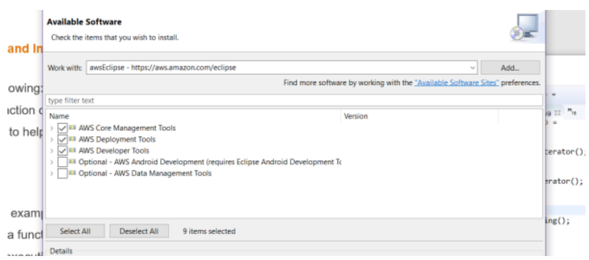
What Is AWS Lambda?

AWS Lambda is a compute service that lets you run code without provisioning or managing servers. **AWS Lambda executes your code only when needed and scales automatically**, from a few requests per day to thousands per second. You pay only for the compute time you consume - there is no charge when your code is not running.

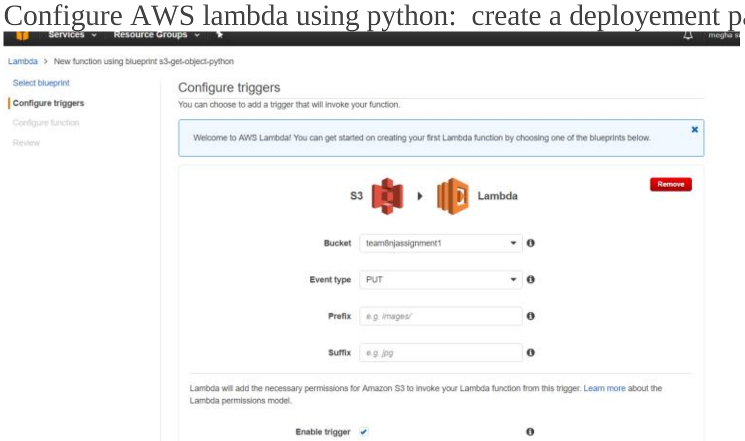
Create a Lambda Function and Invoke It Manually (Using Sample Event Data)

In this section, you do the following:

1. Create a Lambda function deployment package using the sample code provided.

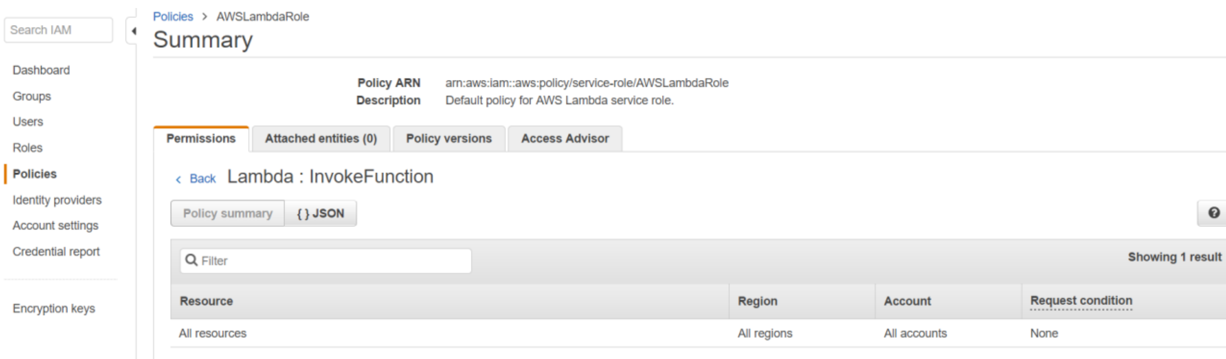


2. Configure AWS lambda using python: create a deployement package using python



3. Create an IAM role (execution role). At the time you upload the deployment package, you need to specify an IAM role (execution role) that Lambda can assume to execute the function on your behalf.
4. Create the Lambda function by uploading the deployment package, and then test it by invoking it manually using sample Amazon S3 event data.

Invoke lambda function from IAM role.



Citations

https://www.ibm.com/developerworks/community/blogs/jfp/entry/using_ipython_notebooks_in_docker_containers_on_windows?lang=en

boto.cloudhackers.com/en/latest/ref/s3.html

http://www.slashroot.in/dockerfile-tutorial-building-docker-images-for-containers

https://stackoverflow.com

http://docs.aws.amazon.com/batch/latest/userguide/Batch_GetStarted.html

http://rapidsms.readthedocs.io/en/develop/topics/celery.html