Unit 8: Automation

Contents

In [1]:
```
import sys
!{sys.executable} -m pip install boto3 pandas-datareader
```

```
Requirement already satisfied: boto3 in /anaconda3/lib/python3.7/site-packages (1.9.191)
Requirement already satisfied: pandas-datareader in /anaconda3/lib/python3.7/site-packages (0.7.0)
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /anaconda3/lib/python3.7/site-packages (from boto3) (0.9.4)
Requirement already satisfied: botocore<1.13.0,>=1.12.191 in /anaconda3/lib/python3.7/site-packages (from boto3) (1.12.191)
Requirement already satisfied: s3transfer<0.3.0,>=0.2.0 in /anaconda3/lib/python3.7/site-packages (from boto3) (0.2.1)
Requirement already satisfied: requests>=2.3.0 in /anaconda3/lib/python3.7/site-packages (from pandas-datareader) (2.21.0)
Requirement already satisfied: pandas>=0.19.2 in /anaconda3/lib/python3.7/site-packages (from pandas-datareader) (0.24.2)
Requirement already satisfied: wrapt in /anaconda3/lib/python3.7/site-packages (from pandas-datareader) (1.11.1)
Requirement already satisfied: lxml in /anaconda3/lib/python3.7/site-packages (from pandas-datareader) (4.3.2)
Requirement already satisfied: docutils>=0.10 in /anaconda3/lib/python3.7/site-packages (from botocore<1.13.0,>=1.12.191->boto3) (0.
14)
Requirement already satisfied: urllib3<1.26,>=1.20; python_version >= "3.4" in /anaconda3/lib/python3.7/site-packages (from botocore
<1.13.0,>=1.12.191->boto3) (1.24.1)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1; python_version >= "2.7" in /anaconda3/lib/python3.7/site-packages (from
botocore<1.13.0,>=1.12.191->boto3) (2.8.0)
Requirement already satisfied: idna<2.9,>=2.5 in /anaconda3/lib/python3.7/site-packages (from requests>=2.3.0->pandas-datareader)
(2.8)
Requirement already satisfied: certifi>=2017.4.17 in /anaconda3/lib/python3.7/site-packages (from requests>=2.3.0->pandas-datareade
r) (2019.3.9)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /anaconda3/lib/python3.7/site-packages (from requests>=2.3.0->pandas-datarea
der) (3.0.4)
Requirement already satisfied: numpy>=1.12.0 in /anaconda3/lib/python3.7/site-packages (from pandas>=0.19.2->pandas-datareader) (1.1
6.2)
Requirement already satisfied: pytz>=2011k in /anaconda3/lib/python3.7/site-packages (from pandas>=0.19.2->pandas-datareader) (2018.
9)
Requirement already satisfied: six>=1.5 in /anaconda3/lib/python3.7/site-packages (from python-dateutil<3.0.0,>=2.1; python_version
>= "2.7"->botocore<1.13.0,>=1.12.191->boto3) (1.12.0)
```

In [2]:
```
# Set keys for report_gen_kelly
ACCESS_KEY = 'AKIAS5TSMA54BY76MH7U'
SECRET_KEY = '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'
```

Creating a Data Store

Our objective is to automate data collection and processing and report generation and delivery. Once we've collected the data and completed some initial processing, we'll likely need to store it for future use. While we could create a relational database using the Amazon Relational Database Service (https://aws.amazon.com/rds/), we'll instead rely on a NoSQL store using Amazon SimpleDB (https://aws.amazon.com/simpledb/).

To work with SimpleDB from boto, we must first create a *client()* (https://boto3.readthedocs.io/en/latest/guide/quickstart.html) indicating the service we intend to use and with our credentials.

For this example, we'll use the "us-east-1" region as it includes all the AWS services we need.

In [3]:
```
import boto3
labdb_client = boto3.client('sdb',
                            aws_access_key_id = ACCESS_KEY,
                            aws_secret_access_key = SECRET_KEY,
                            region_name="us-east-1")
```

Instead of databases and tables, SimpleDB uses *domains* to categorize data. To create a new domain, we can use the client's *create_domain()* (https://boto3.readthedocs.io/en/latest/reference/services/sdb.html#SimpleDB.Client.create_domain) method.

In [4]:
```
# Delete domain if it exists
labdb_client.delete_domain(DomainName='lab_domain')
```

Out[4]:
```
{'ResponseMetadata': {'RequestId': '169bf43c-7747-c440-acc7-e8122792d579',
  'BoxUsage': '0.0055590278',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'date': 'Sun, 21 Jul 2019 18:00:37 GMT',
   'content-type': 'text/xml',
   'transfer-encoding': 'chunked',
   'connection': 'keep-alive',
   'vary': 'Accept-Encoding',
   'server': 'Amazon SimpleDB'},
  'RetryAttempts': 0}}
```

In [5]:
```
labdb_client.create_domain(DomainName="lab_domain")
```

Out[5]:
```
{'ResponseMetadata': {'RequestId': 'aaf04266-789b-2553-50bd-adfa78ddc113',
  'BoxUsage': '0.0055590278',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'date': 'Sun, 21 Jul 2019 18:00:38 GMT',
   'content-type': 'text/xml',
   'transfer-encoding': 'chunked',
   'connection': 'keep-alive',
   'vary': 'Accept-Encoding',
   'server': 'Amazon SimpleDB'},
  'RetryAttempts': 0}}
```

If successful, the *create_domain()* method returns information related to the domain. Note that an HTTP status code of 200 (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes#2xx_Success) indicates success. We can also verify that the domain was created successfully by viewing its metadata. To do this, we can use the client's *domain_metadata()* (https://boto3.readthedocs.io/en/latest/reference/services/sdb.html#SimpleDB.Client.domain_metadata) method. When calling the method, we specify the domain name using the `DomainName` keyword argument.

Retrieve the metadata for the recently-created domain using the *domain_metadata()* method.

In [6]:
```
labdb_client.domain_metadata(DomainName="lab_domain")
```

Out[6]:
```
{'ItemCount': 0,
 'ItemNamesSizeBytes': 0,
 'AttributeNameCount': 0,
 'AttributeNamesSizeBytes': 0,
 'AttributeValueCount': 0,
 'AttributeValuesSizeBytes': 0,
 'Timestamp': 1563732039,
 'ResponseMetadata': {'RequestId': '4f9b6624-b779-15cd-90dd-45ec1411414f',
  'BoxUsage': '0.0000071759',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'date': 'Sun, 21 Jul 2019 18:00:39 GMT',
   'content-type': 'text/xml',
   'transfer-encoding': 'chunked',
   'connection': 'keep-alive',
   'vary': 'Accept-Encoding',
   'server': 'Amazon SimpleDB'},
  'RetryAttempts': 0}}
```

Data Retrieval and Processing

In [7]:
```
import requests
token='pk_5e593db6de2f4e809c319ec92d7bf9a2'
response = requests.get('https://cloud.iexapis.com/stable/stock/goog/chart/5d/quote?token=' + token)
# OLD response = requests.get("https://api.iextrading.com/1.0/stock/goog/price")
#price = response.json()
#price
goog5days = response.json()
goog5days
```

Out[7]:

```
[{'date': '2019-07-15',
  'open': 1146.86,
  'close': 1150.34,
  'high': 1150.82,
  'low': 1139.4,
  'volume': 903780,
  'uOpen': 1146.86,
  'uClose': 1150.34,
  'uHigh': 1150.82,
  'uLow': 1139.4,
  'uVolume': 903780,
  'change': 0,
  'changePercent': 0,
  'label': 'Jul 15',
  'changeOverTime': 0},
 {'date': '2019-07-16',
  'open': 1146,
  'close': 1153.58,
  'high': 1158.58,
  'low': 1145,
  'volume': 1238807,
  'uOpen': 1146,
  'uClose': 1153.58,
  'uHigh': 1158.58,
  'uLow': 1145,
  'uVolume': 1238807,
  'change': 3.24,
  'changePercent': 0.2817,
  'label': 'Jul 16',
  'changeOverTime': 0.002817},
 {'date': '2019-07-17',
  'open': 1150.97,
  'close': 1146.35,
  'high': 1158.36,
  'low': 1145.77,
  'volume': 1170047,
  'uOpen': 1150.97,
  'uClose': 1146.35,
  'uHigh': 1158.36,
  'uLow': 1145.77,
  'uVolume': 1170047,
  'change': -7.23,
  'changePercent': -0.6267,
  'label': 'Jul 17',
  'changeOverTime': -0.003469},
 {'date': '2019-07-18',
  'open': 1141.74,
  'close': 1146.33,
  'high': 1147.6,
  'low': 1132.73,
  'volume': 1291281,
  'uOpen': 1141.74,
  'uClose': 1146.33,
  'uHigh': 1147.6,
  'uLow': 1132.73,
  'uVolume': 1291281,
  'change': -0.02,
  'changePercent': -0.0017,
  'label': 'Jul 18',
  'changeOverTime': -0.003486},
 {'date': '2019-07-19',
  'open': 1148.19,
  'close': 1130.1,
  'high': 1151.14,
  'low': 1129.62,
  'volume': 1647245,
  'uOpen': 1148.19,
  'uClose': 1130.1,
  'uHigh': 1151.14,
  'uLow': 1129.62,
  'uVolume': 1647245,
  'change': -16.23,
  'changePercent': -1.4158,
  'label': 'Jul 19',
  'changeOverTime': -0.017595}]
```


We can store price data and number of shares for each stock in a SimpleDB domain. To start, we'll create the domain and an item for each stock where the stock's name will be used as the item name. Price and number of shares will be stored as attributes.

In [8]:
```
print(len(goog5days))
print(goog5days[0].get("date"))
```

```
5
2019-07-15
```

In [10]:
```
for day in goog5days:
    labdb_client.put_attributes(DomainName='lab_domain',
                                ItemName='goog',
                                 Attributes=[{'Name': str(day['date']), 'Value': str(day['close'])},]
                                )
```

Confirm that the items existing using the *select()* method.

In [11]:
```
labdb_client.select(SelectExpression="select * from lab_domain")
```

Out[11]:
```
{'Items': [{'Name': 'goog',
   'Attributes': [{'Name': '2019-07-15', 'Value': '1150.34'},
    {'Name': '2019-07-16', 'Value': '1153.58'},
    {'Name': '2019-07-17', 'Value': '1146.35'},
    {'Name': '2019-07-18', 'Value': '1146.33'},
    {'Name': '2019-07-19', 'Value': '1130.1'}]}],
 'ResponseMetadata': {'RequestId': 'e3b6e832-3ab8-d773-c50d-016b3097b9c2',
  'BoxUsage': '0.0000228616',
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'date': 'Sun, 21 Jul 2019 18:03:09 GMT',
   'content-type': 'text/xml',
   'transfer-encoding': 'chunked',
   'connection': 'keep-alive',
   'vary': 'Accept-Encoding',
   'server': 'Amazon SimpleDB'},
  'RetryAttempts': 0}}
```

In [12]:
```
ses_client = boto3.client('ses',
                          aws_access_key_id = ACCESS_KEY,
                          aws_secret_access_key = SECRET_KEY,
                          region_name="us-east-1")
```

We can now combine the ability to retrieve data from SimpleDB with our ability to send email using the Simple Email Service, to create and send a report. To do this, we'll create two functions. The first will generate the report and the second will send the report email.

In [24]:
```
def report(labdb_client):
    email_lines = []

    results = labdb_client.select(SelectExpression="select * from lab_domain")
    for item in results['Items']:
        name = item['Name']
        for attribute in item['Attributes']:
            msgtxt = name + " " + attribute.get('Name') + " " + attribute.get('Value')
#            print(msgtxt)
            message = f"{msgtxt}"
            email_lines.append(message)
    return "\n".join(email_lines)
#
print(report(labdb_client))
```

```
goog 2019-07-15 1150.34
goog 2019-07-16 1153.58
goog 2019-07-17 1146.35
goog 2019-07-18 1146.33
goog 2019-07-19 1130.1
```

To send the email, we have the following.

In [25]:
```
def send_report(ses_client, dest_addr, message):
    CHARSET = "UTF-8"

    ses_client.send_email(
        Source=dest_addr,
        Destination={
            'ToAddresses': [
                dest_addr
            ]
        },
        Message={
            'Body': {
                'Text': {
                    'Charset': CHARSET,
                    'Data': message
                }
            },
            'Subject': {
                'Charset': CHARSET,
                'Data': "Report"
            }
        }
    )
```

Report generation and message delivery.

In [27]:
```
ADDRESS = 'kbloom1@student.cscc.edu'
report(labdb_client)
send_report(ses_client, ADDRESS, report(labdb_client))
```