Taylor Shrode
Using APIs and Data Architecture
2/16/2020
MSDS 610

Written Portion

1. What is an API, and what can we use them for?

   API stands for Application Programming Interface. Breaking down the acronym, an application provides a function, deals with inputs and outputs, and it needs to communicate with the user and other applications (Wodehouse, 2016). The "programming is the engineering part of the app's software that translates input into output" (Wodehouse, 2016). Finally, the interface is how we interact with an application (Wodehouse, 2016). An API lists a bunch of operations that developers use with a description of what they do (Hoffman, 2016).

   An APIs main function is to allow applications to communicate with one another (Eising, 2017). An API is not a database or the server, but rather the code that governs the access point for the server (Eising, 2017). APIs make it easier to develop programs because they provide the "building blocks", all the programmer must do is put the blocks together (Beal).

   A few popular APIs include the Google Maps API, YoutTube APIs, Flickr API, Twitter APIs, and Amazon Product Advertising API (Beal). The Google Maps API allows developers embed Google Maps on webpages and is designed to work on mobile devices and desktops (Beal). YouTube APIs allows developers to integrate YouTube videos and functionality into websites or applications (Beal). The Flickr API allows developers to access Flick photo sharing community data and Twitter APIs allow users to access Twitter data (Beal). Simply put, APIs make life easier for developers. For example, if you want to capture a photo or video using an iPhone camera, the camera API to embed the iPhone's built-in camera in your app. Without the API, app developers would have to create their own camera software and interpret the camera hardware's input (Hoffman, 2018).

2. When should we consider putting API-fetched data in SQL vs a NoSQL database?

   Recall that SQL is a relational database with data organized in a tabular structure, higher cost, follows ACID, and is vertically scalable. On the other hand, NoSQL is lower in cost, horizontally scalable, follows the CAP theorem, and uses different models (document, graph, key value, or column) (Vineet, 2016). When deciding which is better to put the API-fetched data, we need to consider the data-retention policy, data growth, and the API traffic. API traffic has a few characteristics such as high frequency, payload sizes, data structure, tables, volume, and objects (Vineet, 2016). Most services have a rate-limit for their customers based on their usage. For example, the MetaWeather website API tells users not to make requests less than a minute apart. When choosing between SQL or NoSQL, we need to consider factors such as data growth, online versus archived data, search filter flexibility, search performance, and clustering (Vineet, 2016). SQL can be used to store API-fetched data for manageable data sizes, low time period or size-based retention policies, low usage frequencies, and when the query interface needs to be standardized (Vineet, 2016). NoSQL can be used when scale and volume are important, fast queries are important, and deep analytics are needed (Vineet, 2016).

3. What was challenging about using APIs?

   https://api2cart.com/business/6-difficulties-api-integration-way-avoid/

Technical Portion

An Application Programming Interface, API, provides a way for two systems to communicate. The two systems can be a programming language and a web service (REST APIs), another programming language (Python or R), an operating system (os in Python), computer hardware (GPUs), or a database (pymongo with MongoDB). We will be using web-based API, which has become more popular for getting and serving data. Instead of giving access to a SQL database, a user may create a REST API to allow others to get data. REST stands for Representational State Transfer. REST determines what the API looks like and it has a set of rules that developers follow when they create their API (Zell, 2018). One rule states that a user should be able to get a piece of data (resource) when you link to a URL. The URL is called a **request** and the data sent back to the user is called a **response** (Zell, 2018). We will be retrieving data from a web API and store it in a database, specifically in MongoDB using Python.

For this assignment, we will be using data from MetaWeather. MetaWeather "is an automated weather data aggregator that takes the weather predictions from various forecasters and calculates the most likely outcome." MetaWeather provides an API that delivers JSON over HTTPS for access to our data. To access the weather prediction API for a particular location, we use the URL in the format https://www.metaweather.com/api/location/(woeid)/, where 'woeid' stands for 'where on Earth ID'. For this, we will be using Denver, CO weather predictions. The 'woeid' for Denver is 2391279, which makes URL for the API https://www.metaweather.com/api/location/2391279/. To test the API to the meta weather website, we can use the VM Instance created in Week 1 and use the Curl command to 'get' the data.

Taylor Shrode
Using APIs and Data Architecture
2/16/2020
MSDS 610

```
g07hockeychix67@instance-1:~$ curl https://www.metaweather.com/api/location/2391279/
```


```
{"consolidated_weather":[{"id":4570164722925568,"weather_state_name":"Light Cloud","weather_state_abbr":"lc","wind_directi
on_compass":"ESE","created":"2020-02-13T15:49:11.816291Z","applicable_date":"2020-02-13","min_temp":-10.170000000000002,"m
ax_temp":0.09,"the_temp":-2.515,"wind_speed":2.5851128575174314,"wind_direction":121.39250037145105,"air_pressure":1023.5,
"humidity":74,"visibility":12.155263262546725,"predictability":70},{"id":4831712393560064,"weather_state_name":"Clear","we
ather_state_abbr":"c","wind_direction_compass":"SSW","created":"2020-02-13T15:49:12.137127Z","applicable_date":"2020-02-14
","min_temp":-7.9,"max_temp":5.73,"the_temp":3.375,"wind_speed":5.232551504652828,"wind_direction":202.83326608975213,"air
_pressure":1012.0,"humidity":79,"visibility":14.136194623399348,"predictability":68},{"id":6521043290161152,"weather_state
_name":"Light Cloud","weather_state_abbr":"lc","wind_direction_compass":"SSE","created":"2020-02-13T15:49:15.800817Z","app
licable_date":"2020-02-15","min_temp":-6.699999999999999,"max_temp":3.855,"the_temp":2.1950000000000003,"wind_speed":4.549
488535859912,"wind_direction":163.8336647919731,"air_pressure":1018.5,"humidity":74,"visibility":14.487890718205678,"predi
ctability":70},{"id":5372683120279552,"weather_state_name":"Light Cloud","weather_state_abbr":"lc","wind_direction_compass
":"SSW","created":"2020-02-13T15:49:18.122812Z","applicable_date":"2020-02-16","min_temp":-3.56,"max_temp":8.9050000000000
01,"the_temp":6.41,"wind_speed":3.5007031838887563,"wind_direction":197.39097722886868,"air_pressure":1010.0,"humidity":68
,"visibility":14.358334824624194,"predictability":70},{"id":5429084026830848,"weather_state_name":"Snow","weather_state_ab
br":"sn","wind_direction_compass":"E","created":"2020-02-13T15:49:21.808846Z","applicable_date":"2020-02-17","min_temp":-3
.4749999999999996,"max_temp":7.0649999999999995,"the_temp":5.26,"wind_speed":5.291254308133832,"wind_direction":94.3418542
3030522,"air_pressure":1009.5,"humidity":65,"visibility":11.124097769028872,"predictability":90},{"id":6344851853410304,"w
eather_state_name":"Snow","weather_state_abbr":"sn","wind_direction_compass":"NNE","created":"2020-02-13T15:49:24.044755Z"
,"applicable_date":"2020-02-18","min_temp":-8.705,"max_temp":-1.135,"the_temp":-4.41,"wind_speed":3.1964236856756543,"wind
_direction":17.500000000000007,"air_pressure":1023.0,"humidity":87,"visibility":0.1938678119780482,"predictability":90}],"
time":"2020-02-13T10:28:42.857932-07:00","sun_rise":"2020-02-13T06:55:22.590420-07:00","sun_set":"2020-02-13T17:32:18.4133
01-07:00","timezone_name":"LMT","parent":{"title":"Colorado","location_type":"Region / State / Province","woeid":2347564,"
latt_long":"38.997921,-105.550957"},"sources":[{"title":"BBC","slug":"bbc","url":"http://www.bbc.co.uk/weather/","crawl_ra
te":360},{"title":"Forecast.io","slug":"forecast-io","url":"http://forecast.io/","crawl_rate":480},{"title":"HAMweather","
slug":"hamweather","url":"http://www.hamweather.com/","crawl_rate":360},{"title":"Met Office","slug":"met-office","url":"h
ttp://www.metoffice.gov.uk/","crawl_rate":180},{"title":"OpenWeatherMap","slug":"openweathermap","url":"http://openweather
map.org/","crawl_rate":360},{"title":"Weather Underground","slug":"wunderground","url":"https://www.wunderground.com/?apir
ef=fc30dc3cd224e19b","crawl_rate":720},{"title":"World Weather Online","slug":"world-weather-online","url":"http://www.wor
ldweatheronline.com/","crawl_rate":360}],"title":"Denver","location_type":"City","woeid":2391279,"latt_long":"39.740009,-1
04.992264","timezone":"US/Mountain"}g07hockeychix67@instance-1:~$
```


The API loads correctly and returns data for Denver, CO. Now, we can stop our VM Instance and we can switch and use a Jupyter Notebooks Python Program. In the Python program, we first import **requests,** which allows us to send HTTP/1.1 requests without having to manually add query strings to URLs (PyPi). Notice, after we 'get' the weather data, which we place in a variable called **response**, and print **response** we see **<Response [200]>.** The 200 is the status code for that web address, which means that the Jupyter **response** status is okay. If the status is in the 400's, then an error occurred. Notice, after we convert the data to JSON format and print it, we return data that is in a dictionary format like MongoDB.

Taylor Shrode
Using APIs and Data Architecture
2/16/2020
MSDS 610

```
import requests as req
```

```
response = req.get('https://www.metaweather.com/api/location/2391279/')
```

```
response
```

```
]: <Response [200]>
```

```
response.json()
```


```
]: {'consolidated_weather': [{'id': 4570164722925568,
     'weather_state_name': 'Light Cloud',
     'weather_state_abbr': 'lc',
     'wind_direction_compass': 'ESE',
     'created': '2020-02-13T15:49:11.816291Z',
     'applicable_date': '2020-02-13',
     'min_temp': -10.170000000000002,
     'max_temp': 0.09,
     'the_temp': -2.515,
     'wind_speed': 2.5851128575174314,
     'wind_direction': 121.39250037145105,
     'air_pressure': 1023.5,
     'humidity': 74,
     'visibility': 12.155263262546725,
     'predictability': 70},
    {'id': 4831712393560064,
     'weather_state_name': 'Clear',
     'weather_state_abbr': 'c',
     'wind_direction_compass': 'SSW',
     'created': '2020-02-13T15:49:12.137127Z',
```


We can see when the data is in JSON format that the data is in a curly bracket, then has a 'key', and a value pair. Notice that it has a name key of **consolidated weather** and then a list of JSON objects with different dates. Below, we can see the output of **response.json()['consolidated_weather'].**

```
response.json()['consolidated_weather']
```


```
]: [{'id': 4570164722925568,
    'weather_state_name': 'Light Cloud',
    'weather_state_abbr': 'lc',
    'wind_direction_compass': 'ESE',
    'created': '2020-02-13T15:49:11.816291Z',
    'applicable_date': '2020-02-13',
    'min_temp': -10.170000000000002,
    'max_temp': 0.09,
    'the_temp': -2.515,
    'wind_speed': 2.5851128575174314,
    'wind_direction': 121.39250037145105,
    'air_pressure': 1023.5,
    'humidity': 74,
    'visibility': 12.155263262546725,
    'predictability': 70},
   {'id': 4831712393560064,
    'weather_state_name': 'Clear',
    'weather_state_abbr': 'c',
    'wind_direction_compass': 'SSW',
    'created': '2020-02-13T15:49:12.137127Z',
    'applicable_date': '2020-02-14',
    'min_temp': -7.9,
    'max_temp': 5.73,
    'the_temp': 3.375,
    'wind_speed': 5.232551504652828,
    'wind_direction': 202.83326608975213,
```


The next thing we need to do is store our data in a database so we can collect it and make it easily available for querying. An easy way to do this is by using pandas and then to insert the data into MongoDB. Thus, we need to import the pandas database as **pd**. Then, we can use the **json_normalize** function to normalize the data.

```
import pandas as pd
```

```
df = pd.io.json.json_normalize(response.json())
```

```
df.head()
```

	consolidated_weather	time	sun_rise	sun_set	timezone_name	sources	title	location_type	woeid	
0	[{'id': 4570164722925568, 'weather_state_name'...	2020-02-13T11:11:16.061781-07:00	2020-02-13T06:55:22.590420-07:00	2020-02-13T17:32:18.413301-07:00	LMT	[{'title': 'BBC', 'slug': 'bbc', 'url': 'http:...	Denver	City	2391279	39.740009

Except, we won't be able to normalize the data this way because the data has the **consolidated_weather** as a primary key and then a list of days. Instead, we will get the data for all the days and store it in a variable called **days**. We can then look at the data for a specific day by using **days[]**.

```python
days = response.json()['consolidated_weather']
```

```python
days[0]
```

```
]: {'id': 4570164722925568,
    'weather_state_name': 'Light Cloud',
    'weather_state_abbr': 'lc',
    'wind_direction_compass': 'ESE',
    'created': '2020-02-13T15:49:11.816291Z',
    'applicable_date': '2020-02-13',
    'min_temp': -10.170000000000002,
    'max_temp': 0.09,
    'the_temp': -2.515,
    'wind_speed': 2.5851128575174314,
    'wind_direction': 121.39250037145105,
    'air_pressure': 1023.5,
    'humidity': 74,
    'visibility': 12.155263262546725,
    'predictability': 70}
```

```python
days[1]
```

```
]: {'id': 4831712393560064,
    'weather_state_name': 'Clear',
    'weather_state_abbr': 'c',
    'wind_direction_compass': 'SSW',
    'created': '2020-02-13T15:49:12.137127Z',
    'applicable_date': '2020-02-14',
    'min_temp': -7.9,
    'max_temp': 5.73,
    'the_temp': 3.375,
    'wind_speed': 5.232551504652828,
    'wind_direction': 202.83326608975213,
    'air_pressure': 1012.0,
    'humidity': 79,
    'visibility': 14.136194623399348,
    'predictability': 68}
```

Now, we can use the **json_normalize** function to normalize the data. This gives us a clean dataframe where all the information is in the dataframe. Next, we have to go through the rest of the days and add the data to our normalized dataframe by using a for loop and the **df.append()** function which will loop through the **days** array and append each day to the date frame. To confirm the data was added to the dataframe, we can use **df.shape** to see the dimensions of the array and then print out the dataframe to see the data.

```python
#create days array
df = pd.io.json.json_normalize(days[0])
```

```python
df.head()
```

	id	weather_state_name	weather_state_abbr	wind_direction_compass	created	applicable_date	min_temp	max_temp	the_te
0	4570164722925568	Light Cloud	lc	ESE	2020-02-13T15:49:11.816291Z	2020-02-13	-10.17	0.09	-2.

```python
#for Loop to Loop through days array
for day in days[1:]:
    df = df.append(pd.io.json.json_normalize(day))
```

```python
df.shape
```

```
(6, 15)
```

```python
df
```

	id	weather_state_name	weather_state_abbr	wind_direction_compass	created	applicable_date	min_temp	max_temp	the_te
0	4570164722925568	Light Cloud	lc	ESE	2020-02-13T15:49:11.816291Z	2020-02-13	-10.170	0.090	-2.
0	4831712393560064	Clear	c	SSW	2020-02-13T15:49:12.137127Z	2020-02-14	-7.900	5.730	3.
0	6521043290161152	Light Cloud	lc	SSE	2020-02-13T15:49:15.800817Z	2020-02-15	-6.700	3.855	2.
0	5372683120279552	Light Cloud	lc	SSW	2020-02-13T15:49:18.122812Z	2020-02-16	-3.560	8.905	6.
0	5429084026830848	Snow	sn	E	2020-02-13T15:49:21.808846Z	2020-02-17	-3.475	7.065	5.
0	6344851853410304	Snow	sn	NNE	2020-02-13T15:49:24.044755Z	2020-02-18	-8.705	-1.135	-4.

We then use **df.info** to tell us the summary of the dataframe and see what the ‘type’ of data is in each column.

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6 entries, 0 to 0
Data columns (total 15 columns):
id                        6 non-null int64
weather_state_name        6 non-null object
weather_state_abbr        6 non-null object
wind_direction_compass    6 non-null object
created                   6 non-null object
applicable_date           6 non-null object
min_temp                  6 non-null float64
max_temp                  6 non-null float64
the_temp                  6 non-null float64
wind_speed                6 non-null float64
wind_direction            6 non-null float64
air_pressure              6 non-null float64
humidity                  6 non-null int64
visibility                6 non-null float64
predictability            6 non-null int64
dtypes: float64(7), int64(3), object(5)
memory usage: 768.0+ bytes
```

Notice that the columns ‘created’ and ‘applicable_date’ are classified as an object, which means it’s a string. We want those columns in actual date/time format. The ‘applicable_date’ is just the data, but 'created' has a time zone at the end. The time zone, Z, means Zulu time, which is UTC or GMT time zone. First, we use the pandas **pd.to_datetime** function and use **utc=True** to convert the time zone. Next, we use the same **pd.to_datetime** function to convert the 'applicable_date' column and use **dt.tz_localize** to change to 'US/Mountain' time zone. Also, we can drop any unnecessary columns, such as 'weather_state_abbr' and 'id'. The **inplace=True** just means we drop them from the dataframe without creating a new dataframe and **axis=1** drops the columns and not the rows. To confirm these changes, we use **df.info()**, or we can view the dataframe in tabular format by using **df**.

```
#clean up df datatypes
#Z means Zulu time, or GMT or UTC
#format dates
```

```
df['created'] = pd.to_datetime(df['created'], utc=True)
```

```
df['applicable_date'] = pd.to_datetime(df['applicable_date']).dt.tz_localize('US/Mountain')
```

```
#drop unnecessary columns
df.drop(['weather_state_abbr','id'], inplace=True, axis =1)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6 entries, 0 to 0
Data columns (total 13 columns):
weather_state_name       6 non-null object
wind_direction_compass   6 non-null object
created                  6 non-null datetime64[ns, UTC]
applicable_date          6 non-null datetime64[ns, US/Mountain]
min_temp                 6 non-null float64
max_temp                 6 non-null float64
the_temp                 6 non-null float64
wind_speed               6 non-null float64
wind_direction           6 non-null float64
air_pressure             6 non-null float64
humidity                 6 non-null int64
visibility               6 non-null float64
predictability           6 non-null int64
dtypes: datetime64[ns, US/Mountain](1), datetime64[ns, UTC](1), float64(7), int64(2), object(2)
memory usage: 672.0+ bytes
```

Now that we have a clean pandas dataframe, we can connect to MongoDB and put the data there. We connect to the database by importing MongoClient and then creating the clients and connecting to the database and collection.

```
from pymongo import MongoClient
```

```
client = MongoClient()
db = client['weather_test']
collection = db['denver']
```

In order to store this from a dataframe to pandas, we can use the **df.to_dict** function and give it the argument **records,** which will give us a list of JSON objects. Like above, but now we have the times formatted differently. This allows us to drop the data into MongoDB easily.

```
df.to_dict('records')
```

```
[{'weather_state_name': 'Light Cloud',
  'wind_direction_compass': 'ESE',
  'created': Timestamp('2020-02-13 15:49:11.816291+0000', tz='UTC'),
  'applicable_date': Timestamp('2020-02-13 00:00:00-0700', tz='US/Mountain'),
  'min_temp': -10.170000000000002,
  'max_temp': 0.09,
  'the_temp': -2.515,
  'wind_speed': 2.5851128575174314,
  'wind_direction': 121.39250037145105,
  'air_pressure': 1023.5,
  'humidity': 74,
  'visibility': 12.155263262546725,
  'predictability': 70},
 {'weather_state_name': 'Clear',
  'wind_direction_compass': 'SSW',
  'created': Timestamp('2020-02-13 15:49:12.137127+0000', tz='UTC'),
  'applicable_date': Timestamp('2020-02-14 00:00:00-0700', tz='US/Mountain'),
  'min_temp': -7.9
```

To add the data to MongoDB, we use the **collection.insert_many** function with our **df.to_dict('records').** To view an entry from the collection in MongoDB, we use **collection.find_one()**.

```
collection.insert_many(df.to_dict('records'))
```

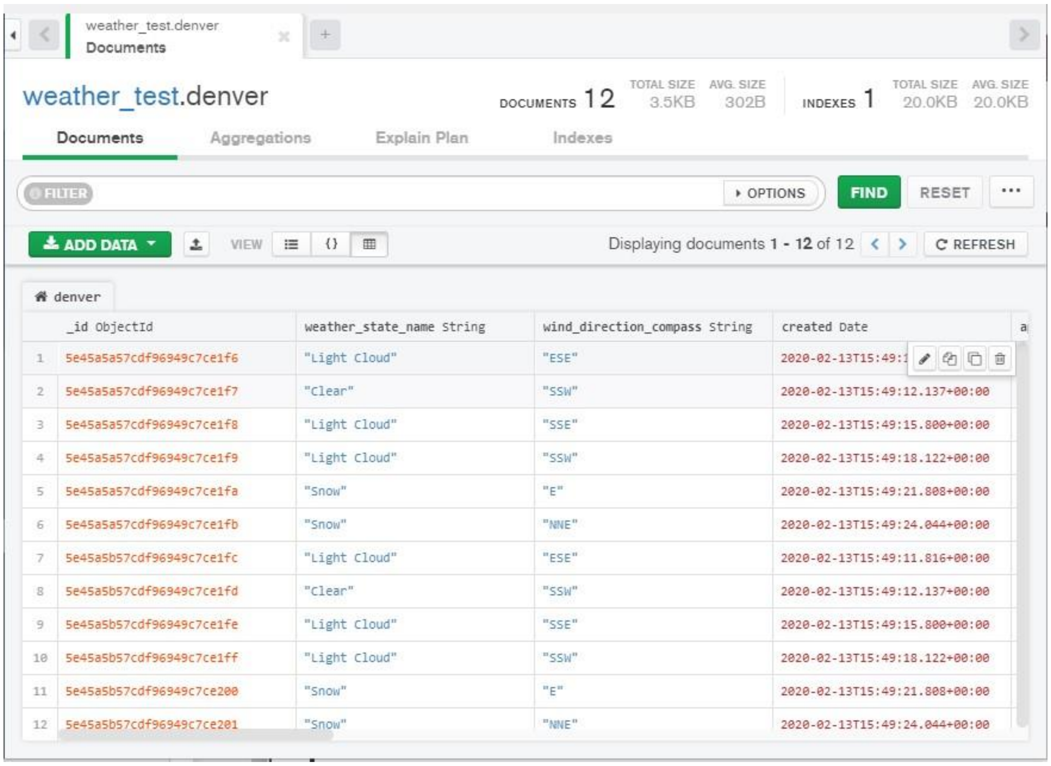
```
<pymongo.results.InsertManyResult at 0x21d60b46288>
```

```
collection.find_one()
```

```
{'_id': ObjectId('5e45a5a57cdf96949c7ce1f6'),
 'weather_state_name': 'Light Cloud',
 'wind_direction_compass': 'ESE',
 'created': datetime.datetime(2020, 2, 13, 15, 49, 11, 816000),
 'applicable_date': datetime.datetime(2020, 2, 13, 7, 0),
 'min_temp': -10.170000000000002,
 'max_temp': 0.09,
 'the_temp': -2.515,
 'wind_speed': 2.5851128575174314,
 'wind_direction': 121.39250037145105,
 'air_pressure': 1023.5,
 'humidity': 74,
 'visibility': 12.155263262546725,
 'predictability': 70}
```

To further confirm that our data was imported to MongoDB, we can open MongoDB Compass to view the data.

Taylor Shrode
Using APIs and Data Architecture
2/16/2020
MSDS 610



	_id ObjectId	weather_state_name String	wind_direction_compass String	created Date
1	5e45a5a57cdf96949c7ce1f6	"Light Cloud"	"ESE"	2020-02-13T15:49:1
2	5e45a5a57cdf96949c7ce1f7	"Clear"	"SSW"	2020-02-13T15:49:12.137+00:00
3	5e45a5a57cdf96949c7ce1f8	"Light Cloud"	"SSE"	2020-02-13T15:49:15.800+00:00
4	5e45a5a57cdf96949c7ce1f9	"Light Cloud"	"SSW"	2020-02-13T15:49:18.122+00:00
5	5e45a5a57cdf96949c7ce1fa	"Snow"	"E"	2020-02-13T15:49:21.808+00:00
6	5e45a5a57cdf96949c7ce1fb	"Snow"	"NNE"	2020-02-13T15:49:24.044+00:00
7	5e45a5b57cdf96949c7ce1fc	"Light Cloud"	"ESE"	2020-02-13T15:49:11.816+00:00
8	5e45a5b57cdf96949c7ce1fd	"Clear"	"SSW"	2020-02-13T15:49:12.137+00:00
9	5e45a5b57cdf96949c7ce1fe	"Light Cloud"	"SSE"	2020-02-13T15:49:15.800+00:00
10	5e45a5b57cdf96949c7ce1ff	"Light Cloud"	"SSW"	2020-02-13T15:49:18.122+00:00
11	5e45a5b57cdf96949c7ce200	"Snow"	"E"	2020-02-13T15:49:21.808+00:00
12	5e45a5b57cdf96949c7ce201	"Snow"	"NNE"	2020-02-13T15:49:24.044+00:00

We can see that our data was imported successfully. As we have demonstrated above, we have retrieved data from a web API and stored it in MongoDB using Python. The website MetaWeather provides a free and open-access API that gets weather forecast data. After retrieving the data from the API, we cleaned the data and then we stored it in MongoDB which allows us to easily query the data later if we wish.

Taylor Shrode
Using APIs and Data Architecture
2/16/2020
MSDS 610

Resources

Beal, V. (n.d.). *What is api - application program interface? Webopedia definition*. Retrieved February 15, 2020, from https://www.webopedia.com/TERM/A/API.html

Eising, P. (2017, December 7). *What exactly is an api?* Medium. https://medium.com/@perrysetgo/what-exactly-is-an-api-69f36968a41f

Hoffman, C. (2018, March 21). *What is an api?* How-To Geek. https://www.howtogeek.com/343877/what-is-an-api/

Joshi, V. (2016, August 4). *Relational vs. NoSQL Databases for API Traffic*. https://dzone.com/articles/relational-vs-nosql-databases-for-api-traffic-1

Kholod, A. (2016, October 25). 6 difficulties in api integration and the way to avoid them. *API2Cart - Unified Shopping Cart Data Interface*. https://api2cart.com/business/6-difficulties-api-integration-way-avoid/

Reitz, K. (2019). *Requests: Python http for humans.* [Python]. http://python-requests.org

Wodehouse, C. (2016, September 26). *Intro to apis: What are they & what do they do?* Hiring Headquarters. https://www.upwork.com/hiring/development/intro-to-apis-what-is-an-api/

Zell. (2018, January 17). *Understanding and using rest apis*. Smashing Magazine. https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/