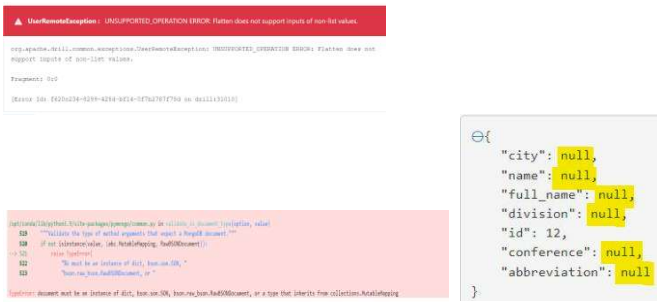
IST 769
Matthew Belizaire
Melissa Mosier
Nicholas Nguyen

NBA Analysis

API LINK: https://www.balldontlie.io/home.html#introduction

Pipeline

1. Apache Drill
    a. Gather data through web API/URL
    b. Configure our own HTTP in Drill to get data through API/URL
2. PySpark
    a. Will create data frames using Spark
    b. Use PySpark as a connector to other tools
3. MongoDB
    a. Create database (balldontlie) in Mongo Express
    b. Using Spark, we will load the json data into MongoDB
    c. From the Mongo client and Drill we will gather the data necessary to answer our analysis questions through queries
4. Neo4j
    a. Import data into Neo4j
    b. After creating data frames with Spark, can create a graph database with Neo4j
    c. Need to connect to Neo4j through Spark configuration
    d. See relationships between data points and will have a final database for data

How it started	How it's going
APACHE DRILL --> PySpark --> mongoDB --> neo4j 	

Data

For this project we decided to get our data from https://www.balldontlie.io/home.html#introduction. This site had multiple APIs that gave us various NBA data. We thought this would be great for this project as it would give us depth and detail of multiple categories within the NBA and the players. The APIs we decided to choose from the site are the players, teams, and statistics APIs.

Tools

For our project we will be using 4 different tools that we learned throughout the class: Apache Drill, MongoDB, PySpark, and Neo4j. We will be saving the data that we get from our API into MongoDB and Neo4j, while using Apache Drill and PySpark to query data, answer questions and make data frames.

1. Apache Drill

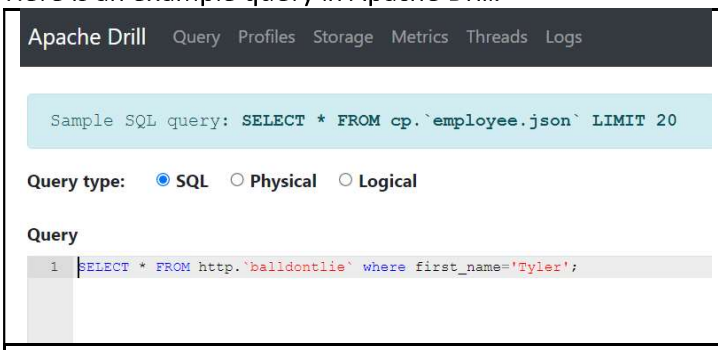
To begin we wanted to use a tool that could gather the API data and be able to display/query quickly. For this we decided to use Apache Drill. Once we got the data into Apache, we believed that we would be able to query some of our business questions, create tables and be able to connect with other tools used throughout the class. The most difficult part of using Apache Drill was getting our configuration correct. We started our configuration from scratch and had some trouble in the beginning. We thought by entering in the URL/API and the params the configuration would work, but when we tried to query it would turn up null values. After some troubleshooting, we figured out we needed to enter in the correct parameters and dataPath in the configuration. With that, our configuration was able to pull all the necessary data from the API.

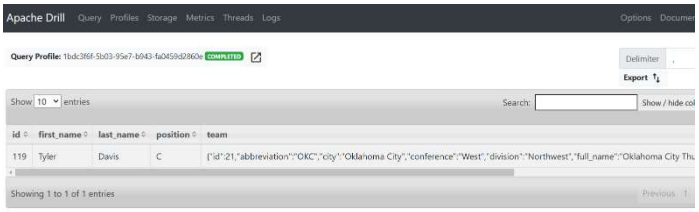
We used the following configuration in Apache Drill to access data from "balldontlie"

{
 "type": "http",
 "cacheResults": true,
 "connections": {
 "balldontlie": {
 "url": "https://www.balldontlie.io/api/v1/players",
 "method": "GET",
 "headers": null,
 "authType": "none",
 "userName": null,
 "password": null,
 "postBody": null,
 "params": [
 "id",
 "dates",
 "seasons",
	Notes:
	
	 For the highlighted portion, changing it would give us access to different datasets in the API. For example, changing "players" to "stats" would give us different data.
	
	 However, datasets had nested data in columns, and that had to be resolved.

"player_ids",
"game_ids",
"postseason",
"start_date",
"end_date"
],
"dataPath": "data",
"requireTail": false,
"inputType": "json",
"xmlDataLevel": 1
}
},
"proxyType": "direct",
"enabled": true
}

Here is an example query in Apache Drill.





"Un-Nest"ing the data In the "team" column, required us to alias the file in Drill.

Ex:

```
select
    g.id,
    g.team.city as teamcity,
    g.team.name as teamname
    from http.`balldontlie` g
```

Extra Drill Queries

Locating specific player with all attributes:

Apache Drill Query Profiles Storage Metrics Threads Logs

Sample SQL query: **SELECT * FROM cp.`employee.json` LIMIT 20**

Query type: ◉ **SQL** ○ **Physical** ○ **Logical**

Query

```
SELECT id, first_name, last_name, position, height_feet, height_inches, weight_pounds, t.team.full_name as team_name
FROM http.`balldontlie` t
WHERE search = 'LeBron'
```

Show 10 entries Search: Show / hide columns

id	first_name	last_name	position	height_feet	height_inches	weight_pounds	team_name
237	LeBron	James	F	6	8	250	Los Angeles Lakers

Showing 1 to 1 of 1 entries Previous 1 Next

Taking a sample of players from the first few pages and counting the total number of players at each position. There were mostly nulls which we filtered out:

Query

```
SELECT DISTINCT position, COUNT(*) AS player_count
FROM http.balldontlie
WHERE page = 0 and position IS NOT NULL and position <> ''
GROUP BY position
UNION
SELECT DISTINCT position, COUNT(*) AS player_count
FROM http.balldontlie
WHERE page = 1 and position IS NOT NULL and position <> ''
GROUP BY position
UNION
SELECT DISTINCT position, COUNT(*) AS player_count
FROM http.balldontlie
WHERE page = 2 and position IS NOT NULL and position <> ''
GROUP BY position
```

Show 10 entries

position	player_count
C	4
G	10
F	4
C-F	1
F-C	1
G-F	1

Showing 1 to 6 of 6 entries

Counting the number of teams in each division:

Query

```
SELECT division, COUNT(*) AS team_count
FROM http2.balldontlie2
GROUP BY division
```

Show 10 entries

division	team_count
Central	5
Southwest	5
Atlantic	5
Pacific	5
Northwest	5
Southeast	5

Finding the date, home team name and score, and visitor team name and score for games that occurred in the 2018-2019 season

Query

```
SELECT `date`, g.home_team.name AS home_team_name, home_team_score, g.visitor_team.name AS visitor_team_name, visitor_team_score
FROM http3.`balldontlie3` g
WHERE season = 2018 and per_page = 100
ORDER BY `date`
```

Show 10 entries Search:

date	home_team_name	home_team_score	visitor_team_name	visitor_team_score
2019-01-30T00:00:00.000Z	Celtics	126	Hornets	94
2019-02-08T00:00:00.000Z	Wizards	119	Cavaliers	106
2019-02-08T00:00:00.000Z	76ers	117	Nuggets	110
2019-02-08T00:00:00.000Z	Kings	102	Heat	96
2019-02-08T00:00:00.000Z	Pelicans	122	Timberwolves	117
2019-02-09T00:00:00.000Z	Bucks	83	Magic	103
2019-02-09T00:00:00.000Z	Rockets	112	Thunder	117
2019-02-09T00:00:00.000Z	Celtics	112	Clippers	123
2019-02-09T00:00:00.000Z	Pacers	105	Cavaliers	90
2019-02-09T00:00:00.000Z	Hawks	120	Hornets	129

Showing 1 to 10 of 100 entries Previous 1 2 3

Finding how well players named Bill shot from the free throw line in a game.

Query

```
SELECT fta, ftm, ft_pct, player
FROM http4.`balldontlie4` p
WHERE p.player.first_name = 'Bill'
```

Show 10 entries

fta	ftm	ft_pct	player
1	0	0.0	{"id":46400220,"first_name":"Bill","last_name":"McGill","position":"","team_id":30}
4	2	0.5	{"id":46396110,"first_name":"Bill","last_name":"Russell","position":"","team_id":2}

Showing 1 to 2 of 2 entries

Although we were able to pull and query the Balldontlie data from the API in Apache Drill there were some problems with using just Drill. First, we could not query all the data we wanted at once. Because of default parameters already set with the API, searches can only be done at max 100 at a time per page. There were hundreds of pages that we wanted to query with, so this was inefficient. Next was saving the data. Although we were able to query and get some answers there was no official database that we could save our API in. Because of this we knew we would have to use a database. This is where our next tool came into play, MongoDB.

2. PySpark / MongoDB

Since we could not store our data into Apache Drill, we decided to create a database using MongoDB. Inside of our database we created collections that would allow us to use all the APIs within Balldontlie. Using all the APIs will be essential to answering our business questions.

We created a database called balldontlie:

```
> show databases;
admin        0.000GB
balldontlie  0.000GB
config       0.000GB
labf         0.000GB
local        0.000GB
```

Collections:

```
> show collections;
Players
Stats
Teams
delete_me
```

Below is the configuration we used to connect Spark to Mongo.

```
# Spark init
# MONGO CONFIGURATION
mongo_uri = "mongodb://admin:mongopw@mongo:27017/balldontlie.feedback?authSource=admin"
spark = SparkSession \
    .builder \
    .master("local") \
    .appName('jupyter-pyspark') \
      .config("spark.mongodb.input.uri", mongo_uri) \
      .config("spark.mongodb.output.uri", mongo_uri) \
      .config("spark.jars.packages","org.mongodb.spark:mongo-spark-connector_2.12:3.0.1")\
    .getOrCreate()
sc = spark.sparkContext
sc.setLogLevel("ERROR")
```

We created an API endpoint so it would gather NBA data. With each section of the API, we wrote them to their proper collection. Since there were multiple APIs that we wanted to use, we used the same code below for each URL.

```python
url = 'https://www.balldontlie.io/api/v1/players'
response = requests.get(url)
data = response.text
if response.status_code != 200:
    print('Failed to get data:', response.status_code)
else:
    print('First 100 characters of data are')
    print(data[:100])
```

```
First 100 characters of data are
{"data":[{"id":14,"first_name":"Ike","height_feet":null,"height_inches":null,"last_name":"Anigbogu",
```

```python
json1 = response.json()
json1
```

```
{'data': [{'id': 14,
   'first_name': 'Ike',
   'height_feet': None,
   'height_inches': None,
   'last_name': 'Anigbogu',
   'position': 'C',
   'team': {'id': 12,
    'abbreviation': 'IND',
    'city': 'Indiana',
    'conference': 'East',
    'division': 'Central',
    'full_name': 'Indiana Pacers',
    'name': 'Pacers'},
   'weight_pounds': None},
```

This was where a problem arose while trying to enter our API data into MongoDB. Since it was not a .csv or .json file like in our homework, we had to find a different way of putting the data into a collection. This took some time as we had to do a significant amount of troubleshooting to get it to work. The biggest problem that arose was that most of the values would return null when initially put into a collection or data frame. This was troublesome because we could not query any information and barely any data was being shown.

first_name	height_feet	height_inches	id	last_name	position	team	weight_pounds
Ike	null	null	14	Anigbogu	C	{division -> null...	null
Ron	null	null	25	Baker	G	{division -> null...	null
Jabari	null	null	47	Bird	G	{division -> null...	null
MarShon	null	null	67	Brooks	G	{division -> null...	null
Lorenzo	null	null	71	Brown	G	{division -> null...	null

We were able to resolve this by editing the original schema. By changing the schema to their proper data types, the values will be able to enter data frames with .createDataFrame. Next, we used Spark to write and append the data frames into our collection. Once the data was put into the right database and collection, we were able to query anything we would like.

```
Test_Schema = StructType([
    StructField("id", IntegerType(), True),
    StructField("first_name", StringType(), True),
    StructField("last_name", StringType(), True),
    StructField("position", StringType(), True),
    StructField("height_feet", IntegerType(), True),
    StructField("height_inches", IntegerType(), True),
    StructField("weight_pounds", IntegerType(), True),
    StructField("team", StructType([
        StructField("id", IntegerType(), True),
        StructField("abbreviation", StringType(), True),
        StructField("city", StringType(), True),
        StructField("conference", StringType(), True),
        StructField("division", StringType(), True),
        StructField("full_name", StringType(), True),
        StructField("name", StringType(), True)]), True)
])
```

```
df = spark.createDataFrame(json1['data'], schema=Test_Schema)
ball2 = df.write.format("mongo").mode("append").option("database","balldontlie").option("collection","Stats").save()
```

```
df.show()
```

```
[Stage 0:>                                                          (0 + 1) / 1]
+------------+------------+----------+---------+--------------------+---+------------+
|abbreviation|        city|conference| division|           full_name| id|        name|
+------------+------------+----------+---------+--------------------+---+------------+
|         ATL|     Atlanta|      East|Southeast|       Atlanta Hawks|  1|       Hawks|
|         BOS|      Boston|      East| Atlantic|      Boston Celtics|  2|     Celtics|
|         BKN|    Brooklyn|      East| Atlantic|       Brooklyn Nets|  3|        Nets|
|         CHA|   Charlotte|      East|Southeast|   Charlotte Hornets|  4|     Hornets|
|         CHI|     Chicago|      East|  Central|       Chicago Bulls|  5|       Bulls|
|         CLE|   Cleveland|      East|  Central| Cleveland Cavaliers|  6|   Cavaliers|
|         DAL|      Dallas|      West|Southwest|    Dallas Mavericks|  7|   Mavericks|
|         DEN|      Denver|      West|Northwest|      Denver Nuggets|  8|     Nuggets|
|         DET|     Detroit|      East|  Central|     Detroit Pistons|  9|     Pistons|
|         GSW|Golden State|      West|  Pacific|Golden State Warr...| 10|    Warriors|
|         HOU|     Houston|      West|Southwest|     Houston Rockets| 11|     Rockets|
|         IND|     Indiana|      East|  Central|      Indiana Pacers| 12|      Pacers|
|         LAC|          LA|      West|  Pacific|         LA Clippers| 13|    Clippers|
|         LAL| Los Angeles|      West|  Pacific|  Los Angeles Lakers| 14|      Lakers|
|         MEM|     Memphis|      West|Southwest|   Memphis Grizzlies| 15|   Grizzlies|
|         MIA|       Miami|      East|Southeast|          Miami Heat| 16|        Heat|
|         MIL|   Milwaukee|      East|  Central|     Milwaukee Bucks| 17|       Bucks|
|         MIN|   Minnesota|      West|Northwest|Minnesota Timberw...| 18|Timberwolves|
|         NOP| New Orleans|      West|Southwest|New Orleans Pelicans| 19|    Pelicans|
|         NYK|    New York|      East| Atlantic|     New York Knicks| 20|      Knicks|
+------------+------------+----------+---------+--------------------+---+------------+
```

Since our API had thousands of pages and millions of documents/data points, we knew we would have to come up with an efficient way to gather all that information. This is when we came up with a “for loop” below and were able to scrape all the NBA data into its proper collection in MongoDB. This was done with all other APIs to fill out our database and the collections.

```python
for i in range(3,15446):
    url = 'https://www.balldontlie.io/api/v1/stats?page=i&per_page=100'
    response = requests.get(url)
    data = response.text
    if response.status_code != 200:
        print('Failed to get data:', response.status_code)
    else:
        print('First 100 characters of data are')
        print(data[:100])
    json1 = response.json()
    df = spark.createDataFrame(json1['data'], schema=Test_Schema)
    ball2 = df.write.format("mongo").mode("append").option("database","balldontlie").option("collection","Stats").save()
```

```
First 100 characters of data are
{"data":[{"id":13462423,"ast":null,"blk":null,"dreb":null,"fg3_pct":null,"fg3a":null,"fg3m":null,"fg
```

Before and after schema change results of querying through MongoDB

```
> db.Players.find({first_name: 'Ike'})
{ "_id" : ObjectId("6427bb7f16036d7728d0ba9f"), "first_name" : "Ike", "id" : NumberLong(14), "last_name" : "Anigbogu", "position" : "C", "team" : { "city" :
 null, "name" : null, "full_name" : null, "division" : null, "id" : NumberLong(12), "conference" : null, "abbreviation" : null } }
```

Before Schema Change

```
> db.Players.find({first_name: "Ike"})
{ "_id" : ObjectId("64292500b790bb388a4e0be9"), "id" : 14, "first_name" : "Ike", "last_name" : "Anigbogu", "position" :
"C", "team" : { "id" : 12, "abbreviation" : "IND", "city" : "Indiana", "conference" : "East", "division" : "Central", "f
ull_name" : "Indiana Pacers", "name" : "Pacers" } }
```

After Schema Change

What our documents look like in their collections inside of MongoDB Express

_id	abbreviation	city	conference	division	full_name	id	name
642914c0b790bb388a4e0afa	ATL	Atlanta	East	Southeast	Atlanta Hawks	1	Hawks
642914c0b790bb388a4e0afb	BOS	Boston	East	Atlantic	Boston Celtics	2	Celtics
642914c0b790bb388a4e0afc	BKN	Brooklyn	East	Atlantic	Brooklyn Nets	3	Nets
642914c0b790bb388a4e0afd	CHA	Charlotte	East	Southeast	Charlotte Hornets	4	Hornets
642914c0b790bb388a4e0afe	CHI	Chicago	East	Central	Chicago Bulls	5	Bulls
642914c0b790bb388a4e0aff	CLE	Cleveland	East	Central	Cleveland Cavaliers	6	Cavaliers
642914c0b790bb388a4e0b00	DAL	Dallas	West	Southwest	Dallas Mavericks	7	Mavericks
642914c0b790bb388a4e0b01	DEN	Denver	West	Northwest	Denver Nuggets	8	Nuggets
642914c0b790bb388a4e0b02	DET	Detroit	East	Central	Detroit Pistons	9	Pistons
642914c0b790bb388a4e0b03	GSW	Golden State	West	Pacific	Golden State Warriors	10	Warriors

Teams Collections

_id	id	ast	blk	dreb	fg3_pct	fg3a	fg3m	fg_pct	fga	fgm	ft_pct	fta	ftm
642939402b3bde4cda650eac	11264479	4	0	6	0.4000000059604645	5	2	0.4169999957084656	12	5	1	2	2

pf	pts	reb	stl	turnover	game	player	team
2	14	7	1	3	{ "id": 474526, "date": "2022-03-27T00:00:00.000Z", "home_team_id": 3, "home_team_score": 110, "season": 2021, "visitor_team_id": 4, "visitor_team_score": 119 }	{ "id": 403, "first_name": "Terry", "last_name": "Rozier", "position": "G", "team_id": 4 }	{ "abbreviation": "CHA", "city": "Charlotte", "conference": "East", "division": "Southeast", "full_name": "Charlotte Hornets", "name": "Hornets" }

Stats Collection

_id	id	first_name	last_name	position	team
64292500b790bb388a4e0bfd	448	Gary	Trent Jr.	G	{ "id": 25, "abbreviation": "POR", "city": "Portland", "conference": "West", "division": "Northwest", "full_name": "Portland Trail Blazers", "name": "Trail Blazers" }
64292500b790bb388a4e0bfe	494	Michael	Smith		{ "id": 2, "abbreviation": "BOS", "city": "Boston", "conference": "East", "division": "Atlantic", "full_name": "Boston Celtics", "name": "Celtics" }

Player Collection

After getting all our data into MongoDB we had to make the following configuration to connect Mongo with Apache Drill.

```
{
  "type": "mongo",
  "connection": "mongodb://admin:mongopw@mongo:27017/",
  "enabled": true
}
```

Once all the data was in MongoDB and the configuration was set up, we began querying with Drill. We noticed right away how much more information we were able to get with this setup compared to the beginning when we tried to query our API straight from Drill. With Spark, Mongo and Drill we got our data faster and more efficiently.

To compare our queries before and after using MongoDB, we queried all the position player counts again. This time we are able to get every single player and their position with a shorter code. This also showed that the data did not put any positions for some of their players.

Query

```sql
SELECT DISTINCT position, COUNT(*) AS player_count
FROM mongo.balldontlie.Players
GROUP BY position
```

position	player_count
C	130
G	539
F	469
C-F	18
F-C	50
G-F	67
	3838
F-G	15

Queried players and amount of 30-point games

Query **Hint:** Use *Ctrl+Enter* to submit

```sql
select S.player.first_name as First_Name, S.player.last_name as Last_name, count (pts) as `30_pts_games`
from mongo.balldontlie.Stats as S
where pts > 30
group by First_Name, Last_name
```

First_Name	Last_name	30_pts_games
Charles	Barkley	221
Carmelo	Anthony	91
Earl	Monroe	13
Billy	Cunningham	13
Dan	Roundfield	1
Michael	Jordan	301
Nate	Archibald	1
James	Harden	128
Kevin	Johnson	357
Dale	Ellis	1

After successfully saving all our APIs into MongoDB, we realized how much better using MongoDB was. Although Drill had the capability and simplicity of getting the API with one configuration, there was no way of getting all the data at once. With the right code and configuration, using Spark with MongoDB efficiently gathered all data points from the API. Then we can configure a connection with Mongo and Drill and run better queries compared to Drill alone. Finally, to have a better visualization of our data we will be using Neo4j as our last tool.

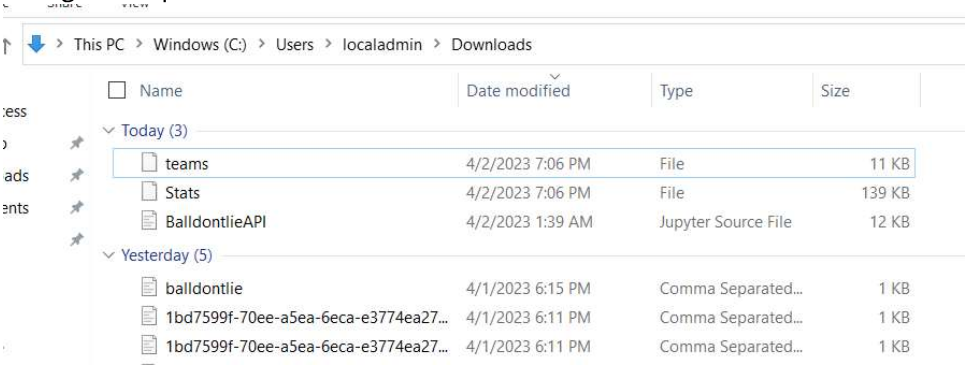
3. Neo4j

For our final tool we used Neo4j, we made a couple of attempts at creating a visualization with Neo4j.

We attempted to use a mongodb connector in Python, but we were unable to make that code work. So we tried exporting the data as csv and into Neo4j that way.

MongoDB's default export function does not export as a csv, so we tried exporting from Drill.

MongoDB export



Drill export

id	first_name	last_name	position	team_name
448	Gary	Trent Jr.	G	Portland Trail B
494	Michael	Smith		Boston Celtics
495	John	Morton		Cleveland Cava
496	Howard	Wright		Atlanta Hawks
497	Michael	Ansley		Orlando Magic

In addition to the fact that our VMs did not have Excel, we were struggling with the syntax pulling data from the csv.

```
load csv with headers from 'C:\Users\localadmin\Downloads' as line
create ( :Player {id: line.id, position: line.position, teamname: line.team_name})
```

Finally, we also tried pulling data in from the REST API directly. That also looked a few different ways.

```
CALL apoc.load.json('https://www.balldontlie.io/api/v1/players')
YIELD value

CREATE (data:data {id: value.id})
SET data.firstname = value.first_name,
    data.lastname = value.last_name,
    data.teamid = value.team.id
```

ERROR Neo.ClientError.Procedure.ProcedureCallFailed

```
Failed to invoke procedure `apoc.load.json`: Caused by: java.lang.RuntimeException: Can't read url or key
https://www.balldontlie.io/api/v1/players as json: Server returned HTTP response code: 503 for URL:
https://www.balldontlie.io/api/v1/players
```

We did eventually get Neo4j to connect to the API but it was unable to put the data into a readable table.

```
neo4j$ call apoc.load.json($url) yield value return value;
```

Started streaming 1 records after 39 ms and completed after 536 ms.

```
neo4j$ call apoc.load.json($url) yield value unwind value.items as p return p.data.first_name
```

Completed after 519 ms.

```
neo4j$ call apoc.load.json($url) yield value unwind value.items as p return p.id
```

Completed after 1555 ms.

```
neo4j$ call apoc.load.json($url) yield value unwind value.items as p return keys(p)
```

Completed after 4648 ms.

```
neo4j$ call apoc.load.json($url)
```

It was able to load the data, but as a collection of documents. We were unable, with the codes in the middle, to actually have Neo4j read the data, only return it exactly as it was streaming in. If we were to do this again, we would probably try this or the csv import to achieve displaying a graph. All things considered, however, we were able to answer our business questions with Drill and connect MongoDB to Drill.