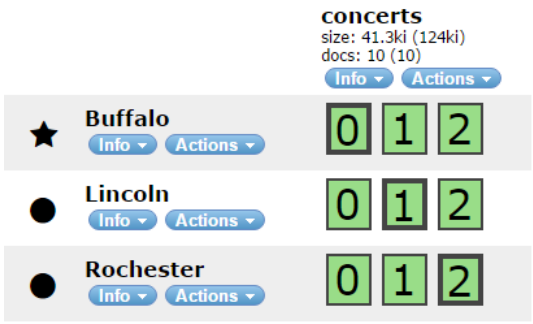
HudlU Elasticsearch Capstone Project

Val Booth

- Design Decisions:
  - **Hardware:**
    - Machine: Due to time constraints I chose to run my Elasticsearch nodes locally as opposed to being hosted on AWS. I ran my Elasticsearch nodes on the MacBook Pro given to me by Hudl, as it had over 10x faster read/write speeds to the hard drive and 4x as much RAM as my own personal Dell Inspiron N5040. Read/write speeds were timed by Blackmagic Disk Speed and CrystalDisk Mark for Mac and Windows respectively. Although my index was rather small, I wanted to give it the speed benefit of my MacBook's SSD and far superior RAM.
    - Node/Shard Number: I decided to run 3 nodes with 3 shards each, with 2 replicas per shard. Given the small nature of my index I thought 3 was a good number to make sure there were enough replicas of shards, that the primary of each shard was on its own separate node, and that the overhead wasn't too high. The default setting of 5 shards seemed like overkill for my index.



  - **Settings:**
    - File: create_index.JSON
    - Mapping: My index was a database of concerts, and for the concert mapping I decided to make the artist and friends strings, but the venue and the date nested objects. After reviewing all of the date formats on elastic.co I found I didn't favor any of them, mostly because if I was going to represent it as a string, I wanted month to be first. I made date a nested object with integer month, day, and year properties. I chose to make venue a nested object because I felt like if this *were* a relational database, it would have been it's own table, so it made more sense as a nested object here in Elasticsearch.
    - Analyzer: I had two analyzers: a custom defined analyzer named "standard", so it would be used for most fields, and a "city_custom" analyzer. Both analyzers had whitespace tokenizers, as I felt it was the most useful for searching for band names, friend names, and concert venues, which were all going to either be a single word or a phrase. I picked whitespace as opposed to "standard" because some bands ("alt-J" and "fun." specifically) have punctuation dependent names that I didn't want removed, and I wanted to keep any punctuation in cities as well. Both analyzers had lowerspace filters as all of the previously listed properties aren't case sensitive. The standard analyzer had a shingle filter, as I felt that it could be useful to search for either one word band names ("Macklemore") or phrase bandnames ("Walk the Moon"). The city_custom analyzer had a custom shingle filter that didn't return unigrams unless no shingles of length 2 could be found. This would successfully return single word city names, like "Rochester", but keep phrase cities, like "Council Bluffs", together.

- Queries
  - For my searches I tried out different aggregation queries. I put size as 0 for all my searches so that it would be more readable without the documents themselves being returned.
    - Concerts with Laura:
      - File: find_laura.json
      - Result: The query found all 8 concert documents that Laura was in.
    - Concerts by year:
      - File: concerts_by_year.json
      - Results: Query identified that I went to 2 concerts in 2012, 2 in 2013, 1 in 2014, and 7 in 2017.
    - NY concerts by year:
      - File: ny_by_year.json
      - Results: Query identified that I went to two concerts in NY in 2012 and 2013, only 1 in 2014, and 4 concerts in NY in 2015.
    - Stats about the years Adam went to concerts:
      - File: adam_stats.json
      - Results: The search for "adam" in friends had hits on 4 documents, with the min year being 2013, the may being 2015, and the average being 2014.5.
    - Stats about friend count:
      - File: friends_stats.json, my_script.groovy
      - Results: The minimum amount of friends I went to a concert with was 0, the max was 9, and the average was 2.75.
    - Distinct number of friends:
      - File: distinct_friends.json
      - Result: I've be to concerts with 19 different people.
    - Average number of friends per city:
      - File: avg_friends_per_city.json, my_script.groovy
      - Results: I went to concerts with an average of 3 people in Rochester, 2.5 people in Rochester, 5 in Council Bluffs, 0 in Lincoln, and 2 in Papillion.
- Lessons Learned: The most important lesson I learned was that data might not always be as simple as it appears. My original settings only had one analyzer, which tokenized on whitespace and had a lowercase filter with a simple shingle filter. I had tried to be careful and consider which analyzer would be best for all of the concert data I had. I was able to figure out ahead of time that I didn't want the default standard analyzer, because I didn't want it to get rid of the "-" in "alt-J", for example. My original settings looked pretty good until I reached the nested aggregation for average friends per city, and there was one entry each for "council" and "bluffs". At that point I started over and created a custom city analyzer that would return "Council Bluffs" as one entity. Aside from learning how important picking the right analyzer was for Elasticsearch, I also got to read up on more advanced ES functionality that we didn't cover in class. The groovy script I used was really simple, but I could see lots of uses for additional functionality in scripts in both searching and coming up with match numbers.