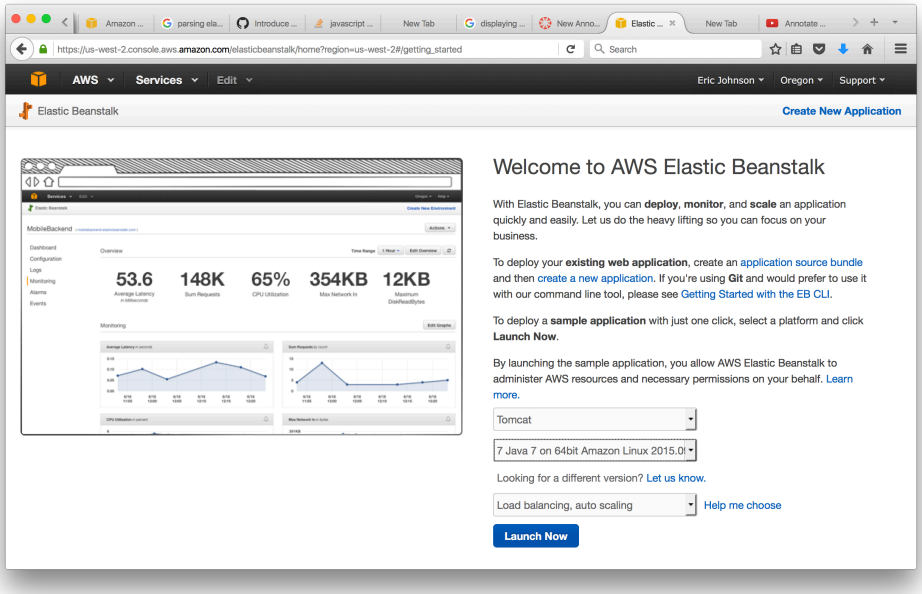
Eric Johnson efj2106
Homework Assignment 1 TweetMap

Part 1 - Create / Deploy Customized Web Application (Javascript Webite)

Step 1: Create Elastic Beanstalk Web Application

Deploy using Tomcat 8 w/ Javascript + HTML



Step 2: Customize and Upload Javascript Website

In designing the front end website I found a nice open source package called **elasticsearch.js** this will perform the AJAX request to the ElasticSearch backend and parse and return the results needed to display on the GoogleMaps front end. I designed the layout of my website so the Twitter Streaming and ElasticSearch were not running on the same box as the ElasticBeanStalk that was hosting the website. The reason for this is I found some instructions on how to create an ElasticSearch cluster with a shared ElasticIP and I thought this would scale better / independently if the webhosting server was not also bogged down by the ElasticSearch indexing.

I ran into a rather annoying problem when setting up the site. After I assigned my ElasticIP to my ElasticSearch cluster I was able to successfully get the results by curling that IP/URL through any of my AWS machines. I curled the data from the ElasticSearch index using the API instructions like so:

curl -XGET 'elastic-IP:9201/apple/_search?q=text: &size=10000&pretty=true'

Where "apple" is the name of the various indices I created.

When doing this in the Javascript portion of the website I ran into a rather annoying problem called Cross-Domain restriction where because the webserver hosting the wesbite is requesting data from an external the website rejects the get.JSON request.

I spent probably 15+ hrs debugging this and researching the issue and found that it requires you to change the CORS in your web.xml file on your website and webserver to allow Cross-Domain access. Although that solved half of the problem you must also embed a header in the return from the server you are requesting it from (my ElasticSearch cluster).

In the end I found an easier solution where I could set up a cron job and push my parsed results to an S3 bucket using the **aws-cli**. This had much easier instructions on how to enable a CORS header on the S3 bucket and it was as simple as writing a script to execute on **cron** and sync the files. This ended up putting less load on the website since I was able to return much smaller and condensed JSON for only the fields I displayed in my UI.

Note: When launching the ElasticSearch I also needed to configure the IP to not just run localhost

Modification to ElasticSearch (to publish to my ElasticIP)

```
network.bind_host: "my Elastic-IP"
```

Now I can access the results from my EC2 instance using my ElasticIP address:

```
curl –XGET "elasticIP:9201/apple/_search?q=text:&size=10000&pretty=true"
```

Part 2 - Create and Deploy ElasticSearch Index on Twitter Live Streaming

Following a guide on how to install ElasticSearch I first launch a Ubuntu Instance of AWS EC2 and then configured the EC2 instance to be accessible from my Elastic Beanstalk WebApplication by assigning it an ElasticIP and configuring it to accept HTTP requests.

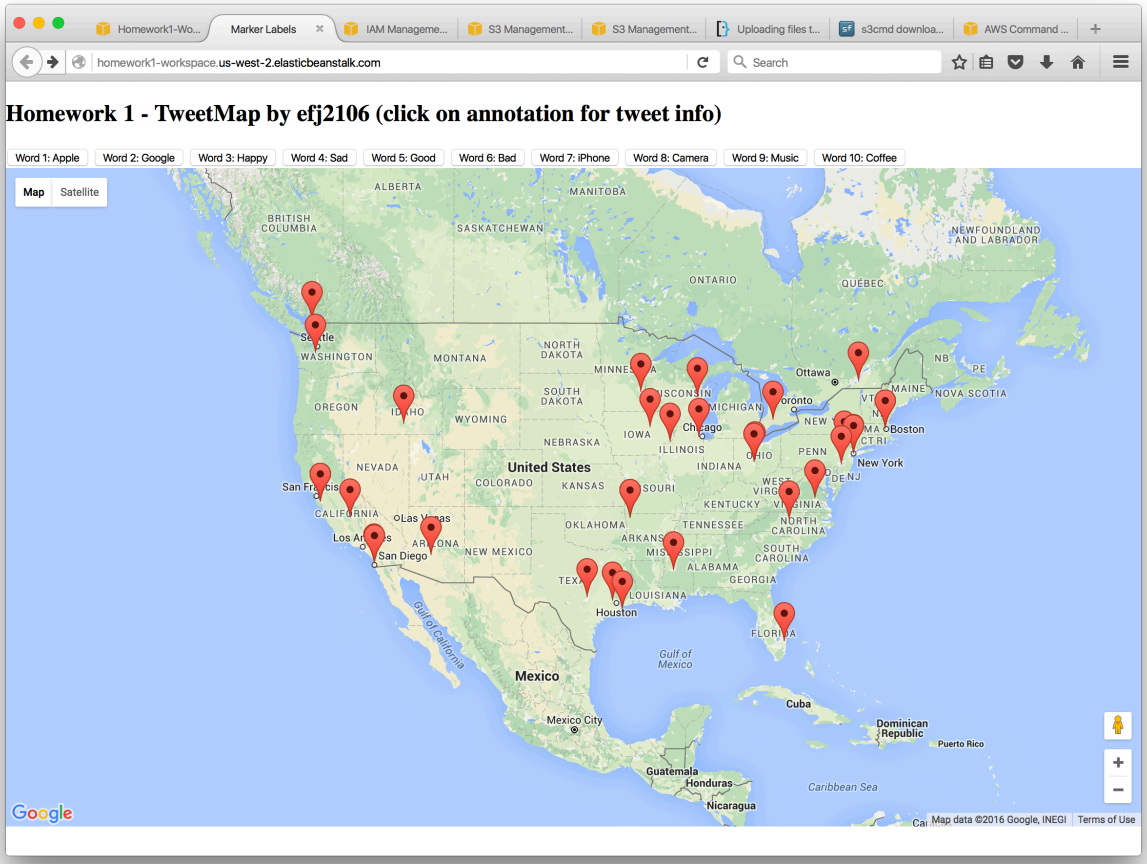
Once I had the ElasticSearch index running on my EC2 instance I installed Tweepy and configured a Python script called **twitter_tweepy.py** (included in submission) to create a Twitter Live Monitoring thread and save the results to my ElasticSearch index. The original script was designed to do keyword filtering but I modified it to do geolocation filtering as well to focus on the continental United States. I did this merely out of size constraints (my box only allows 8gb of data) and if you don't put a geolocation filter you will end up getting a large large amount of tweets with no geodata that are useless for this assignment.

Once the Twitter LiveMonitoring stream was created an populating my ElasticSearch index it was just a matter of querying an parsing the JSON results for each of my keywords in the Javascript portion of my Elastic Beanstalk WebApplication.

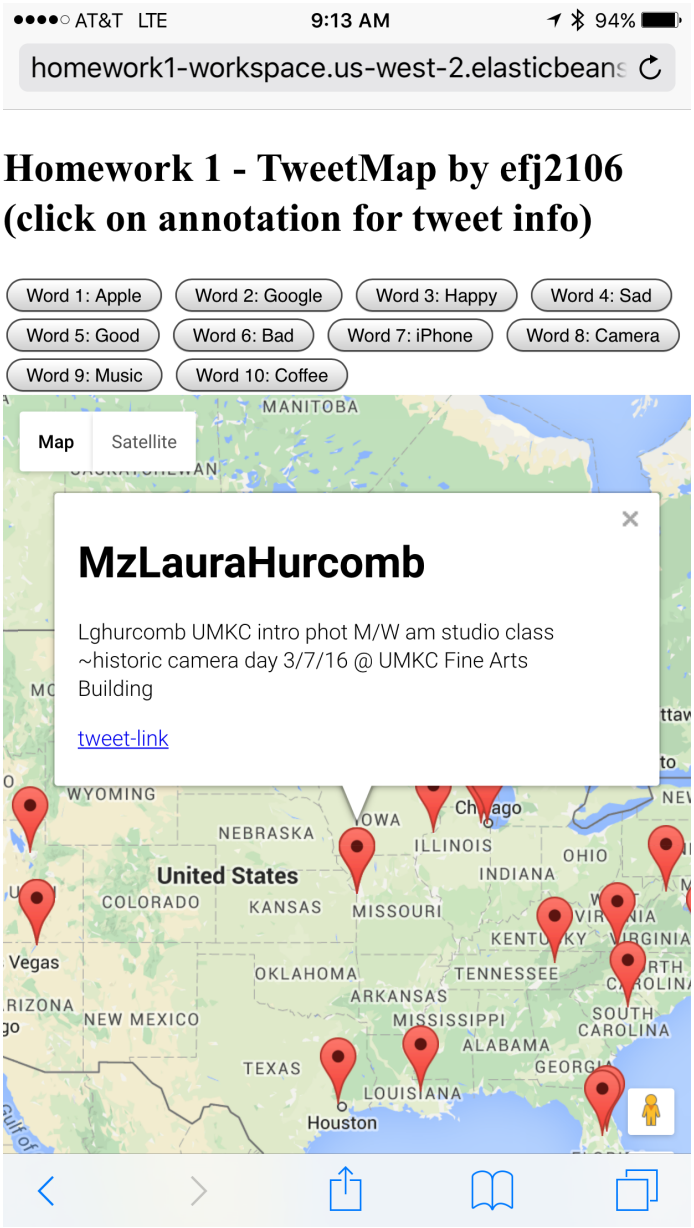
Part 3 - Google Maps API

I should note that I am very very new to Javascript so this assignment was kind of a crash course for me. The majority of my development has been on iOS or basic HTML. That being said using the example templates that ElasticBeanstalk provides I was able to modify the original container for the website to fit this assignment perfectly.

I embedded a Google Maps object in my page and create 10 buttons for each of the keywords that I wanted to display tweet content for. Then I linked each of the buttons to a function that would download, parse, and create annotation objects for each of the tweets that were returned from the S3 bucket call.

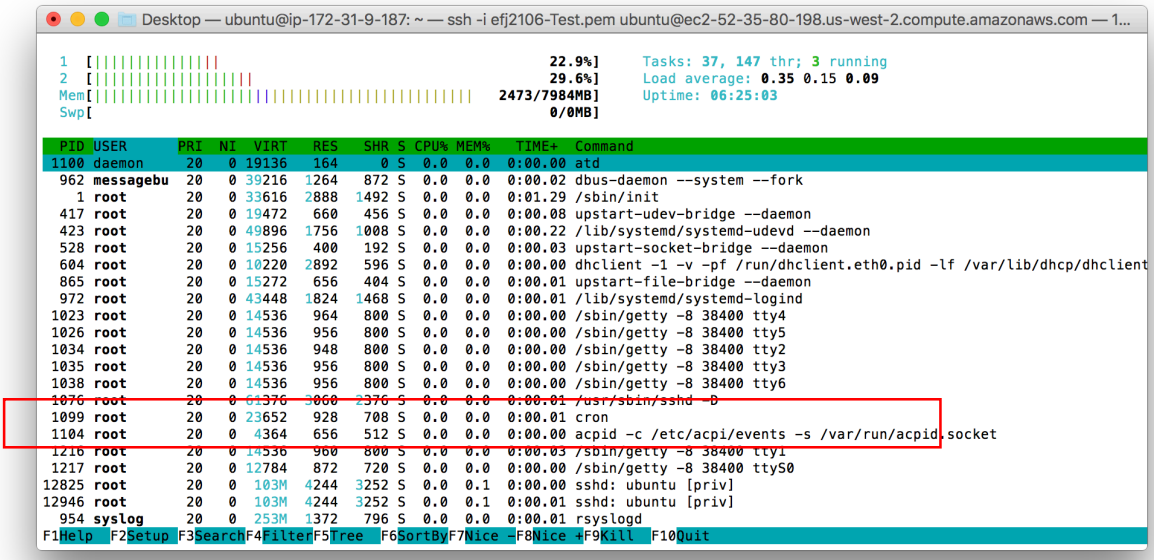


Here is actually a screenshot from my phone of the information you see when clicking on the annotation. I put the @twitter handle in bold at the top, the tweet content in the body, and then actually provide a link to that person's twitter website so you can see the tweet information and their profile. As you can see this worked pretty nicely even on a mobile phone.



****Scheduled CRON Job (every 15 minutes)**

To do the backend processing of the ElasticSearch index I created a cron job to run my **run.sh** script every 15 minutes. This download the query results from the index as text files, processes those text files in R (subsetting them to only output the data I need for my UI) and then syncs those results to my S3 bucket that the ElasticBeanStalk is using for the UI over the aws-cli.



run.sh that executes every 15 minutes to parse and sync data with S3 bucket