Twitter Spam & Fraudulent Websites

A MIDS (W205-Storing And Retrieving Data) Course Project

Kyle Hamilton, Carlos Rodriguez, & Sharmila Velamur

Table of Contents

[1. Hypothesis]
[2. Envisioned]
[3. Deliverables]
  [3.1. AMI]
  [3.2. Scripts & Code]
  [3.4. Documentation]
[4. Accomplished]
  [4.1. Option A - The Full Scope Pipeline]
[5. Release Notes]
[6. Future Work]
  [6.1. Error Recovery]
  [6.2. Scalability]
  [6.3. Enhancements]
[7. Complete Set of Instructions to Launch]
  [7.1. AWS Instructions]
  [7.2. Initial Setup Instructions]
[8. Troubleshooting Tips]
[9. Viewing the Dashboard]
[10. Installing Browser Plug-in]
[Appendix A - The Spark-Streaming App (Option B)]
[Appendix B - Data Flow At A Glance]
[Appendix C - The Final Architecture]
[Appendix D - The Streaming Solution Architecture]
[Appendix E - The Tweet Object Relationship Analysis]
[References]
[Twitter Spam]
[Development Related]

1. Hypothesis

By scanning twitter public stream data we can identify suspicious websites proliferated by alleged spammers in social media.

2. Envisioned

- Develop a big data solution to identify suspicious websites by scanning for spam in twitter public stream data.
- Develop a dashboard to view and explore pertinent aspects of the data.
- Develop a browser plugin that can highlight suspected user tweets and links to fraudulent websites.

3. Deliverables

3.1. AMI

AMI ID	ami-ba0449d0
AMI NAME	w205Project_V2.0
Publically Available	Yes
Region	N. Virginia
Operating System	CentOS 6
Loaded With	• JDK 1.7, Python 2.6.6 (standard CentOS distribution), Python 2.7.6 contained in its own virtualenv • Cloudera Distribution of Hadoop (2.6) • Spark 1.5.2 built over Hadoop 2.6 & Scala 2.11 • Flume ● Postgres ● Shiny-Server ● R

3.2. Scripts & Code

These are distributed throughout the code base. We will assume that the top level directory of the project cloned from github resides here: `/data/w205Project`

AMI Setup Scripts (Manual - once only per EBS volume)	provision/provision.sh provision/bootstrap.sh provision/twitter-keys.sh Provisioner also in S3: https://s3-us-west-2.amazonaws.com/w205twitterproject/provision.sh
Project Setup Scripts	setup-scripts/start_postgres.sh

(Manual - once only per EBS volume) (Setup by bootstrap.sh)	setup-scripts/stop_postgres.sh setup-scripts/start_metastore.sh setup-scripts/stop_metastore.sh setup-scripts/setup-hive-for-postgres.sh setup-scripts/stop_all.sh
Cron (Manual - at restart of instance)	cron/simple_sched.sh cron/simple_sched_cron.txt
Load - Flume (Triggered by Cron)	flume/start-flume.sh flume/conf/flume-env.sh flume/conf/flume.conf flume/twitter4j-stream-4.0.4.jar
Load-Extract-Hive (Triggered by Cron)	load/load-hive-table.sh load/load.sql load/hive-serdes-1.0-SNAPSHOT.jar
Transform - Hive (Triggered by Cron)	transform/transform.sh
Transform - Postgres (Setup by bootstrap.sh)	postgres/twitter.sql
Classification (Triggered by Cron)	spark/getLinks.py honeypot/*
Scraping & Upload to S3 (Triggered by Cron)	python/url_spider/*
Dashboard	shiny-server/dashboard/*
Browser Plugin	**chromeExtension.crx** chromeExtension/*
Spark Streaming (Work-in-progress)	spark/scala/*

3.4. Documentation

Apart from the github wiki, specific documentation exists within each module. However, this is to assist developers communicate and integrate their changes with other developers. Please use the wiki and this document for in-depth information.

4. Accomplished

4.1. Option A - The Full Scope Pipeline currently in "production"

- **Load**

We used Flume for connecting to the twitter stream. Since twitter is a very popular social media, custom sources for twitter already exist. Flume integrates very well with the Hadoop ecosystem and is used widely and we considered it a stable choice. Flume has three major components to its architecture: the source, the sink and the channel. The source was obviously the custom twitter source which could be configured using keywords. Flume requires that keywords be provided to ingest tweets. In order to get a reasonably unbiased sample of tweets, we use the top 100 words in the Twitter public stream as keywords. For the sink, we used HDFS. For the channel, we used a simple memory channel. We are simply using the tweets to study spammers and the URLs they use in their tweets. Losing some data is not a concern here. Because durability is not needed, we could configure an in memory channel.

- **Extract-Transform**
  Flume dumps the streaming data in HDFS directories based on the date:
  flume/tweets/YYYY/MM/DD
  We use an external table in HIVE and JSONSerDe to load the data into a structured format. Every day the HIVE partition is overwritten with the current day's data, and transformed into the necessary columns for classification using pyspark.

- **Classify**
  We apply a logistic regression model to an existing training dataset. We use this model to predict spam vs ham in the current HIVE partition.

- **Store**
  We append the classified data to a Postgres table for use by the Dashboard.
  We also output a json file of all urls contained in tweets classified as spam.

- **Scrape**
  We use Scrapy, a python framework for crawling and scraping the web, to enrich our links repository obtained from the previous steps in the pipeline (load, extract-transform, classify and store). The python code loads the local JSON file produced in the store phase as the starting urls for the crawler. The crawler then recursively goes through the urls as well as the urls present in the crawled pages until the stop condition is met (either 1000 responses crawled or 30 minutes have gone by). Afterwards, a separate python script loads the resulting scraped urls into S3 for the chrome plugin.

- **Scheduler**
  We use cron to trigger a shell script once a day, which in turn contains all the commands necessary to run through the whole application cycle.

- **Dashboard**
  To monitor the state of the classification model, and spammers' behavior over time, we have a dashboard which displays pertinent statistics about our classified data. This is built using HTML5 and R on the front-end, and Shiny-Server and Postgres on the backend. This is in a "proof-of-concept" stage, with a limited array of analytics.

- **Plugin**
  Chrome Extension for the end user. Clicking on the plugin highlights any links we have classified as spam on the user's webpage.

5. Release Notes

- Cron
    - Stopping flume is not optimal. Ideally, we should have a service to stop and start flume. Second best option would be to capture the pid of the flume agent. In the current implementation we terminate the last hdfs process.
- Flume
    - We are not listening for errors such as reaching the rate limit or twitter servers not responding. Flume requires keywords. To keep our sample unbiased we use the top 100 words collected from a public twitter stream.
- Python Scraper
    - The python scraper is called through the framework's “crawl” command by the scheduler rather than through a separate script
- R based dashboard
    - As R works with data in-memory, the dashboard uses a random sample of the collected and classified data, not all of it.
- Spark Streaming
    - AWS SDK support for Scala is poor. Building some Spark-Scala connectors to cloud based data store services would make future updates very efficient. For example, writing to Amazon's DynamoDB from Spark proved quite challenging.
    - Implementing a connection pool based design pattern as suggested by Spark documentation will also benefit us with respect to performance and effective resource usage.

6. Future Work

Improve the architecture by considering: Error recovery and Scalability

6.1. Error Recovery

In this pipeline, losing tweets is acceptable. We are working with a public stream and not on particular twitter users as part of a social media campaign. However, recovering from platform or program failures is important and would make our code more suited for high visibility production environments.

6.2. Scalability

Also, twitter already restricts the rate of public streaming. And we are using a sample of the sample and scaling is not a major concern in the loading phase. However, Spark clustering would help a lot. We are running our classification algorithm as well as the scraping over Spark and clustering (EMR) is definitely something we want to implement in the future.

Postgres may not be the best choice for appending new data. We would like to try columnar storage solutions, which would also be more suitable to analytics.

We would like to explore a more complete Lambda architecture, in particular, a speed layer for the analytics dashboard.

6.3. Enhancements

Some other aspects we would like to implement are unit tests, REST APIs & trying different applications such as semantic analysis.

We would also like to use more sophisticated methods such as reverse DNS look-ups to find fraudulent websites using the list from classification.

Our classification model is naive. We would like to write a more sophisticated model. At a minimum we would also like to verify it against the Twitter @spam API.

7. Complete Set of Instructions to Launch

7.1. AWS Instructions

1. Sign in to AWS EC2 console and create an EBS volume in region US East (recommended: 100 GB, SSD)
2. Go to AWS EC2 and search for AMI by name or ID (w205Project_V2.0 or ami-ba0449d0).
3. Launch an instance using this AMI; make sure it is at least an m3-large.
4. Security config must allow the following ports:
   ```
   4040    tcp 0.0.0.0/0
   50070   tcp 0.0.0.0/0
   8080    tcp 0.0.0.0/0
   22      tcp 0.0.0.0/0
   10000   tcp 0.0.0.0/0
   8020    tcp 0.0.0.0/0
   8088    tcp 0.0.0.0/0
   ```
5. Make sure that your EBS volume can see the running instance (the regions have to sync up).
6. Attach your EBS store to the running instance.
7. Open a terminal and connect to the running instance.
8. **You do not need to mount your volume. The provision script will take care of this for you.**

7.2. Initial Setup Instructions

9. First, mount and configure your volume by running the provisioner script. This script is hosted on S3.
   Find your device path:
   ```
   fdisk -l
   wget https://s3-us-west-2.amazonaws.com/w205twitterproject/provision.sh
   . provision.sh <DEVICE PATH>
   ```

10. Next setup your Git keys so that you can clone our repo.
    (If you know your keys, you can use the git-keys-template.sh to automate this - this is optional)

11. To clone our repo onto the EBS volume, do the following:
    ```
    cd /data
    git clone git@github.com:kyleiwaniec/w205Project.git
    cd /data/w205Project
    . provision/bootstrap.sh
    ```
    Bootstrap script will start Hadoop-HDFS, Postgres, Hive Metastore, and configure Twitter keys and AWS keys for posting to S3.

12. Run the scheduler:
    ```
    crontab /data/w205Project/cron/simple_shed_cron.txt
    ```

13. Watch the logs to monitor:
    ```
    tail -f /data/cronlog.log
    ```

8. Troubleshooting Tips

1. Check cron logs to see current state of application:
   ```
   tail -f /data/cronlog.log
   ```

2. Check HDFS to verify tweets were collected:
   ```
   sudo -u hdfs hdfs dfs -ls -R /user/flume/tweets/
   ```
   Or, go to the hdfs web ui:
   ```
   http://xx.x.xxx.xx:50070/explorer.html#/user/flume/tweets/
   ```

3. Sanity checks on Hive tables:
   ```
   today=$(date +"%Y/%m/%d")

   hive -e "ADD JAR /data/w205Project/load/hive-serdes-1.0-SNAPSHOT.jar;select *
   from tweets where today_date='$today' limit 1"

   hive -e "ADD JAR /data/w205Project/load/hive-serdes-1.0-SNAPSHOT.jar;select
   num_words from USERS_TWEETS_ATTRIBUTES limit 10"
   ```

4. URLs are stored here after classification (local FS):
   ```
   /data/w205Project/python/url_spider/url_spider/logs/temp_urls.log
   ```

5. Logs generated during crawling can be found here (local FS):
   `/data/w205Project/python/url_spider/url_spider/logs/spammy_urls.log.`

6. Python 2.7.6 Virtual Environment is activated. Python Web Crawler & Scraper is initiated.
   `source ~/ENV27/bin/activate`

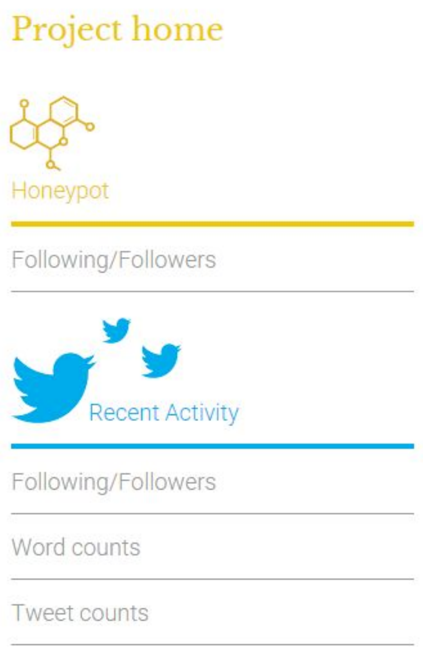
7. URLs that are likely fraudulent are stored in S3, Amazon's Object Storage Database.
   `https://w205twitterproject.s3-us-west-2.amazonaws.com/links2.json`

8. If for some reason you need to add Twitter keys or AWS keys again (maybe you pulled an update from git):
   ```
   provision/twitter-keys.sh
   provision/aws-keys.sh
   ```

9. Viewing the Dashboard

The dashboard is running on Shiny Server using R from the EBS Store attached to your instance. This was added to the AMI and started as part of bootstrap. You should not have to do anything specific to view the dashboard. In order to view the dashboard, please go to your instance URL on por 10000. **The tweet ingestion process has to complete at least once to view the dashboard**. Before then you may get an error that no data exists. Here is a template URL:
http://ec2-xx-x-xxx-xx.compute-1.amazonaws.com:10000/dashboard/

The home page:



From here, navigate to the various options listed in the side-nav to glean some insights from the tweets.

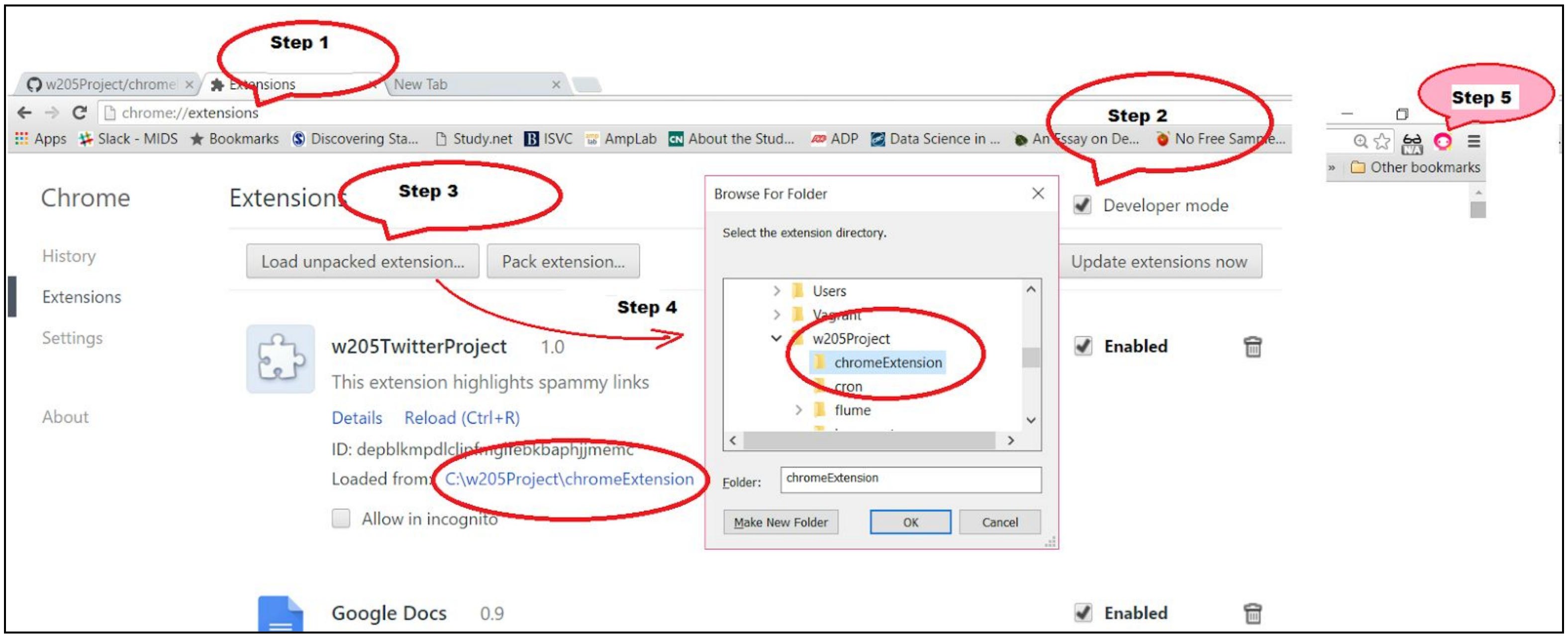
10. Installing Browser Plug-in

Drag the **chromeExtension.crx** file into Google Chrome browser. Follow the prompts. That’s it.

Alternatively, if you want to hack the javascript yourself, install in developer mode:

1. Open Google Chrome. Then in the address bar type: **chrome://extensions/**
2. Enable **'Developer mode'**
3. Click on the **'Load unpacked extensions...'** button.
4. Navigate to th**e chromeExtension f**older, and click select.
5. Click on the icon (the little pink guy). You should see spammy links highlighted in yellow

The Screenshots below provide guidance of the steps above:

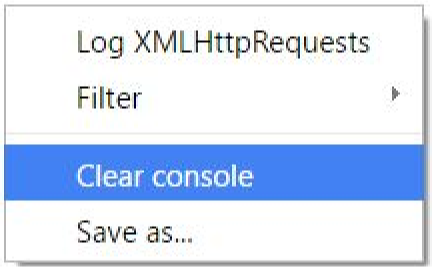


You can see what the browser plugin is actually doing:

1. Go to a site with at least some links. Here are our suggestions for a spammy site: `http://yourtrustedhacks.com/`

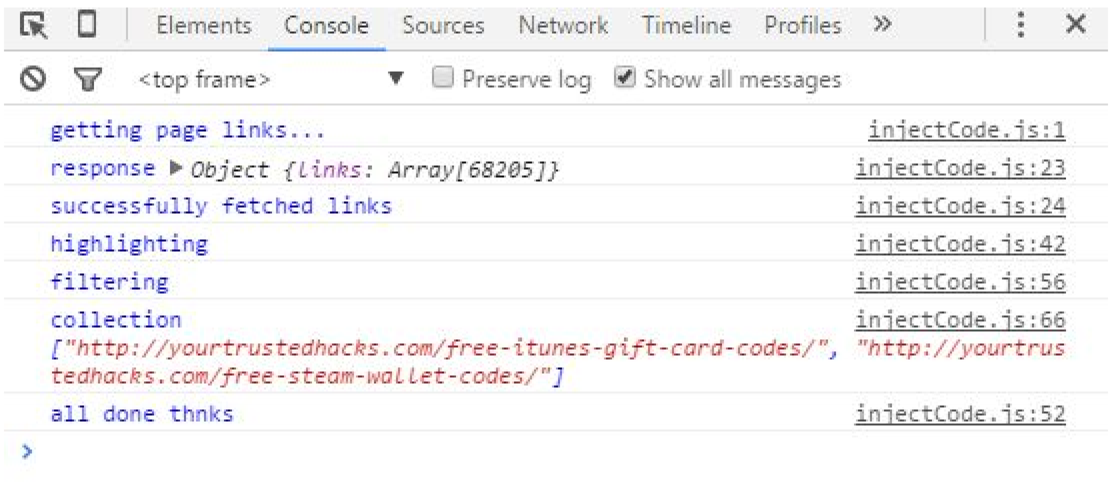
2. Press F-12 to open the Web Developer Console. There are many tabs in this docked window. Please make sure you are on the Console tab.

3. Right click and choose “Clear Console”. This will help you see the browser plugin action clearly.

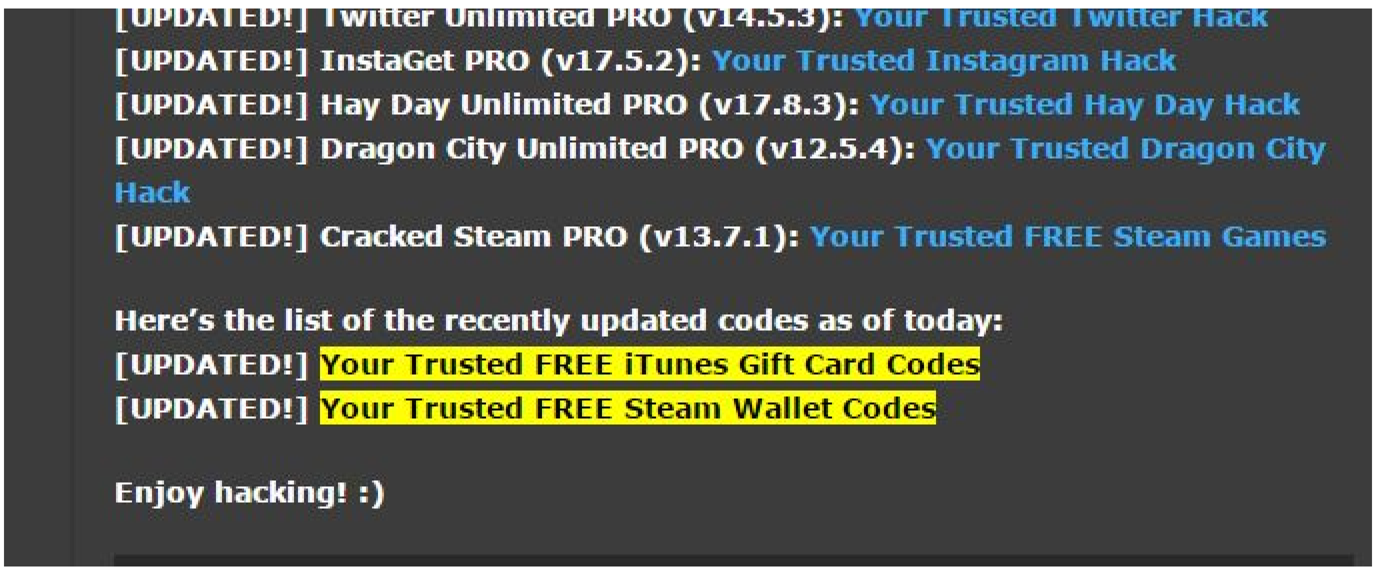


4. Now click on the in the right top corner of your browser.

5. Watch the console for all the steps taken by the plugin to highlight spammy links on the page you are currently on. **Also scan the page for the highlighted spammy links. And, please don’t click on those links!**



```
getting page links...                                          injectCode.js:1
response ▶ Object {links: Array[68205]}                        injectCode.js:23
successfully fetched links                                     injectCode.js:24
highlighting                                                   injectCode.js:42
filtering                                                      injectCode.js:56
collection                                                     injectCode.js:66
["http://yourtrustedhacks.com/free-itunes-gift-card-codes/", "http://yourtrus
tedhacks.com/free-steam-wallet-codes/"]
all done thnks                                                 injectCode.js:52
>
```



[UPDATED!] Twitter Unlimited PRO (v14.5.3): Your Trusted Twitter Hack
[UPDATED!] InstaGet PRO (v17.5.2): Your Trusted Instagram Hack
[UPDATED!] Hay Day Unlimited PRO (v17.8.3): Your Trusted Hay Day Hack
[UPDATED!] Dragon City Unlimited PRO (v12.5.4): Your Trusted Dragon City Hack
[UPDATED!] Cracked Steam PRO (v13.7.1): Your Trusted FREE Steam Games

Here's the list of the recently updated codes as of today:
[UPDATED!] Your Trusted FREE iTunes Gift Card Codes
[UPDATED!] Your Trusted FREE Steam Wallet Codes

Enjoy hacking! :)

Appendix A - The Spark-Streaming App (Option B)

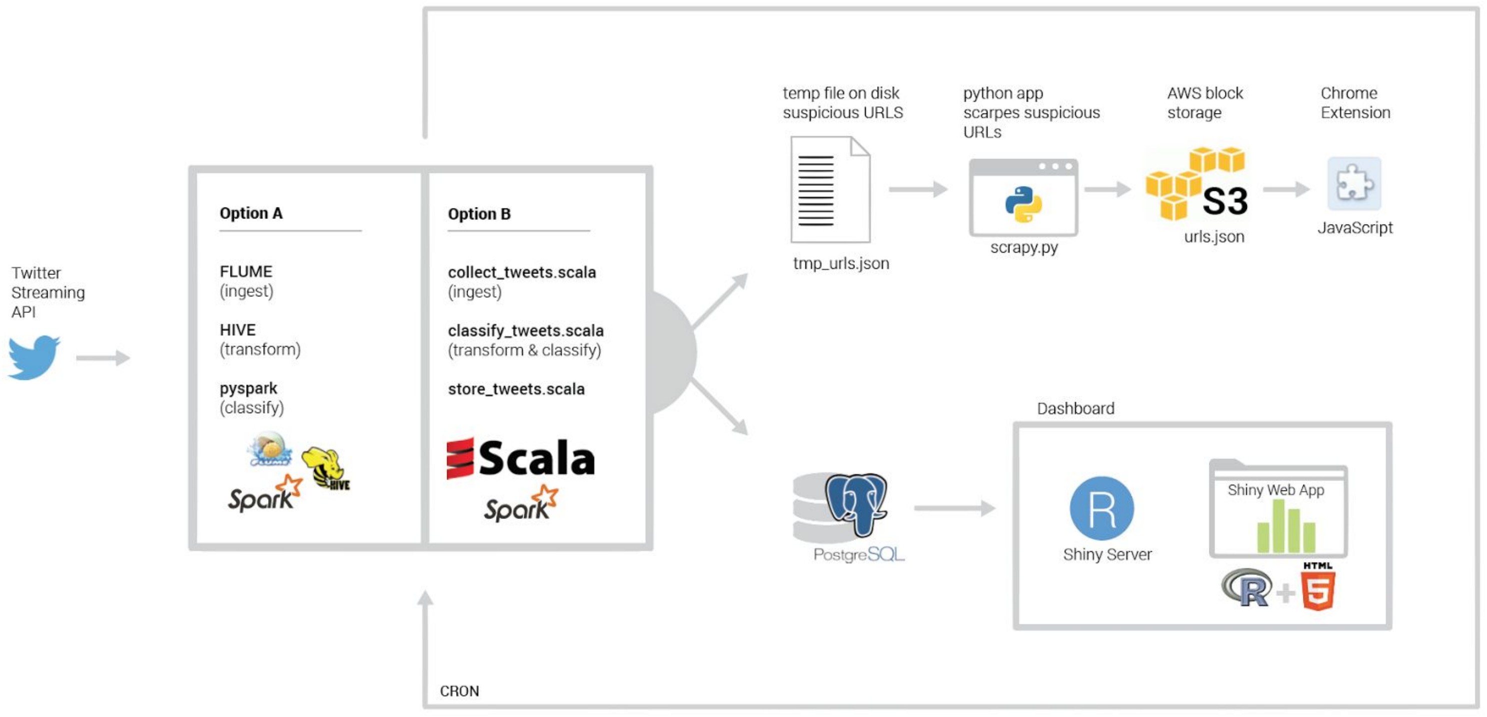
This is still a work in progress. There are some known issues we will continue to work on. We are unable to guarantee that all modules will run successfully for this option. The tested modules are highlighted in green below. The others have not been rigorously tested yet.

(Since this is a streaming solution, there is no cron scheduler for this part.)

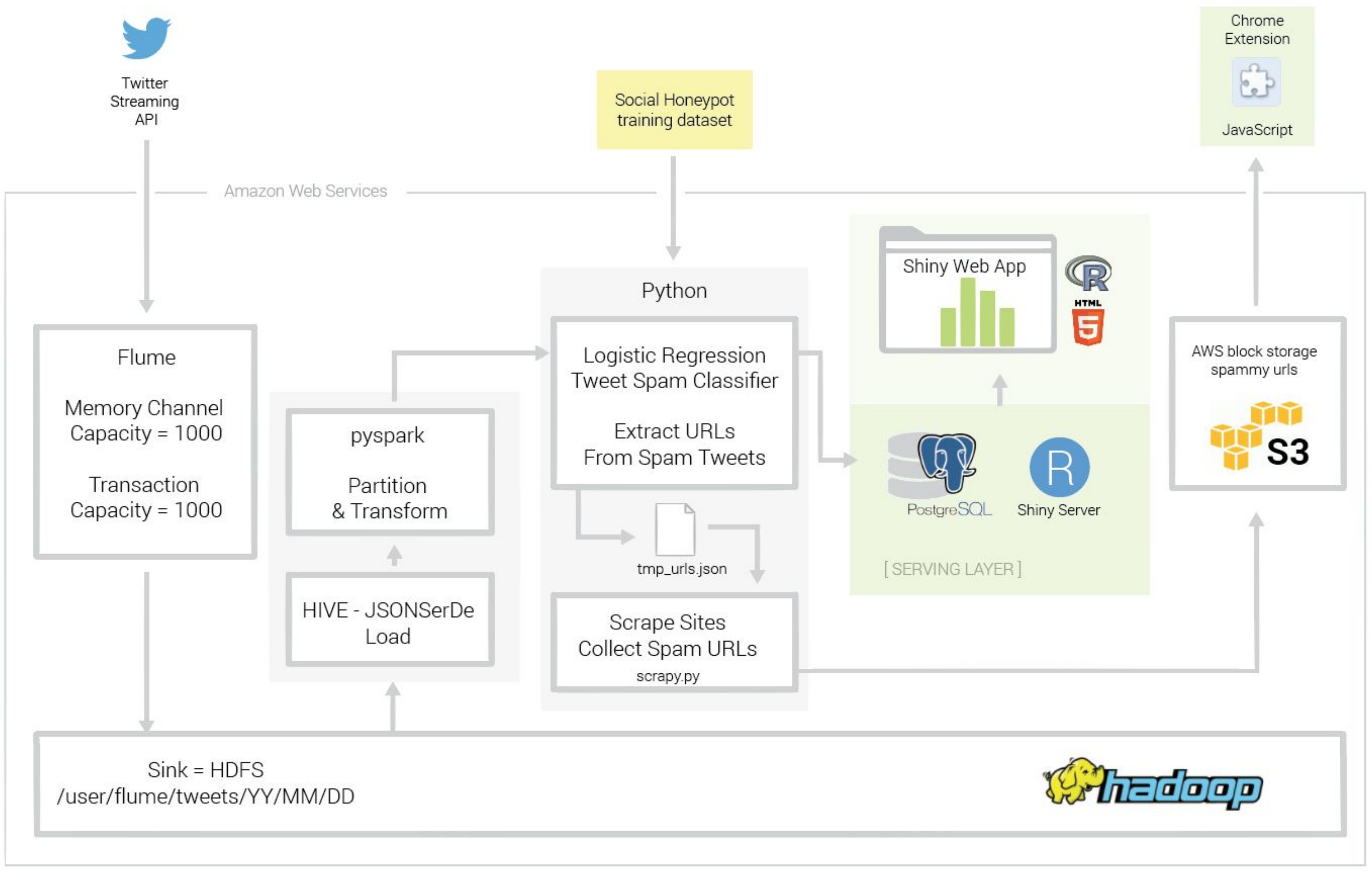
1. Make sure hadoop is running and then run the following script:
   /data/w205Project/spark/scala/setup_scala_app.sh

2. First kick off tweet collection. This app will collect 100K tweets and exit:
   /data/w205Project/spark/scala/classify_store_tweets/start-tweet-collection.sh
3. Next, build the classification model and store it:
   /data/w205Project/spark/scala/classify_store_tweets/start-learning-spam.sh
4. Start the tweet stream classification and storing:
   /data/w205Project/spark/scala/classify_store_tweets/start-streaming-classification.sh
5. Run this script to install and start Zeppelin:
   /data/w205Project/spark/scala/zeppelin/setup_zeppelin.sh
6. Load the notebook to visualize some data points:
   /data/w205Project/spark/scala/zeppelin/simple_dashboard.ipynb
7. HDFS Paths:
   Tweets: /user/spark/tweets
   Training data: /user/spark/honeypot/*.csv
   Classification Model: /user/spark/honeypot/lr_sgd

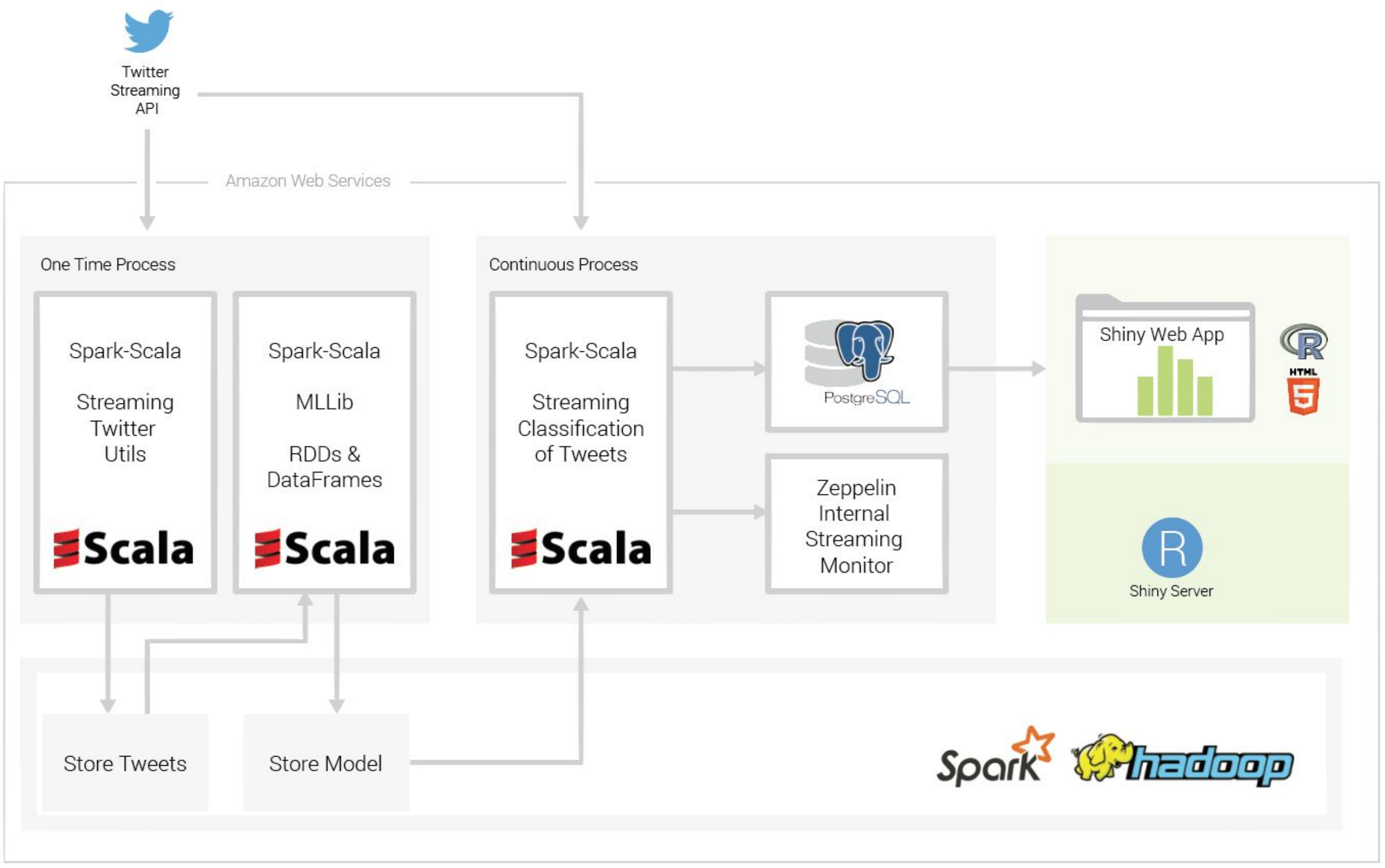
Appendix B - Data Flow At A Glance


Twitter Streaming API
Option A
FLUME (ingest)
HIVE (transform)
pyspark (classify)
Spark
HIVE
Option B
collect_tweets.scala (ingest)
classify_tweets.scala (transform & classify)
store_tweets.scala
Scala
Spark
temp file on disk suspicious URLS
tmp_urls.json
python app scarpes suspicious URLs
scrapy.py
AWS block storage
S3
urls.json
Chrome Extension
JavaScript
PostgreSQL
Dashboard
Shiny Server
Shiny Web App
HTML
CRON

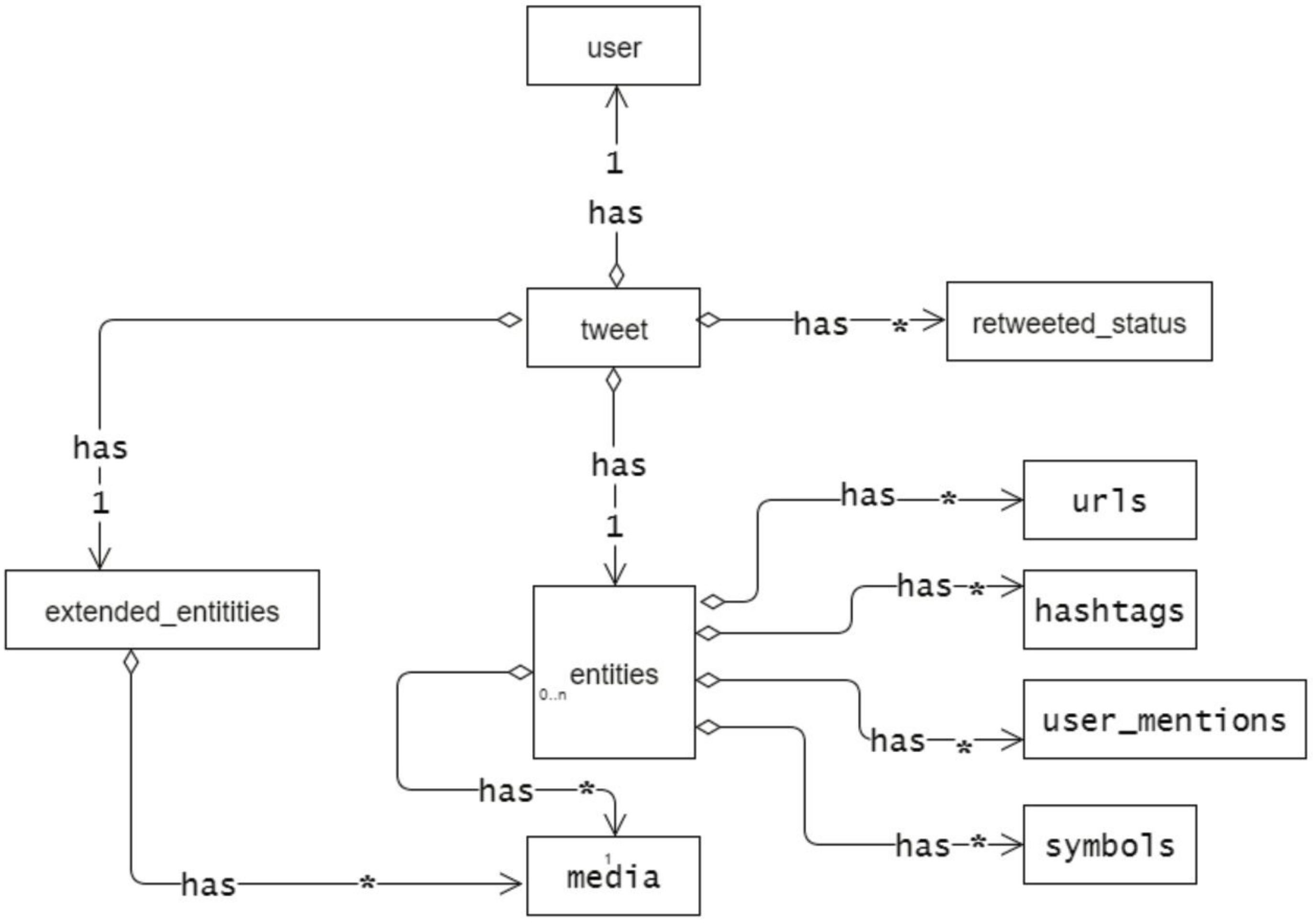
Appendix C - The Final Architecture


Twitter Streaming API
Social Honeypot training dataset
Chrome Extension
JavaScript
Amazon Web Services
Python
Shiny Web App
HTML
Flume
Memory Channel Capacity = 1000
Transaction Capacity = 1000
pyspark
Partition & Transform
Logistic Regression Tweet Spam Classifier
Extract URLs From Spam Tweets
PostgreSQL
Shiny Server
AWS block storage spammy urls
S3
tmp_urls.json
[SERVING LAYER]
HIVE - JSONSerDe Load
Scrape Sites Collect Spam URLs
scrapy.py
Sink = HDFS /user/flume/tweets/YY/MM/DD
hadoop

Appendix D - The Streaming Solution Architecture



Appendix E - The Tweet Object Relationship Analysis



References

Twitter Spam

[1] [Twitter Spam Conditions] (https://support.twitter.com/articles/18311)
[2] [Finding Fraudulent Websites Using Twitter Streams] (http://techscience.org/a/2015092905/)
[3] [Machine Learning Techniques applied to Twitter Spammers Detection]
(http://www.wseas.us/e-library/conferences/2014/Florence/CSCCA/CSCCA-23.pdf)
[4] [Suspended Accounts in Retrospect: An Analysis of Twitter Spam]
(http://www.icir.org/vern/papers/twitter-susp-accounts.imc2011.pdf)
[5] [Kyumin Lee, honeypot](http://digital.cs.usu.edu/~kyumin/pubs/lee10sigir.pdf)
[6] [Kyumin Lee, long term study]
(http://www.aaai.org/ocs/index.php/ICWSM/ICWSM11/paper/view/2780/3296)
[7] [Detection of Internet Scam Using Logistic Regression]
(http://www.cs.cmu.edu/~eugene/research/full/detect-scam.pdf)

Development Related

[1] [Twitter Dev Site] (https://dev.twitter.com/)
[2] [Hive related shell script] (https://www.mapr.com/blog/quick-tips-using-hive-shell-inside-scripts)
[3] [Spark Documentation] (https://spark.apache.org/docs/latest/index.html)
[4] [Databricks Reference Implementation] (https://www.gitbook.com/@databricks)
[5] [Shiny Server Admin Guide] (http://rstudio.github.io/shiny-server/latest/)
[6] [Cloudera Flume Integration]
(http://blog.cloudera.com/blog/2012/10/analyzing-twitter-data-with-hadoop-part-2-gathering-data-with-flume/)