Application Idea

The application is a way to analyze live streaming tweets to determine current trends of tweeters. The application takes in whole tweet texts, parses the tweet into individual words (and cleans out certain formatting), then counts the words. A database keeps track of how often individual words have been seen allowing analysis of word importance. Higher word counts indicate increased discussion about a topic. Changes in word counts relative to itself or other words indicate increased or decreased importance of the topic.

Description of architecture

Streamparse works by running Python code against real-time streams of data which we collect using tweepy (more on this later). By integrating with Apache Storm, there is an easy-to-use way of running local and remote computation clusters. This will allow me to digest, parse, and analyze a live stream of tweets.

Streamparse relies on a topology (written in Clojure) which defines how Spouts and Bolts are connect. My tweetwordcount topology indicates I have 1 spout (tweet-spout) and 2 bolts (parse-tweet-bolt and count-bolt). The general flow is tweets arrive in tweet-spout via tweepy, are passed to parse-tweet-bolt where they are cleaned and parsed into individual words, and the word collections are sent to count-bolt to be counted in a postgres database tweetwordcount. The cleaning step I've left as preset by the assignment with getting rid of certain non-word symbols, retweets, and website links as these all seem valid nonsensical data for analyzing tweet trends. I've also left the requirement that the tweets have certain common words (such as a, the, is) so that there's a modicum of validation on the tweet being a real english thought. I've chosen to set parallelism of tweet-spout and parse-tweet-bolt to 1 to reduce the chance of common words between tweets or within a tweet being miscounted. For count-bolt, I have it use fields groupings on 'words' from parse-tweet-bolt so miscounting will not be an issue. Increasing the parallelism will reduce the bottleneck potential of the heavier processing involvement of this final bolt step.

Tweet-spout uses tweepy as a way of interacting with Twitter. I will be using tweepy.StreamListener which creates a streaming session to download twitter messages in real time. To satisfy Twitter's requirement that all requests to use OAuth for authentication, Tweepy makes OAuth authentication painless by creating an OAuthHandler instance. The OAuthHandler takes in customer token and secret from a Twitter Application to verify an authenticated user and an access token and secret to gain secure access to tweets. We then use the Twitter streaming API wrapper to authenticate our OAuthHandler, create a class from tweepy.StreamListener (to receive the live stream of tweets), use the class to create a Stream object (which pushes the tweets to the StreamListener), and connect to the streaming API using Stream.

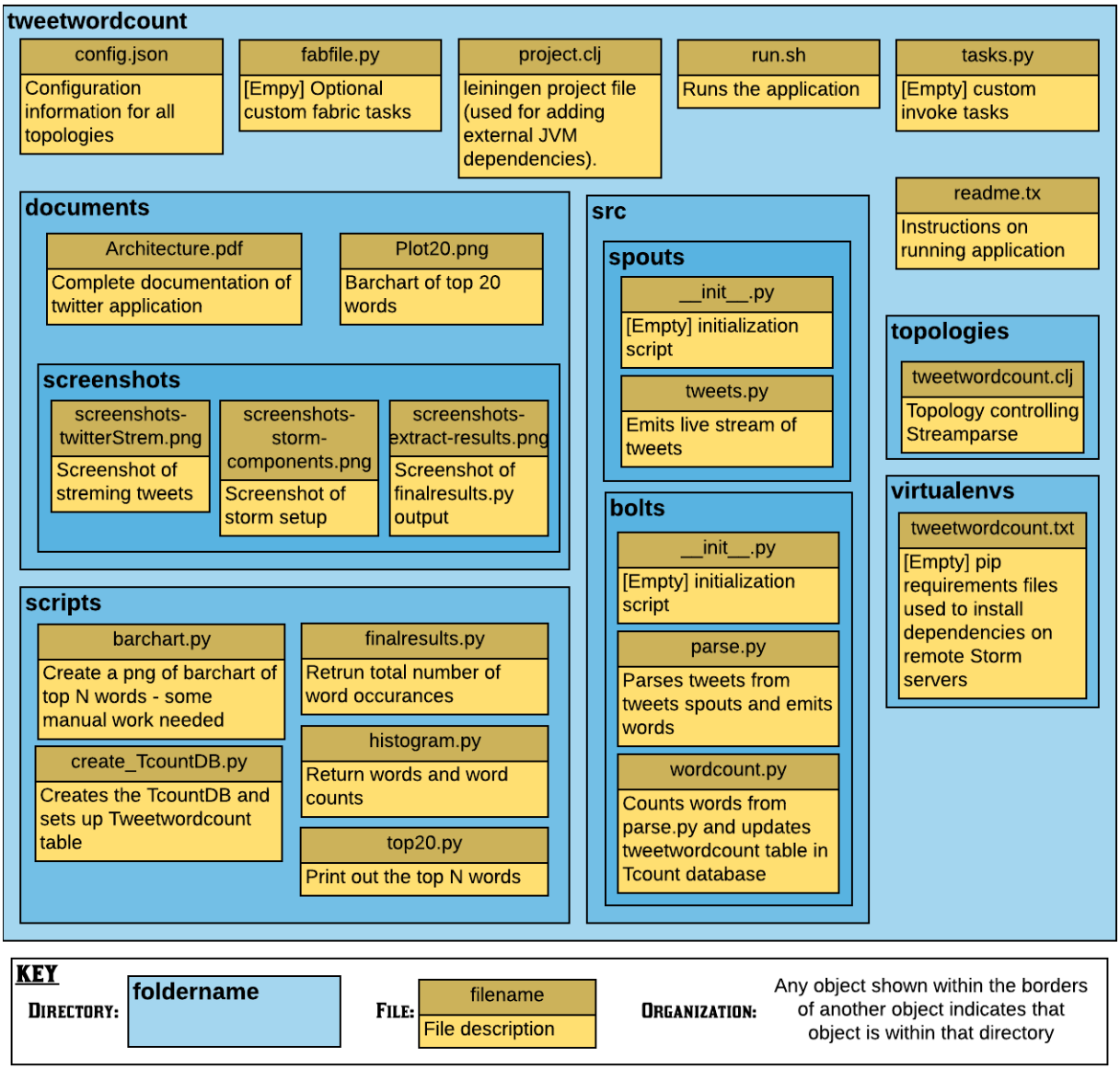
As tweet-spout emits tuples of tweets, parse-tweet-bolt will accept the tweet tuple, split the tweet into words, filter out certain known nonwords or retweets, and emit the (many) non-empty tweet words. If the filtering process got rid of the tweet's contents, nothing is emitted.

Finally, count-bolt takes the tweet words and increments the count of that word in a PostgresQL database. PostgresSQL is an object-relational database system which will persist after the streamparse process ends. By storing words and counts in PostgresSQL, I can analyze the results of spreamparse at later dates and in more depth. In PostgresSQL, I created a database Tcount and a table tweetwordcount

within Tcount which houses all the tweet words encountered and how often they've been encountered (their count). Count-bolt connects to Tcount, checks if the current word is part of tweetwordcount, adds the word if it is not, then increments the word's count by 1.

Please see File Dependencies for a map of which files relate to which to set up this process flow.

Directory and file structure:



File Dependencies