# Build a sentiment analysis application with Node.js, Express, sentiment, and ntwitter

**27 June 2014** **Share:**

Watch on YouTube

**Scott Rich**
*VP of Engineering at Two Sigma*
*LinkedIn*
*@ScottRich*

As a software app developer, I've been trying to solve the following problem: how to get immediate feedback from users, specifically online users. Getting an instant read on the Twitterverse's reactions to an app, product launch, campaign, or current event, for example, would be enormously helpful. Are the reactions positive, negative, neutral?

As I thought through the requirements for an application to quickly gauge public sentiment on Twitter, I wanted it to be quick to develop, interface with a Twitter web service, have a simple mobile interface, and quickly analyze large volumes of data over time.

*With existing building blocks, you can put together, very quickly, a fun little sentiment analysis application.*

I chose to build it as a PaaS app, using JavaScript and the popular Node.js runtime for PaaS apps. The app uses the service-composition programming model supported by PaaS environments like Cloud Foundry and Heroku. I settled on developing this app at DevOps Services (formerly JazzHub). This provides me with a place to publish the application, as well as a nice browser-based IDE to develop the code.

Run the app Get the code

**Got a Bluemix question?**

Ask on Stack Overflow

Ask on dW Answers

What's the difference between asking on Stack Overflow and asking on dW?

## What you'll need to build a similar app

1. A basic familiarity with Node.js and a Node.js development environment.
2. These Node.js modules:

   Express framework: Makes it easy to build Node.js web apps.

   sentiment module: Does simple sentiment analysis on a string.

   ntwitter module: Provides a simple interface to Twitter, including support for Twitter's Stream API.

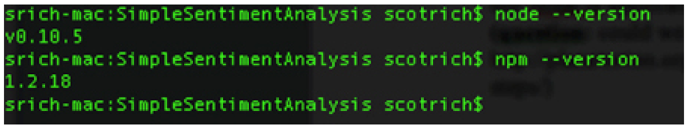
**WATCH:** Getting started with Node.js (demo)

The steps below describe the most basic approach. For more advanced tooling to speed your development, see Additional tooling considerations.

## Step 1. Build an Express application

1. Run the `node` and `npm` commands:

   **Figure 1. Confirming that Node.js and NPM are set up on your system**

```
srich-mac:SimpleSentimentAnalysis scotrich$ node --version
v0.10.5
srich-mac:SimpleSentimentAnalysis scotrich$ npm --version
1.2.18
srich-mac:SimpleSentimentAnalysis scotrich$
```

2. Create the simplest Node.js Express app possible: Hello Node.js (see Listing 1).

   In a new directory, create the app.js file, the main file of the Node.js app. Tell Node.js that the Express module is required, and create the Express app. Register a route so that requests for `/hello` will be handled by the anonymous function shown in Listing 1. The handler for `/hello` uses the Express helpers to send a simple text response. The send method handles details such as defaulting the content type and length. When using Express, we can ignore some of the details of the underlying HTTP conversation.

   **Listing 1. Hello Node.js**

```
var port = (process.env.VCAP_APP_PORT || 3000);
var express = require("express");

var app = express();

app.get('/hello', function(req, res) {
    res.send("Hello world.");
});

app.listen(port);
console.log("Server listening on port " + port);
```

**Note:** The program could be even simpler, but a few additional lines of code enable it to run in the CloudFoundry runtime. In CloudFoundry, the app will be relocated to a different port (the port can be discovered from the environment).

3. Give Node.js some additional information about the app, as shown in Listing 2.

   Create a `package.json` file to tell the app that the Express module will be used, and name the app "Simple Sentiment Analysis App." Since this app is not intended to be published to the NPM registry, you can mark it as private. Finally, declare that the app depends on a 3.x version of the Express module.

   **Listing 2. package.json**

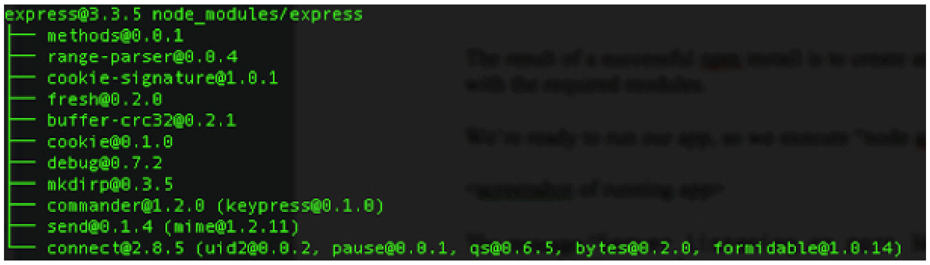
```
package.json for Hello Node
{
  "name": "SimpleSentimentAnalysisApp",
  "description": "Simple Sentiment Analysis App",
  "version": "0.9.0",
  "private": true,
  "dependencies": {
    "express": "3.x"
  }
}
```

## Run and test your Hello Node.js app

Next, run and test the Hello Node.js app using these steps:

1. Before running your app, prepare the node environment using NPM to fetch the dependencies declared for the app (and their dependencies and so on).

   **Figure 2. Prepare the node environment**

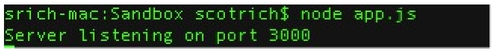


```
express@3.3.5 node_modules/express
├── methods@0.0.1
├── range-parser@0.0.4
├── cookie-signature@1.0.1
├── fresh@0.2.0
├── buffer-crc32@0.2.1
├── cookie@0.1.0
├── debug@0.7.2
├── mkdirp@0.3.5
├── commander@1.2.0 (keypress@0.1.0)
├── send@0.1.4 (mime@1.2.11)
└── connect@2.8.5 (uid2@0.0.2, pause@0.0.1, qs@0.6.5, bytes@0.2.0, formidable@1.0.14)
```

   The result of a successful NPM install is the creation of the `node_modules` directory, populated with the required modules.

2. Type `node app.js` at the command line.

   **Figure 3. Run the node app.js command**

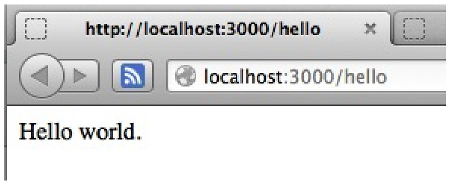


```
srich-mac:Sandbox scotrich$ node app.js
Server listening on port 3000
```

The message "Server listening on port 3000" confirms that the app has started and is listening for requests.

3. Test the behavior of the app by pointing a browser to http://localhost:3000/hello.

**Figure 4. Test the behavior of the app**



Hello world.

## Step 2. Add sentiment analysis to your application

Now that your basic app is running and the environment is set up, it's time to add some real function using the very cool **sentiment** module by Andrew Sliwinsky. He describes it as "a Node.js module that uses the AFINN-111 wordlist to perform sentiment analysis on arbitrary blocks of input text." This is a relatively simple implementation of sentiment analysis, as you'll see. It simply scores the words in your text using a dictionary of English words that have been rated for positive or negative sentiment.

1. To begin using the sentiment module, update the `package.json` to include it as a dependency:

   **Listing 3. Update package.json**

```
{
  "name": "SimpleSentimentAnalysisApp",
  "description": "Simple Sentiment Analysis App",
  "version": "0.9.0",
  "private": true,
  "dependencies": {
    "express": "3.x",
    "sentiment": "0.2.1"
  }
}
```

2. Tell Node.js to get this new module with the command `npm update`. (Note: The docs suggest that `npm install` should also have picked up the new dependency, but that was not the behavior I saw.)

3. Require the new sentiment module:

```
var express = require("express");
```

4. Create another route to experiment with the sentiment module:

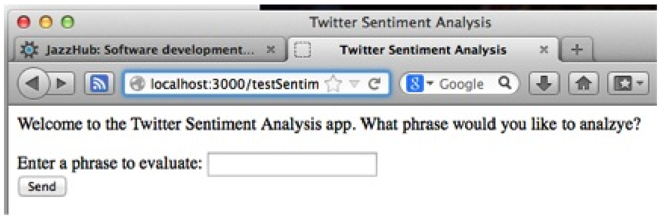
   **Listing 4. The sentiment analysis app interface code**

```
app.get('/testSentiment',
    function (req, res) {
        var response = "<HEAD>" +
          "<title>Twitter Sentiment Analysis</title>\n" +
          "</HEAD>\n" +
          "<BODY>\n" +
          "<P>\n" +
          "Welcome to the Twitter Sentiment Analysis app.  " +
          "What phrase would you like to analzye?\n" +
          "</P>\n" +
          "<FORM action=\"/testSentiment\" method=\"get\">\n" +
          "<P>\n" +
          "Enter a phrase to evaluate: <INPUT type=\"text\" name=\"phrase\"><BR>\n" +
          "<INPUT type=\"submit\" value=\"Send\">\n" +
          "</P>\n" +
          "</FORM>\n" +
          "</BODY>";
        var phrase = req.query.phrase;
        if (!phrase) {
            res.send(response);
        } else {
            sentiment(phrase, function (err, result) {
                response = 'sentiment(' + phrase + ') === ' + result.score;
                res.send(response);
            });
        }
    });
```

Apologies to web architects and designers everywhere for the brute force simplicity of this new function. I promise we'll do something more intelligent later. For now, we're going to put up a form that lets us enter a phrase and then respond with the rating provided by sentiment. The key function call is `sentiment(phrase, function)`. In the spirit of Node.js, the sentiment library is asynchronous, so the result of the analysis is processed by a callback function.

5. Restart the app and point a browser to http://localhost:3000/testSentiment.

**Figure 5. The sentiment analysis app user interface**


Twitter Sentiment Analysis
JazzHub: Software development...
Twitter Sentiment Analysis
localhost:3000/testSentim
Google

Welcome to the Twitter Sentiment Analysis app. What phrase would you like to analzye?

Enter a phrase to evaluate:

Send

*Et voilà, a fabulous user interface for sentiment analysis!*

## Scoring sample sentiments

Here are some hypothetical results from our sentiment analysis app:

**Table 1. Hypothetical results of analyzing "Node.js"**

| Sentiment | Score | Interpretation |
|---|---|---|
| Node.js is cool, I love it | 4 | Very positive sentiment |
| Node.js is uncool, I hate it | -3 | Very negative sentiment |
| Mi piace Node.js. Node.js é bellissima | 0 | No sentiment detected. Sentiment has only an English dictionary. |
| Node.js is not cool, I do not love it | 4 | Oops! Very positive, same score as the first example. Sentiment is not aware of grammar or negation. |

You can see the limitations of this approach, namely that the sentiment module only has an English dictionary and is not aware of grammar or negation. Still, it's probably good enough to get the gist of most tweets.

## Step 3. Connect the application to Twitter

The other essential component of the app is the connection to Twitter, so that it can obtain a stream of tweets to analyze.

1. Use the ntwitter module, which provides a Node.js interface to the Twitter API, to request a Stream of public tweets mentioning one or more keywords.

2. Obtain a set of keys by registering it under your Twitter developer account. Go to the Twitter "My Applications" page to create and manage the app.

3. Add a couple of functions to your app to test the connection to Twitter. First construct a twitter object using ntwitter and OAuth keys, then make a simple call to verify those credentials.

**Twitter's Stream interface** is perfect for observing trends in sentiment, since it allows access to all of the Twitter content on a topic over time.

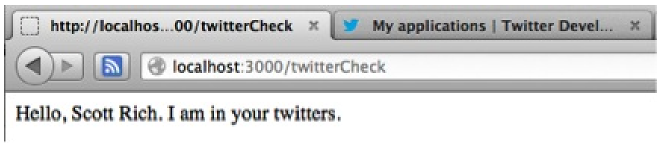
Our app works only with the public Twitter stream, so Twitter's Application-only Authentication scheme is sufficient. This simplified OAuth exchange doesn't require a user login or consent prompt. Using this method does require an app-specific set of OAuth keys, which are encoded in the app. For more, see Twitter streaming APIs.

**Listing 5. Testing your Twitter connection**

```
// Sample keys for demo and article - you must get your own keys
//   if you clone this application!
// Create your own app at: https://dev.twitter.com/apps
var tweeter = new twitter({
    consumer_key: 'your',
    consumer_secret: 'keys',
    access_token_key: 'go',
    access_token_secret: 'here'
});

app.get('/twitterCheck', function (req, res) {
    tweeter.verifyCredentials(function (error, data) {
        res.send("Hello, " + data.name + ".  I am in your twitters.");
    });
});
```

4. Restart the app and visit `/twitterCheck` to see that the connection and login to Twitter are working:

**Figure 6. Test the connection to Twitter**


http://localhos...00/twitterCheck
My applications | Twitter Devel...
localhost:3000/twitterCheck

Hello, Scott Rich. I am in your twitters.

Note that it does identify me as the owner of the app that is connecting.

5. Now that the connection to Twitter is established, add a function to monitor Twitter for one or more phrases. ntwitter provides a very simple method for

doing this:

a. Call `stream()` and pass the phrase to be monitored.

b. Use a callback function to process tweets as they arrive.

   The Twitter Stream API allows the app to open a Stream and keep pulling data as long as it is available. This function works perfectly for the Node.js app, because it can keep a stream open on the server and update counters and average sentiment as it processes the stream.

c. Create a simple interface to set the phrase to be monitored and to display the results as they come in. A more sophisticated app could store the Tweets to a database for more complex processing.

Listing 6 shows the essence of the code to open a stream and log a sample of the tweets:

**Listing 6. Opening a stream and logging tweets**

```
app.get('/watchTwitter', function (req, res) {
    var stream;
    var testTweetCount = 0;
    var phrase = 'bieber';
    tweeter.verifyCredentials(function (error, data) {
        if (error) {
            res.send("Error connecting to Twitter: " + error);
        }
        stream = tweeter.stream('statuses/filter', {
            'track': phrase
        }, function (stream) {
            res.send("Monitoring Twitter for \'" + phrase
                + "\'...  Logging Twitter traffic.");
            stream.on('data', function (data) {
                testTweetCount++;
                // Update the console every 50 analyzed tweets
                if (testTweetCount % 50 === 0) {
                    console.log("Tweet #" + testTweetCount + ":  " + data.text);
                }
            });
        });
    });
});
```

## Step 4. Put it all together

Okay, now we have all the parts in place to complete your app:

A basic Express app to serve up various pages

A sentiment analysis function to assess the sentiment of text

A connection to Twitter to provide a source of tweets to analyze

Now we just need to combine them to do something interesting. The finished app will prompt the user for a phrase, call Twitter to open a stream filtered on the phrase, analyze the sentiment of each matching tweet, and create a page to monitor the sentiment over time.

Keeping the user interface very simple, Listing 7 shows the full application, in all its Old School Struts-style glory:

**Listing 7. The full application**

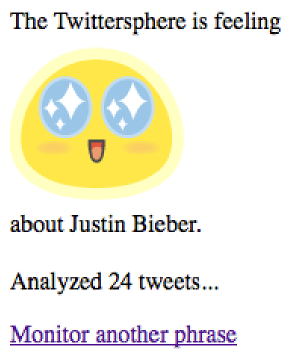
```
var tweetCount = 0;
var tweetTotalSentiment = 0;
var monitoringPhrase;

function resetMonitoring() {
    monitoringPhrase = "";
}

function beginMonitoring(phrase) {
    var stream;
    // cleanup if we're re-setting the monitoring
    if (monitoringPhrase) {
        resetMonitoring();
    }
    monitoringPhrase = phrase;
    tweetCount = 0;
    tweetTotalSentiment = 0;
    tweeter.verifyCredentials(function (error, data) {
        if (error) {
            return "Error connecting to Twitter: " + error;
        } else {
            stream = tweeter.stream('statuses/filter', {
                'track': monitoringPhrase
            }, function (stream) {
                console.log("Monitoring Twitter for " + monitoringPhrase);
                stream.on('data', function (data) {
                    // only evaluate the sentiment of English-language tweets
                    if (data.lang === 'en') {
                        sentiment(data.text, function (err, result) {
                            tweetCount++;
                            tweetTotalSentiment += result.score;
                        });
                    }
                });
            });
            return stream;
        }
    });
}

function sentimentImage() {
    var avg = tweetTotalSentiment / tweetCount;
    if (avg > 0.5) { // happy
        return "/images/excited.png";
    }
    if (avg < -0.5) { // angry
        return "/images/angry.png";
    }
    // neutral
    return "/images/content.png";
}

app.get('/',
    function (req, res) {
        var welcomeResponse = "<HEAD>" +
            "<title>Twitter Sentiment Analysis</title>\n" +
            "</HEAD>\n" +
            "<BODY>\n" +
            "<P>\n" +
            "Welcome to the Twitter Sentiment Analysis app.<br>\n" +
            "What would you like to monitor?\n" +
            "</P>\n" +
            "<FORM action=\"/monitor\" method=\"get\">\n" +
            "<P>\n" +
            "<INPUT type=\"text\" name=\"phrase\"><br><br>\n" +
            "<INPUT type=\"submit\" value=\"Go\">\n" +
            "</P>\n" + "</FORM>\n" + "</BODY>";
        if (!monitoringPhrase) {
            res.send(welcomeResponse);
        } else {
            var monitoringResponse = "<HEAD>" +
                "<META http-equiv=\"refresh\" content=\"5; URL=http://" +
                req.headers.host +
                "/\">\n" +
                "<title>Twitter Sentiment Analysis</title>\n" +
                "</HEAD>\n" +
                "<BODY>\n" +
                "<P>\n" +
                "The Twittersphere is feeling<br>\n" +
                "<IMG align=\"middle\" src=\"" + sentimentImage() + "\"/><br>\n" +
                "about " + monitoringPhrase + ".<br><br>" +
                "Analyzed " + tweetCount + " tweets...<br>" +
                "</P>\n" +
                "<A href=\"/reset\">Monitor another phrase</A>\n" +
                "</BODY>";
            res.send(monitoringResponse);
    }
});
```

We’ve finally put something at the root path of the app as Listing 7 shows. The first time through, the app presents a welcome prompt and gathers a phrase to monitor. It then sets up the monitoring stream and presents a results page. The tweeter’s stream callback is updated to pass the tweet contents through the sentiment function, increment the tweet count, and record the sentiment. Non-English tweets are filtered out.

For presentation purposes, the app uses the `sentimentImage()` function, which returns an image URL for a given sentiment value. The ranges for happy and grumpy are arbitrary. I found the range of sentiment to be surprisingly narrow for most topics, maybe due to the relative shortness of tweets. Feel free to play with these ranges.

So let’s give it a try. Here are the results of a few test runs:

**Figure 7. Twittersphere is positive about Justin Bieber**



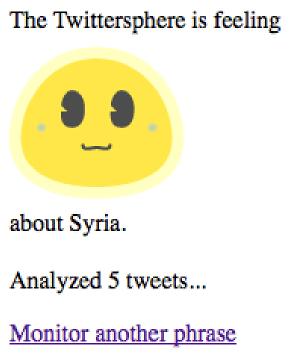
The Twittersphere is feeling

about Justin Bieber.

Analyzed 24 tweets...

Monitor another phrase

**Figure 8. Twittersphere is ambivalent about Syria**



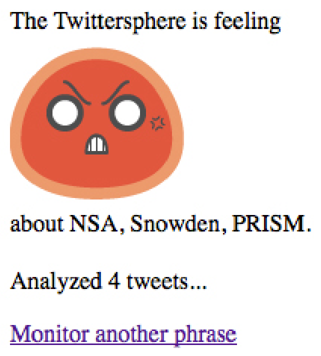
The Twittersphere is feeling

about Syria.

Analyzed 5 tweets...

Monitor another phrase

**Figure 9. Twittersphere is angry about NSA, Snowden, Manning, PRISM**



The Twittersphere is feeling

about NSA, Snowden, PRISM.

Analyzed 4 tweets...

Monitor another phrase

## Additional tooling considerations

In Step 1, I showed how you can use the `node` and `npm` commands to manage and run the application. You can easily do all of the coding with a text editor. However, in developing the app, I actually used a couple of other tools to ease my development.

Also, I ultimately wanted to push the app to a Cloud Foundry-compatible Node.js runtime environment, such as CloudFoundry.com or the IBM® Bluemix™ platform. So the final app includes a few extra files required for that environment: the manifest.yml file describes the app and its resource needs to the Cloud Foundry runtime, and the npm-shrinkwrap.json files tell the Cloud Foundry Node.js runtime exactly which modules the app should be deployed with.

## Conclusion

In developing this app using Node.js, Express, ntwitter, and sentiment, I got a real appreciation for how easy it is to consume capabilities like Twitter access and sentiment analysis when they are packaged as Node.js modules. It's easy to see why Node.js is so popular for developing web and mobile apps.

I'm now curious to try using Express to add a more professional user interface to the app. I think there's plenty of room to improve it. ; )

Sign up for a free Bluemix trial

Connect with Bluemix developers

**RELATED TOPICS:** Node.js JavaScript Social analytics