Case Study: Putting The Watson Developer Cloud to Work

See the source code and thinking behind NewsMedia Inc.'s (a fictitious company) implementation of the deep learning and natural language processing APIs available on Watson Developer Cloud.

By Doron Katz & Lucian Boboc

Editors:

David Berlind
Deb Donston-Miller

Making the Case For Natural Language Processing APIs

As the API economy exits nascency and enters its second decade, many organizations—from startups to the Fortune 500—are settling on best practices when it comes to both providing and consuming APIs.

On the provisioning side, many organizations are recognizing the transformational power of APIs. Although lighter in weight and easier to work with than any of their service-oriented predecessors (for example, XML-RPC), current-day Web APIs are no less powerful in terms of their potential to deconstruct enterprise software and data into its most basic units of logic. In turn, developers are empowered to reuse and recompose those units—like building blocks—into innovative, game-changing applications. The result is an agile enterprise in which any one of these "blocks" can be rebuilt without disrupting others.

On the consumption side, developers can now draw from a nearly infinite palette of internal and public APIs in an effort to outsource almost all of their application functionality. The days of heavy lifting in the form of writing millions of lines of code—while often reinventing the wheel—are over. Savvy developers know where the APIs are and how to put them to use in a way that enables iterative and agile application development. This allows organizations to get their wares to market faster and for less money than those that have yet to embrace the benefits of APIs.

But while the API model itself has almost unlimited potential, the first wave of APIs was focused on relatively simple functionality. Today, a new generation of API provider is packing the power of PhD-grade science, natural language processing, deep learning, image recognition and the semantic Web into something as simple as a basic RESTful API call. One such provider, Watson Developer Cloud, offers developers a portfolio of more than 28 APIs delivering just this kind of functionality. This makes it possible for organizations of all types and sizes to benefit from the experience, expertise and intelligence of a team of PhDs—without having to actually find and employ them.

The Fictitious News Media Inc.

This use-case deployment focuses on News Media Inc., a fictitious company that wants to transform its tedious manual content workflow into a more automated, cost-efficient process.

The company will use APIs available on Watson Developer Cloud to implement a more intelligent and orchestrated approach to tagging posts in its content management system based on context and keywords. In the past, this job would have required a team of academics to pore through reams of text in an attempt to properly classify content. But with the APIs from Watson Developer Cloud, there is no need for such a team—just the APIs that pack the power of such a team, and Web connection over which those APIs can be reached by News Media Inc.'s infrastructure.

About News Media Inc.

A fictitious blogging/publishing company, News Media Inc., publishes multiple news articles on a daily basis. The articles, authored by a number of different writers, are created in Microsoft Word and sent to an editor via email. The editor reviews the articles and manually enters each one into the content management system. There, the editor adds relevant tags to each article, based on keywords, companies mentioned in the article, and so on.

The Problem

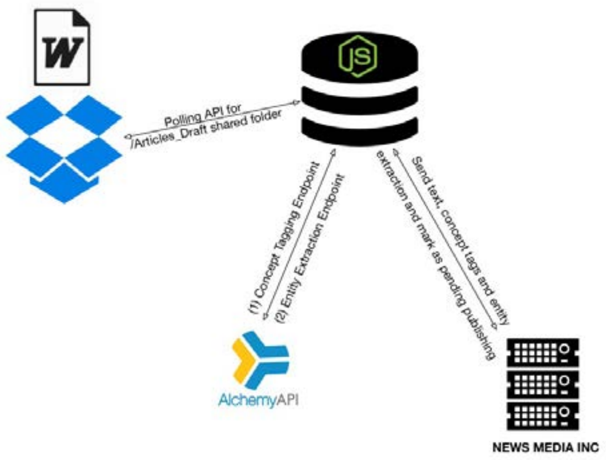
This process is error-prone and time-consuming, which has prompted News Media to investigate a faster, more efficient solution.

Solution Implementation

Solution Overview

After some research, News Media has decided to utilize third-party APIs. The company's core competency is content creation, not content parsing. So it only makes sense to automate the detection of new articles and outsource the parsing and tagging to an API provider that specializes in natural language processing: Watson Developer Cloud.

The proposed solution calls for a Node.js server that will act as an intermediary between the Dropbox folder where new articles reside and the content management system. The server will retrieve files, extract their content, submit that content to a Watson Developer Cloud API, and then pre-loads a content management system with the new content, appropriately tagged.


Polling API for
/Articles_Draft shared folder
(1) Concept Tagging Endpoint
(2) Entity Extraction Endpoint
Send text, concept tags and entity
extraction and mark as pending publishing
AlchemyAPI
NEWS MEDIA INC

To achieve this, the Node.js server will poll the Dropbox folder regularly, and if a new Word document appears, or is updated, the server will then extract the entire text.

The text will then be sent for processing by two APIs:

1. *Concept Tagging API*—extracts the concepts that will auto-populate an array of comma-delimited tags that, in turn, are supplied to the CMS along with the article text.

2. *Entity Extraction API*—used to identify and extract the name of the organization mentioned in the article. The organization field associated with each article in the CMS will then be populated.

Once both the tag arrays and organization parameters have been returned, these parameters—along with the entire article text—will then be supplied to the company's internal CMS through its API. Editors are then automatically notified by email to review the articles prior to publishing.

Out of Scope

The focus of this paper is on demonstrating the interaction between the Node.js-process and the Watson developer endpoints; the two other vendor endpoints—Dropbox and the internal News Media CMS—will be briefly mentioned, to provide greater contextual understanding of the overall workflow.

Our code is not optimized for elegance, performance or sophisticated error trapping. Rather, it provides a brief example of the simplicity involved in calling the APIs offered by Watson Developer Cloud.

It is assumed that the *Dropbox* polling Restful API provides continuous polling. Upon detecting new file additions to a specific shared folder location, it will return the entire text extracted from a Microsoft Word document, at which point the Node.js process calls the two endpoints.

Node Server (Backend Server)

Node.js serves as the backbone and conduit server in this authoring workflow. The lightweight and scalable Javascript-powered server will interact with the Dropbox API, orchestrate the returned text through two endpoints, return the results, and structure them into meaningful parameters before packaging the final request to the company's internal CMS server endpoint.

As with any Javascript code project, we must initialize our environment by identifying the sources of our code and any other frameworks that the project depends on. We start our project off by setting up the imports. We reference our own external Javascript file (newsmediaAP.js), which contains the code behind our automated publishing workflow; a module from Dropbox for interacting with the Dropbox API via Node.js; and the Request and Querystring modules for working with query strings.

```
'@import newsmediaAPI.js';

var Dropbox = require('dropbox');
var request = require('request');
var querystring = require('querystring');
```

ProgrammableWeb Code Sample

Dropbox (Document Repository)

First we set up two global variables: one that will hold our tags and one that will hold the name of the organization. We make use of the Dropbox Node.js module to establish an authentication session. (Note, where keys and secrets are mentioned, you must supply your own data.)

```
var tagsArray = undefined;
var organization = undefined;

// app key and app secret are placeholders
var dbClient = new Dropbox.Client({
    key: DROPBOX_APP_KEY, secret: DROPBOX_APP_SECRET, sandbox: true
});
```

ProgrammableWeb Code Sample

```
dbClient.authDriver(new Dropbox.AuthDriver.NodeServer(8191));
dbClient.authenticate(function(error, client) {
    console.log('token ', client.oauth.token);
    console.log('secret', client.oauth.tokenSecret);
});
```

ProgrammableWeb Code Sample

Dropbox provides an API that enables News Media to *long-poll* a shared Dropbox folder (one that News Media's writers are saving their articles to). Via this long-polling, our Node.js server is notified when files in that shared folder are added or changed.

Once the changes have been accounted for, our Node.js process receives a filename of the file that has changed. That filename, referred to as *changedFileName* in our code, is then passed to Dropbox's dbClient.readFile routine (sourced earlier when we "required" Dropbox). If there is an error, we break out of the routine. If not, we stringify the text (stored in data) and store the results in the variable *content*.

```
// get the wordFile from Dropbox
dbClient.readFile(changedFileName, function(error, data) {
    if (error) {
        return console.log(error);
    }

  var content = data.toString('utf8');

// submit content for discovery of first organization
 discoverOrganization(content);

// submit content for discovery of concept tags
 tagForConcept(content);

// add content and associated tags, organization to CMS
    if(tagsArray != undefined && organization != undefined) {
        callCMS(content, tagsValue, organization);
    }
});
```

ProgrammableWeb Code Sample

We subsequently call both the discoverOrganization tagForConcept routines, passing the content variable to both. Finally, assuming nothing has gone wrong, we call the content management system with the article's text, the tags and the organization as parameters.

API Endpoints (NLP Parsing)

Upon processing the *Dropbox* tasks, and successfully receiving the changed article in text format and storing it within the *content* variable, it is time to call the first of the two Watson Delevoper Cloud APIs: Entity Extraction, in order to determine the organization that is associated with the article. Note that while our code only discovers the first organization for demonstration purposes, the Entity Extraction API is capable of discovering multiple organizations in one pass.

The endpoint for calling the Entity Extraction API is:

http://gateway-a.watsonplatform.net/calls/text/TextGetRankedNamedEntities

For the text version of the endpoint we are using, it accepts the following parameters:

- *apiKey*
- *text*
- *maxRetrieve (max number of concept tags to retrieve)*
- *outputMode (choice of json, XML or rdf)*

First, we need a little sub-routine to gather up these parameters package them with the endpoint into a URL.

ProgrammableWeb Code Sample

```
// routine to construct URL with endpoint and parameters
function buildURL(text) {
 // actual API key is retrieved from the Watson Developer API portal and is added
    with quotations
 var apikey = YOUR_API_KEY;
 var options = {
                    outputMode:"json",
                    apikey:apikey,
                    text:text,
                };
 var params = querystring.stringify(options)
 // concatenate endpoint URL with parameters
 var url = "http://gateway-a.watsonplatform.net/calls/text/TextGetRankedNamedEntities?"
 + params
        return url
}
```

Then comes our routine for calling the endpoint with the previously stringified text.

```
function discoverOrganization(text){
    // build the URL
        var url = buildURL(text);
     // call the API, setup the callback
       request(url, function (error, response, body) {

                if (!error && response.statusCode == 200) {
            /* load responseOBJ with parsed API call result */
                  var responseObj = JSON.parse(body)

                if (responseObj["status"] !== "ERROR") {
              /* find the first organization in response
                set global variable to that organization */
                          organization = firstOrg(responseObj["entities"]);
                }
                }
       });
};

function firstOrg(responseArray) {

       // step thru array until first organization is discovered

       for(var i = 0; i < responseArray.length; i++) {
                var object = responseArray[i];
          // not all objects returned by Entity Extraction API are organizations
                if(object["type"] === 'Organization') {
                          console.log(object);
                          return object;
                }
       }
       return null
}
```

ProgrammableWeb Code Sample

We pass the required parameters to the /TextGetRankedNamedEntities endpoint, capture the response as a JSON array, extract the first organization that appears in the array, and store it in the global variable organization. The JSON array that is returned from the API might look like this:

ProgrammableWeb Code Sample

```
{
    "status": "OK",
    "statusMessage": "HTTP retrieval from domain www.programmableweb.com
rate limited for 1 seconds",
    "usage": "By accessing AlchemyAPI or using information generated by
              AlchemyAPI, you are agreeing to be bound by the AlchemyAPI Terms
              of Use: http://www.alchemyapi.com/company/terms.html",
    "url": "http://www.programmableweb.com/news/google-introduces-voice-
           interaction-api-to-initiate-app-tasks/2015/06/09",
    "totalTransactions": "2",
    "language": "english",
    "text": "At Google I/O 2015, Google announced the Voice Interaction API on
            Android M.\nGoogle has introduced the Voice Interaction API as part
            of the recently announced Android M. The API enables apps to interact
            with users via spoken dialogue. While Google has supported voice
            commands and text to speech for some time, the API creates a true
            dialogue. For example, if a user engages with a smart home app using
            the command \"Turn on the lights,\" the Voice Interaction API can
            respond accordingly by inquiring as to which room the user means.
            \n\"I can't wait for the day when I can talk to my watch and phone like
            they do in the movies,\" Sunil Vemuri, Google's voice actions product
            manager, explained in an introductory YouTube video. \"We're still a
            ways away from that. But the good news is [Google is] releasing a new
            API on Android M that takes us an important step in that direction.
            \"\nMany anticipated the release of a Voice Access API at Google I/O
            2015. However, the Voice Access API was removed from the schedule.
            Instead, the Voice Interaction API was released as part of Android M.....\n",
    "entities": [
        {
            "type": "Company",
            "relevance": "0.863244",
            "sentiment": {
                "type": "positive",
                "score": "0.284421",
                "mixed": "1"
            },
            "count": "12",
              "text": "Google",
            "disambiguated": {
                "subType": [
                     "AcademicInstitution",
                     "AwardPresentingOrganization",
                     "OperatingSystemDeveloper",
                     "ProgrammingLanguageDeveloper",
                     "SoftwareDeveloper",
                     "VentureFundedCompany"
                ],
                "name": "Google",
                "website": "http://www.google.com/",
                "dbpedia": "http://dbpedia.org/resource/Google",
                "freebase": "http://rdf.freebase.com/ns/m.045c7b",
                "yago": "http://yago-knowledge.org/resource/Google",
                "crunchbase": "http://www.crunchbase.com/company/google"
            }
        },
        {
            "type": "OperatingSystem",
            "relevance": "0.421659",
            "sentiment": {
                "type": "positive",
                "score": "0.121432",
                "mixed": "1"
            },
            "count": "5",
            "text": "Android"
 },
```

ProgrammableWeb Code Sample

```
        {
            “type”: “Person”,
            “relevance”: “0.265637”,
            “sentiment”: {
            “type”: “positive”,
                “score”: “0.627101”
            },
            ...
            “count”: “1”,
            “text”: “product manager”
        }
    ]
```

ProgrammableWeb Code Sample

In the example above, we extracted the first organization entity (type Company), Google, omitting any irrelevant entities such as person.

With the global variable organization now loaded, we still have to analyze the text for any concept tags. The endpoint for concept tagging is:

http://gateway-a.watsonplatform.net/calls/text/TextGetRankedConcepts

For the text version of the endpoint we are using, it accepts the following parameters:

- *apiKey*
- *text*
- *maxRetrieve (max number of concept tags to retrieve)*
- *outputMode (choice of json, XML or rdf)*

```
function buildTagsURL(text) {
    // set api key obtained from the developer portal
    // note that the actual key must be contained in quotes
    var apikey = YOUR_API_KEY;

    // maxRetrieve parameter not include (optional)
    var options = {
                outputMode:”json”,
                apikey:apikey,
                text:text,
                };
```

ProgrammableWeb Code Sample

```
    var params = querystring.stringify(options)
    // concantenate endpoint URL with parameters
    var url = "http://gateway-a.watsonplatform.net/calls/text/TextGetRankedConcepts?"
+ params
    return url
}
```

ProgrammableWeb Code Sample

Now that our URL-building routine is done, we need a routine for calling the Concept Tagging API with the parameterized URL.

```
function tagForConcept(text){

    // establish parameterized endpoint URL
    var url = buildTagsURL(text);

    // call endpoint, setup callback
    request(url, function (error, response, body) {

        if (!error && response.statusCode == 200) {

            // set responseObj to parsed call result
            var responseObj = JSON.parse(body)

            if (responseObj["status"] !== "ERROR") {
                // set global tagsArray array to returned list of concepts
                tagsArray = responseObj["concepts"];
            }
        }
    });
};
```

ProgrammableWeb Code Sample

News Media Inc. CMS

At this point, we have:

- The complete article text stored in the global variable content
- The organization associated with that text stored in the global variable *organization*
- A list of concepts associated with the text stored in the global variable *tagsArray*

Updating the CMS is all that is left to do. Looking back to the parent routine at the top of this code section, it calls the following sub-routine:

```
function callCMS(text, concepts, org) {
    console.log(concepts);
    console.log(org);
    request.post({url:'http://newsmediainc-cms.com/tag/upload', form: {text:text,
    tags: concepts, organization: org}}, function(err,httpResponse,body){
        // console.log(body);
    });
}
```

ProgrammableWeb Code Sample

Again, note that the aforementioned API call to our CMS is not specific to any particular CMS. It simply assumes that there is a CMS that accepts an API structured to receive text, concepts and an organization.

In a real-world scenario, an organization deploying this sort of workflow would have to decide which of the Watson Devloper Cloud APIs it wanted to fully leverage and analyze its existing CMS's API to see how capable it is of receiving those inputs. Depending on that API's capability, the organization may have to design and implement a custom API to handle all the inputs.

Lastly, as mentioned earlier, this code sample is not optimized for performance, elegance, security or sophisticated error trapping. Nor is it the only way to accomplish the stated objectives. For example, instead of updating the CMS with the new article and tags all at once, the workflow could have published the article first and circled back to add the tags later. Because of the way this API can target a text object as easily as it can target a URL, this would have been easy to do.

Solution Conclusion

Though not shown here, the final CMS call would set a flag and trigger an automated email to the editors, letting them know that the CMS has a new article that is ready to be proof-read. Thanks to the capabilities automated by the APIs, the only thing left for the editors to do is approve the post with the click of a button.

This graceful solution eliminates the need to manually check emails and save or copy attached documents to the News Media CMS.

In the future, News Media could use other APIs from Watson Developer Cloud to further streamline its content publishing workflow while enhancing the way it classifies its content. These efforts could continue to improve the content's utility and searchability for the media company's audience.

About This Report

This paper was produced by ProgrammableWeb on behalf of IBM Corporation as a part of ProgrammableWeb's custom content program. It's technical content (the source code and explanations) were excerpted from a more comprehensive whitepaper that is available for download. Questions about this paper's content or ProgrammableWeb's program should be directed to editor@programmableweb.com. For more information about IBM's Watson Developer Cloud, please contact the IBM Watson Developer Cloud community.

ProgrammableWeb

Please Note:

Certain passages and segments of this paper include references to "Alchemy", "AlchemyAPI", and IBM's Watson Developer Network. Shortly after the content for this paper was produced, IBM Watson acquired AlchemyAPI. Where possible, some occurrences of "Alchemy" have been edited to reflect AlchemyAPI's inclusion in IBM's portfolio of Watson Developer Cloud's APIs. The phrases "Alchemy" and "AlchemyAPI" may still appear in certain source code and endpoint references where it was still syntactically correct at the time of this paper's production.