Chapter 3: Creating your first Watson app

Learning Bluemix & Cognitive

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales

Chapter 3: Building your first Watson App

- Setting up your workstation
- Clone the ‘Class’ repository
- Getting your Speech to Text credentials
- Creating the User Experience
- Creating a simple service in NodeJS
- Connecting your app to NodeJS
- Connecting your app to Watson Speech to Text
- Talking to your computer/iPhone

Setting up your workstation for nodeJS: Windows

- There is a simple two step process to install nodeJS on Window:
    1. Go to https://nodejs.org and click on the LTS (Long Term Support) image.
       at the time of this writing, the LTS version is 4.5.0

       This will download a file named node-v4.5.0-x64.msi

    2. Go to your Downloads folder (or other location if you told Windows to save the file somewhere else) and open the node-v4.5.0-x64.msi file.
       This will start the Windows Program Installer.

- You're done!

Setting up your workstation for nodeJS: OSX

- I recommend that you start by installing the “homebrew” package manager for OSX. You’ll need to be connected to the internet to do this. Go to this site (http://brew.sh)and follow the directions to install Homebrew.
  - Type this into your terminal window:
  - `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)”`
  - If brew is already installed, you’ll get this message:

```
• It appears Homebrew is already installed. If your intent is to reinstall you
• should do the following before running this installer again:
•     ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/uninstall)"
```

  - update Brew
    - `brew update`
  - Install the Node Package Manager (npm)
    - `brew install npm`

Setting up your workstation for nodeJS: Linux

- The following installation instructions are based on the Ubuntu Linux platform and use the “terminal” window.

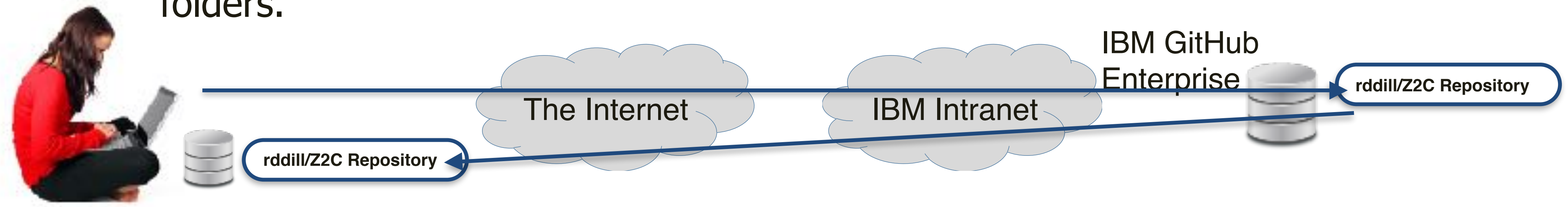
1. Open a terminal window and enter the following commands:
    1. `sudo apt-get update`
        - enter your log-in password when prompted
    - `sudo apt-get install curl`
        - enter your log-in password if prompted
    - `curl -sL https://deb.nodesource.com/setup_4.x | sudo -E bash -`
        - enter your log-in password if prompted
    - `sudo apt-get install -y nodejs`
    - create a symbolic link so that you can use either node, or nodejs to invoke the program:
        - `sudo ln -s /usr/bin/nodejs /usr/bin/node`
    - Verify the installation:
        - `npm -v` (should return a value like `v2.15.8`)
        - `node -v` (should return a value like `v4.4.47`)

Getting the code: step 1

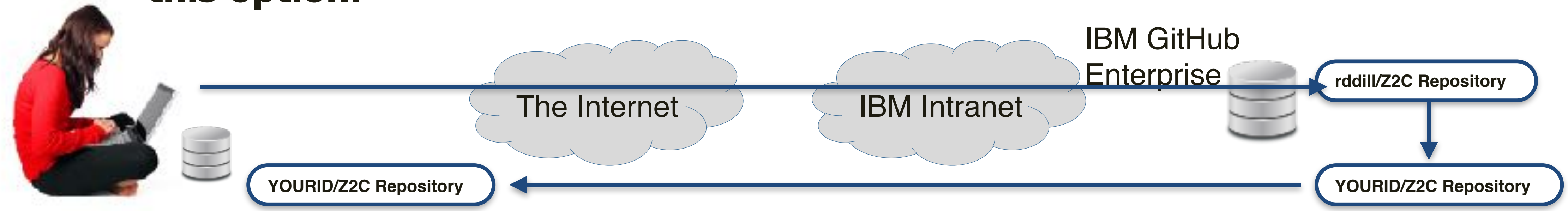
This activity has the pre-requisite that you have already successfully installed Git and completed the process to allow Git to authenticate to our GitHub Enterprise service.

You have two options to retrieve information from GitHub:

1. Clone the repository. This makes a copy of the repository on your laptop. If you want to work with this in your own repository, you would then copy the contents of the clone into your own folders.



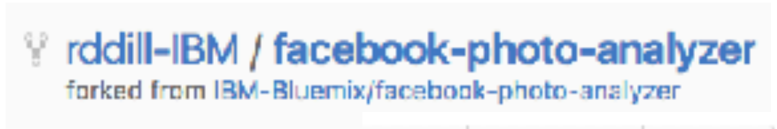
2. Fork the repository. This creates your own personal version of the repository and allows you to make updates and store your updates in your version of the Z2C repository. **We will use this option.**



Getting the code: step 2

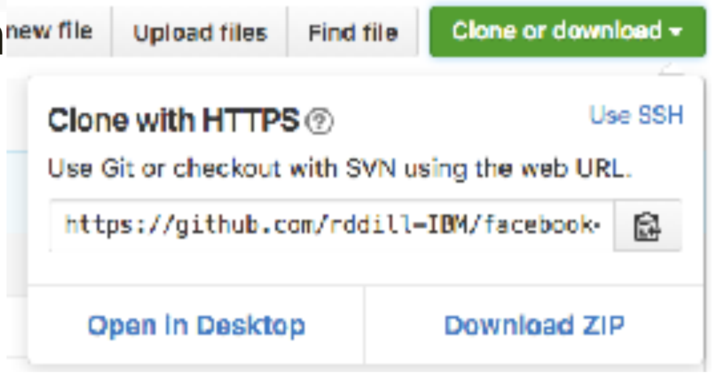
You've successfully forked the repository… what happened?

1. GitHub created a copy of the repository, prefaced by your github short name. You'll see this in the top left corner of the new github browser page. For example, when I fork the Bluemix/ photoanalyzer repository, I see the following:



rddill-IBM / facebook-photo-analyzer
forked from IBM-Bluemix/facebook-photo-analyzer

2. Clone YOUR repository to your desktop by pressing the (green) Clone button the right hand side of the web page and then selecting "Open in Desktop". GitHub Desktop will automagically open and show you where the file is going. Press Enter



Use Git or checkout with SVN using the web URL.

https://github.com/rddill-IBM/facebook-

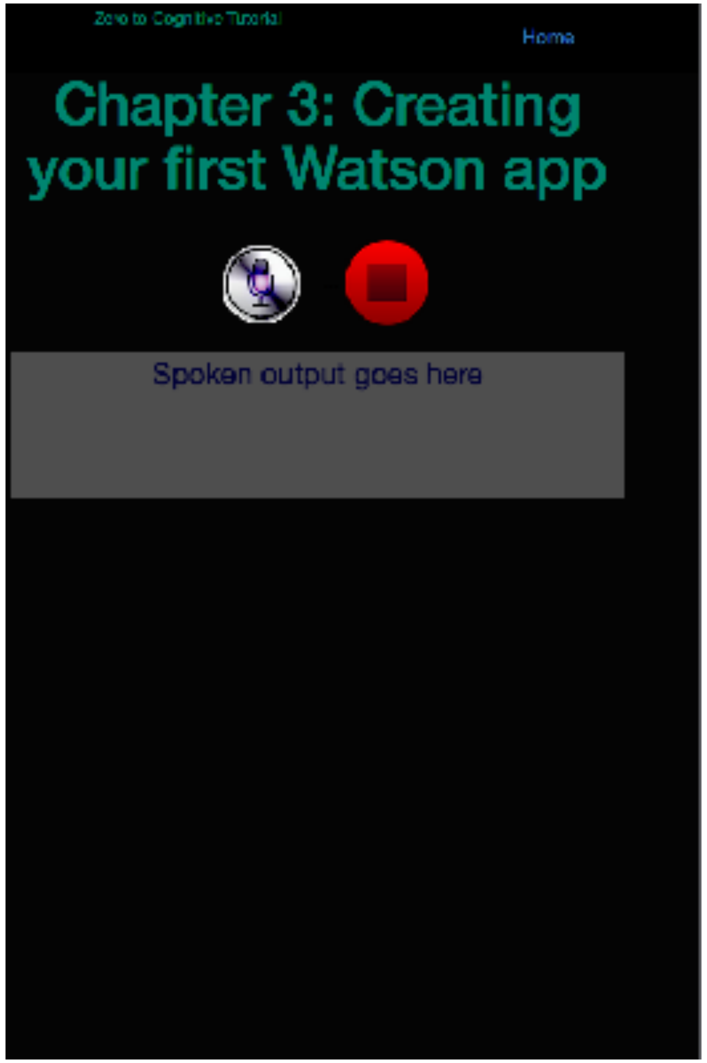
3. On Windows and OS X, git defaults to the following path: Documents/GitHub/your-repository-name

Set up Node and the Watson SDK

You’ve got the code on your system, now let’s make it work locally.

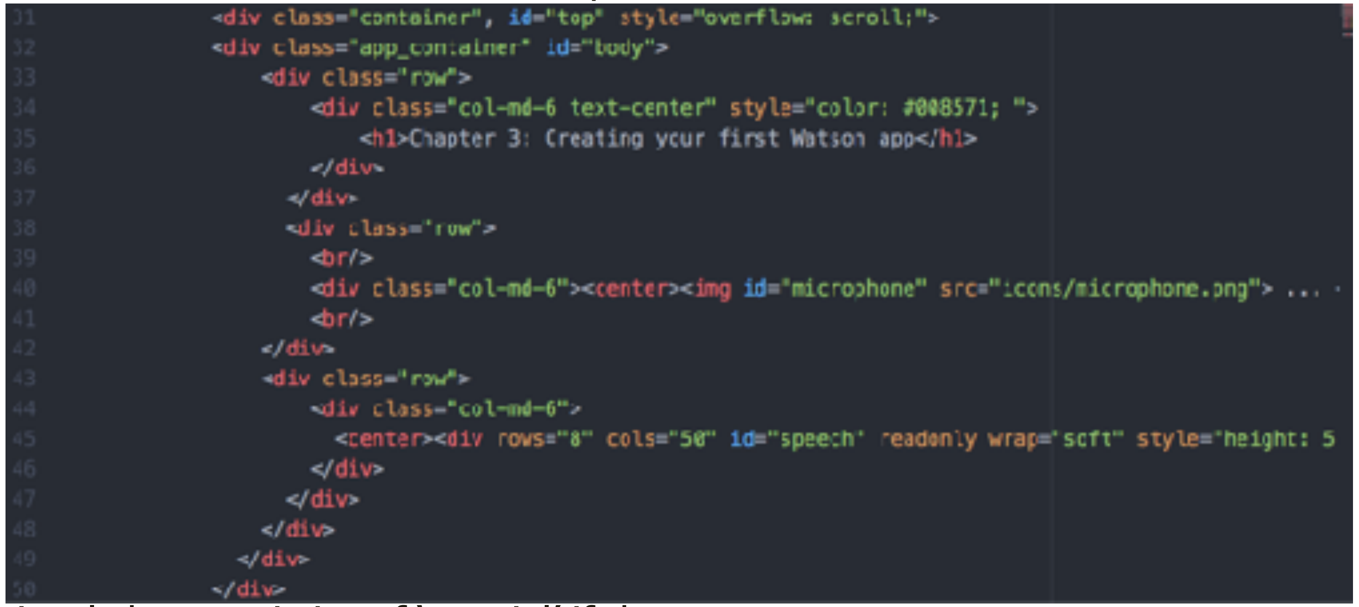
1. Open a terminal window and change to the directory for your new, forked, repository.
    1. `cd ~/Documents/GitHub/ZeroToCognitive/Chapter03/`
    2. `npm install`
        1. This tells npm to look at your ‘package.json’ file and make sure that everything in that file is available to node.
    3. `npm install watson-developer-cloud —save`
        1. This tells npm to go find a node package called ‘watson-developer-cloud’ and install it locally. Then add an entry to your package.json file. This becomes important later on.
    4. Did it work? Type the following command into your terminal window:
        1. `node index.js`

           This should respond with “z2c-chapter03 is listening on port: 6003”
        2. This means that you should be able to point your browser to `localhost:6003` and see the page to the right.
    5. Let’s look at the code … and then make it do something.

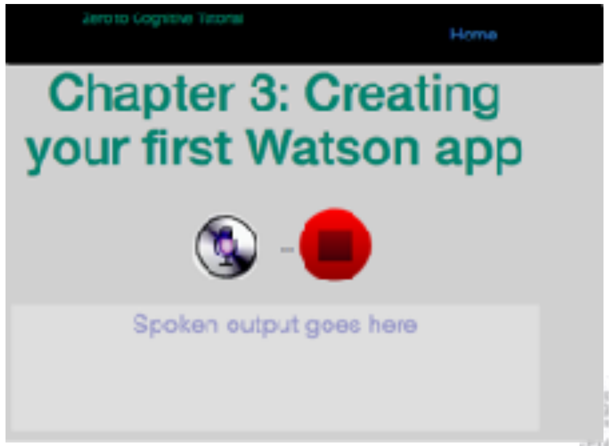

Zero to Cognitive Tutorial
Home
Chapter 3: Creating your first Watson app
Spoken output goes here

The html file

- The default file loaded by a server is “index.html”. You’ll find it in the “Chapter03/HTML” folder.
- The top navigation bar is managed by lines 9 to 30 … we’ll come back to them in a later chapter.
- What you see in the browser is managed by lines 31 to 50
- Lines 53 to 59 provide some framework support and manage the colors and shapes which you see.
- Let’s start with a simple change.
  update the background from near black to a lighter color
- Open index.html in your favorite browser and search for body class=“tutorial"
- The word tutorial tells your browser that it can update the visual characteristics of ‘tutorial’ if the CSS (Cascading Style Sheet) file has an entry for it.
- Open the pageStyles.css file in the Chapter03/HTML/CSS folder and search for the word tutorial. You’ll see the following line: `.tutorial {background: #040404; color: #000000;}`
- Change the `#040404` to `#D0D0D0` and save the file to the same location and name.
- “Shift-reload” your browser (pressing the Shift key while clicking on the reload button tells the browser to go back to the server.
- Your page now has a new background. Congratulations.



```
31          <div class="container", id="top" style="overflow: scroll;">
32          <div class="app_container" id="body">
33              <div class="row">
34                  <div class="col-md-6 text-center" style="color: #008571; ">
35                      <h1>Chapter 3: Creating your first Watson app</h1>
36                  </div>
37                </div>
38                <div class="row">
39                  <br/>
40                  <div class="col-md-6"><center><img id="microphone" src="icons/microphone.png"> ...
41                  <br/>
42              </div>
43              <div class="row">
44                  <div class="col-md-6">
45                    <center><div rows="8" cols="50" id="speech" readonly wrap="soft" style="height: 5
46                  </div>
47                </div>
48              </div>
49            </div>
50          </div>
```


Zero to Cognitive Tutorial
Home
Chapter 3: Creating your first Watson app
Spoken output goes here

Making the page do something…

- Nothing happens when anything on the page is clicked. We need to create function on the server and in the browser to make the web page work.
- Let's look at the server code, Chapter03/index.js which has 4 basic parts:
    - lines 19-32 define core variables and ensure that base capability is available
    - lines 34-47 set up the application for your local environment.
        - we'll update lines 38-41 to move between Bluemix and local execution.
        - Later we'll write a routine to automate that switch
    - lines 54-57 create the web server. (really, only 4 lines of code????)
    - lines 59-73 handle file loading for the server (html, icons, javascript, css)
- On the client side, everything gets kicked off by the browser after the web page has completed loading. This is defined in the "`<body>`" tag by stating: `onLoad="initPage()"`
- So where is initPage() and what does it do?
        - look in z2c-speech.js (`HTML/js/z2c-speech.js`) and you'll seen an empty routine, which looks like this:
        - `function initPage () { }`
- We need to do the following things:
    - monitor the microphone and stop buttons
    - connect to Watson and display what is said.

Activating the microphone

1. Enable the microphone, disable the stop button:

```
    var _mic = $('#microphone'); var _stop = $("#stop");
    _mic.addClass("mic_enabled");
    _stop.addClass("mic_disabled");
```

Monitor the microphone button for a click:

```
  _mic.on("click", function ()
    {
      var _className = this.className;
      if(this.className == "mic_enabled")
      {
        _mic.addClass("mic_disabled");
        _mic.removeClass("mic_enabled");
        _stop.addClass("mic_enabled");
        _stop.removeClass("mic_disabled");
        $.when($.get('/api/speech-to-text/token')).done(
          function (token) {
            stream = WatsonSpeech.SpeechToText.recognizeMicrophone({
               token: token,
               outputElement: '#speech' // CSS selector or DOM
Element
             });
            stream.on('error', function(err) { console.log(err); });
          });
        }
      });
```

1. Enable the microphone, disable the stop button:

   These two add class statements are monitored by CSS and change how the button looks

2. Monitor the microphone button for a click:

   Do something when the microphone is clicked

   But ignore the click if the microphone has been disabled by the app.

   - Disable the microphone
   - Enable the stop button
   - Remove conflicting class
   - Remove conflicting class

   Call the server using an http get and go to the /api/speech/ folder and execute token

   Give the token to the Watson API to monitor the microphone

   Direct output to go back to the #speech element in the index.html file

   If there is an error, print the error message to the browser console log.

Activating the microphone

1. Monitor the Stop button for a click:

```
  _stop.on("click",  function() {
    console.log("Stopping text-to-speech service...");
    if (stream != undefined) {stream.stop(); }
    _mic.addClass("mic_enabled");
    _mic.removeClass("mic_disabled");
    _stop.addClass("mic_disabled");
    _stop.removeClass("mic_enabled");
});
```

1. Monitor the stop button for a click:
   Do something when the microphone is clicked

   write a message to the log file to record the click
   If a stream is active, stop processing voice.
      enable the microphone button
      disable the stop button
      remove conflicting class
      remove conflicting class

You will have noticed that there are quite a few places in this code where the code includes “`$(`“ or “`$.`” This is a javascript framework called `jQuery`. We’re using jquery version 3.1.0 in this demo. The demo is not dependent on that specific version and you are welcome to upgrade to the latest version.
jQuery Foundation website: https://jquery.org/
jQuery code website: http://jquery.com

Activating Watson

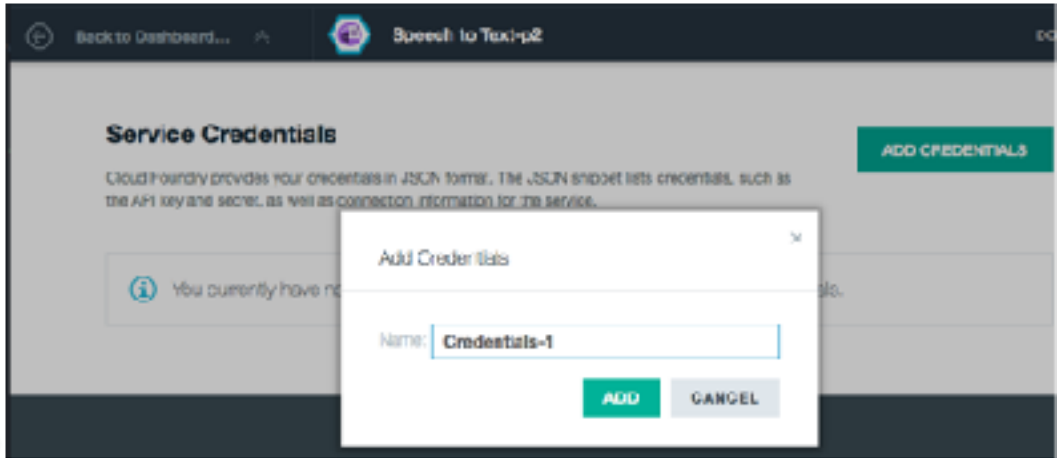
There are two actions to take here:

1. Get the Watson Speech-to-Text credentials from Bluemix
2. Write some code to access those credentials and connect to Watson Speech-to-Text

Get the Watson Credentials from Bluemix

1. Log in to your Bluemix account
2. Click on Dashboard
3. Click on the space you created
4. Click on Services
5. Click on Watson Speech-to-Text
6. Click on Service Credentials
7. Click on Add Credentials
8. Copy the returned credentials into your index.js file

In the preceding section, we saw that the browser send a get request to "/api/speech/token". Since it doesn't exist, we need to create it.

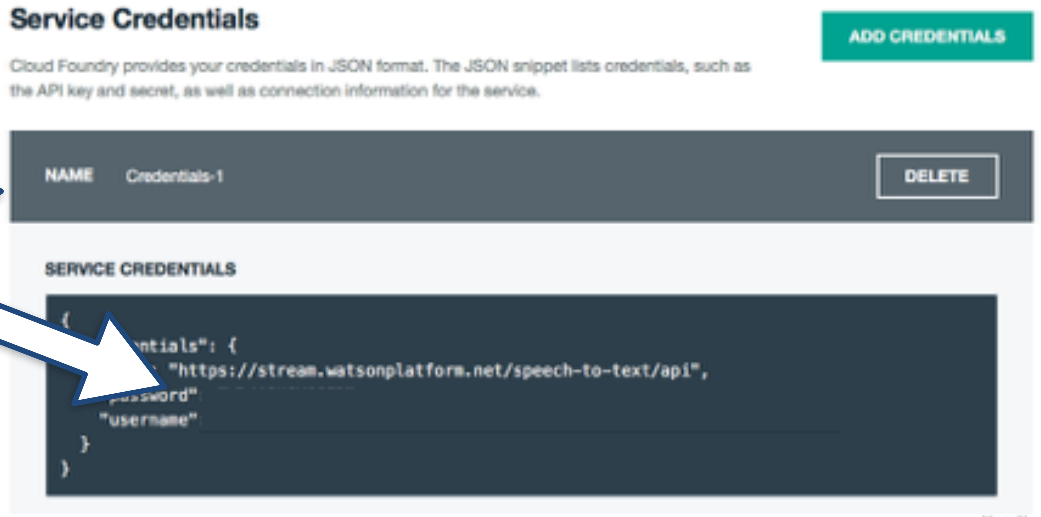


Service Credentials

Cloud Foundry provides your credentials in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

You currently have n

Name: Credentials-1



Service Credentials

Cloud Foundry provides your credentials in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

NAME Credentials-1

SERVICE CREDENTIALS

```
{
      ntials": {
        : "https://stream.watsonplatform.net/speech-to-text/api",
      password"
      "username"
   }
}
```

Creating the server code in NodeJS

- We have two basic approaches we can use to write the code on the server side.
  - put it directly in NodeJS, it's only 20 or so lines, so no big deal right?
  - Put it in a separate file
- Which to choose? I use architectural principles to help me make decisions like this. Several that I find invaluable on any project are:
  - Separation of concerns
  - Loose coupling
  - Design for reuse
- These tell me that it would be better to put this code in a separate file rather than putting it in the core index.js file. So, let's do that.
- In your github folder, you'll see the following directory structure:
  - controller/restapi/features
- In the features folder, you'll see one file: speech_to_text.js
- We need to tell index.js where to look for this file and we need to put real code in there.
- In index.js, add the following lines of code:

```
app.use('/', require("./controller/restapi/
router"));
```

- This tells node.js two things:
  - load the router.js file from the 'restapi' folder
  - Do this whenever a URL is requested.

```
var express = require('express');
var router = express.Router();
var speech_to_text = require('./features/
speech_to_text');

module.exports = router;router.get('/api/speech-to-
text/token*',speech_to_text.token);
```

- This is a node.js router service.
- Look in a subfolder called `features` for a file called `speech_to_text.js`
- Export `/api/speech-to-text/token*` as a known path to nodes (this is what the browser javascript code calls

The Watson Speech-to-Text code

```
var extend = require('extend');
var watson = require('watson-developer-cloud');
var vcapServices = require('vcap_services');

var config = require('../../env.json');

exports.token = function(req, res) {
   var sttConfig = extend(config.speech_to_text,
vcapServices.getCredentials('speech_to_text'));

   var sttAuthService = watson.authorization(sttConfig);

   sttAuthService.getToken({
      url: sttConfig.url
   }, function(err, token) {
      if (err) {
         console.log('Error retrieving token: ', err);
         res.status(500).send('Error retrieving token');
         return;
      }
      res.send(token);
   });
}
```

1. Include essential nodeJS services
2. get credentials to talk to Watson
3. Tell nodeJS how to get the token
4. Define the authorization service
5. Get the Token
6. Send the token back to the browser

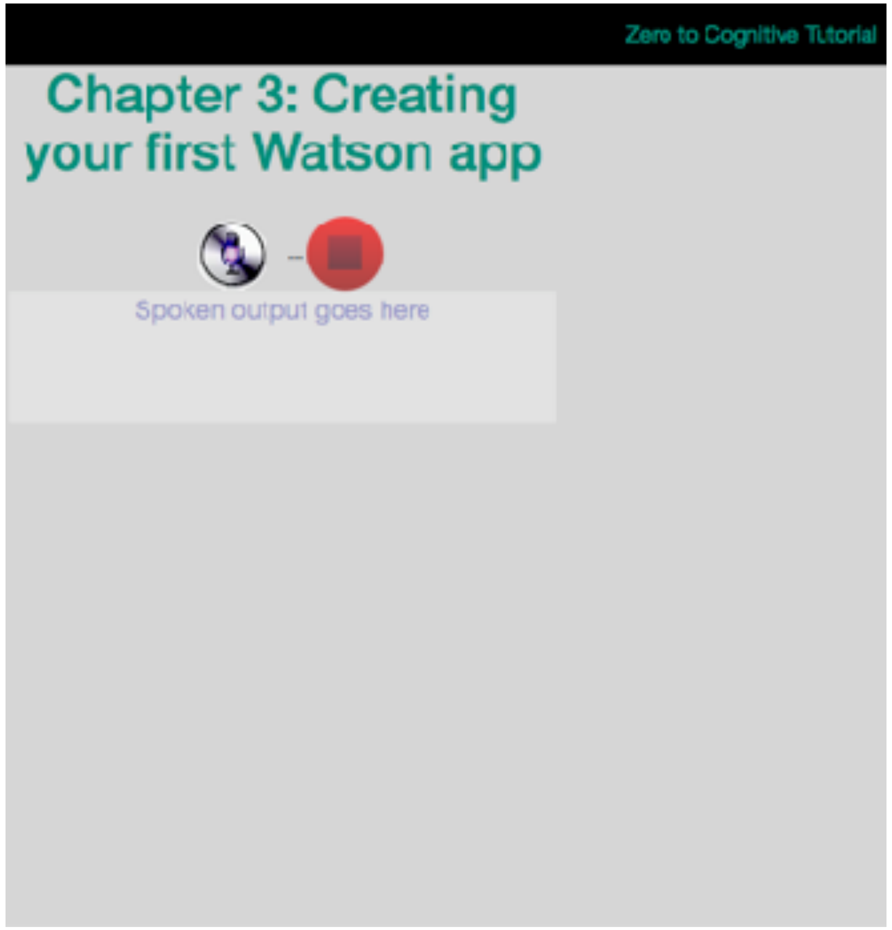
The env.json file

```
{
    "speech_to_text": {
        "version": "v1",
        "url": "https://stream.watsonplatform.net/speech-to-text/api",
        "username": “your.Speech-to-Text.user.name.from.Bluemix”,
        "password": “your.Speech-to-Text.password.from.Bluemix”
    }
}
```

Put your credentials here

You've done it!

- in your terminal window, type:
  `nodejs index.js`
- your application should respond with:
- Go to your browser (Chrome or FireFox. Safari does not work!) and type `localhost:6003` as your url. You should see a web page like the one to the right appear.
- Click on the microphone icon. Your browser should ask you if it's OK to use the microphone - make sure you pick the one you want to use. Please see this video if you're using Chrome to activate your microphone. http://w3.tap.ibm.com/data/medialibrary/additional/159501.mp4/60286/Enable_Microphone_in_Chrome.mp4
- Start Talking!
- Congratulations! You've completed your first Watson app!


Zero to Cognitive Tutorial
Chapter 3: Creating your first Watson app
Spoken output goes here

Saving your work

- Saving your work:
  - go to the terminal window running node-js
  - Stop nodejs (CTRL-C).
  - Type git status.
  - Using git add, add each of the changed files to git.
  - Type git status when you're done to ensure you didn't forget anything.
  - Type git commit –m 'WooHoo!! It works!!'
  - Type git push.

- Congratulations! You've now updated your personal repository with the latest version of your code.

Pushing your app to BlueMix

- Go to Bluemix and look carefully at the name for your speech to text service. It will look something like this:
  - Speech to Text-**aa** The '**aa**' part will be different.
- Open the manifest.yml file and add this line after **services**:
  - - Speech to Text-**aa**
- Replacing **aa** with whatever letters appear on your version of Bluemix
- and then save the file

- Open a Terminal Window and navigate to Chapter03
  - (cd ~/Documents/GitHub/ZeroToCognitive/Chapter03/)
- Delete the node-modules folder so that you don't load an extra 90Mb to Bluemix.
- Type cf login and enter your IBM intranet user id and password.
- Select the organization and space linked to your Zero-To-Cognitive work.
- Go to your terminal window and type in cf push and press enter.
- In 3-5 minutes, your app will be up and running on bluemix.
- Go to your Bluemix dashboard, go to your app and click on it. You'll see a URL highlighted at the top of the page. Click on it and it will load your app!!!

- Congratulations. You have now successfully built and deployed a Watson app on Bluemix!!!!

The Plan: 30 minute Chapters with an hour or two of practice

1. The Story, Architecture for this app	
2. Setting up Bluemix	
3. Building your first Watson App	(Watson Speech to Text)
4. Getting Watson to talk back	(Watson Text to Speech)
5. Understanding Classifiers	(Watson NLC)
6. Creating a custom dialog with Watson	(custom Q&A, session management)
7. Authentication	(puts C2 thru 6 together)
8. Alchemy News	(Watson Alchemy)
9. Visual Recognition and Images	(Watson Visual Recognition)
10. Watson Conversations	(Watson Conversations)
11. Rank & Retrieve	(Watson Alchemy + Rank & Retrieve)
12. Getting started on my first client prototype	Design Thinking, Stories, Architecture, Keeping it simple

Chapter 4: Getting Watson to talk back

Learning Bluemix & Cognitive

Bob Dill, IBM Distinguished Engineer, CTO Global Technical Sales