# Watson Conversation Services Setup

1. Create a new Watson Conversation service (I used Lite/Free)
2. Launch the configuration tool.
3. Select the upload button next to workspaces



1. Select the “CTA\_WatsonConversation.json” file from the demonstration setup files in the “Config Files” folder.
2. Click the deploy button  and make a note of the following:
   1. Username -
   2. Password -
   3. WorkspaceID -

# Watson Discovery Services Setup

## Create Document Repository

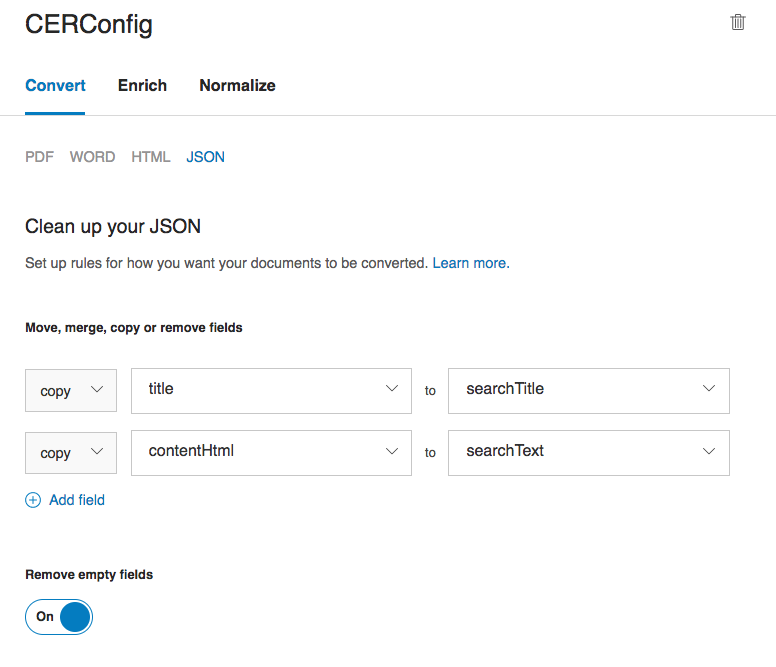
1. Setup a Box repository for use in your demonstration to display the source files in their original format.
2. Copy the files from the demonstration “Show Files” folder to the box repository.
3. Edit each JSON file in the “Ingest Files” folder and update the “sourceURL” to the appropriate document in your box repository.
   1. cwgeneral.json = Cooling Water General Document r0.pdf
   2. cwgeneral1.json = Cooling Water General Document r0.pdf
   3. cwgeneral2.json = Cooling Water General Document r0.pdf
   4. cwmaintmanual.json = Cooling Water Maintenance Manual r0.pdf
   5. cwmaintmanual2.json = Cooling Water Maintenance Manual r0.pdf
   6. cwopsmanual.json = Cooling Water Standard Operation Manual r0.pdf
   7. D-60251C2r0.json = D-6025IC2r0\_Searchable.pdf

## Create Service

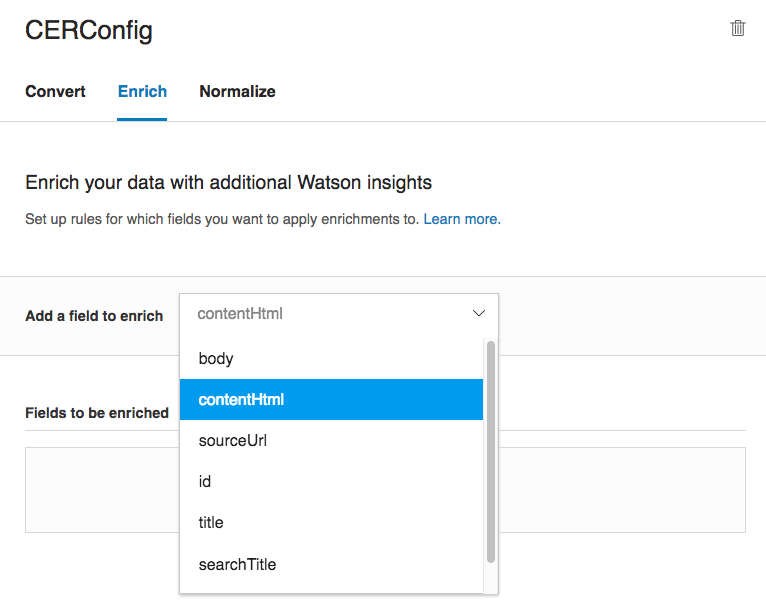
1. Create a new Watson Discovery Service (I used Free)
2. Launch the configuration tool.
3. Select “Create a data collection” – this will setup storage for you automatically.
4. Complete the collection details:
   1. Name = Cognitive Engineering Repository
   2. Configuration = Default
   3. Language = English
5. Go on to the Create Custom Configuration section of the setup document.

## Create Custom Configuration

1. From the main screen of your Cognitive Engineering Repository, click the button next to “Default Configuration”
2. Select 
3. Name the configuration CERConfig.
4. From the configuration screen, first, click to “browse from computer” to select some sample files we will use for the configuration. From the demonstration files in the Ingest Files folder, select the first two files (cwgeneral.json and cwgeneral1.json).
5. After they upload, select the “cwgeneral1.json” file and you should see a preview of it in the window.
6. Select the “Convert” tab. We are using JSON documents, so that is the only cleanup is needed. If you use other document types they will be converted into html, then into json and you have options to do clean up for both of those conversions. We will create two new fields that are copies of the original body and title fields, then we can use the copies in a later step to create a searchable text field.
   1. To create these fields, select “Add field”, set the action to “copy” and enter “title” to searchTitle, and repeat for “contentHTML” to searchText
   2. To see how the preview is affected by this step and ensure it is creating the new fields, click Apply & Save at the bottom of the page then select manual\_0.json. The right side preview pane should update to display the results of the changes.

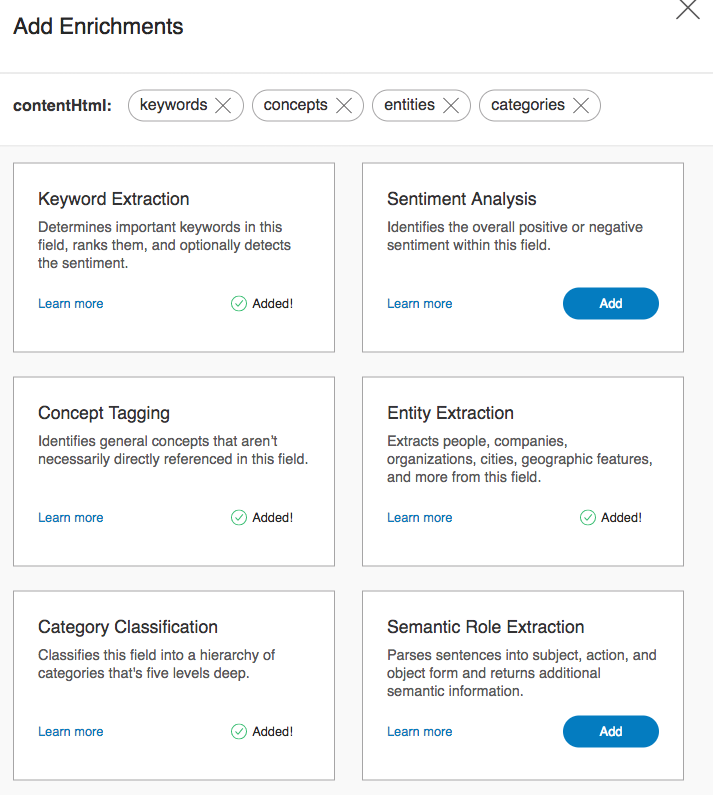


1. Next, select “Enrich”. Now we will setup the configuration for adding enrichments from Watson Natural Language Understanding on the text fields of the json documents. The enrichments add cognitive metadata and improve the search capabilities.
   1. First remove the existing “text” field that is being enriched with the default configuration by clicking the – to the right of the field.
   2. Under “Add a field” enter “ContentHtml” or select “ContentHtml” from the dropdown.. This is the field Watson will act upon to add enrichments.

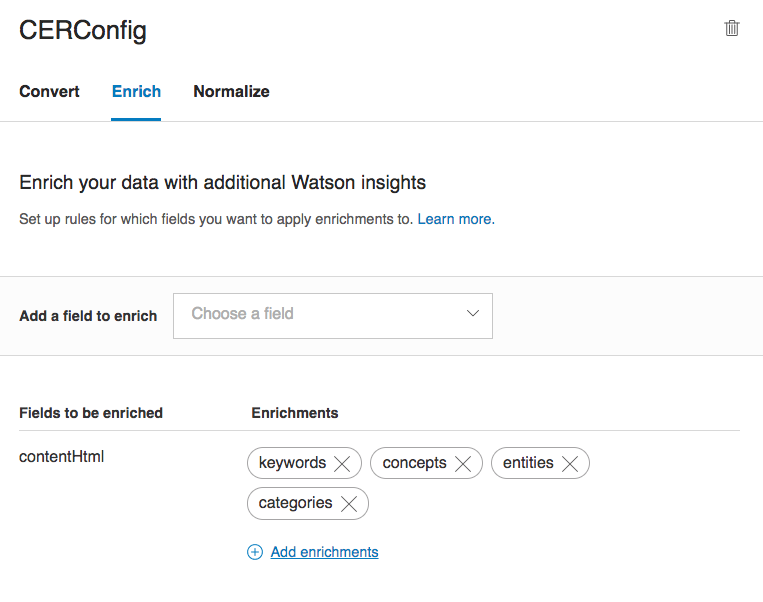


This will add a new field to be enriched below. You can then select the enrichments that should be applied to this field by clicking “Add enrichments”.

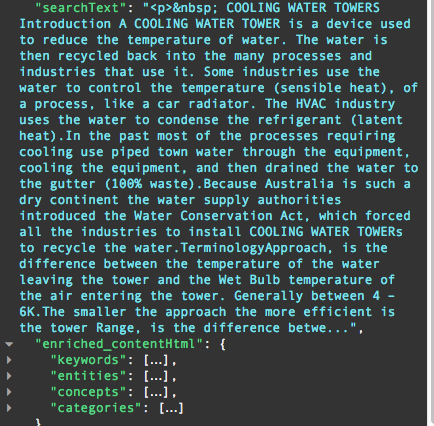
In this case we will apply Keyword Extraction, Concept Extraction, Category Classification, and Entity Extraction. These enrichments add meta data to the documents that help improve search. Select the “Add” button on each of these 4, then select “Done”.



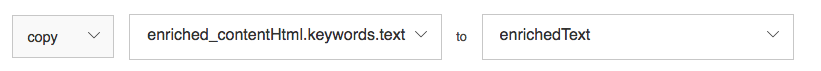
The final Enrich configuration should look like the picture below.



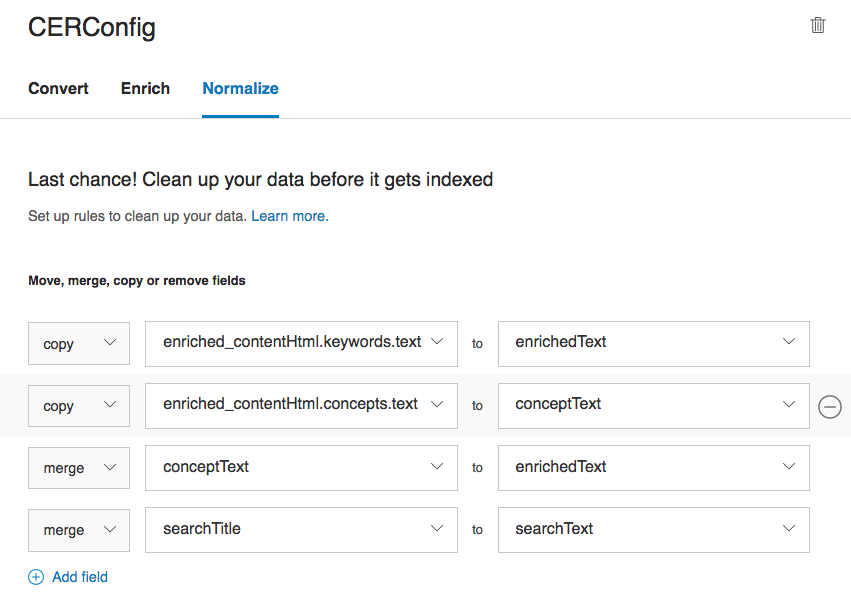
Now choose “Apply and Save” to see the results of the changes. There should be an “enriched\_contentHTML” field in the preview that shows all the applied enrichments over the document. You can expand the arrows to the see results of the enrichment.



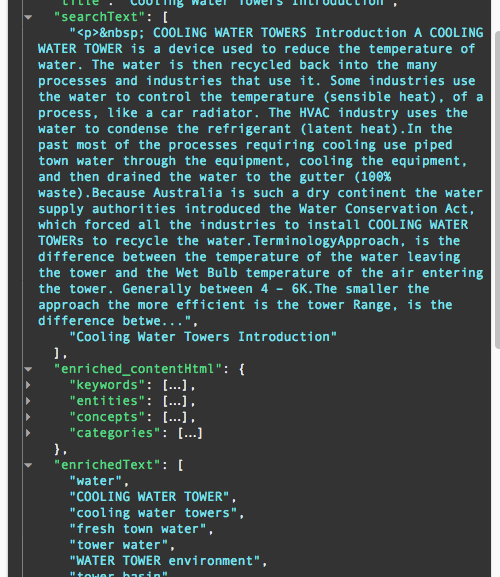
1. The last step is to Normalize the document, which allows you to clean up the data and fields to have a consistent structure for your queries and results. For these documents, we will create two fields, searchText that contains the combination of title and body, and enrichedText field that contains the combination of extracted concepts and keywords from the contentHTML field. To start the process, elect the “Normalize” option.
   1. First, we will copy the concepts and keyword text fields into intermediate fields.. Choose “Add Field”, then choose “copy” and enter the fully qualified path to the keyword text: “enriched\_contentHtml.keywords.text” and type in enrichedText in the new field name (it will prompt you to “add”).



* 1. Repeat this step for enriched\_contentHtml.concepts.text to conceptText
  2. Now we will merge these two intermediate fields into one (enrichedText). Select “Add a field”, choose “merge” and type in “conceptText” to “enrichedText”.
  3. Repeat the merge for the “searchTitle” and “searchText” fields.



Now select “Apply & Save” to store the updated the configuration and see the preview results. In the preview pane scroll to the bottom of the new document, and you should see a searchText field containing title + body and an enrichedText field containing a list of the concepts and keywords extracted from the data.



1. Once the configuration is complete, you can return to the main screen for the Cognitive Engineering Repository, and go on to ingest the sample documents to setup the repository.

## Ingest Documents

1. Make sure the Configuration is set to be “CERConfig” so the settings you created are applied to the entire set of documents being uploaded.
2. Select the option to “Browse from computer” under the “Add data to this collection” section.
3. Select all 7 json files in the demonstration files, in the “Ingest files” folder. It will take a few moments, then you should see a message that they were all uploaded.

## Query Collection

1. You can test out a query at this point using the “Query this collection button”. Then “Build your own query”.
2. Leave the option to “Use natural language”.
3. Type in “tell me about evaporation” then select “Run query”.
4. You should see passages returned related to evaporation (one of the default settings) as well as 4 matching documents.

## Gather Credentials

1. From the main screen of the Cognitive Engineering Repository, make a note of the following under API information:
   1. Collection\_id =
   2. Environment\_id =
2. Return to the Bluemix Service screen for Discovery Services. Click the “Service credentials” menu item. Then click “View Credentials” next to the first key name. Note the following:
   1. Username =
   2. Password =

# Setup Application

1. Download and install the [Cloud-Foundry CLI](https://docs.cloudfoundry.org/cf-cli/install-go-cli.html). This will be used to set up your Watson Services in Bluemix.
2. Connect the CLI to Bluemix by running the following:

cf api https://api.ng.bluemix.net

cf login --sso

1. Download and install the [Bluemix CLI](https://console.bluemix.net/openwhisk/learn/cli?env_id=ibm%3Ayp%3Aus-south). Follow instructions from the download site, except at the login you will need to use sso if an IBM employee, and use the appropriate API url for the region you are working in (NOTE: you may have to type in the organization and space name at this step, so be familiar with those):

bx login -a https://api.ng.bluemix.net -sso

1. Insure that you have a recent version of Node.js, (7.9.0 or higher) installed and npm. Please download and follow instructions from <https://nodejs.org/en/> or via package manager: <https://nodejs.org/en/download/package-manager/>.
2. We will use an existing Watson developer cloud project on github for the application code. Create a local working directory, then clone (or fork) the repository from https://github.com/watson-developer-cloud/conversation-with-discovery-openwhisk.git

# Cloud Functions

We will setup two actions using IBM Bluemix Cloud Functions and a sequence that controls the flow of these actions.

The first action is a call to the conversation API. When the user inputs text to the application it is sent to the Conversation service then the output is returned and sent to the next action.

The second action is for Discovery Service, which will look for the flag from conversation indicating if it should execute a query against its repository. If that flag isn’t set, it will return the results back to the user interface unchanged.

We will establish an API to communicate with the sequence from our front end application in the last step.

1. First, we will create two actions using javascript files from our repository. From your local root directory containing the source code for the application, run the following commands:

wsk action create conversation actions/conversation.js --web true

wsk action create discovery actions/discovery.js --web true

1. Next, edit the parameter files for each service to include your credentials for Conversation and Discovery Services.
   1. actions/conversationParams.json

{

"username": "<CONVERSATION\_USERNAME>",

"password": "<CONVERSASTION\_PASSWORD",

"workspace\_id": "<WORKSPACE\_ID>"

}

* 1. actions/discoveryParams.json

{

"username": "<DISCOVERY\_USERNAME>",

"password": "<DISCOVERY\_PASSWORD",

"environment\_id": "<ENVIRONMENT\_ID>", [NOTE: Source Code from git erroneously says workspace id]

"collection\_id": "<COLLECTION\_ID" [NOTE: Source Code from git erroneously says environment id].

}

1. Now update the default parameters for the actions you created in the previous stage:

wsk action update conversation --param-file actions/conversationParams.json

wsk action update discovery --param-file actions/discoveryParams.json

1. Next we will create the sequence that will tie the services together.

wsk action create conversation-with-discovery-sequence --sequence conversation,discovery --web true

1. Then create the API that will be called by the client application. The API will consist of a POST operation, named “submit” that will run the sequence created in the previous steps.

wsk api create -n "Conversation-With-Discovery-API" /conversation-with-discovery /submit POST conversation-with-discovery-sequence

1. Copy the URI provided (once the API has been created) and save it. We will need to update the REACT application to use this route in the next step.

The URI should be similar to the following (the string of x’s will be a unique number for your service:

*https://service.us.apiconnect.ibmcloud.com/gws/apigateway/api/xxxxxxxxxx/conversation-with-discovery/submit*

# Web Application

The front-end application consists of static assets (HTML, CSS, and React). The CSS was written with Sass for cleaner, more maintainable source code. Thank you to Christopher Sanchez (cvsanchez42) for creating these.

First we will setup and run the application locally, then we can deploy it to Bluemix.

## Run Locally

1. Link your API to the React App. Do this by creating a file named .env in the root of your working directory (with a text/code editing tool). Copy and paste the following line into your .env file and subsitute the API URL from the [cloud function API section](#API) where indicated. NOTE: The REACT\_APP\_ prefix on the environment variable indicates that React will substitute in the value at build.

REACT\_APP\_API\_URL="<Your API URL>"

1. In the root directory of your repository, install the project dependencies.

npm install

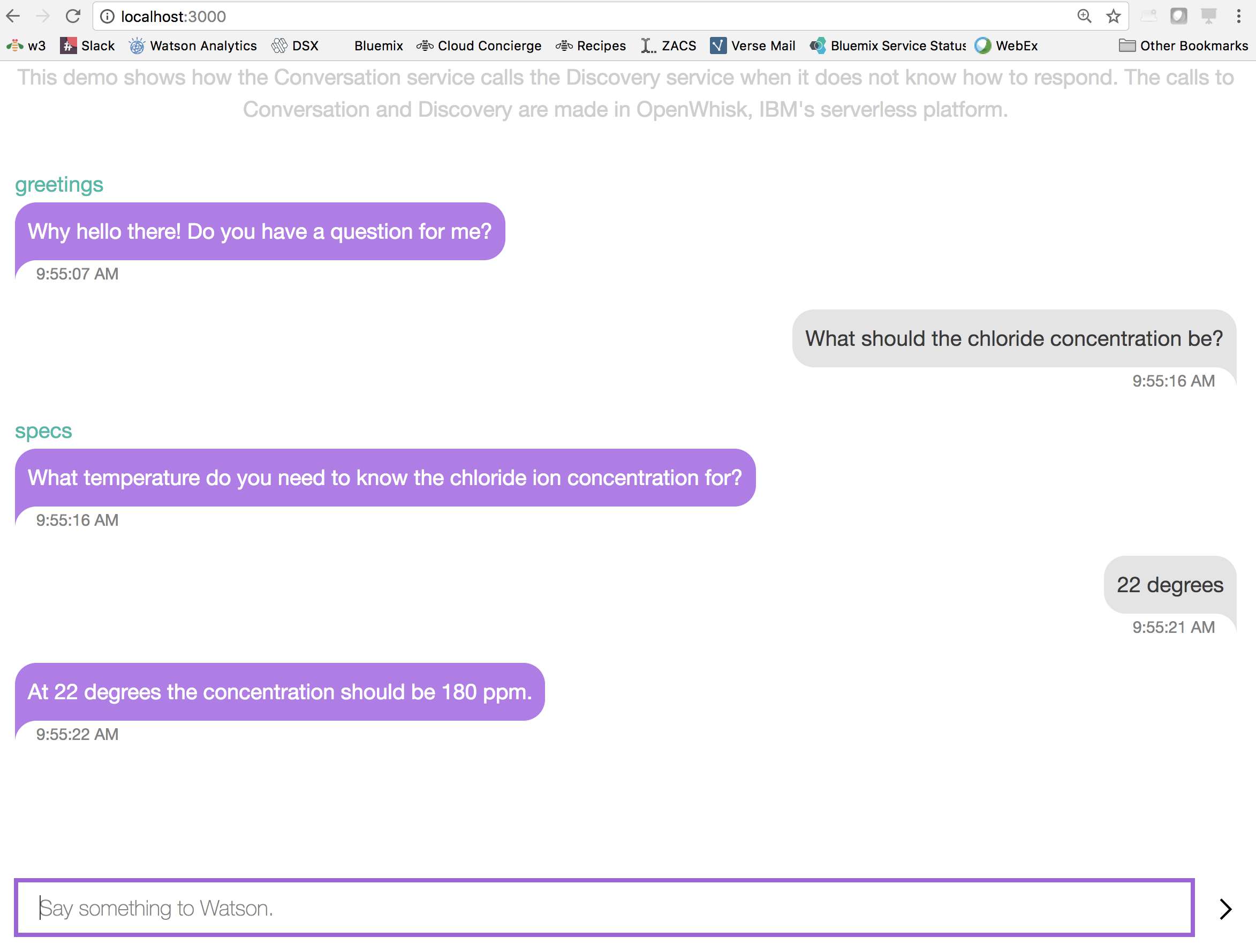
1. Create an optimized build of your project. During this stage, the environment variable (REACT\_APP\_API\_URL) will be inserted into App.js for use by your components.

npm run build

1. Running the App. All that's left is to serve your static files locally by starting the server:

npm start

1. You should see the project running in a new tab in your default browser. Default URL is http://localhost:3000



## Deploy to Bluemix

1. We can now deploy this application to bluemix so it will be publically available. From the terminal, login to cloud foundry (IBM employee SSO instructions):

cf login --sso

Navigate to the URI provided and enter the one-time access code, then select your organization and namespace.

1. The manifest.yml file in the root of your working directory controls the deployment – the defaults will work if you’ve named everything as directed above. However, if your services have a different name (for example), edit this file appropriately.
2. Deploy the application to bluemix by running the following command from the root directory of your application files:

cf push

1. The application URL is available once the application is built, staged and started. Use this link to access your application publically.

