

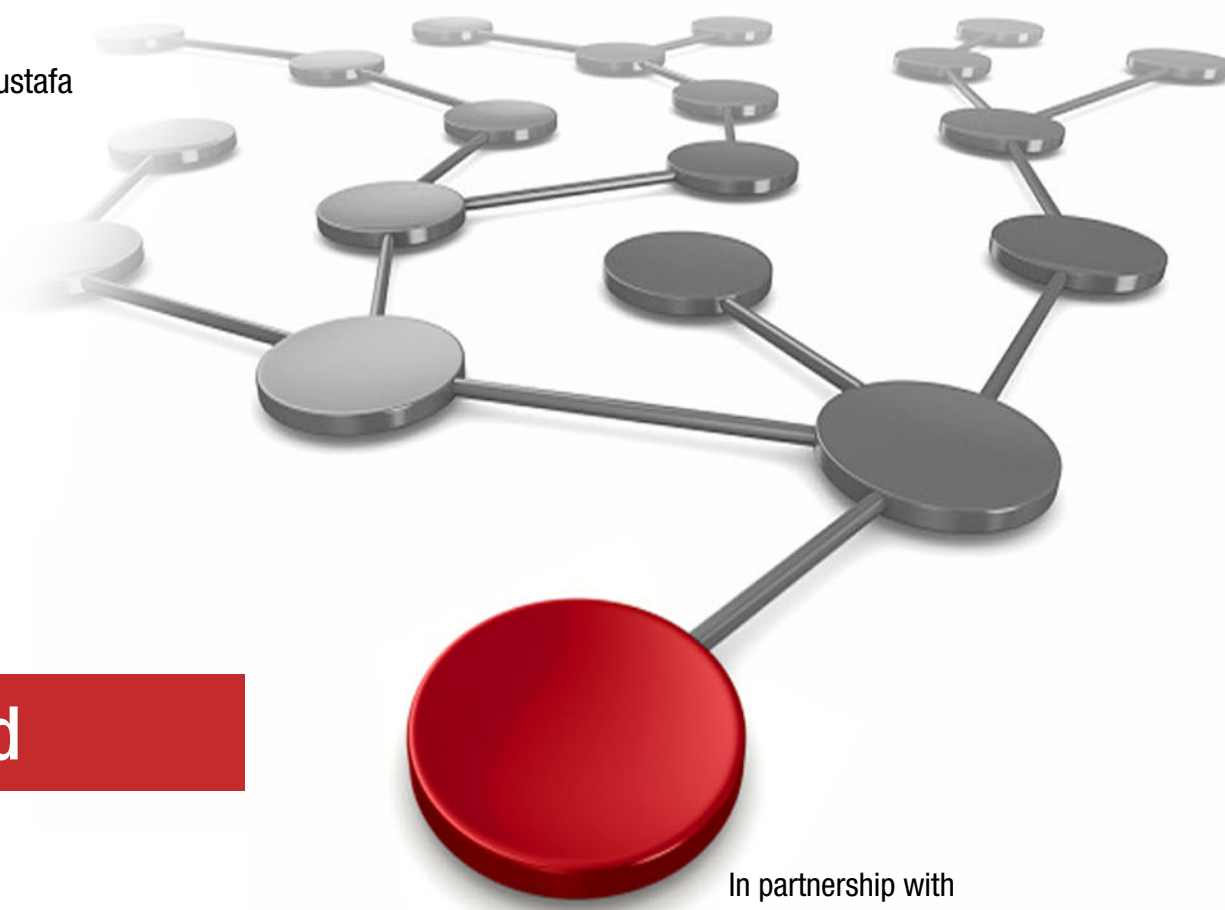
# Building Cognitive Applications with IBM Watson Services: Volume 5 Language Translator

Valerie Lampkin

Thanh Lam

Muhammad Zain Mustafa

Lak Sri



In partnership with  
**IBM Skills Academy Program**





International Technical Support Organization

**Building Cognitive Applications with IBM Watson  
Services: Volume 5 Language Translator**

May 2017

**Note:** Before using this information and the product it supports, read the information in “Notices” on page v.

**First Edition (May 2017)**

This edition applies to IBM Watson services in IBM Bluemix.

**© Copyright International Business Machines Corporation 2017. All rights reserved.**

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

<b>Notices</b> .....	v
Trademarks .....	vi
<b>Preface</b> .....	vii
Authors .....	vii
Now you can become a published author, too! .....	viii
Comments welcome .....	ix
Stay connected to IBM Redbooks .....	ix
<b>Chapter 1. Basics of Watson Language Translator service</b> .....	1
1.1 Watson Language Translator service .....	2
1.1.1 Authentication .....	2
1.1.2 Use case example .....	2
1.1.3 Language Translator flow .....	3
1.1.4 Code snippets: Node.js and Java .....	3
1.2 Language identification .....	6
1.2.1 Authenticate .....	7
1.2.2 Identify language flow .....	7
1.2.3 Node-RED example .....	7
1.3 References .....	9
<b>Chapter 2. Integrating sentiment analysis and language translation in applications</b> .....	11
2.1 Getting started .....	12
2.1.1 Objectives .....	12
2.1.2 Prerequisites .....	12
2.1.3 Expected results .....	12
2.2 Architecture .....	13
2.3 Two ways to deploy the application: Step-by-step and quick deploy .....	14
2.4 Step-by-step implementation .....	14
2.4.1 Creating a Node-RED starter application in Bluemix .....	15
2.4.2 Creating a Language Translator service instance and connecting it to the Node-RED application .....	17
2.4.3 Building the REST language translator flow in Node-RED .....	21
2.4.4 Deploying the Node-RED application .....	28
2.4.5 Testing the REST call to the Language Translator service .....	28
2.5 Quick deployment of application .....	29
2.5.1 Deploying the application to Bluemix .....	29
2.5.2 Creating a Language Translator service instance and connecting it to the application .....	32
2.5.3 Testing the application .....	34
2.6 References .....	36
<b>Chapter 3. Customizing Language Translator linguistic models</b> .....	37
3.1 Introduction .....	38
3.2 Custom dictionary .....	38
3.3 Expanding the model to improve translation quality .....	41
3.4 References .....	44

<b>Appendix A. Additional material</b> .....	45
Locating the web material .....	45
<b>Related publications</b> .....	47
IBM Redbooks .....	47
Online resources .....	47
Help from IBM .....	48

# Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

*IBM Director of Licensing, IBM Corporation, North Castle Drive, MD-NC119, Armonk, NY 10504-1785, US*

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide in any way it believes appropriate without incurring any obligation to you.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.


## COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

# Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com) are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at “Copyright and trademark information” at <http://www.ibm.com/legal/copytrade.shtml>

The following terms are trademarks or registered trademarks of International Business Machines Corporation, and might also be trademarks or registered trademarks in other countries.

Bluemix®	IBM Watson®	Redpapers™
Cloudant®	IBM Watson IoT™	Tivoli®
developerWorks®	Power Systems™	Watson™
Global Business Services®	Redbooks®	Watson IoT™
IBM®	Redbooks (logo)  ®	

The following terms are trademarks of other companies:

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Other company, product, or service names may be trademarks or service marks of others.



# Preface

The *Building Cognitive Applications with IBM Watson Services* series is a seven-volume collection that introduces IBM® Watson™ cognitive computing services. The series includes an overview of specific IBM Watson® services with their associated architectures and simple code examples. Each volume describes how you can use and implement these services in your applications through practical use cases.

The series includes the following volumes:

- ▶ *Volume 1 Getting Started*, SG24-8387
- ▶ *Volume 2 Conversation*, SG24-8394
- ▶ *Volume 3 Visual Recognition*, SG24-8393
- ▶ *Volume 4 Natural Language Classifier*, SG24-8391
- ▶ *Volume 5 Language Translator*, SG24-8392
- ▶ *Volume 6 Speech to Text and Text to Speech*, SG24-8388
- ▶ *Volume 7 Natural Language Understanding*, SG24-8398

Whether you are a beginner or an experienced developer, this collection provides the information you need to start your research on Watson services. If your goal is to become more familiar with Watson in relation to your current environment, or if you are evaluating cognitive computing, this collection can serve as a powerful learning tool.

This IBM Redbooks® publication, Volume 5, describes how the Watson Language Translator service can be used to translate text from one language to another. It introduces the concepts that you need in order to understand how to use the Watson Language Translator service. It provides an example of an application that integrates the Watson Language Translator service. You can develop and deploy the sample application by following along in a step-by-step approach and using provided code snippets. Alternatively, you can download an existing Git project to more quickly deploy the application. This book also covers a mechanism provided by the Watson Language Translator service to train the service to perform domain-specific translations by customizing the existing linguistic models.

## Authors

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

**Valerie Lampkin** is an Advisory Technical Specialist for IBM Bluemix® Cloud offering, specializing in middleware and IBM Watson IoT technologies. Valerie has previously co-authored four Redbooks publications on the topics of Messaging, Internet of Things, MQTT, and Microservices. She is a regular contributor to blogs for IBM developerWorks® and Bluemix Cloud.

**Thanh Lam** is an Instructor and Course Developer with IBM Technical Training and Lab Services. Thanh creates courses for Cloud Computing and IBM Power Systems™. These courses provide technical training for clients around the world through IBM Global Training Partners. Thanh also designs and creates hands-on labs for these courses to help clients work with IBM products in the Cloud Computing areas. Thanh teaches these courses in virtual classrooms and at conferences. He is the co-author of several Redbooks publications. Thanh holds a degree in Doctor of Professional Studies in Computing from Pace University, New York.

**Muhammad Zain Mustafa** is an Advanced Analytics Consultant in IBM Global Business Services®, IBM Pakistan. He has performed a variety of technical jobs in IBM. His vast experience helped Zain to build a solid understanding of the principles necessary to extract actionable insights from structured and unstructured data. Zain has in-depth knowledge of advanced analytics and data engineering systems and he helps clients to design and build solutions to complex problems. He holds a Masters degree in Computer Science with a focus in machine learning and data manipulation.

**Lak Sri** currently serves as a Program Director in IBM developerWorks part of the IBM Digital Business Group organization. Lak leads innovation in the developer activation space. He was the Technical Leader for the *Building Cognitive Applications with IBM Watson Services* Redbooks series. Lak led the development of the IBM Cloud Application Developer Certification program and the associated course. Earlier he worked as a Solution Architect for Enterprise Solutions in Fortune 500 companies using IBM Tivoli® products. He also built strategic partnerships in education and IBM Watson IoT™. Lak is an advocate and a mentor in several technology areas, and he volunteers to plan and support local community programs.

The project that produced this publication was managed by **Marcela Adan**, IBM Redbooks Project Leader, ITSO.

Thanks to the following people for their contributions to this project:

Iain McIntosh

**IBM Watson and Cloud Platform**

Juan Pablo Napoli

**Skills Academy Worldwide Leader, Global University Programs**

Donald McNeil

**IBM Watson and Cloud Platform**

Thanks to the following people for contributing their expertise to this book and for their article, [Add language translation to your apps with IBM Watson](#), available on IBM developerWorks:

Fabian Dubacher

Romeo Kienzler

**IBM Switzerland**

## Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

[ibm.com/redbooks/residencies.html](http://ibm.com/redbooks/residencies.html)

## Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- ▶ Use the online **Contact us** review Redbooks form found at:

[ibm.com/redbooks](http://ibm.com/redbooks)

- ▶ Send your comments in an email to:

[redbooks@us.ibm.com](mailto:redbooks@us.ibm.com)

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization  
Dept. HYTD Mail Station P099  
2455 South Road  
Poughkeepsie, NY 12601-5400

## Stay connected to IBM Redbooks

- ▶ Find us on Facebook:

<http://www.facebook.com/IBMRedbooks>

- ▶ Follow us on Twitter:

<http://twitter.com/ibmredbooks>

- ▶ Look for us on LinkedIn:

<http://www.linkedin.com/groups?home=&gid=2130806>

- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>

- ▶ Stay current on recent Redbooks publications with RSS Feeds:

<http://www.redbooks.ibm.com/rss.html>





# Basics of Watson Language Translator service

The Watson Language Translator service translates text from one language to another.

This chapter introduces the Watson Language Translator service. It provides basic information about the service, how to use it, and includes simple code examples.

This chapter also introduces the language identification API, which you can use to identify the language in the input text.

Code examples (snippets) are provided in these programs:

- ▶ Java
- ▶ Node.js
- ▶ Node-RED

**Note:** A runtime environment that is specific to each technology is required in order to run the code snippets provided in this chapter. The purpose of the snippets in this chapter is to serve as code examples for future reference. Documentation about the installation and configuration of the technology-specific runtime environment is not included.

The following topics are covered in this chapter:

- ▶ Watson Language Translator service
- ▶ Language identification
- ▶ References

## 1.1 Watson Language Translator service

The Watson Language Translator service translates text from one language to another. The service offers real-time, multiple domain-specific models that you can customize based on your unique terminology and language. For more information about customization, see Chapter 3, “Customizing Language Translator linguistic models” on page 37.

The input to and output from this service are as follows:

- ▶ Input:
  - Text to translate. The input text to identify language against, in UTF-8 format. Multiple text query parameters indicate multiple input paragraphs, and a single string is valid input.
  - Source and target languages.
- ▶ Output:
  - JSON or plain output of text translated to the target language selected.

Translation is available in Arabic, Chinese, English, French, Korean, German, Portuguese, and Spanish. Some languages might not be available for all domains. For information about models and domains see Chapter 3, “Customizing Language Translator linguistic models” on page 37.

For a complete list of parameters and responses, see the [Watson Language Translator API Reference](#) documentation.

### 1.1.1 Authentication

Authenticate to the Language Translator APIs by specifying the user name and password that are provided in the service credentials for the service instance that you want to use. The API uses Basic Authentication.

After creating an instance of the Language Translator service in IBM Bluemix, select **Service Credentials** to display the user name and password that are associated with the instance.

### 1.1.2 Use case example

This service can be used in practical applications such as these situations:

- ▶ Take news from across the globe and present it in the reader’s language.
- ▶ A help desk assistant chatbot that communicates with customers in their own language.
- ▶ Create speaker’s captions in a video.

### 1.1.3 Language Translator flow

Figure 1-1 shows the basic flow.

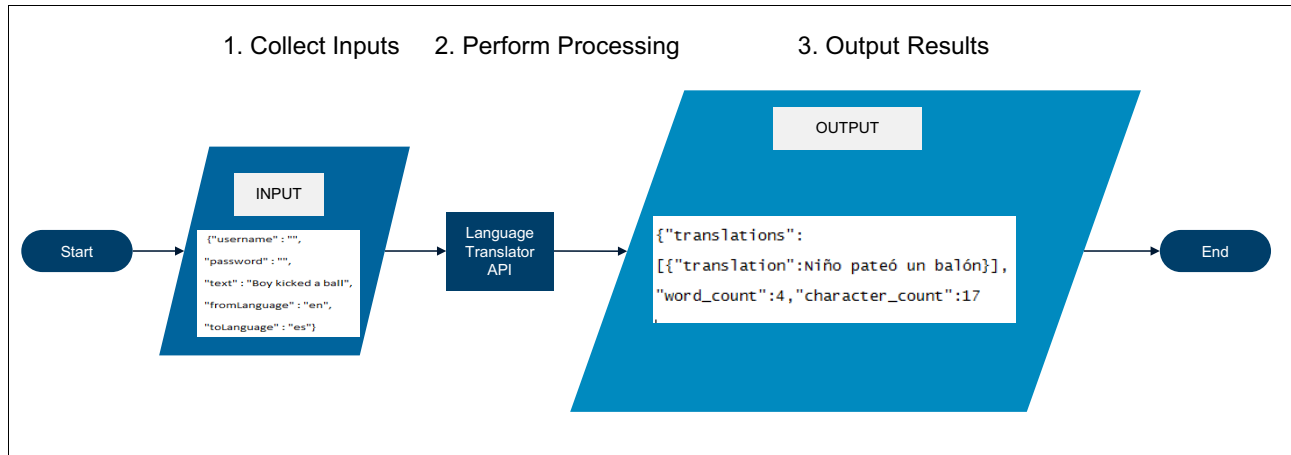


Figure 1-1 Language Translator flow

The credentials to access the Language Translator service instance and the text input to be translated are shown in Example 1-1.

#### Example 1-1 Input

```
{\"username\" : \"\",  
  \"password\" : \"\",  
  \"text\" : \"Boy kicked a ball\",  
  \"fromLanguage\" : \"en\",  
  \"toLanguage\" : \"es\"}
```

The processed output is shown in Example 1-2.

#### Example 1-2 Output

```
{\"translations\":  
  [\"translation\": \"Niño pateó un balón\"],  
  \"word_count\":4,\"character_count\":17}
```

### 1.1.4 Code snippets: Node.js and Java

Example 1-3 shows a basic code snippet to demonstrate how to use the Language Translator service in a Node.js program.

#### Example 1-3 Code snippet: Node.js

```
var parameters = {  
  \"username\" : \"\",  
  \"password\" : \"\",  
  \"text\" : \"Hello my friend\",  
  \"fromLanguage\" : \"en\",  
  \"toLanguage\" : \"es\",  
  \"url\" : \"https://gateway.watsonplatform.net/language-translator/api/\"  
}
```

```

function handler(req_parameters, callback) {
  var LanguageTranslatorV2 =
require('watson-developer-cloud/language-translator/v2');

  var language_translator = new LanguageTranslatorV2({
    username: req_parameters.username, // SET YOUR USERNAME
    password: req_parameters.password, // SET YOUR PASSWORD
    url: req_parameters.url
  });

  language_translator.translate({
    text: req_parameters.text,
    source: req_parameters.fromLanguage,
    target: req_parameters.toLanguage
  }, function(err, response) {
    if (err) {
      console.log('error:', err);
      if (typeof callback !== 'undefined' && typeof callback=="function") return
callback(err);
    } else {
      console.log(JSON.stringify(response, null, 2));
      if (typeof callback !== 'undefined' && typeof callback=="function") return
callback(response);
    }
  });
}

//Allows Execution of this handler
//will run if only called directly
if (require.main === module) {
  handler(parameters,null);
} else {

// name of the unit for logging and servlet path also
var unitpath = "";

// Template for making above code available
// as service via superglue routine
var superglue = require('sandbox-superglue');
module.exports = {
  path: '/' +unitpath,
  priority: 1,

  init: function (app) {
    // something to do initially
  },
  GET: function(req, res) {superglue.GET(req,res,parameters,unitpath)},
  POST: function(req, res) {superglue.POST(req,res,handler)}
}
}

```

---



Example 1-4 shows the code snippet in Java.

*Example 1-4 Code snippet: Java*

---

```
// LanguageTranslationTest
//Select a domain, then identify or select the language of text, and then
//translate the text from one supported language to another.
//Example: Translate 'hello my friend' from English to Spanish using the Language
//Translation service.
import javax.servlet.annotation.WebServlet;

import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import com.ibm.watson.developer_cloud.language_translator.v2.LanguageTranslator;
import com.ibm.watson.developer_cloud.language_translator.v2.model.Language;
import
com.ibm.watson.developer_cloud.language_translator.v2.model.TranslationResult;

//After deployment go to the relative URI to test the functionality.
//You would see a form to provide the input values.
@WebServlet("/")
public class Snippet extends SuperGluev2 {

    public String parameters =
"{\"username\":\"\", \"password\":\"\", \"text\":\"hello my
friend\", \"fromLanguage\":\"ENGLISH\", \"toLanguage\":\"SPANISH\"}";

    @Override
    protected JsonObject process(String jsonString) {
        JsonParser parser = new JsonParser();
        JsonObject myBean = parser.parse(jsonString).getAsJsonObject();

// LanguageTranslation service = new LanguageTranslation();
LanguageTranslator service = new LanguageTranslator();

        service.setUsernameAndPassword(myBean.get("username").getAsString(),
myBean.get("password").getAsString());

        //fromlanguage and tolanguage arg need to be upper case
        TranslationResult translationResult =
service.translate(myBean.get("text").getAsString(),

Language.valueOf(myBean.get("fromLanguage").getAsString().toUpperCase()),
Language.valueOf(myBean.get("toLanguage").getAsString().toUpperCase())).execute();

        JsonObject json =
parser.parse(translationResult.toString()).getAsJsonObject();

        return json;
    }

    public static void main(String[] args) {
        Snippet myclass = new Snippet();
    }
}
```

```

//***** Process method contains the key logic *****
JsonObject processResult = myclass.process(myclass.parameters);

Gson gson = new GsonBuilder().setPrettyPrinting().create();
System.out.println(gson.toJson(processResult));
}

@Override
JsonObject getParameters() {
    return new JsonParser().parse(parameters).getAsJsonObject();
}

private static final long serialVersionUID = 1L;
}

```

---

## 1.2 Language identification

The Language Translator service can be used to identify one or more languages in text. The input and output for identification by the service are defined as follows:

- ▶ Input: The input text to identify language against, in UTF-8 format.
- ▶ Output: The identified language or array of identified languages in JSON or plain text format with the associated confidence score.

Example 1-5 shows a response

*Example 1-5 Identify language response*

---

```

{
  "languages": [
    {
      "confidence": 0.9143,
      "language": "en-US"
    },
    {
      "confidence": 0.0396,
      "language": "hu-HU"
    },
    {
      "confidence": 0.0093,
      "language": "ro-RO"
    },
    {
      "confidence": 0.005,
      "language": "nl-NL"
    },
    //...
  ]
}

```

---

For more information see [Identify language](#) in the Language Translator API Reference.

## 1.2.1 Authenticate

Authenticate to the Language Translator API by providing the user name and password that are provided in the service credentials for the service instance that you want to use. The API uses Basic Authentication.

After creating an instance of the Language Translator service in Bluemix, select **Service Credentials** to display the user name and password that are associated with that instance.

## 1.2.2 Identify language flow

Figure 1-2 shows the flow of the *identify language* API.

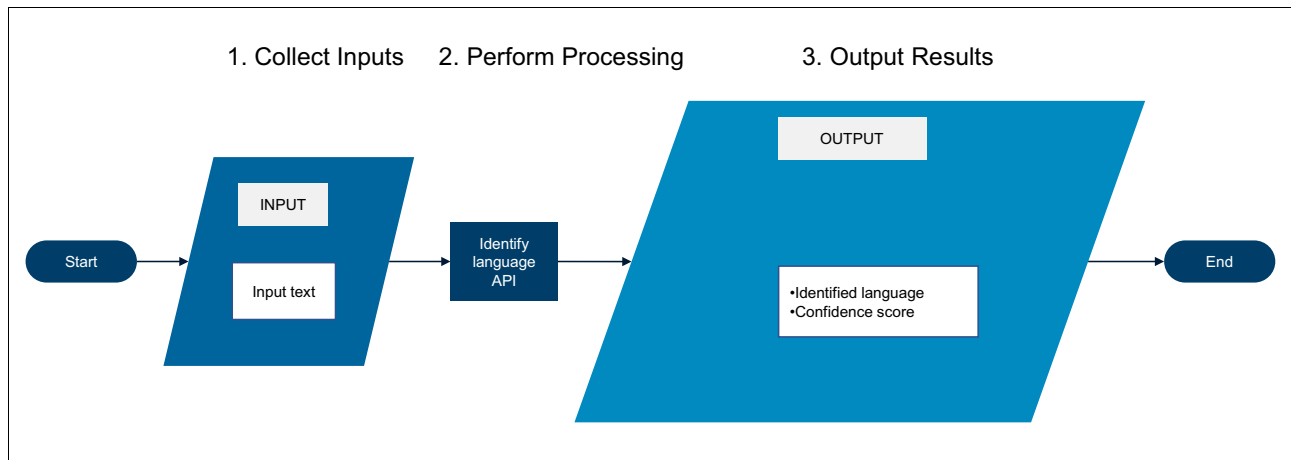


Figure 1-2 Identify language flow

## 1.2.3 Node-RED example

The Node-RED language *identify* node is used to identify one or more languages in a text. The output shows the detected languages with the level of confidence. If you bind your Language Translator service instance to a Node-RED application that is running in Bluemix, the user name or password does not have to be configured when using the language *identify* node.

### The inject node configuration (input text)

The *inject* node allows you to inject messages into a flow. This example uses three *inject* nodes to input the text in three languages.

Figure 1-3 on page 8 shows how to use the Edit *inject* node dialog to configure the *inject* node to input a text string in English.

**Edit inject node**

Cancel Done

✉ Payload

☰ Topic

🔄 Repeat

Inject once at start?

📌 Name

**Note:** "interval between times" and "at a specific time" will use cron. See info box for details.

Figure 1-3 Edit inject node

### Output from the language identify node

The language identify node output is as follows:

- msg.lang** The identified language with the highest confidence level.
- msg.languages** An array of identified languages with the five-letter ISO code and the associated confidence score.

Figure 1-4 shows examples of msg.lang and msg.languages values.

```

10/14/2016, 4:00:10 PM d0299ees.eb3ad
msg.lang : Object
{ "language": "en", "confidence": 0.974798 }

10/14/2016, 4:00:10 PM 28b9e8ff.189ea
msg.languages : array [62]
[ { "language": "en", "confidence": 0.974798 }, { "language": "hu",
"confidence": 0.0108785 }, { "language": "nn", "confidence": 0.00374943 }, {
"language": "it", "confidence": 0.00157705 }, { "language": "nl",
"confidence": 0.00148452 }, { "language": "ku", "confidence": 0.00124234 },
{ "language": "es", "confidence": 0.000624638 }, { "language": "sq",
"confidence": 0.0006135 }, { "language": "is", "confidence": 0.000601705 }, {
"language": "ro", "confidence": 0.000574835 }, { "language": "af",
"confidence": 0.000517337 }, { "language": "pt", "confidence": 0.000438042
}, { "language": "nb", "confidence": 0.000375821 }, { "language": "sk",
"confidence": 0.000300419 }, { "language": "fi", "confidence": 0.000299594 },
{ "language": "cs", "confidence": 0.000278817 }, { "language": "et",
"confidence": 0.000277003 }, { "language": "tr", "confidence": 0.000167125
}, { "language": "f ....

```

Figure 1-4 The msg.lang object and msg.languages array

## Node-RED flow

Figure 1-5 shows the flow to identify languages.

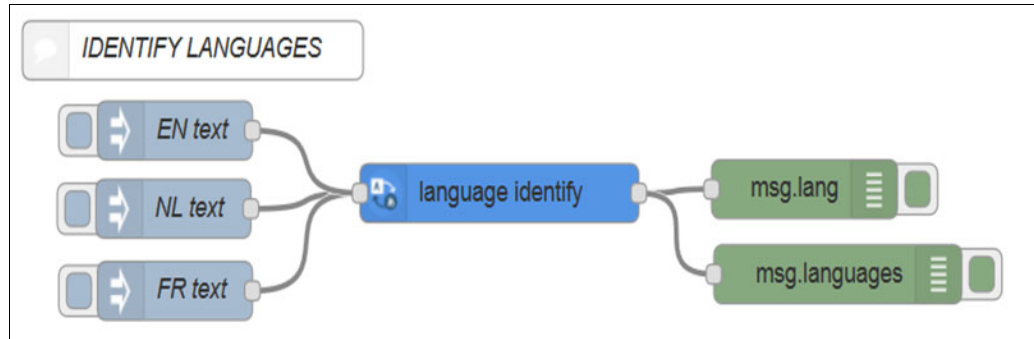


Figure 1-5 Identify languages flow

## Snippet Node-RED flow

You can copy the code for the flow from Example 1-6 and then import it from the clipboard into your Node-RED flow editor.

*Example 1-6 Code snippet: For Node-RED flow*

```
[{"id":"ea5c5d47.c41bb","type":"watson-language-translator-identify","z":"37730b3a.c6f034","name":"","x":315,"y":304,"wires":[["148a95ee.6994ba","65680f08.8ae5e"]]}, {"id":"a421f263.5b32e","type":"inject","z":"37730b3a.c6f034","name":"NL text","topic":"","payload":"hallo dit is een test","payloadType":"str","repeat":"","crontab":"","once":false,"x":104.5,"y":312,"wires":[["ea5c5d47.c41bb"]]}, {"id":"cc0339bd.5e4418","type":"comment","z":"37730b3a.c6f034","name":"IDENTIFY LANGUAGES","info":"","x":117,"y":231,"wires":[]}, {"id":"148a95ee.6994ba","type":"debug","z":"37730b3a.c6f034","name":"","active":true,"console":"false","complete":"lang","x":512,"y":302,"wires":[]}, {"id":"7db04ada.d35c34","type":"inject","z":"37730b3a.c6f034","name":"FR text","topic":"","payload":"Bonjour ceci est un test","payloadType":"str","repeat":"","crontab":"","once":false,"x":105.5,"y":354,"wires":[["ea5c5d47.c41bb"]]}, {"id":"63133052.1b744","type":"inject","z":"37730b3a.c6f034","name":"EN text","topic":"","payload":"hello, this is a test !","payloadType":"str","repeat":"","crontab":"","once":false,"x":105.5,"y":272,"wires":[["ea5c5d47.c41bb"]]}, {"id":"65680f08.8ae5e","type":"debug","z":"37730b3a.c6f034","name":"","active":true,"console":"false","complete":"languages","x":533.5,"y":345.5,"wires":[]}]
```

## 1.3 References

For helpful information, see the following resources:

- ▶ Language Translator service documentation:  
<https://www.ibm.com/watson/developercloud/doc/language-translator/index.html>
- ▶ API Explorer: Language Translator:  
<https://watson-api-explorer.mybluemix.net/apis/language-translator-v2>
- ▶ Identify language:  
<https://www.ibm.com/watson/developercloud/language-translator/api/v2/#identify>
- ▶ Language Translator API Reference:  
<https://www.ibm.com/watson/developercloud/language-translator/api/v2/>





# Integrating sentiment analysis and language translation in applications

This chapter describes a sample Node-RED application that integrates the IBM Watson Language Translator service and sentiment analysis to translate text, entered by the user, and to get the sentiment score for it. The simple approach shown in this chapter can be applied to many use cases for business applications, such as a customer service tool that translates the input text from customers and at the same time provides information about how the customer feels during the interaction with the operator or chatbot.

The [IBM Watson Language Translator](#) service in [IBM Bluemix](#) uses the IBM Watson cognitive computing technology to convert user entered text to other languages, allowing developers to add language translation to their applications.

With the [Node-RED](#) flow editor (available from IBM Bluemix Catalog), you can easily use the Watson Language Translator service in your Bluemix apps. This chapter shows how to use Node-RED to create an app that exposes a REST endpoint for a Language Translator service instance. Then, you add a user interface (UI) that was created with [AngularJS](#). The UI includes a visualization of the [AFINN-111 sentiment score](#) for the text that the user enters. In just minutes, your application will be translating English to Spanish, and reporting on the user's feelings based on the text the user inputs.

The scenario in this chapter is based on the article [Add language translation to your apps with IBM Watson](#), which is available on IBM developerWorks.

The following topics are covered in this chapter:

- ▶ Getting started
- ▶ Architecture
- ▶ Two ways to deploy the application: Step-by-step and quick deploy
- ▶ Step-by-step implementation
- ▶ Quick deployment of application
- ▶ References

## 2.1 Getting started

To start, read through the objectives, prerequisites, and expected results of this use case.

### 2.1.1 Objectives

By the end of this chapter, you should be able to accomplish these objectives:

- ▶ Recognize use cases where you can integrate the Watson Language Translator service and sentiment analysis with your application.
- ▶ Configure a Node-RED flow to implement a REST call for the Watson Language Translator service.
- ▶ Understand how a Bluemix web application can interface with the Language Translator service to translate input text.

### 2.1.2 Prerequisites

To build the sample application in this chapter, you must have the following accounts and resources:

- ▶ Any Internet browser, such as Chrome, Firefox, Internet Explorer, and Safari
- ▶ An [IBM Bluemix account](#). Bluemix is an open-standard cloud platform for building, running, and managing apps and services.

### 2.1.3 Expected results

The web application presents a web page to the user. The user enters the text to be translated, selects the source and target language, and clicks **Translate**.

The input text is analyzed, a sentiment score is returned, and the text is translated to the target language. The translated text and sentiment score are displayed to the user.

Figure 2-1 on page 13 shows the results of running this sample application.





Figure 2-1 Watson Language Translator: English to Spanish, and sentiment score

## 2.2 Architecture

Figure 2-2 shows the architectural flow for this application.

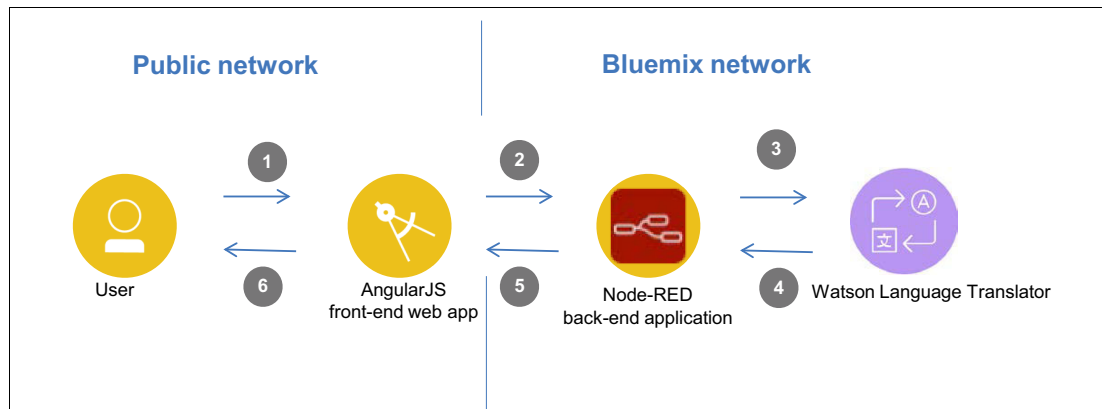


Figure 2-2 Architectural flow diagram

The steps identified in Figure 2-2 on page 13 are as follows:

1. The user inputs text to the web front-end AngularJS web application. The user also indicates the source and target languages.
2. The AngularJS web app uses a REST call to pass the user's input message to the Node-RED back-end application.
3. The user's message is analyzed by the sentiment node in the Node-RED application, running in Bluemix, and it returns a sentiment score. Then, the Node-RED application sends the input text to the Watson Language Translator service.
4. The Watson Language Translator service translates the input text to the target language and returns the translated text to the Node-RED application.
5. The Node-RED application sends the translated message and the sentiment score to the web app.
6. The web app presents the translated text and sentiment score to the user.

## 2.3 Two ways to deploy the application: Step-by-step and quick deploy

The two ways to experience this use case are as follows:

- ▶ Step-by-step implementation of the application

This approach takes you through the key steps to integrate the IBM Watson Language Translator service with the application logic. All sections of 2.4, "Step-by-step implementation" on page 14 take you through all steps to create and deploy the sample application. This section does not include deploying the front-end web application.

- ▶ Quick deployment of the application

A Git repository that contains the final version of the application is provided with this book. If you want to bypass the implementation steps and instead run the application as a demonstration, download this full version. If you prefer this option, skip to 2.5, "Quick deployment of application" on page 29.

## 2.4 Step-by-step implementation

This section guides you through the steps to get started using the Watson Language Translator service in Bluemix. It contains instructions to complete a Node-RED application that makes a REST call to the Language Translator service.

Implementing this use case involves the following steps:

- ▶ Creating a Node-RED starter application in Bluemix.
- ▶ Creating a Language Translator service instance and connecting it to the Node-RED application.
- ▶ Building the REST language translator flow in Node-RED.
- ▶ Deploying the Node-RED application.
- ▶ Testing the REST call to the Language Translator service.

## 2.4.1 Creating a Node-RED starter application in Bluemix

The Language Translator flow in this chapter is created by using the Node-RED capabilities in Bluemix. To develop the flow, first create a Node-RED application and then add the Language Translator service, sentiment analysis, and functions that are needed in order to manipulate the input and output parameters.

Complete these steps:

1. Log in to Bluemix.
2. In the catalog, scroll to the Apps section, and then select **Boilerplates** → **Node-RED Starter** (Figure 2-3).

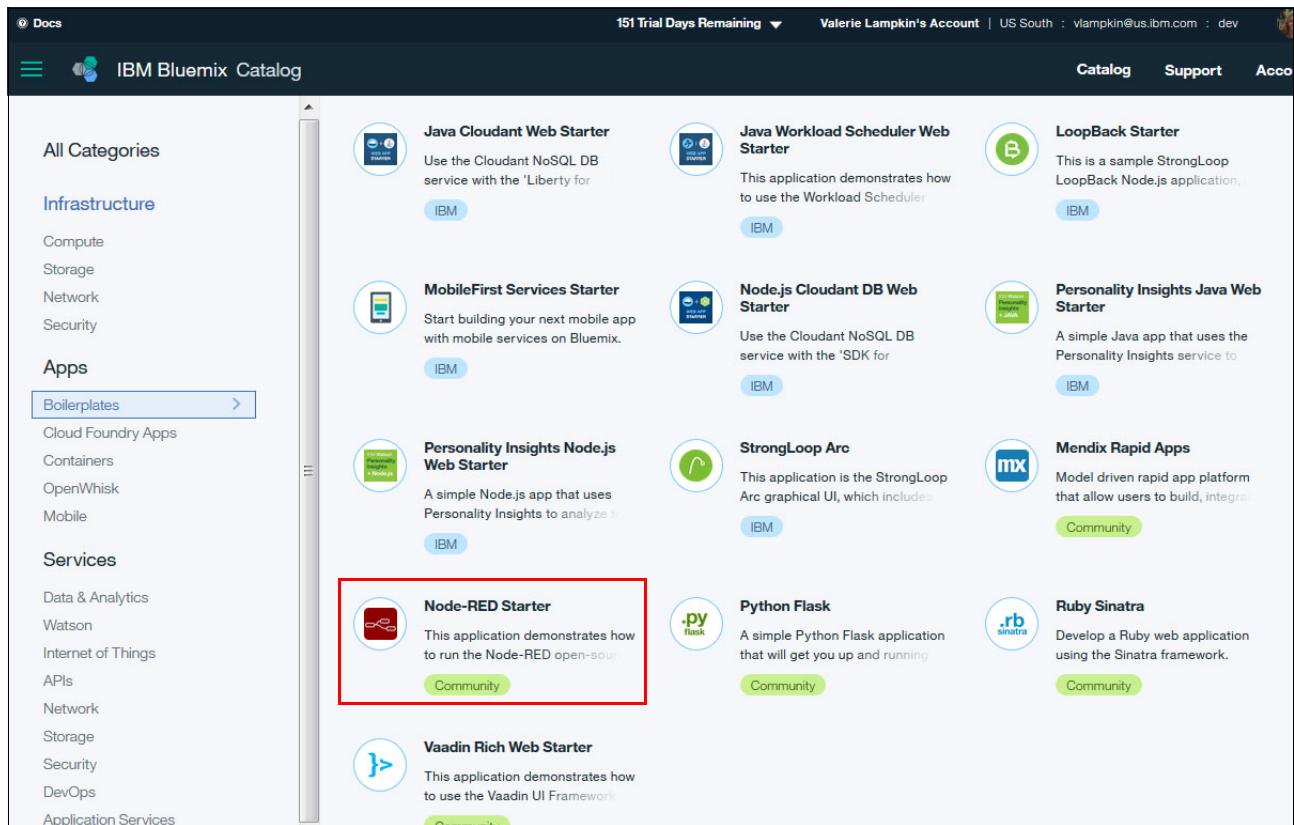


Figure 2-3 Node-RED Starter in the Bluemix catalog

3. Enter a unique name in the App name and Host name fields (Figure 2-4). In the Selected Plan section, keep SDK for Node.js as **Default** and IBM Cloudant® NoSQL DB as **Lite**. Click **Create**.

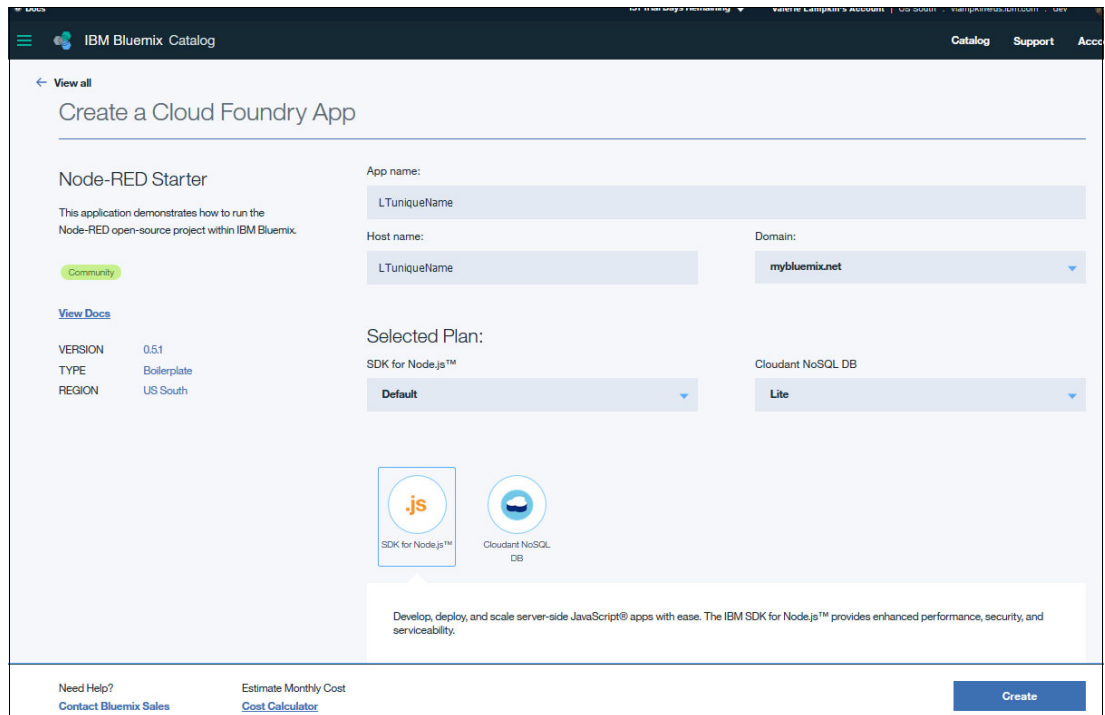


Figure 2-4 Create Node-RED starter app

4. Completion of the application deployment can take several minutes. You can switch from Getting started to the **Logs** selection to monitor the progress (Figure 2-5).

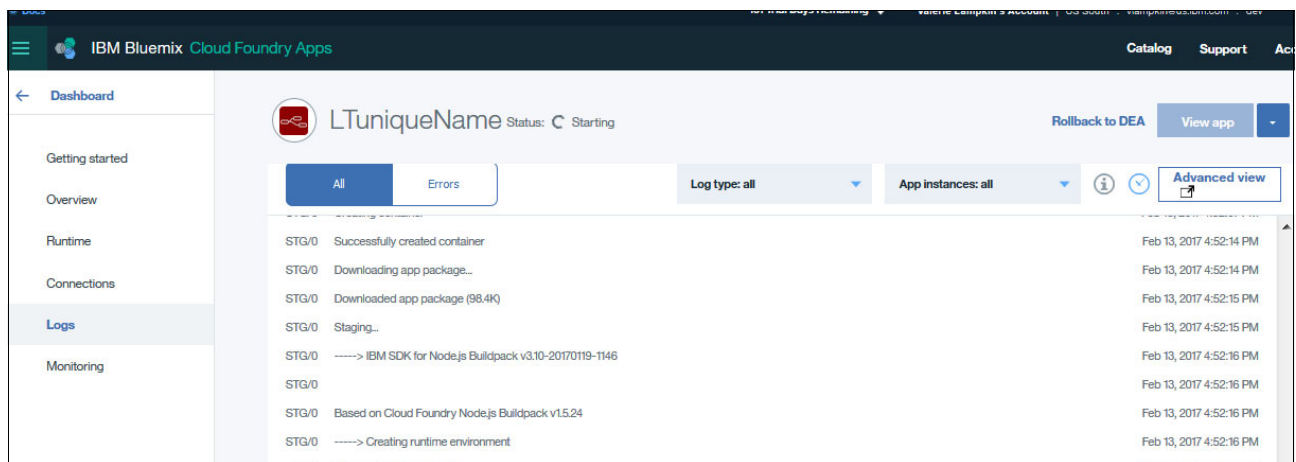


Figure 2-5 Logs view

5. You can scroll through the log to see the most recent messages. When the logs indicate that the flows have started, return to the Bluemix dashboard to see the Node-RED application that you just created.

Figure 2-6 shows the application running in the Bluemix dashboard.

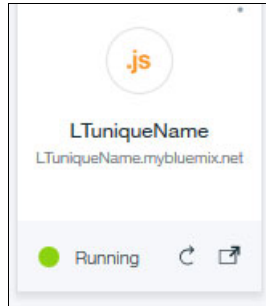


Figure 2-6 Running application

## 2.4.2 Creating a Language Translator service instance and connecting it to the Node-RED application

To create a Watson Language Translator service instance and connect it to your Node-RED application, complete these steps:

1. From the dashboard, click the application to view the application details. When the information is displayed, click **Connections** on the left menu (Figure 2-7), which allows you to connect existing or new services to your application. Click **Connect new**.

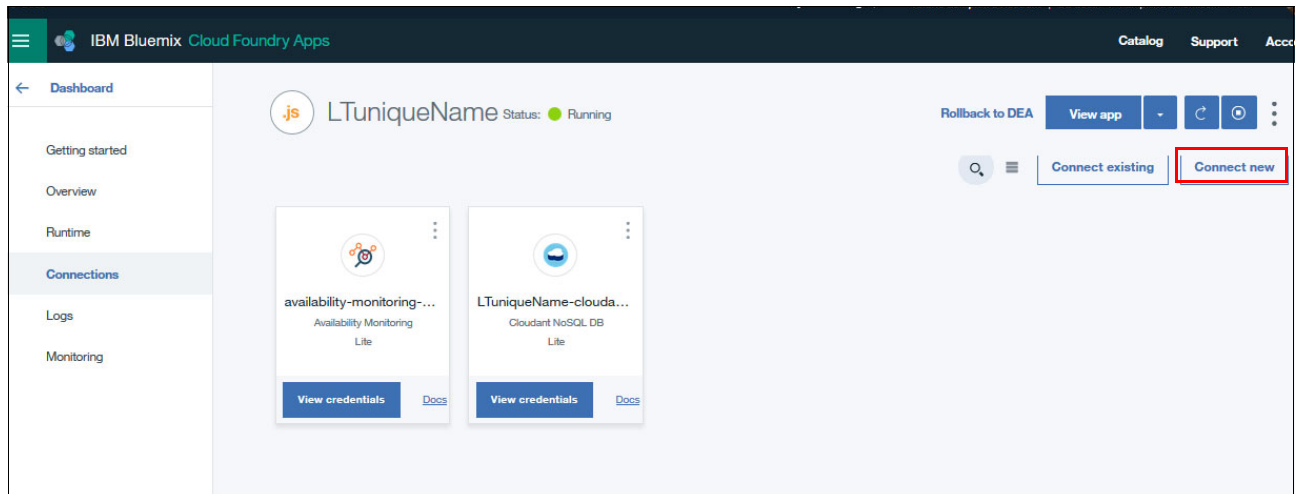


Figure 2-7 Connecting a new service to your application

2. Under Services, click **Watson** and select **Language Translator** (Figure 2-8).

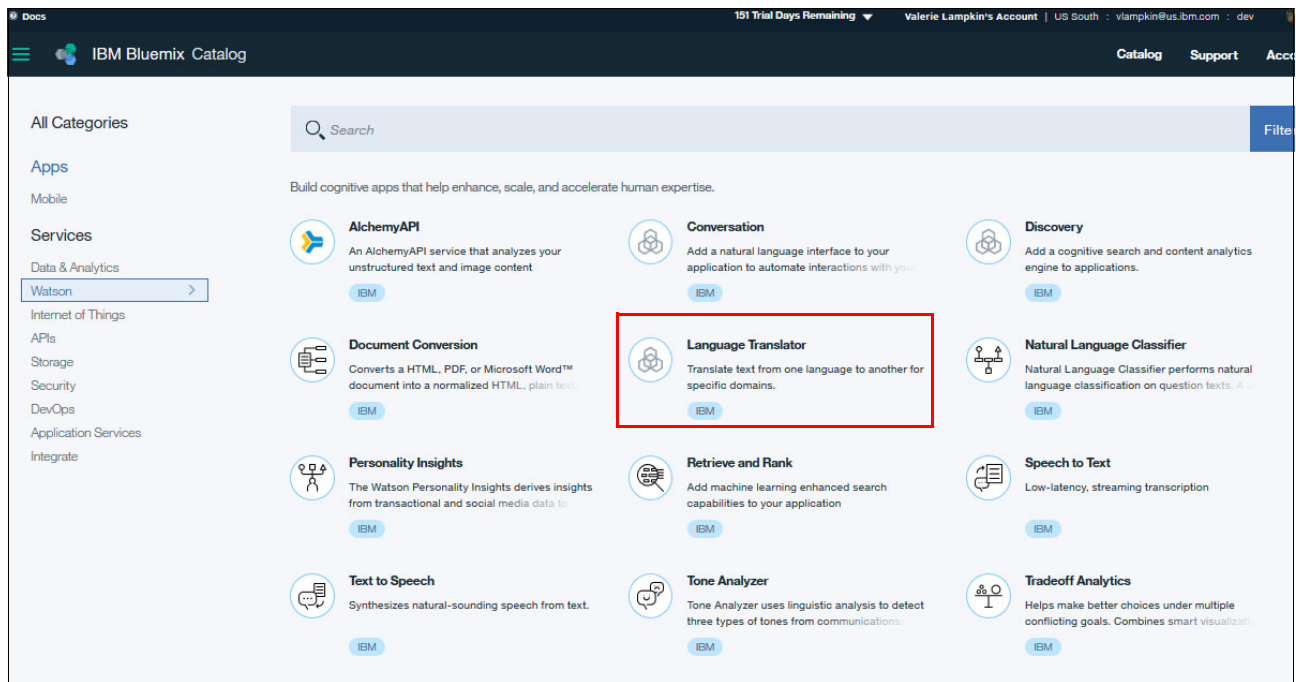


Figure 2-8 Select Language Translator

3. On the Language Translator page (Figure 2-9), either use the default name or edit it to give it a customized name. For this example, name the service LanguageTranslatorxxx where xxx are your initials. Then, click **Create**.

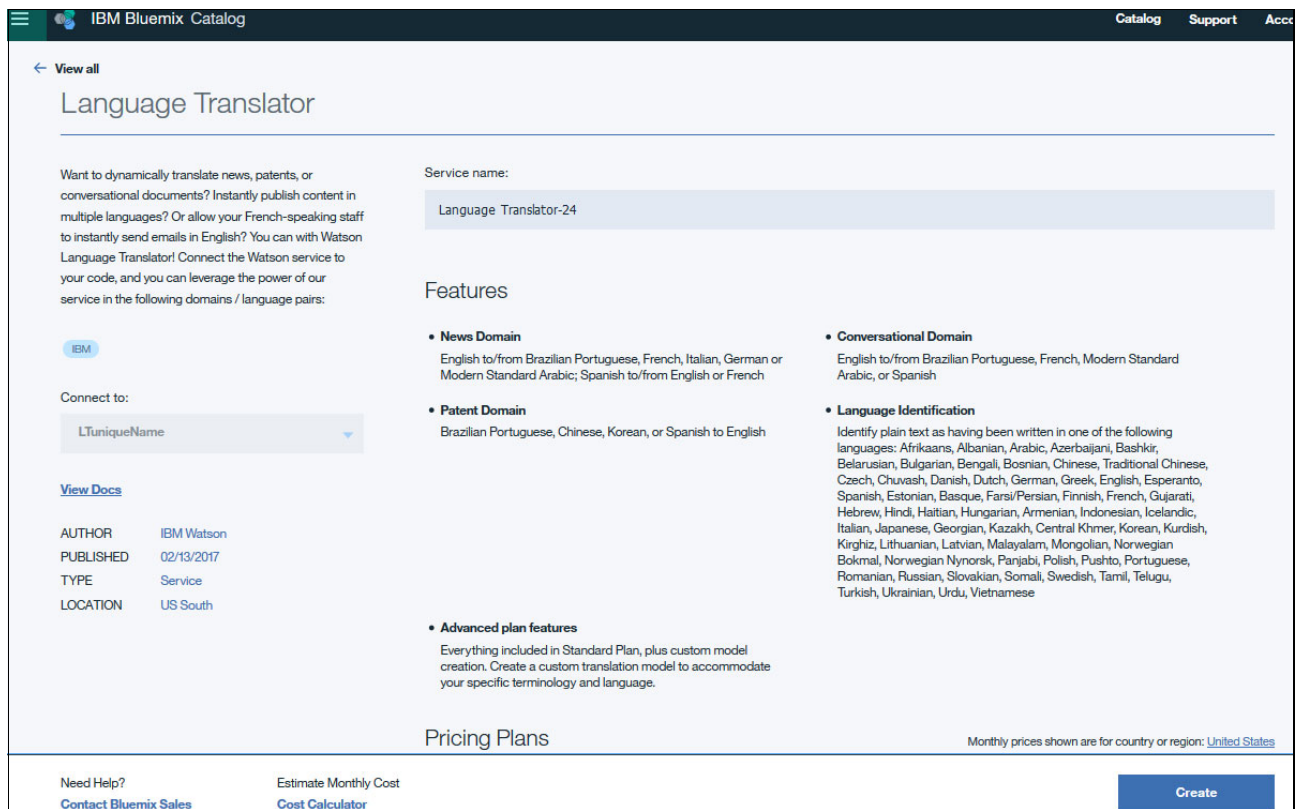


Figure 2-9 Create Language Translator service instance

4. When prompted, click **Restage**.

The application must be restaged after connecting to the newly created Language Translator service instance so it can use it (Figure 2-10).

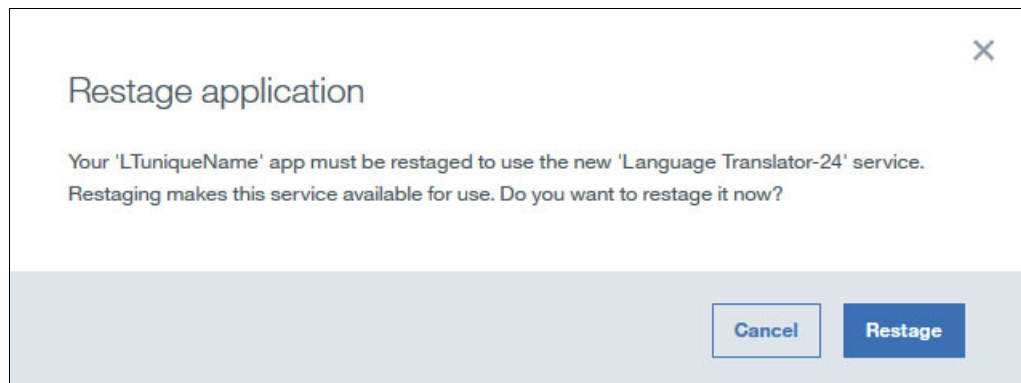


Figure 2-10 Restage the Node-RED application

As your application restages, click **Logs** to view the progress (Figure 2-11).

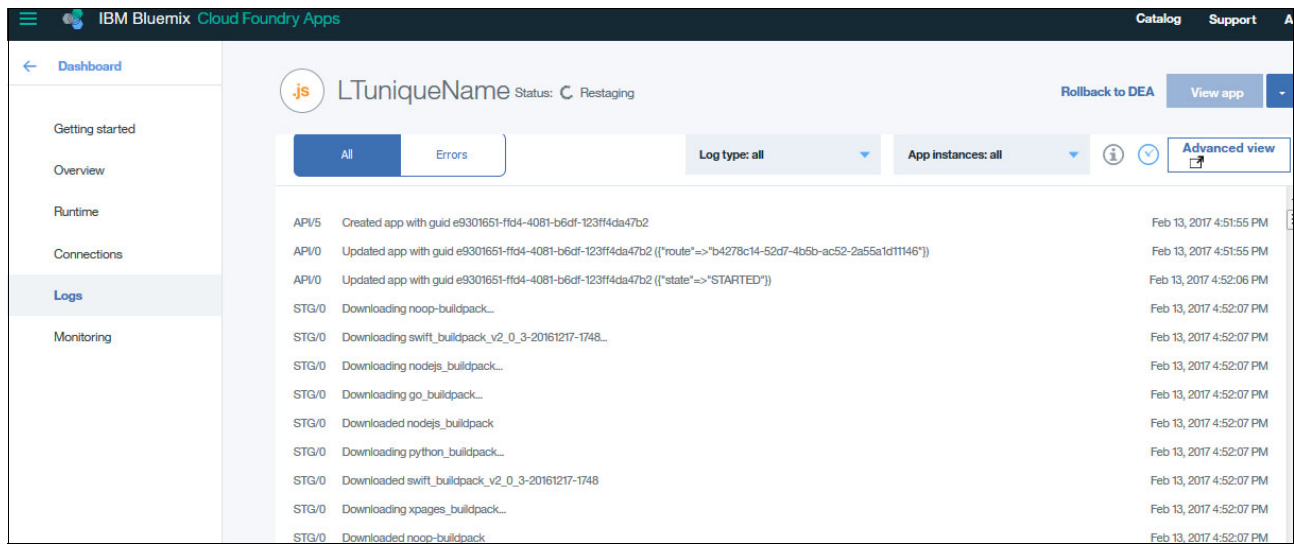


Figure 2-11 Restaging application: Logs view

5. After the restage is completed, the Logs view shows that the flows are started and the application status is Running. Click the application route, which is the link to access your application, as shown in Figure 2-12.

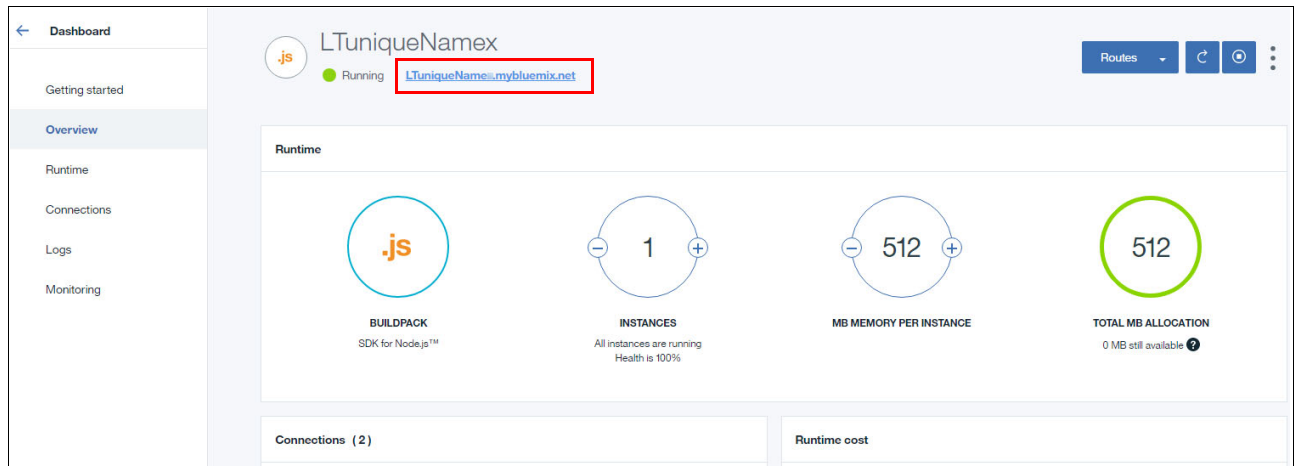


Figure 2-12 Running application: Click app route



6. Your Node-RED application opens in a new browser tab (Figure 2-13). Click **Go to your Node-RED flow editor**.

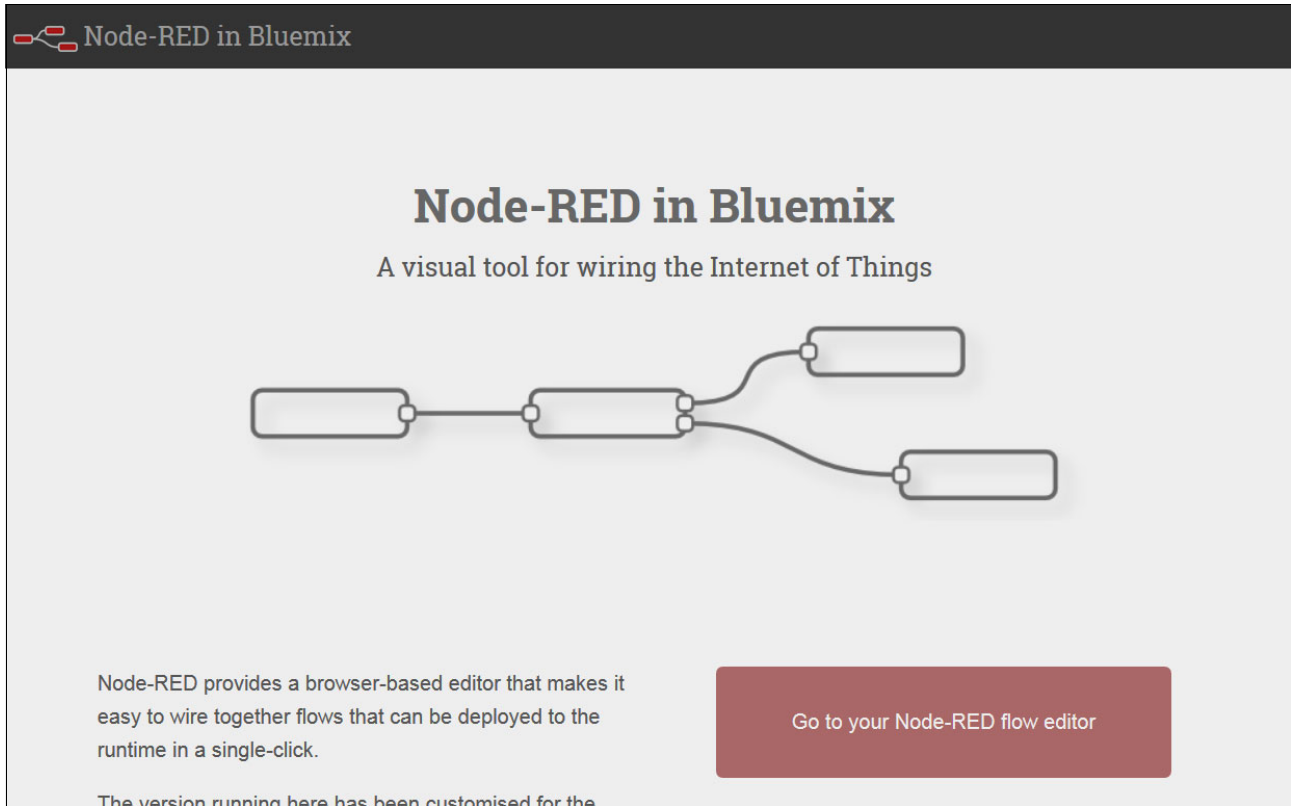


Figure 2-13 Node-RED in Bluemix

### 2.4.3 Building the REST language translator flow in Node-RED

You use the Node-RED flow editor to build a REST service. In this example, the flow takes an input message, analyzes it to get the sentiment score, and passes the text message to the Language Translator service to translate the message, from English to Spanish in this example.

Follow these steps to create and configure the flow:

1. From the node palette, drag an **http** input node onto the flow canvas (Figure 2-14 on page 22). On the palette, expand the **input** section to see the **http** node.

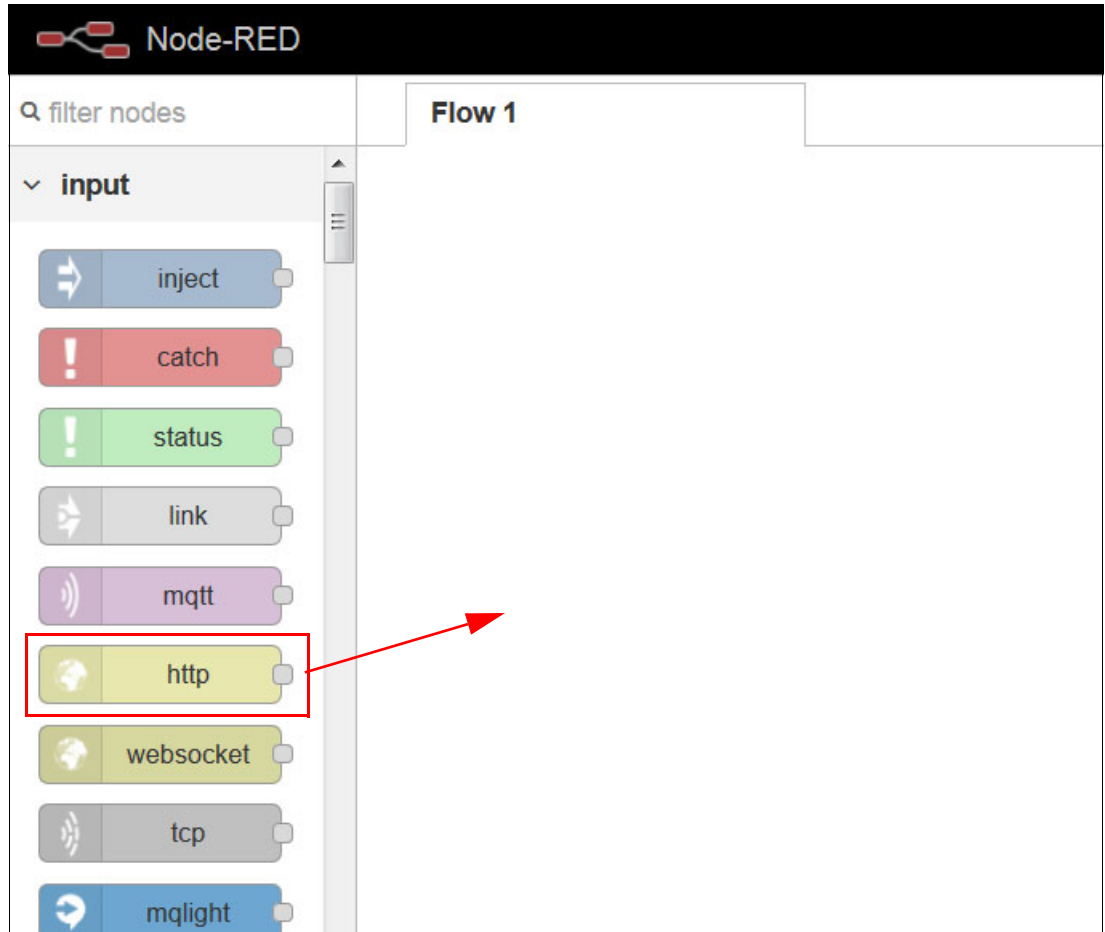


Figure 2-14 Drag the http input node to the canvas

2. After you drag the **http** input node to your canvas, double-click it to open the Edit http in node dialog (Figure 2-15). Configure the node for the GET method and enter /o11i in the URL field. Then, click **Done** to create a REST service listening on /o11i.

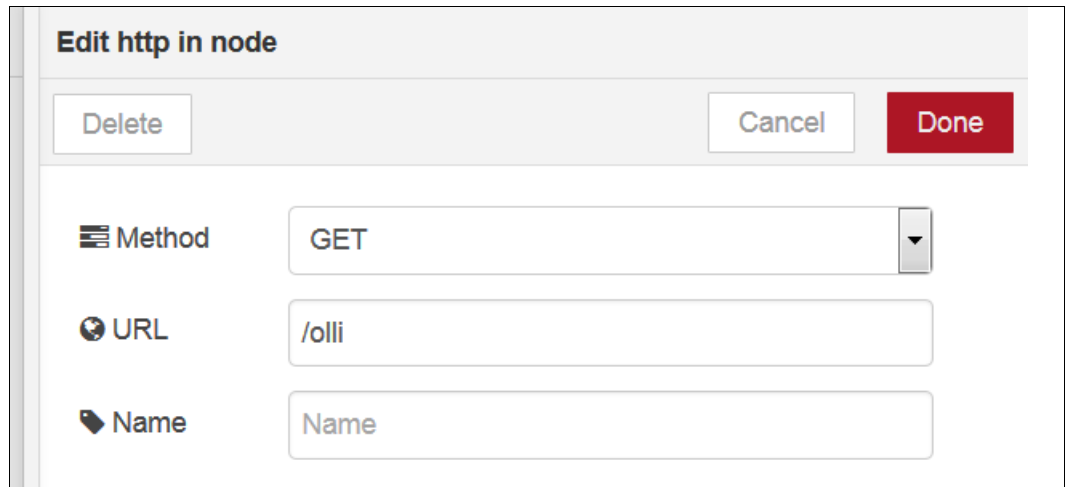


Figure 2-15 Edit http in node dialog (GET method)

3. Drag the **function** node to the canvas (Figure 2-16) beside the http node (GET /o11i) that you configured.

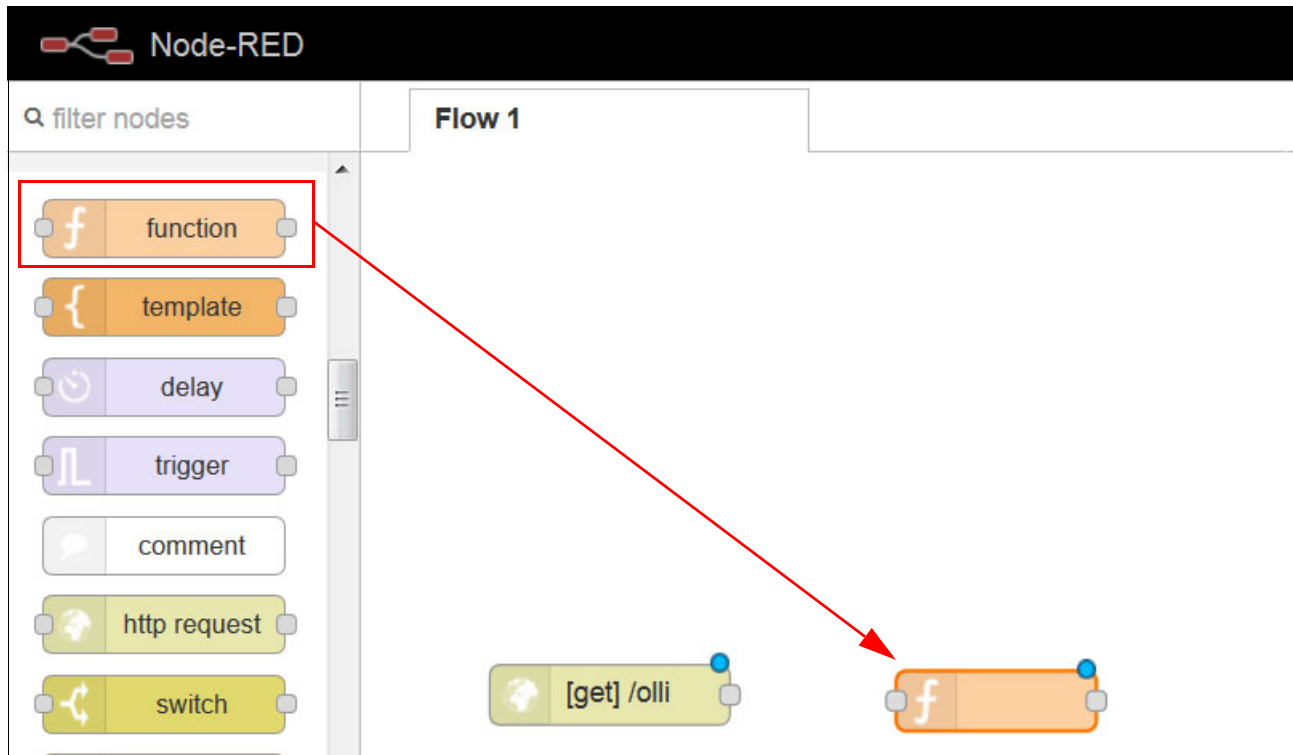


Figure 2-16 Drag the function node to the canvas

4. Connect the output connector on the http input node to the input connector on the function node. The results of the connection are shown in Figure 2-17.

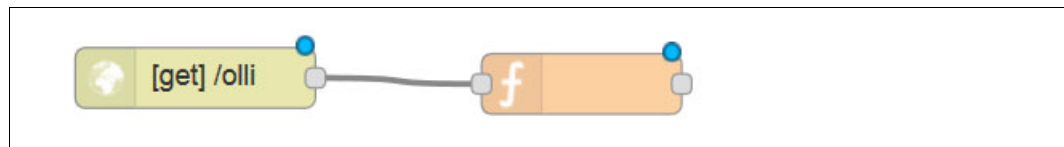


Figure 2-17 Connect the http node to function node

This function is needed because the default message payload output (from the http input node) is a complex JSON object in which each request parameter is a field. The Language Translator service and sentiment analysis both expect a simple string. The function copies the incoming message as a request parameter into the payload field of the msg object so that it is consumable by the remainder of the flow.

5. Double-click the **function** node.
6. The Edit function node dialog opens (Figure 2-18 on page 24). In the Name field, type Swap Arguments. In the Function field, add the following three lines of code, and then click **Done**:

```
msg.lang=msg.payload.lang;  
msg.payload=msg.payload.message;  
return msg;
```

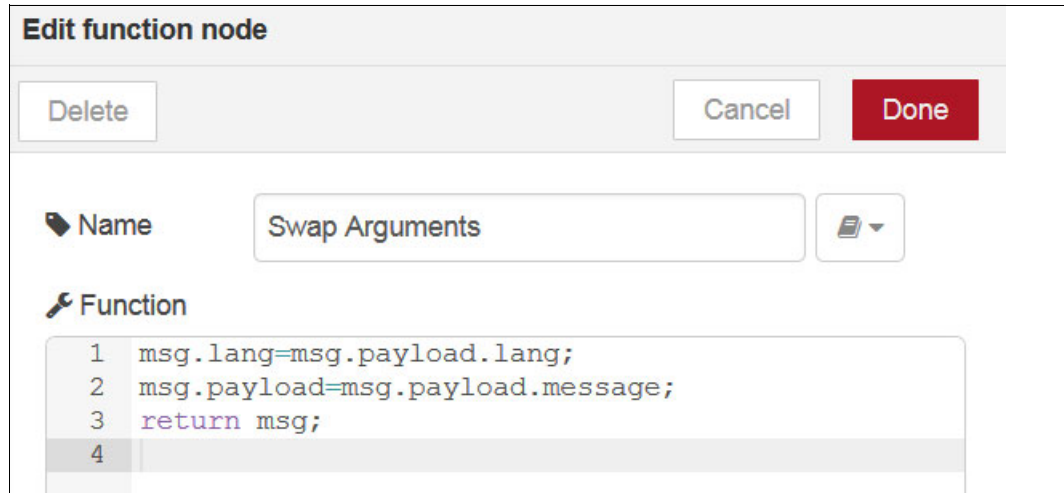


Figure 2-18 Edit function node dialog

7. Drag the **sentiment** analysis node to the canvas and connect it with the function node that you named Swap Arguments (Figure 2-19).

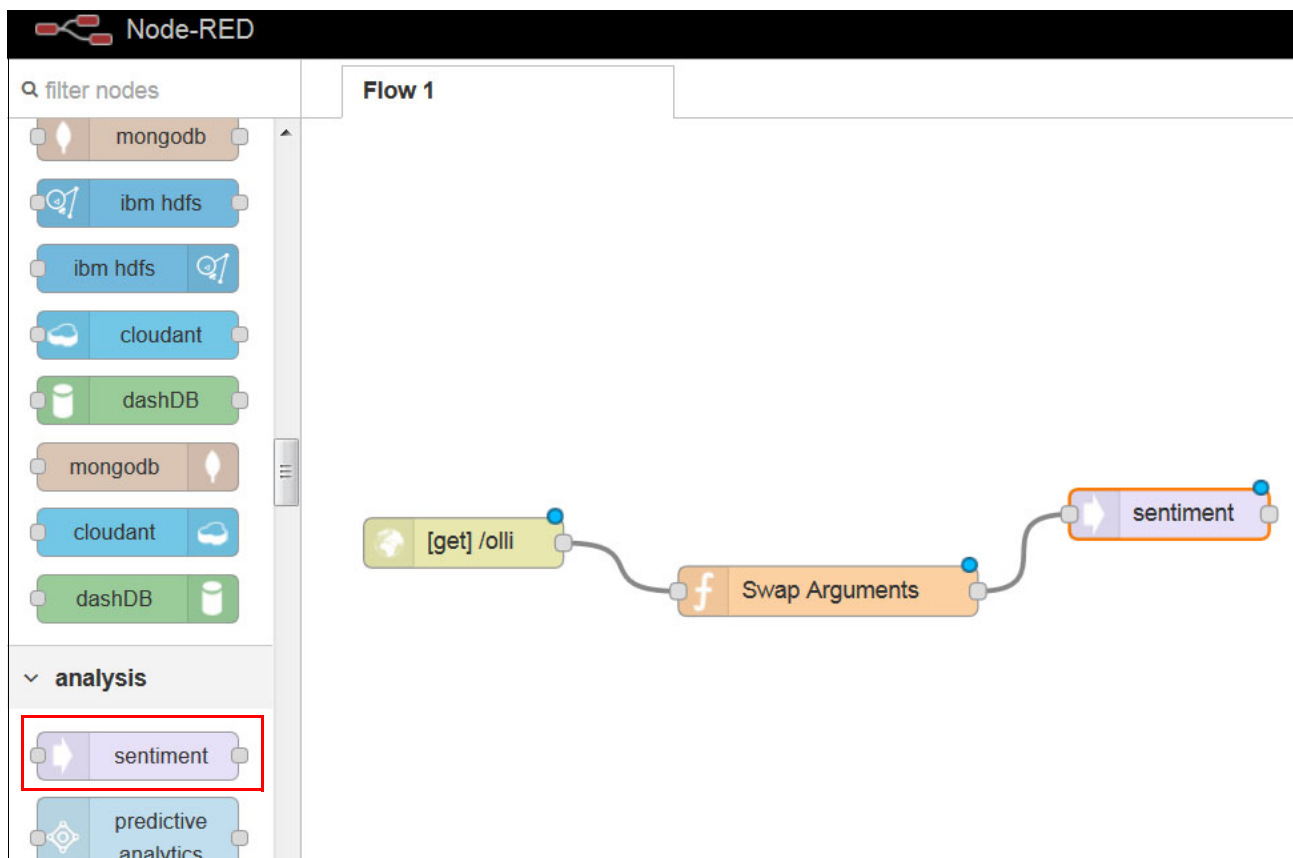


Figure 2-19 Connect the Swap Arguments function node to the sentiment analysis node

Sentiment analysis expects a string in human language and returns a sentiment score ranging from -5 to +5, where negative values code for negative sentiment, positive values code for positive sentiment, and zero means neutral sentiment.

8. Drag the **language translator** node to the canvas and connect it with the sentiment node (Figure 2-20).

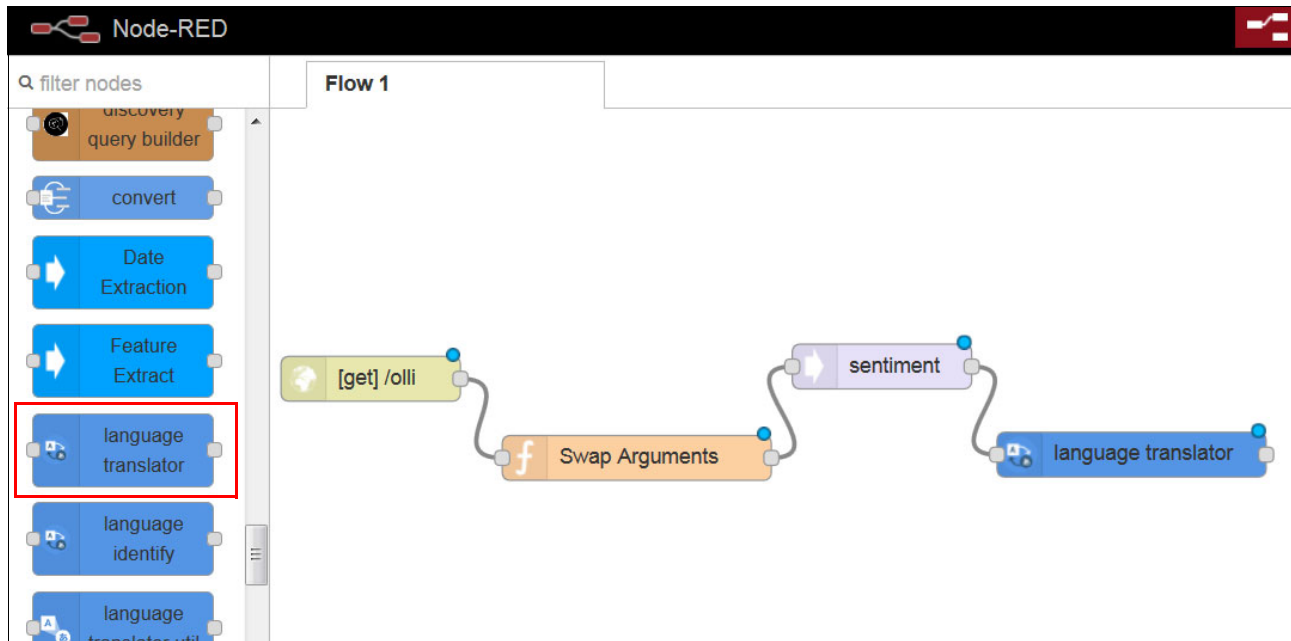


Figure 2-20 Connect language translator node and sentiment node

The language translator node extracts the output from the Language Translator service call and the sentiment node and transforms the output into an easily consumable string to return.

9. Double-click the **language translator** node to edit it.
10. In the Edit language translator node dialog (Figure 2-21 on page 26), select these values and then click **Done**:

<b>Name</b>	You can leave this field blank.
<b>Mode</b>	Translate.
<b>Domain</b>	Conversational.
<b>Source</b>	English.
<b>Target</b>	Spanish.
<b>Parameters Scope</b>	Be sure this check box is selected.

### Edit language translator node

Delete
Cancel
Done

**Name**

**Mode** Translate

**Domains** Conversational

**Source** English

**Target** Spanish

**Parameters**

Scope Not using translation utility

**Note:** When not using the language translation utility to set credentials then 'Not using translation utility' must be checked, else credentials will not be found.

Figure 2-21 Edit language translator node dialog

11. Add another function node and connect it to the language translator node. In this example, the new function node will be named Concatenate Response (Figure 2-22).

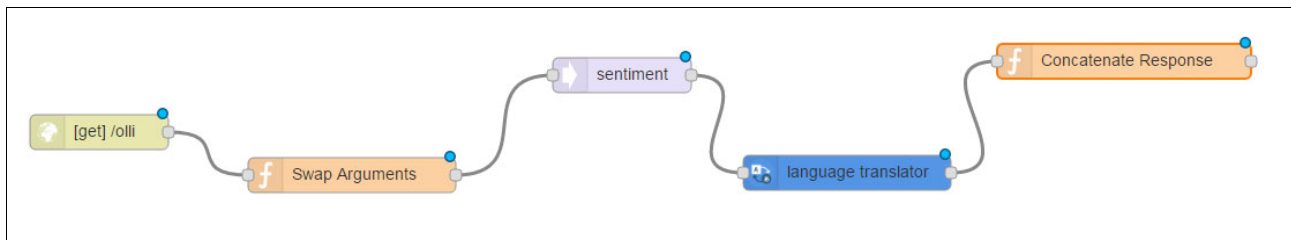


Figure 2-22 Add Concatenate Response node

12. Double-click the new node to open the Edit function node dialog. Name the node **Concatenate Response**, add the following two lines of code to the Function field, and then click **Done** (Figure 2-23):

```
msg.payload=msg.payload+";"+msg.sentiment.score;
return msg;
```

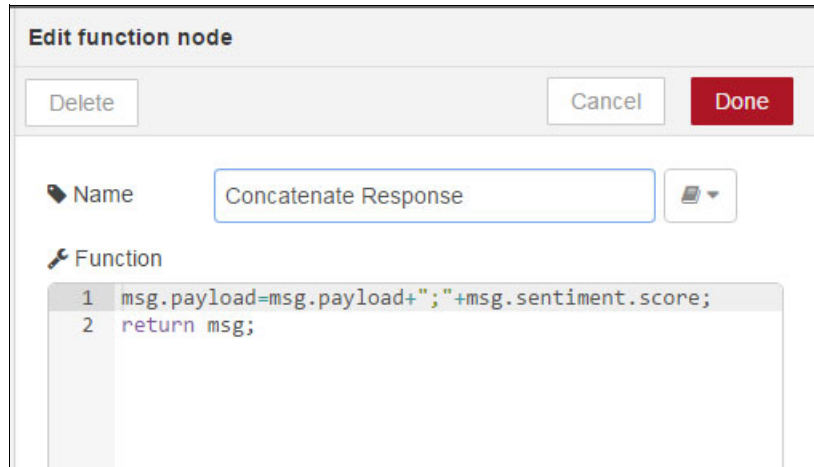


Figure 2-23 Edit function node dialog (Concatenate Response)

13. Add an **http response** node from the available output nodes and connect it to the **Concatenate Response** function node. The completed flow in Node-RED should look like Figure 2-24.

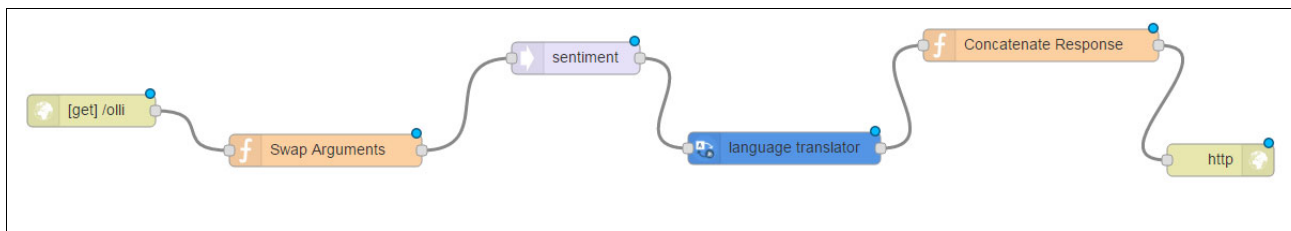


Figure 2-24 Add http response node and connect to Concatenate Response function node

14. If you want to skip the configuration steps, you can import the code in Example 2-1. To do that, copy the flow to your clipboard, and then in the Node-RED flow editor, click the **Menu** icon, and then select **Import** → **Clipboard** (see Figure 2-25 on page 28).

*Example 2-1 Create the flow*

```
[{"id":"223f9f83.0b5bd8","type":"tab","label":"Flow 1"}, {"id":"9740e5f5.2f45b8","type":"http in", "z":"223f9f83.0b5bd8", "name": "", "url": "/olli", "method": "get", "swaggerDoc": "", "x": 73.44999694824219, "y": 181.85000610351562, "wires": [[{"id": "66ca9eb6.77491"}, {"id": "66ca9eb6.77491", "type": "function", "z": "223f9f83.0b5bd8", "name": "swap arguments", "func": "msg.lang=msg.payload.lang;\nmsg.payload=msg.payload.message;\nreturn msg;", "outputs": 1, "noerr": 0, "x": 228.44998168945312, "y": 242.01666259765625, "wires": [[{"id": "a5857265.f3e978"}]}, {"id": "a5857265.f3e978", "type": "sentiment", "z": "223f9f83.0b5bd8", "name": "", "x": 359.4499816894531, "y": 173.3000030517578, "wires": [[{"id": "99f93e2b.a84c98"}]}, {"id": "99f93e2b.a84c98", "type": "watson-translator", "z": "223f9f83.0b5bd8", "name": "", "action": "translate", "basemodel": "ar-en", "domain": "conversational", "srcLang": "en", "destLang": "es", "password": "", "custom": "", "domainhide":
```

```
n": "conversational", "srcLangHidden": "en", "destLangHidden": "es", "baseModelHidden": "", "customHidden": "", "fileType": "forcedglossary", "trainId": "", "lgParams2": true, "ldParamsHidden2": "true", "x": 504.4499816894531, "y": 243.9666748046875, "wires": [{"id": "200bebf2.24ec74"}]}, {"id": "200bebf2.24ec74", "type": "function", "z": "223f9f83.0b5bd8", "name": "concatenate response", "func": "msg.payload=msg.payload+\\\";\\\"+msg.sentiment.score;\\nreturn msg;", "outputs": 1, "noerr": 0, "x": 679.4500122070312, "y": 178.10000610351562, "wires": [{"id": "76849a2d.52dd4c"}]}, {"id": "76849a2d.52dd4c", "type": "http response", "z": "223f9f83.0b5bd8", "name": "", "x": 842.4499816894531, "y": 239.7166748046875, "wires": []}}
```

Figure 2-25 shows the Import menu for importing the flow from your clipboard instead of manually configuring nodes.

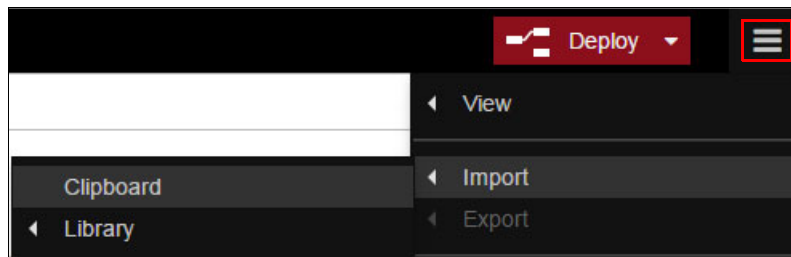


Figure 2-25 Using Import menu

## 2.4.4 Deploying the Node-RED application

From the Node-RED flow editor click **Deploy**. Your REST service is up and running and can be used by everyone (Figure 2-26).

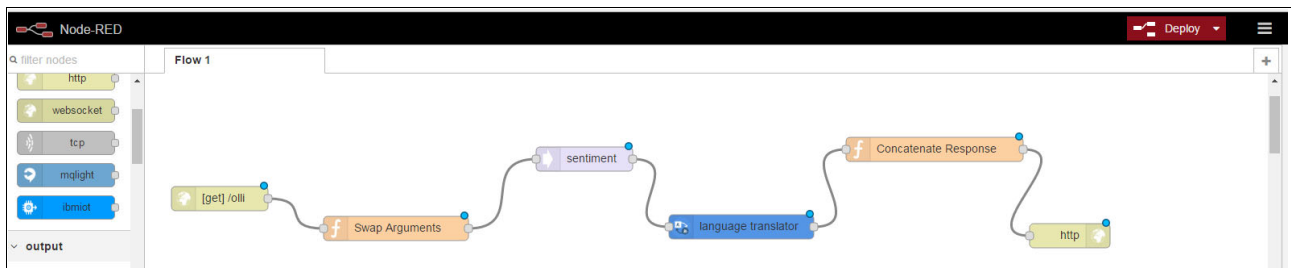


Figure 2-26 REST service

## 2.4.5 Testing the REST call to the Language Translator service

Complete these steps to test your REST service:

1. In your browser, test your REST service by sending a message:

```
http://<<yourappname>>.mybluemix.net/olliv?message=the%20world%20is%20great&lang=mt-enus-eses
```

2. Check that the response translates the text to Spanish and returns a sentiment score:

```
E! mundo es genial;3
```



## 2.5 Quick deployment of application

A Git repository is provided for this chapter so that you can deploy the Node-RED flow without performing the steps in 2.4.3, “Building the REST language translator flow in Node-RED” on page 21.

A sample web page application using AngularJS is also included in this repository. This app consists of an `index.html` file and calls to the REST services.

The quick deployment of the application involves the following steps:

- ▶ Deploying the application to Bluemix.
- ▶ Creating a Language Translator service instance and connecting it to the application.
- ▶ Testing the application.

### 2.5.1 Deploying the application to Bluemix

To deploy the application to Bluemix from the Git repository provided with this book, complete these steps:

1. Enter the following URL

<https://bluemix.net/deploy?repository=https://github.com/snippet-java/redbooks-1t-201-nodesentiment-nodered>

2. The window shown in Figure 2-27 is displayed. Click **Log In**.

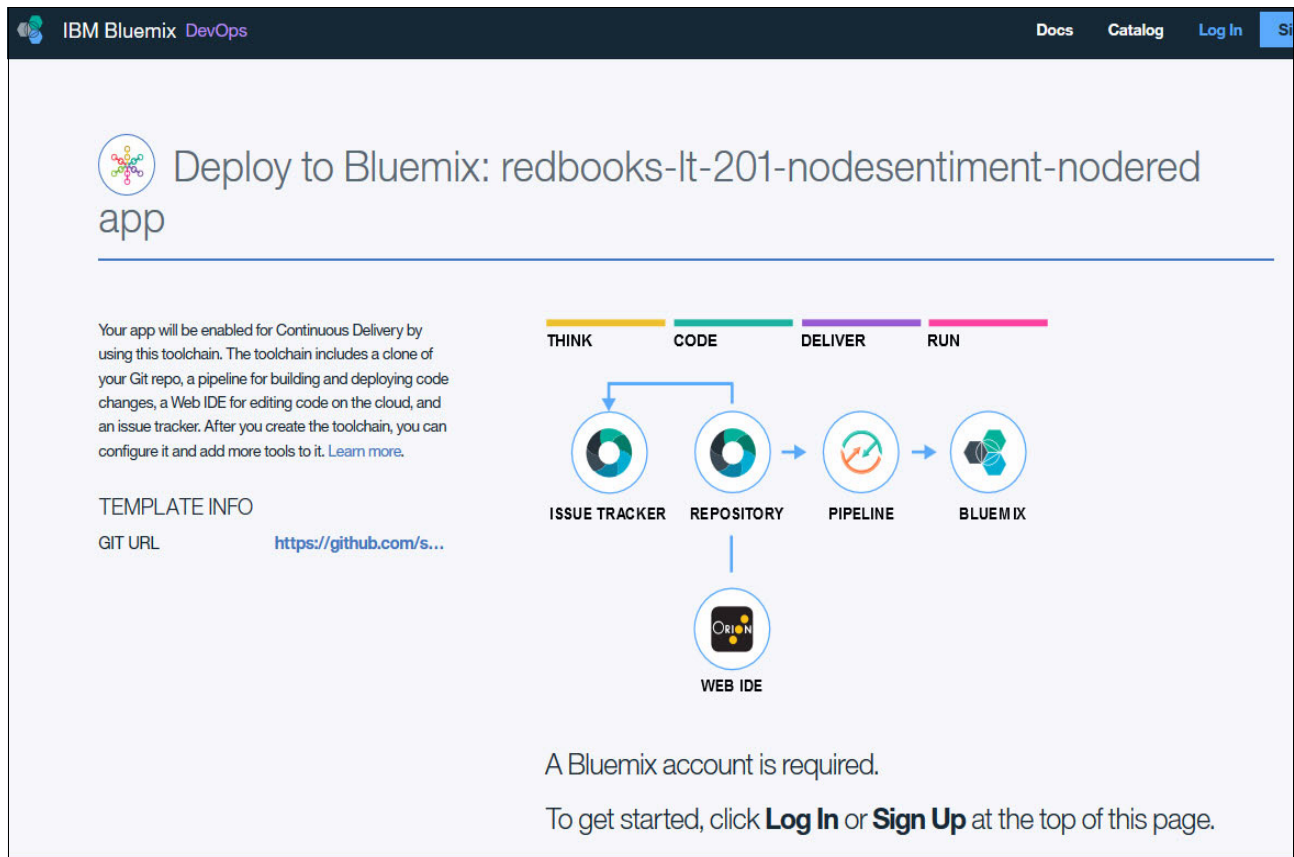


Figure 2-27 Deploy the sample application to Bluemix (part 1 of 3)

3. Enter your Bluemix ID and password.
4. In the next window (Figure 2-28), enter a unique name for your app and then click **Create**.

The screenshot shows the IBM Bluemix DevOps interface for configuring a toolchain. At the top, the breadcrumb is "Deploy to Bluemix: redbooks-lt-201-nodesentiment-nodered app". Below the title, there are three paragraphs of text explaining the toolchain's purpose and how to use it. A central diagram illustrates the CI/CD pipeline with four stages: THINK (Issue Tracker), CODE (Repository), DELIVER (Pipeline), and RUN (Bluemix). A Web IDE tool is also shown connected to the Repository stage. At the bottom, there are form fields for "Organization" (set to "myorg") and "Toolchain Name" (set to "LTdemox"), along with a "Create" button.

Figure 2-28 Deploy the sample application to Bluemix (part 2 of 3)

5. Wait several minutes for deployment to complete. Figure 2-29 shows that the application is deploying.

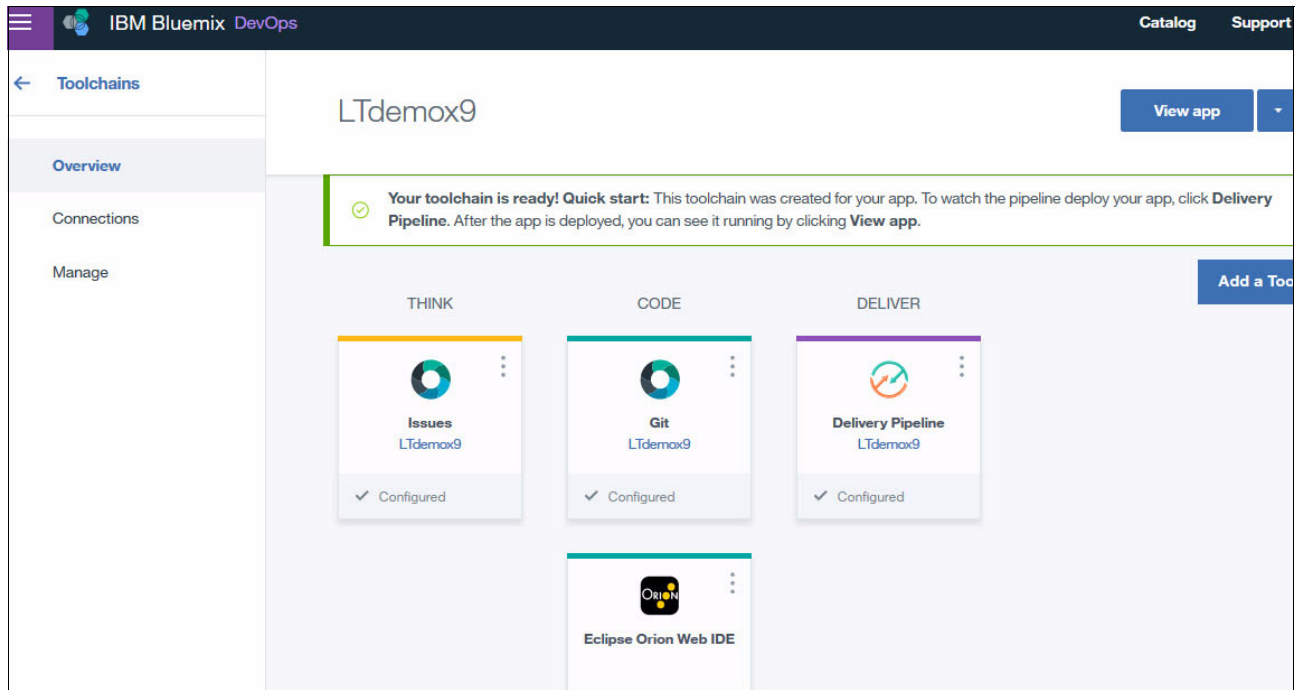


Figure 2-29 Deploy the sample application to Bluemix (part 3 of 3)

6. Check the log in the Delivery Pipeline to verify that deployment is complete (Figure 2-30).

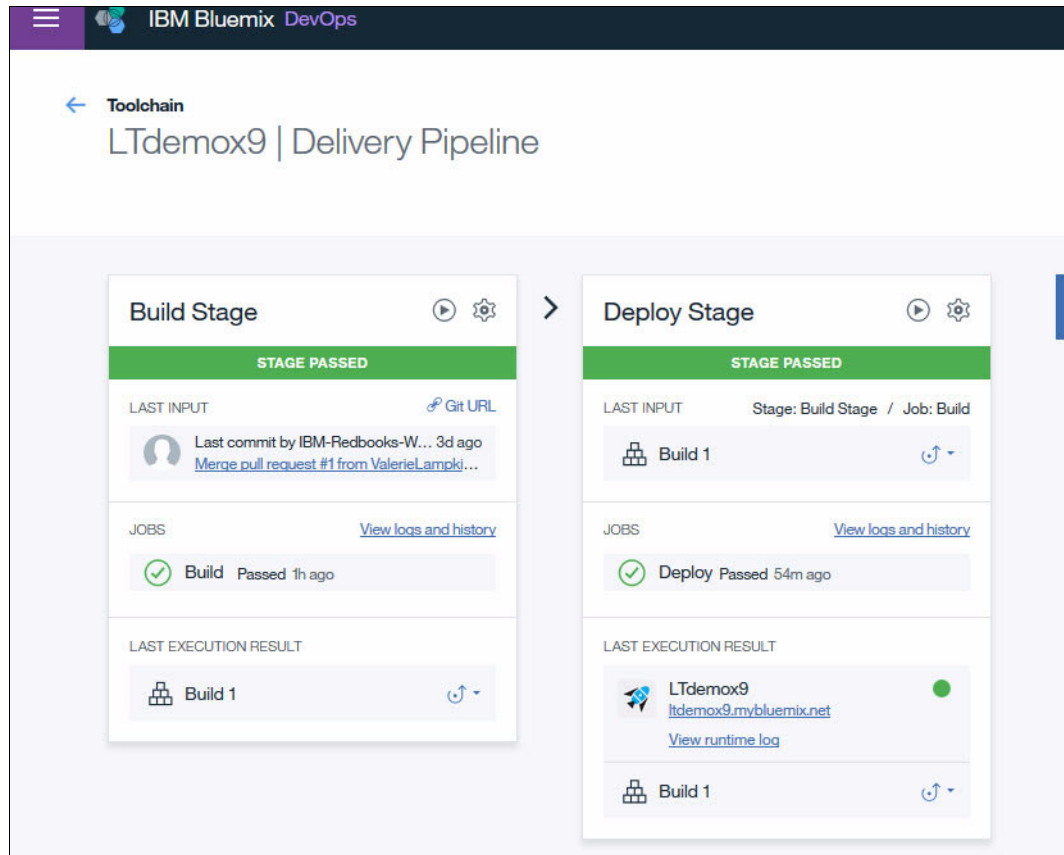


Figure 2-30 Successful deployment of application from Git repository

7. Return to your Bluemix dashboard. You now see the application listed in your dashboard.

## 2.5.2 Creating a Language Translator service instance and connecting it to the application

Complete these steps:

1. From your Bluemix dashboard, click the application to see details. Select **Connections** and click **Connect new** (Figure 2-31).

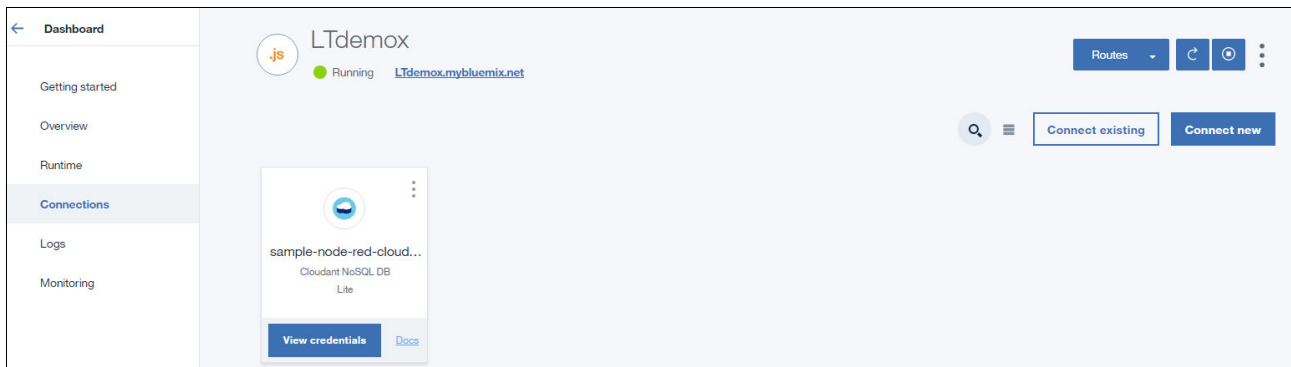


Figure 2-31 Connecting your application to a new service

2. Under Services, click **Watson** and then select **Language Translator** (Figure 2-32).

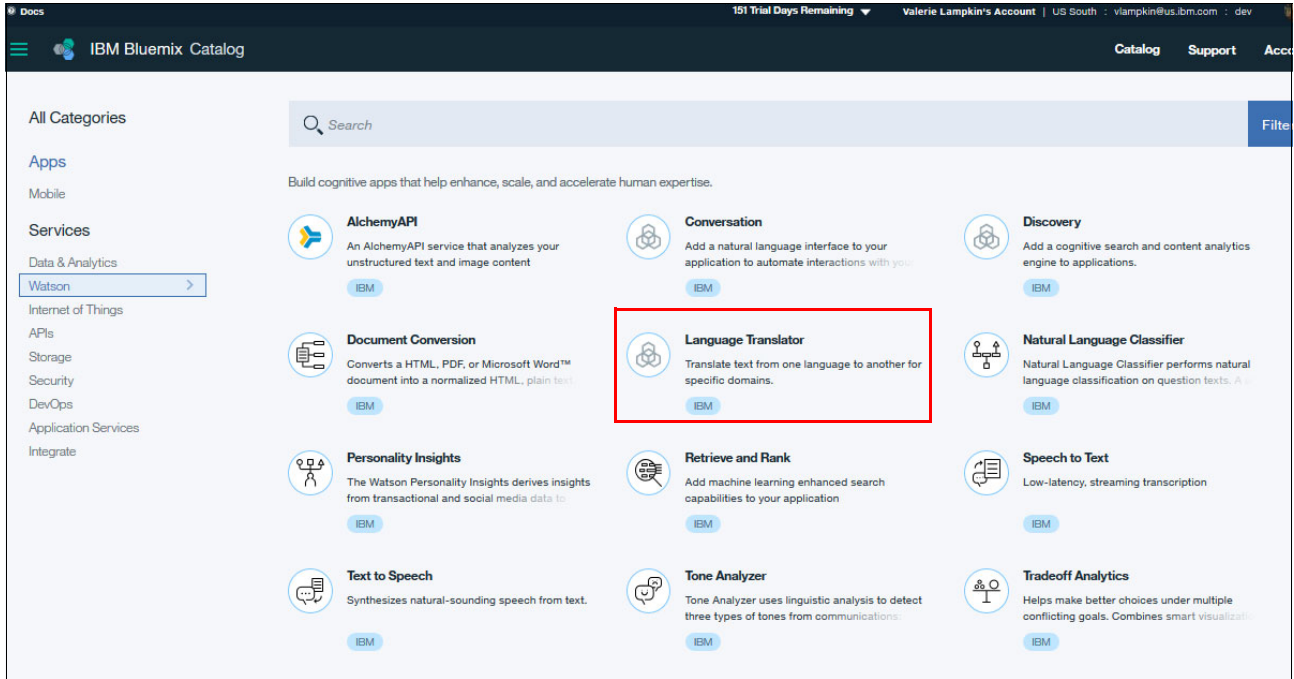


Figure 2-32 Select Language Translator

3. On the Language Translator page (Figure 2-33), either use the default name or edit it to give it a customized name. Then, click **Create**.

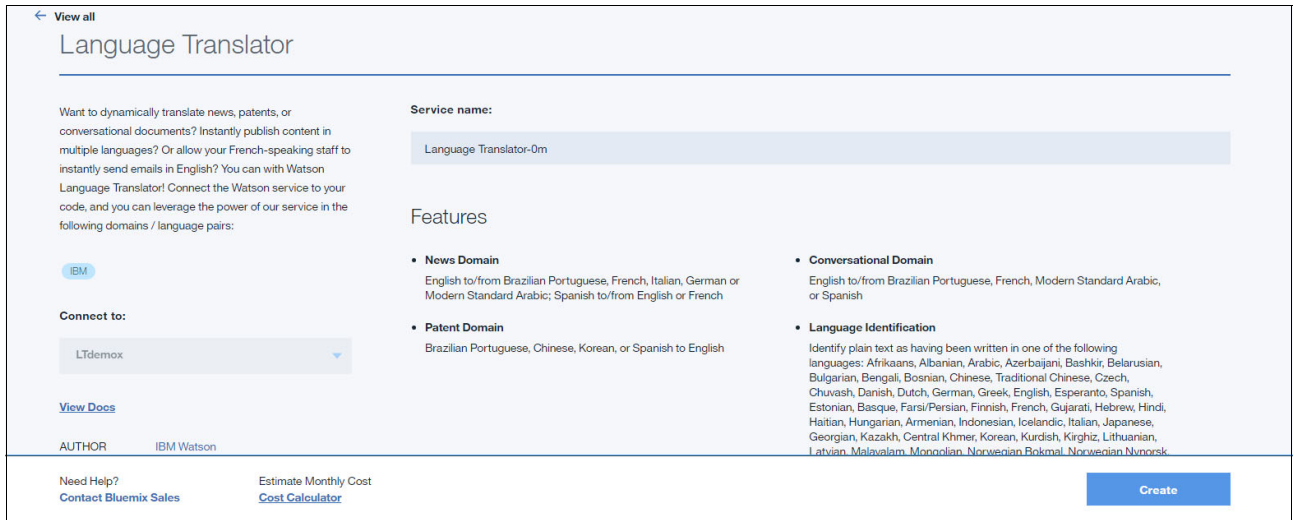


Figure 2-33 Create Language Translator service

4. Click **Restage** and wait while the application is restaging (Figure 2-34).

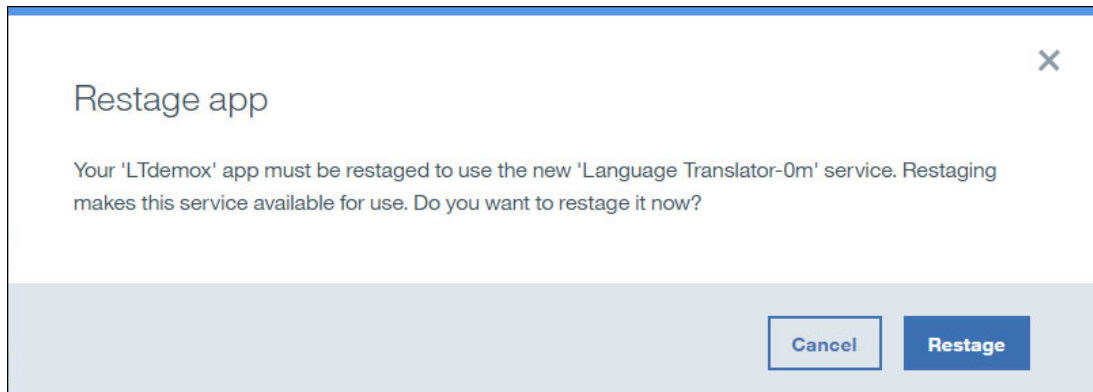


Figure 2-34 Restage application after connecting Language Translator service

5. After a few minutes, the application starts and the status is shown as Running. Your app is running at `http://<your_app_name>.mybluemix.net/ui`.

You can view the Language Translator web app UI by using the following web address:

`https://<<yourappname>>.mybluemix.net/ui`

### 2.5.3 Testing the application

In your browser, open the following address, and then select the language translation **English to Spanish**, enter the text to translate, and click **Translate** (Figure 2-35):

`http://<<yourappname>>.mybluemix.net/ui`



Figure 2-35 Translation application

In the background, the AngularJS code initiates a REST service call to your Node-RED flow. The AngularJS application forms the string returned by the REST service.

Verify that the application worked correctly. You see the text translated from English to Spanish, the sentiment score, and the image of a stylized face, which has an expression that reflects the score. Figure 2-36 on page 35 and Figure 2-37 on page 35 show text translation and emoticon with sentiment score.

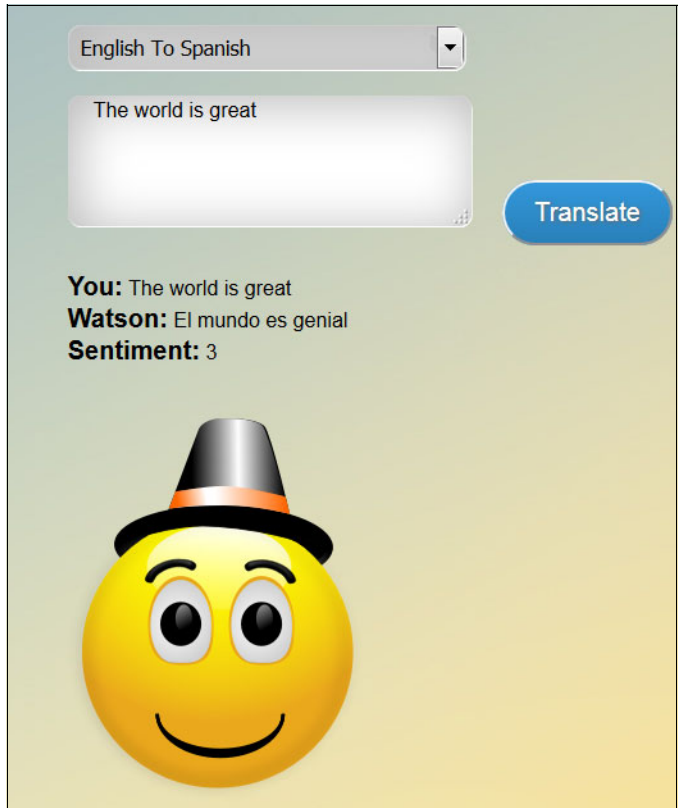


Figure 2-36 Application test results: Sentiment is 3



Figure 2-37 Application test results: Sentiment is -3

## 2.6 References

For helpful information, see the following resources:

- ▶ IBM Bluemix Catalog:  
<https://console.ng.bluemix.net/catalog/>
- ▶ Watson Language Translator service:  
<https://console.ng.bluemix.net/catalog/services/language-translator>
- ▶ The Node-RED Starter demonstrates how to run the Node-RED open source project within Bluemix:  
<https://console.ng.bluemix.net/catalog/starters/node-red-starter>
- ▶ Creating apps with Node-RED Starter:  
<https://console.ng.bluemix.net/docs/starters/Node-RED/nodered.html#nodered>
- ▶ Node-RED Library:  
<https://flows.nodered.org/>
- ▶ IBM Bluemix in IBM developerWorks:  
<https://developer.ibm.com/sso/bmregistration>
- ▶ Watson Language Translator service in Watson Developer Cloud:  
<https://www.ibm.com/watson/developercloud/doc/language-translator/index.html>
- ▶ AngularJS:  
<https://angularjs.org/>
- ▶ GitHub, Inc. sentiment README.md file:  
<https://github.com/thisandagain/sentiment/blob/master/README.md>





## Customizing Language Translator linguistic models

The following linguistic models are provided with the Watson Language Translator service:

- ▶ News: Targeted at news articles and transcripts
- ▶ Conversational: Targeted at conversational colloquialisms
- ▶ Patents: Targeted at technical and legal terminology

Watson Language Translator service provides a mechanism to train the service to perform domain-specific translations by customizing the existing models. It provides a useful function to update the existing models to add context and improve their quality from situation to situation.

For example, perhaps you are creating a translator for customer support and you have company-specific terms that you want handled in a certain way in conversations. Or you are creating a way for your engineers in one country to look up patent data in another language, and you usually file patents on a specific technology. You can use your own data to create a custom dictionary and a custom translation model in the Language Translator service.

**Note:** Customizing models is not supported with IBM Bluemix trial accounts.

The following topics are covered in this chapter:

- ▶ Introduction
- ▶ Custom dictionary
- ▶ Expanding the model to improve translation quality
- ▶ References

## 3.1 Introduction

The provided translation models are updated by adding an input source file. Customization depends on *type* and *content* of the input.

The inputs are in either of the following file formats:

- ▶ Translation Memory Exchange (TMX) file, which has a [specific format](#)
- ▶ Plain text file, which holds a large body of text

Both files must be UTF-8 encoded. Watson Language Translator service currently provides three ways of inputting a source to customize the translation models:

- ▶ Forced glossary

Forced glossary is a collection of terms and phrases with their translations in the target language. Forced glossaries replace the existing terms with their translation from those in the input file. Forced glossaries are used in the TMX format with the Language Translator service.

- ▶ Parallel corpus

Parallel corpus is used for a wide range of applications outside of Language Translator, including building a new translation model from scratch. In the scope of the Language Translator service, it contains pairs of terms or phrases that serve as alternate translation suggestions that you want the translation service to consider. It is used to enhance a provided model to add terms and contexts in the form of phrases that might not be present in original model.

In contrast to the forced glossary, parallel corpora are used to train the existing models, adding the terms and phrases from the input file to the existing training data rather replacing it. They do not override the original domain data.

The parallel corpus is also used in the TMX format with Language Translator. To successfully train a custom model, a parallel corpus document must contain a minimum of 5,000 term and translation pairs.

- ▶ Monolingual corpus

Monolingual corpus is a UTF-8 encoded plain text file that contains a body of text in the target language that is related to what you are translating.

A monolingual corpus serves as a language sample that the service can evaluate and use to improve overall translation quality, for example, to make it more human-like, fluent, and natural. To successfully train a custom model, a monolingual corpus document must contain a minimum of 1,000 sentences.

This chapter includes two use cases as examples to train custom translation models:

- ▶ Custom dictionary
- ▶ Expanding the model to improve translation quality

## 3.2 Custom dictionary

In this use case, you provide a glossary or corpus of terms for training the Language Translator service that will either override the translation that is provided with the default models or add new terms. It uses the forced glossary approach. It is useful when you want to provide specific translation, for example for technical terms, company-specific terminology, country-specific terms, and so on.

Complete these steps to create a custom translation model with a forced glossary option:

1. Prepare the glossary.

A template file with the basic structure of the glossary file is provided in the [Structure of the training data](#) topic at the Customizing your model page of Watson Developer Cloud.

Term and translation pairs that are to be added or that override the existing translation must be enclosed in <tu> tags.

The `xml:lang` attribute in the <tuv> tag identifies the language in which a term is expressed, while the <seg> tag contains the term or the translation.

You can find the language codes at the [ISO 639 Code Tables](#) web page.

The Language Translator service uses ISO 639-1 language codes for all languages except Egyptian Arabic, which uses the ISO 639-3 code.

A TMX-formatted code snippet is in Example 3-1.

*Example 3-1 Code snippet: The glossary file*

---

```
<tu>
  <tuv xml:lang="en">
    <seg>Hello</seg>
  </tuv>
  <tuv xml:lang="es">
    <seg>Hola</seg>
  </tuv>
</tu>
```

---

Example 3-2 shows the sample Dictionary2.tmx file used in this sample.

*Example 3-2 Dictionary2.tmx sample file*

---

```
<?xml version="1.0" encoding="UTF-8"?>
<tmx version="1.4">
  <header creationtool="" creationtoolversion=""
    segtype="phrase" o-tmf="" adminlang="EN"
    srclang="en" datatype="rtf" o-encoding="UTF-8" />
  <body>
    <tu>
      <tuv xml:lang="en">
        <seg>Hello</seg>
      </tuv>
      <tuv xml:lang="es">
        <seg>Saludos</seg>
      </tuv>
    </tu>
    <tu>
      <tuv xml:lang="en">
        <seg>Dictionary</seg>
      </tuv>
      <tuv xml:lang="es">
        <seg>traductor</seg>
      </tuv>
    </tu>
  </body>
</tmx>
```

---

## 2. Train the model.

Example 3-3 shows a `curl` command that can be used to submit the URL with the required parameter in order to train the model with the specified glossary file, `Dictionary2.tmx` in this example.

Replace `username` and `password`, in the example, with the credentials of your Language Translator service instance. A service created using a Bluemix free trial account is not supported.

### *Example 3-3 Command to initiate model training*

---

```
curl -u "{username}":"{password}" -X POST -F base_model_id="en-es" -F
name="english-to-spanish-dictionary" -F forced_glossary=@Dictionary2.tmx
https://gateway.watsonplatform.net/language-translator/api/v2/models
```

---

When the file for training is uploaded successfully, a message is returned (Example 3-4).

### *Example 3-4 Message returned after glossary file is uploaded successfully*

---

```
{
  "model_id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "source": "en",
  "target": "es",
  "base_model_id": "en-es",
  "domain": "news",
  "customizable": false,
  "default_model": false,
  "owner": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "status": "dispatching",
  "name": "english-to-spanish-dictionary",
  "train_log": null
}
```

---

The model ID returned in the response is used for monitoring the status of the model and for custom translations.

To be able to monitor the status of the training, use the `curl` command (Example 3-5).

### *Example 3-5 Command to monitor the status of training*

---

```
curl -u "{username}":"{password}"
https://gateway.watsonplatform.net/language-translator/api/v2/models/xxxxxxxx-x
xxx-xxxx-xxxx-xxxxxxxxxxxx
```

---

This command returns one of four states:

- training
- queue@<#>
- error
- available

Example 3-6 shows a response to check the status.

### *Example 3-6 Response with status*

---

```
{
  "model_id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx ",
  "source": "en",
  "target": "es",
  "base_model_id": "en-es",
```

```
"domain":"news",
"customizable":false,
"default_model":false,
"owner":"xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
"status":"available",
"name":"english-to-spanish-dictionary",
"train_log":null
}
```

---

### 3. Use the trained model.

After the model is successfully trained, specify the model identifier (retrieved in step 2 on page 40) to invoke the custom translation model.

For example, to see the effect of training the model using the Dictionary2.tmx glossary file, issue the `curl` command, shown in Example 3-7.

*Example 3-7 Command to see the effect of training the model*

---

```
curl -u "{username}":"{password}" -X POST -F "text=Hello" -F
"model_id=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
https://gateway.watsonplatform.net/language-translator/api/v2/translate
```

---

The response is the message *Saludos* as defined in the dictionary.

Similarly, translation for using the word *Dictionary* is shown in Example 3-8. The response is *Traductor* as defined in the dictionary.

*Example 3-8 Translation using Dictionary*

---

```
curl -u "{username}":"{password}" -X POST -F "text=Dictionary" -F
"model_id=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx"
https://gateway.watsonplatform.net/language-translator/api/v2/translate
```

---

This example shows that the translation for the words *Hello* (*Hola*) and *Dictionary* (*Diccionario*) in the provided English to Spanish translation model have been overridden by *Saludos* and *Traductor* as specified in the glossary file.

## 3.3 Expanding the model to improve translation quality

Translation models are not trained to handle all possible jargons and contexts. Certain contextual translations, such as translations that are specific to IT terms or industry and corporate jargon, can be provided by expanding a linguistic model. A parallel corpus TMX file is used in these situations. This section provides an example.

Complete the following steps:

### 1. Obtain or create a parallel corpus.

Following a similar format as described in 3.2, “Custom dictionary” on page 38, existing models can be expanded to cover translations not previously included in their training sets.

In this case, use the parallel corpus that has the name `fren.tmx`.

**Note:** The `fren.tmx` file is not provided with this book.

## 2. Train the model.

For this example, use your own corpus for expanding the French translation model. The corpus uses the TMX format but must contain a minimum of 5,000 term-translation pairs and they will not overwrite the original domain data of the model, but will add to it. To train the model, use the `curl` command, shown in Example 3-9.

### *Example 3-9 Command to train the model*

---

```
curl -u "{username}":"{password}" -X POST -F base_model_id="en-fr" -F
name="english-to-french-expansion" -F parallel_corpus=@fren.tmx
https://gateway.watsonplatform.net/language-translator/api/v2/models
```

---

After the model is submitted for training, you receive a string in the following format:

```
xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
```

This string is the model ID. The model ID that is returned in the response is used for monitoring the status of the model and for custom translations. In this sample, the response is shown in Example 3-10.

### *Example 3-10 Response to training*

---

```
{
  "model_id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "source": "en",
  "target": "fr",
  "base_model_id": "en-fr",
  "domain": "news",
  "customizable": false,
  "default_model": false,
  "owner": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "status": "dispatching",
  "name": "english-to-french-expansion",
  "train_log": null
}
```

---

The training status can be monitored by using the command in Example 3-5 on page 40. That command returns one of four states:

- training
- queue@<#>
- error
- available

Depending on the size of the parallel corpus, the training can take some time. When the model is trained, you will see a similar response when you check the model status (Example 3-11).

### *Example 3-11 Response from checking model status*

---

```
{
  "model_id": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
  "source": "en",
  "target": "fr",
  "base_model_id": "en-fr",
  "domain": "news",
  "customizable": false,
  "default_model": false,
  "owner": "xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
}
```

```
"status":"available",  
"name":"english-to-french-expansion",  
"train_log":null  
}
```

---

### 3. Use the custom translations.

After the model is successfully trained, specify the model identifier (retrieved in step 2 on page 42) to invoke the custom translation model.

For example, to see the effect of training the model in this example, issue the `curl` command shown in Example 3-12.

#### *Example 3-12 Command to see the effect of training*

---

```
curl -u "{username}":"{password}" -X POST -F "text=Set the current state to up  
if you want the gateway activated at startup or to down if you want the gateway  
inactive at startup." -F "model_id=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx"  
"https://gateway.watsonplatform.net/language-translator/api/v2/translate"
```

---

The response to this model is according to the trained parallel corpus, shown in Example 3-13.

#### *Example 3-13 Response to using trained model*

---

```
Définissez Démarrage comme état en cours si vous souhaitez que la passerelle  
soit activée au démarrage ou Arrêt si vous souhaitez qu'elle ne soit pas active
```

---

To see the difference, if you translate the same sentence using the parent model from which the custom model was trained, the results differ (Example 3-14).

#### *Example 3-14 Command to translate using the parent model*

---

```
curl -u "{username}":"{password}" -X POST -F "text=Set the current state to up  
if you want the gateway activated at startup or to down if you want the gateway  
inactive at startup." -F "source=en" -F "target=fr"  
"https://gateway.watsonplatform.net/language-translator/api/v2/translate"
```

---

Example 3-15 shows the result of translation using the parent model.

#### *Example 3-15 Results using parent model*

---

```
Définissez l'état actuel à la si vous souhaitez que la passerelle soit activée  
au démarrage ou si vous souhaitez arrêter la passerelle inactive au démarrage.
```

---

## 3.4 References

For helpful information, see the following resources:

- ▶ Customizing your model:

<https://www.ibm.com/watson/developercloud/doc/language-translator/customizing.html>

<http://www.ibm.com/watson/developercloud/doc/language-translator/customizing.shtml#training>

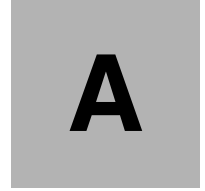
- ▶ TMX (Translation Memory eXchange):

<http://www.ttt.org/oscarstandards/tmx/tmx14.dtd>

- ▶ ISO 639 Code Tables:

<http://www-01.sil.org/iso639-3/codes.asp>





## Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

### Locating the web material

The following Git repository is available to help you with the examples in this book:

<https://github.com/snippet-java/redbooks-1t-201-nodesentiment-nodered>



# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

## IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

The volumes in the *Building Cognitive Applications with IBM Watson Services* series:

- ▶ *Volume 1 Getting Started*, SG24-8387
- ▶ *Volume 2 Conversation*, SG24-8394
- ▶ *Volume 3 Visual Recognition*, SG24-8393
- ▶ *Volume 4 Natural Language Classifier*, SG24-8391
- ▶ *Volume 5 Language Translator*, SG24-8392
- ▶ *Volume 6 Speech to Text and Text to Speech*, SG24-8388
- ▶ *Volume 7 Natural Language Understanding*, SG24-8398

You can search for, view, download or order these documents and other IBM Redbooks, Redpapers™, Web Docs, draft and additional materials, at the following website:

[ibm.com/redbooks](http://ibm.com/redbooks)

## Online resources

These websites are also relevant as further information sources:

- ▶ *Add language translation to your apps with the IBM Watson* article, available in IBM developerWorks:

<https://www.ibm.com/developerworks/cloud/library/cl-add-language-translation-to-your-apps-with-watson-app>

- ▶ Watson Language Translator API Reference documentation:

<https://www.ibm.com/watson/developercloud/language-translator/api/v2/>

- ▶ The “Identify language” topic in the Watson Language Translator API Reference:

<https://www.ibm.com/watson/developercloud/language-translator/api/v2/#identify>

- ▶ IBM Watson Language Translator service:

[https://console.ng.bluemix.net/catalog/services/language-translator?env\\_id=ibm:yp:us-south](https://console.ng.bluemix.net/catalog/services/language-translator?env_id=ibm:yp:us-south)

- ▶ IBM Bluemix catalog:

<https://console.ng.bluemix.net/catalog/>

- ▶ Node-RED Starter:

[https://console.ng.bluemix.net/catalog/starters/node-red-starter?env\\_id=ibm:yp:us-south](https://console.ng.bluemix.net/catalog/starters/node-red-starter?env_id=ibm:yp:us-south)

- ▶ Node-RED flow editor:  
<https://nodered.org/>
- ▶ AngularJS:  
<https://angularjs.org/>
- ▶ AFINN-111 sentiment score:  
<https://github.com/thisandagain/sentiment/blob/master/README.md>
- ▶ IBM Bluemix account:  
<https://console.ng.Bluemix.net>
- ▶ Translation Memory Exchange (TMX) file format:  
<http://www.ttt.org/oscarstandards/tmx/tmx14.dtd>
- ▶ Custom translation model glossary template file to structure the training data:  
<https://www.ibm.com/watson/developercloud/doc/language-translator/customizing.html#structure>
- ▶ ISO 639 Code Tables:  
<http://www-01.sil.org/iso639-3/codes.asp>

Also see the list of online resources for the following chapters in this book:

- ▶ Basics of Watson Language Translator API: 1.3, “References” on page 9
- ▶ Integrating sentiment analysis and language translation in applications: 2.6, “References” on page 36
- ▶ Customizing Language Translator linguistic models: 3.4, “References” on page 44

## Help from IBM

IBM Support and downloads

[ibm.com/support](http://ibm.com/support)

IBM Global Services

[ibm.com/services](http://ibm.com/services)









SG24-8392-00

ISBN 0738442585

Printed in U.S.A.

Get connected

