Building Cognitive Applications with IBM Watson Services: Volume 6
Speech to Text and Text to Speech

Felipe Santiago
Pallavi Singh
Lak Sri
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.
## Contents

### Notices

- Trademarks
  - vi

### Preface

- vii

### Authors

- vii

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

- viii

### Comments welcome

- ix

### Stay connected to IBM Redbooks

- ix

### Chapter 1. Basics of Watson Speech to Text and Text to Speech services

- 1.1 Watson Speech to Text service
  - 2
    - 1.1.1 Authentication
      - 2
    - 1.1.2 Use case example
      - 2
    - 1.1.3 Speech to Text flow
      - 2
    - 1.1.4 Code snippets
      - 3
- 1.2 Watson Text to Speech service
  - 7
    - 1.2.1 Authentication
      - 7
    - 1.2.2 Use case example
      - 7
    - 1.2.3 Text to Speech flow
      - 8
    - 1.2.4 Code snippets
      - 8

### Chapter 2. Creating Bluemix services

- 15
  - 2.1 Creating the Speech to Text service
    - 16
  - 2.2 Creating the Text to Speech service
    - 17
  - 2.3 Creating the Language Translator service
    - 18
  - 2.4 Creating the Personality Insights service
    - 19
  - 2.5 Retrieving the service credentials
    - 20

### Chapter 3. Personality Analyzer

- 23
  - 3.1 Getting started
    - 24
    - 3.1.1 Objectives
      - 24
    - 3.1.2 Prerequisites
      - 24
    - 3.1.3 Expected results
      - 24
  - 3.2 Architecture
    - 26
  - 3.3 Two ways to deploy the application: Step-by-step and quick deploy
    - 26
  - 3.4 Step-by-step implementation
    - 27
    - 3.4.1 Downloading the project from Git
      - 27
    - 3.4.2 Reviewing the project structure
      - 27
    - 3.4.3 Setting up Watson services in Bluemix
      - 29
    - 3.4.4 Providing credentials for the Watson services
      - 29
    - 3.4.5 Using the Watson Speech to Text (STT) service
      - 29
    - 3.4.6 Using the Watson Personality Insights (PI) service
      - 31
    - 3.4.7 Providing a unique name for your application
      - 33
    - 3.4.8 Deploying the application on Bluemix
      - 33
    - 3.4.9 Testing the application
      - 34
  - 3.5 Quick deployment of application
    - 35
  - 3.6 References
    - 36

### Chapter 4. Real-time transcription

- 37
4.1 Getting started ................................................................. 38
  4.1.1 Objectives .................................................................. 38
  4.1.2 Prerequisites ............................................................. 38
  4.1.3 Expected results ....................................................... 39
4.2 Architecture ....................................................................... 40
4.3 Two ways to deploy the app: Step-by-step and quick deploy ............... 41
4.4 Step-by-step implementation ............................................... 41
  4.4.1 Preparing the mobile device where the app will run .................. 41
  4.4.2 Downloading the project from the Git repository .................... 42
  4.4.3 Importing the project to Android Studio ............................. 42
  4.4.4 Reviewing the project structure and the code provided ............. 43
  4.4.5 Importing the required Watson libraries ............................. 45
  4.4.6 Using the Speech to Text (STT) service ............................ 46
  4.4.7 Using the Text to Speech (TTS) service ............................ 49
  4.4.8 Setting up the credentials ............................................ 52
  4.4.9 Testing the application ............................................... 52
4.5 Quick deployment of application ....................................... 52
4.6 References ....................................................................... 54

Appendix A. Additional material ............................................. 55
Locating the Git repositories .................................................. 55

Related publications ............................................................. 57
IBM Redbooks ..................................................................... 57
Online resources .................................................................... 57
Help from IBM ..................................................................... 58
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 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®
developerWorks®
IBM®
IBM Watson®
IBM Watson IoT™
Redbooks®
Redbooks (logo) ®
Redpapers™
Tivoli®
Watson™
Watson IoT™

The following terms are trademarks of other companies:

Weather Insights, and Wundersearch are trademarks or registered trademarks of TWC Product and Technology LLC, an IBM Company.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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 6, introduces the Watson Speech to Text (converts audio voice into written text) and Watson Text to Speech (converts written text into natural sounding audio in various languages and voices) services. This book introduces concepts that you need to understand to use these Watson services and provides simple code examples to illustrate the use of the APIs. This book includes examples of applications that demonstrate how to use the Watson Text to Speech and Speech to Text services in practical use cases. You can develop and deploy the sample applications 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.

Authors

This book was produced by a team of specialists from around the world working in collaboration with the IBM International Technical Support Organization.

Felipe Santiago is a Software Engineer at the Linux Technology Center, IBM Brazil. His areas of expertise include applications development, tooling software, Linux systems, cloud computing, software automation tools, and cognitive computing. Felipe participated in several projects to design and build cognitive solutions, such as an application to analyze and compare sentiments generated by articles, news, and social media, and an Internet of Things (IoT) project that developed a cognitive speaking plant vase with sensors attached to it, capable of establishing conversations, all using Speech to Text, Text to Speech, Weather Insights® and other APIs. Felipe holds a Bachelor of Engineering degree in Computer Engineering from Universidade Federal do Rio Grande do Norte, Brazil.
Pallavi Singh is a Senior Software Engineer in India Software Labs, IBM India Pvt Ltd. She has 15 years of experience in the software industry and is currently working as a Test Architect with the IBM Watson Customer Engagement team. Pallavi is also a member of the IBM Academy of Technology. She has designed and developed various cognitive projects at IBM. Additionally, she has co-authored several IBM Redbooks and has a patent to her name. She holds a Masters degree in Computer Applications.

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:

Swin Voon Cheok
Ecosystem Development (EcoD) Strategic Initiative, IBM Systems

Iain McIntosh
IBM Watson and Cloud Platform

Juan Pablo Napoli
Skills Academy Worldwide Leader, Global University Programs

Teja Tummalapalli
IBM Digital Business Group

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
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

▸ Send your comments in an email to:
  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:

▸ Stay current on recent Redbooks publications with RSS Feeds:
  http://www.redbooks.ibm.com/rss.html
Basics of Watson Speech to Text and Text to Speech services

This chapter introduces the following Watson services:

- The Watson Speech to Text service converts audio voice into written text.
- The Watson Text to Speech service converts written text into natural sounding audio in a variety of languages and voices.

This chapter provides basic information about how to use the APIs and includes simple code examples in the following technologies:

- Node.js
- Java
- cURL
- Node-RED

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

The following topics are covered in this chapter:

- Watson Speech to Text service
- Watson Text to Speech service
1.1 Watson Speech to Text service

The Speech to Text service converts speech into readable text according to the language that the user specifies. The service is capable of transcribing speech from various languages and audio formats to text with low latency. For most languages, the service supports two sampling rates: broadband and narrowband.

Watch the IBM Speech to Text video to see an introduction of how this service works.

1.1.1 Authentication

You authenticate to the Speech to Text service by providing the user name and password that are provided in the service credentials for the service instance that you want to use. For more information, see 2.5, “Retrieving the service credentials” on page 20.

1.1.2 Use case example

The Speech to Text service can take an audio voice file as input and convert the audio into written text.

This service can be used in practical applications such as these:
- Transcribe calls in a contact center to identify what is being discussed, when to escalate calls, and to understand content from multiple speakers.
- Transcribe a technical explanation provided by a subject matter expert to help to create documentation.
- Transcribe speech from a movie and create subtitles or captions.
- Transcribe voice mail messages to obtain a written log of messages left, for example, by clients or patients.
- Create voice control applications, for example an automated home control system that is based on your speech. Functions can include switching the light on and off, controlling electronic devices, or even opening the front door remotely.

1.1.3 Speech to Text flow

The flow of the Speech to Text service is shown in Figure 1-1.
The figure shows the following flow:
1. User specifies an audio file, which can be recorded live or be previously prepared.
2. The speech is sent to the Speech to Text Service for processing.
3. The Speech to Text service generates the text based on the audio file.

1.1.4 Code snippets

The calls shown in this section are performed with no session established. In all examples, user credentials must be inserted in order to access the API.

Download the audio samples to use as input:
https://www.ibm.com/watson/developercloud/speech-to-text/api/v1/audio-files.zip

Save the audio files in the same directory as the program.

In these code examples, the Speech to Text service is used to transcribe the audio files that are named audio-file1.flac and audio-file2.flac. The speech is recorded in English and some keywords are provided for better accuracy, but providing the keywords is not required.

**Node.js**

Example 1-1 shows a Node.js code snippet for Speech to Text.

```javascript
// Specify the references to required libraries
var SpeechToTextV1 = require('watson-developer-cloud/speech-to-text/v1');
var fs = require('fs');

// Provide the credentials for using Speech to Text API
var speech_to_text = new SpeechToTextV1 (
    username: '<enter your username here>',
    password: '<enter your password here>'
);

// Specify the input audio files and other configuration parameters
var files = ['audio-file1.flac', 'audio-file2.flac'];
for (var file in files) {
    var params = {
        audio: fs.createReadStream(files[file]),
        content_type: 'audio/flac',
        timestamps: true,
        word_alternatives_threshold: 0.9,
        keywords: ['colorado', 'tornado', 'tornadoes'],
        keywords_threshold: 0.5,
        continuous: true
    };

    // Pipe the generated text to a file
    speech_to_text.recognize(params, function(error, transcript) {
        if (error)
            console.log('Error:', error);
        else
            console.log(JSON.stringify(transcript, null, 2));
    });
}
```
**Java**

Example 1-2 shows a Java code snippet for Text to Speech.

```java
example 1-2  Code snippet: Java

import java.io.File;
// Specify references to required libraries
import com.ibm.watson.developer_cloud.speech_to_text.v1.SpeechToText;
import com.ibm.watson.developer_cloud.speech_to_text.v1.model.RecognizeOptions;
import com.ibm.watson.developer_cloud.speech_to_text.v1.model.SpeechResults;
public class WatsonSTT {
    public static void main(String[] args) {
        SpeechToText service = new SpeechToText();
        // Provide the credentials to use the Speech to Text API
        service.setUsernameAndPassword("<enter your username here>", "<enter your password here>");

        // Specify the input audio files and other configuration parameters
        RecognizeOptions options = new RecognizeOptions.Builder()
                                .contentType("audio/flac").timestamps(true)
                                .wordAlternativesThreshold(0.9)
                                .keywords(new String[]{"colorado", "tornado", "tornadoes"})
                                .keywordsThreshold(0.5).continuous(true).build();

        String[] files = {"audio-file1.flac", "audio-file2.flac"};
        for (String file : files) {
            SpeechResults results = service.recognize(new File(file), options).execute();
            // Print the generated text
            System.out.println(results);
        }
    }
}
```

**cURL**

The cURL tool allows you to access the API directly from the command line. The following parameters are used in Example 1-3.

- `-u` Specifies user name and password.
- `--header` Specifies the header of the HTTP request. The header contains information such as file type, path to the file, and other parameters that can be specified according to the API specification.
- `-X` Specifies the HTTP method for the request.

```cURL
example 1-3  Code snippet: cURL

curl -X POST -u "<enter your username here>"":"<enter your password here>" --header
"Content-Type: audio/flac" --data-binary @audio-file1.flac" --data-binary
"@audio-file2.flac"
"https://stream.watsonplatform.net/speech-to-text/api/v1/recognize?timestamps=true
&word_alternatives_threshold=0.9&keywords=%22colorado%22%2C%22tornado%22%2C%22torn
adoes%22&keywords_threshold=0.5&continuous=true"
```

The parameters for a POST request in the `curl` command are passed using the service URL. Separate the parameters and their values by using the ampersand (&) character. For more information, access the [Speech to Text API documentation](https://www.ibm.com/watson/documentation/speech-to-text/).
Node-RED

Figure 1-2 shows how to build Watson Speech to text using Node-RED.

![Figure 1-2 The four steps in Node-RED](image)

The steps in Node-RED are as follows:

1. Drag an **inject** node to the canvas to enable the start of the execution of the program. Edit the node (Figure 1-3).

![Figure 1-3 Edit inject node dialog](image)

2. Insert a **file** node to specify the location of the audio file (Figure 1-4 on page 6).
3. Drag the **Speech to Text** service node. Provide the credentials of the service and configure the parameters (Figure 1-5).

![Edit file in node dialog](image)

**Figure 1-4  Edit file in node dialog**

**The File node:** The default Node-RED distribution does not include the file node. You can add the node by modifying the `package.json` file. See the **Adding Nodes** topic on the Node-RED website.

![Edit speech to text node dialog](image)

**Figure 1-5  Edit speech to text node dialog**
4. Insert a **debug** node to show the text that is being output after conversion by the Watson service (Figure 1-6).

![Edit debug node dialog](image)

**Figure 1-6  Edit debug node dialog**

### 1.2 Watson Text to Speech service

Watson Text to Speech is a speech synthesizer API that converts written text into audible speech. It is multilingual, so it accepts text as input and outputs an audio file in various languages. The input text can be plain text or written in Speech Synthesis Markup Language (SSML). Additionally, it outputs various speaking styles, pronunciation, pitch, and speaking rate.

#### 1.2.1 Authentication

You authenticate to the Text to Speech service by supplying the user name and password that are provided in the service credentials for the service instance that you want to use. For more information, see 2.5, “Retrieving the service credentials” on page 20.

#### 1.2.2 Use case example

The service can take a text file as input and convert the contents into an audible file in the specified voice.

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

- Create audio narration from a written script for a variety of applications, such as online tutorials for an e-learning course, audio books, and so on.
- Provide callers with information, such as company location, store hours, and account information that is extracted from a database or organization's documentation and converted to audible speech.
- Develop interactive educational material for children.
- Communicate directions, hands-free.
1.2.3 Text to Speech flow

The flow of the service is shown in Figure 1-7.

![Diagram](image)

Figure 1-7  Text to Speech flow

The figure shows the following flow:
1. The user inserts the text into the service call.
2. The text is sent to the Text to Speech service.
3. The Text to Speech service generates speech from the text.

1.2.4 Code snippets

Code examples of Watson Text to Speech are demonstrated in this section. In these examples, user credentials must be provided in order to access the API.

In these code examples, the Text to Speech service creates an audio file named HelloWatson.wav. The file contains a speech that is recorded in English and says “Hello from IBM Watson.” The generated audio file type is WAV.

- Node.js
- Java
- cURL
- Node-RED

The code structure shown in the Node.js and Java examples is common to several programming languages. The structure follows these steps:
1. Import the required Watson libraries.
2. Provide the credentials to authenticate with the service instance that you want to access.
3. Configure the service, specify the output format, languages, and other details that are specific for each API.
4. Display the API results.

Note: cURL and Node-RED are not programming languages and they follow a different structure. See the specific cURL and Node-RED examples.
Node.js

Example 1-4 shows a Node.js code snippet for Text to Speech.

**Example 1-4  Code snippet: Node.js**

```javascript
// Specify the references to required libraries
var TTS1 = require('watson-developer-cloud/text-to-speech/v1');
var fs = require('fs');

//Provide the credentials for using Watson Text to Speech API
var text_to_speech = new TTS1({
  username: '<enter your username here>',
  password: '<enter your password here>'
});

// Specify the input-text, output voice and the output audio file format
params = {
  text: 'Hello from IBM Watson',
  voice: 'en-US_AllisonVoice', // Optional voice
  accept: 'audio/wav'
};

// Pipe the synthesized text to a file
text_to_speech.synthesize(params).pipe(fs.createWriteStream('HelloWatson.wav'));
```

Java

Example 1-5 shows a Java code snippet for Text to Speech.

**Example 1-5  Code snippet: Java**

```java
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
//Specify the references to required libraries
import com.ibm.watson.developer_cloud.text_to_speech.v1.TextToSpeech;
import com.ibm.watson.developer_cloud.text_to_speech.v1.model.AudioFormat;
import com.ibm.watson.developer_cloud.text_to_speech.v1.model.Voice;
import com.ibm.watson.developer_cloud.text_to_speech.v1.util.WaveUtils;

class WatsonTTS {
  public static void main(String[] args) {
    try {
      // Specify the input-text, output voice and the output audio file
      String text = "Hello world";
      InputStream stream = service.synthesize(text, Voice.EN_ALLISON, AudioFormat.WAV).execute();

      // Pipe the synthesized text to a file
      InputStream in = WaveUtils.reWriteWaveHeader(stream);
      OutputStream out = new FileOutputStream("hello_world.wav");
      byte[] buffer = new byte[1024];
      int length;

      while ((length = in.read(buffer)) > 0) {
        out.write(buffer, 0, length);
      }
      out.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```
```java
in.close();
stream.close();
} catch (Exception e) {
    e.printStackTrace();
}
}
```

**cURL**

The cURL tool allows you to access the API directly from the command line. The following parameters are used in Example 1-6:

- `-u` Specifies the user name and password.
- `--output` Specifies the name of the file where the API response should be placed.
- `-X` Specifies the HTTP method for the request.
- `--header` Specifies the headers of the HTTP request.

**Example 1-6   Code snippet: cURL**

**EXAMPLE GET**

```bash
curl -X GET -u "<enter your username here>":"<enter your password here>" --output HelloWatson.wav
```

**EXAMPLE POST**

```bash
curl -X POST -u "<enter your username here>":"<enter your password here>" --header "Content-Type: application/json" --header "Accept: application/wav" --data "{"text":"Hello Watson"}" --output hello_world.wav
```

**Node-RED**

Figure 1-8 shows how to build a Text to Speech application using Node-RED.

![Node-RED Diagram](image-url)

**Figure 1-8   The five steps in Node-RED**
The steps in Node-RED are as follows:

1. Create an **inject** node and complete the fields in the edit node dialog (Figure 1-9).

   ![Inject node details](image1)

   *Figure 1-9  Inject node details*

2. Drag the **Text to Speech** node to the canvas and edit the node (Figure 1-10).

   ![Text To Speech Service node](image2)

   *Figure 1-10  Text To Speech Service node*
3. Add a **function** node to specify the headers (Figure 1-11). Header values are as follows:

```javascript
msg.headers={'Content-Type': 'audio/l16'; rate=44100;}; return msg;
```

![Figure 1-11 Specify the header in the function tab](image)

4. Use a **debug** node to specify the logs in the debug tab (Figure 1-12).

![Figure 1-12 Edit the debug node](image)
5. Use the file node to store the converted speech in the file (Figure 1-13).

![Edit file node](image)

Figure 1-13  File node to specify the speech filename

The File node: The default Node-RED distribution does not include the File node. You can add the node by modifying the package.json file. See the Adding Nodes topic at the Node-RED website.
Creating Bluemix services

IBM Bluemix® provides resources to your applications through a service instance. Before you can use the Watson services, you must create an instance of the corresponding service.

This chapter explains how to create the Bluemix services that are required for the use cases described in Chapter 3, “Personality Analyzer” on page 23 and Chapter 4, “Real-time transcription” on page 37.

This chapter also explains how to get the credentials to authenticate to your service instance from your app.

The following topics are covered in this chapter:

► Creating the Speech to Text service
► Creating the Text to Speech service
► Creating the Language Translator service
► Creating the Personality Insights service
► Retrieving the service credentials
2.1 Creating the Speech to Text service

To create the Speech to Text service, complete these steps:

1. Log in to IBM Bluemix.
2. Click Catalog.
3. Click Watson (under Services). The Watson services, available in Bluemix, are listed.
4. Click Speech to Text (Figure 2-1).

5. On the next web page (Figure 2-2), do these steps:
   a. Enter the service name or accept the default name provided.
   b. Select the pricing plan you want to use
   c. Click Create and wait for Bluemix to create an instance of your Speech to Text service.
2.2 Creating the Text to Speech service

To create the Text to Speech service, complete these steps:

1. Log in to IBM Bluemix.
2. Click Catalog.
3. Click Watson (under Services). The Watson services, available in Bluemix, are listed.
4. Click Text to Speech (Figure 2-3).

Figure 2-3    Watson services in Bluemix: Select Text to Speech

5. On the next web page (Figure 2-4), complete these steps:
   a. Enter the service name or accept the default name provided.
   b. Select the pricing plan you want to use.
   c. Click Create and wait for Bluemix to create an instance of your Text to Speech service.

Figure 2-4    Create Text to Speech service
2.3 Creating the Language Translator service

To create the Language Translator service, complete these steps:

1. Log in to IBM Bluemix.
2. Click Catalog.
3. Click Watson (under Services). The Watson services, available in Bluemix, are listed.
4. Click Language Translator (Figure 2-5).

![Figure 2-5 Watson services in Bluemix: Language Translator](image)

5. On the next web page (Figure 2-6), do these steps:
   a. Enter the service name or accept the default name provided.
   b. Select the pricing plan you want to use.
   c. Click Create and wait for Bluemix to create an instance of your Language Translator service.

![Figure 2-6 Create Language Translator service](image)
2.4 Creating the Personality Insights service

To create Personality Insights service, complete these steps:

1. Log in to IBM Bluemix.
2. Click Catalog.
3. Click Watson (under Services). The Watson services, available in Bluemix, are listed.
4. Click Personality Insights (Figure 2-7).

![Figure 2-7 Watson services in Bluemix: Personality Insights](image)

5. On the next web page (Figure 2-8), do these steps:
   a. Enter the service name or accept the default name provided.
   b. Select the pricing plan you want to use.
   c. Click Create and wait for Bluemix to create an instance of your Personality Insights service.

![Figure 2-8 Create Personality Insights service](image)
2.5 Retrieving the service credentials

To retrieve the service credentials, follow these steps:

1. On your Bluemix dashboard, scroll to the Services section.
2. Click the service for which you want to retrieve the credentials (Figure 2-9).

3. On the service landing page, click the Service Credentials tab (Figure 2-10).

4. Click View Credentials (Figure 2-11).
5. Copy the username and password (Figure 2-12). You will use these credentials to access the service from your application.

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

**Figure 2-12**  Service credentials: url, password, username

**Note:** The service credentials (service URL, username and password) are used in your application when you initialize the service. If the service credentials are not properly set, you will not be able to access the APIs. When using the Watson SDK, you will notice that the URL is already set in the library, so you are required to specify only the username and password.
This chapter introduces the use of speech recognition technology in an application. It guides you through the creation of a sample cognitive application called Personality Analyzer to analyze an individual’s personality.

The chapter describes how to use the Watson Speech to Text service in your application and apply the Watson Personality Insights service to extract personality characteristics based on the speech that an individual records through the application.

The approach demonstrated in the Personality Analyzer application can be used in practical use cases. For example, it can help an organization’s human resources department match the skills of an aspiring candidate with the applied job profile or help the department place an employee in a suitable job profile within the organization. It can also be helpful for clinical psychologists to assess the personality traits of a patient in order to provide applicable care.

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
3.1 Getting started

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

3.1.1 Objectives

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

- Recognize use cases for Watson Speech to Text and Watson Personality Insights services.
- Create a cognitive application integrating Watson Speech to Text and Watson Personality Insights services using the Node.js platform.

3.1.2 Prerequisites

To complete the steps in this chapter, you must have access to certain infrastructure and have several basic technology skills.

**Infrastructure**
Be sure you have the following items ready:

- A Bluemix account must be created.
- Bluemix services for Watson APIs must be created and credentials obtained for the following Watson services (see Chapter 2, “Creating Bluemix services” on page 15):
  - Watson Speech to Text
  - Watson Personality Insights
- A web browser, such as Firefox or Chrome.

**Technology**
Be sure you have the following basic technology skills:

- Fundamental use of Git and Cloud Foundry tools
- Basic programming language skills of Node.js, JavaScript, and HTML
- Basic concepts of Watson services:
  - Watson Speech to text
  - Watson Personality Insights

3.1.3 Expected results

Figure 3-1 on page 25 shows the user interface of the application. The user speaks to the application through the microphone by clicking **Record**. The application records the speech and converts the speech to text. The converted text is displayed in the text box. The recording is stopped by clicking **Stop**. The personality analysis is displayed by clicking **Analyze**.
Figure 3-1 User interface of Personality Analyzer
### 3.2 Architecture

The user provides input in the form of real-time text or audio. If the input is audio, the application converts the input audio to text by using the Watson Speech to Text service. The text is provided as input to the Watson Personality Insights service. The Personality Insights service extracts personality characteristics from the text and displays the personality information.

Figure 3-2 shows the components of the application and the flow of events:

1. The user invokes the microphone of the application and speaks about a person.
2. The application receives the audio as input.
3. The application calls the Watson Speech to Text API and converts the audio into text format as output. The API returns the text to the application.
4. The application calls the Watson Personality Insights API and passes the text as input. The API receives the text, processes it, and outputs the personality information about the person.
5. The application passes the Personality Insights output to the user through the browser.

![Figure 3-2  Architecture diagram and flow of events](image)

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

Two Git repositories are provided for this use case:

- **Step-by-step implementation (incomplete) version of the application.**

  This repository contains an incomplete version of the application and is used in all sections of 3.4, “Step-by-step implementation” on page 27. This version takes you through the key steps to integrate the IBM Watson APIs with the application logic.

- **Quick-deploy (complete) version of the application**

  This repository contains the final version of the application. If you want to bypass the implementation steps and instead run the application as a demonstration, download this full version. Downloading and running this full version demonstration is explained in 3.5, “Quick deployment of application” on page 35.
About the sample code
The sample code in this chapter is based on examples provided by German Attanasio Ruiz, IBM Watson Developer.

3.4 Step-by-step implementation

Implementing this use case involves the following steps:

1. Downloading the project from Git.
2. Reviewing the project structure.
3. Setting up Watson services in Bluemix.
4. Providing credentials for the Watson services.
5. Using the Watson Speech to Text (STT) service.
6. Using the Watson Personality Insights (PI) service.
7. Providing a unique name for your application.
8. Deploying the application on Bluemix.
9. Testing the application.

3.4.1 Downloading the project from Git

Clone or download the repository from the following location:

https://github.com/snippet-java/redbooks-tts-sst-201-pi-nodejs-student

Continue with the following sections to complete the development.

3.4.2 Reviewing the project structure

This section reviews the project structure shown in Figure 3.3.

Figure 3-3  Project structure folders and files
Some of the major folders and files are as follows:

- **manifest.yml**
  This file contains the details of the application, such as the domain name in which the application will be deployed (in this case, mybluemix.net). It also contains the unique name for the application, which is used in the URL to access the application. The memory and the disk quota that are allocated to the application are also specified here.

- **package.json**
  This file contains all the dependencies and engines, and defines the starter script for the application (Example 3-1).

  **Example 3-1  Snapshot of the package.json file that contains dependencies for the application**

  ```json
  "dependencies": {
    "body-parser": "^1.11.0",
    "cookie-parser": "^1.4.0",
    "csurf": "^1.8.3",
    "ejs": "^2.3.4",
    "express": "^4.13.3",
    "express-rate-limit": "^2.0.2",
    "express-secure-only": "^0.2.1",
    "helmet": "^0.13.0",
    "i18next": "^1.10.2",
    "morgan": "^1.6.1",
    "request": "^2.65.0",
    "vcap_services": "^0.2.0",
    "watson-developer-cloud": "^2.15.2",
    "watson-speech": "^0.30.1"
  },
  "engines": {
    "node": "^=0.10"
  },
  "scripts": {
    "start": "node app.js"
  }
  ```

- **config**
  This folder contains configuration settings for Watson services, security, and Express.js.

- **I18n**
  This folder contains the property files for various languages.

- **public**
  This folder contains stylesheets, images, Node.js files, and more:
  - `/public/js/watson-speech.js`
    This file contains the logic to record the audio using a microphone, stream the audio, convert audio into text, and display the results in the interface.
  - `/public/js/demo.js`
    This file contains the logic to display the personality traits in various forms as text, sunburst graph, and so on.

- **views**
  This folder contains the index.ejs file that you can modify if you want to take advantage of the user interface for your applications. See the next section (“Understanding the sample user interface” on page 29).
Understanding the sample user interface

The user interface (UI) that this application provides is intended as an example, and is not proposed as the user interface for your applications. However, if you want to use this interface, modify the following file:

- views/index.ejs

This is the landing page for the application. This file uses embedded JavaScript. You can change the UI by editing the file.

3.4.3 Setting up Watson services in Bluemix

This chapter uses two Watson services:

- Speech to Text
- Personality Insights

To use the Watson service, the service instances for the Watson APIs must be created. Follow the steps in Chapter 2, “Creating Bluemix services” on page 15.

3.4.4 Providing credentials for the Watson services

Next, provide credentials for the Watson services, which you created in 3.4.3, “Setting up Watson services in Bluemix” on page 29.

The credentials for the Watson services consist of three parts:

- URL
- Username
- Password

Those credentials will be stored in a JSON file, defined in the config folder. The code snippet shown in Example 3-2 provides the credentials of the Watson services. Insert this snippet into the /config/creds.json file:

Example 3-2  Code snippet: /config/creds.json (to provide the credentials of the Watson services)

```json
{
    "STT_URL" : "<STT URL>",
    "STT_username" : "<STT username>",
    "STT_password" : "<STT password>",
    "PI_URL" : "<PI URL>",
    "PI_username" : "<PI username>",
    "PI_password" : "<PI password>
}
```

3.4.5 Using the Watson Speech to Text (STT) service

In this application, when the user clicks the Record button on the UI, the STT service is invoked and the authorization is checked. If the STT authorization is valid, the recording starts, which calls the recognizeMicrophone() function in the views/index.ejs file. The recognizeMicrophone() function starts the microphone, records the audio, and buffers the audio. Then, it sends the audio buffer to the STT API for converting the input audio into text. The output text is displayed in the text box.
Example 3-3 shows a code snippet in the views/index.ejs file. It showcases the authorization check of Watson STT, invocation of recognizeMicrophone(), and the converted text being displayed in the UI. This code snippet is invoked from the Record button in the UI.

Example 3-3  Code snippet: index.ejs (to show authorization check and output text rendered in the UI)

```<script>
  document.querySelector('#usevoice').onclick = function () {
    fetch('/api/speech-to-text/token').then(function(response) {
      return response.text();
    }).then(function (token) {
      var stream = WatsonSpeech.SpeechToText.recognizeMicrophone({
        token: token,
      });
      stream.setEncoding('utf8'); // get text instead of Buffers for on data events
      stream.on('data', function(data) {
        console.log(data);
        document.querySelector('#text').value = document.querySelector('#text').value + data;
      });
      stream.on('error', function(err) {
        console.log(err);
      });
      document.querySelector('#stop').onclick = stream.stop.bind(stream);
    }).catch(function(error) {
      console.log(error);
    });
  }
</script>
```

Now you can set up and use the Watson Speech to Text API.

**Prerequisites**

Before you start the steps, be sure these prerequisites are met:

1. You set up the Speech to Text service as described in 3.4.3, “Setting up Watson services in Bluemix” on page 29.
2. You completed the steps as described in 3.4.4, “Providing credentials for the Watson services” on page 29.

**Steps**

Set up and use the Watson Speech to Text API:

1. Initialize the Watson Speech to Text API in the /config/servicedefs.js file. The credentials will be used to create the service wrapper. To initialize the Speech to Text API, insert the code snippet (Example 3-4) into the /config/servicedefs.js file.

   Example 3-4  Code snippet: /config/servicedefs.js (for initializing the Watson Speech to Text API)

```javascript
//create the service wrapper for Watson Speech to Text API
var speech_to_text = extend({
  version: 'v1',
  url: creds.STT_URL,
  username: process.env.STT_USERNAME || creds.STT_username,
  password: process.env.STT_PASSWORD || creds.STT_password,
}, vcapServices.getCredentials('speech_to_text'));
```
2. Export the speech_to_text object as a module so that it can be referenced in other files. To export the Speech to Text API, insert the code snippet (Example 3-5) into the /config/servicedefs.js file.

   **Example 3-5 Code snippet: config/servicedefs.js (to export speech-to-text variable)**
   ```javascript
   //Export the speech_to_text object to be referenced and used in other files.
   exports.STT=speech_to_text;
   ```

3. Set up an endpoint for the Speech to Text API by inserting the code snippet (Example 3-6) into the /stt-token.js file.

   **Example 3-6 Code snippet: /stt-token.js - (to set up an endpoint for the Speech to Text API)**
   ```javascript
   //require Watson Service credentials
   var creds=require('./config/creds.json');
   //Require the Watson Speech to Text API defined in servicedefs.js
   var sttVar = require('./config/servicedefs.js');
   // set up an endpoint to serve speech-to-text auth tokens
   var sttAuthService = watson.authorization(sttVar.STT);
   router.get('/token', function(req, res) {
     sttAuthService.getToken({url: creds.STT_URL}, function(err, token) {
       if (err) {
         console.log('Error retrieving token: ', err);
         res.status(500).send('Error retrieving token');
         return;
       }
       res.send(token);
     });
   });
   module.exports = router;
   ```

4. Invoke the Watson Speech to Text API.

   As soon as the application is started, the app.js file is invoked, which calls the stt-token.js file. To invoke Speech to Text API, insert the code snippet (Example 3-7) into the /app.js file.

   **Example 3-7 Code snippet: /app.js (to invoke the Speech to Text API)**
   ```javascript
   // Call the Watson Speech to Text API
   app.use('/api/speech-to-text/', require('./stt-token.js'));
   ```

### 3.4.6 Using the Watson Personality Insights (PI) service

This section describes how to integrate the Personality Insights service with the application. The application calls the Personality Insights API and passes the text that was generated by the Speech to Text API as input. The Personality Insights API receives the text, processes it to infer personality characteristics, and outputs the personality information.

Now, you can set up and use the Watson Personality Insights (PI) API.
Prerequisites
Before you start the exercise steps, be sure these prerequisites are met:

1. The Watson Personality Insights service is set up as described in 3.4.3, “Setting up Watson services in Bluemix” on page 29.
2. The credentials are provided as described in 3.4.4, “Providing credentials for the Watson services” on page 29.

Steps
Set up and use the Watson Personality Insights API by following these steps:

1. Initialize the Personality Insights API in the /config/servicedefs.js file. Insert the code snippet (Example 3-8) into the /config/servicedefs.js file.

   **Example 3-8  Code snippet: /config/servicedefs.js (to initialize the Personality Insights API)**

   ```
   // Create the service wrapper for Personality Insights API
   var personality_insights = watson.personality_insights(
       version: 'v2',
       username: creds.PI_username,
       password: creds.PI_password,
   );
   ```

2. Export the personality_insights object as a module so that it can be referenced in other files. Insert the code snippet (Example 3-9) into the /config/servicedefs.js file.

   **Example 3-9  Code snippet: /config/servicedefs.js (to export the personality_insights object)**

   ```
   //Export the personality_insights object to be referenced and used in other files.
   exports.PI=personality_insights;
   ```

3. Create a reference for Personality Insights and call the API. To initialize the Personality Insights API, insert the code snippet (Example 3-10) into the /app.js file.

   **Example 3-10  Code snippet: /app.js (to initialize the Personality Insights API)**

   ```
   // Watson Personality Insights service settings
   var pivar=require('./config/servicedefs.js');

   // Setting the profile for Watson Personality Insights.
   pivar.PI.profile(parameters, function(err, profile) {
       if (err)
           return next(err);
       else
           return res.json(profile);
   });
   ```
3.4.7 Providing a unique name for your application

Every application running on Bluemix needs a unique name. To specify a unique name for the application, edit the `manifest.yml` file as shown in Example 3-11.

*Example 3-11  Code snippet: manifest.yml (to provide a unique name for the application)*

```yaml
applications:
- path: .
  memory: 512M
  instances: 1
  domain: mybluemix.net
  name: <your application unique name>
  host: <your application unique name>
  disk_quota: 1024M
```

3.4.8 Deploying the application on Bluemix

Deploy the application as follows:

1. Use a command prompt to navigate to the project root.
2. Use the Cloud Foundry command-line interface (CLI) to deploy the app.
   - At the command prompt, run the following commands:
     - `cf login`
       Follow the instructions to specify the Bluemix credentials and space name.
     - `cf push "<your unique application name>"`
       This command builds all the files and the dependencies and deploys the application on Bluemix.

Congratulations! You successfully deployed the application.
### 3.4.9 Testing the application

After successfully deploying the application in Bluemix, test the application:

1. Open the application in any browser by using your application URL:
   
   \[ \text{http://<app_name>.mybluemix.net} \]

2. Click **Clear** to clear any existing text in the text box (Figure 3-4).

3. Click **Record** to start the voice recording.

4. A dialog might open, prompting you to share the microphone. Select the microphone you want to share and click **Share Selected Device**.

---

**Figure 3-4** Record audio and display insights

**Figure 3-5** Sharing the microphone
5. Start recording. The recorded text is displayed in the text box.
6. When you finish recording, click **Stop** to stop the recording.
7. Click **Analyze** (Figure 3-4 on page 34). The personality insights are displayed, as shown in Figure 3-1 on page 25.

### 3.5 Quick deployment of application

As described in 3.3, “Two ways to deploy the application: Step-by-step and quick deploy” on page 26, a Git repository that contains the full application code is provided so that you can run the application with minimal steps.

To run the application, follow these steps:

1. Access the Git repository and download the complete application code:
   
   https://github.com/snippet-java/redbooks-tts-sst-201-pi-nodejs

2. Create the Watson services on Bluemix (3.4.3, “Setting up Watson services in Bluemix” on page 29).

3. Set up the credentials for the Bluemix Watson services (3.4.4, “Providing credentials for the Watson services” on page 29).

4. Provide a unique name for your application (3.4.7, “Providing a unique name for your application” on page 33).

5. Deploy the application (3.4.8, “Deploying the application on Bluemix” on page 33).

6. Open a browser and paste the application URL:

   http://<application_name>.mybluemix.net

Congratulations! You are able to run the application.
3.6 References

See the following resources:

- Tutorial:
- Personality Insights:
- Build with Watson:
  https://www.ibm.com/watson/developercloud/
- CLI and Dev Tools, uploading your application:
  https://console.ng.bluemix.net/docs/starters/upload_app.html
- Service credentials for Watson services:
- Bluemix environment variables:
- Bluemix tokens for authentication:
This chapter introduces the Speech to Text and Text to Speech Watson APIs by developing a simple Android application that performs real-time transcription and translation. In addition to the Text To Speech and Speech to Text APIs, this chapter uses the Watson Language Translator API, Android Studio as the integrated development environment, Java as the development language, and other resources described later.

The approach demonstrated in the Real-time Transcription application can be used in practical use cases, as in these examples:

- A smart meeting room that listens to the speakers in their native language, translates their speech in real time, and displays the translation as text captions in the native language of the meeting attendees.
- A smart chat application that translates the speech of the chat partners and sends it as text to the receiver.
- A meeting transcriber for any specific language.
- A customer service app that receives questions from callers in their native language and responds with information, such as company location, store hours, and account information that is extracted from a database or organization’s documentation and is converted back to the caller’s language as audible speech.

The focus of this chapter is on the Watson Speech to Text and Text to Speech APIs, therefore the chapter does not provide details about Android development. The chapter covers how to optimize the use of the APIs and how you can benefit from their power.

Because you will work with partially developed code, this chapter has code snippets that you can use.

The following topics are covered in this chapter:

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

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

4.1.1 Objectives

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

► Recognize use cases for the Speech to Text and Text to Speech API.
► Configure the Watson SDK for Android projects.

4.1.2 Prerequisites

To complete the steps in this chapter, you must have access to certain infrastructure and have several basic technology skills.

**Infrastructure**

Be sure you have the following items ready:

► A Bluemix account.
► Bluemix services for Watson APIs must be created and credentials obtained for the following Watson APIs (see Chapter 2, “Creating Bluemix services” on page 15):
  – Speech to Text
  – Text to Speech
  – Language Translator
► Android Studio, installed.

**Additional resources**

The application developed in this chapter requires a microphone so you must provide an Android phone in order to have a smoother experience while developing the app.

**Technology**

Be sure you have the following basic technology skills:

► Fundamental use of Git and Cloud Foundry tools
► Basic Java programming skills
► Basic concepts of Watson APIs:
  – Speech to text API
  – Text to Speech API
► Android Studio administration
► Bluemix administration
4.1.3 Expected results

Figure 4-1 shows the user interface of the app. As the user speaks, the app provides automatic English speech transcription and the speech is also translated and played in a target language.

![Real Time Transcription app user interface](image.png)
4.2 Architecture

This section provides an overview of the architecture of the application.

The Speech to Text service is used to convert voice into text. Then, the text is sent as input to the Language Translator service, which translates the text. Finally, the translated text response is used as input to the Text to Speech service to generate the translated output.

Figure 4-2 shows the components of the application architecture:

1. Speaking in English, the user’s voice can be captured by the microphone.
2. The voice stream is parsed by the application logic according to the settings specified in the application.
3. The voice stream is sent to the Speech to Text service and a response in text is received by the application.
4. The text generated by the Speech to Text is then used as input for the Language Translator service, which will translate the text according to the language specified.
5. The translated text is used in the Text to Speech service, which converts it back into speech according to the selected target language and sends it back to the application.
6. The converted speech is then played by the phone player to the user.
4.3 Two ways to deploy the app: Step-by-step and quick deploy

Two Git repositories are provided for this use case:

- **Step-by-step implementation (incomplete) version of the application.**
  
  This repository contains an incomplete version of the application and is used in all sections of 4.4, “Step-by-step implementation” on page 41. This version takes you through the key steps to integrate the IBM Watson APIs with the application logic.

- **Quick-deploy (complete) version of the application**
  
  This repository contains the final version of the application. If you want to bypass the implementation steps and instead run the application as a demonstration, download this full version. Downloading and running this full version demonstration is explained in 4.5, “Quick deployment of application” on page 52.

4.4 Step-by-step implementation

Implementing this use case involves the following steps:

1. Preparing the mobile device where the app will run.
2. Downloading the project from the Git repository.
3. Importing the project to Android Studio.
4. Reviewing the project structure and the code provided.
5. Importing the required Watson libraries.
6. Using the Speech to Text (STT) service.
7. Using the Text to Speech (TTS) service.
8. Setting up the credentials.
9. Testing the application.

4.4.1 Preparing the mobile device where the app will run

To access development functions on your device, enable the USB debugging mode on your device. By default, this mode is disabled to avoid unwanted modifications on your device.

On Android 4.2 or later, complete these steps:

1. Select **Settings → General → About phone → Software info**.
2. Tap **Build number** seven times until a message displays on your screen.
3. Return to the previous screen and you see the **Developer Options**.
4. Tap on **Developer Options** and enable the **USB Debugging** mode.

**Note:** This activation sequence might differ depending on your Android version. If that is the case, check Android forums and discover how to enable the USB debugging mode on your device. The steps should be similar.
4.4.2 Downloading the project from the Git repository

The following repository is used for the step-by-step implementation version of deployment. Although the code is basically the same as shown in the full quick-deploy version, it does not contain some important parts (that you will add) and includes a description to help you with the development.

Download the step-by-step Git repository from the following location:

4.4.3 Importing the project to Android Studio

After you download the project and import it to Android Studio, complete these steps:
1. Open the project in the Android Studio by selecting File → Open.
2. Select the path where you downloaded the project and then click OK (Figure 4-3).

![Open File or Project](image)

*Figure 4-3 Open File or Project*

Android Studio will configure the files so the project can run.
4.4.4 Reviewing the project structure and the code provided

After you import the project in Android Studio, you see the project structure (Figure 4-4).

![Project structure in Android Studio](image)

The implementation uses the `MainActivity.java` class. This class is an incomplete version so you are required to provide the missing code snippets. To help you understand, the `MainActivity.java` file (Example 4-1) is described next. Refer to this description if you have future problems with the code structure.

**Example 4-1 MainActivity.java structure**

```java
// UI Components
private TextView transcription;
private Spinner targetLanguage;
private Language selectedTargetLanguage;
private ImageButton micButton;
private Boolean recording;

// Peripherals
private MicrophoneInputStream capture;
private StreamPlayer player = new StreamPlayer();

// Services
private SpeechToText speechService;
private TextToSpeech textService;
private LanguageTranslator translationService;

// Credentials file
Properties properties = null;

@Override
protected void onCreate(Bundle savedInstanceState) {...}

private TextToSpeech initTextToSpeechService(){...}

private void initRecognition() {...}

private LanguageTranslator initLanguageTranslatorService() {...}

private SpeechToText initSpeechToTextService() {...}

private RecognizeOptions getRecognizeOptions() {...}

private String getProperty(String key) {...}
```
private class MicrophoneCallback implements RecognizeCallback {...}

private class TranslateAndSynthesizeTask extends AsyncTask<Object, Void, Void> {...}

The MainActivity.java file structure contains the following information:

- **Private attributes**
  - The private attributes are divided into four groups:
    - UI Components: Represents objects that the app needs in order to draw on the graphical interface.
    - Peripherals: Objects related to the device itself, in this case a media player and a microphone.
    - Services: Represents the objects that you use to interact with Watson services.
    - Credentials file: Holds your credentials, retrieved from the config.properties file.

- **onCreate()**
  - Is responsible for initializing objects once the activity is being started.

- **initTextToSpeechService(), initSpeechToTextService(), initLanguageTranslatorService()**
  - These methods are responsible for creating the Watson connection instances.

- **initRecognition()**
  - Starts the microphone capturing process and send it to the Speech to Text service.

- **getProperty()**
  - This auxiliary method accesses the config.properties file and retrieves user credentials.

- **getRecognizeOptions()**
  - Creates an instance of the RecognizeOptions class, which describes how the speech recognition should behave.

- **MicrophoneCallback**
  - This callback class is responsible for handling the speech results from the STT service.

- **TranslateAndSynthesizeTask**
  - This asynchronous task object performs the translation and audio synthesis.
The chart in Figure 4-5 describes the code flow of the MainActivity.java file, starting at number 1 and finishing at number 16.

![Figure 4-5  MainActivity.java code flow](image)

**4.4.5 Importing the required Watson libraries**

To use the Watson libraries, you must add them as dependencies of your project. By default, Android Studio creates the projects by using Gradle, which is an open source build automation tool that helps you manage all your dependencies. To add those dependencies so that you can use the libraries in your project, complete these steps:

1. Under **Gradle Scripts**, select **build.gradle** (Figure 4-6).

![Gradle Scripts](image)
2. Go to the dependencies configuration section, which is similar to Figure 4-7.

```java
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
        exclude group: 'com.android.support', module: 'support-annotations'
    })
    compile 'com.android.support:appcompat-v7:25.1.0',
    'com.ibm.watson.developer_cloud:java-sdk:3.5.2',
    'com.ibm.watson.developer_cloud:android-sdk:0.2.1'
}
```

*Figure 4-7  The dependencies configuration in build.gradle*

3. Note the addition of two dependencies; be sure that they are present in your project, otherwise the app will not run:

   - `com.ibm.watson.developer_cloud:java-sdk:3.5.2`
     To use Watson services, this dependency is required.
   - `com.ibm.watson.developer_cloud:android-sdk:0.2.1`
     To easily access resources from the Android phone, such as the microphone or the audio player, this dependency is required.

4. If you have any trouble with the library versions, change the last part of the dependency, which is the version number, to the version of your preference. For example, the `java-sdk` version is 3.5.2 and the `android-sdk` version is 0.2.1.

   **Hint:** To show line numbers in Android Studio, press Ctrl+Shift+A and search for this text:
   `show line numbers`

### 4.4.6 Using the Speech to Text (STT) service

You can now develop the code that performs speech recognition:

1. On the `onCreate()` method, initialize the `speechService` variable that was previously declared as a class attribute:

   ```java
   speechService = initSpeechToTextService();
   ```

   The `onCreate()` method executes as soon as the application starts. Therefore, the `speechService` will be initiated as soon as the application boots.

   The same method has a call to `initRecognition()`. This method starts the speech recognition every time it is called and it needs your attention.

2. Open the `initRecognition()` method (Figure 4-8).

   ```java
   private void initRecognition() {
       capture = new MicrophoneInputStream(true);
       new Thread((Runnable) () -> {
           try {
               // Call STT service here
           } catch (Exception e) {
               e.printStackTrace();
           }
       }).start();
   }
   ```

*Figure 4-8  The initRecognition method*
Chapter 4. Real-time transcription

The figure shows that the microphone instance is being created and it is also created as a new Thread. This thread will "listen" to the user speaking. It is required because running network operations from the UI thread is not recommended.

Pay attention in the try block statement. The thread is empty so it will perform no activity if started. You must call the STT service, as the comment states (//Call STT service here).

3. Call the recognizeUsingWebSocket method from speechService. Insert the code snippet (Example 4-2) under the comment //Call STT service here (Figure 4-8 on page 46).

Example 4-2   Invoking WebSocket operation on STT

```java
speechService.recognizeUsingWebSocket(
    capture,
    getRecognizeOptions(),
    new MicrophoneCallback()
);
```

This method receives three parameters:
- The source input for the service (in this case, the capture object that represents the microphone)
- A group of settings for the service (related to the type of audio file, language, keywords, so on)
- A callback class that is responsible for handling the text received from the API

The getRecognizedOptions() method returns an instance of the class RecognizeOptions. This class is an abstraction that sets the recognition parameters for the STT service.

4. Check the available options for the getRecognizedOptions method (Figure 4-9). It can be useful to you for future applications.

```
private RecognizeOptions getRecognizeOptions() {
    return new RecognizeOptions.Builder()
        .continuous(true)
        .contentType(ContentType.OPUS.toString())
        .model("en-US_BroadbandModel")
        .interimResults(true)
        .inactivityTimeout(2000)
        .build();
}
```

Figure 4-9   Configuring STT service with the getRecognizeOptions method

As Figure 4-9 shows, you can specify the audio type, the language, and many other settings. For instance, you can change the model to "es-ES_BroadbandModel" to transcribe from Spanish instead of English. This way, you can change the default speaking language to another language of your preference.

For more information, see the Recognize audio topic for the Speech to Text API.

5. The last part of the STT service setup is the MicrophoneCallback callback class. As mentioned previously, this class defines what you should do with the text received from the API after the transcription finishes.
Open the onTranscription method on class MicrophoneCallback (Figure 4-10).

```java
public void onTranscription(SpeechResults speechResults) {
    // Retrieve the text, display to the user and
    // call the LT and TTS services
}
```

Figure 4-10 The onTranscription() method that requires completion

The onTranscription method is called every time a speech result is available. The result is available through the SpeechResults class provided by the Watson SDK. Save the result so it can be displayed in the user interface and use it as input for the other APIs.

**Hint:** You can always press Ctrl+Space to activate the auto-completion feature in Android Studio.

6. Complete the onTranscription method according to the code snippet in Example 4-3.

   Remember that the speechResults.isFinal method is the piece of code responsible for detecting when the user stops speaking. This is a useful option, provided by the Speech to Text service, to save you time from filtering audio that is recorded by the device.

**Example 4-3 The onTranscription method already completed**

```java
public void onTranscription(SpeechResults speechResults) {
    if (!speechResults.getResults().isEmpty()) {
        final String recognizedText =
            speechResults.getResults().get(0).getAlternatives().get(0).getTranscript();

        runOnUiThread(new Runnable() {
            @Override
            public void run() {
                transcription.setText(recognizedText);
            }
        });
    }

    if (speechResults.isFinal()) {
        try {
            capture.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    String selected = (String) targetLanguage.getSelectedItem();
    switch (selected) {
    case "Spanish":
        selectedTargetLanguage = Language.SPANISH;
        break;
    case "Portuguese":
        selectedTargetLanguage = Language.PORTUGUESE;
        break;
    case "Italian":
        selectedTargetLanguage = Language.ITALIAN;
        break;
    case "French":
        selectedTargetLanguage = Language.FRENCH;
    }
```
Although this step concludes the Speech to Text service setup, and because the Speech to Text work is integrated with the other APIs, if you try to run the application now it will not work. Continue to the next section and discover how to properly configure the Text to Speech service.

4.4.7 Using the Text to Speech (TTS) service

To prepare and call the Text to Speech service, complete these steps:

1. Initialize the Text to Speech service.
   
   The same way you initialized the Speech to Text service in step 1 on page 46, you must also initialize the Text to Speech service. In the onCreate() method, initialize the Text to Speech service as follows:
   
   ```java
   textService = initTextTospeechService();
   ```

2. Implement what the app should do after the user stops recording. The app should be capable of getting the text from the STT service and sending it to the Language Translator and the TTS services. Open the TranslateAndSynthesizeTask class, which currently is incomplete (Figure 4-11 on page 50).
Figure 4-11 The TranslateAndSynthesizeTask class is incomplete

This class inherits from AsyncTask (Asynchronous Task). After you start this task, it executes a simple task in a different thread and terminates.

As Figure 4-11 shows, the variables you need to use are already set, so you have three jobs:

- Call the TTS service.
- Play the audio to the user.
- Restart the speech recognition again.

3. To call the TTS service, select the respective voice according to the language that the user selected. Add the code snippet to select the voice according to the target language, as Example 4-4 shows.

Example 4-4 Setting up voice selection according to the language

```java
Voice respectiveVoice = null;
switch (selectedTargetLanguage) {
    case SPANISH:
        respectiveVoice = Voice.ES_ENRIQUE;
        break;
    case PORTUGUESE:
        respectiveVoice = Voice.PT_ISABELA;
        break;
    case ITALIAN:
        respectiveVoice = Voice.IT_FRANCESCA;
        break;
```
case FRENCH:
    respectiveVoice = Voice.FR_RENEE;
}

4. Call the TTS service that you already instantiated in the previous sections, as Example 4-5 shows.

**Example 4-5 Calling the TTS service**

```java
InputStream audio;
try {
    audio = textService.synthesize(
        translatedText, respectiveVoice).execute();
} catch (UnauthorizedException e) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(
                MainActivity.this,
                "Invalid credentials for TTS service.",
                Toast.LENGTH_SHORT
            ).show();
        }
    });
    return null;
}

runOnUiThread(new Runnable() {
    @Override
    public void run() {
        Toast.makeText(
            MainActivity.this,
            "Playing Translated Speech",
            Toast.LENGTH_SHORT
        ).show();
    }
});
player.playStream(audio);
```

Do not forget to play the audio file with the player on the last line.

5. In the TranslateAndSynthesizeTask, on the onPostExecute() method, add the call that is shown in Example 4-6. This step ensures that as soon as the AsyncTask ends, the app will start listening to the user again.

**Example 4-6 Restart the voice recognition after the audio was played to the user**

```java
@Override
protected void onPostExecute(Void aVoid) {
    initRecognition();
}
```

**Hint:** To learn more about the Language Translator API, see *Building Cognitive Applications with IBM Watson APIs: Volume 5 Language Translator*, SG24-8392.
4.4.8 Setting up the credentials

To access the services, you must add the user credentials. This is the final step before testing the app. Complete these steps:
1. Open the config.properties file. The contents should be similar to Figure 4-12.

![Figure 4-12 The config.properties file](image)

2. Replace the stt-user, stt-pw, tts-user, tts-pw, lt-user, and lt-pw variables, with your user credentials.

As you might recall, these credentials are in the STT service on Bluemix. See Chapter 2, “Creating Bluemix services” on page 15 for more information.

As a reminder, these credentials are not your Bluemix account credentials. They are the service credentials specific for each service instance.

4.4.9 Testing the application

Congratulations, the app is now complete and ready to test. Complete these steps:
1. Connect your phone with a USB cable to your computer.
2. Click Run (or press Shift+F10) in the Android Studio (Figure 4-13).

![Figure 4-13 Click the Run app button in Android Studio](image)

3. A pop-up window opens. It lists the available devices to deploy the application. Select your phone, which should now be in the list.

Because this is the first time you are running the app, Android Studio might require that you download some files in order to deploy the app. Let it download and finish deploying your app. After it finishes, the application will automatically start on your device.

4.5 Quick deployment of application

A second Git repository is provided so that you can run the real time transcription application even if you did not follow the steps in 4.4, “Step-by-step implementation” on page 41.
This quick deployment contains instructions to run the app without going through all the steps in the previous sections, although it does refer to some of those steps:

1. Access the repository and download the project:
   
   https://github.com/snippet-java/redbooks-tts-sst-301-rtt-android-java

2. Open the project in the Android Studio by selecting **File → Open**.

3. Select the path where you downloaded the project and then click **OK** (Figure 4-14).

   ![Figure 4-14: Open File or Project](image)

   Android Studio configures the files so the project can run. When it finishes processing, set up the credentials to run the app, otherwise you receive an **Unauthorized Access** message.

4. Insert user service credentials in the **config.properties** file, as explained in 4.4.8, “Setting up the credentials” on page 52.

5. Connect your phone with a USB cable to your computer.

6. Click **Run** (Figure 4-15).

   ![Figure 4-15: Click the Run app button in Android Studio](image)

7. In the pop-up window, select the device to deploy the app and confirm the selection.

At this point, Android Studio builds and deploys the app.
4.6 References

See the following resources:

- **Speech to Text tutorial:**

- **Text to Speech tutorial:**

- **Android SDK to use the IBM Watson services:**
  https://github.com/watson-developer-cloud/android-sdk

- **Build with Watson:**
  https://www.ibm.com/watson/developercloud/
Additional material

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

Locating the Git repositories

The following Git repositories are available to help you with the examples in these chapters:

- Chapter 3, “Personality Analyzer” on page 23:
  - https://github.com/snippet-java/redbooks-tts-sst-201-pi-nodejs-student
  - https://github.com/snippet-java/redbooks-tts-sst-201-pi-nodejs

- Chapter 4, “Real-time transcription” on page 37:
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 APIs 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 Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- Adding Nodes topic on the Node-RED website:
  https://nodedoc.com/docs/getting-started/adding-nodes
- Speech to Text video:
  https://youtu.be/JWnLgZ58zsw
- Speech to Text API documentation:
  https://www.ibm.com/watson/developercloud/speech-to-text/api/v1/?curl#recognize
  _sessionless_nonmp12
- IBM Bluemix:
  https://console.ng.bluemix.net/
- Cloud Foundry command-line interface (CLI):
  https://console.ng.bluemix.net/docs/cli/index.html
- Recognize audio topic for the Speech to Text API:
  https://www.ibm.com/watson/developercloud/speech-to-text/api/v1/#recognize_audi
  o_websockets
Also see the list of online resources for the following chapters in this book:

- Personality Analyzer: 3.6, “References” on page 36
- Real-time transcription: 4.6, “References” on page 54

**Help from IBM**

IBM Support and downloads

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

IBM Global Services

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