IBM Image-Analysis– Node.js
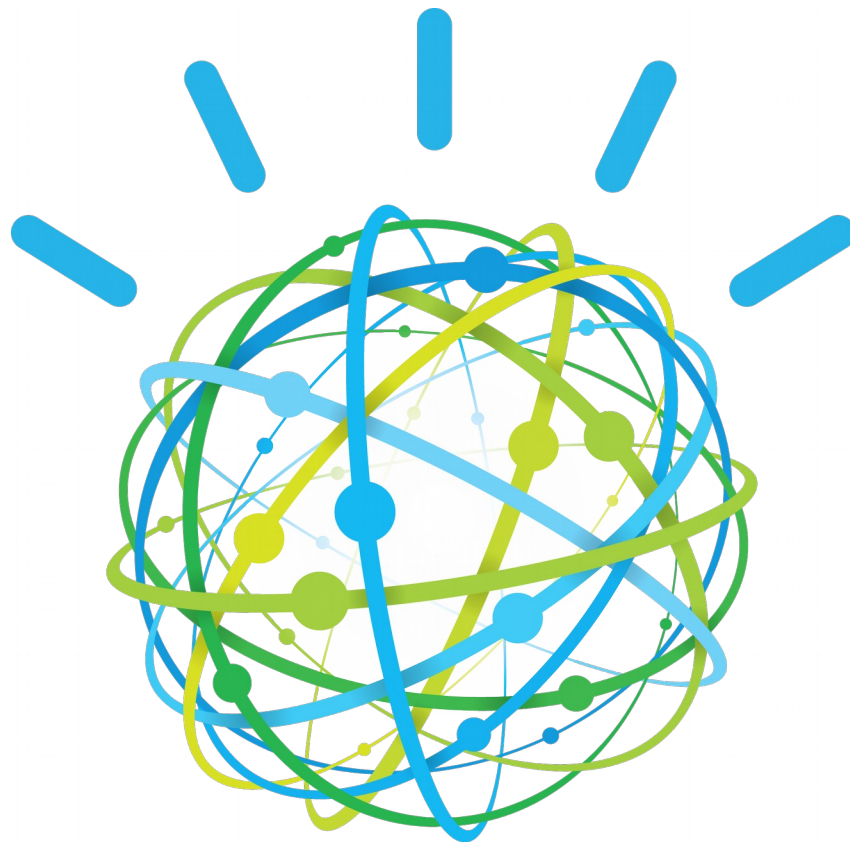
Cognitive Solutions Application Development

IBM Global Business Partners
Duration: 90 minutes
Updated: Apr 11, 2019
Klaus-Peter Schlotter
kps@de.ibm.com

Version 1.5

Overview

The IBM Watson Developer Cloud (WDC) offers a variety of services for developing cognitive applications. Each Watson service provides a Representational State Transfer (REST) Application Programming Interface (API) for interacting with the service. Some services, such as the Speech to Text service, provide additional interfaces.

The Watson Image Analysis application will use several services (Visual Recognition, Text-to-Speech, Language Translator) to show how they can be combined to achieve a desired goal.

This app will be created and run from a local workstation.

- In a first step, the application provides the ability to classify an image and get a descriptive name back in English (Visual Recognition).

- In a second step, this name is spoken thanks to the Text-to-Speech service.

- The following step is to translate the description in German language with the Language Translator Service and have it pronounced by the Text-To-Speech service in that language.

- Finally, we deploy the Node.js application to IBM Cloud.

Objectives

- Learn how to use the Cloud Foundry command-line interface to create and manage Watson services

- Learn how to implement the Image-Analysis application using Node.js

- Learn to orchestrate 3 services in a single application

- Learn how to create a Node.js application running in IBM Cloud

Prerequisites

Before you start the exercises in this guide, you will need to complete the following prerequisite tasks:

- Guide – Getting Started with IBM Watson APIs & SDKs

- Create an IBM Cloud account

You need to have a workstation with the following programs installed:

1. Node.js

2. Npm

3. Git → only for optional **Step 17 a)**

> **Note:** Copy and Paste from this PDF document does not always produce the desired result, therefore open the code snipptes for this lab in the browser and copy from there!

Section 1: Testing Watson APIs with a REST Client
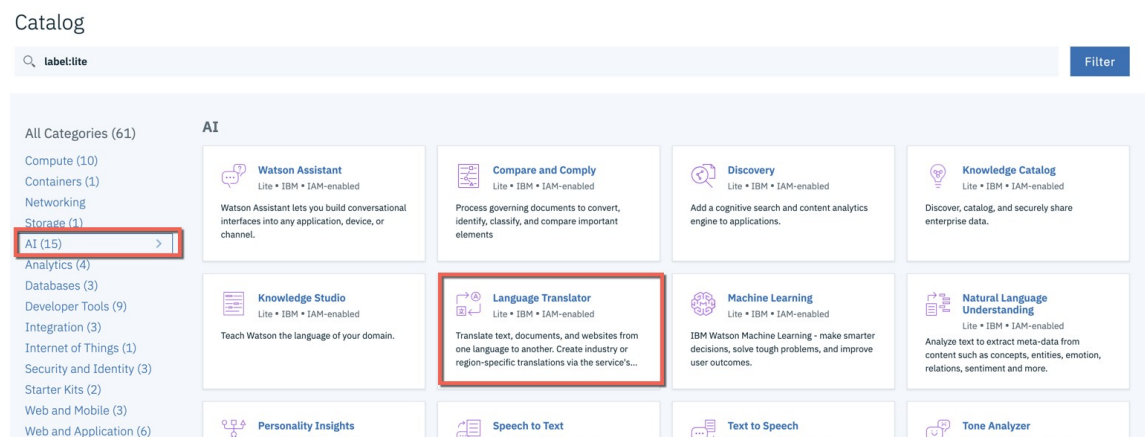
Create services in IBM Cloud

Step 1 Create a Language Translation service (now in the IBM Cloud console)
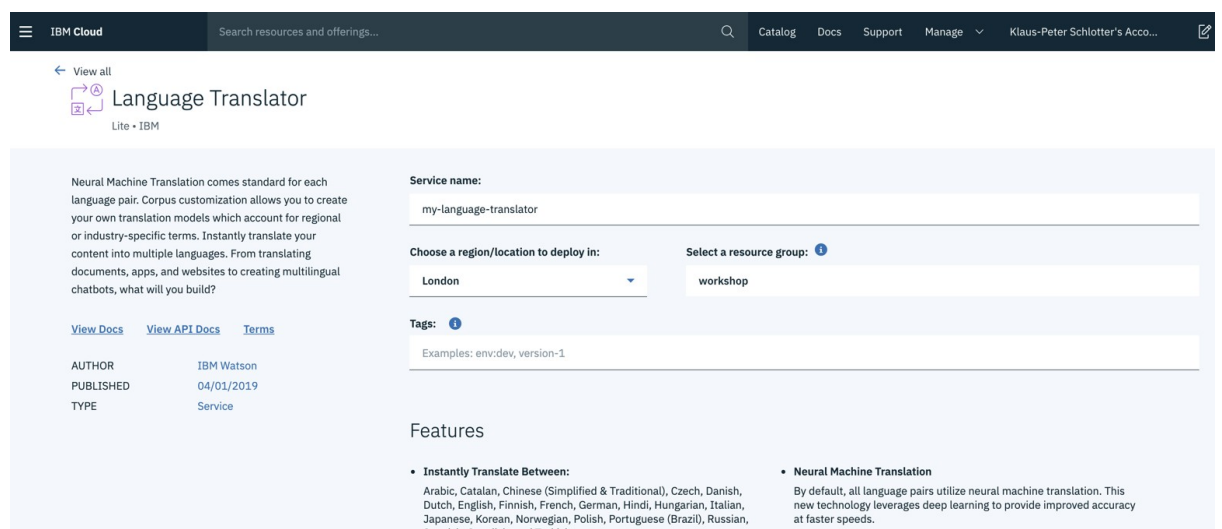
a) Open a Browser and navigate to https://cloud.ibm.com

You should see your dashboard with the services created above.

b) Click the [Create resource] in the top right.

c) In the left menu **select** *AI* and **click** the *Language Translator* service.

d) As *Service name* specify **my-language-translator**

e) Click the [Create] button

LAB 01

f) After the service is created you will see the **Manage** page of the service with the getting started infos and the credentials. These credentials you need in a later tutorial step.

Step 2 Repeat Step 1 and create

a) *Visual Recognition* service named **my-visual-recognition**

b) *Text to Speech* service named **my-text-to-speech** at an appropriate location.

LAB 01

Work with your services using Postman (REST Client)

The Watson Services in Bluemix have a REST (Representational State Transfer) interface and therefore they can be easily tested with a REST client like Postman.

You can find the API reference in the Documentation of each service.

Step 3 Postman is available as an application for MacOS, Windows, Linux https://www.getpostman.com.
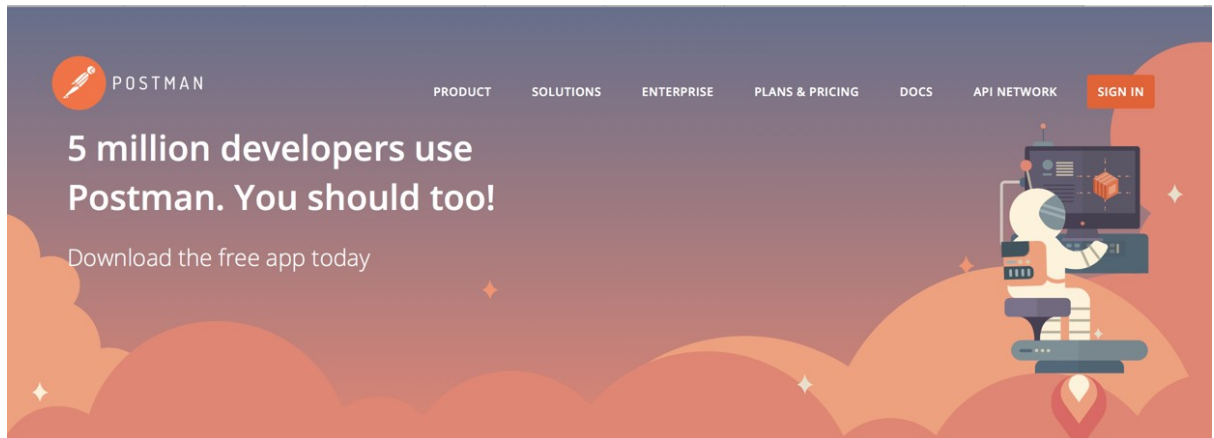
Step 4 Download and install the application and then open it.

Test Visual Recognition

Note: You may Copy and Paste from the [code snipptes](#) file.

Step 5 Open this URL and save a copy of the image to your local hard drive

https://upload.wikimedia.org/wikipedia/commons/thumb/5/50/Queen_Elizabeth_II_March_2015.jpg/455px-Queen_Elizabeth_II_March_2015.jpg

Step 6 In Postman select **GET** as the HTTP request type to use

Build the URL for testing with the Visual Recognition service

a) Get the *url* and the *apikey* from within your **my-visual-recognition** service in the IBM Cloud console.

b) In the *Authorization* section of Postman **enter** *apikey* as the username and apikey from your service as password.

Note: The services in the IBM Cloud platform are currently transferred to an IAM authentication. So if your service was created with an **apikey** instead of **userid** and **password**, you specify the **string apikey** as userid and the **generated apikey** as password.

c) Select GET as request type and enter the URL from your service's credentials. To this url append */v3/classify* as the method we want to execute.

d) Click the [Params] button and enter the following parameters:

- ○ url
  https://upload.wikimedia.org/wikipedia/commons/thumb/5/50/
  Queen_Elizabeth_II_March_2015.jpg/455px-Queen_Elizabeth_II_March_2015.jpg

- ○ version
  2018-03-19

	Key	Value	Description
☑	url	https://upload.wikimedia.org/wikipedia/commons/thumb…	
☑	version	2018-03-19	

e) Depending on your browser's default language (f.e. when it is German) you have to specify the following header variable:

key: `Accepted-Language`   value: `en`

Otherwise you may get a message "*unsupported language de*"

f) Press the **Send** button and see the result

```json
{
    "images": [
        {
            "classifiers": [
                {
                    "classifier_id": "default",
                    "name": "default",
                    "classes": [
                        {
                            "class": "queen",
                            "score": 0.908,
                            "type_hierarchy": "/person/queen"
                        },
                        {
                            "class": "person",
                            "score": 0.976
                        },
                        {
                            "class": "Queen of England",
                            "score": 0.706,
                            "type_hierarchy": "/person/Queen of England"
                        },
                        {
                            "class": "queen mother",
                            "score": 0.5,
                            "type_hierarchy": "/person/queen mother"
                        },
                        {
                            "class": "headdress",
                            "score": 0.788
                        },
                        {
                            "class": "Tyrian purple color",
                            "score": 0.81
                        },
                        {
                            "class": "fuschia color",
                            "score": 0.788
                        }
                    ]
                }
            ],
            "source_url": "https://upload.wikimedia.org/wikipedia/commons/thumb/5/50/Queen_Elizabeth_II_March_2015.jpg/455px
                -Queen_Elizabeth_II_March_2015.jpg",
            "resolved_url": "https://upload.wikimedia.org/wikipedia/commons/thumb/5/50/Queen_Elizabeth_II_March_2015.jpg/455px
                -Queen_Elizabeth_II_March_2015.jpg"
        }
    ],
    "images_processed": 1,
    "custom_classes": 0
}
```

Status: 200 OK   Time: 995 ms   Size: 944 B

Step 7 Save the Postman request definition

In Postman, beside the *Send* button there is a *Save* button with a *down arrow*. Click the **down arrow** and select **Save As..** and enter something like

SAVE REQUEST   ✕

Request Name

VisualRecognition_GET_classify

Request description (Optional)

Classify the image with Watson Visual Recognition

Descriptions support Markdown

Save to existing collection / folder

Type to filter   ⌄

Or create new collection

WatsonServices

the following to save the request for future use.

Step 8 Test Visual Recognition to detect faces.
In the Postman request url just change the **classify** to **detect_faces** and submit again. The following result appears.

```
Pretty    Raw    Preview    JSON ▼    ⇥

1  ▾ {
2  ▾     "images": [
3  ▾         {
4  ▾             "faces": [
5  ▾                 {
6  ▾                     "age": {
7                           "min": 72,
8                           "max": 76,
9                           "score": 0.7684922
10                      },
11 ▾                     "face_location": {
12                          "height": 207,
13                          "width": 200,
14                          "left": 141,
15                          "top": 164
16                      },
17 ▾                     "gender": {
18                          "gender": "FEMALE",
19                          "score": 0.9959388
20                      }
21                  }
22              ],
23              "source_url": "https://upload.wikimedia.org/wikipedia/commons/thumb/5/50/Queen_Elizabeth_II_March_2015.jpg/455px
                   -Queen_Elizabeth_II_March_2015.jpg",
24              "resolved_url": "https://upload.wikimedia.org/wikipedia/commons/thumb/5/50/Queen_Elizabeth_II_March_2015.jpg/455px
                   -Queen_Elizabeth_II_March_2015.jpg"
25          }
26      ],
27      "images_processed": 1
28 }
```

Step 9 Test Visual Recognition with an Image from your local drive

    a) Decide whether you want to **classify** or **detect_faces** the image.

    b) **Change** the HTTP method to **POST**

    c) In Postman **remove** the url parameter from the request.

```
url=https://upload.wikimedia.org/wikipedia/commons/thumb/5/50/Queen_Eliza-
beth_II_March_2015.jpg/455px-Queen_Elizabeth_II_March_2015.jpg
```

Step 10 In the Body section select **form-data** and add the following parameters:

    key: size   value: original

    key: file   value: *Choose the file* you have saved in **Step 8**

POST ∨	https://gateway-a.watsonplatform.net/visual-recognition/api/...	Params	**Send** ∨	Save ∨

Authorization   Headers   **Body** ●   Pre-request Script   Tests   Code

● form-data   ○ x-www-form-urlencoded   ○ raw   ○ binary

Key	Value	Bulk Edit
☑ size	original	
☑ file	Choose Files Queen_Elizabeth_II_March_2015.jpg	
New key	value	

The result should be the same as in the previous step.
Optional: You can now test with other images.

Test the Language Translation service

You can find the description of the API in the [API Reference](#).

Step 11 In Postman specify the following parameter for the **POST translate** request.

a) Url (region specific) and apikey from your service definition and add the method
/v3/translate

```
f.e. https://gateway.watsonplatform.net/language-translator/
api/v3/translate
```

b) add the version parameter to the url in the Params section

c) This service uses basic authentication. In the *Authentication* section enter *apikey* as username and the apikey for your service definition as password.

Note: The services in the IBM Cloud platform are currently transferred to an IAM authentication. So if your service was created with an **apikey** instead of **userid** and **password**, you specify the **string apikey** as userid and the **generated apikey** as password.

d) In the *Headers* section specify the following parameters:

- Key: **Content-Type**　　　Value: **application/json**

- Key: **Accept:**　　　　　Value: **application/json**

POST ∨	https://gateway.watsonplatform.net/language-translator/api/v2/translate	Params

Authorization ● 　　Headers (2)　　Body ●　　Pre-request Script　　Tests

	Key	Value
☑	Content-Type	application/json
☑	Accept	application/json

e) In the *Body* section, type *raw*, specify the following JSON data

```
{
    "text":"Hello, how are you!",
    "source":"en",
    "target":"de"
}
```

POST ∨	https://gateway.watsonplatform.net/language-translator/api/v2/translate	Params

Authorization ●　　Headers (2)　　Body ●　　Pre-request Script　　Tests

○ form-data　　○ x-www-form-urlencoded　　● raw　　○ binary　　JSON (application/json) ∨

```
1 ▾ {
2        "text":"Hello, how are you?",
3        "source":"en",
4        "target":"de"
5   }
```

f) Press the **Send** button and check the result.

Pretty　　Raw　　Preview　　JSON ∨　　⇄

```
1 ▾ {
2 ▾    "translations": [
3 ▾      {
4           "translation": "Hallo, wie geht es Ihnen?"
5        }
6      ],
7      "word_count": 4,
8      "character_count": 19
9   }
```

　　LAB 01

Step 12 In Postman specify the following parameter for the
GET Identifiable languages request.

    a) Select **GET** as the Postman HTTP method.

    b) In the Url section change **translate** to `identifiable_languages`

    c) Keep your *Authorization* section and *Params* from previous step.

    d) For a **GET** request the *Body* section is ignored.

> **Note:** The services in the IBM Cloud platform are currently transferred to an IAM authentication. So if your service was created with an **apikey** instead of **userid** and **password**, you specify the **string apikey** as userid and the **generated apikey** as password.

GET ∨	https://gateway.watsonplatform.net/language-translator/api/v2/identifiable_languages	Params

Authorization ●   Headers **(3)**   Body   Pre-request Script   Tests

Type   Basic Auth   ∨

Username   <your username>   The authorization header will be generated and added as a custom header

Password   ••••••••••••   ☐ Save helper data to request

     ☐ Show Password

e)   Press the **Send** button and see the result. A long list of available languages.

Test the Text to Speech service

Step 13 In Postman specify the following parameter for the **POST synthesize** request.

a)   Url from your service definition. See **Step 3 c)** and add the method **/v1/synthesize**

```
https://stream.watsonplatform.net/text-to-speech/api/v1/
synthesize
```

b)   Add the Voice you want to use (see the API Reference) to the url above using the [Params] button.

```
Voice
de-DE_BirgitVoice
```

c)   This service uses basic authentication. In the *Authentication* section **enter** *apikey* as *Username* and the apikey from your service definition as *password*.

d) In the *Headers* section specify the following parameters:

Key: C**ontent-Type**   Value: **application/json**
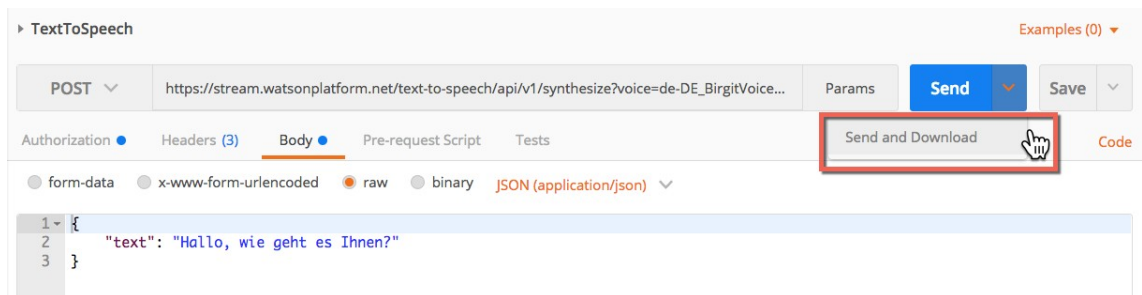
Key: **Accept:**   Value: **audio/wav**

e) Specify the following for the *Body* section in *raw* format.

```
{"text": "Hallo, wie geht es Ihnen!"}
```

f) Because we will receive a binary audio response, **do not** press the **Send** button, but on the Send button's down arrow the **Send and Download**.

Note: With the standalone version of Postman you can now also use the *Send* button.

g) Save the file as response.wav to a location on your hard drive and **double-click** it afterwards.

Section 2: Watson APIs in Web Applications

Running the Sample Application Locally

To see the IBM Watson Services integrated in a Web application we have created a sample that you need to download or clone from Github to your Workstation.

The following screenshots are based on the **Google Chrome** browser and **Microsoft Visual Studio Code**. Both tools can be downloaded for free. Other browsers and development environments/code editors work as well.

Step 14 In the Browser navigate to the following project repository on Github

https://github.com/iic-dach/image-analysis

Step 15 Copy the Github clone URL to the clipboard or download the ZIP.
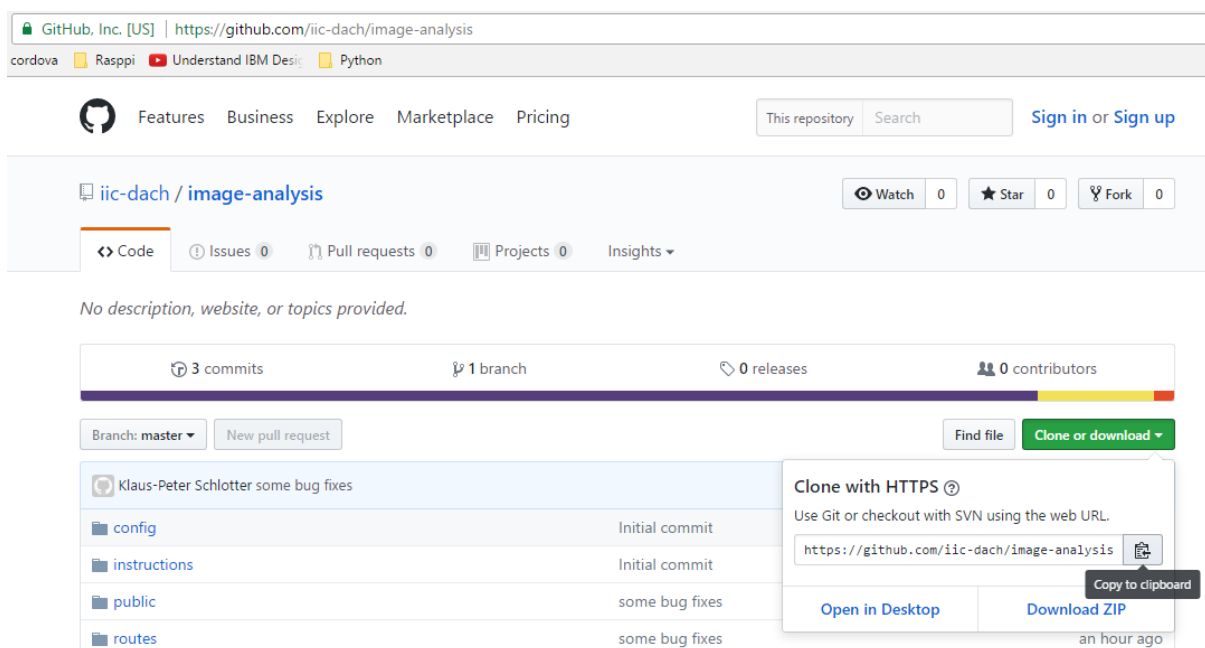
Step 16 Open a Terminal/Command window and navigate to a folder that will contain the project. (F.e. */home/<user>/iicworkshop* on Mac OS).

Step 17 Do the following based on Step 15

    a)   In the terminal enter git clone and past the URL behind it.

```
git clone https://github.com/iic-dach/image-analysis.git
```

    b)   Unpack the downloaded ZIP file into the folder from Step 16.

Step 18 Move to the folder created in Step 17 (f.e. *image-analysis*)

The project consists of the following components

a) A server component based on **Node.js**. The main file is **app.js** with with the appropriate routes defined in the *routes* folder.

b) A client component based on jQuery and bootstrap with the business logic defined in the *j***s** folder and a root **index.html** file.

Step 19 Start Visuals Studio Code and open the project folder (f.e. /Users/<user>/ image-analysis).
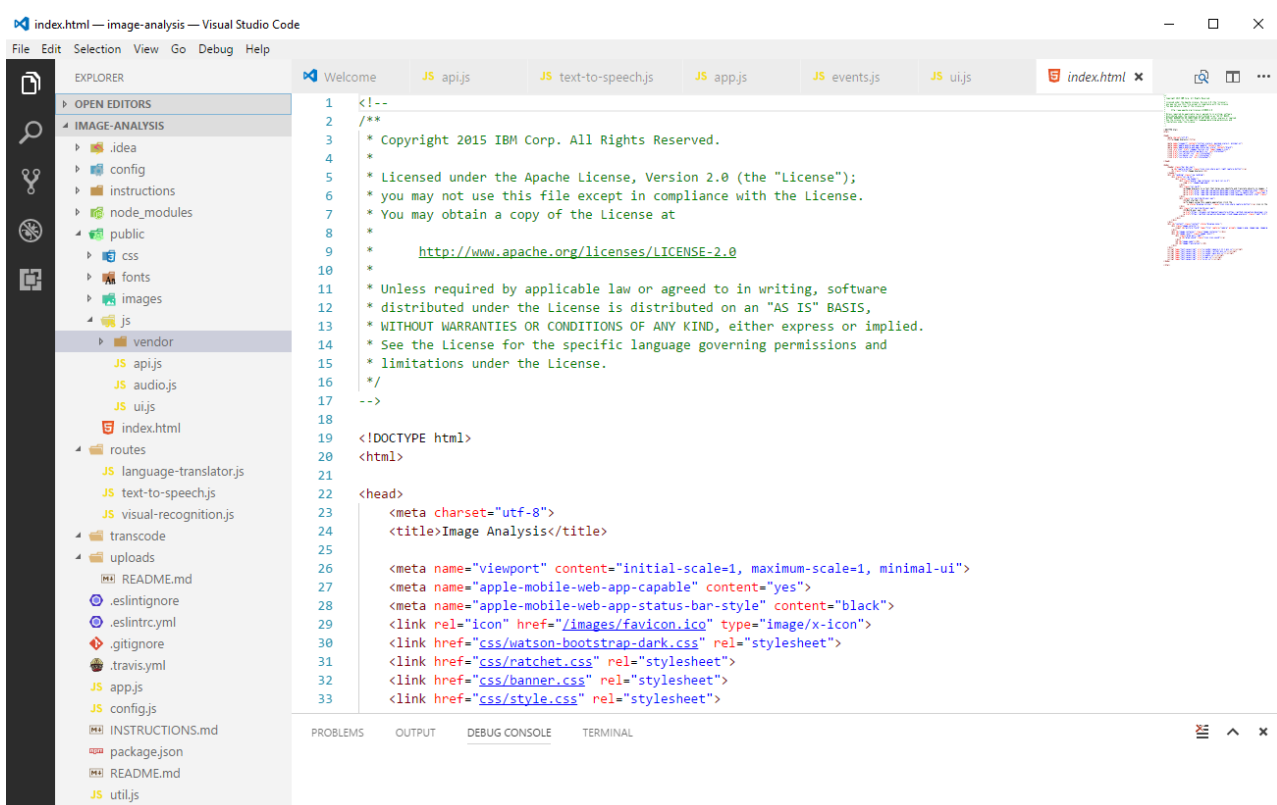
Step 20 In the terminal Window type

```
npm install
```

This command installs all the dependencies of the project defined in package.json (the node_modules folder is created during this process).

Step 21 Copy *config.sample.js* to *config.js*

Step 22 In the config.js file enter the user ids, passwords, and api key created in Steps 2 – 4.

Step 23 Save config.js

Step 24 Start the server by either Step 25 a) or Step 25 b)

a) **Enter** the following commend will start the nodejs server

```
npm start
```
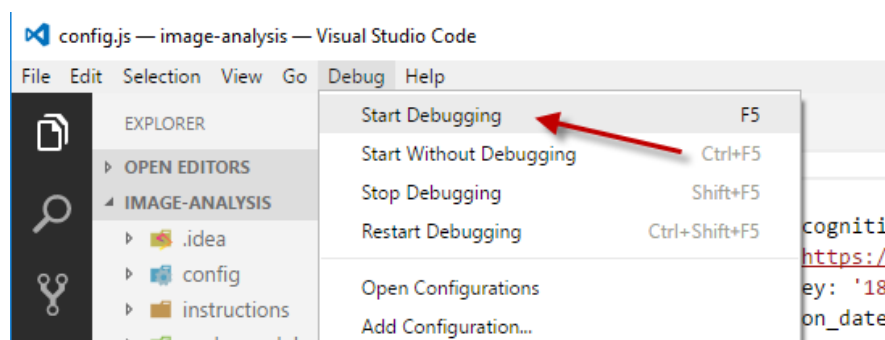
The servers console is now displayed in the terminal.

b) Start the server in debug modus in Visual Studio Code

You then see the server's debug console in Visual Studio code.

Step 25 Structure of the server component:

- **app.js**: Here the *multer* and *body-parser* nodejs modules, to handle the request data from the client, are imported and initialized.
  The express app is created and the middleware functionality is bound with the app.use() commands.
  The functionality of the Watson services is bound with the *watsonRoutes* middleware.
  To serve the client a path to the **public** folder is set to static.
  Finally the server is started with the app.listen() command.

- **routes/watson.js**: An express Router is created and the url fragments for the client requests are bound to the appropriate Watson controller function.

- **controllers/watson.js**: The Watson services are imported and initialized with the data from the config.js file that holds the Watson Services definitions.

  ○ **recognize**: This function checks whether an image url or image file is posted to prepare the parameters for the **classify** method of the Watson Visual Recognition service. On a successful return the file gets deleted and the response, in JSON format, is returned to the client.

  ○ **translate**: Translate get the text to be translated from the request body, sets the language to German and calls the **translate** function of the Watson Language Translator service. On a successful return the response, in JSON format, is returned to the client.

  ○ **speak**: Here the text, the voice and the audio format for the **synthesize** method of the Watson Text-to-Speech service are retrieved from the request body. On a successful return the arraybuffer with the audio content is returned to the client.

Step 26 Structure of the client component:

- **public/index.html**: This static html page is loaded whenever a request with the root url '/' arrives at the server. The page loads additional client files that contain all the client logic, jquery based, and some styling (/**public/css**).

- **public/js/ui.js**: This is the main file for the client logic. Depending on the page content, the ui components will display or hide (toggle function) and certain **onclick** and **onchange** handler are bound to the page components.
The page submit button (Analyze Image) is captured by jquery and has an associated **pre-submit** callback, to activate the spinner, and a post-submit callback to receive the *analyze image* result.
On success the received content is sent to the server via the **translate** function.
On a successful translate, function *onSuccess* the result table is build and the table rows are bound with a call to the **speak** function on each speaker icon. The spinner is stopped.
On a successful speak, a buffer with audio data is received and send to the audio player.

- **public/js/api.js**: To control the response type of 'arraybuffer' a native ajax call is implemented for the **speak** function.
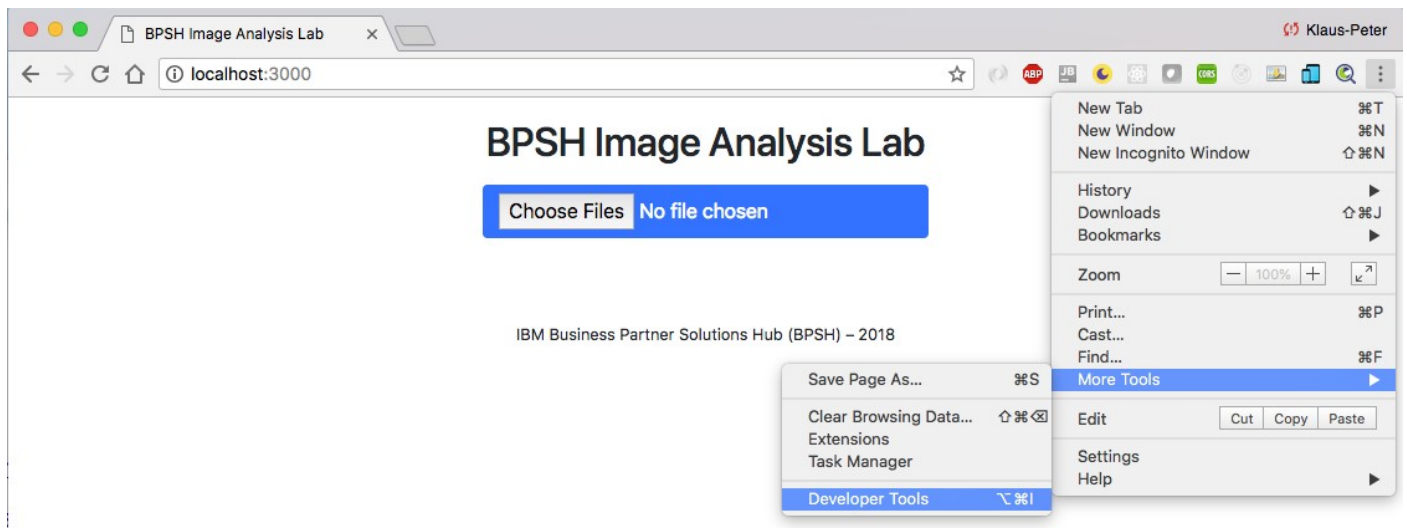
- **public/js/audio.js**: contains the sound logic.

Step 27 In the browser navigate to

http://localhost:3000

The index.html file gets loaded from the nodejs server.
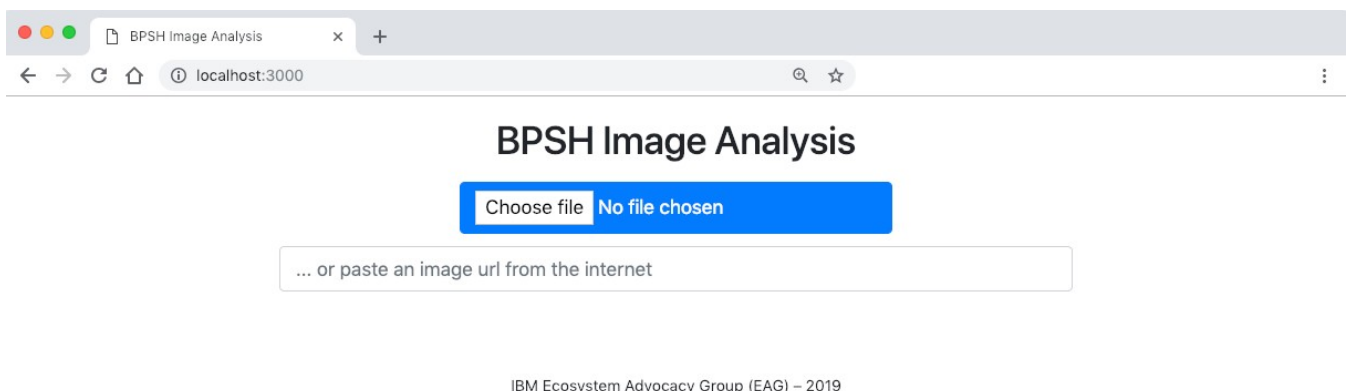
**Step 28** To see how the process works, open the Developer Tools of the Chrome Browser.

Step 29 Click the [Choose Files | No file chosen] button

Step 30 The button opens the appropriate file selection panel of your operating system. **Navigate** to the *<project folder>/public/images* and select one of the images, then click **Open**.

Step 31 Alternatively you can past an image url from the internet.

Step 32 When the image is selected, click [Analyze Image] to start the Visual Recognition of the image and the translation of the detected image classes..
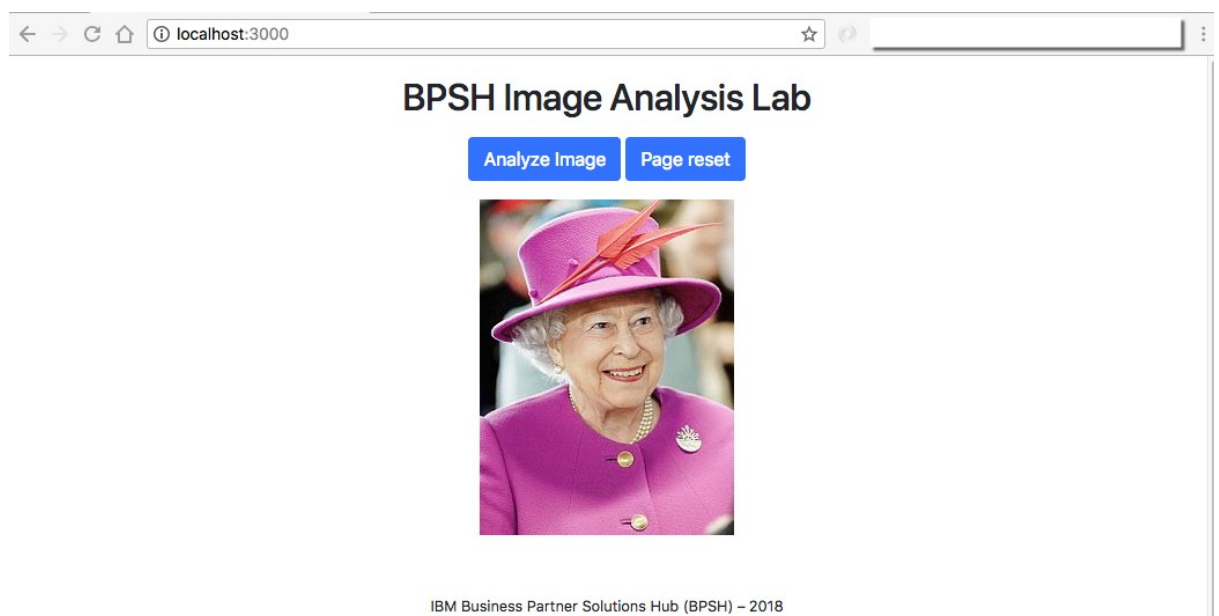
Step 33 Under the image a table is displayed that shows the detected classes, their translations and the confidence score of the detection.

Detected Class	Translated Class	Confidence	
queen	queen	0.836	🔊
person	Person	0.985	🔊
Queen of England	Königin von England	0.815	🔊
queen mother	Königin Mutter	0.55	🔊

In the *Developer Tools* on the *Network* tab you can inspect the responses from the **recognition** and the **translate** request.

Step 34 Now on the table you can click any of the result rows to hear the translated class.

The response to this call contains a buffer with the content of a .wav file buffer that is automatically played in the browser.

Section 3: Push the local Web Application to IBM Cloud

The final step of this lab is to push the application that runs locally on your machine into your Bluemix account and make it publicly available.

Note: This simple example currently works on Android with images saved previously, not from the camera. iOS only allows camera images that do not work.

Step 35 The following step assumes, that your terminal is connected to your IBM Cloud account as described in **Setup document Step 9**.

Step 36 In the Terminal window, in the applications root folder (f.e. /Users/<user>/ image-analysis) enter following command: (Note: Step 39).

```
ibmcloud app push xxxImageAnalysis
```

where xxx stands for a prefix you choose for your application, because this url is public and must therefore be unique.

This process takes some time to upload, build and start the app. In the end you should see something like the following:

```
Uploading droplet...
Uploaded build artifacts cache (7M)
Uploaded droplet (29.5M)
Uploading complete

0 of 1 instances running, 1 starting
1 of 1 instances running

App started


OK

App kpsImageAnalysis was started using this command `./vendor/initial_startup.rb`

Showing health and status for app kpsImageAnalysis in org iic_ehningen / space demo as iic_stuttgart@de.ibm.com...
OK

requested state: started
instances: 1/1
usage: 1G x 1 instances
urls: kpsimageanalysis.mybluemix.net
last uploaded: Wed Jun 14 11:40:39 UTC 2017
stack: cflinuxfs2
buildpack: SDK for Node.js(TM) (ibm-node.js-6.10.2, buildpack-v3.12-20170505-0656)

     state     since                    cpu    memory   disk     details
#0   running   2017-06-14 01:41:39 PM   0.0%   0 of 1G  0 of 1G
```
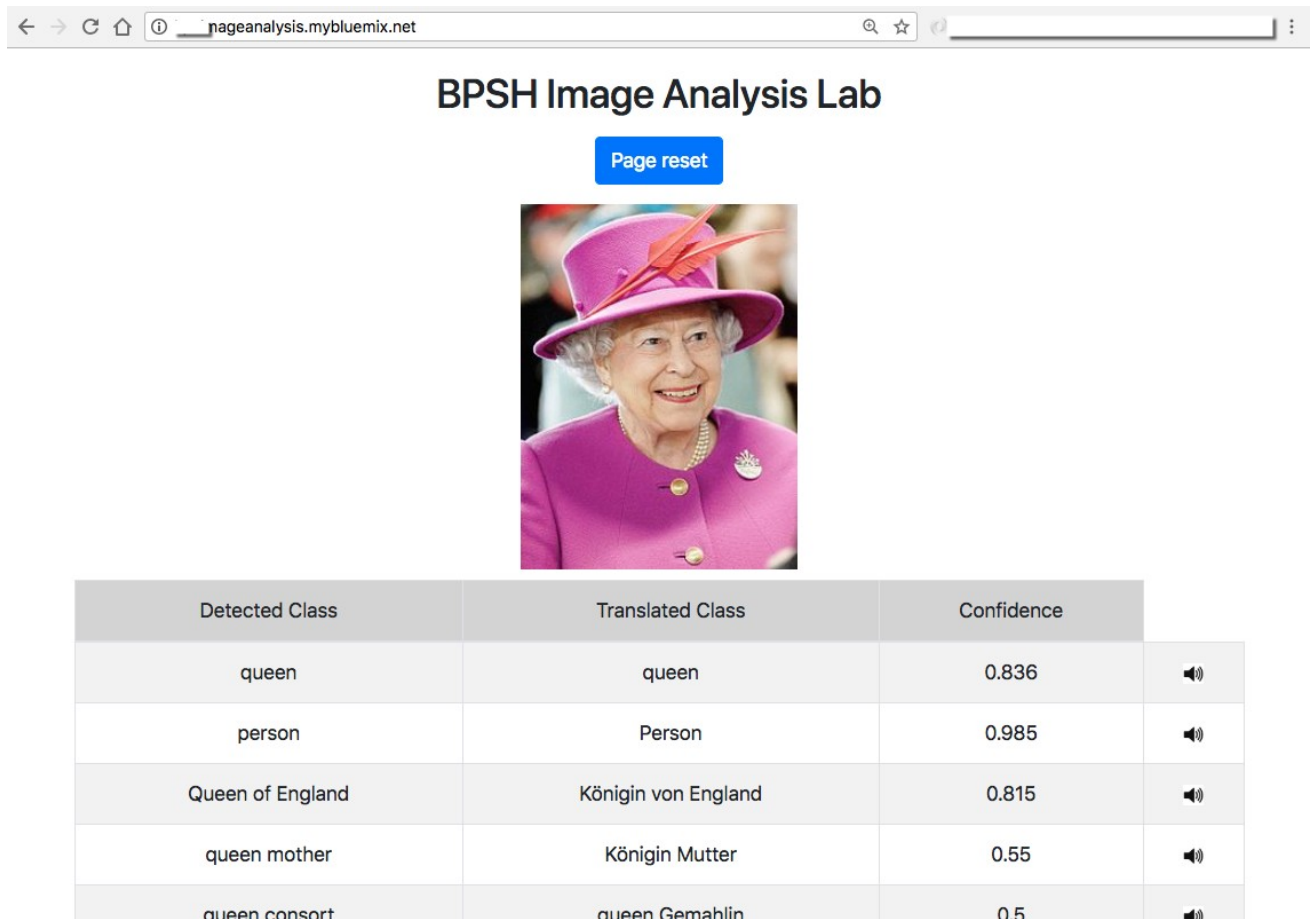
Step 37 In the browser navigate to http://xxximageanalysis.mybluemix.net. The application should work as locally installed.

Step 38 You can display the server console from IBM Cloud in your Terminal with the following command

```
ibmcloud app logs xxximageanalysis
```

Step 39 When you have a Lite account, the actual start of the application will fail because the default memory assigned to this application will exceed the limit of 256MB. Therefore you have to create a manifest.yml file in your projects root folder.

    a) Create a manifest.yml in your root folder, f.e. in image-analysis.

b) Add the following content

```
applications:
  - name: xxx-imageanalysis
    path: .
    buildpack: sdk-for-nodejs
    command: node app.js
    memory: 256M
```

Specify your unique prefix instead of xxx-.

c) **Save** and **close** the file

d) You can now push by simply

```
ibmcloud app push
```