

IBM Natural Language Understanding – Node.js

Cognitive Solutions Application Development

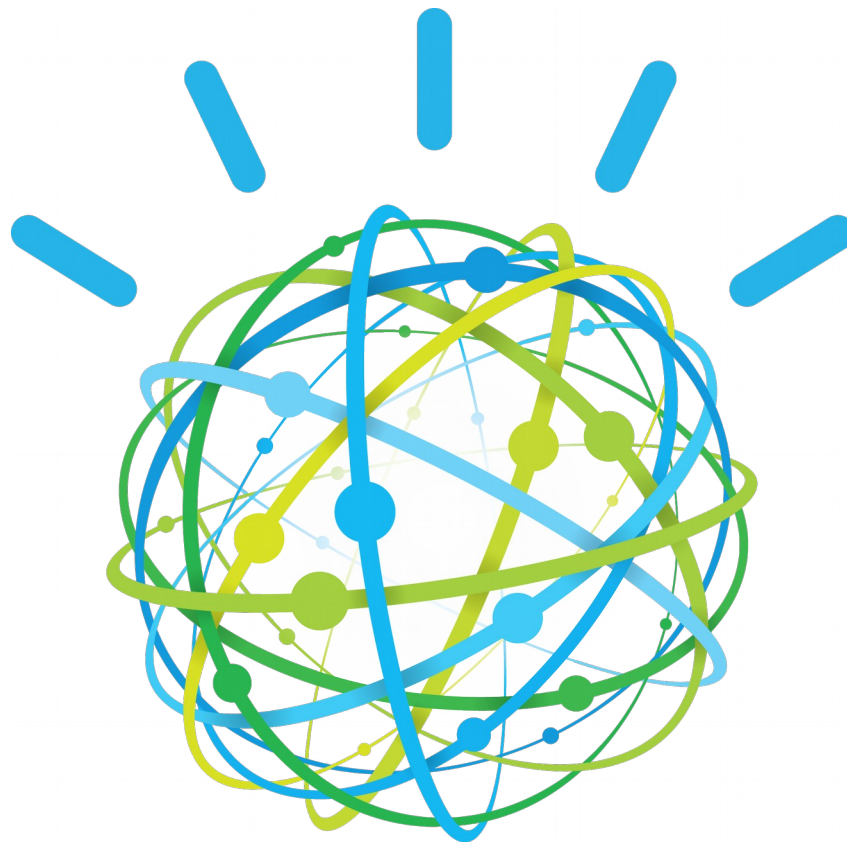
IBM Global Business Partners

Duration: 90 minutes

Updated: Oct 16, 2019

Klaus-Peter Schlotter

kps@de.ibm.com



IBM

Version 4.0 (Watson library V4.x)

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.

With [Natural Language Understanding](#), developers can analyze semantic features of input text and extract metadata from content, such as categories, concepts, emotion, entities, keywords, metadata, relations, semantic roles, and sentiment. With custom annotation models developed using IBM Watson Knowledge Studio, you can further customize the service to identify domain-specific entities and relations in your content.

Natural Language Understanding can be useful in many scenarios that demand rapid analysis of unstructured text without requiring in-depth natural language processing expertise. For example, you can monitor sentiment and emotion in customer support chat transcripts, or you can quickly categorize blog posts and sort them based on general concepts, keywords, and entities.

The features that are available depend on the language. This list may change over time, so always check this in the [online documentation](#).

The Application we will create uses the [Watson Natural Language Understanding \(NLU\)](#) service available in the IBM Cloud.

This app will be created and run from a local workstation and later to be deployed on the IBM Cloud.

- In a first step, an instance of the Watson NLU service will be created.
- In a second step, this service instance will be used with the Postman REST Client to get accustomed to capabilities of the service.
- Then a Node.js application will be created to use the NLU service from an application.
- 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 the capabilities of the NLU service.
- Learn how to implement the NLU application using Node.js
- 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 **Git clone** at the beginning of Section 2.

Note: Copy and Paste from this PDF document does not always produce the desired result, therefore open the [code snippets](#) for this lab in the browser and copy from there!

Section 1: Testing Watson APIs with a REST Client

Create services in IBM Cloud

IBM Cloud offers services, or cloud extensions, that provide additional functionality that is ready to use by your application's running code.

You have two options for working with applications and services in IBM Cloud. You can use the IBM Cloud web user interface or the [IBM Cloud command-line interface \(CLI\)](#). This CLI only works when you have completed **Step 4** in the [workstation setup](#) document while creating your IBM Cloud Account.

Note: This is the option chosen here because it is the base for automated scripts.

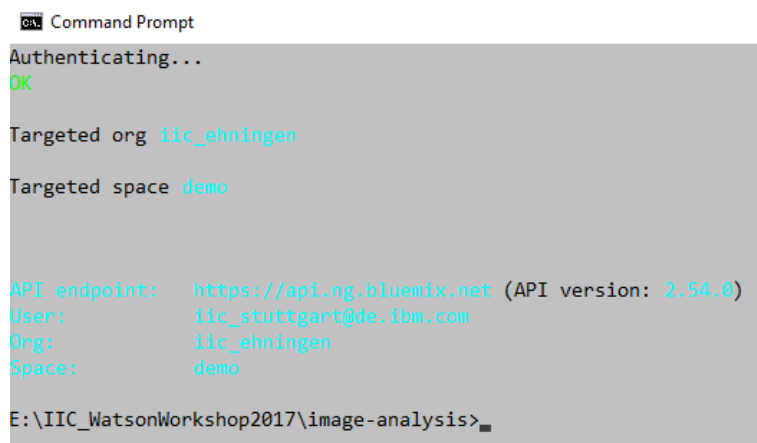
Step 1 Open a Terminal/Command Window and enter the following commands

```
ibmcloud api https://api.<region>.bluemix.net
```

where region is either

- ng – for region “US South”
- eu-gb – for region “United Kingdom”
- eu-de – for region “Germany”

```
ibmcloud login -u <your Bluemix user account> -o <yourOrg> -s <yourSpace>
```



```

C:\ Command Prompt
Authenticating...
OK
Targeted org iic_ehningen
Targeted space demo

API endpoint: https://api.ng.bluemix.net (API version: 2.54.0)
User: iic_stuttgart@de.ibm.com
Org: iic_ehningen
Space: demo

E:\IIC_WatsonWorkshop2017\image-analysis>
  
```

When prompted, enter your IBM Cloud password.

Step 2 Create the NLU service

- a) (Optional) Get information about all services available

```
ibmcloud catalog service-marketplace
```

You should see an entry *natural-language-understanding* that has an option “lite”

5932737c-e24b-4836-9e37-d8cead922316	gc_migrate,ibm_created,ibm_dedicated_public,ibm_third_party	natural-language-generation-apis
bb6393bc-7d81-446b-9728-61b704f6eca5	eu_access,gc_migrate,ibm_created,ibm_dedicated_public,lite,watson	natural-language-understanding
3f5debff-044f-4292-b387-9c9a50ba81fe	gc_migrate,ibm_created,rc_compatible,virtual_data_center,vmware	nexmo

- b) Create the free service with name *my-nlu*

```
ibmcloud service create natural-language-understanding free my-nlu
```

- c) Generate the authentication credentials

```
ibmcloud service key-create my-nlu credentials-1
```

- d) Retrieve the new credentials


```
ibmcloud service key-show my-nlu credentials-1
```

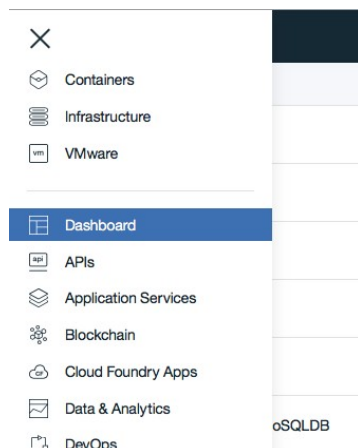
You should see the credentials and url of your “NLU” service. Later in this exercise you will need these values. Feel free to copy these values to a text file or just leave the terminal/command window open until it is needed.

Step 3 You could have done Step 2 in the IBM Cloud console so let’s check what has been created.

- a) Open a Browser and navigate to <https://console.<region>.bluemix.net>

Use the region you have specified in **Step 1**.

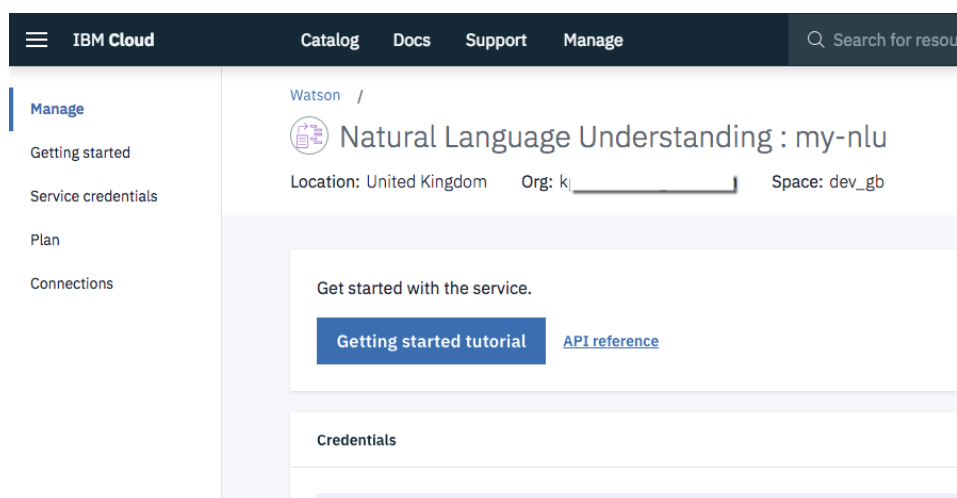
b) Click the  icon and then **click** Dashboard.



On the list of services you should see the NLU service “my-nlu created in previous steps.

Cloud Foundry Services 18/40 Used		
Name	Service Offering	Plan
Secure Gateway-cy	Secure Gateway	securegatewayplan
my-nlu	Natural Language Understanding	Lite
kpsWeather	Weather Company Data	Free Plan

c) Click the **my-nlu** service to open it. Here you see an overview of your service instance



Here you can see the details of the service you have instantiated in **Step 2**.

d) Click the *Service credentials* and under **ACTIONS** click *View credentials*

The screenshot shows the IBM Watson console for the 'Natural Language Understanding : my-nlu' service. On the left sidebar, 'Service credentials' is highlighted. The main content area shows a table of service credentials. The first row is 'Auto-generated service credentials' and the second row is 'credentials-1', which is highlighted with a red box. The 'credentials-1' row has a 'View credentials' button next to it, also highlighted with a blue box. Below the table, a JSON snippet is displayed, showing the API key and secret for the 'credentials-1' entry.

Service credentials

Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service. [Learn more](#)

Service credentials [New credential](#)

10 Items per page | 1-2 of 2 items 1 of 1 pages < 1 >

KEY NAME	DATE CREATED	ACTIONS
<input type="checkbox"/> Auto-generated service credentials	AUG 10, 2018 - 01:13:08 PM	View credentials
<input type="checkbox"/> credentials-1	AUG 10, 2018 - 01:29:32 PM	View credentials

```
{
  "url": "https://gateway.watsonplatform.net/natural-language-understanding/api",
  "username": "b6d4c...8c3d",
  "password": "2...B"
}
```

Here you see the credentials created in **Step 2d)**

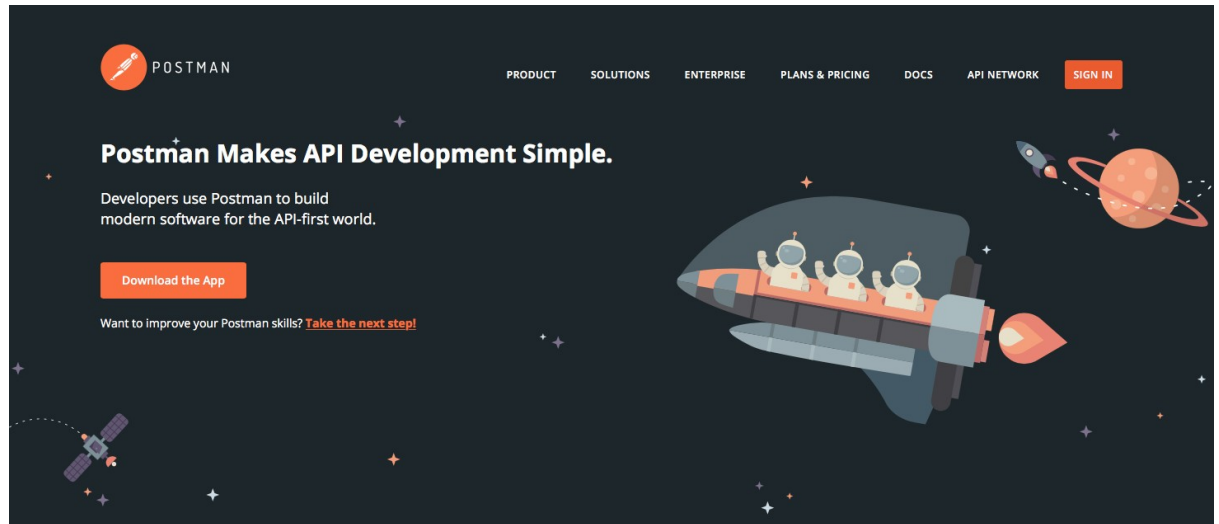
Note: There is also an auto-generated set of credentials that we do not use.

Work with your service 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 4 [Postman is available](https://www.getpostman.com) as an application for MacOS, Windows, and Linux.
<https://www.getpostman.com>



Step 5 Open the Postman client

Test Natural Language Understanding (NLU)

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

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

A simple first api call will be to retrieve the machine learning models that are associated to the service, a GET request

- Get the *url* and the *credentials* retrieved in **Step 2 d)**, or copy the these values from within your **my-nlu service – Service credentials** in the Bluemix console
- Build the URL for testing with the NLU service (GET request)

url: <https://gateway.watsonplatform.net/natural-language-understanding/api/v1/models>
 params: version 2017-02-27"

The screenshot shows the Postman interface for a GET request. The URL is `https://gateway.watsonplatform.net/natural-language-understanding/api/v1/models?ver...` and the parameter is `version` with the value `2017-02-27"`. The 'Params' tab is selected, and the 'Send' button is visible.

- In the Authorization section select Basic Auth and enter the username and password of your service instance. In case your service has an apikey, use the word *apikey* as username and the apikey from the service as password.

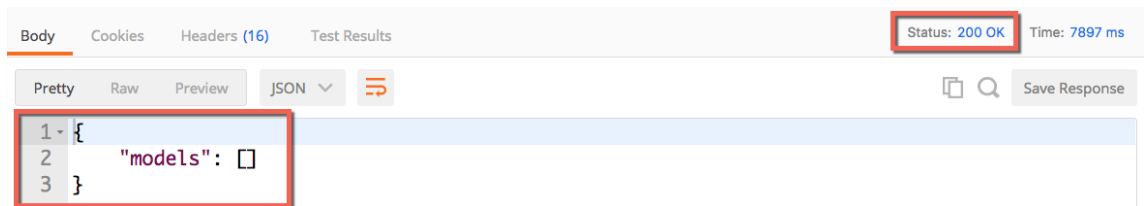
The screenshot shows the Postman interface with the 'Authorization' tab selected. The 'Type' is set to 'Basic Auth'. The 'Username' field contains `38c...` and the 'Password' field is empty. The 'Update Request' button is visible.

- You can save your request(s) for further use with Save As..

The screenshot shows the Postman interface with the 'Save As...' button highlighted in the bottom right corner. The URL and parameters are the same as in the previous screenshots.

- e) Click the **Send** button

When all values are entered correctly you should see a Status 200 OK and json object with an empty models array.



Step 7 Testing NLU Concepts

The NLU Concepts feature identifies high-level concepts that might not be directly referenced in the input text. Concept-related API functions understand how concepts relate. Concepts that are detected typically have an associated link to a *DBpedia* resource. See the following input and response examples

- a) In Postman you should have the request from previous step. **Select**

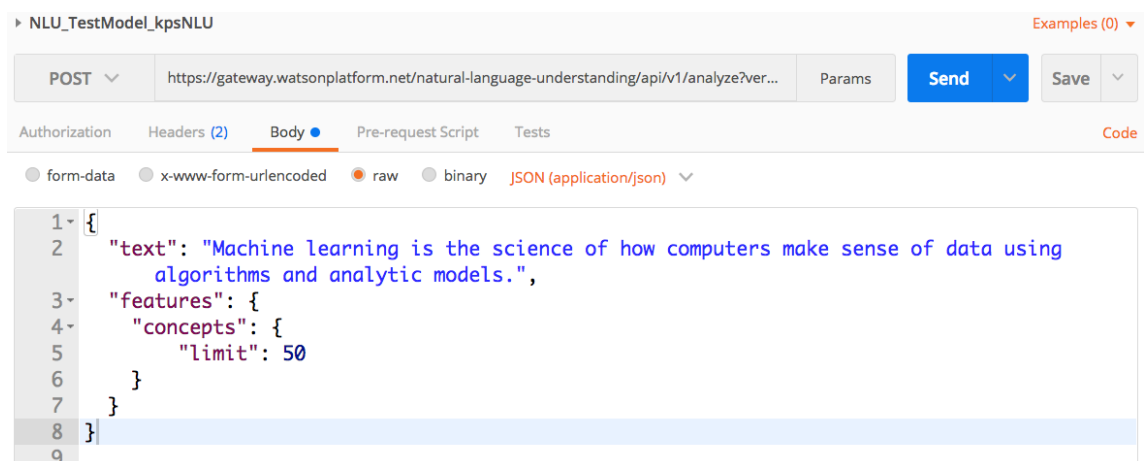
POST as the request type.

In the **URL** just change the uri **model** to **analyze**.

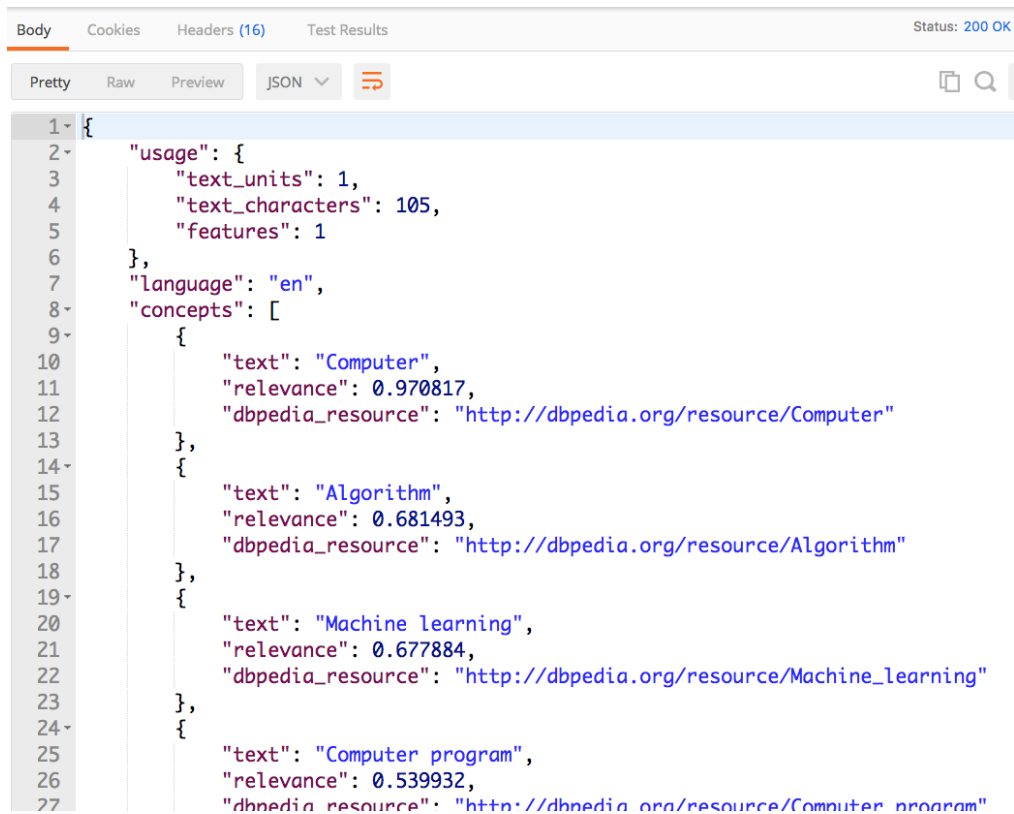
In the **Body** of the request specify the following json input

In the **Header** specify **Content-Type** (Key) and **application/json** (Value)

```
{
  "text": "Machine learning is the science of how computers make sense of
data using algorithms and analytic models.",
  "features": {
    "concepts": {
      "limit": 50
    }
  }
}
```



You should see the following result:



```

1 {
2   "usage": {
3     "text_units": 1,
4     "text_characters": 105,
5     "features": 1
6   },
7   "language": "en",
8   "concepts": [
9     {
10      "text": "Computer",
11      "relevance": 0.970817,
12      "dbpedia_resource": "http://dbpedia.org/resource/Computer"
13    },
14    {
15      "text": "Algorithm",
16      "relevance": 0.681493,
17      "dbpedia_resource": "http://dbpedia.org/resource/Algorithm"
18    },
19    {
20      "text": "Machine learning",
21      "relevance": 0.677884,
22      "dbpedia_resource": "http://dbpedia.org/resource/Machine_learning"
23    },
24    {
25      "text": "Computer program",
26      "relevance": 0.539932,
27      "dbpedia_resource": "http://dbpedia.org/resource/Computer_program"
28    }
29  ]
30 }
  
```

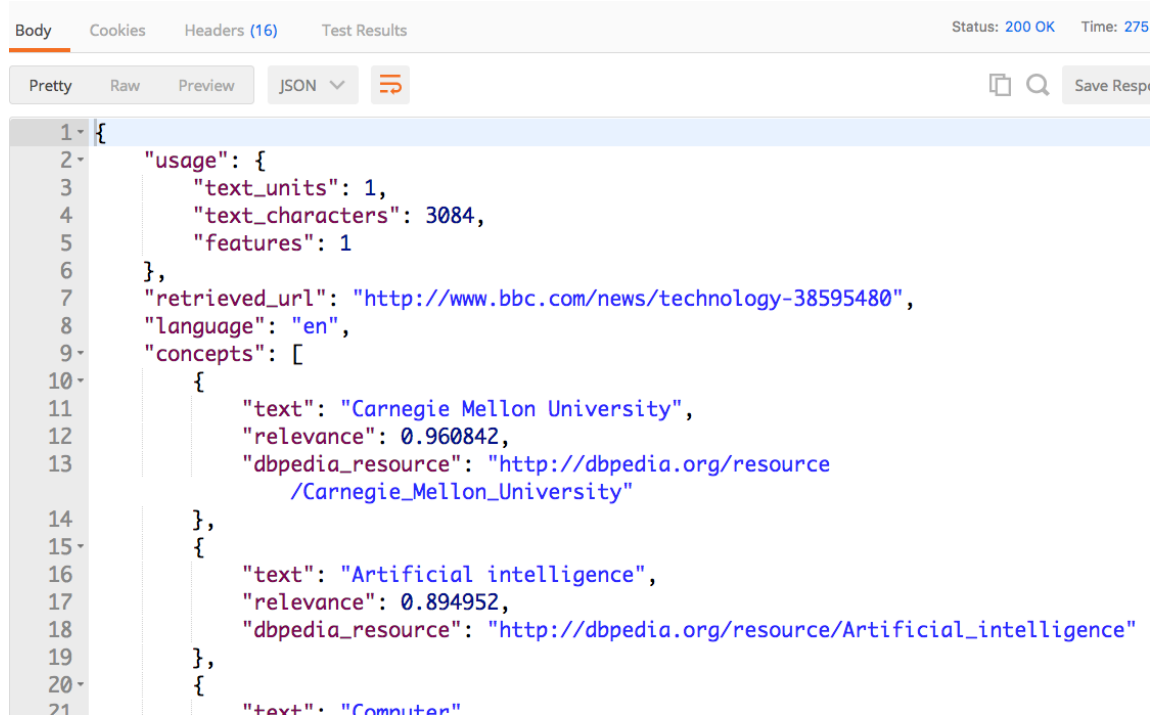
b) Instead of text, you can also enter a url instead of text.

Body:

```

{ "url": "http://www.bbc.com/news/technology-38595480",
  "features": {
    "concepts": {
      "limit": 50
    }
  }
}
  
```

Result:



```

1 {
2   "usage": {
3     "text_units": 1,
4     "text_characters": 3084,
5     "features": 1
6   },
7   "retrieved_url": "http://www.bbc.com/news/technology-38595480",
8   "language": "en",
9   "concepts": [
10    {
11      "text": "Carnegie Mellon University",
12      "relevance": 0.960842,
13      "dbpedia_resource": "http://dbpedia.org/resource/Carnegie_Mellon_University"
14    },
15    {
16      "text": "Artificial intelligence",
17      "relevance": 0.894952,
18      "dbpedia_resource": "http://dbpedia.org/resource/Artificial_intelligence"
19    },
20    {
21      "text": "Computer"

```

Step 8 Testing NLU Entities

The NLU Entities feature helps you identify people, cities, organizations, and many other types of entities in your text. It returns items such as persons, places, and organizations that are present in the input text. Entity extraction adds semantic knowledge to content to help understand the subject and context of the text that is being analyzed.

- a) In the body of the previous Postman POST request change the Body to the following:

```
{ "text": "IBM is an American multinational technology company
headquartered in Armonk, New York, United States, with operations
in over 170 countries.",
  "features": {
    "entities": {
      "limit": 250
    }
  }
}
```

Result:

```
1 {
2   "usage": {
3     "text_units": 1,
4     "text_characters": 140,
5     "features": 1
6   },
7   "language": "en",
8   "entities": [
9     {
10      "type": "Company",
11      "text": "IBM",
12      "relevance": 0.33,
13      "disambiguation": {
14        "subtype": [
15          "SoftwareLicense",
16          "OperatingSystemDeveloper",
17          "ProcessorManufacturer",
18          "SoftwareDeveloper",
19          "CompanyFounder",
20          "ProgrammingLanguageDesigner",
21          "ProgrammingLanguageDeveloper"
22        ],
23        "name": "IBM",
24        "dbpedia_resource": "http://dbpedia.org/resource/IBM"
25      },
26      "count": 1
27    },
28    {
29      "type": "Location",
30      "text": "Armonk",
31      "relevance": 0.33,
32      "disambiguation": {
33        "subtype": [
34          "City"
35        ]
36      },
37      "count": 1
38    },
39    {
40      "type": "Location"
```

- b) A text in German:
Die Fußball Weltmeisterschaft 2006 hat in Deutschland stattgefunden. Das Finale in Berlin gewann Italien gegen Frankreich. Dritter wurde Deutschland in Stuttgart mit einem Sieg gegen Portugal.

Step 9 Testing NLU Relations

The NLU Relations feature identifies subject, action, and object relations within sentences in the input content. After parsing sentences into subject, action, and object form, the Relations feature can use this information for subsequent processing by other Natural Language Understanding features such as entities, keywords, and so on.

Relation information can be used to automatically identify buying signals, key events, and other important actions.

- a) In the Postman change the body of the POST to the following:

Body:

```
{ "text": "Bob Dylan won the Nobel Prize in Literature
in 2016. Bob Dylan was born in Duluth, Minnesota.",
  "features": {
    "relations": {}
  }
}
```

Result:

```
1 {
2   "usage": {
3     "text_units": 1,
4     "text_characters": 93,
5     "features": 1
6   },
7   "relations": [
8     {
9       "type": "affectedBy",
10      "sentence": "Bob Dylan won the Nobel Prize in Literature in 2016.",
11      "score": 0.892004,
12      "arguments": [
13        {
14          "text": "Bob Dylan",
15          "location": [
16            0,
17            9
18          ],
19          "entities": [
20            {
21              "type": "Person",
22              "text": "Bob Dylan"
23            }
24          ]
25        },
26        {
27          "text": "won",
28          "location": [
29            10,
30            13
31          ],
32          "entities": [
33            {
34              "type": "EntertainmentAward",
35              "text": "won"
36            }
37          ]
38        }
39      ]
40    },
41    {
42      "type": "awardedTo",
43      "sentence": "Bob Dylan won the Nobel Prize in Literature in 2016."
44    }
45  ]
46 }
```

Section 2: Watson APIs in Web Applications

Running the Sample Application Locally

In this section we will create a sample application based on Node.js step by step or you can download or clone the completed app from [Github](#).

Note: See **Lab 1 Section 2** on how to clone a Github repository.

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 10 In a Terminal window **create a folder** for your project f.e. *nlutester*

Step 11 Move into this folder and execute the following command

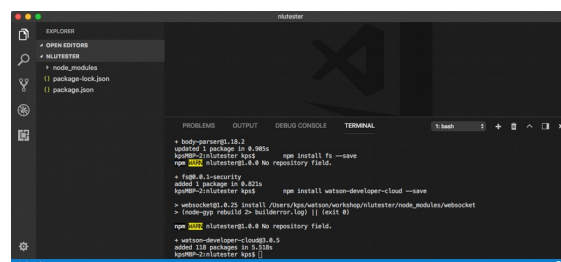
```
npm init
```

Specify a *name*, *description* and *author* when prompted, keep the other *defaults*. The result should be similar to the following in *package.json*:
Add the blue lines as show below:

```
{
  "name": "nlutester",
  "version": "1.0.0",
  "description": "A simple Watson Natural Language Understanding tester",
  "main": "index.js",
  "scripts": {
    "start": "node app.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "engines": {
    "node": "^10.15.0",
    "npm": "^6.4.1"
  },
  "author": "Klaus-Peter Schlotter",
  "license": "ISC"
}
```

Step 12 Open your project folder in Visual Studio code or you can start Visual Studio Code with the following command from within the project folder. In the menu **select** *View → Integrated Terminal*

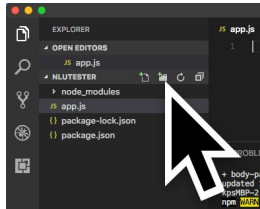
code .



Step 13 In the *Integrated Terminal* or another terminal execute the following commands to install the additional node modules needed for our project

```
npm install --save express ejs body-parser ibm-watson@4
```

The modules will be stored in the `node_modules` folder and your `package.json` file should now have a dependencies section.



Step 14 In the project folder *create* an `app.js` and an `helper.js` file. You can create files and folders for your project in the *Visual Studio Code Explorer* on the left.

Step 15 Create `config.js` in the project root folder and enter your service credentials.

```
var config = {
  watson: {
    nlunderstanding: {
      username: "<your service username>",
      password: "<your service password>",
      version: "2018-11-16",
    }
  }
};

module.exports = config;
```

```
const config = {
  watson: {
    nlunderstanding: {
      url: "<Url to your service>",
      iam_apikey: "<your API_Key>",
      version: "2018-11-16",
    }
  }
};

module.exports = config;
```

Depending on your service, use `username/password` or `apikey`.

Step 16 In the project *root* **create** a *folder* named `utils` and in there **create** a *file* named `helpers.js` with the following content: → (`utils/helpers.js`). **Save** and **close** the file.

`setSelection()` transforms the received array of selected checkboxes into the appropriate selection properties.

```
exports.setSelection = (items, selectedItems) => {
  const selection = [];
  items.forEach(f => {
    if (selectedItems.indexOf(f) > -1) {
      selection.push(true);
    } else {
      selection.push(false);
    }
  });
  return selection;
}
```


buildFeaturesRequest() transforms the Watson Natural Language service attributes received with the request to the appropriate parameter object.

```
exports.buildFeatureRequest = (features, model) => {
  let reqFeatures = {
    features: {}
  };
  features.forEach(f => {
    if (f === 'entities' || f === 'relations') {
      if (model[f]) {
        reqFeatures.features[f] = {
          model: model.ids[0]
        };
      } else {
        reqFeatures.features[f] = {};
      }
    } else {
      reqFeatures.features[f] = {};
    }
  });
  return reqFeatures;
};
```

Step 17 In `app.js` enter the following code to import the dependencies and instantiate the variables. **Save** and **close** the file.

```
const path = require('path');

const express = require('express');
const bodyParser = require('body-parser');

const app = express();
app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));

const port = process.env.PORT || 3000;
app.set('view engine', 'ejs');
app.set('views', 'views');

const watsonRoutes = require('./routes/watson');
app.use(watsonRoutes);

app.use(function (request, response) {
  response.status(404).render("404");
});

app.listen(port, () => {
  console.log('Express app started on port ' + port);
});
```

Step 18 In the project *root* **create** a *folder* named *routes* and in there **create** a *file* named *watson.js* with the following content: → (routes/watson.js). **Save** and **close** the file.

```
const express = require('express');

const watsonController = require('../controllers/watson');

const router = express.Router();

router.get('/', watsonController.getIndex);

router.post('/', watsonController.postNlu);

router.get('/getmodel', watsonController.getNluModel);

module.exports = router;
```

The router defines the urls we accept from the web with an appropriate function called that is defined in the controller file.

Step 19 In the project *root* **create** a *folder* named *controllers* and in there **create** a *file* named *watson.js* with the following content: → (controllers/watson.js)

In line 1 – 19 add the following code:

```
const NaturalLanguageUnderstandingV1 = require('ibm-watson/natural-language-understanding/v1');

const config = require('../config');
const { setSelection, buildFeatureRequest } = require('../utils/helpers');

const nlu = new NaturalLanguageUnderstandingV1(config.watson.nlunderstanding);

// fixed values for rendered page
const features = ["categories", "concepts", "emotion", "entities",
  "keywords", "metadata", "relations", "semantic_roles"];
const inputTypes = ["url", "text"];

let model = {
  label: '',
  ids: [],
  selected: false,
  entities: false,
  relations: false
}
```

Step 20 The `getIndex` function renders the initial page. (Line 21 – 36)

```
exports.getIndex = (req, res, next) => {
  let result = '';
  let bodyText = '';
  model.selected = false;
  model.entities = false;
  model.relations = false;
  res.render('index', {
    result: result,
    bodyText: bodyText,
    features: features,
    inputTypes: inputTypes,
    fselected: setSelection(features, features[3]),
    iselectd: setSelection(inputTypes, inputTypes[1]),
    model: model
  });
};
```

Initial page load, forces certain default values.

Step 21 The `getNluModel` function, called when the browser page is loaded (*app/scripts/jqhelpers.js* `getModel()`), checks for a deployed model and displays it when found. (Line 38 – 55)

```
exports.getNluModel = (req, res, next) => {
  nluListModels()
    .then(response => {
      if (response.models.length > 0) {
        model.label = 'Model: '
        model.ids[0] = response.models[0].model_id;
        console.log(model.ids[0]);
        res.send({modelId: model.ids[0]});
      } else {
        console.log("no model deployed to service");
        res.send({text: "no model deployed to service"});
      }
    })
    .catch(err => {
      console.log('error:', err.message);
      res.status(200).send({text: "checkModel(): " + err.message});
    });
};
```

Step 22 When the submit (form action=post) button on the *index* page is pressed, the POST for the root url "/" gets called. (57 – 109)

```
exports.postNlu = (req, res, next) => {
  if (!req.body.inputType || !req.body.body || !req.body.features) {
    res.status(400).send("All fields required!");
    return;
  }
  const bodyText = req.body.body;
  let receivedFeatures;
  model.selected = false;
  model.entities = false;
  model.relations = false;
  if (req.body.mlModel) {
    model.selected = true;
    if (req.body.mlEntities) {
      model.entities = true;
    }
    if (req.body.mlRelations) {
      model.relations = true;
    }
  }
  if (Array.isArray(req.body.features)) {
    receivedFeatures = req.body.features;
  } else {
    receivedFeatures = req.body.features.split(',');
  }
  reqFeatures = buildFeatureRequest(receivedFeatures, model);
  const parameters = {
    [req.body.inputType]: bodyText,
    features: reqFeatures.features
  }
  nluAnalyze(parameters)
    .then(response => {
      result = JSON.stringify(response, null, 2);
      res.status(200).render("index", {
        result: result,
        bodyText: bodyText,
        inputTypes: inputTypes,
        features: features,
        fselected: setSelection(features, req.body.features),
        iselectd: setSelection(inputTypes, req.body.inputType),
        model: model
      });
    })
    .catch(err => {
      result = JSON.stringify(err, null, 2);
      res.status(200).render("index", {
        result: result,
        bodyText: bodyText,
        inputTypes: inputTypes,
        features: features,
        fselected: setSelection(features, req.body.features),
        iselectd: setSelection(inputTypes, req.body.inputType),
        model: model
      });
    });
};
```

Within the **postNlu** function the request parameters for the Watson NLU service are extracted from the request body and the `analyze` function of the `nlu` object gets called. In case of success and error a result object gets created and returned to the browser with the `res.render` command.

Step 23 With two helper functions the Watson API calls will be casted to a JavaScript Promise. This provides clean code with `then()` and `catch()` blocks instead of `if(err)...else`. Especially when multiple calls are nested. (Line 111 – 135)

```
// Convert nlu.listModels() to Promise
nluListModels = () => {
  return new Promise((resolve, reject) => {
    nlu.listModels({}, (err, res) => {
      if (err) {
        console.log('listModel: ' + err);
        reject(err);
      } else {
        resolve(res);
      }
    });
  });
};

// convert nlu.analyze() to a promise
nluAnalyze = (params) => {
  return new Promise((resolve, reject) => {
    nlu.analyze(params, (err, res) => {
      if (err) {
        reject(err);
      } else {
        resolve(res);
      }
    });
  });
};
```

Step 24 In the root folder **create** a folder named `views`.

Step 25 In the `views` folder **create** a `header.ejs` file with the following content:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>EAG NLU-Tester</title>
    <link rel="stylesheet"
href="//maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css">
  </head>
  <body class="container">
    <h2>
      IBM EAG NLU-Tester
    </h2>
```

Step 26 In the *views* folder create a *footer.ejs* file with the following content:

```
<p><small>IBM Ecosystem Advocacy Group - 2019</small></p>
</body>
</html>
```

Step 27 In the *views* folder create a *404.ejs* file with the following content:

```
<% include header %>
<h2>404! Page not found.</h2>
<% include footer %>
```

Step 28 In the *views* folder create an *index.ejs* file with the following content:

```
<% include header %>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script src="../scripts/jghelpers.js"></script>
<h3>Analyze text with IBM Watson Natural Language Understanding service</h3>
<form method="post" role="form">
  <hr>
  <div class="form-group" id="inputType">
    <% if (inputTypes.length) { %>
    <% for(var i=0; i < inputTypes.length; i++) { %>
    <input style="margin-right: 5px" type="radio" id=<%= inputTypes[i] %> name="inputType" value=<%= inputTypes[i] %> <%= isSelected[i] ? "checked" : "" %>>
    <label style="margin-right: 5px" for=<%= inputTypes[i] %>><%= inputType[i] %></label>
    <% } %>
    <% } %>
  </div>
  <div class="form-group checkbox-group required" id="features">
    <% if (features.length) { %>
    <% for(var x=0; x < features.length; x++) { %>
    <input style="margin-right: 5px" type="checkbox" id=<%= features[x] %> name="features" value=<%= features[x] %> <%= isSelected[x] ? "checked" : "" %>>
    <label style="margin-right: 5px" for=<%= features[x] %>><%= features[x] %></label>
    <% } %>
    <% } %>
  </div>
  <div class="form-group" style="display:none; margin-left: 20px" id="modelGroup">
    <input style="margin-right: 5px" type="checkbox" id="mlModel" name="mlModel" value="mlModel" <%= model.selected ? "checked" : "" %>>
    <label style="margin-right: 5px" id="modelLabel" for="mlModel"><%= model.label + model.ids[0] %></label>
    <div style="display:none; margin-left: 20px" id="mlEntGroup">
    <input style="margin-right: 5px" type="checkbox" id="mlEntities" name="mlEntities" value="mlEntities" <%= model.entities ? "checked" : "" %>>
    <label style="margin-right: 5px" for="mlEntities">Entities</label>
    </div>
    <div style="display:none; margin-left: 20px" id="mlRelGroup">
    <input style="margin-right: 5px" type="checkbox" id="mlRelations" name="mlRelations" value="mlRelations" <%= model.relations ? "checked" : "" %>>
    <label style="margin-right: 5px" for="mlRelations">Relations</label>
    </div>
  </div>
  <hr>
</div class="form-group">
```

```

<label for="content">Enter text or url</label>
<textarea class="form-control" id="body" name="body" placeholder="Enter text or an URL," rows="3" required><%= bodyText
%></textarea>
</div>
<div class="form-group">
  <input type="submit" id="submit" value="Post entry" class="btn btn-primary">
  <input type="button" id="reload" value="Page reset" class="btn btn-primary">
</div>
</form>
<p><small>Result:</small></p>
<div><pre id="result"><%= result %></pre></div>
<script>
  $( document ).ready(function() {
    $('#features').click(checkFeatures);
    $('#modelGroup').click(checkModel);
    $('#reload').click(function () {
      window.location.replace("/");
    });
  });
</script>
<%= include footer %>

```

Step 29 To control the checkboxes and the section where the Machine Learning model is displayed some jQuery helper functions are needed. In the project root folder **create** a **js** folder and then **create** a **jqhelpers.js** file with the following content: → (public/js/jqhelpers.js). **Save** and **close** the file.

```

function documentReady() {
  $('#mlEntGroup').toggle(false);
  $('#mlRelGroup').toggle(false);
  checkFeatures();
  if (($('#modelLabel').text().indexOf("Model:") != 0) && ($('#modelLabel').text().indexOf("no model") != 0)) {
    $('#modelGroup').toggle(false);
    getModel();
  }
}
function checkFeatures() {
  if ($('#div.checkbox-group.required :checkbox:checked').length == 0 ) {
    $('#submit').prop('disabled', true);
  } else {
    $('#submit').prop('disabled', false);
    checkModel();
  }
  if ($('#modelLabel').text().indexOf("Model:") == 0) {
    if ($('#entities').is(':checked') || $('#relations').is(':checked')) {
      $('#modelGroup').toggle(true);
    } else {
      $('#modelGroup').toggle(false);
      $('#mlModel').prop("checked", false);
      $('#mlEntities').prop("checked", false);
      $('#mlRelations').prop("checked", false);
    }
  }
}
function checkModel () {
  if ($('#modelLabel').text().indexOf("Model:") == 0) {
    if ($('#mlModel').is(':checked') && !$('#mlEntities').is(':checked') && !$('#mlRelations').is(':checked')) {
      $('#submit').prop('disabled', true);
    } else {

```

```

        $('#submit').prop('disabled', false);
    }
    if ($("#mlModel").is(':checked') && $("#entities").is(':checked')) {
        $('#mlEntGroup').toggle(true);
    } else {
        $('#mlEntGroup').toggle(false);
        $('#mlEntities').prop("checked", false);
    }
    if ($("#mlModel").is(':checked') && $("#relations").is(':checked')) {
        $('#mlRelGroup').toggle(true);
    } else {
        $('#mlRelGroup').toggle(false);
        $('#mlRelations').prop("checked", false);
    }
    } else {
        $('#modelGroup').toggle(false);
    } }
function getModel() {
    $.get( "/getModel", function( data ) {
        if (data.hasOwnProperty('modelId')) {
            $("#modelLabel").text("Model: " + data.modelId);
            $('#modelGroup').toggle(true);
        } else {
            $("#result").text(data.text);
        }
    });
}

```

Step 30 In the terminal you can now start the app with the following command:

```
node app.js
```

The screenshot shows a terminal window with the following content:

```

37 | response.status(400).send("All fields required!");
    |
    | PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: node
    |
kpsMBP-2:nlutester kps$ node app.js
NLU Tester started on port 3000.

```

Step 31 In the Browser load the index.js with the url

<http://localhost:3000>

The screenshot shows a web browser window with the title 'IBM EAG NLU-Tester' and the address 'localhost:3000'. The page has a header with the title 'IBM EAG NLU-Tester' and a subtitle 'Analyze text with IBM Watson Natural Language Understanding service'. Below the subtitle, there are several checkboxes for configuration: 'url' and 'text' (selected), 'categories', 'concepts', 'emotion', 'entities' (checked), 'keywords', 'metadata', 'relations' (checked), and 'semantic_roles'. There is also a section for 'Model' with a dropdown menu showing 'e96363d0-859e-4e80-ba90-8cbc98d9e564' and sub-options 'Entities' (checked) and 'Relations'. A text input field contains the sentence: "NCR, which counts later IBM Thomas Watson as one of its early employees, said its products and services account for more than \$400 billion in annual commerce and 23 billion consumer selfservice transactions". Below the input field are two buttons: 'Post entry' and 'Page reset'. A 'Result:' label is followed by a large empty text area. At the bottom, it says 'IBM Ecosystem Advocacy Group – 2018'.

IBM EAG NLU-Tester

Analyze text with IBM Watson Natural Language Understanding service

☐ url ☒ text

☐ categories ☐ concepts ☐ emotion ☒ entities ☐ keywords ☐ metadata ☒ relations ☐ semantic_roles

☒ Model: e96363d0-859e-4e80-ba90-8cbc98d9e564

☒ Entities

☐ Relations

Enter text or url

"NCR, which counts later IBM Thomas Watson as one of its early employees, said its products and services account for more than \$400 billion in annual commerce and 23 billion consumer selfservice transactions"

Post entry Page reset

Result:

IBM Ecosystem Advocacy Group – 2018

The model section only appears, when you have a machine learning model deployed via Watson Knowledge Studio. See the [demo document](#) on how to create and deploy a model. This demo will be shown at the end of the workshop.

Step 32 F.e. past the following text into the textarea and see the result

Bob Dylan won the Nobel Prize in Literature in 2016. Bob Dylan was born in Duluth,Minnesota.

A formatted json view is displayed in the *Result* field.

IBM EAG NLU-Tester

Analyze text with IBM Watson Natural Language Understanding service

☐ url ☒ text

☐ categories ☐ concepts ☐ emotion ☒ entities ☐ keywords ☐ metadata ☐ relations ☐

semantic_roles

☐ Model: e96363d0-859e-4e80-ba90-8cbc98d9e564

Enter text or url

Bob Dylan won the Nobel Prize in Literature in 2016. Bob Dylan was born in Duluth,Minnesota.

Post entry

Page reset

Result:

```
{
  "usage": {
    "text_units": 1,
    "text_characters": 92,
    "features": 1
  },
  "language": "en",
  "entities": [
    {
      "type": "Person",
      "text": "Bob Dylan",
      "relevance": 0.951339,
      "disambiguation": {
        "subtype": [
          "Composer",
          "MusicalArtist",
```

Step 33 By selecting the URL Radio button you can f.e. paste the url for a new article and see the result.

<http://www.bbc.com/news/technology-38595480>

Post entry

Page reset

Result:

```
{
  "usage": {
    "text_units": 1,
    "text_characters": 3084,
    "features": 1
  },
  "retrieved_url": "http://www.bbc.com/news/technology-38595480",
  "language": "en",
  "entities": [
    {
      "type": "Organization",
      "text": "AI",
      "relevance": 0.688013,
      "count": 3
    },
    {
      "type": "Organization",
      "text": "Carnegie Mellon University",
      "relevance": 0.659662,
      "disambiguation": {
        "subtype": [
          "Location",
          "CollegeUniversity",
          "University"
        ],
        "name": "Carnegie Mellon University",
        "dbpedia_resource": "http://dbpedia.org/resource/Carnegie_Mellon_University"
      },
      "count": 3
    },
    {
      "type": "Person",
      "text": "Tuomas Sandholm",
      "relevance": 0.631051,
      "count": 3
    },
    {
      "type": "Person",
      "text": "Claudico",
      "relevance": 0.628641,
      "count": 3
    },
    {
      "type": "Location",
      "text": "Pittsburgh",

```

Step 34 Select other features for your urls or text entries and see the results.

Step 35 With the **Page reset** button, you can re-initialize the page.

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.

Step 36 Create a manifest.yml file in the project folder with the following content: Use the name of you NLU service created in **Step 2c**). As name you should specify a unique name. There is also an option available to automatically generate unique names. You have to insert below your name:

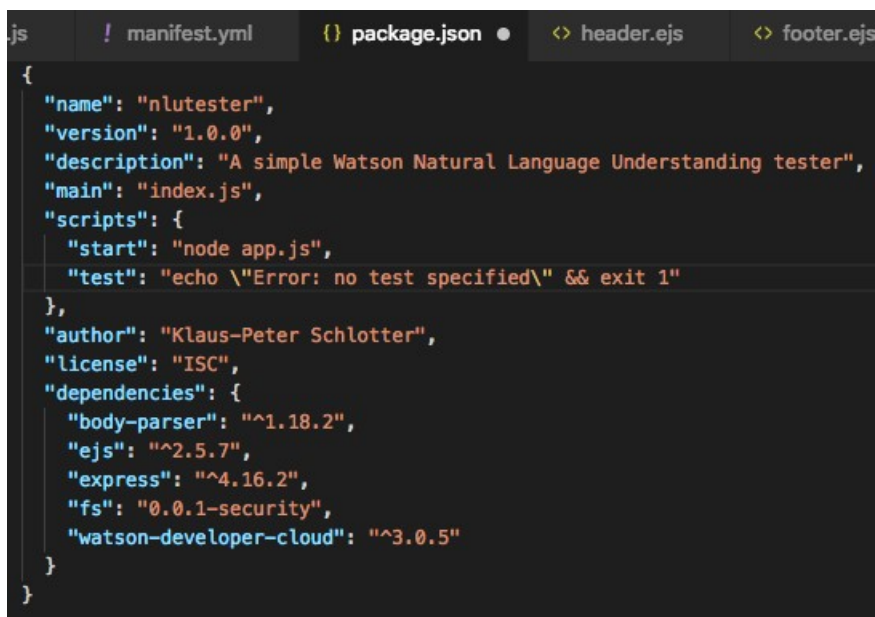
```
random-route: true
```

```
applications:  
- name: xxx-nlu-demo  
  path: .  
  buildpack: sdk-for-nodejs  
  command: node app.js  
  memory: 256M  
services:  
- my-nlu
```

Step 37 Save the manifest.yml file.

Step 38 We have to update the package.json file with a start script.

```
"start": "node app.js"
```



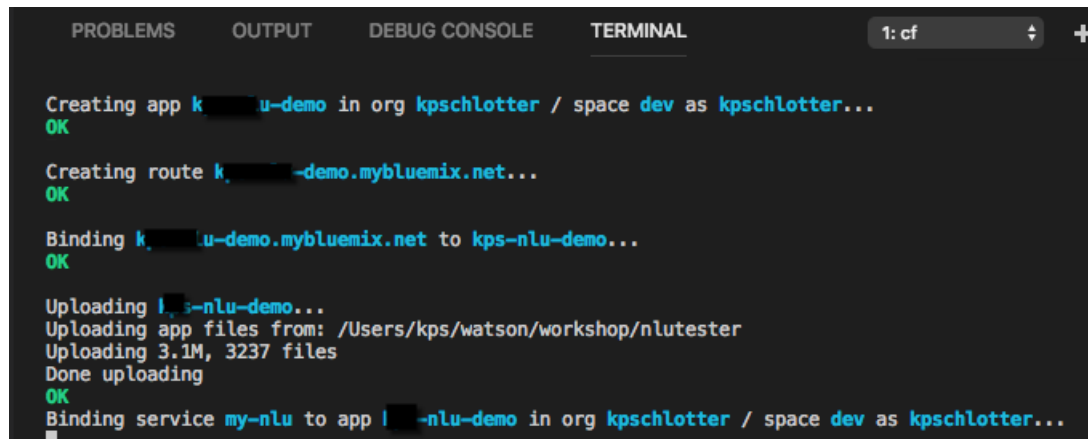
```
{  
  "name": "nluetester",  
  "version": "1.0.0",  
  "description": "A simple Watson Natural Language Understanding tester",  
  "main": "index.js",  
  "scripts": {  
    "start": "node app.js",  
    "test": "echo \\\"Error: no test specified\\\" && exit 1"  
  },  
  "author": "Klaus-Peter Schlotter",  
  "license": "ISC",  
  "dependencies": {  
    "body-parser": "^1.18.2",  
    "ejs": "^2.5.7",  
    "express": "^4.16.2",  
    "fs": "0.0.1-security",  
    "watson-developer-cloud": "^3.0.5"  
  }  
}
```

Step 39 Depending on your account, to avoid an “exceeding memory limit” error you should stop applications that are already running on your account.

Step 40 Make sure you are logged in to your IBM Cloud account.
(See [Workstation Setup](#) document Step 9)

Step 41 In the terminal (in the project folder) push your app to the IBM Cloud using the following command.

```
ibmcloud app push
```



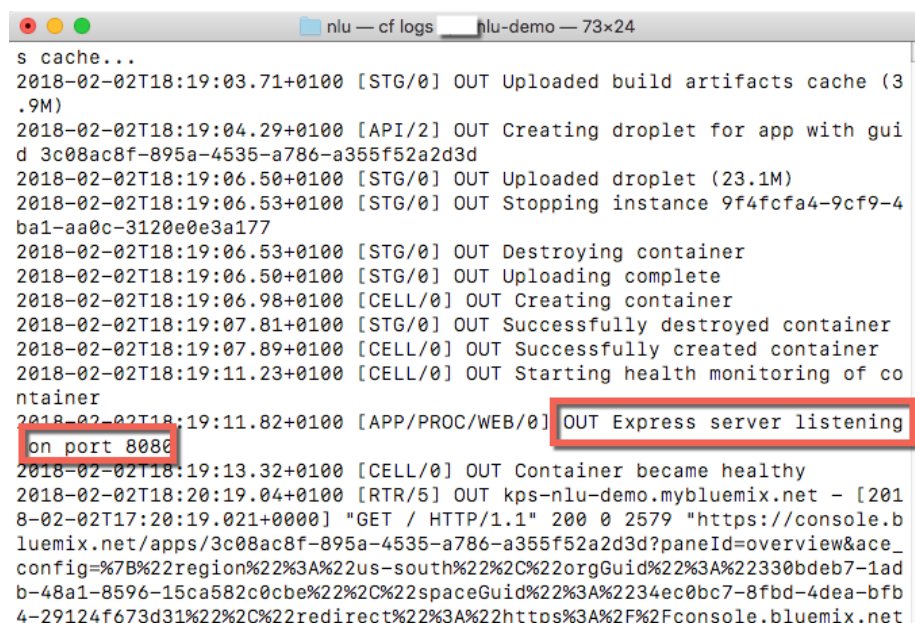
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: cf +
Creating app kpschlottter-u-demo in org kpschlottter / space dev as kpschlottter...
OK
Creating route kpschlottter-u-demo.mybluemix.net...
OK
Binding kpschlottter-u-demo.mybluemix.net to kps-nlu-demo...
OK
Uploading kps-nlu-demo...
Uploading app files from: /Users/kps/watson/workshop/nlutester
Uploading 3.1M, 3237 files
Done uploading
OK
Binding service my-nlu to app kps-nlu-demo in org kpschlottter / space dev as kpschlottter...

```

Step 42 You can display the server console from IBM Cloud in your Terminal with the following command

```
ibmcloud app logs xxx-nlu-demo
```



```

nlu — cf logs kps-nlu-demo — 73x24
s cache...
2018-02-02T18:19:03.71+0100 [STG/0] OUT Uploaded build artifacts cache (3.9M)
2018-02-02T18:19:04.29+0100 [API/2] OUT Creating droplet for app with guid 3c08ac8f-895a-4535-a786-a355f52a2d3d
2018-02-02T18:19:06.50+0100 [STG/0] OUT Uploaded droplet (23.1M)
2018-02-02T18:19:06.53+0100 [STG/0] OUT Stopping instance 9f4fcfa4-9cf9-4ba1-aa0c-3120e0e3a177
2018-02-02T18:19:06.53+0100 [STG/0] OUT Destroying container
2018-02-02T18:19:06.50+0100 [STG/0] OUT Uploading complete
2018-02-02T18:19:06.98+0100 [CELL/0] OUT Creating container
2018-02-02T18:19:07.81+0100 [STG/0] OUT Successfully destroyed container
2018-02-02T18:19:07.89+0100 [CELL/0] OUT Successfully created container
2018-02-02T18:19:11.23+0100 [CELL/0] OUT Starting health monitoring of container
2018-02-02T18:19:11.82+0100 [APP/PROC/WEB/0] OUT Express server listening on port 8080
2018-02-02T18:19:13.32+0100 [CELL/0] OUT Container became healthy
2018-02-02T18:20:19.04+0100 [RTR/5] OUT kps-nlu-demo.mybluemix.net - [2018-02-02T17:20:19.021+0000] "GET / HTTP/1.1" 200 0 2579 "https://console.bluemix.net/apps/3c08ac8f-895a-4535-a786-a355f52a2d3d?paneId=overview&ace_config=%7B%22region%22%3A%22us-south%22%2C%22orgGuid%22%3A%22330bdeb7-1adb-48a1-8596-15ca582c0cbe%22%2C%22spaceGuid%22%3A%2234ec0bc7-8fbd-4dea-bfb4-29124f673d31%22%2C%22redirect%22%3A%22https%3A%2F%2Fconsole.bluemix.net

```

Step 43 Now open the browser with the URL pointing to your app in the IBM Cloud

<https://xxx-nlu-demo.mybluemix.net>

IBM EAG NLU-Tester

Analyze text with IBM Watson Natural Language Understanding service

☐ url ☒ text

☐ categories ☐ concepts ☐ emotion ☒ entities ☐ keywords ☐ metadata ☒ relations ☐ semantic_roles

☒ Model: e96363d0-859e-4e80-ba90-8cbc98d9e564

☒ Entities

☐ Relations

Enter text or url

"NCR, which counts later IBM Thomas Watson as one of its early employees, said its products and services account for more than \$400 billion in annual commerce and 23 billion consumer selfservice transactions"

Post entry Page reset

Result:

IBM Ecosystem Advocacy Group - 2018

Step 44 Analyze the text from Step 35

IBM EAG NLU-Tester

Analyze text with IBM Watson Natural Language Understanding service

☐ url ☒ text

☐ categories ☐ concepts ☐ emotion ☒ entities ☐ keywords ☐ metadata ☐ relations ☐ semantic_roles

☐ Model: e96363d0-859e-4e80-ba90-8cbc98d9e564

Enter text or url

Bob Dylan won the Nobel Prize in Literature in 2016. Bob Dylan was born in Duluth, Minnesota.

Post entry Page reset

Result:

```
{
  "usage": {
    "text_units": 1,
    "text_characters": 92,
    "features": 1
  },
  "language": "en",
  "entities": [
    {
      "type": "Person",
      "text": "Bob Dylan",
      "relevance": 0.951339,
      "disambiguation": {
        "subtype": [
          "Composer",
          "MusicalArtist",
```