ChitChat Bot with Watson Assistant and IBM Functions

HandsOn Lab

By	Helen Kemnitz
Address	IBM-Allee 1, 71139 Ehningen
Telephone	+
E-mail	Helen.Kemnitz@de.ibm.com

and

By	Hartmut Seitter
Address	IBM-Allee 1, 71139 Ehningen
Telephone	+ 1707859522
E-mail	Hartmut.Seitter@de.ibm.com

5 June 2018

Table of contents

Table of figures

1 Introduction

1.1 Lab Overview

This Lab will give you an introduction on how a ChitChat Bot using **IBM Watson Assistant** (formerly Conversation Service), **IBM Functions** (OpenWhisk implementation of IBM) and a **REACT Javascript Application** can be build.

At first, you will deploy a Watson Assistant Service and build a basic conversation dialog. In the appendix you can additionally find out how the **Watson Discovery service** can be integrated which provides data collected document of a specific topic that extend the 'standard dialog' capabilities.

To invoke the Watson services, you will implement IBM Cloud Functions which are connected to an OpenWhisk Sequence.

A small React Application will build the user interface to interact with the user.

Moreover, a skeleton of training data for the Conversation and the Discovery Service will be provided also for the IBM Cloud Functions and React application. However, the skeleton can be extended so that you can build your own ChitChat flavour.

After the lap you will have a basic understanding of:

- The IBM Watson Assistant service and how training data is used to build a conversation dialog
- How to build IBM Functions actions and invoke Watson Services
- How to interact with a REACT User Frontend application based on JavaScript.

Advanced developer can extend the ChitChat e.g. to implement multilanguage support.

1.2 The Architectural overview Diagram of the Lap Application

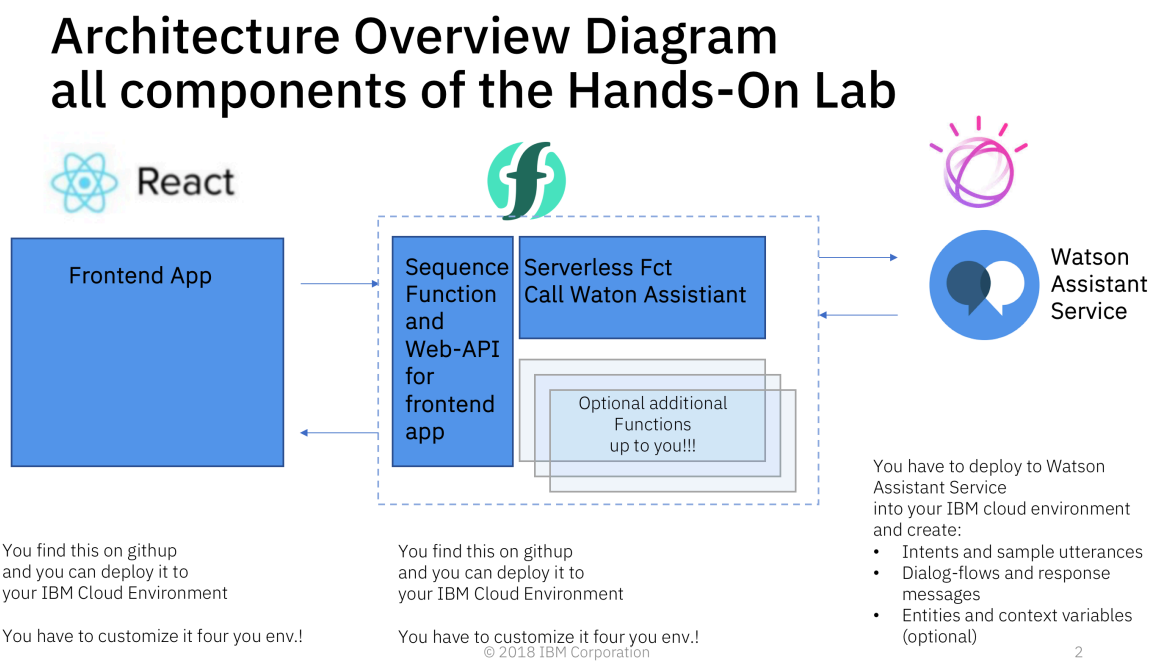


Figure 1 Architecture Overview Diagram – All components of the Hands-On Lab

The goal of this lab is to build an end-to-end application which can interact with the Watson Assistant Service
to get a basic understanding of a Chat-Bot Application and a user dialog with 'AI based service'.

Implementing all parts mentioned above will not be possible in the time available for this lab.

- Therefore, we recommend starting with the **Watson Assistant Service** which you can use to build a basic dialog. How to launch the tool and get started is described in the following chapter. After completing this part, you will have a basic understanding for the Watson Assistant Service and you can test different user dialogs directly in the Launch Tool environment.
  For details see chapter Let's start with the Watson Assistant Service

- To move forward we want to build an 'end-to-end' application for which you need an interface program to interact with the Watson Assistant passing questions from an end-user to the service and handling the return information of the service. This can be done by implementing a 'Serverless function'. In this lab we use **IBM Functions** (also part of IBM cloud) to interact with the Watson service.
  A sample function is available on Github which you can use to interact with your Watson Assistant Service from step one.
  For details see chapter Now we need a Backend Application Function

- To enrich the two parts *'Watson Assistant Service – handling questions and responses'* and *'IBM Serverless Function to interact with the Watson Service'* you can use a front-end application based on NodeJS, JavaScript and **REACT** which is also available on Github.
  For details see chapter xxxx

2 Let's start with the Watson Assistant Service

2.1 Watson Assistant Service Basics for this Session

The following diagram shows the overall architecture of a complete solution:

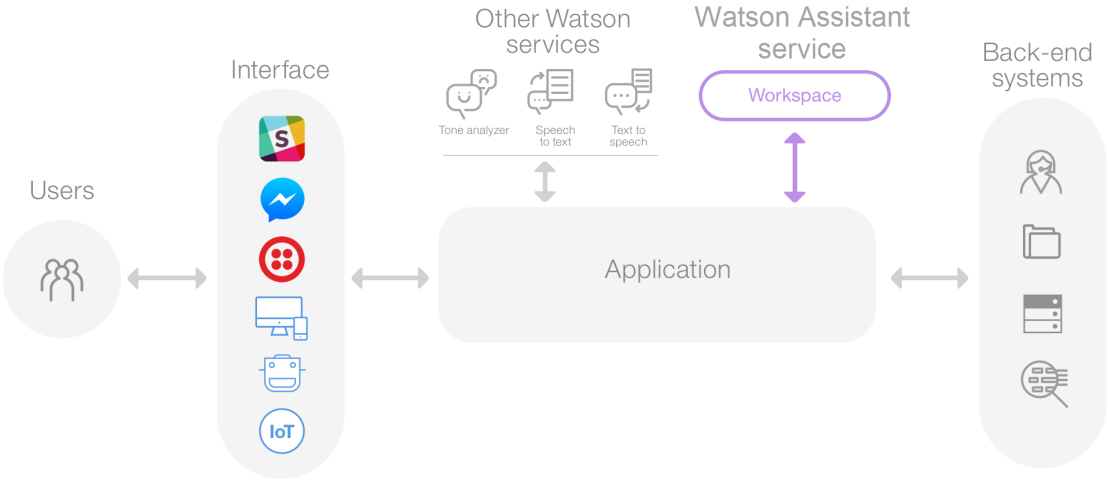
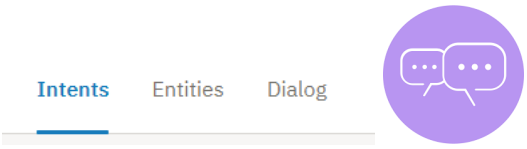


Figure 2 Create a Service Instance of Watson Assistant on IBM Cloud

https://console.bluemix.net/docs/services/conversation/index.html#about

Watson Assistant is an IBM Cloud service and consists of different workspaces. Each workspace consists of **intents**, **entities** and a **dialog**.



Intents are purposes or goals expressed in a customer's input, such as answering a question or processing a bill payment. By recognizing the intent expressed in a customer's input, the Conversation service can choose the correct dialog flow for responding to it. Intents are marked with a #.

Entities represent a class of object or a data type that is relevant to a user's purpose. By recognizing the entities that are mentioned in the user's input, the Conversation service can choose the specific actions to take to fulfil an intent. Entities are marked with a @.

The dialog uses the intents and entities that are identified in the user's input, plus context from the application, to interact with the user and ultimately provide a useful response.

The dialog flow is represented graphically in the tool as a tree. You can add a branch to process each of the intents that you want the service to handle. You can then add branch nodes that handle the many possible permutations of a request based on other factors, such as the entities found in the user input or information that is passed to the service from your application or another external service.

2.2 *The Watson Assistant Service for this Scenario*

To work with Watson Assistant Service, you need to prepare your data. After that you can define intents, sample utterances, and entities. You can then build a dialog and test your conversation service with the “Try Out” function.

2.2.1 How to prepare your Watson Assistant Service

Step 1: Create an account on IBM Cloud.
Sign-up for free: https://console.eu-gb.bluemix.net/registration/?target=%2Fdeveloper%2Fwatson%2Fdashboard
You'll receive an email to confirm and activate your account.

Step 2: After you activate your account and log in, click Watson Services from the Watson console.

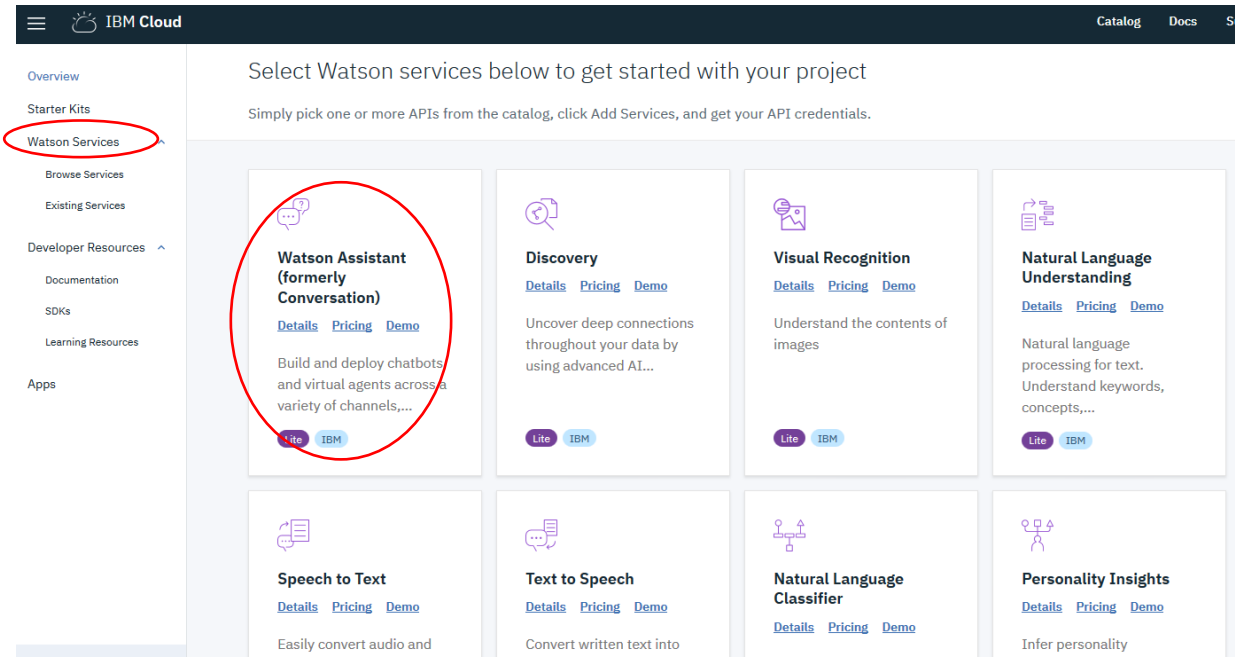


Figure 3 Create a Service Instance of Watson Assistant on IBM Cloud

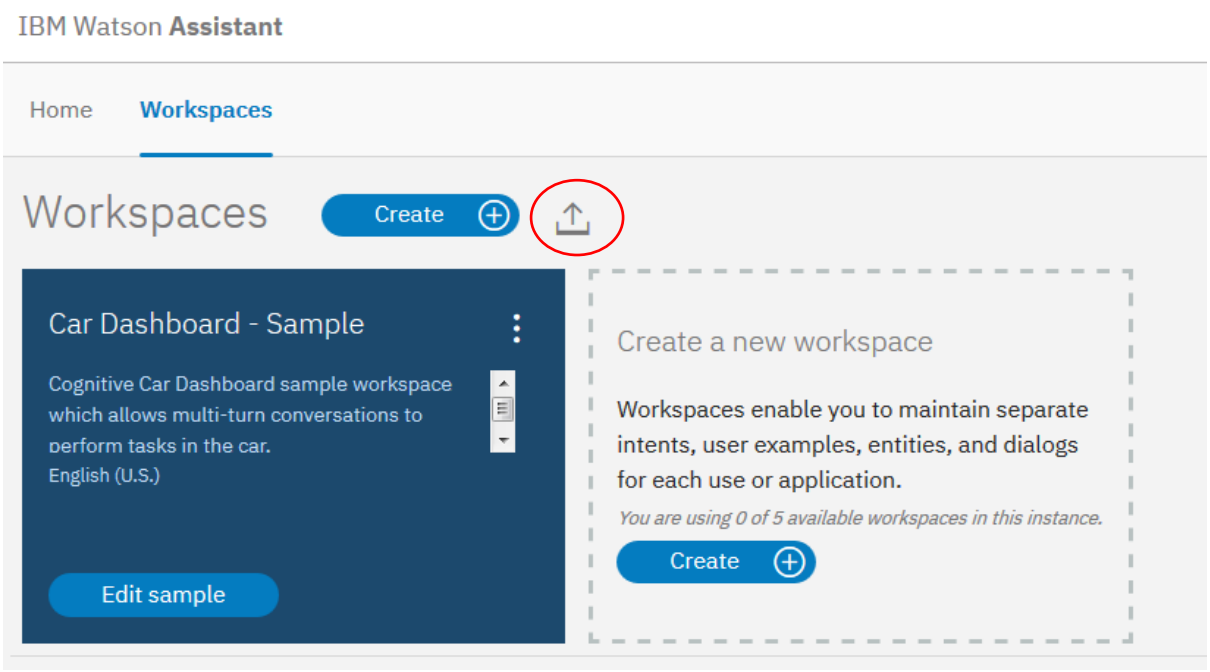
Step 3: Create an instance of the Watson Assistant Service.

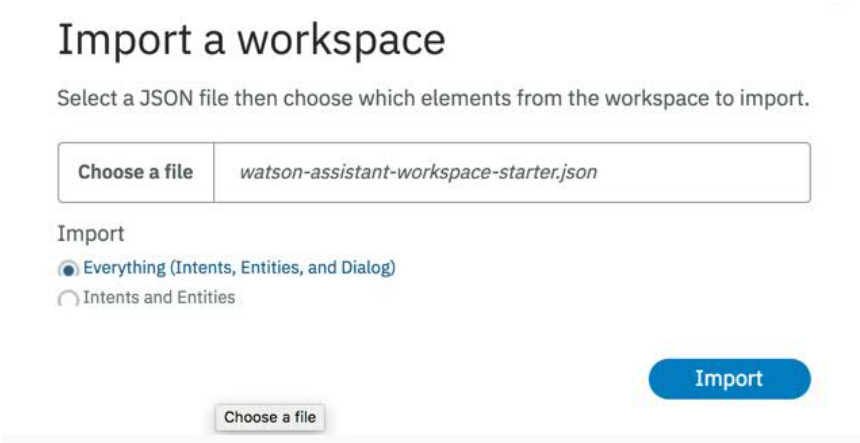
Step 4: Launch tool in the new created service in the Cloud Dashboard.

Step 5: Now you can get stated and create a workspace

Step 6: Import workspace:
Download from GitHub **'watson-assistant-workspace-starter.json'** to your local file system and import it in Cloud

Use GitHub link:
https://github.com/HartmutSeitter/hs-rewire-ind-bus-chatbot/tree/master/assistant-ws





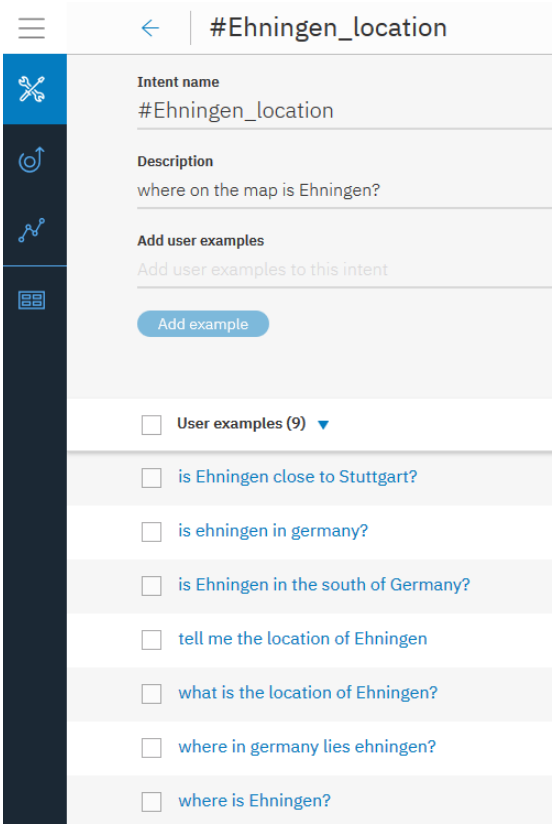
Step 7: Now you are ready to extent the workspace by your own intents, utterances and dialog.

2.2.2 Define Intents and Utterances

Determine what your virtual assistant will understand by providing training examples, so Watson can learn.

Define one intent for each goal that can be identified in a user's input.

First example: #Ehningen_location

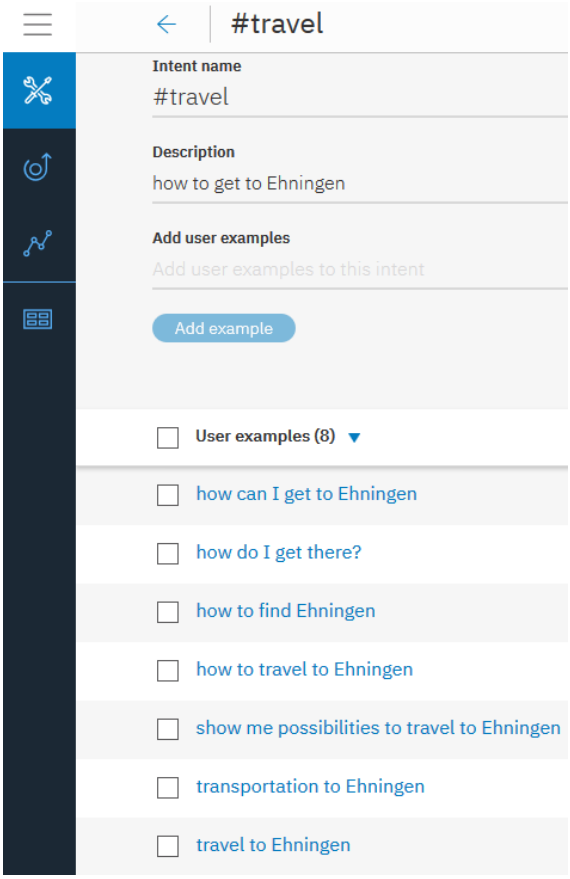


You might define an intent named: **'Ehningen_location'** that answers questions about where Ehningen is located.

For each intent, you add sample utterances that reflect the input customers might use to ask for the information they need, such as, **"Where on a map is Ehningen?"**.

The more user examples that are implemented, the better the response rate of the Conversation Service. It can be helpful to ask different people for their description of a topic and also dialects can be considered.

Second example: #travel



You now might define an intent named: **'travel'** that answers questions about how to travel to Ehningen.

Add user examples, such as, **"How can I get to Ehningen?"**.

In addition, add intents for three different options: #**car, #train, and #airplane**.

Again, add user exemples, such as, **"I want to travel by car"** etc.

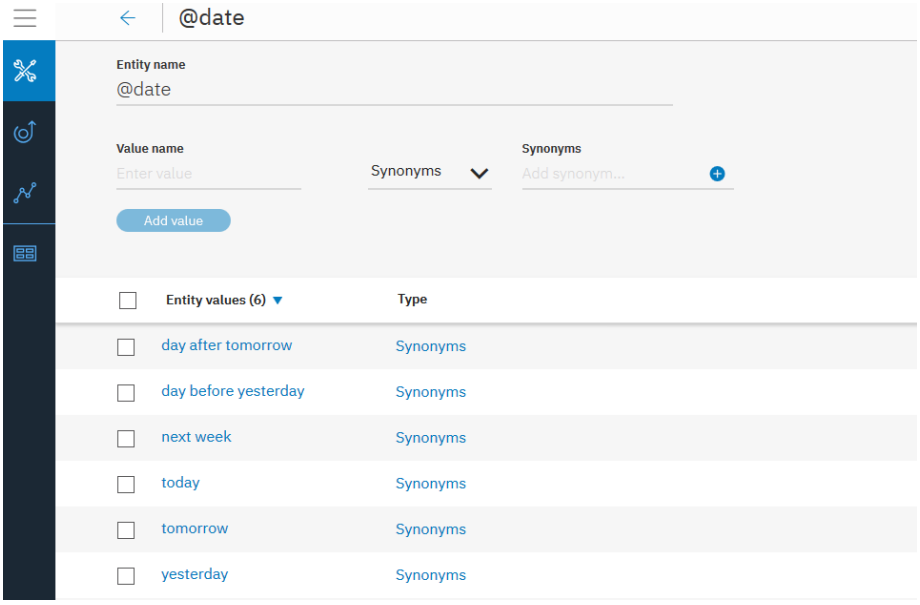
Third example: #yourownintent

Now you can get creative and create your own intent! Use as much user examples as you like.

2.2.3 Define Entities (optional)

Whereas intents point to a specific dialog flow, entities trigger the action or answer in the flow themselves.

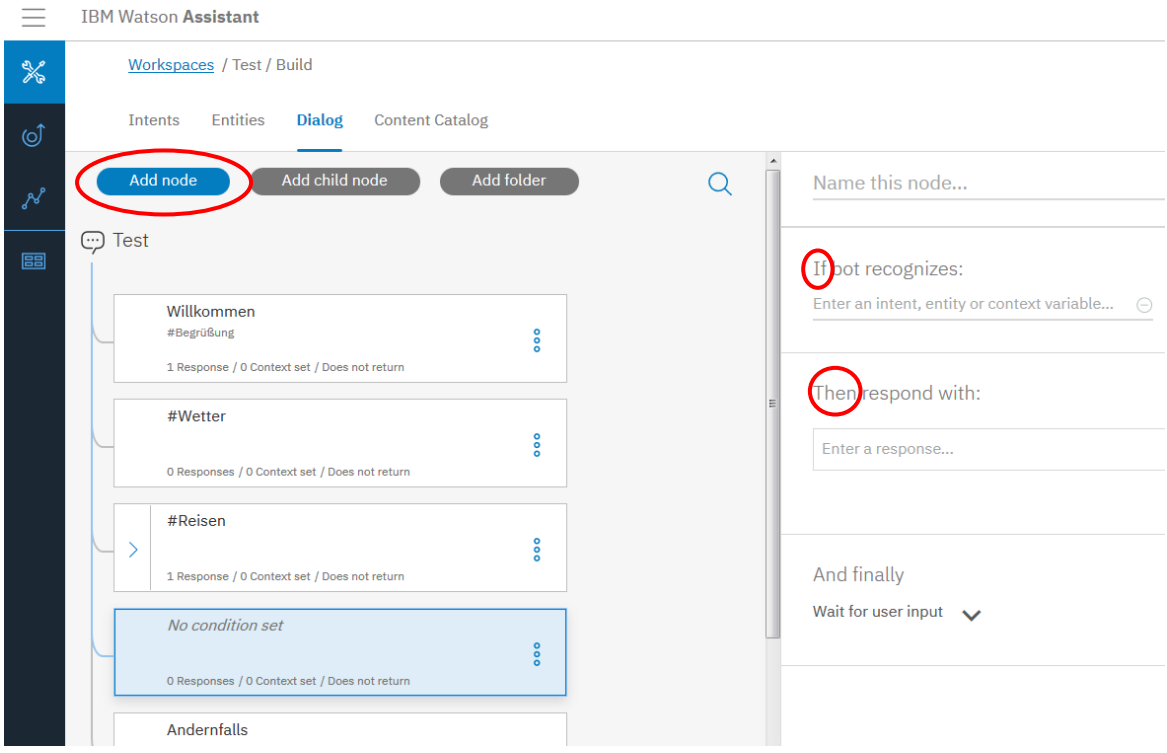
Since there are many terms and descriptions that share the same meaning, synonyms can be added to each item. In our workspace 'Rewire Industrial Business' you can find the entity @date as example:



Both intents and entities can also be imported from external sources and subsequently changed or deleted.

2.2.4 Define Dialogs

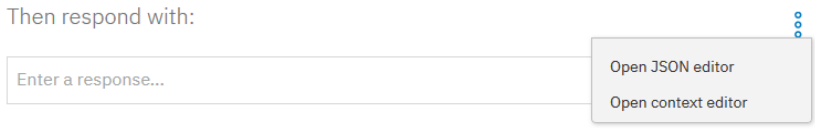
Utilize the intents and entities you created, so your virtual assistant responds appropriately. The dialog flow is represented graphically in the tool as a tree. You can add a node to process each of the intents that you want the service to handle. Each node consists of an **If-Then query**.



In the **If** part, it is queried which input (intent, context variable or entity) must be recorded in order to jump into the node.

If the input matches the condition specified in the If part, it is called a trigger. If the value of the If part is true, the dialog flow jumps to the Then part of the query.

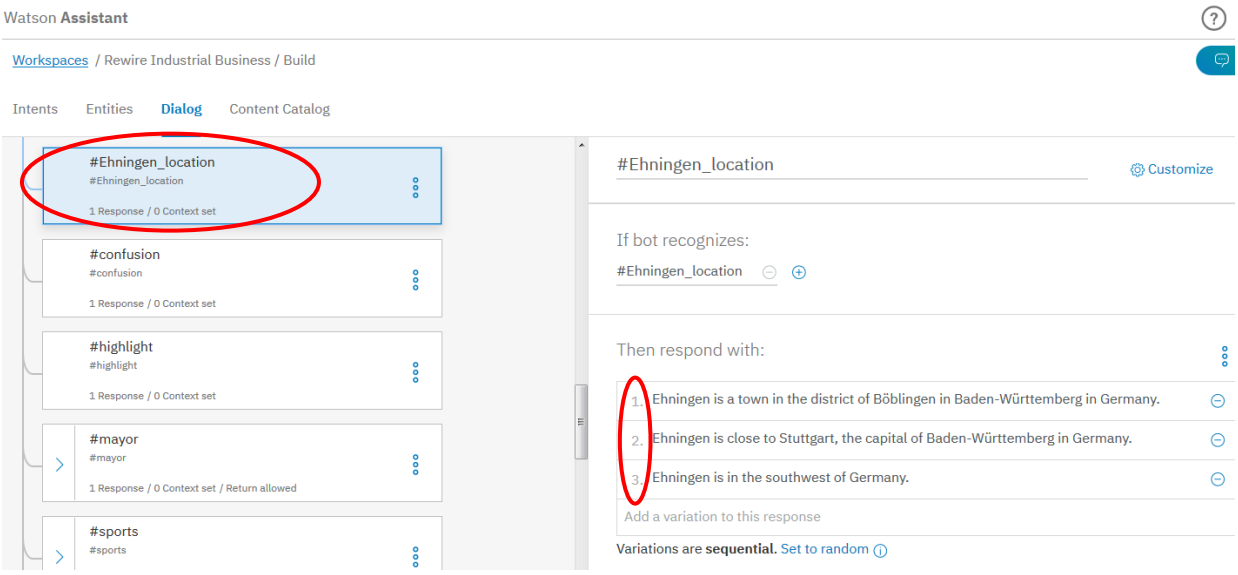
In the **Then** part of the node, an answer is assigned to the utterance. For simple questions, a response sentence can be implemented right away. You can also use the Context or the JSON editor to implement a response.



Now let's get started by using our 3 examples from above to continue: #Ehningen_location, #travel, #yourownintent. Click 'Add node' to create a dialog for each example.

First example: #Ehningen_location

For our fist example you can set variations of answer sentences. Watson then chooses one of the answers, either in a fixed order or randomly.

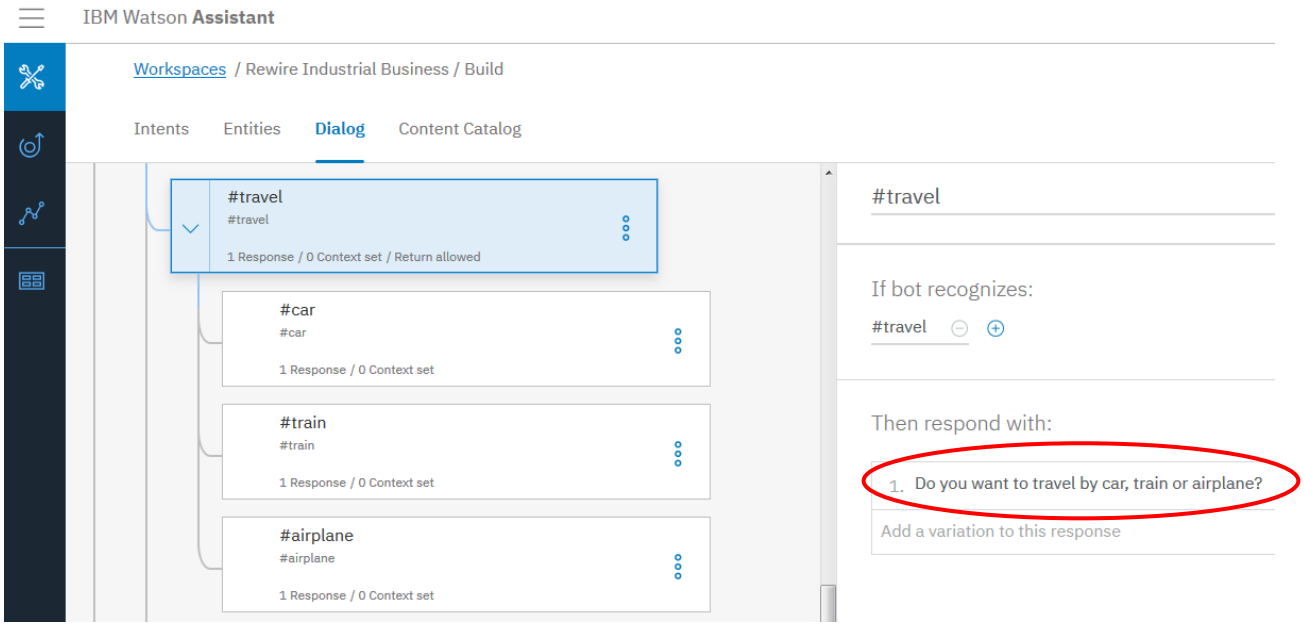


Second example: #travel

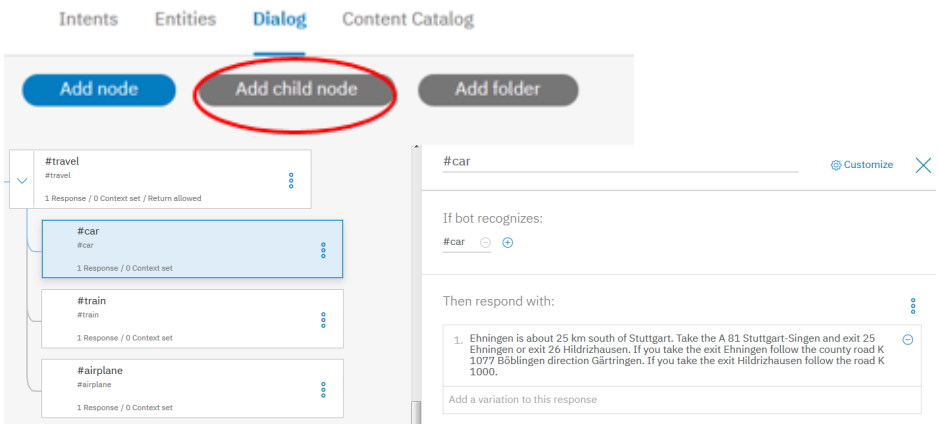
Often one simple answer is not enough, and it is necessary to start a more complex conversation. To implement this, the node can be connected to other nodes. The 'Then'

part then answers with a counter question and the possible answers to it are again implemented in the attached nodes in their condition. The attached nodes are called child nodes.

Let's use this function for our intent #travel and ask about the different options: car, train, airplane. First, create the node #travel and type in a counter question, such as, "Do you want to travel by car, train or airplane".



Now you can add child nodes for the three options: car, train, airplane, and type in a response sentence.



This makes it possible to design a dialog flow with many different branches. The dialogue can quickly become complex, so it must be remembered that the flow of dialogue always goes from top to bottom. It jumps to the first node that matches the trigger. Nodes below, which would also match the trigger, are then ignored.

It is also possible to implement a link or picture to your response sentence. For the child node "train" we can, for example, add the link:
http://www.fahrplanauskunft.de/

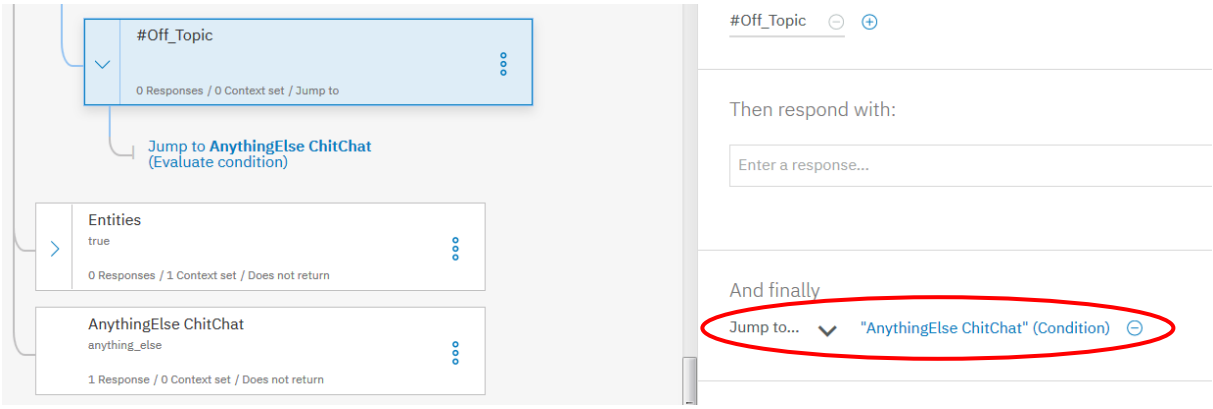
To implement the link open the JSON editor:

Third example: #yourownintent

Let's build a dialog flow for your own example.

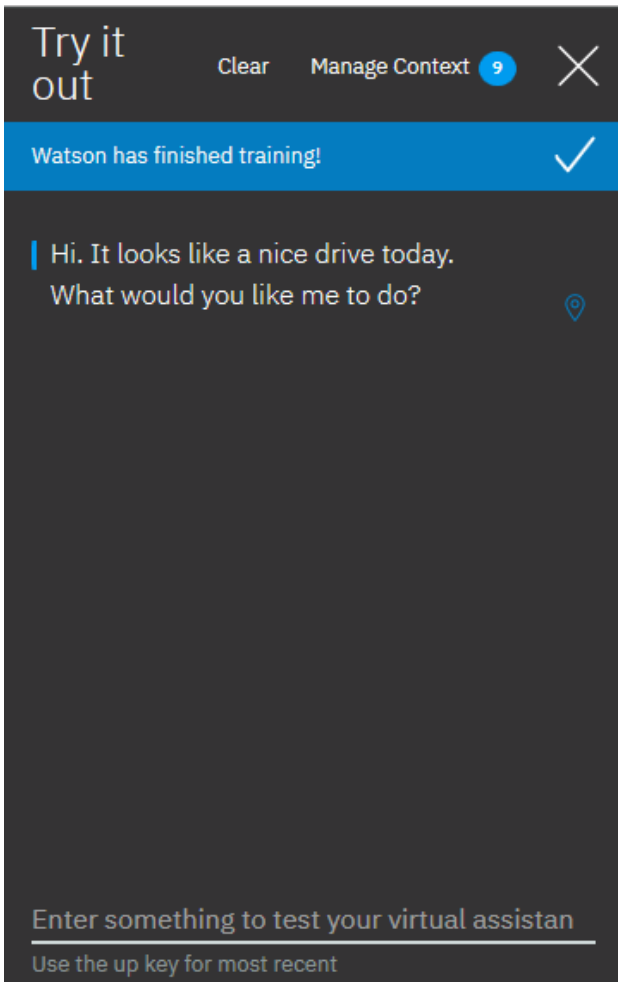
2.2.5 Jumps (optional)

Jumps between the various dialog strings are possible but must be explicitly stated. For example, the node "Anything_else" is set to give an answer to input that is not intercepted in the other nodes. You can use the "Jump to" functions to guide questions that are off topic (described in the intent #Off_Topic) to the "Anything_else" node.



2.2.6 Test the conversation using IBM Watson Assistant

Test your virtual assistant in the ‘Try it out’ section of the tool to see how it recognizes the intents and entities and how it responds first-hand.

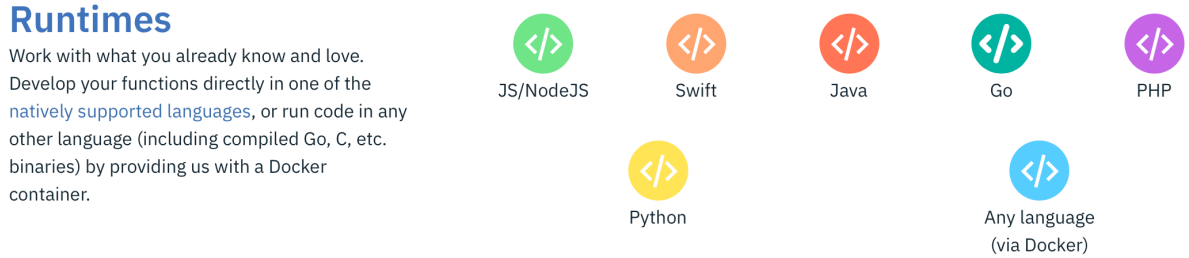


3 Now you need a Backend Application Functions

3.1 IBM Serverless Functions Basics for this Session

IBM Cloud Functions (based on Apache OpenWhisk) is a Function-as-a-Service (FaaS) platform which executes functions in response to incoming events and costs nothing when not in use.

.



See also Getting started with IBM Cloud Functions

In this Lab we use the JS/NodeJS runtime environment and the ‘watson-developer-cloud' library to build the interface for the Watson Services.

3.2 IBM Watson Assistant API Reference Basics for this Session

The complete API reference for the Watson Assistant Services is listed here: Watson Assistant API reference

In this session, however, we only use the Message API request to get a response to the user input.

Example request

```
var watson = require('watson-developer-cloud');

var assistant = new watson.AssistantV1({
  username: '{username}',
  password: '{password}',
  version: '2018-02-16'
});

assistant.message({
  workspace_id: '9978a49e-ea89-4493-b33d-82298d3db20d',
  input: {'text': 'Hello'}
},  function(err, response) {
  if (err)
    console.log('error:', err);
  else
    console.log(JSON.stringify(response, null, 2));
});
```

Figure 4 Watson Assistant API Reference Sample - Message

3.3 Build a Conversation Service for this Session

As you can see from the example above, the interface used to communicate with the Watson Assistant Service is very simple and only a view parameters and lines of code are needed to pass information to the service and to get back the response.

3.3.1 Collect the Parameters you need

Parameters you need:

- Username and password of the Watson Assistant Service which you can find here:

  Go to the dashboard of IBM cloud and select the Watson Assistant Service you deployed in chapter 2
  You should get something similar to this:

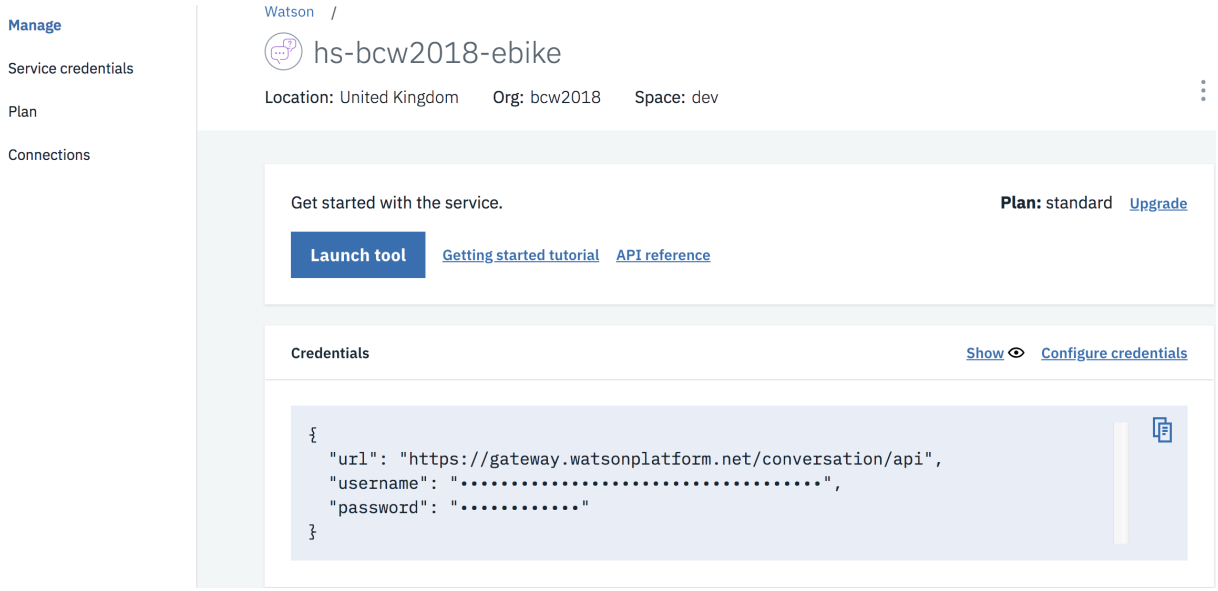


Figure 5 Watson assistant service credentials

- The workspace ID from the Watson assistant service which you can find here:

  Go to the dashboard of the Watson assistant service and select Launch tool, click on the ‘three dots to display additional options’

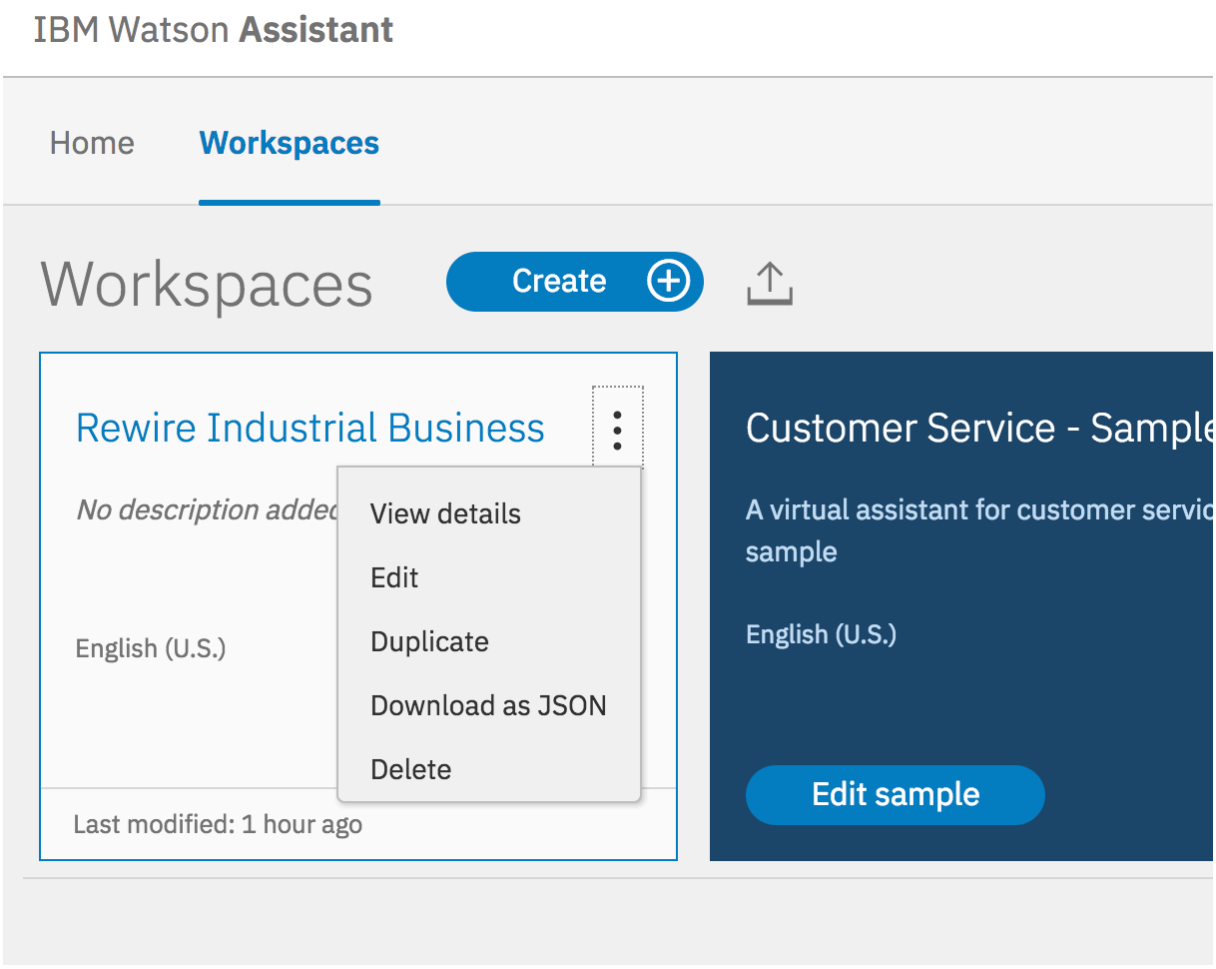


Figure 6 Watson Assistant Workstation ID-1

And now select view details: here you can find the workspace ID

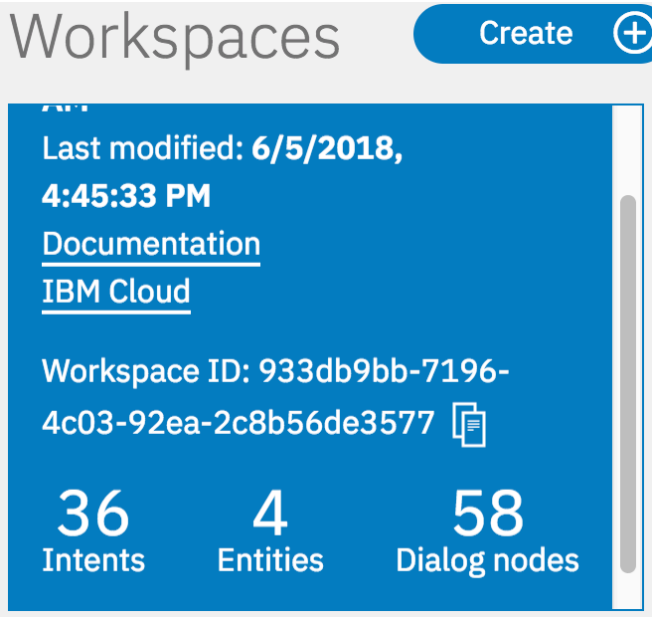


Figure 7 Watson Assistant Workstation ID-2

Now that we have all parameters available, let's create an 'Action in IBM Functions' to invoke the API.

3.3.2 Create an Action in IBM Function to invoke the Watson Assistant API

Step1: Select Functions in your IBM Cloud Dashboard

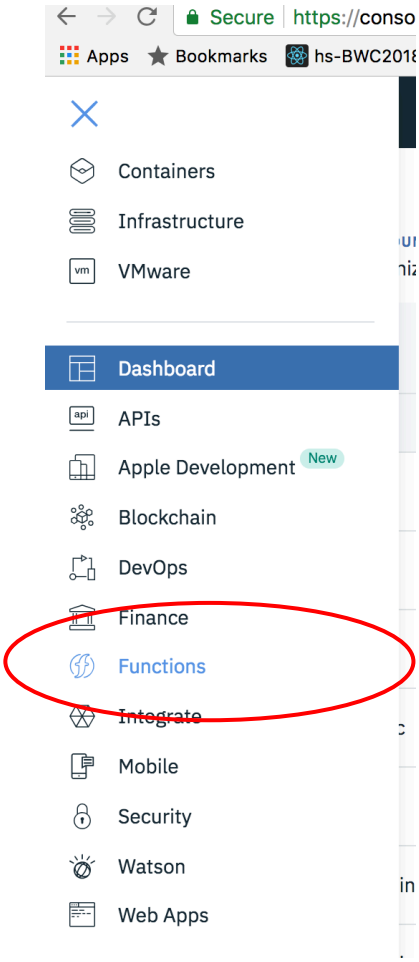


Figure 8 IBM Functions – create it

Step 2: Select Create Action

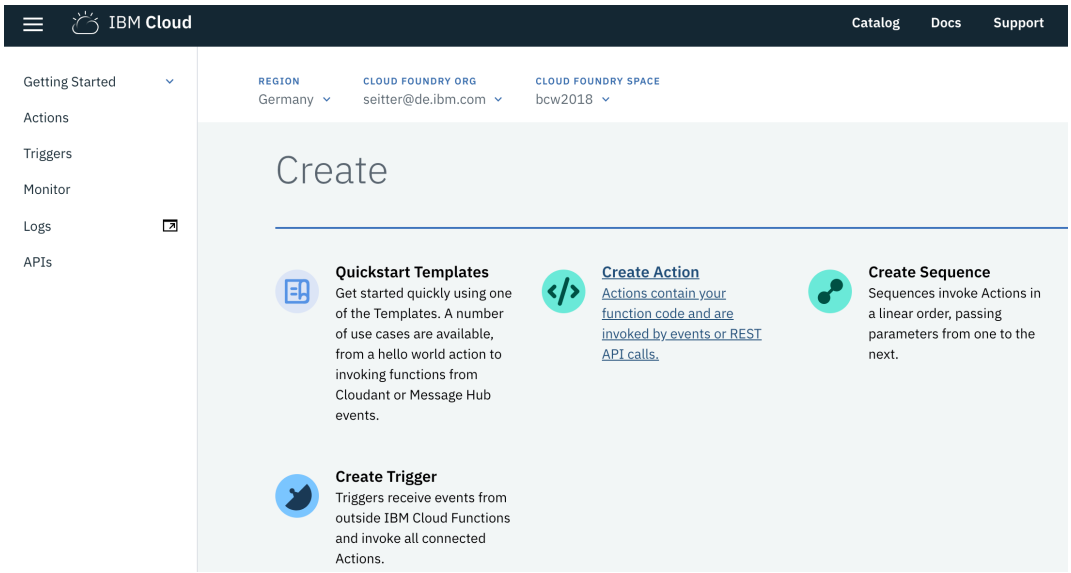


Figure 9 IBM Functions – create Action

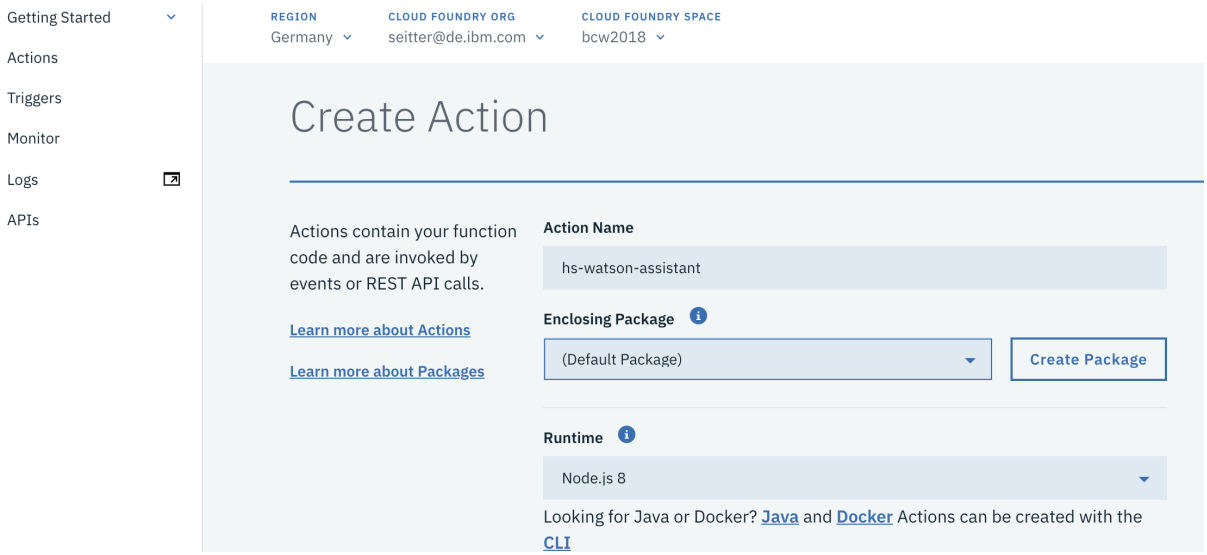


Figure 10 IBM Functions – create Action-2

Step 3: Now select parameters and define the parameter values (username, password and workstation ID) to avoid 'hard coding' it in the program code.

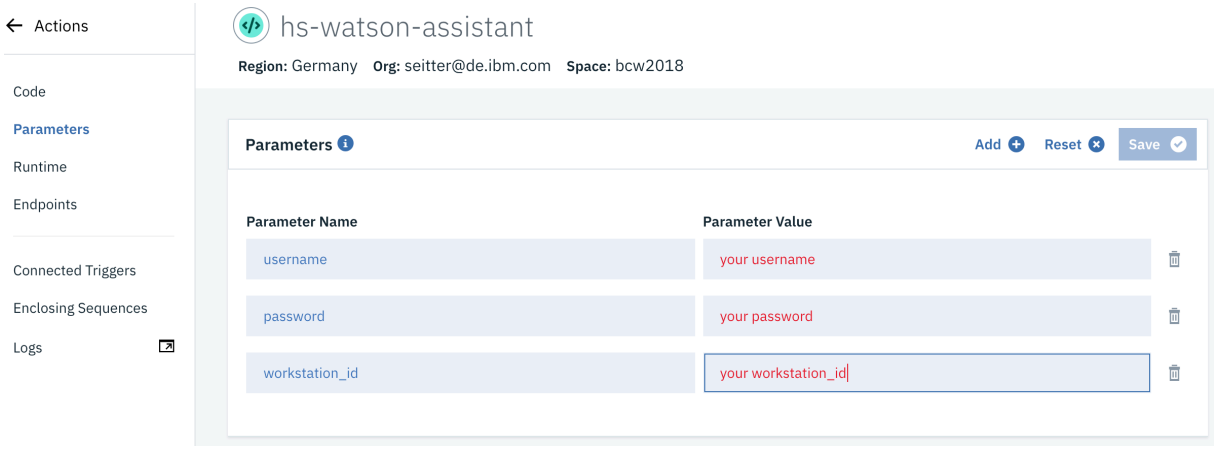
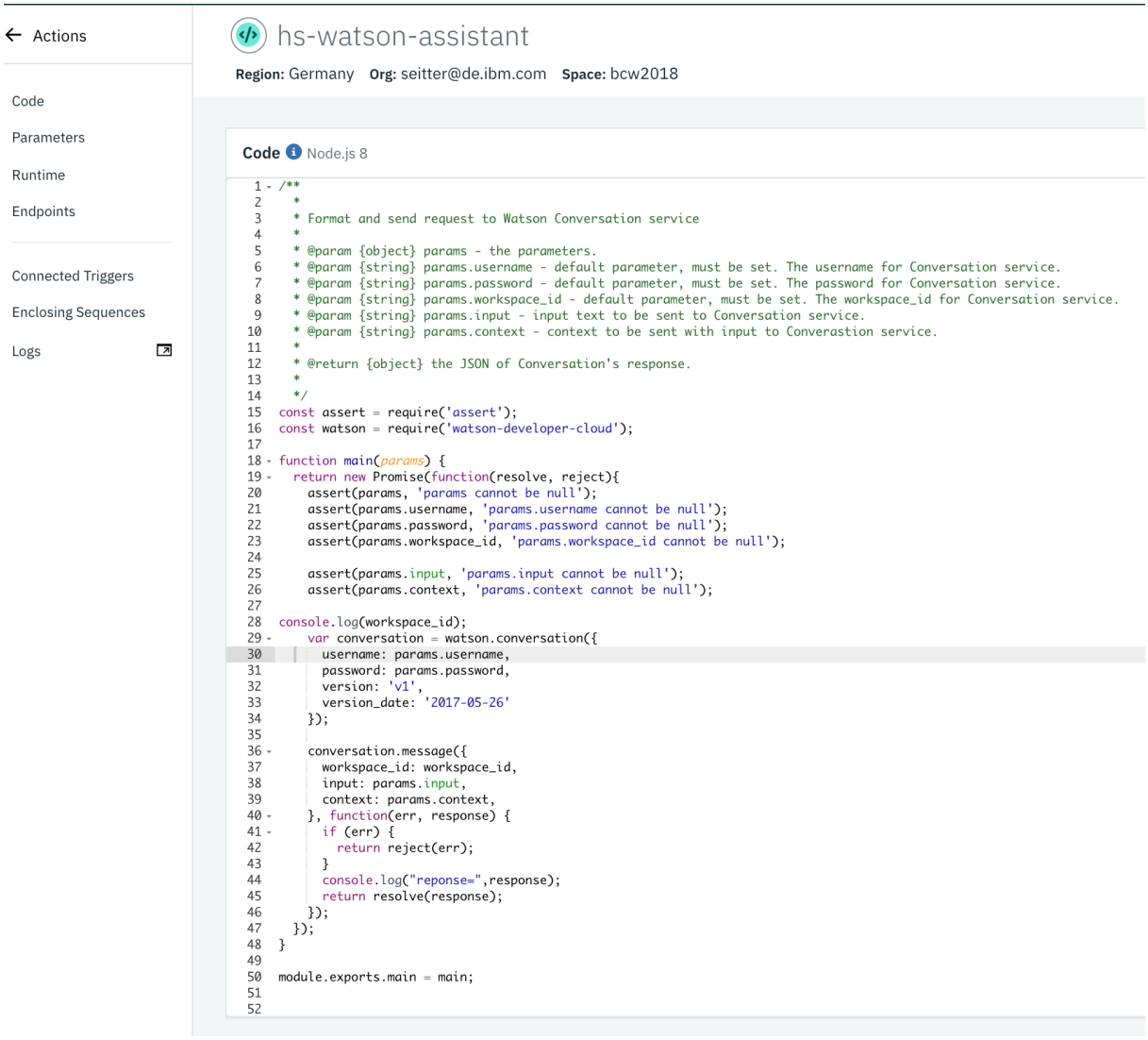


Figure 11 IBM Functions – Action – define the parameter needed to interact with Watson Service

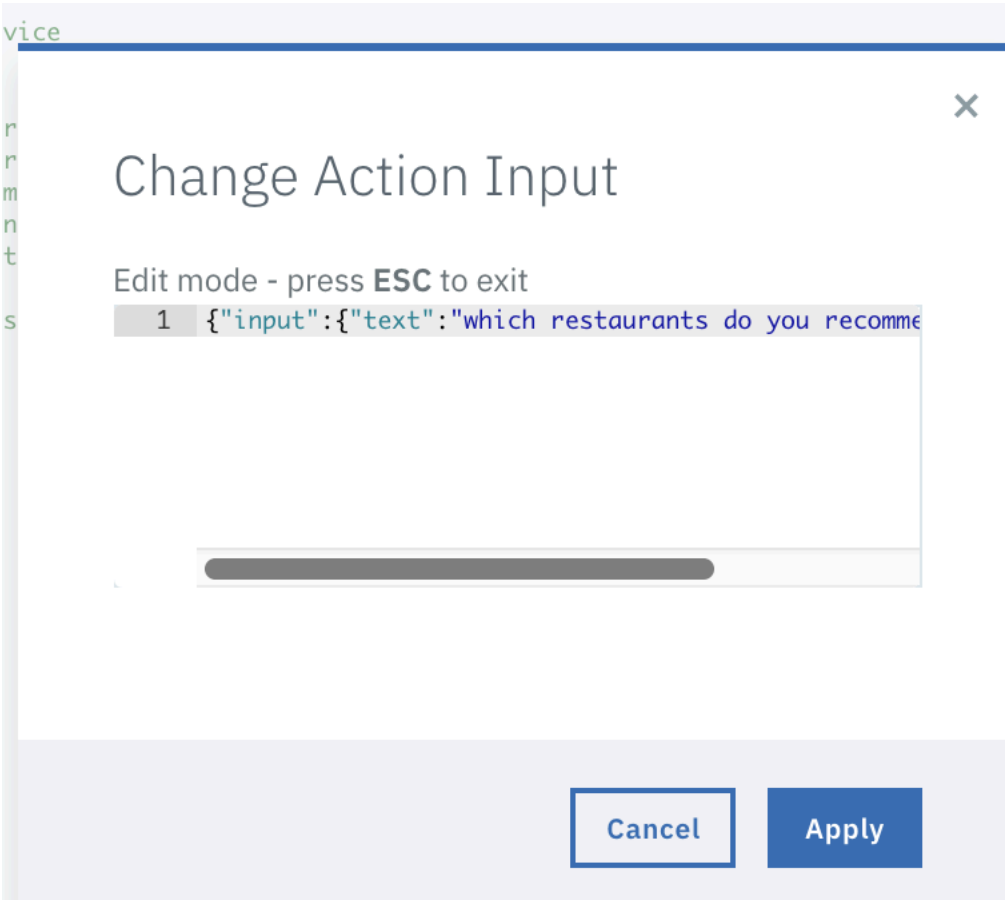
Step 4: Coding. The program code itself is quite simple and very similar to the example mentioned above. It should look like this one here:

A sample can be retrieved from Github

https://github.com/HartmutSeitter/hs-rewire-ind-bus-chatbot/blob/master/actions/hs-watson-assistant.js



Step 5: But before invoking the action, let's specify the input parameter.?



Step 5: After saving your code, you can test it and verify the result you retrieve form Watson Assistant.

You can also find the Input data on github

https://github.com/HartmutSeitter/hs-rewire-ind-bus-chatbot/blob/master/actions/input-data.json

3.3.3 Create a Sequence to be prepared to invoke more than one Action in this scenario

Even if is not really necessary to create a sequence to invoke a single action, let's do so.
Just in case you would like to extend the function of this application (e.g. to invoke other Watson services) or to save the conversation in a cloudant database a sequence function will be very helpful.

Step 1: Select 'Create Sequence'

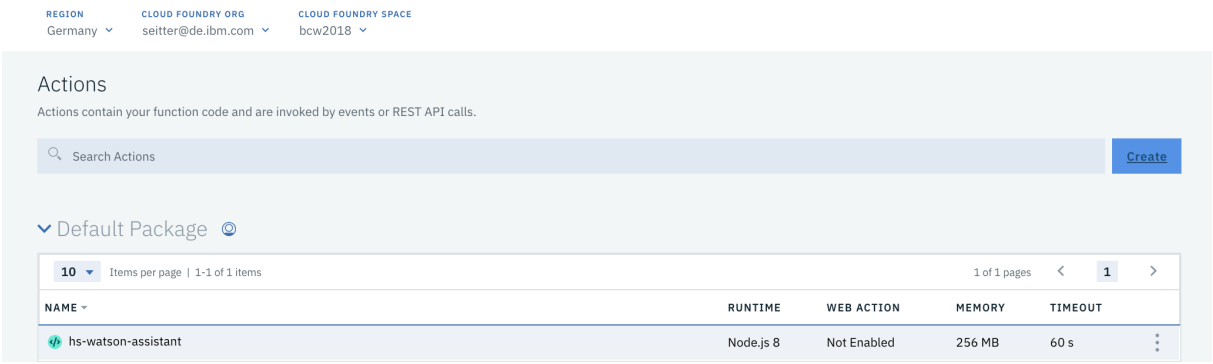


Figure 12 IBM Functions – create a sequence - 1

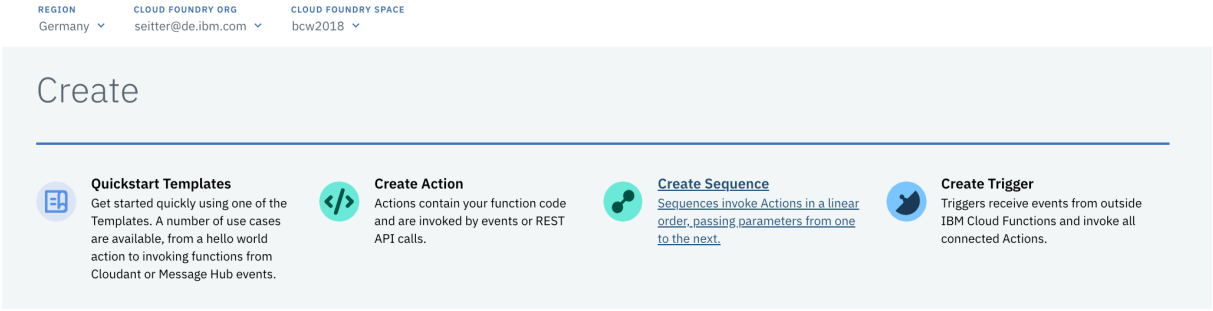


Figure 13 IBM Functions – create a sequence - 2

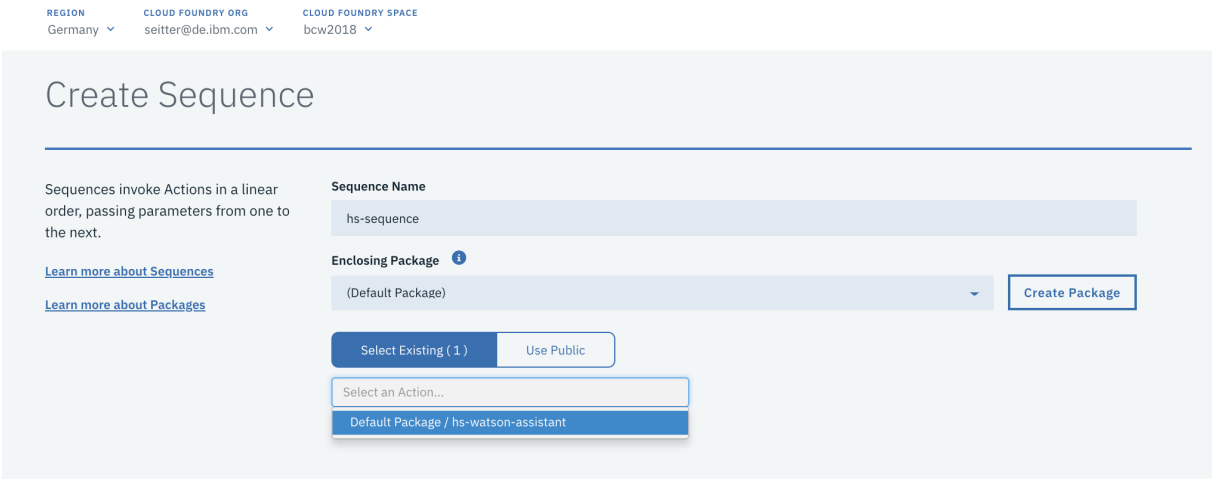


Figure 14 IBM Functions – create a sequence – 3

Step2: Now enable this sequence as a Web Action. This is necessary to call this sequence from the front-end application which we will deploy in the next chapter, or to use e.g. postman or curl command to interact with it.

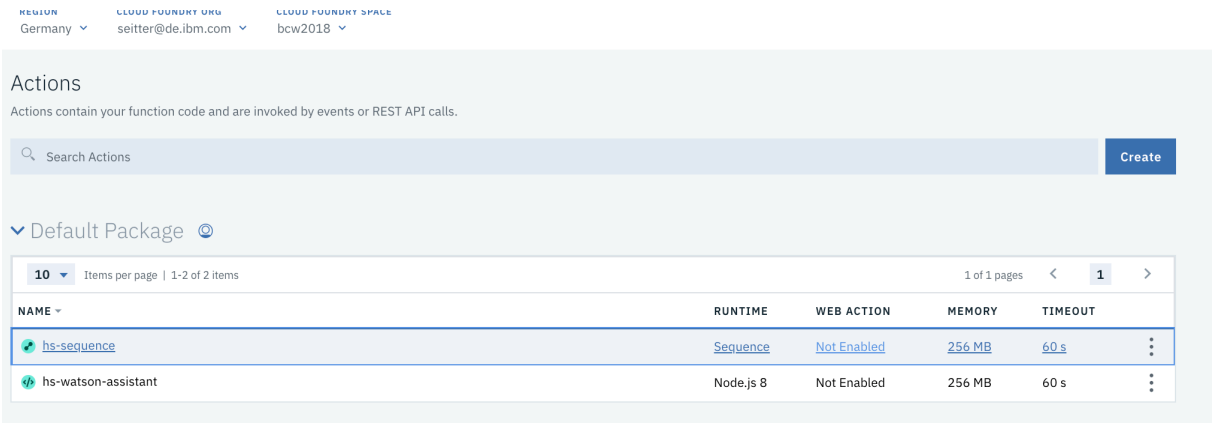


Figure 15 IBM Functions – Enable Web Action of the sequence- 1

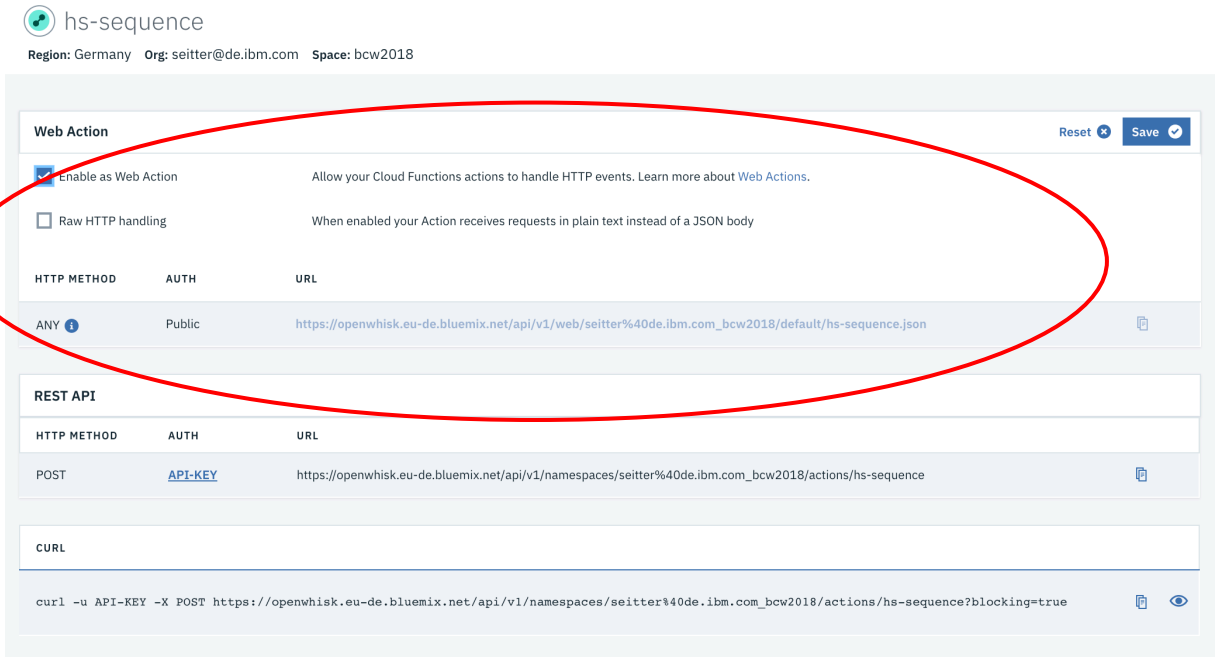


Figure 16 IBM Functions – Enable Web Action of the sequence- 2

4 Now you need a Frontend Application

There are different ways to implement a Frontend Application or Web Application and you should choose an environment which you are familiar.

For this session we have prepared a NodeJS – REACT application which can be used.

The complete application is available on github and you can directly (more or less) deploy it to IBM could.

4.1 Create from GitHub the Frontend Application

Go to github and on the README.md you find this blue button – Deploy to IBM Cloud

https://github.com/HartmutSeitter/hs-rewire-ind-bus-chatbot

a

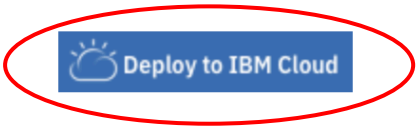
6. Copy the javascript code from Github and create a new Function and specify the pa
ws-id in the parameter section of the Action

7. Define a IBM Function Sequence and invoke the Action

8. Web enable the Sequence you created - save the url for later usage

Using a REACT App as a front end application

9. Deploy this REACT Application direct to Bluemix

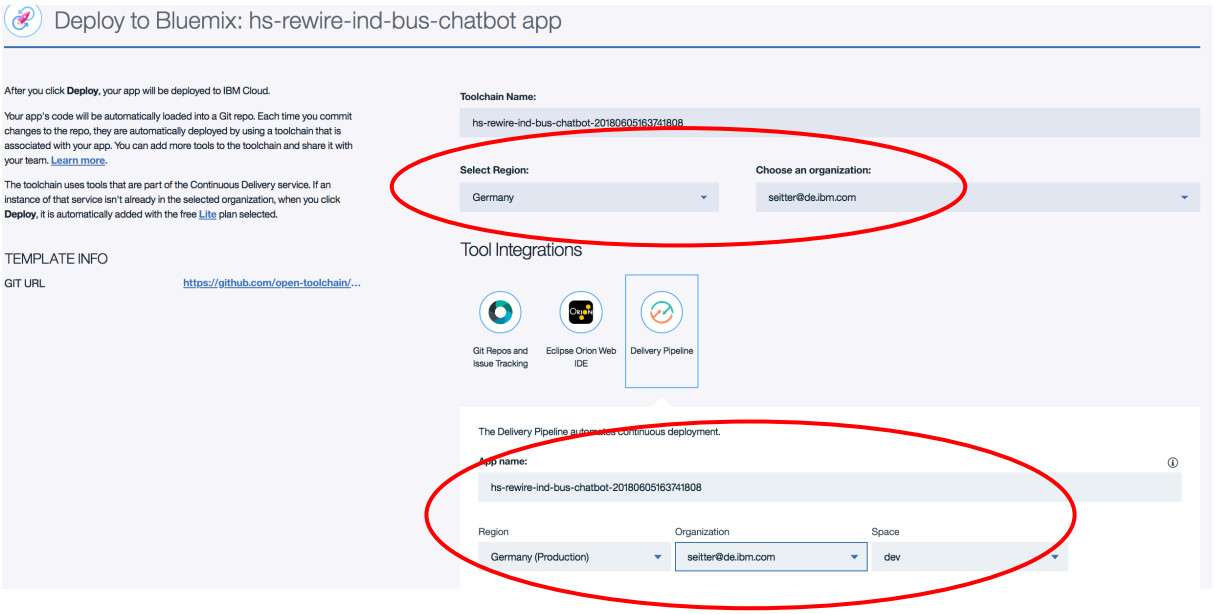


This will generate a DevOps toolchain in IBM Cloud and copy the source code to a IBM Cloud git repository

4.1.1 DevOps Toolchain

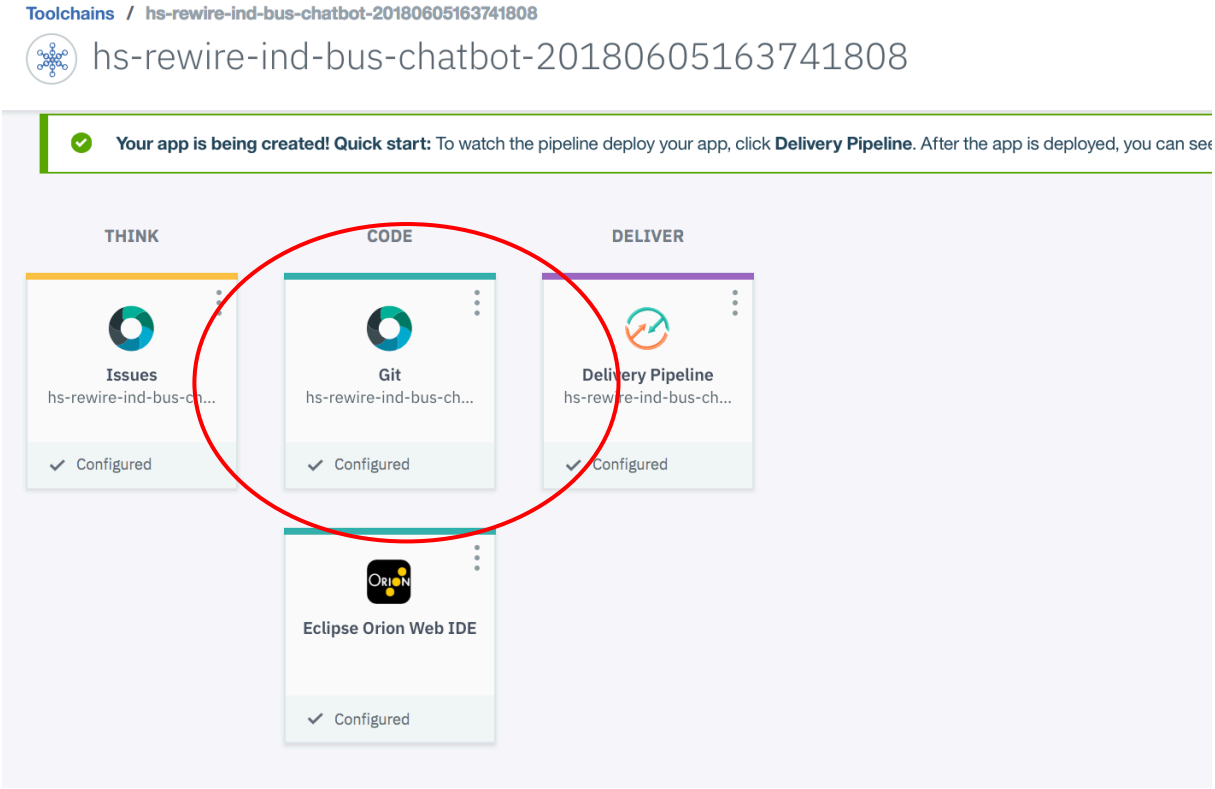
Check the setting so the Deployment screen

The IBM Cloud Region, organisation, ….



Then select deploy

4.1.2 Before you continue – Change App.js in the IBM Cloud Git

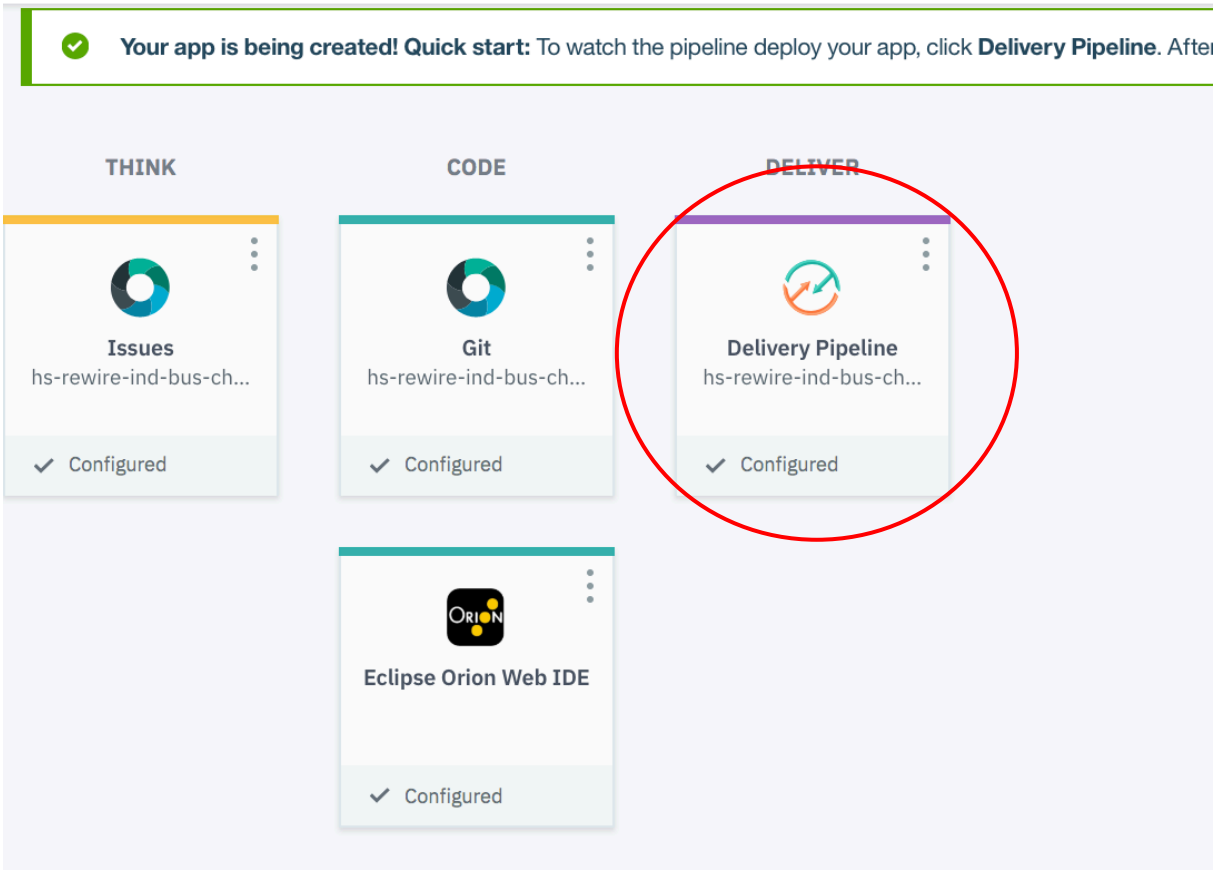


```
  }
  this.handleClick = this.handleClick.bind(this);
  this.handleSubmit = this.handleSubmit.bind(this);
}

callWatson(message) {
    /*** */
    //
    // here you have to specify the RESTAPI End point of your Function to invoke Watson Assistant
    //
    /*** */
  const watsonApiUrl = "put here your IBM Function sequence Endpoint address";

  const requestJson = JSON.stringify({
```

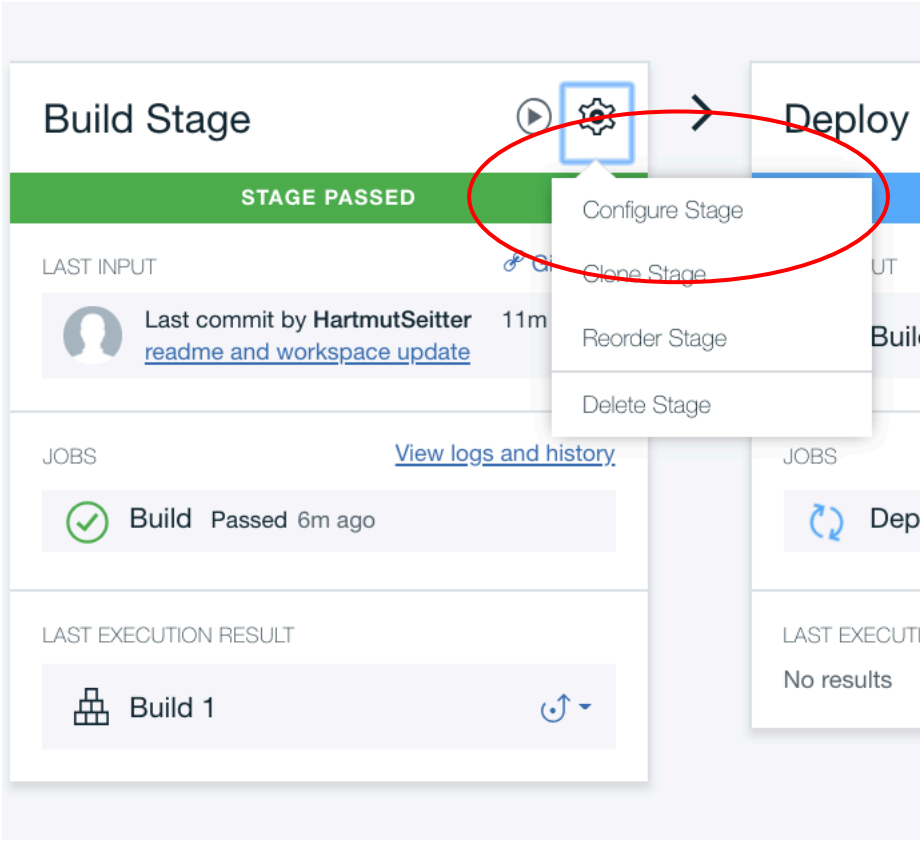
Here you have to copy the link from chapter 3.3.3 (the https address of the Function sequence – web enable)

Save and commit the App.js file to git.

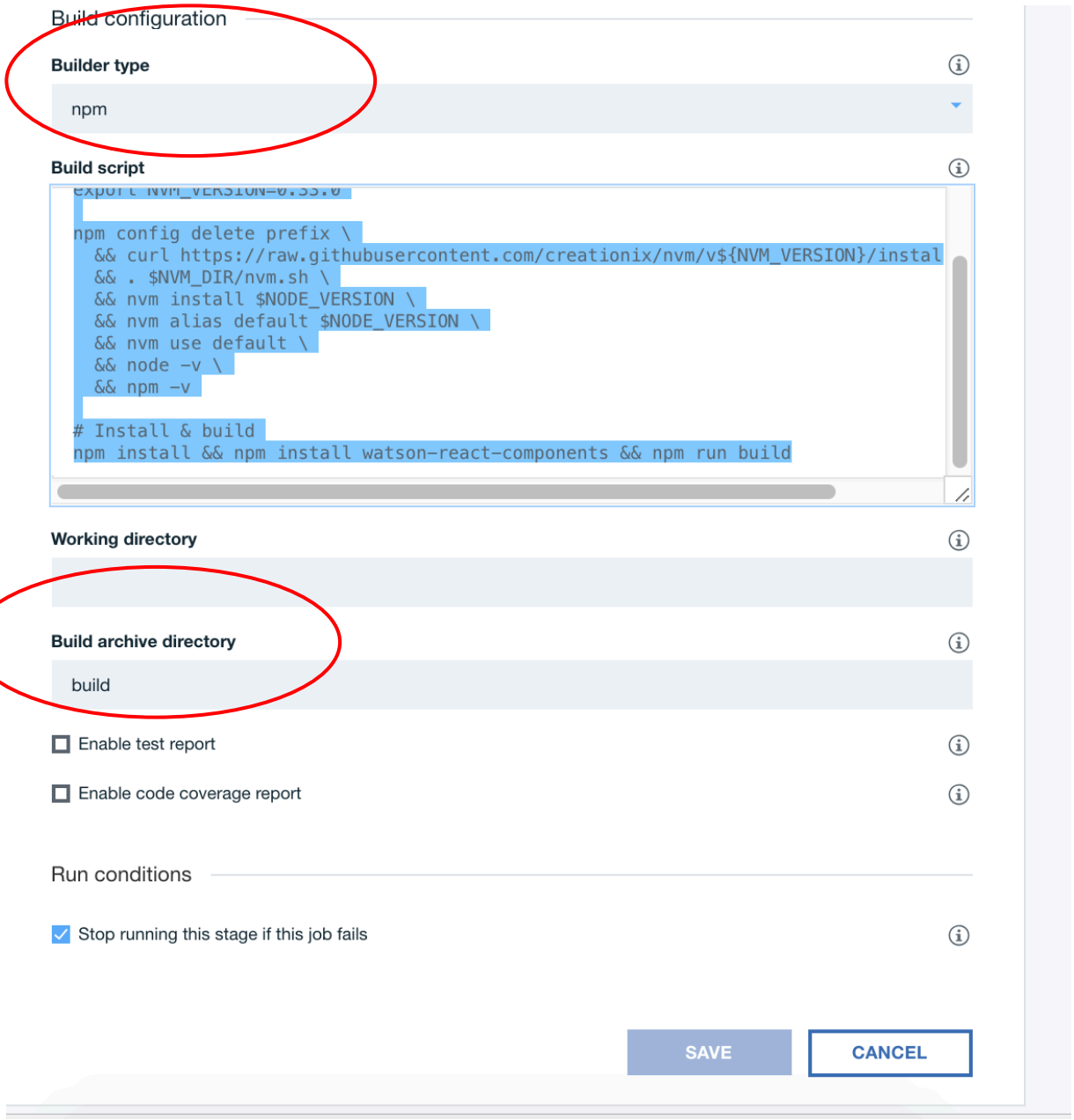
4.1.3 Specify the build and deployment script to the REACT Application

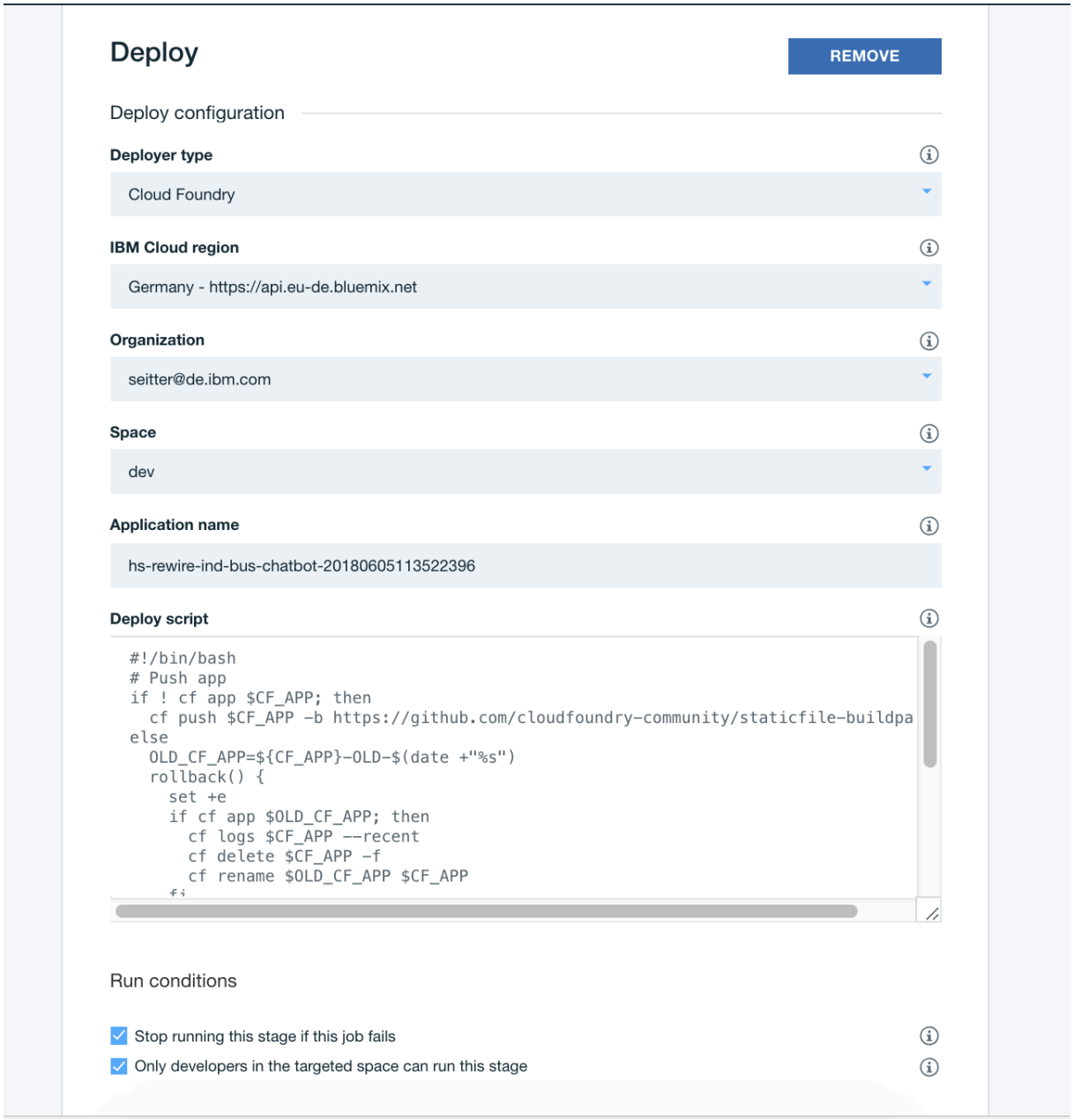


hs-rewire-ind-bus-chatbot-20180605163

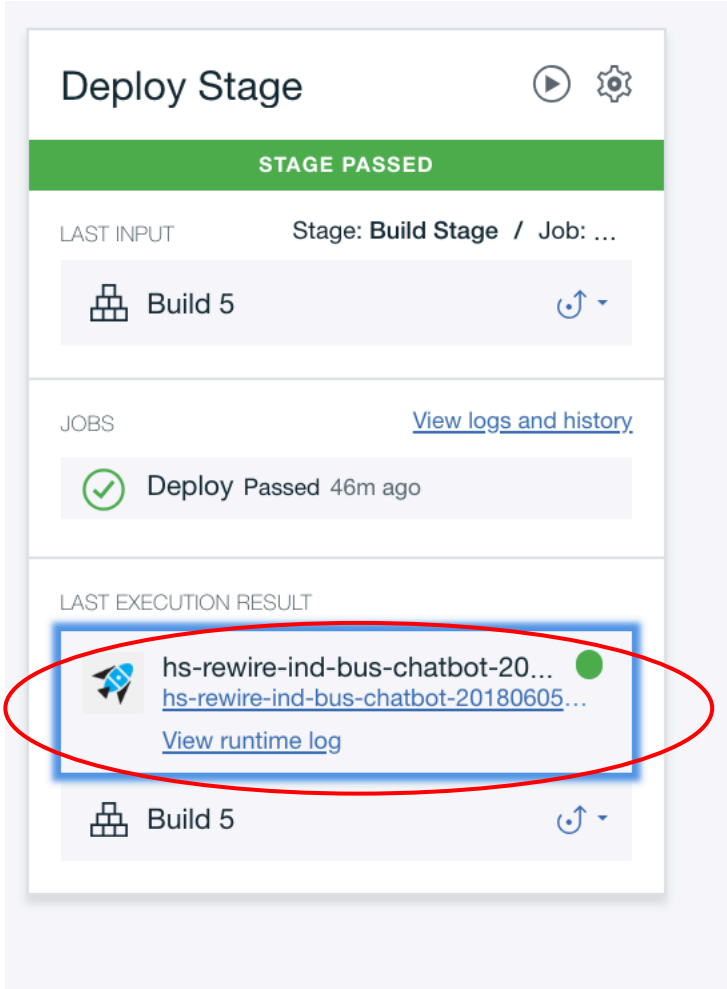


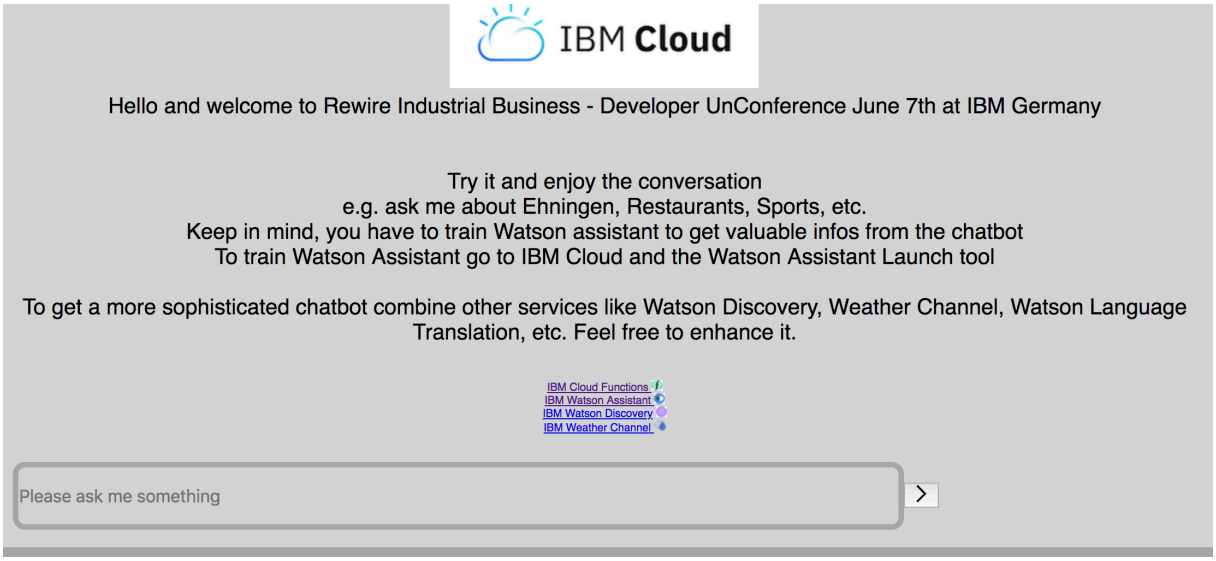
The build and deployment script you can also find on github


Build configuration
Builder type
npm
Build script
export NVM_VERSION=0.33.0
npm config delete prefix \
&& curl https://raw.githubusercontent.com/creationix/nvm/v${NVM_VERSION}/instal
&& . $NVM_DIR/nvm.sh \
&& nvm install $NODE_VERSION \
&& nvm alias default $NODE_VERSION \
&& nvm use default \
&& node -v \
&& npm -v
# Install & build
npm install && npm install watson-react-components && npm run build
Working directory
Build archive directory
build
Enable test report
Enable code coverage report
Run conditions
Stop running this stage if this job fails
SAVE
CANCEL



To build and deploy everything in IBM Cloud will take some minutes, if it completed successfully you should be ready and you can invoke that application


Deploy Stage
STAGE PASSED
LAST INPUT
Stage: Build Stage / Job: ...
Build 5
JOBS
View logs and history
Deploy Passed 46m ago
LAST EXECUTION RESULT
hs-rewire-ind-bus-chatbot-20...
hs-rewire-ind-bus-chatbot-20180605...
View runtime log
Build 5


IBM Cloud
Hello and welcome to Rewire Industrial Business - Developer UnConference June 7th at IBM Germany
Try it and enjoy the conversation
e.g. ask me about Ehningen, Restaurants, Sports, etc.
Keep in mind, you have to train Watson assistant to get valuable infos from the chatbot
To train Watson Assistant go to IBM Cloud and the Watson Assistant Launch tool
To get a more sophisticated chatbot combine other services like Watson Discovery, Weather Channel, Watson Language Translation, etc. Feel free to enhance it.
IBM Cloud Functions
IBM Watson Assistant
IBM Watson Discovery
IBM Weather Channel
Please ask me something

Appendix