

ChitChat Bot with Watson Assistant and IBM Functions

HandsOn Lab

By Capar Goldman

Address IBM-Allee 1, 71139 Ehningen

Telephone +

E-mail

and

By Hartmut Seitter

Address IBM-Allee 1, 71139 Ehningen

Telephone + 1707859522

E-mail Hartmut.Seitter@de.ibm.com

10 September 2018

Table of contents

1	Introduction	4
1.1	Lab Overview	4
1.2	The Architectural overview Diagram of the Lap Application	5
2	Let's start with the Watson Assistant Service.....	7
2.1	Watson Assistant Service Basics for this Session	7
2.2	The Watson Assistant Service for this Scenario	8
2.2.1	How to prepare your Watson Assistant Service.....	8
2.2.2	Define Intents and Utterances	10
2.2.3	Define Entities (optional).....	11
2.2.4	Define Dialogs	12
2.2.5	Anything Else.....	15
2.2.6	Jump to (optional).....	16
2.2.7	Test the conversation using IBM Watson Assistant.....	16
3	Now you need a Backend Application Functions	18
3.1	IBM Serverless Functions Basics for this Session	18
3.2	IBM Watson Assistant API Reference Basics for this Session	18
3.3	Build a Conversation Service for this Session	19
3.3.1	Collect the Parameters you need	19
3.3.2	Create an Action in IBM Function to invoke the Watson Assistant API	21
3.3.3	Create a Sequence to be prepared to invoke more than one Action in this scenario	24
4	Now you need a Frontend Application	27
4.1	Create from GitHub the Frontend Application.....	27
4.1.1	DevOps Toolchain.....	27
4.1.2	Before you continue – Change App.js in the IBM Cloud Git.....	28
4.1.3	Specify the build and deployment script to the REACT Application.....	29

Table of figures

Figure 1 Architecture Overview Diagram – All components of the Hands-On Lab	5
Figure 2 Create a Service Instance of Watson Assistant on IBM Cloud	7
Figure 3 Create a Service Instance of Watson Assistant on IBM Cloud	8
Figure 4 Import workspace on IBM Cloud for Watson Assistance	9
Figure 5 Define intent: First example.....	10
Figure 6 Define intent: Second example.....	11
Figure 7 Example entity	12
Figure 8 Define dialogs by using the If-Then query.....	12
Figure 9 Define dialog: First example.....	13
Figure 10 Define dialog: Second example.....	14
Figure 11 Add child nodes for a more complex conversation.....	14
Figure 12 How to use JSON editor for a response.....	15
Figure 13 How to implement a Jump to function.....	16
Figure 14 Use the Try it our function to test your conversation service	17
Figure 15 IBM Cloud Functions to save costs, scale, integrate	18
Figure 16 Watson Assistant API Reference Sample - Message.....	19
Figure 17 Watson Assistant Service credentials	20
Figure 18 Watson Assistant Workstation ID-1	20
Figure 19 Watson Assistant Workstation ID-2	21
Figure 20 IBM Functions – create it.....	21
Figure 21 IBM Functions – create Action.....	22
Figure 22 IBM Functions – create Action-2.....	22
Figure 23 IBM Functions – Action – define the parameter needed to interact with Watson Service ..	23
Figure 24 Coding	23
Figure 25 Change action input	24
Figure 26 IBM Functions – create a sequence	25
Figure 27 IBM Functions – Enable Web Action of the sequence.....	26
Figure 28 Deploy REACT App to IBM Cloud	27
Figure 29 Devops Toolchain	28
Figure 30 Devops Toolchain Git repository.....	28
Figure 31 REACT Application – Apps.js WatsonApiUrl.....	29
Figure 32 IBM Devops Toolchain Delivery Pipeline	29
Figure 33 IBM Devops Toolchain Build Stage	30
Figure 34 IBM Devops Toolchain Build parameter.....	30
Figure 35 IBM Devops Toolchain Deploy stage.....	31
Figure 36 IBM Devops Toolchain Deploy script	32
Figure 37 IBM Devops Toolchain Launch Application.....	33
Figure 38 Final Application.....	33

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 lab 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

Architecture Overview Diagram all components of the Hands-On Lab

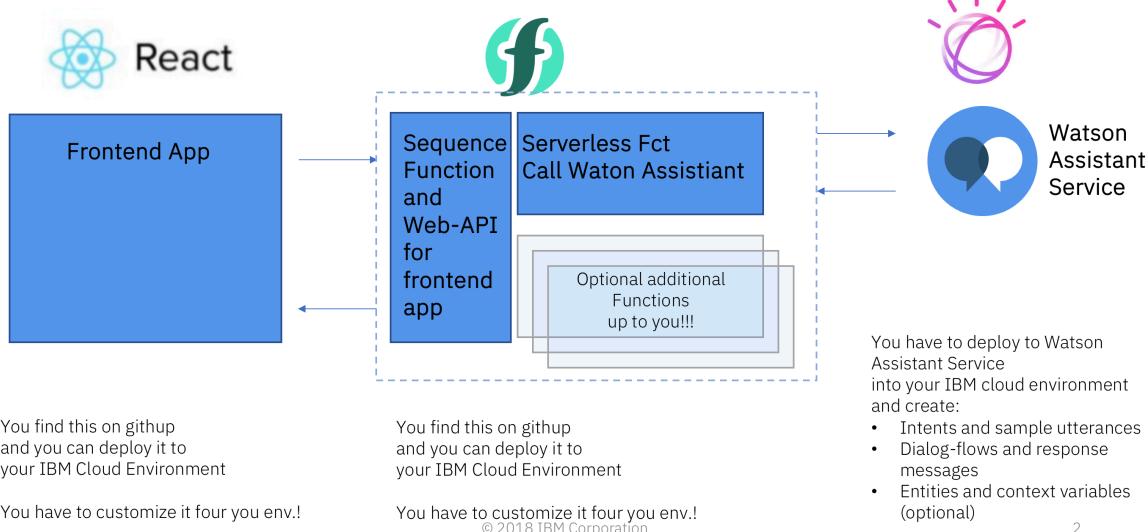


Figure 1 Architecture Overview Diagram – All components of the Hands-On Lab

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 2

- 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 3

- 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 4

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:

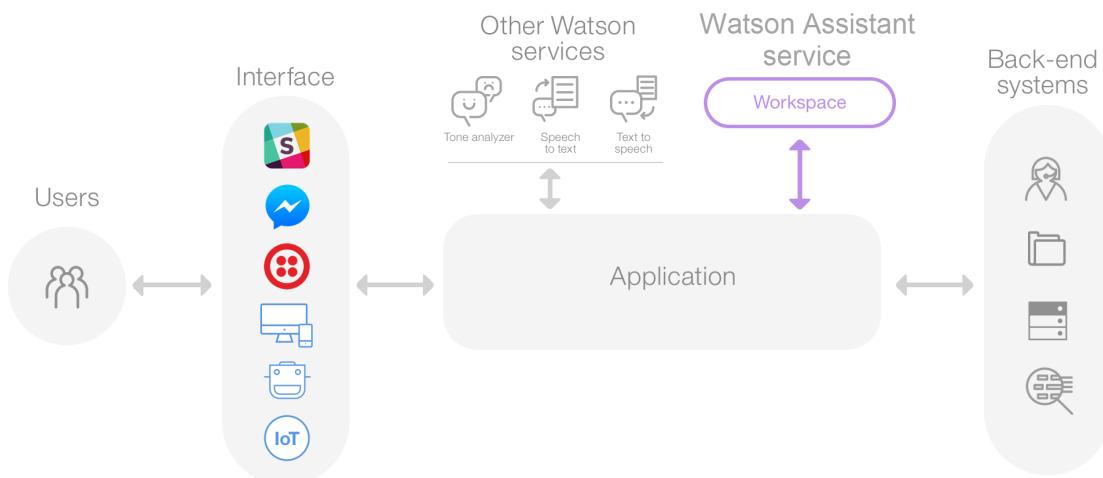
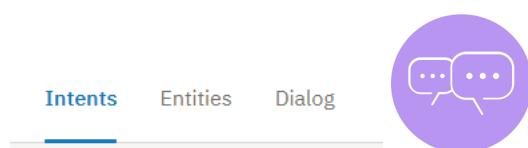


Figure 2 Create a Service Instance of Watson Assistant on IBM Cloud

<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. The dialog that you create is processed by the service from the first node in the tree to the last.

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.

Watson Assistant (formerly Conversation) is highlighted with a red circle. The service description says: "Build and deploy chatbots and virtual agents across a variety of channels..."

Service	Description	Links
Watson Assistant (formerly Conversation)	Build and deploy chatbots and virtual agents across a variety of channels...	Details Pricing Demo
Discovery	Uncover deep connections throughout your data by using advanced AI...	Details Pricing Demo
Visual Recognition	Understand the contents of images	Details Pricing Demo
Natural Language Understanding	Natural language processing for text. Understand keywords, concepts,...	Details Pricing Demo
Speech to Text	Easily convert audio and	Details Pricing Demo
Text to Speech	Convert written text into	Details Pricing Demo
Natural Language Classifier		Details Pricing Demo
Personality Insights	Infer personality	Details Pricing Demo

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>

IBM Watson Assistant

Home Workspaces

Workspaces

Create

Car Dashboard - Sample Cognitive Car Dashboard sample workspace which allows multi-turn conversations to perform tasks in the car. English (U.S.)	Create a new workspace Workspaces enable you to maintain separate intents, user examples, entities, and dialogs for each use or application. <i>You are using 0 of 5 available workspaces in this instance.</i>
--	---

Edit sample

Import a workspace

Select a JSON file then choose which elements from the workspace to import.

Choose a file

Import

Everything (Intents, Entities, and Dialog)
 Intents and Entities

Import

Figure 4 Import workspace on IBM Cloud for Watson Assistance

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 such as #greetings and #goodby. For this lab we ask you to prepare three examples:

First example: #Zurich_location

You might define an intent named: '**Zurich_location**' that answers questions about where Zürich 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 Zurich?**".

Figure 5 Define intent: First example

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_to_Zurich

The screenshot shows the IBM Watson Assistant interface. On the left is a sidebar with icons for Home, Create, Import, Export, and Help. The main area has a header with a back arrow and the text '#travel'. Below this, there are three sections: 'Intent name' containing '#travel', 'Description' containing 'how to get to Ehningen', and 'Add user examples' with a link to 'Add user examples to this intent'. A blue button labeled 'Add example' is visible. Below these sections is a list titled 'User examples (8)'. The examples listed are: 'how can I get to Ehningen', 'how do I get there?', 'how to find Ehningen', 'how to travel to Ehningen', 'show me possibilities to travel to Ehningen', 'transportation to Ehningen', and 'travel to Ehningen'. The first two examples are highlighted with a light gray background.

Figure 6 Define intent: Second example

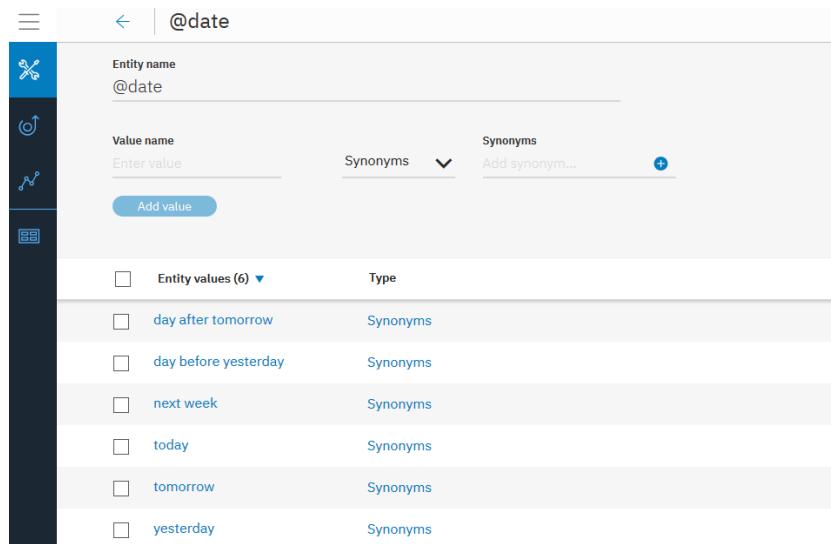
Third example: #yourownintent

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:

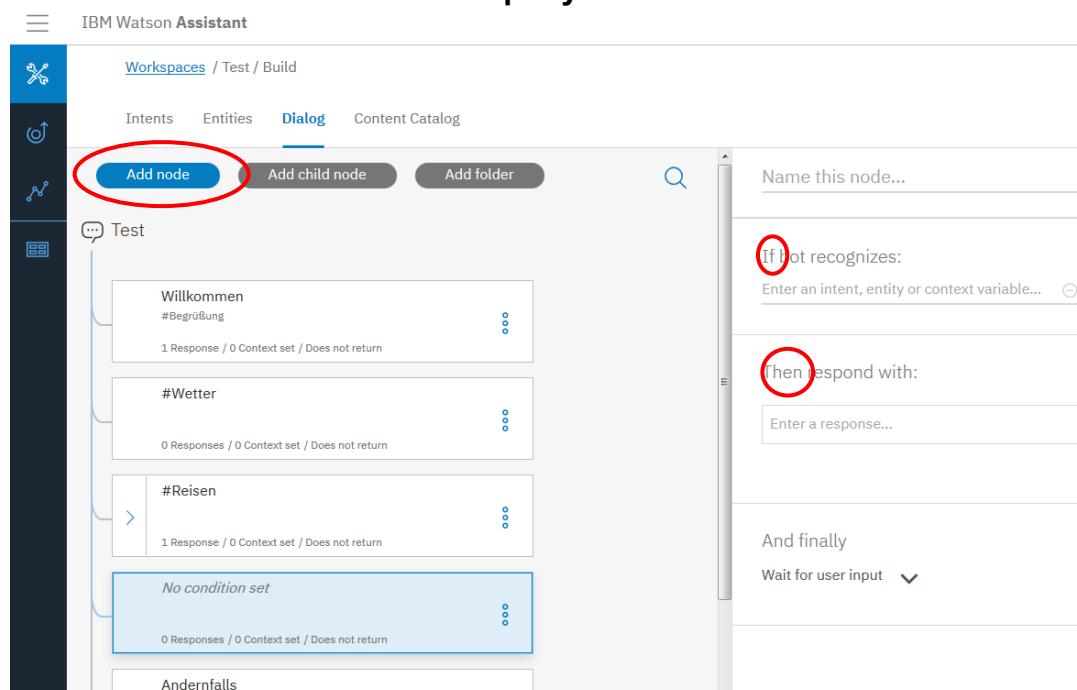
**Figure 7 Example entity**

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

2.2.4 Define Dialogs

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 and child nodes that handle the many possible permutations of a request.

Each node consists of an **If-Then query**.

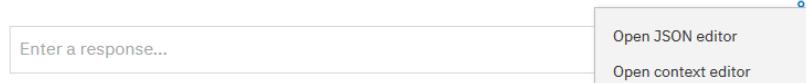
**Figure 8 Define dialogs by using the If-Then query**

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.

Then respond with:



Now let's get started by using our 3 examples from above:

#Zurich_location, #travel_to_Zurich, #yourownintent.

Use the prepared branch “intents” to implement your three example nodes and make sure you put them above the **anything_else** node (see page 15). The order of the branches is important because nodes are evaluated from first to last. **The first root node whose condition matches the input is used. Any nodes which would also match the trigger but come later in the tree are not triggered.**

First example: #Zurich_location

Add a new node named #Zurich_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.

Watson Assistant

[Workspaces](#) / Rewire Industrial Business / Build

Intents Entities Dialog Content Catalog

#Ehningen_location
#Ehningen_location
1 Response / 0 Context set

#confusion
#confusion
1 Response / 0 Context set

#highlight
#highlight
1 Response / 0 Context set

> #mayor
#mayor
1 Response / 0 Context set / Return allowed

> #sports
#sports

#Ehningen_location

If bot recognizes:
#Ehningen_location

Then respond with:

1. Ehningen is a town in the district of Böblingen in Baden-Württemberg in Germany.
2. Ehningen is close to Stuttgart, the capital of Baden-Württemberg in Germany.
3. Ehningen is in the southwest of Germany.

Add a variation to this response

Variations are sequential. Set to random

Figure 9 Define dialog: First example

Second example: #travel_to_Zurich

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 also child nodes.

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

The screenshot shows the IBM Watson Assistant interface in 'Dialog' mode. On the left, there's a sidebar with icons for Workspaces, Intents, Entities, Dialog (which is selected), and Content Catalog. The main area displays a tree-like dialog structure under the '#travel' intent. The root node '#travel' has three child nodes: '#car', '#train', and '#airplane'. Each child node has its own response section. To the right of the dialog tree, there's a panel for the '#travel' intent. It shows the current response: 'If bot recognizes: #travel' and 'Then respond with: 1. Do you want to travel by car, train or airplane?'. A red oval highlights the first response option '1. Do you want to travel by car, train or airplane?'.

Figure 10 Define dialog: Second example

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

This screenshot shows the same interface as Figure 10, but with a focus on adding child nodes. In the top navigation bar, the 'Dialog' tab is selected. Below it, there are buttons for 'Add node', 'Add child node' (which is highlighted with a red oval), and 'Add folder'. The main area shows the same dialog structure as Figure 10. To the right, the response panel for the '#travel' intent is shown. It includes the 'If bot recognizes:' section with '#travel' and the 'Then respond with:' section containing the counter question '1. Do you want to travel by car, train or airplane?'. A red oval highlights the 'Add child node' button in the navigation bar.

Figure 11 Add child nodes for a more complex conversation

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.

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:

Then respond with:

```

1 {
2   "output": {
3     "text": {
4       "values": [
5         "Ehningen is located directly on the S-Bahn line S1 Herrenberg - Stuttgart -  

6         Kirchheim\nTo get to the timetable of Deutsche Bahn click here:  

7         http://www.fahrplanauskunft.de/"
8       ],
9       "selection_policy": "sequential"
10      },
11      "linktext": [
12        "train time schedule"
13      ],
14      "textlink": [
15        "http://www.fahrplanauskunft.de/"
16      ]
17    }
18  }
19 }
```

Figure 12 How to use JSON editor for a response

Third example: #yourownintent

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

2.2.5 Anything Else Node

The service travels down the tree and moves along the triggered node to check the user input against any child node conditions. It works its way through the dialog tree until it reaches the last node in the branch.

This last node can be called "anything_else"-node as it is set to give an answer to input that is not intercepted in the other nodes. You can use a general response sentence such as: "I am sorry but I don't understand your question".

Keep in mind that any nodes that come later in the tree are not triggered.

AnythingElse ChitChat

If bot recognizes:

anything_else - +

Then respond with:

1. sorry, nobody told me how to handle this question -

2.2.6 Jump to (optional)

For complex conversations, jumps between the various dialog strings are possible but must be explicitly stated.

You can use the “Jump to” functions to guide questions to a different node, for example to the mother node, as you can see in this example:

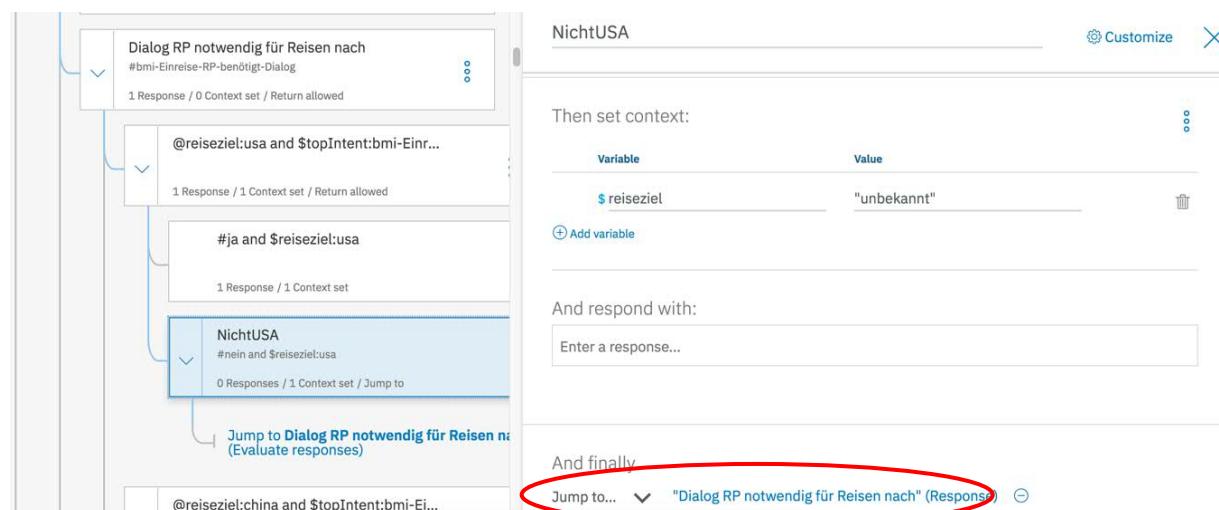


Figure 13 How to implement a Jump to function

2.2.7 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.

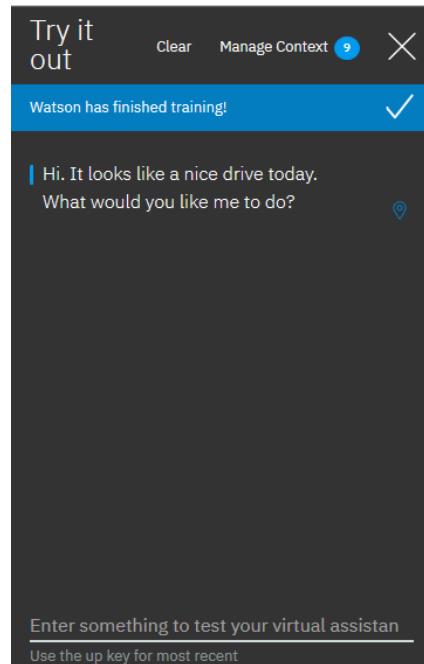


Figure 14 Use the Try it out function to test your conversation service

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.

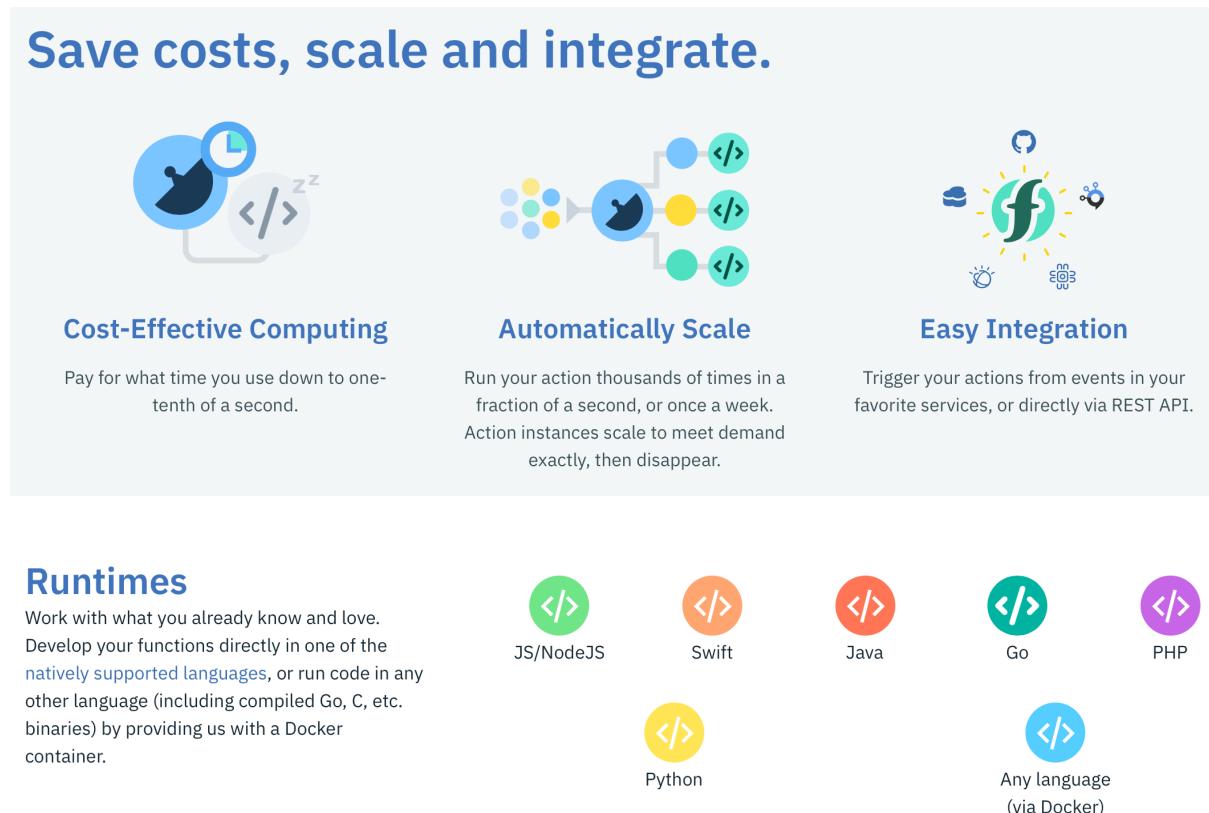


Figure 15 IBM Cloud Functions to save costs, scale, integrate

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 16 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 few 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:

The screenshot shows the Watson Assistant service dashboard. On the left, a sidebar lists 'Manage', 'Service credentials', 'Plan', and 'Connections'. The main area is titled 'Assistant : Watson Assistant - Rewire Industrial B...' and shows 'Location: US South', 'Org: seitter@de.ibm.com', and 'Space: dev'. Below this, a message says 'Get started with the service.' with links to 'Launch tool', 'Getting started tutorial', and 'API reference'. Under 'Credentials', a JSON configuration is displayed, with the URL, username, and password fields highlighted by a red oval.

```

    {
      "url": "https://gateway.watsonplatform.net/assistant/api",
      "username": "*****",
      "password": "*****"
    }
  
```

Figure 17 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'

The screenshot shows the Watson Assistant Workspaces page. At the top, there are 'Home' and 'Workspaces' tabs, with 'Workspaces' selected. Below this, a 'Create' button and an upload icon are visible. The main area displays two workspaces: 'Rewire Industrial Business' (selected) and 'Customer Service - Sample'. The 'Rewire Industrial Business' workspace has a 'No description added' note, language set to 'English (U.S.)', and a 'Last modified: 1 hour ago' timestamp. A context menu is open over this workspace, showing options: 'View details', 'Edit', 'Duplicate', 'Download as JSON', and 'Delete'. The 'Customer Service - Sample' workspace has a 'A virtual assistant for customer service sample' note, language set to 'English (U.S.)', and a blue 'Edit sample' button.

Figure 18 Watson Assistant Workstation ID-1

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

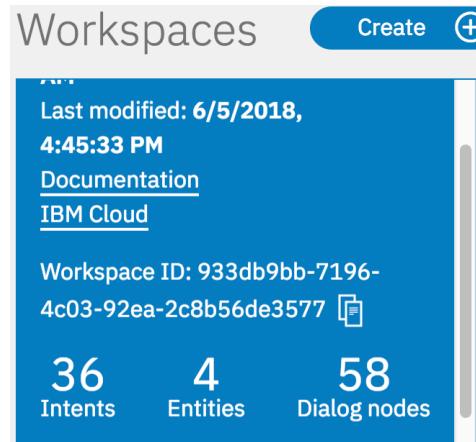


Figure 19 Watson Assistant Workstation ID-2

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

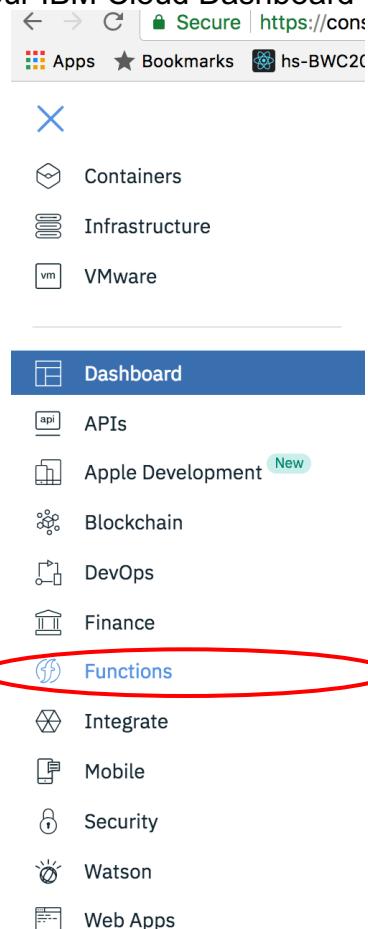


Figure 20 IBM Functions – create it

Step 2: Select Create Action

The screenshot shows the 'Create' page in the IBM Cloud Functions interface. On the left, there's a sidebar with links for Getting Started, Actions, Triggers, Monitor, Logs, and APIs. The main area has tabs for REGION (Germany), CLOUD FOUNDRY ORG (seitter@de.ibm.com), and CLOUD FOUNDRY SPACE (bcw2018). Below these tabs, there are four main sections: 'Quickstart Templates' (with a blue icon of a document with code snippets), 'Create Action' (with a green icon of code brackets), 'Create Trigger' (with a blue icon of a person icon), and 'Create Sequence' (with a green icon of two circular arrows).

Figure 21 IBM Functions – create Action

This screenshot shows the 'Create Action' configuration page. The sidebar on the left includes links for Getting Started, Actions, Triggers, Monitor, Logs, and APIs. The main form has fields for 'Action Name' (set to 'hs-watson-assistant'), 'Enclosing Package' (set to '(Default Package)'), 'Runtime' (set to 'Node.js 8'), and a note about Java or Docker support. There are also links for 'Learn more about Actions' and 'Learn more about Packages'.

Figure 22 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.

The screenshot shows the 'Parameters' section of the IBM Functions interface. It contains three entries:

Parameter Name	Parameter Value
username	your username
password	your password
workstation_id	your workstation_id

Figure 23 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:

```

1  /**
2   *
3   * Format and send request to Watson Conversation service
4   *
5   * @param {object} params - the parameters.
6   * @param {string} params.username - default parameter, must be set. The username for Conversation service.
7   * @param {string} params.password - default parameter, must be set. The password for Conversation service.
8   * @param {string} params.workspace_id - default parameter, must be set. The workspace_id for Conversation service.
9   * @param {string} params.input - input text to be sent to Conversation service.
10  * @param {string} params.context - context to be sent with input to Conversation service.
11  *
12  * @return {object} the JSON of Conversation's response.
13  *
14  */
15 const assert = require('assert');
16 const watson = require('watson-developer-cloud');
17
18 function main(params) {
19 return new Promise(function(resolve, reject){
20 assert(params, 'params cannot be null');
21 assert(params.username, 'params.username cannot be null');
22 assert(params.password, 'params.password cannot be null');
23 assert(params.workspace_id, 'params.workspace_id cannot be null');
24
25 assert(params.input, 'params.input cannot be null');
26 assert(params.context, 'params.context cannot be null');
27
28 console.log(workspace_id);
29 var conversation = watson.conversation({
30   username: params.username,
31   password: params.password,
32   version: 'v1',
33   version_date: '2017-05-26'
34 });
35
36 conversation.message({
37   workspace_id: workspace_id,
38   input: params.input,
39   context: params.context,
40 }, function(err, response) {
41   if (err) {
42     return reject(err);
43   }
44   console.log("reponse=",response);
45   return resolve(response);
46 });
47 });
48 }
49
50 module.exports.main = main;

```

Figure 24 Coding

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.?

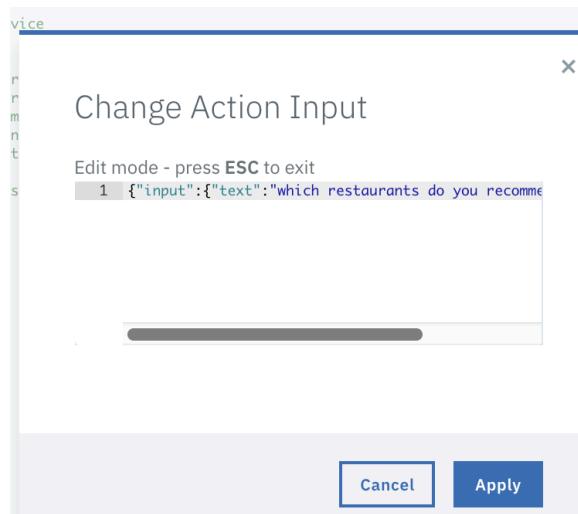


Figure 25 Change action input

Step 5: After saving your code, you can test it and verify the result you retrieve from 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 it 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'

REGION Germany CLOUD FOUNDRY ORG seitter@de.ibm.com CLOUD FOUNDRY SPACE bcw2018

Actions

Actions contain your function code and are invoked by events or REST API calls.

Search Actions Create

Default Package ?

NAME	RUNTIME	WEB ACTION	MEMORY	TIMEOUT
hs-watson-assistant	Node.js 8	Not Enabled	256 MB	60 s

↓ ↓ ↓

REGION Germany CLOUD FOUNDRY ORG seitter@de.ibm.com CLOUD FOUNDRY SPACE bcw2018

Create

Quickstart Templates Get started quickly using one of the Templates. A number of use cases are available, from a hello world action to invoking functions from Cloudant or Message Hub events.

Create Action Actions contain your function code and are invoked by events or REST API calls.

Create Sequence Sequences invoke Actions in a linear order, passing parameters from one to the next.

Create Trigger Triggers receive events from outside IBM Cloud Functions and invoke all connected Actions.

↓ ↓ ↓

REGION Germany CLOUD FOUNDRY ORG seitter@de.ibm.com CLOUD FOUNDRY SPACE bcw2018

Create Sequence

Sequences invoke Actions in a linear order, passing parameters from one to the next.

[Learn more about Sequences](#) [Learn more about Packages](#)

Sequence Name: hs-sequence

Enclosing Package: (Default Package) Create Package

Select Existing (1) Use Public

Select an Action... Default Package / hs-watson-assistant

Figure 26 IBM Functions – create a sequence

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 (you have to specify this address in the App.js file), or to use e.g. postman or curl command to interact with it.

REGION: Germany CLOUD FOUNDRY ORG: seitter@de.ibm.com CLOUD FOUNDRY SPACE: bcw2018

Actions

Actions contain your function code and are invoked by events or REST API calls.

Search Actions Create

Default Package

NAME	RUNTIME	WEB ACTION	MEMORY	TIMEOUT
hs-sequence	Sequence	Not Enabled	256 MB	60 s
hs-watson-assistant	Node.js 8	Not Enabled	256 MB	60 s

↓ ↓ ↓

hs-sequence
Region: Germany Org: seitter@de.ibm.com Space: bcw2018

Web Action Reset Save

Enable as Web Action Allow your Cloud Functions actions to handle HTTP events. Learn more about [Web Actions](#).

Raw HTTP handling When enabled your Action receives requests in plain text instead of a JSON body

HTTP METHOD	AUTH	URL
ANY	Public	https://openwhisk.eu-de.bluemix.net/api/v1/web/seitter%40de.ibm.com_bcw2018/default/hs-sequence.json

REST API

HTTP METHOD	AUTH	URL
POST	API-KEY	https://openwhisk.eu-de.bluemix.net/api/v1/namespaces/seitter%40de.ibm.com_bcw2018/actions/hs-sequence

CURL

```
curl -u API-KEY -X POST https://openwhisk.eu-de.bluemix.net/api/v1/namespaces/seitter%40de.ibm.com_bcw2018/actions/hs-sequence?blocking=true
```

Figure 27 IBM Functions – Enable Web Action of the sequence

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>

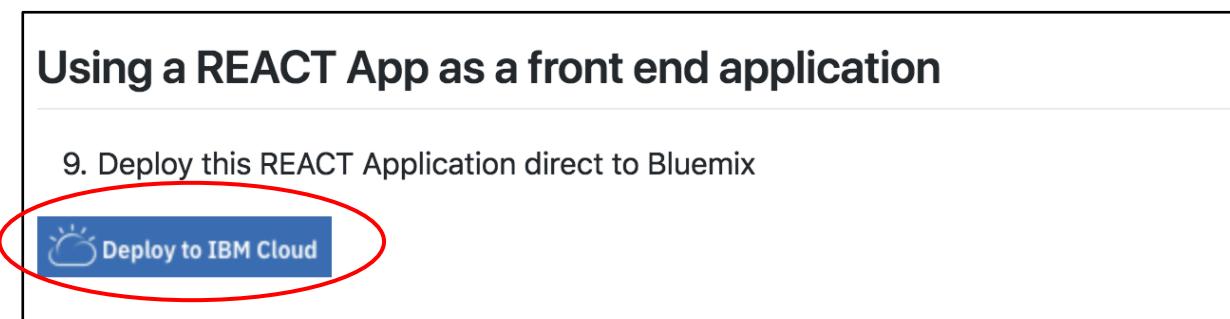
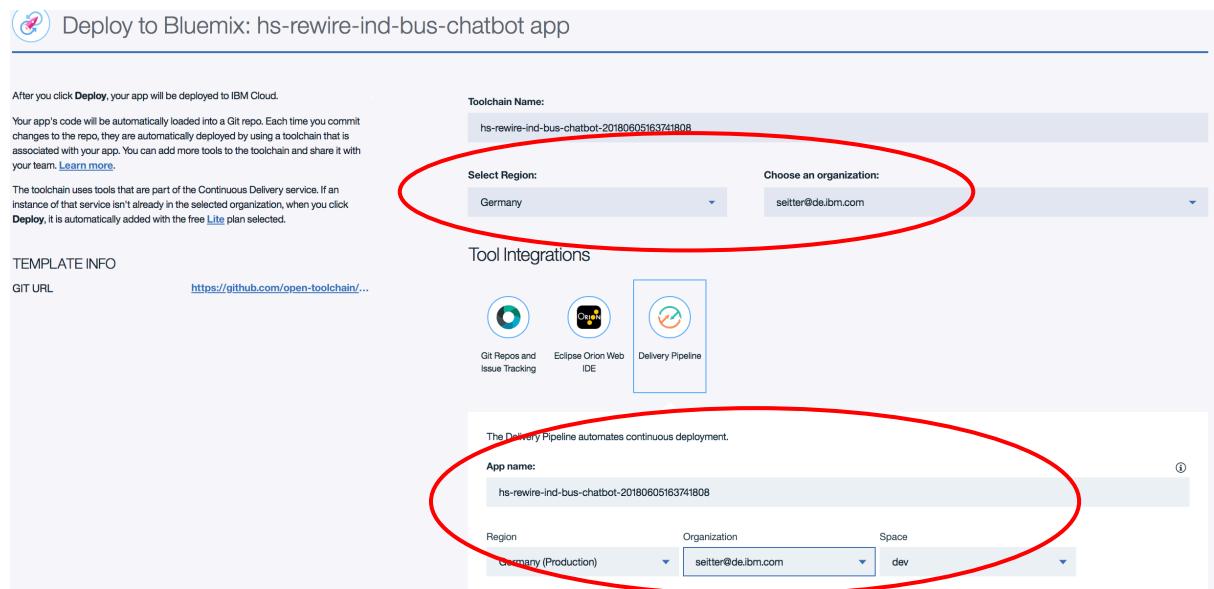


Figure 28 Deploy REACT App to IBM Cloud

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

4.1.1 DevOps Toolchain

Check the setting so the Deployment screen looks like this:

**Figure 29 Devops Toolchain**

Then select deploy.

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

Toolchains / hs-rewire-ind-bus-chatbot-20180605163741808

hs-rewire-ind-bus-chatbot-20180605163741808

THINK

CODE

Your app is being created! Quick start: To watch the pipeline deploy your app, click **Delivery Pipeline**. After the app is deployed, you can see the results in the **Delivery Pipeline** tab.

DELIVER

Git
hs-rewire-ind-bus-ch...
✓ Configured

Delivery Pipeline
hs-rewire-ind-bus-ch...
✓ Configured

Eclipse Orion Web IDE
✓ Configured

Figure 30 Devops Toolchain Git repository

```

    }
    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({

```

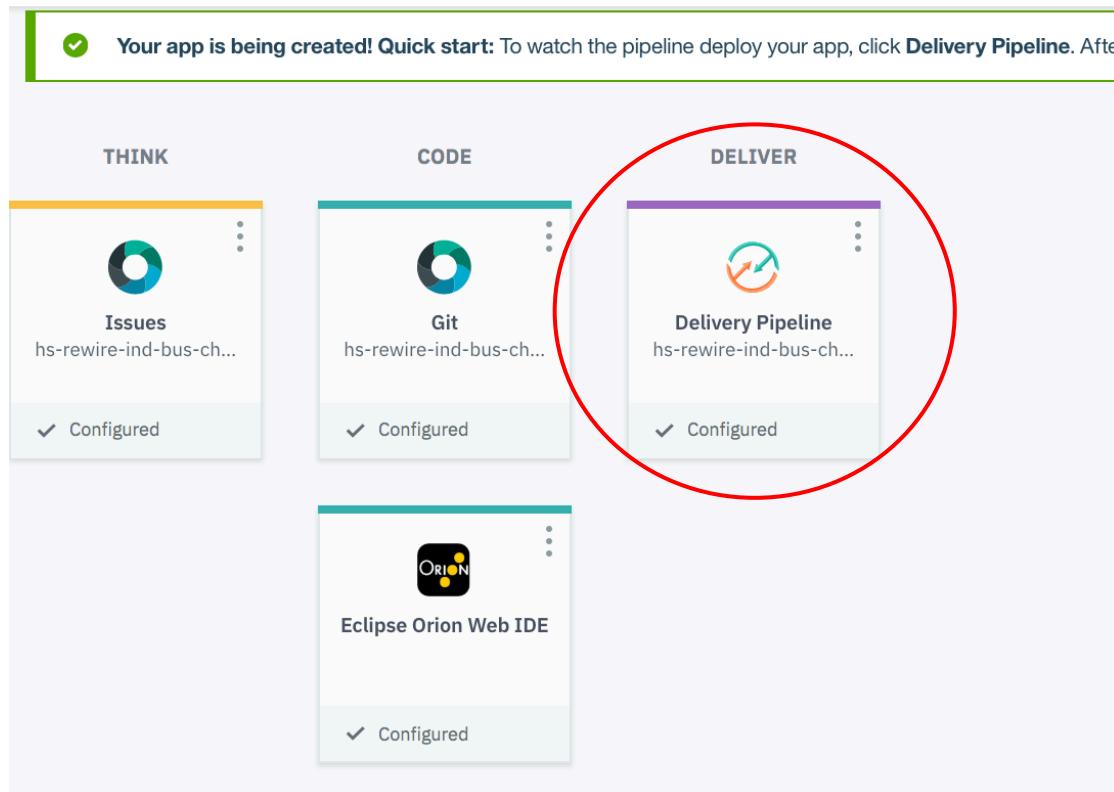
Figure 31 REACT Application – Apps.js WatsonApiUrl

Here you must 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

Step1: Select delivery pipeline:

**Figure 32 IBM Devops Toolchain Delivery Pipeline**

Step2: Select configure stage. You have to specify a Build script in the Build stage. You can find the Build script on GitHub: devops-toolchain-script/build-script.txt

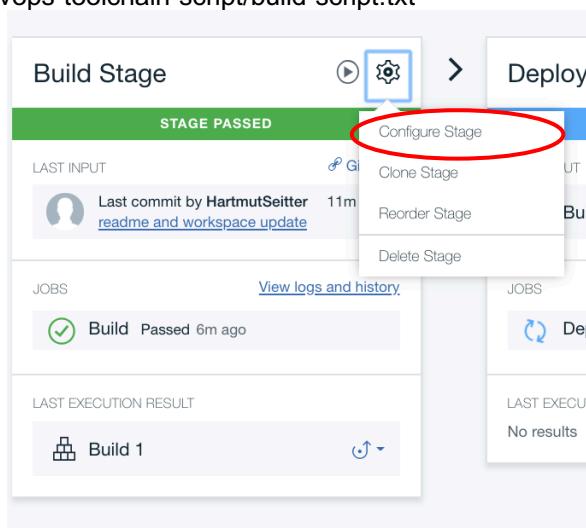


Figure 33 IBM Devops Toolchain Build Stage

Step 3: Select npm in Builder type, copy paste the Build script and specify in Build archive directory build.

Build configuration

Builder type: npm

Build script

```

#!/bin/bash -e
# Export NVM version
export NVM_VERSION=v0.35.3
# Remove existing nvm
rm -rf .nvm
# Download and install nvm
curl https://raw.githubusercontent.com/creationix/nvm/v${NVM_VERSION}/install.sh | bash
# Set default node version
nvm alias default ${NODE_VERSION}
# Use default node
nvm use default
# Install & build
npm install && npm install watson-react-components && npm run build

```

Working directory

Build archive directory: build

Run conditions

Stop running this stage if this job fails

SAVE **CANCEL**

Figure 34 IBM Devops Toolchain Build parameter

Click save.

Step4: Deploy stage. Click configure stage:

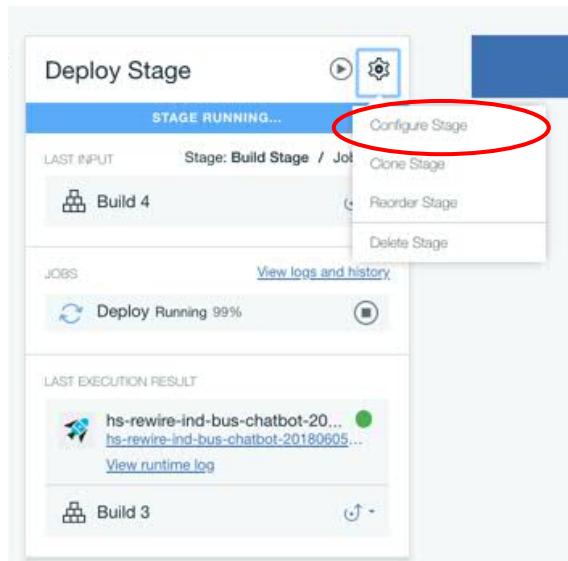


Figure 35 IBM Devops Toolchain Deploy stage

Step 5: Check parameters (Deploy type, Cloud region, Organization, Space...) if they correspond to your Cloud environment settings.

Step 6: Copy paste the Deployment script into the Deploy script box and save changes:

Deploy

REMOVE

Deploy configuration

Deployer type: Cloud Foundry

IBM Cloud region: Germany - https://api.eu-de.bluemix.net

Organization: seitter@de.ibm.com

Space: dev

Application name: hs-rewrite-ind-bus-chatbot-20180605113522396

Deploy script:

```
#!/bin/bash
# push app
if ! cf app $CF_APP; then
  cf push $CF_APP -b https://github.com/cloudfoundry-community/staticfile-buildpack
else
  OLD_CF_APP=${CF_APP}-OLD-$date +"%s"
  rollback() {
    set +
    if cf app $OLD_CF_APP; then
      cf logs $CF_APP --recent
      cf delete $CF_APP -f
      cf rename $OLD_CF_APP $CF_APP
    fi
  }
  rollback()
fi
```

Run conditions

Stop running this stage if this job fails

Only developers in the targeted space can run this stage



Figure 36 IBM Devops Toolchain Deploy script

Step 7: 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.
Open the application by clicking the following link:

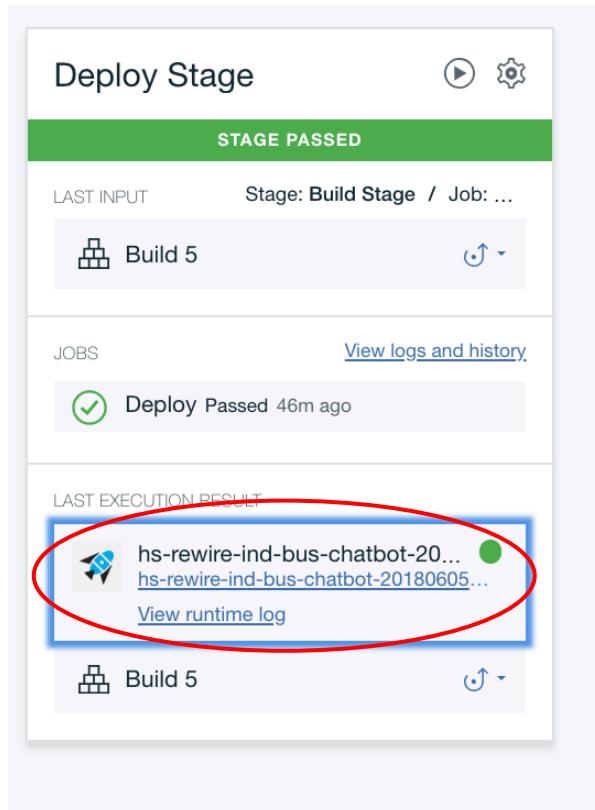


Figure 37 IBM Devops Toolchain Launch Application

It should look like this:

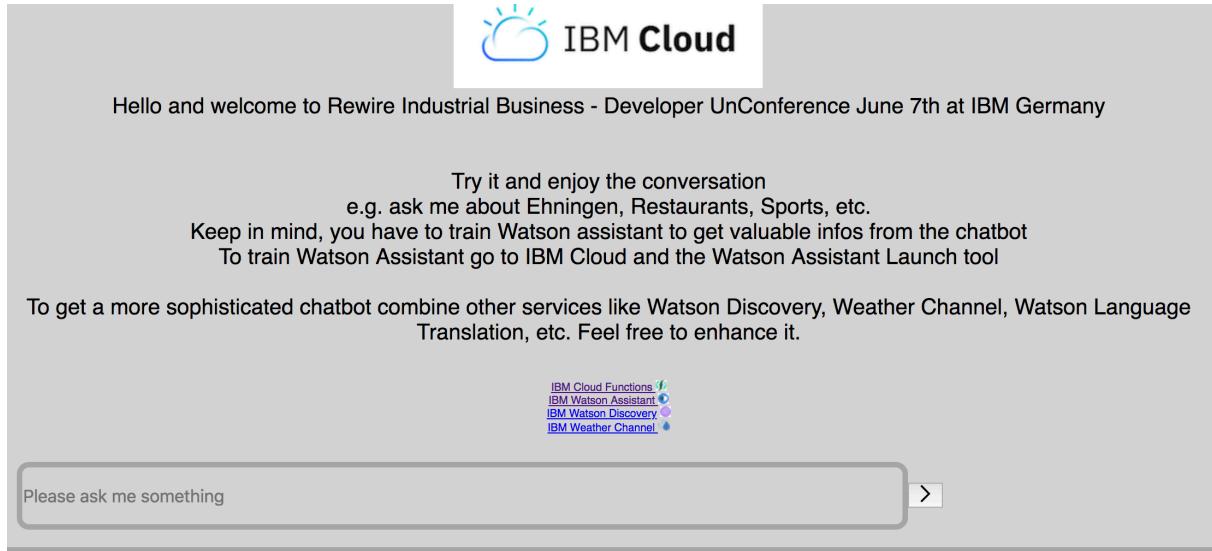


Figure 38 Final Application

Appendix