

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

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	Jumps (optional).....	15
2.2.6	Test the conversation using IBM Watson Assistant.....	16
3	Now you need a Backend Application Functions	17
3.1	IBM Serverless Functions Basics for this Session.....	17
3.2	IBM Watson Assistant API Reference Basics for this Session	17
3.3	Build a Conversation Service for this Session	18
3.3.1	Collect the Parameters you need	18
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	25
4	Now you need a Frontend Application	28
4.1	Create from GitHub the Frontend Application.....	28
4.1.1	DevOps Toolchain.....	28
4.1.2	Before you continue – Change App.js in the IBM Cloud Git.....	29
4.1.3	Specify the build and deployment script to the REACT Application	30

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 Watson Assistant API Reference Sample - Message.....	18
Figure 5 Watson assistant service credentials.....	19
Figure 6 Watson Assistant Workstation ID-1	20
Figure 7 Watson Assistant Workstation ID-2	21
Figure 8 IBM Functions – create it.....	22
Figure 9 IBM Functions – create Action.....	22
Figure 10 IBM Functions – create Action-2.....	23
Figure 11 IBM Functions – Action – define the parameter needed to interact with Watson Service ..	23
Figure 12 IBM Functions – create a sequence - 1	26
Figure 13 IBM Functions – create a sequence - 2	26
Figure 14 IBM Functions – create a sequence – 3.....	26
Figure 15 IBM Functions – Enable Web Action of the sequence- 1.....	27
Figure 16 IBM Functions – Enable Web Action of the sequence- 2.....	27

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

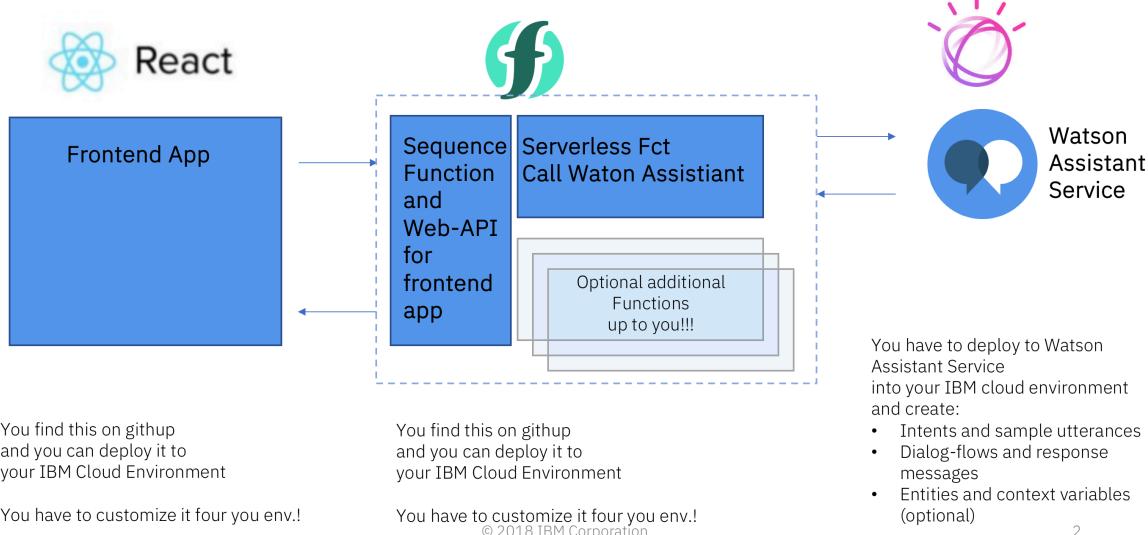


Figure 1 Architecture Overview Diagram – All components of the Hands-On Lab

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 [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:

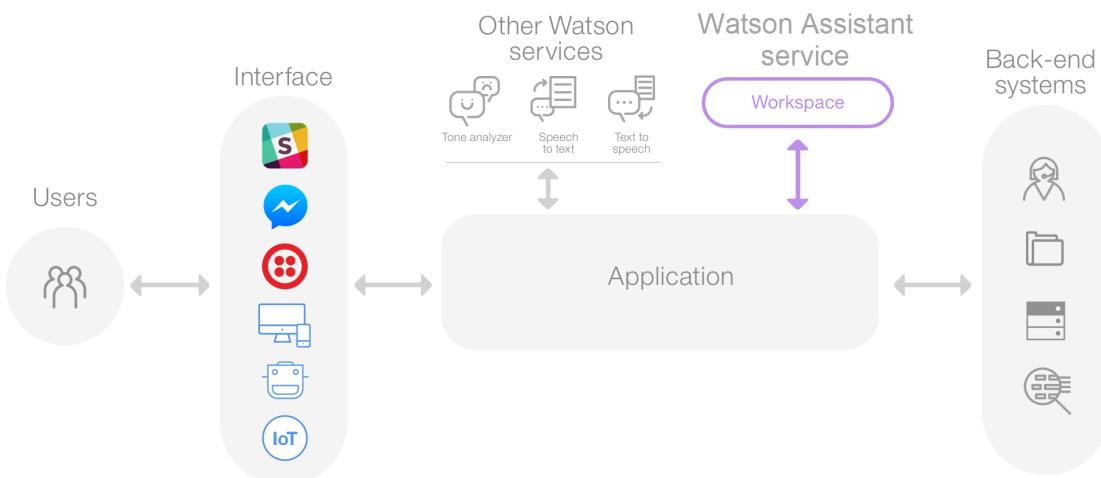
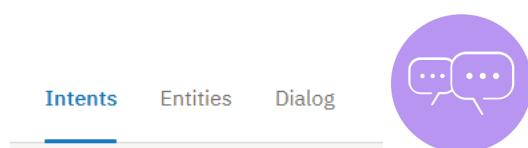


Figure 2 Create a Service Instance of Watson Assistant on IBM Cloud

<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.

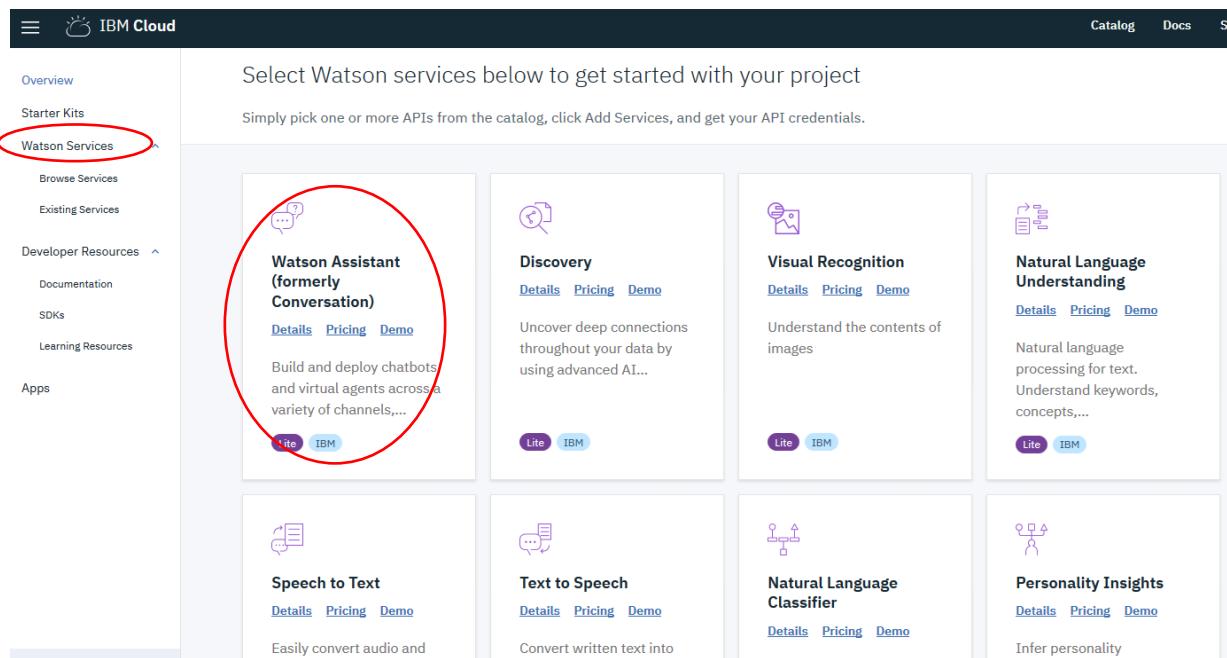


Figure 3 Create a Service Instance of Watson Assistant on IBM Cloud

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>

IBM Watson Assistant

The screenshot shows the 'Workspaces' section of the IBM Watson Assistant interface. At the top, there are 'Home' and 'Workspaces' navigation links. Below that, a 'Create' button and an 'Upload' icon (a blue circle with a white upward arrow) are visible. A red circle highlights the 'Upload' icon. On the left, a workspace named 'Car Dashboard - Sample' is listed, featuring a sample description and an 'Edit sample' button. On the right, a dashed box contains information about workspaces and a 'Create' button.

Import a workspace

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

The screenshot shows a file import dialog. It includes a 'Choose a file' input field containing the file name 'watson-assistant-workspace-starter.json'. Below it is an 'Import' section with two radio button options: 'Everything (Intents, Entities, and Dialog)' (which is selected) and 'Intents and Entities'. At the bottom is a large 'Import' button.

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

The screenshot shows the IBM Watson Assistant interface. On the left is a sidebar with icons for navigation, settings, and help. The main area shows an intent named '#Ehningen_location'. The intent has the following details:

- Intent name:** #Ehningen_location
- Description:** where on the map is Ehningen?
- Add user examples:** Add user examples to this intent
- Add example:** A button to add a new example.

Below the intent details, there is a section titled "User examples (9) ▾" which lists the following examples:

- is Ehningen close to Stuttgart?
- is ehningen in germany?
- is Ehningen in the south of Germany?
- tell me the location of Ehningen
- what is the location of Ehningen?
- where in germany lies ehningen?
- where is Ehningen?

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

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 back arrow and the text '#travel'. Below this, under 'Intent name', it says '#travel'. Under 'Description', it says 'how to get to Ehningen'. There is a button 'Add user examples' and a link 'Add user examples to this intent'. A blue button at the bottom says 'Add example'. Below these, there is a section titled 'User examples (8) ▾' with the following items:

- 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
- travel to Ehningen

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 examples, 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:

The screenshot shows the 'Entities' section of the IBM Watson Assistant interface. A new entity named '@date' has been created. The 'Value name' field contains 'Enter value'. There are buttons for 'Add value' and 'Synonyms'. Below this, a list of entity values is shown with checkboxes and 'Type' columns:

	Type
<input type="checkbox"/> day after tomorrow	Synonyms
<input type="checkbox"/> day before yesterday	Synonyms
<input type="checkbox"/> next week	Synonyms
<input type="checkbox"/> today	Synonyms
<input type="checkbox"/> tomorrow	Synonyms
<input type="checkbox"/> yesterday	Synonyms

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**.

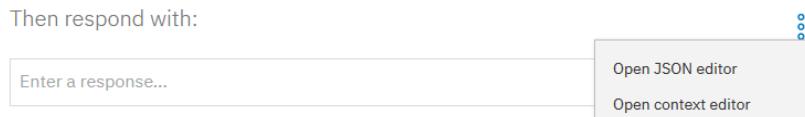
The screenshot shows the 'Dialog' tab of the IBM Watson Assistant interface. A new node is being added to a tree structure under the 'Test' workspace. The 'Add node' button is highlighted with a red circle. The configuration panel on the right shows fields for 'If bot recognizes:' and 'Then respond with:'. The 'If bot recognizes:' field has a placeholder 'Enter an intent, entity or context variable...'. The 'Then respond with:' field has a placeholder 'Enter a response...'. The tree structure on the left shows several nodes: 'Willkommen', '#Wetter', '#Reisen', 'No condition set', and 'Andernfalls'.

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 to continue: #Ehningen_location, #travel, #yourownintent. Click 'Add node' to create a dialog for each example.

First example: #Ehningen_location

For our first 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".

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, and Content Catalog. The main area shows a hierarchical tree of intents. The root intent is '#travel' (with a sub-node '#travel'). It has three child nodes: '#car', '#train', and '#airplane'. Each child node has a response defined as '#car', '#train', and '#airplane' respectively, both with '1 Response / 0 Context set'. To the right, the response for the root intent '#travel' is shown. It contains the sentence '#travel' followed by 'If bot recognizes: #travel'. Below that, under 'Then respond with:', is the counter question '1. Do you want to travel by car, train or airplane?'. This question is circled in red. There's also a link 'Add a variation to this response'.

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 the previous one, but with a focus on adding child nodes. The 'Add child node' button is highlighted with a large red circle. The rest of the interface is identical to the previous screenshot, showing the tree structure and the response definition for the '#car' node.

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:

Then respond with:

```

1 {
2   "output": {
3     "text": {
4       "values": [
5         "Ehningen is located directly on the S-Bahn line S1 Herrenberg - Stuttgart - Kirchheim\nTo get to the timetable of Deutsche Bahn click here:  

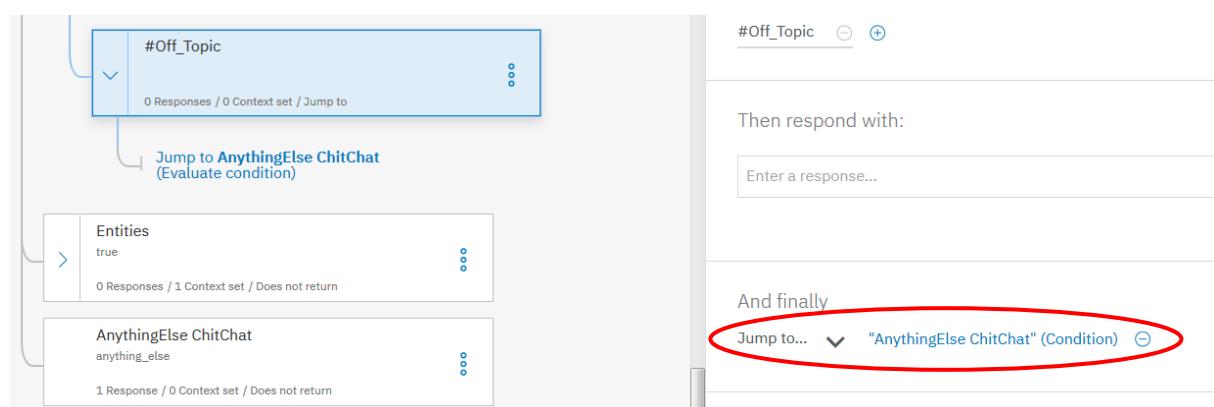
6           http://www.fahrplanauskunft.de/"
7         ],
8         "selection_policy": "sequential"
9       },
10      "linktext": [
11        "train time schedule"
12      ],
13      "textlink": [
14        "http://www.fahrplanauskunft.de/"
15      ]
16    }
17 }
```

Third example: #yourownintent

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

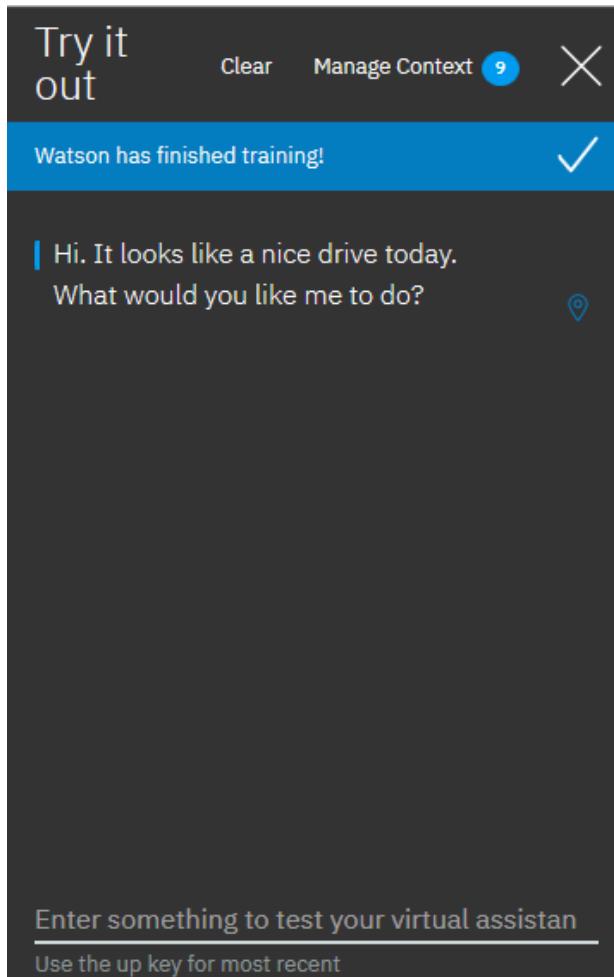
2.2.5 Jumps (optional)

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

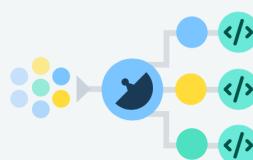
IBM Cloud Functions (based on Apache OpenWhisk) is a Function-as-a-Service (FaaS) platform which executes functions in response to incoming events and costs nothing when not in use.

Save costs, scale and integrate.



Cost-Effective Computing

Pay for what time you use down to one-tenth of a second.



Automatically Scale

Run your action thousands of times in a fraction of a second, or once a week. Action instances scale to meet demand exactly, then disappear.



Easy Integration

Trigger your actions from events in your favorite services, or directly via REST API.

Runtimes

Work with what you already know and love. Develop your functions directly in one of the **natively supported languages**, or run code in any other language (including compiled Go, C, etc. binaries) by providing us with a Docker container.



JS/NodeJS



Swift



Java



Go



PHP



Python



Any language
(via Docker)

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 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 credentials page. On the left, there's a sidebar with 'Manage' selected, followed by 'Service credentials', 'Plan', and 'Connections'. The main area displays the workspace 'hs-bcw2018-ebike' with details: Location: United Kingdom, Org: bcw2018, Space: dev. Below this, a message says 'Get started with the service.' and shows 'Plan: standard' with an 'Upgrade' link. A 'Launch tool' button is highlighted in blue, along with links for 'Getting started tutorial' and 'API reference'. Under the 'Credentials' section, there's a JSON snippet showing the configuration:

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

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’

IBM Watson Assistant

Home **Workspaces**

Workspaces

Create +

Rewire Industrial Business No description added English (U.S.) Last modified: 1 hour ago	Customer Service - Sample A virtual assistant for customer service sample English (U.S.) Edit sample
--	---

A screenshot of the IBM Watson Assistant Workspaces interface. The top navigation bar shows 'Home' and 'Workspaces'. Below the title 'Workspaces' is a 'Create' button with a plus sign and an upload icon. Two workspaces are listed: 'Rewire Industrial Business' and 'Customer Service - Sample'. The 'Rewire Industrial Business' workspace has a context menu open over its name, showing options: 'View details', 'Edit', 'Duplicate', 'Download as JSON', and 'Delete'. The 'Customer Service - Sample' workspace has a blue 'Edit sample' button.

Figure 6 Watson Assistant Workstation ID-1

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

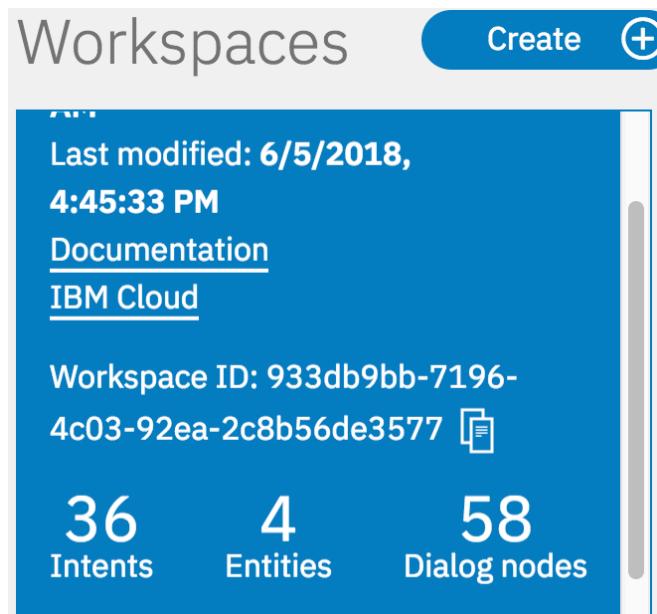
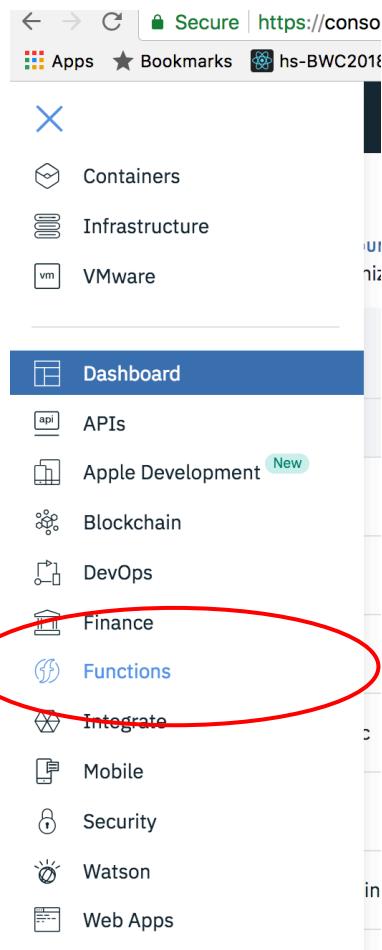


Figure 7 Watson Assistant Workstation ID-2

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

The screenshot shows the 'Create' page for IBM Cloud Functions. On the left, there is a navigation menu with options: Getting Started, Actions, Triggers, Monitor, Logs, and APIs. The main area displays four cards:

- 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 Trigger**: Triggers receive events from outside IBM Cloud Functions and invoke all connected Actions.
- Create Sequence**: Sequences invoke Actions in a linear order, passing parameters from one to the next.

Figure 9 IBM Functions – create Action

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

[Learn more about Actions](#)

[Learn more about Packages](#)

Action Name: hs-watson-assistant

Enclosing Package: (Default Package) [Create Package](#)

Runtime: Node.js 8

Looking for Java or Docker? [Java](#) and [Docker](#) Actions can be created with the [CLI](#)

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.

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

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>

← Actions

hs-watson-assistant

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

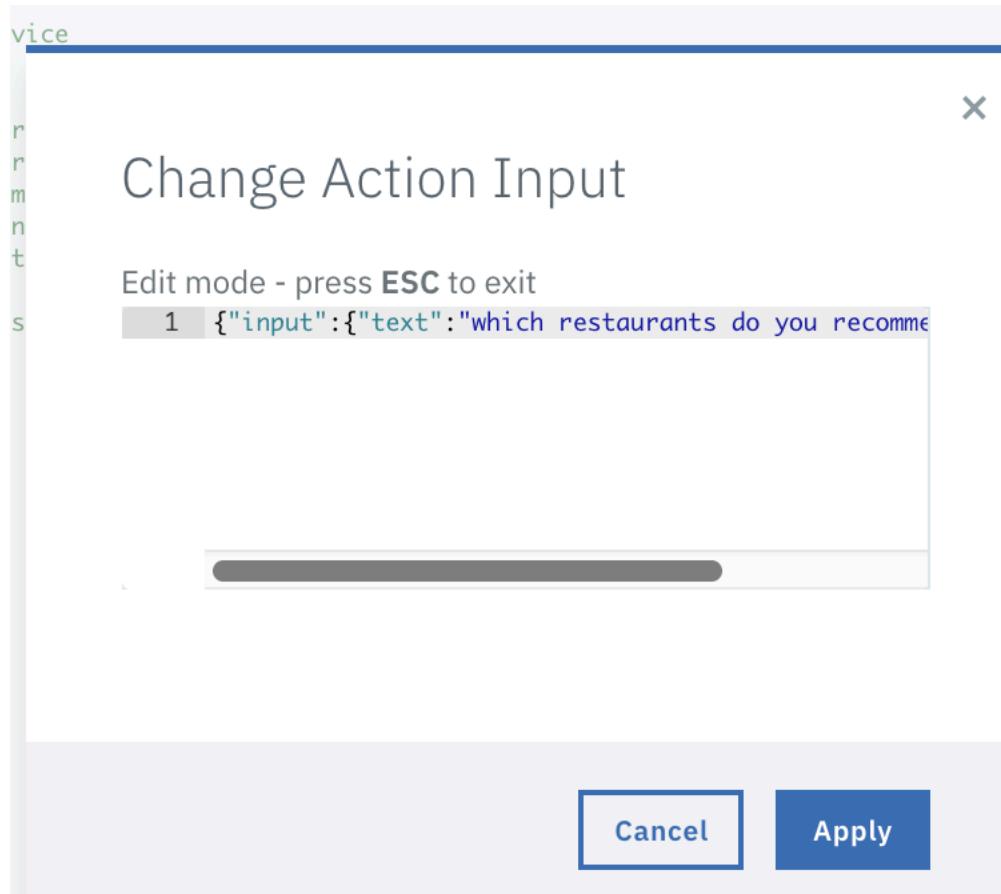
Code Node.js 8

```

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;
51
52

```

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 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.

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

Figure 12 IBM Functions – create a sequence - 1

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.

Figure 13 IBM Functions – create a sequence - 2

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.

Sequence Name
hs-sequence

Enclosing Package (Default Package)

Select Existing (1) Use Public
Select an Action...
Default Package / hs-watson-assistant

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.

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

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

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

Web Action

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

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 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  
```

A large red oval highlights the "Enable as Web Action" checkbox and its associated description.

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

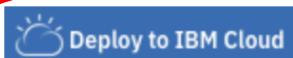
Go to github and on the README.md you find this blue button – Deploy to IBM Cloud

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

- a
 - b. Copy the javascript code from Github and create a new Function and specify the pwes-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

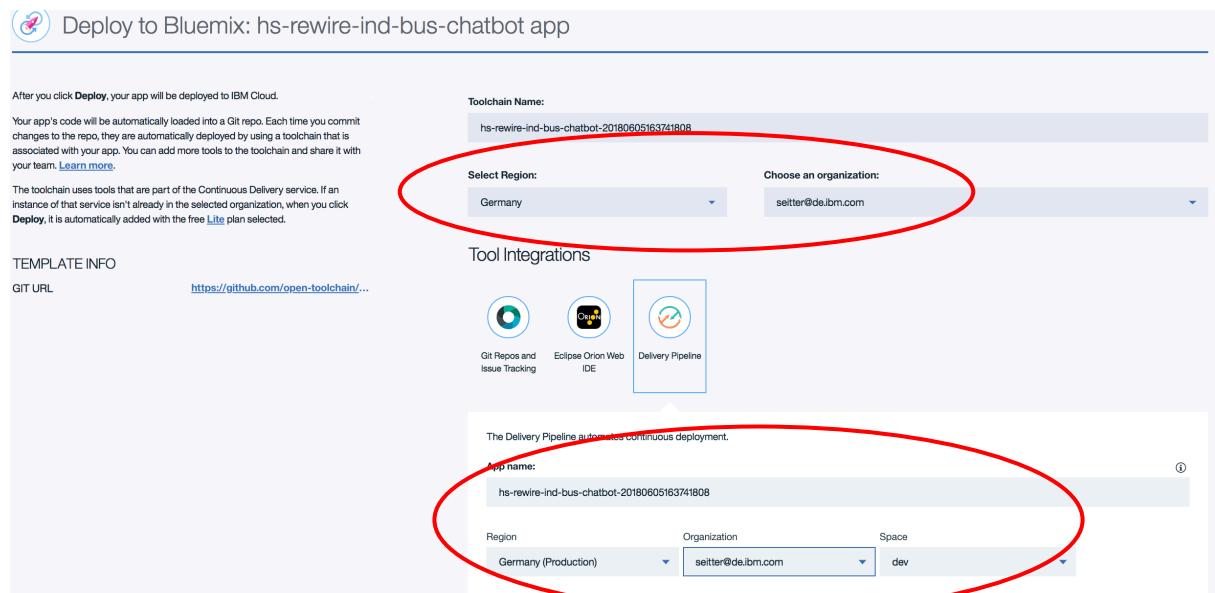


This will generate a DevOps toolchain in IBM Cloud and copy the source code to a IBM Cloud git repository

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

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

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 it here.

THINK

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

CODE

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

DELIVER

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

Eclipse Orion Web IDE ✓ Configured

A red oval highlights the 'Delivery Pipeline' card in the 'DELIVER' column.

```
}

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

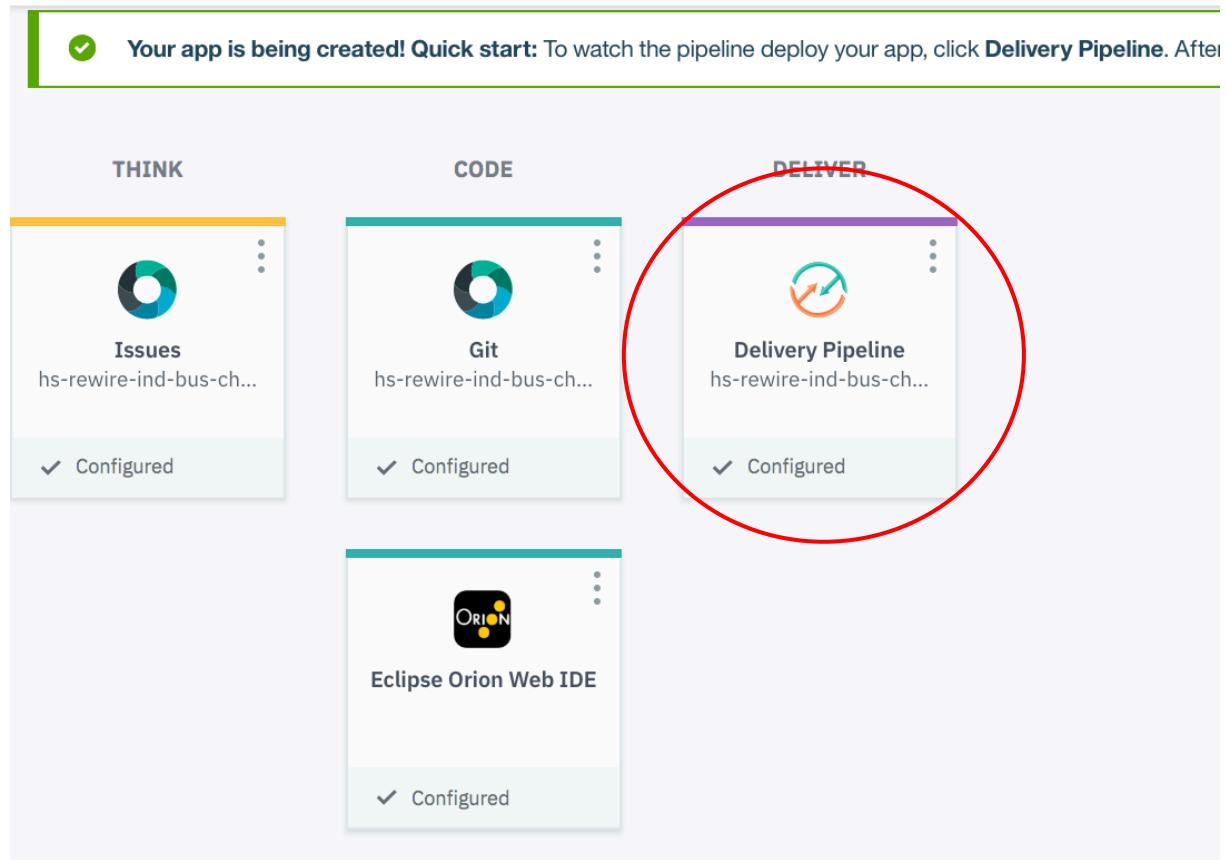
callWatson(message) {
  /**
   *
   * here you have to specify the RESTAPI End point of your Function to invoke Watson Assistant
   */
  /**
   */
  const watsonApiUrl = "put here your IBM Function sequence Endpoint address";

  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 screenshot shows a CI/CD pipeline interface. On the left, there's a 'Build Stage' card with a green 'STAGE PASSED' bar. It displays 'LAST INPUT' from 'HartmutSeitter' (commit message: 'readme and workspace update', 11m ago). Below it, under 'JOBS', is a 'Build' job that passed 6m ago. Under 'LAST EXECUTION RESULT', there's a 'Build 1' entry with a build icon and an up/down arrow. To the right, a 'Deploy' stage is shown with a 'UT' and 'Build' job. A context menu is open over the 'Build Stage' gear icon, listing 'Configure Stage', 'Clone Stage', 'Reorder Stage', and 'Delete Stage'. The 'Configure Stage' option is highlighted with a red circle.

The build and deployment script you can also find on github

Build configuration

Builder type

npm

Build script

```
export NVM_VERSION=0.35.0
npm config delete prefix \
&& curl https://raw.githubusercontent.com/creationix/nvm/v${NVM_VERSION}/install.sh | bash \
&& . $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**

Deploy

Deploy configuration

Deployer type (i)
Cloud Foundry

IBM Cloud region (i)
Germany - <https://api.eu-de.bluemix.net>

Organization (i)
seitter@de.ibm.com

Space (i)
dev

Application name (i)
hs-rewire-ind-bus-chatbot-20180605113522396

Deploy script (i)

```
#!/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
  }
  f;
```

Run conditions

Stop running this stage if this job fails (i)

Only developers in the targeted space can run this stage (i)

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: ...

JOBS [View logs and history](#)

Deploy Passed 46m ago

LAST EXECUTION RESULT

hs-rewire-ind-bus-chatbot-20... [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