InterConnect
2017

March 19–23
MGM Grand &
Mandalay Bay
Las Vegas, NV

IBM

Hands-on Lab
Session 2256
Session Build Your First Cognitive ChatBot Using OpenWhisk

Carlos Santana, IBM Cloud

Table of Contents

Preface

Serverless computing refers to a model where the existence of servers is entirely abstracted away. I.e. that even though servers still exist, developers are relieved from the need to care about their operation. They are relieved from the need to worry about low level infrastructural and operational details such as scalability, high-availability, infrastructure-security, and so forth. Hence, serverless computing is essentially about reducing maintenance efforts to allow developers to quickly focus on developing value-adding code.

Serverless computing simplifies developing cloud-native applications, especially microservice-oriented solutions that decompose complex applications into small and independent modules that can be easily exchanged.

Serverless computing does not refer to a specific technology; instead if refers to the concepts underlying the model described prior. Nevertheless, some promising solutions have recently emerged easing development approaches that follow the serverless model such as OpenWhisk.

OpenWhisk is a Function-as-a-Service (FaaS) platform that allows you to execute code in response to an event.

It provides you with the previously mentioned serverless deployment and operations model, with a granular pricing model at any scale that provides you with exactly the resources – not more not less – you need and only charges you for code really running. It offers a flexible programming model. incl. support for languages like JavaScript, Swift, Python, and Java and even for the execution of custom logic via Docker containers. This allows small agile teams to reuse existing skills and to develop in a fit-for-purpose fashion. It also provides you with tools to declaratively chain together the building blocks you have developed. It is open and can run anywhere to avoid and kind of vendor lock-in.

In summary, OpenWhisk provides...

- ... a rich set of building blocks that they can easily glue/stitch together
- ... the ability to focus more on value-add business logic and less on low-level infra-structural and operational details
- ... the ability to easily chain together microservices to form workflows via composition

In summary, our value proposition and what makes us different is:

- OpenWhisk hides infrastructural complexity allowing developers to focus on business logic
- OpenWhisk takes care of low-level details such as scaling, load balancing, logging, fault tolerance, and message queues
- OpenWhisk provides a rich ecosystem of building blocks from various domains (analytics, cognitive, data, IoT, etc.)
- OpenWhisk is open and designed to support an open community

- OpenWhisk supports an open ecosystem that allows sharing microservices via OpenWhisk packages
- OpenWhisk allows developers to compose solutions using modern abstractions and chaining
- OpenWhisk supports multiple runtimes including JavaScript, Swift, Python, Java, and arbitrary binary programs encapsulated in Docker containers
- OpenWhisk charges only for code that runs

The OpenWhisk model consists of three concepts:

- *trigger*, a class of events that can happen,
- *action*, an event handler -- some code that runs in response to an event, and
- *rule*, an association between a trigger and an action.

Services define the events they emit as triggers, and developers define the actions to handle the events.

The developer only needs to care about implementing the desired application logic - the system handles the rest.

During this lab you will learn about the basic concepts and how to build simple OpenWhisk solutions.

We wish you a lot of fun and success...

About these exercises

Requirements

This section describes the resources requirements for this lab. The two subsections are:

- Bluemix account
- OpenWhisk command-line interface (CLI)
- Watson Translator Service Instance
- Digital Copy of this document
- Source code for this lab

Bluemix account

Login or signup for a Bluemix account http://interconnectlabs.mybluemix.net/

OpenWhisk command-line interface (CLI)

Visit the following url to download the CLI https://console.ng.bluemix.net/openwhisk/learn/cli
Extract wsk binary to ~/Downloads/wsk by double clicking archive
Add ~/Downloads to environment variable PATH
echo 'PATH=$PATH:~/Downloads' >> ~/.bashrc
source ~/.bashrc
Configure wsk CLI with authentication key from the website, the command will be like
wsk property set --apihost openwhisk.ng.bluemix.net --auth xxxxx:yyyyy
Create environment variable for your NAMESPACE

```
echo 'NAMESPACE=`wsk namespace list | tail -n1`' >>~/.bashrc
echo APIHOST=`wsk property get --apihost | awk '{printf $4}'` >>~/.bashrc
source ~/.bashrc
```

Verify the CLI is configured correctly run
wsk list
If you get errors run
wsk property get

Watson Translator service instance

Login into Bluemix and create new Watson Language Translator service

Digital Copy of this document

http://ibm.biz/ic17-lab2256-doc

Source code for this lab

http://ibm.biz/ic17-lab2256-code

Terminal Setup

Open the "Terminal" application, will open in your home directory (~/).
Create a new directory "chatbot"

```
mkdir ~/chatbot
```

Change directory to working directory "chatbot"

```
cd ~/chatbot
```

Code Editor

Open the Atom editor from your working directory

```
atom ~/chatbot
```

Exercises Cognitive Chat Bot

Exercise 1 Web Action hello

Web actions are OpenWhisk actions to handle HTTP events, you can send HTTP requests and the actions will handle the request and control the response back to the HTTP client.

Create a new file hello.js with the following content

```
function main(input){
  let output = input.message || 'Hello'
  return {message:output};
}
```

Deploy your web action using the wsk CLI

```
wsk action update hello hello.js -a web-export true
```

Test your web action from terminal

```
curl
https://openwhisk.ng.bluemix.net/api/v1/web/$NAMESPACE/default/h
ello.json
```

Notice to replace **`$NAMESPACE`** in the URL path with your namespace for example "**user@example.com_dev**"

The output should be:
{
"message" : "hello"
}

Test your web action from the browser, this time passing a query parameter.

```
google-chrome
https://openwhisk.ng.bluemix.net/api/v1/web/$NAMESPACE/default/h
ello.json?message=hola
```

The output should be:
{
"message" : "hola"
}

Exercise 2 Web Action html

Web actions allows you to return the content of a web page.
This is done using the content extension ".html" on the invocation url.
This sets the content-type to "text/html" and the string return is the html field from the action result JSON object.

Create a new file index.js with the following content

```
function main() {
  let html = `
  <!DOCTYPE html><html><body>
  <h1>Hello html</h1>
  </body></html>
  `
  return {
    html: html
  };
}
```

Deploy your web action using the wsk CLI

```
wsk action update index index.js -a web-export true
```

Test your web action from terminal

```
curl
https://openwhisk.ng.bluemix.net/api/v1/web/${NAMESPACE}/default
/index.html
```

The output should be:

```
 <!DOCTYPE html>
<html>
<body>
  <h1>Hello html</h1>
</body>
</html>
```

Test your web action from the browse:

```
google-chrome
https://openwhisk.ng.bluemix.net/api/v1/web/$NAMESPACE/default/i
ndex.html
```

Exercise 3 Web Action form post

The same way you passed query parameters to the web action in exercise 1 with a http GET, you can also use the HTTP verb POST and include data in the body payload of the HTTP request.

Let's implement our Chat Bot message box, and post a message using the browser, the Web action will receive the http request and echo back a http response with the same message in a JSON payload.

On this exercise also shows how to create an action using a zip that includes multiple files.

Create the following files:

- echo.js
- chat.html
- chat.js
- form.js
- package.json

Create echo.js for a web action
This action will echo back your message, create a new file echo.js with the following content:

```
function main(args){
  return args;
}
```

Deploy your web action using the wsk CLI

```
wsk action update echo echo.js -a web-export true
```

Test your web action from terminal

```
curl
https://openwhisk.ng.bluemix.net/api/v1/web/${NAMESPACE}/default
/echo.json
```

Create the front-end web user interface (UI) files chat.html, chat.js, and form.js

Create a new file chat.html, this will be the UI template

```
<!DOCTYPE html>
<html>
<body>
  <h1>Hello form</h1>
  <form id="chat-form">
    <input type="text" id="message" placeholder="enter message
here">
    <button id="send">Send</button>
```

```
  </form>
  <div>Server response:</div>
  <div id="result"></div>
  <script type="text/javascript">
  ${script}
  </script>
</body>
</html>
```

Create a new file chat.js, this will be the UI controller code

```
var btn = document.querySelector('#send');
var message = document.querySelector('#message');
var result = document.getElementById('result');
btn.addEventListener('click', send, false);
function send(e) {
  var API_URL = "echo.json";
  var paramsString = `payload=${message.value}`
  var searchParams = new URLSearchParams(paramsString);
  e.preventDefault();
  fetch(API_URL, {
    method: "POST",
    body: searchParams
  }).then((res) => {
    return res.json()
  }).then((res) => {
    console.log(res);
    result.innerHTML= res.payload;
  }).catch((err) => {
    console.error(err);
  });
}
```

Create a new file form.js, this will be the action serving the UI

```
let fs = require('fs');
let script = fs.readFileSync(__dirname+'/chat.js','utf8');
let template = fs.readFileSync(__dirname+'/chat.html','utf8');
let html = template.replace('${script}',script);
function main(input){
  return {html: html}
}
exports.main = main;
```

Create a new file package.json, this defines the file name of the nodejs module, in our case form.js

```
{
  "name":"form-action",
  "main":"form.js"
}
```

Deploy

Deploy the form web action, by first creating a zip archive, and then using the wsk CLI

```
zip form.zip package.json form.js chat.js chat.html
wsk action update form form.zip -a web-export true --kind nodejs:6
```

Test

Open your browser to test the form

```
google-chrome
https://openwhisk.ng.bluemix.net/api/v1/web/$NAMESPACE/default/form.html
```

The browser will render an input text box and a send button.
Type in any text and then click the send button.
You will see that the echo action gets invoke and the UI displays the message that was sent.

Exercise 4 Translate Chat

IBM Bluemix offers multiple cognitive services to enable you to build cognitive apps that help enhance, scale, and accelerate human expertise. You can leverage this services from your OpenWhisk actions by using the REST API that these cognitive services offer.

Four our Chat we will use the Watson Translator service; The Watson Language Translator service provides domain-specific translation between languages. Our intention is to provide domain-relevant translations. Examples of where Watson Language Translator could be used include: An English-speaking help desk representative assists a Spanish-speaking customer through a chat session that is translated in real-time.

New Watson Service

Create a new **Watson service for Language Translator** under the same organization and space used to get the information to configured the OpenWhisk CLI.

Rename the auto generated name provided by the creation form, replace space with underscore “_”, remove any random characters, making a short name like **“Language_Translator”**

Verify that at least one Service Credential for the Language Translator shows up, a default one is always created and it’s named “Credentials 1”. Service credentials contain the username and password to access the service API.

Now create a package binding using the Language Translator service you created previously by running the following OpenWhisk CLI command

OpenWhisk Package Binding

Make sure that the Language Translator service gets created and shows up in the list of created or updated bindings by running the following command:

```
wsk package refresh

_ refreshed successfully
created bindings:
Bluemix_Language_Translator_Credentials-1
updated bindings:
deleted bindings:
```

Important!!!!

Please attention to the binding name: **Bluemix_Language_Translator_Credentials-1**

This name might be different value for your Bluemix account based on the name you choose during the creation of the service on the Bluemix dashboard.

Notice that the name if the name contains spaces, you will need to use quotes around the name when working with the terminal window

Now you have a translator action with the credentials already set, run the following command to list the actions under the new created package binding

```
wsk package get --summary "Bluemix_Language_Translator_Credentials-1"
```

The following output shows the translator action:

```
package /$NAMESPACE/Bluemix_Language_Translator_Credentials-1
 action /$NAMESPACE/Bluemix_Language_Translator-_Credentials-1/translator: Translate text
   (parameters: password, payload, translateFrom, translateTo, username)
 action /$NAMESPACE/Bluemix_Language_Translator_Credentials-1/languageId: Identify language
   (parameters: password, payload, username)
```

OpenWhisk Action Sequence chat-sequence

Create a sequence to be able to use the translator action from a web action, this command will create a sequence using echo created previously.

```
wsk action update chat-sequence
"Bluemix_Language_Translator_Credentials-1/translator","echo" --
sequence -a web-export true
```

Now update the previous example chat.js and instead of invoking the web action echo invoke chat-sequence.
Modify the file chat.js replacing echo.json with chat-sequence.json:

```
var API_URL = "chat-sequence.json";
```

Modify the file chat.html to it's clear to users what to enter

```
placeholder="enter English here, output is French">
```

Deploy

```
zip form.zip package.json form.js chat.js chat.html
wsk action update form form.zip -a web-export true --kind nodejs:6
```

Testing

Now open a new browser tab or refresh the browser using the URL from the previous exercise

```
google-chrome
https://openwhisk.ng.bluemix.net/api/v1/web/$NAMESPACE/default/form.html
```

Now enter text into the input box in **English**, then click the send button.
The response of the message will now be in **French**.
The translator supports other languages to translate, but the default parameters is to translate from English to French if not specified.

Exercise 5 Cognitive Chat Bot

In the previous example we implemented a Chat that translated only from English to French.
The Watson Translator service also provides an API for natural language identification giving back a confidence level on what language is for the message.

To verify that you have a package binding and the action languageId correctly configured with the service credentials read the package summary:

Important!!!!

Please attention to the binding name: **Bluemix_Language_Translator_Credentials-1**

This name might be different value for your Bluemix account based on the name you choose during the creation of the service on the Bluemix dashboard.

```
wsk package get --summary "Bluemix_Language_Translator_Credentials-1"
```

The following output shows the translator action:

```
package /$NAMESPACE/Bluemix_Language_Translator_Credentials-1
 action /$NAMESPACE/Bluemix_Language_Translator-_Credentials-1/translator: Translate text
    (parameters: password, payload, translateFrom, translateTo, username)
 action /$NAMESPACE/Bluemix_Language_Translator_Credentials-1/languageId: Identify language
    (parameters: password, payload, username)
```

New Code

Create a new file bot.js to handle the backend business logic for your bot application, this example is very simple for demonstration purposes, but take into account that you can have more logic and call external APIs and other OpenWhisk actions from within your business logic backend code.

```
//function gets call by languageId output, and sends the input to translator
function main({ payload: message, confidence, language }) {
  var output = {
    translateTo: 'en',
    translateFrom: language,
    payload: message
  };
  if (confidence < 0.7) {
    output.translateFrom = 'es';
    output.payload = `Baja confianza de ${(confidence * 100).toFixed(2)}% al
detectar mensaje`;
  } else if (language === 'en') {
    output.translateFrom = 'es';
    output.payload = `Entre su mensaje en otro idioma que no sea Ingles`;
  }
  return output;
}
```

Now create a new action “bot” with the action file

```
wsk action create bot bot.js
```

OpenWhisk Action Sequence bot-sequence

Create an action sequence of three actions “languageId”, “bot” and “translator”.

The first action languageId will receive the input parameters from the chat UI, detect the message payload, then send the results to the “bot” action.

The second action bot will receive the input parameters from languageId, and decide if it wants to translate the input message, or send back a message to the user which leverages the translation service also to translate the system message in the case from Spanish to English.

```
wsk action update bot-sequence
"Bluemix_Language_Translator_Credentials-
1/languageId","bot","Bluemix_Language_Translator_Credentials-
1/translator" --sequence -a web-export true
```

Now update the previous example chat.js and instead of invoking the web action echo invoke chat-sequence.

Modify the file chat.js replacing echo.json with bot-sequence.json:

```
var API_URL = "bot-sequence.json";
```

Modify the file chat.html to it’s clear to users what to enter

```
placeholder="enter none English here">
```

Deploy

```
zip form.zip package.json form.js chat.js chat.html
wsk action update form form.zip -a web-export true --kind nodejs:6
```

Testing

Now open a new browser tab or refresh the browser using the URL from the previous exercise

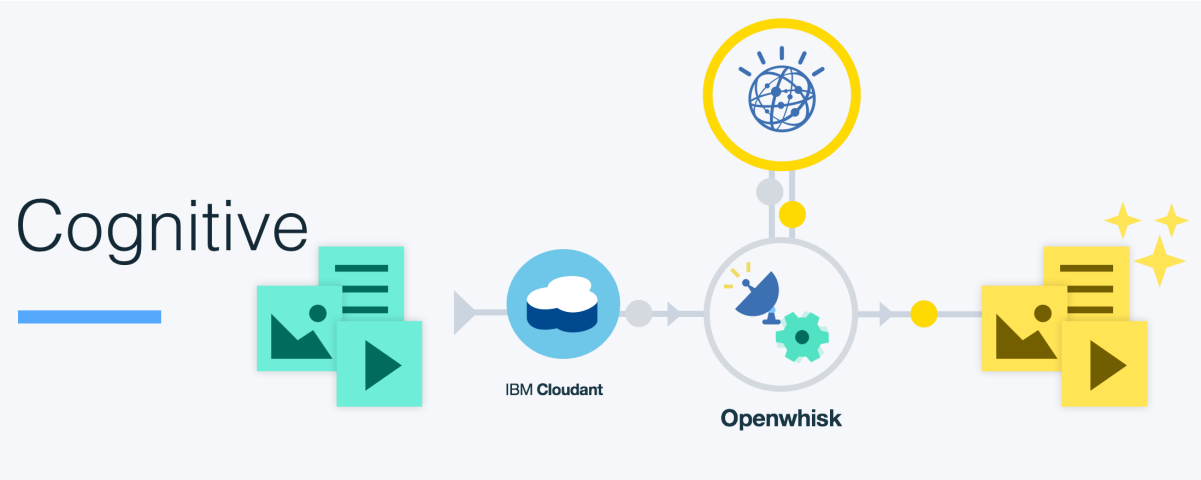
```
google-chrome
https://openwhisk.ng.bluemix.net/api/v1/web/$NAMESPACE/default/form.html
```

Now enter text into the input box in any language except **English**, then click the send button.
The response of the message will now be in English.

Cognitive and beyond

As you can see it's very easy to implement a web application using web actions, also you saw how easy is to use a Watson service to enhance your application with cognitive services.

With service like NoSLQ DB offered by Cloudant, and the many APIs from Watsons you can use OpenWhisk to build your next Cognitive Application.



Enable cognitive computing features in your app using IBM Watson's Language, Vision, Speech and Data APIs.
See the services
https://www.ibm.com/watson/developercloud/services-catalog.html
What's New in Watson:
Introducing Watson Natural Language Understanding
https://www.ibm.com/watson/developercloud/natural-language-understanding.html
Color tagging comes to Watson Visual Recognition
https://www.ibm.com/blogs/watson/2017/02/color-tagging-comes-watson-visual-recognition

Installing OpenWhisk locally

For those being interested in how to install OpenWhisk locally read here:
https://github.com/openwhisk/openwhisk#quick-start

Free-style

As part of this "free-style" slot we would like to ask you to think about a scenario you would like to implement; so, just brainstorm about what you would like to build (and what you should be able to build in the time given) and reach out to the instructors once you have questions and/or need help – they are here to assist (and, once again, they do not bite!). If you should really have no idea get in touch with the instructors, too – they will be happy to brainstorm together with you…

Survey

Finally, we would really appreciate your feedback to further improve OpenWhisk:
https://ibm.biz/BdiRbG

Learning more

- IBM Bluemix OpenWhisk:
  - https://www.ibm.com/cloud-computing/bluemix/de/openwhisk
- Apache OpenWhisk:
  - https://openwhisk.org
- OpenWhisk on GitHub:
  - https://github.com/openwhisk/openwhisk/
- OpenWhisk on Twitter:
  - https://twitter.com/openwhisk
- OpenWhisk on Medium:
  - https://medium.com/openwhisk
- OpenWhisk additional material on Slide share:
  - http://www.slideshare.net/OpenWhisk
- OpenWhisk additional material on YouTube:
  - https://www.youtube.com/channel/UCbzgShnQk8F43NKsvEYA1SA
- OpenWhisk on Slack:
  - http://slack.openwhisk.org/
- Other OpenWhisk material:
  - https://github.com/openwhisk/awesome-openwhisk
- The Serverless Framework:
  - https://github.com/serverless/serverless-openwhisk
- Node-RED:
  - https://nodered.org