

Introduction to Cloud Functions

Hands-On Lab

JeanCarl Bisson | jbisson@us.ibm.com | [@dothewww](https://twitter.com/dothewww)



Create a Cloud Function	2
Create a Web Action	5
Build a Check-In Application.....	6

The screenshot shows the IBM Bluemix Functions interface. On the left, there's a sidebar with 'Code', 'Triggers', 'Default Parameters', 'Runtime', and 'Additional Details'. The main area shows a 'hello' function with a 'Code' tab containing the following Java code:

```
1 /**
2 * This will be invoked when you Run This Action
3 * IBM Cloud Functions actions accept a single parameter, which must be a JSON object.
4 * Return the output of this action, which must be a JSON object.
5 */
6
7
8
9
10 function main(params) {
11   return { message: 'Hello World' };
12 }
```

Below the code is a 'Test' section with a 'Name' field set to 'JeanCarl' and a 'Check-in' button. To the right is a preview panel showing the response: 'Status: Hello JeanCarl' and a JSON object with 'message': "Hello JeanCarl".

Create an application that takes form data and registers guests in a Cloudant database.

The screenshot shows the Cloudant database interface. On the left, there's a sidebar with 'All Documents', 'Query', 'Permissions', 'Changes', and 'Design Documents'. The main area shows a document named 'guests' with the following details:

Document ID	Options	JSON
bfd1ab49ea946bd891268f7a333b00	<input type="checkbox"/>	{ } JSON
Create Document		

The document content is:

```
_id: "bfd1ab49ea946bd891268f7a333b00"
name: "JeanCarl"
```

Create a Cloud Function to run code on demand.

The screenshot shows the Cloudant database interface. On the left, there's a sidebar with 'All Documents', 'Query', 'Permissions', 'Changes', and 'Design Documents'. The main area shows a document named 'guests' with the following details:

Document ID	Options	JSON
bfd1ab49ea946bd891268f7a333b00	<input type="checkbox"/>	{ } JSON
Create Document		

The document content is:

```
_id: "bfd1ab49ea946bd891268f7a333b00"
name: "JeanCarl"
```



A digital copy of this lab and code snippets can be found at:
<http://ibm.biz/cloud-function-intro>



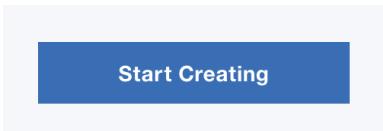
Create a Cloud Function

In this lab, we'll create a Cloud Function in IBM Bluemix. With a Cloud Function, you can write code that lives in the Cloud and can be executed on-demand and scale quickly. IBM Bluemix supports Python, Node.js, Java, Swift, PHP, and Docker environments. In this section, we'll create a basic Cloud Function.

1. Sign up for an IBM Bluemix account at <https://bluemix.net>
2. Click on the menu in the top left corner of the IBM Bluemix console.
3. Select **Functions**.
4. This is the landing page where you can find the command line tool to deploy Cloud Functions via the command line, manage and monitor actions and triggers, and expose Cloud Functions as an API with this tooling. In this lab, we'll use the web-based tooling to create a Cloud Function.

The screenshot shows the 'Getting Started with IBM Cloud Functions' page. On the left, there's a sidebar with a navigation menu under 'Getting Started' (Overview, Pricing, Concepts, Integrations, CLI, iOS SDK, Documentation), 'Manage', 'Develop', 'Monitor', and 'APIs'. The main content area features a large 'f' icon with a speech bubble and three stars, with the text 'Save costs, scale and integrate.' Below it are three small icons representing different functions or services. At the bottom of the main content area is a blue 'Start Creating' button.

5. Click on **Start Creating**.



6. Since no Cloud Functions exist, click on **Create Action** to create a new action.

The screenshot shows the 'Actions' page. It features a large circular icon with a gear symbol. The word 'Actions' is prominently displayed below it, followed by the text 'There are no Actions yet.' A blue 'Create Action' button is located at the bottom. A descriptive note at the bottom states: 'Actions contain code that perform work. They can be invoked directly via REST API or connected to services by using a trigger. [Learn more about Actions.](#)'

7. Enter an Action Name, `hello`, with a Runtime `Node.js 6`. Tick the checkbox next to **Enable as Web Action**. Click **Create** when finished.

Action Name
hello

This Action will be stored in the *Default Package*, or you can [create a new Package](#).

REST API URI [?](#)
https://.../tutorials_api/actions/hello

Runtime [?](#)
Node.js 6

Web Action [?](#)
 Enable as Web Action Raw HTTP handling

Need Help?
Contact Bluemix Sales

Estimate Monthly Cost
Cost Calculator

Cancel Create

8. A Node.js template for a basic action is prepopulated into the textbox. This action returns a JSON object with a property named `message` and a value Hello World. Let's run this action. Click **Invoke**.

Manage / hello

hello

Code

The following code will be executed each time the action is invoked. Input and output are in the form of JSON.

```
1 - /**
2 * 
3 * main() will be invoked when you Run This Action
4 * 
5 * @param Cloud Functions actions accept a single parameter, which must be a JSON object.
6 * 
7 * @return The output of this action, which must be a JSON object.
8 * 
9 */
10- function main(params) {
11   return { message: 'Hello World' };
12 }
```

Open in Develop View

Invoke

The code is executed and the last result is displayed below the code textbox.

Last Result	Change Action Input	Logs
{ "message": "Hello World" }		

9. Next, let's make this function more dynamic. If a **name** parameter is passed into the function, we'll greet the user by name. Change the code to use the value of the parameter labeled **name**.

Code

The following code will be executed each time the action is invoked. Input and output are in the form of JSON.

```
1  /**
2   *
3   * main() will be invoked when you Run This Action
4   *
5   * @param Cloud Functions actions accept a single parameter, which must be a JSON object.
6   *
7   * @return The output of this action, which must be a JSON object.
8   *
9   */
10  function main(params) {
11    var name = params.name||"Guest"
12    return { message: 'Hello '+name };
13  }
14
```

10. Click on the **Invoke** button on the right side of the page to run the code. The result includes the default name, Guest.

A screenshot of the Cloud Functions interface. On the left, there is a sidebar with various icons. In the center, there is a box labeled "Last Result" containing the JSON output of the previous invocation. At the top right of the interface, there is a link labeled "Change Action Input".

```
{ "message": "Hello Guest" }
```

11. Make the name dynamic by changing the Action Input values. Click on the **Change Action Input** link in the box labeled **Last Result**.

12. Add a JSON property labeled **name** with a value of a name, such as your first name. Click **Apply** when finished.

A screenshot of the "Change Action Input" dialog box. It has a title bar with the text "Change Action Input" and a close button. Below the title bar is a text input field containing the JSON object: `1 { "name": "JeanCarl" }`. At the bottom of the dialog are two buttons: "Cancel" and "Apply".

13. When the action is invoked with this name parameter, it is injected into the message value and the following result is returned.

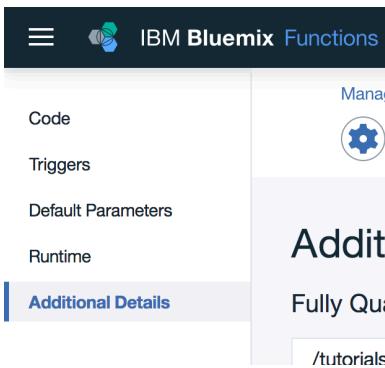
A screenshot of the Cloud Functions interface. The "Last Result" box now shows the updated JSON output. The "Change Action Input" link is still visible at the top right.

```
{ "message": "Hello JeanCarl" }
```

Create a Web Action

Cloud Functions can be made accessible as web actions, or in other words, given a URL that can be accessed by a browser or other application via an HTTP call. In this section, we'll access the Cloud Function we created in the previous section as a web action via a web browser.

1. Select **Additional Details** in the menu in the left sidebar.



2. By ticking the checkbox next to **Enable Web Action** when the Cloud Function was created in the previous section, this Cloud function was given a URL. Copy this URL into the address bar of a new web browser tab.

The screenshot shows the "Web Action" configuration page. It has a heading "Web Action" and a sub-instruction "Allow your Cloud Functions actions to handle HTTP events. [Learn more about Web Actions.](#)". There are two checkboxes: "Enable as Web Action" (which is checked) and "Raw HTTP handling" (which is unchecked). Below the checkboxes is a "Web Action URL" field containing the URL https://openwhisk.ng.bluemix.net/api/v1/web/tutorials_api/default/hello.json.

The JSON result is returned to the web browser.

The screenshot shows a browser window with the URL https://openwhisk.ng.bluemix.net/api/v1/web/tutorials_api/default/hello.json. The page displays a JSON object: { "message": "Hello Guest" }.

3. Add a query **name** parameter to the end of the URL with a custom name as shown below. Query string parameters are passed into the Cloud Functions as an argument to the main function.

.../hello.json?name=JeanCarl

The screenshot shows a browser window with the URL https://openwhisk.ng.bluemix.net/api/v1/web/tutorials_api/default/hello.json?name=JeanCarl. The page displays a JSON object: { "message": "Hello JeanCarl" }.

Build a Check-In Application

Cloud Functions are a great way to have a web service available on demand. It doesn't use any resources and you pay only for the time and resources the Cloud Function uses when you run the web action. In this section, we'll make a simple check-in application that records the name of a person into a database.

1. First, create a **Cloudant NoSQL DB** (database) in the **Data & Analytics** section of the IBM Bluemix catalog.

The screenshot shows the IBM Bluemix Catalog interface. On the left, there's a sidebar with categories like All Categories, Infrastructure, Compute, Storage, Network, Security, Containers, VMware, Platform, Boilerplates, APIs, Application Services, Blockchain, Cloud Foundry Apps, and Data & Analytics (which is selected). The main area displays various services: Apache Spark, BigInsights for Apache Hadoop, BigInsights for Apache Hadoop (Subscription), Compose Enterprise, Compose for Elasticsearch, Compose for etcd, Compose for JanusGraph, Compose for MongoDB, Compose for MySQL, Compose for PostgreSQL, and Compose for RabbitMQ. Each service has a brief description, a small icon, and two buttons labeled 'Lite' and 'IBM'.

2. Click on **Service Credentials** in the sidebar. If the list of credentials are empty, click on **New credential**.

This screenshot shows the 'Service credentials' page for a Cloudant NoSQL DB instance named 'Cloudant NoSQL DB-g6'. The left sidebar includes 'Manage', 'Service credentials' (which is selected), 'Plan', and 'Connections'. The main content area has a heading 'Service credentials' and a note: 'Credentials are provided in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.' A 'View More' button is visible. Below this is another 'Service credentials' section with a 'New credential' button and a note: 'Click New credential to create a set of credentials for this instance.'

3. Click on **View Credentials** to display the service credentials. Copy the `url` property into the code in Step #7.

```
"host": "8alc48cl-3e55-45c5-9aad-b0bfd398a7cb-bluemix.cloudant.com",
"port": 443,
"url": "https://8alc48cl-3e55-45c5-9aad-b0bfd398a7cb-bluemix:d2022e4e7f75d8badf5266e32a972fe52967
60dab83af321eb6a0e66c4141a63@8alc48cl-3e55-45c5-9aad-b0bfd398a7cb-bluemix.cloudant.com"
}
```

4. Click on the green **Launch** button to launch the Cloudant dashboard.

- Click on **Create Database** in the top right corner of the dashboard.
- Give the database a name of **guests**.

The screenshot shows the Cloudant dashboard with a modal window titled 'Create Database'. Inside the modal, there is an input field containing the text 'guests' and a blue 'Create' button to its right. The background of the dashboard shows a list of databases with columns for 'Name', 'Size', and '# of Docs'. A red box highlights the 'Create Database' button in the modal.

- Keep the Cloudant dashboard open. Open another tab and return to the Cloud Functions section in the IBM Bluemix dashboard. Create a new Cloud Function, this time named **checkin**. To create a new Cloud Function, follow the steps on page two of this lab, and stop before you modify the code. Use the following function for this section.

```
function main(params) {
  return new Promise(function(reject, resolve) {

    var name = params.name || "Guest";
    var payload = {
      name: name
    }

    var Cloudant = require("cloudant");
    var cloudant = new Cloudant(___);
    var db = cloudant.db.use("guests");

    db.insert(payload, function(err, body, header) {
      if(err) {
        reject(err);
      } else {
        payload.message = "Hello " + payload.name;
        resolve(payload);
      }
    });
  });
}
```

- This code returns a Promise function that inserts the name of the guest into a Cloudant NoSQL database, and eventually returns the response that is displayed on the webpage. Click on **Additional Details** in the sidebar to get the URL to the web action. Insert this URL in the placeholder (the following code uses the result from Step #3):

```
var cloudant = new Cloudant("https://8alc48c1-3e55-45c5-9aad-b0bfd398a7cb-bluemix:d2022e4e7f75d8badf5266e32a972fe5296760dab83af321eb6a0e66c4141a63@8a1c48c1-3e55-45c5-9aad-b0bfd398a7cb-bluemix.cloudant.com");
```

- Clone the Git repo at ibm.biz/cloud-function-intro-app. This project contains a basic webpage with a form that collects a name. When the user submits the form, an AJAX call is made to the Cloud Function which adds the name to a Cloudant NoSQL database.

Name:

- Edit the index.html file and replace the URL of the AJAX call to point to your Cloud Function web action.

```
$.ajax({
  type: "POST",
  url: "https://openwhisk.ng.bluemix.net/api/v1/web/---_----/default/checkin.json",
  data: {
    name: $("#name").val()
  },
});
```

11. Open the webpage in a browser. Enter your name in the form and submit the form. The name value is posted to the web action. The JSON response is displayed in the textbox.

Name:

Status: Hello JeanCarl

```
{  
  "message": "Hello JeanCarl"  
}
```

12. Return to the Cloudant dashboard and refresh the page. A new entry has been added to the database.

The screenshot shows the Cloudant dashboard for a database named 'guests'. On the left, there's a sidebar with icons for All Documents, Query, Permissions, and Changes. The main area displays a table with one row. The row contains a checkbox, an '_id' column with the value 'bfdb1ab49eaf946bd891268f7a333b00', and a 'name' column with the value 'JeanCarl'. At the top right, there are buttons for 'Create Document', 'Options', and a bell icon. There's also a dropdown for 'Document ID'.