



## QUICK LAB 2

NO INFRASTRUCTURE, JUST CODE. SEE THE  
SIMPLICITY OF SERVERLESS.

Create, build, and run a cloud-native Swift 4 serverless application that uses the Visual Recognition service to determine image content.

# INTRODUCTION

This lab walks you through the steps required to create, build and run a Serverless application using IBM Cloud Functions. **Serverless computing** refers to a model where the existence of servers is entirely abstracted away. Even though servers 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 other details. Serverless computing is essentially about reducing maintenance efforts to allow developers to quickly focus on developing code that adds value.

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. Some promising solutions like Apache OpenWhisk have recently emerged that ease development approaches used in the serverless model. **IBM Cloud Functions** is a Function-as-a-Service (FaaS) platform on IBM Cloud, built using the Apache OpenWhisk open source project, that allows you to execute code in response to an event.

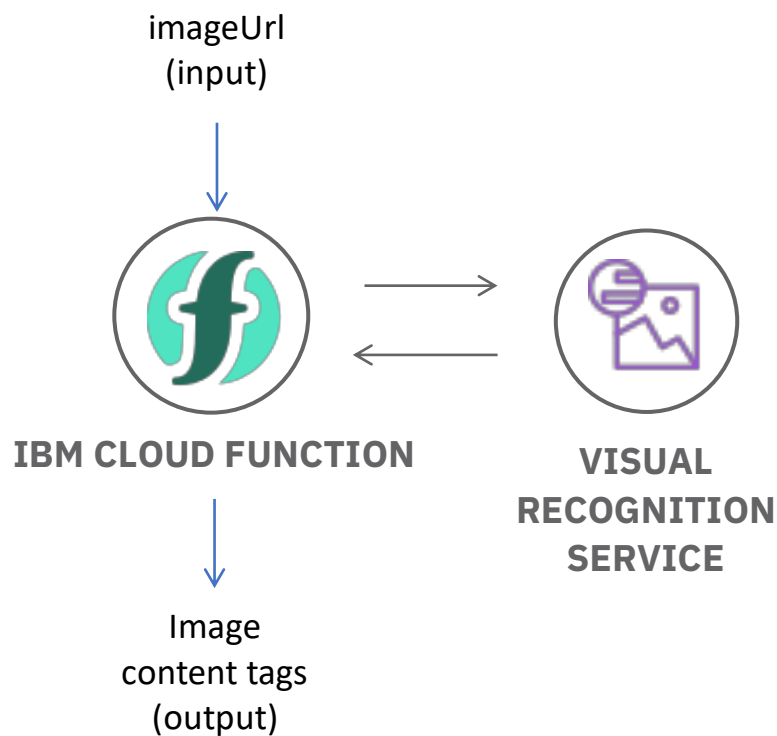
It provides a serverless deployment and operations model. With a granular pricing model that works at any scale, you get exactly the resources you need – not more, not less – and you are charged for code that is really running.

IBM Cloud Functions provides:

- Support for multiple languages, including JavaScript, Python, Swift, and Java
- Support for running custom logic through Docker containers

- The ability to focus more on value-adding business logic, and less on low-level infrastructural and operational details.
- The ability to easily chain together microservices to form workflows via composition.

In this lab, you'll create an IBM Cloud Functions action that takes an image URL as input, and returns some tags describing the content of the image. To get the tags, the action will interact with the Visual Recognition Service on IBM Cloud.



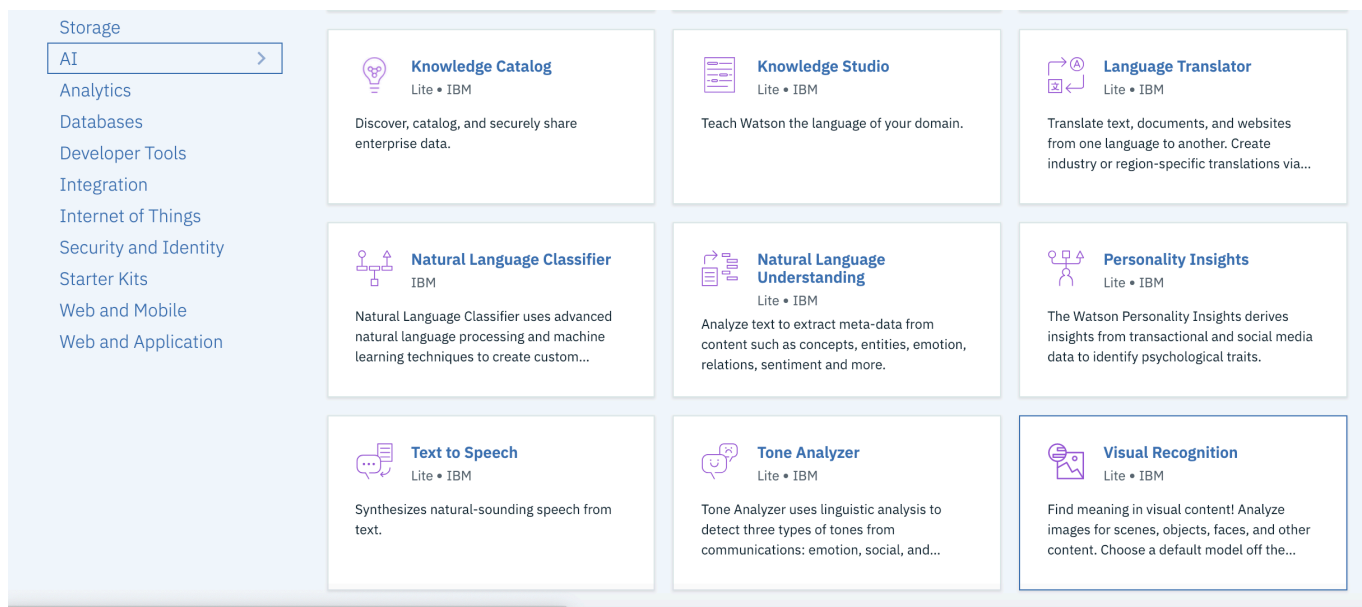
## PREREQUISITES FOR THIS LAB

- You will need an IBM Cloud Account. Either use your existing account, or create a new account by accessing the following link: <https://ibm.biz/Bd2Sfe>

# CREATE A VISUAL RECOGNITION SERVICE

For this quicklab, you will need to provision a Visual Recognition service in the IBM Cloud.

1. Start by logging into the IBM Cloud: <https://cloud.ibm.com/login>, and then select the **Create Resource** button in the top right.
2. Select **AI** in the left menu, and then select **Visual Recognition**.



3. Give your service a name, and then click **Create**.

**Service name:**

**Choose a region/location to deploy in:**

**Select a resource group:** ⓘ

**Tags:** ⓘ

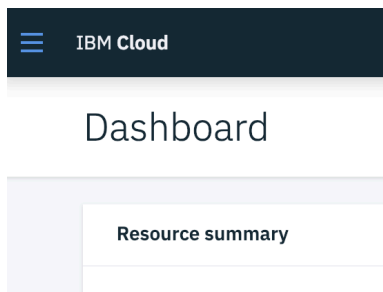
4. Click **Service Credentials** to ensure a set of service credentials were generated for you (you may need to refresh this page).
  - a. If not, click **New Credential**, and then **Add**.
5. Click **view credentials** and take note of the **apikey** provided. You will need this later on in the lab, so you may want to copy it to a notes file.

## CREATE AN ACTION IN THE CLOUD FUNCTIONS UI

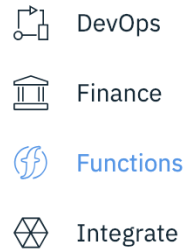
There are two main options to get started with Cloud Functions. Both allow you to work with Cloud Function's basic entities by creating, updating, and deleting actions, triggers, rules and sequences.

The CLI (command line interface) allows you to perform these basic operations from your shell. The IBM Cloud Functions UI (user interface), allows you to perform the same operations from your browser. During this lab we will use the UI to learn how to work with Cloud Functions.

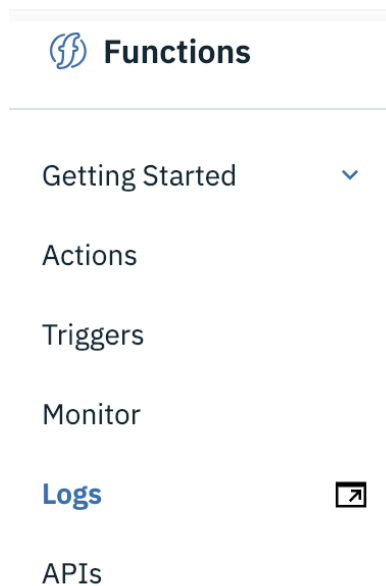
1. Select the hamburger menu in the IBM Cloud header.



2. Then click on **Functions** to access the IBM Cloud Functions development experience on IBM Cloud.

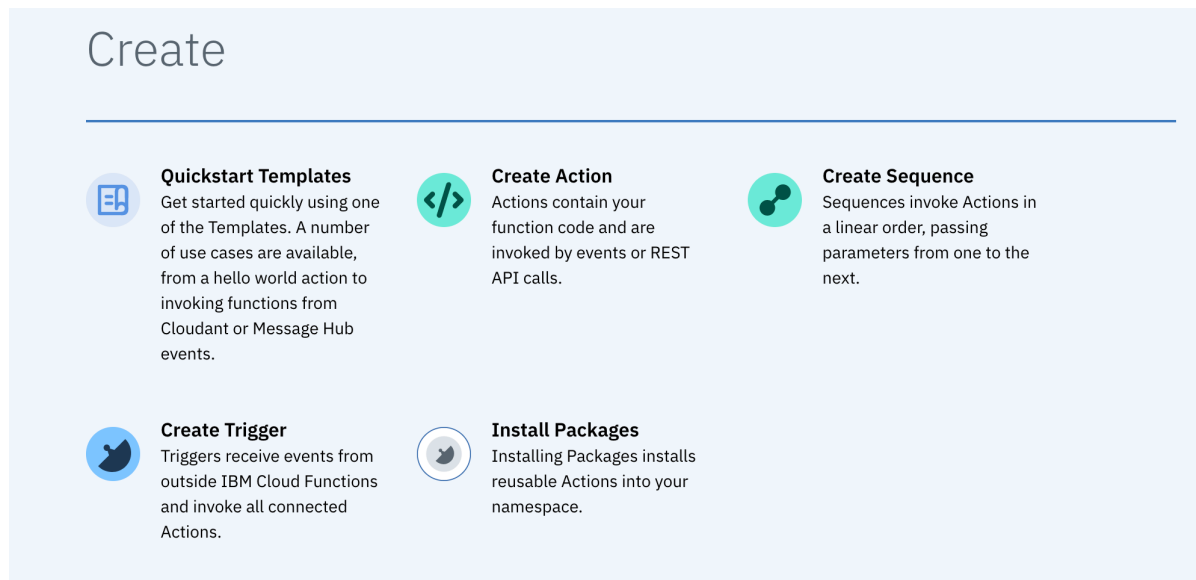


3. The Cloud Functions UI is comprised of the following sections in the left-hand side menu bar.



- a. Actions – The actions section lists all actions you have created prior. An action is a small piece of code that can be explicitly invoked or set to automatically run in response to an event.
- b. Triggers – A trigger is a declaration that you want to react to a certain type of event, whether from a user or by an event source. A trigger can be fired or activated. Triggers can be associated with actions, so that when the trigger is fired the action is run.

- c. Monitor – This section shows you information about your actions and their activity, including an activity summary and timeline.
  - d. Logs – The logs section takes you to the IBM Cloud Logging service, which provides you with the ability to collect, analyze, and build dashboards for your logs.
  - e. APIs – The APIs section allows you to set up an API Gateway and API management for IBM Cloud Functions.
4. Start creating your first action by selecting the **Start Creating** button in the center of the UI, which opens the Create page. Then select the **Create Action** button.



5. Specify an Action Name (e.g. openwhisk-vr), by entering it into the text field, and then select Swift 4 as the runtime. Leave everything else as-is and click the **Create** button at the bottom of the screen.

# 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

openwhisk-vr

## Enclosing Package <sup>i</sup>

(Default Package)

Create Package

## Runtime <sup>i</sup>

Swift 4

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



Cancel

Previous



Create


- This opens a cloud-based code editor that you can use to create and extend your actions. There should already be some hello world code in the action. Note that it uses the **Swift 4 Codable Protocol** for encoding and decoding json inputs and outputs.
- Click **Invoke** to test this action directly from within your browser. You should see an Activations panel show up with the result. Because you did not pass any parameters in, the result should be “Hello stranger!”

Code <sup>i</sup> Swift 4

Change Input  Invoke 

```
1 2
2  *
3  * main() will be run when you invoke this action
4  *
5  * @param Cloud Functions actions accept a single parameter, which must be Codable.
6  *
7  * @return The completion function, which takes as output a Codable and Error?.
8  *
9  */
10 struct Input: Codable {
11     let name: String?
12 }
13 struct Output: Codable {
14     let greeting: String
15 }
16 func main(param: Input, completion: (Output?, Error?) -> Void) -> Void {
17     let result = Output(greeting: "Hello \(param.name ?? "stranger")!")
18     completion(result, nil)
19 }
20
```

Activations Collapse  Clear 

▼  openwhisk-vr 3958 ms 1/17/2019, 19:07:23

**Activation ID:**  
263c675ffb434547bc675ffb43354774

**Results:**  
{ "greeting": "Hello stranger!" }

**Logs:**  
[ "2019-01-18T00:07:23.575282061Z stdout: Compiling",  
"2019-01-18T00:07:23.575336837Z stdout: swiftc statu  
s is 0",  
"2019-01-18T00:07:23.575343833Z stdout: Linking"  
]

- Actions may be invoked with a number of named parameters. Let's try adding parameters. Click on the **Change Input** button, and update the parameters with



the following json:

```
{"name": "Belinda"}
```

×

Change Action Input

Edit mode - press **ESC** to exit

1 `{"name": "Belinda"}`

Cancel

Apply

- Click the **Apply** button, and then click **Invoke** again to invoke your action. You should see an activation result with the name you provided.

Activations

Collapse 

Clear 

✓	✓	openwhisk-vr	17 ms	1/17/2019, 19:12:52
<div><b>Activation ID:</b> 7d1ef530c4e246d49ef530c4e266d4bc</div> <div><b>Results:</b> <pre>{   "greeting": "Hello Belinda!" }</pre></div> <div><b>Logs:</b> <pre>[]</pre></div>				

# USE THE BUILT IN VISUAL RECOGNITION SDK FROM YOUR SWIFT 4 ACTION.

Each IBM Cloud Functions runtime comes with some packages already pre-installed to the environment. For example, the Node runtime includes `bodyparser`, `koa`, `lodash` and a number of other useful libraries. The Swift runtime includes the Watson Developer Cloud SDKs (Software Development Kits) including the visual recognition SDK we'll use today. We'll import this visual recognition SDK to make calls to the service in a swift-native way.

1. Replace the hello world swift code with the following code, found on the next page:

```

import VisualRecognitionV3

struct Input: Codable {
    let imageUrl: String
    let apiKey: String
}

struct Output: Codable {
    let classes: String
}

func main(param: Input, completion: @escaping (Output?, Error?) -> Void) -> Void {
    // set up visual recognition sdk
    let apiKey = param.apiKey
    let version = "2018-03-19"
    let visualRecognition = VisualRecognition(version: version, apiKey: apiKey)

    let imageUrl = param.imageUrl
    let failure = { (error: Error) in print("err",error) }
    var tags = ""

    // make call to visual recognition classify function
    visualRecognition.classify(url: imageUrl, failure: failure) { classifiedImages in
        let image = classifiedImages.images.first
        let classifier = image?.classifiers.first
        let classes = classifier?.classes
        for theclass in classes! {
            print(theclass.className)
            tags = tags + theclass.className + ", "
        }
        let result = Output(classes: tags)
        completion(result, nil)
    }
}

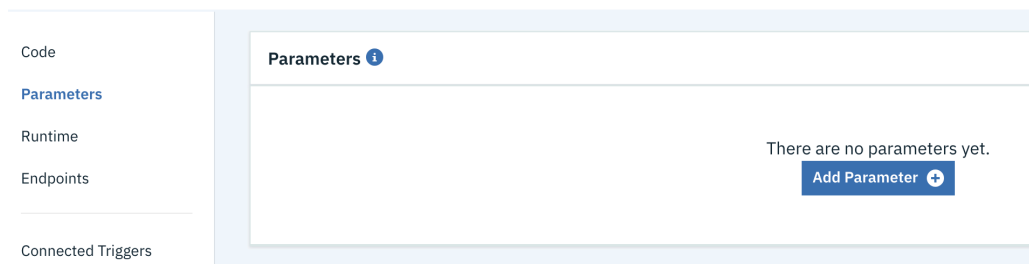
```

2. Click **Save**. Look over the code. You can see we're importing the **VisualRecognitionV3** SDK as promised. Find where we're instantiating the SDK

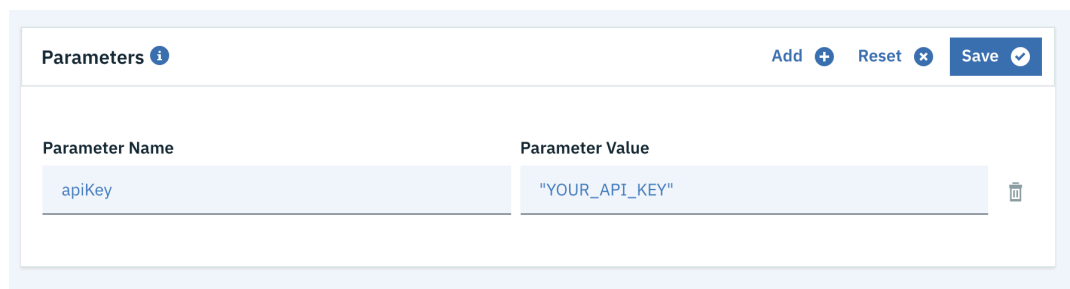
(around line 16). You can see that we will need the apiKey we saved before. This action expects the apiKey to be passed in as a parameter.

```
let apiKey = param.apiKey
let visualRecognition = VisualRecognition(version: version, apiKey: apiKey)
```

- Default parameters can be set for an action, rather than passing the parameters into the action every time. This is a useful option for data that stays the same on every invocation. Let's set the apiKey as one of our default parameters. Click **Parameters** in the left side menu, and then click **Add Parameter +**.



- For parameter name, **apiKey**, with a capital **K**. For parameter value, insert your apiKey value enclosed in quotation marks.



- Click **Save**.

## USE THE BUILT IN VISUAL RECOGNITION SERVICE TO CLASSIFY AN IMAGE

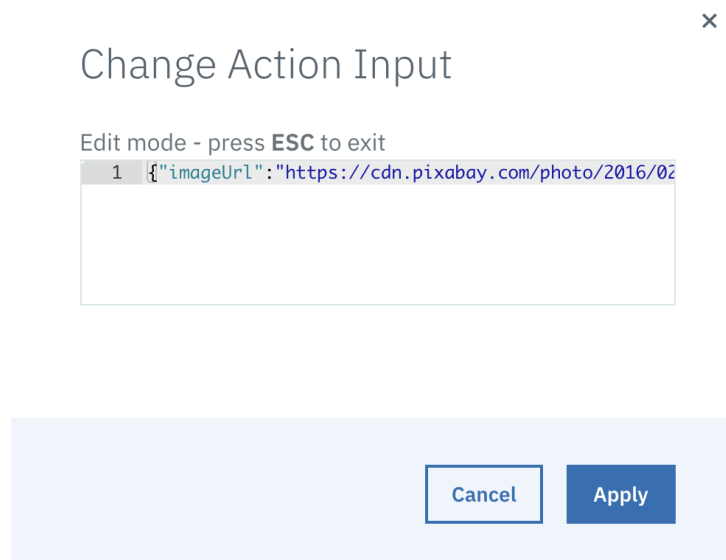
- Let's inspect the code a little more. You can see that this action also takes as input an imageUrl. It then passes that imageUrl to the visualRecognition service,

and does some simple parsing of the results – ultimately returning the “classes” representing the contents of the image.

```
struct Input: Codable {
    let imageUrl: String
    let apiKey: String
}
struct Output: Codable {
    let classes: String
}
```

2. Change the input for this function to be an image URL by clicking **Change Input**, and then pasting in the following **json**.

```
{"imageUrl":"https://raw.githubusercontent.com/beemarie/ow-  
vr/master/images/puppy.jpg"}
```



3. This is an image of a cute puppy. Click **Apply**.



4. Click **Invoke** to run the action. This action will pass the image to the Visual Recognition service to classify, then parse the returned information, and finally output just the classes or tags of the image.
5. You should see some results in the Activations window like **Labrador Retriever, dog, pup, and animal**.

**Activations** [Collapse](#) [Clear](#)

✓ openwhisk-vr 2804 ms 1/17/2019, 19:55:55

**Activation ID:**  
b60cfe08eb8441198cfe08eb84511933

**Results:**  

```
{
  "classes": "Labrador retriever dog, retriever dog,
dog, domestic animal, animal, pup, young, pale yellow
color, "
}
```

## CONCLUSION

**Congratulations!** You have completed this lab. You have successfully created and used a Visual Recognition service. You have also built and deployed a Serverless Cloud Function, saw some Swift 4 code and features, and used the Visual Recognition SDK built in with the language runtime – all from within a browser! Feel free to reach out should you have any questions.