



QUICK LAB 2

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

Create, build, and run a cloud-native Python 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.

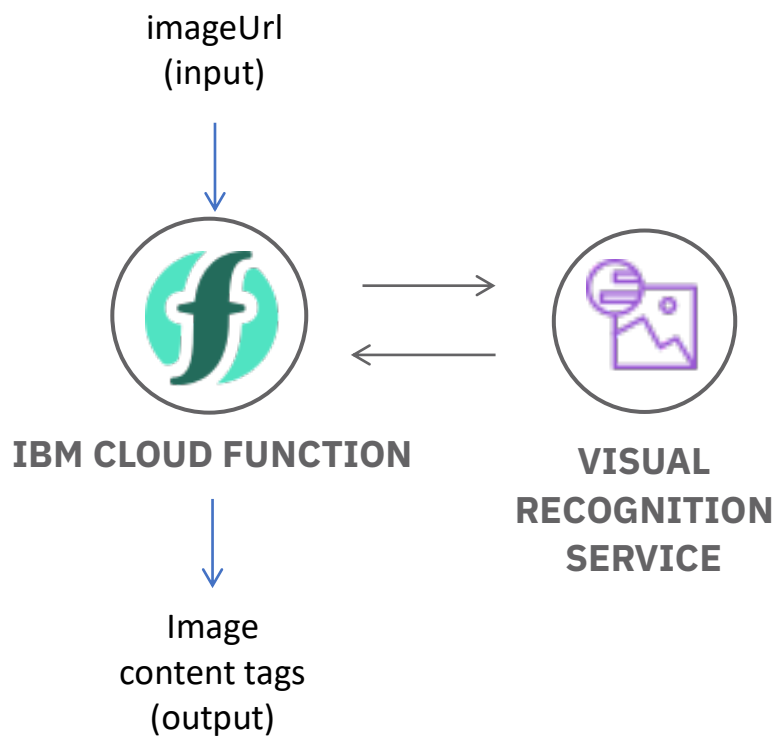
It provides a serverless deployment and operations model. With a granular pricing model that works at any scale, you get exactly the resources you need – not more, not less – and you are charged for code that is really running.

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.



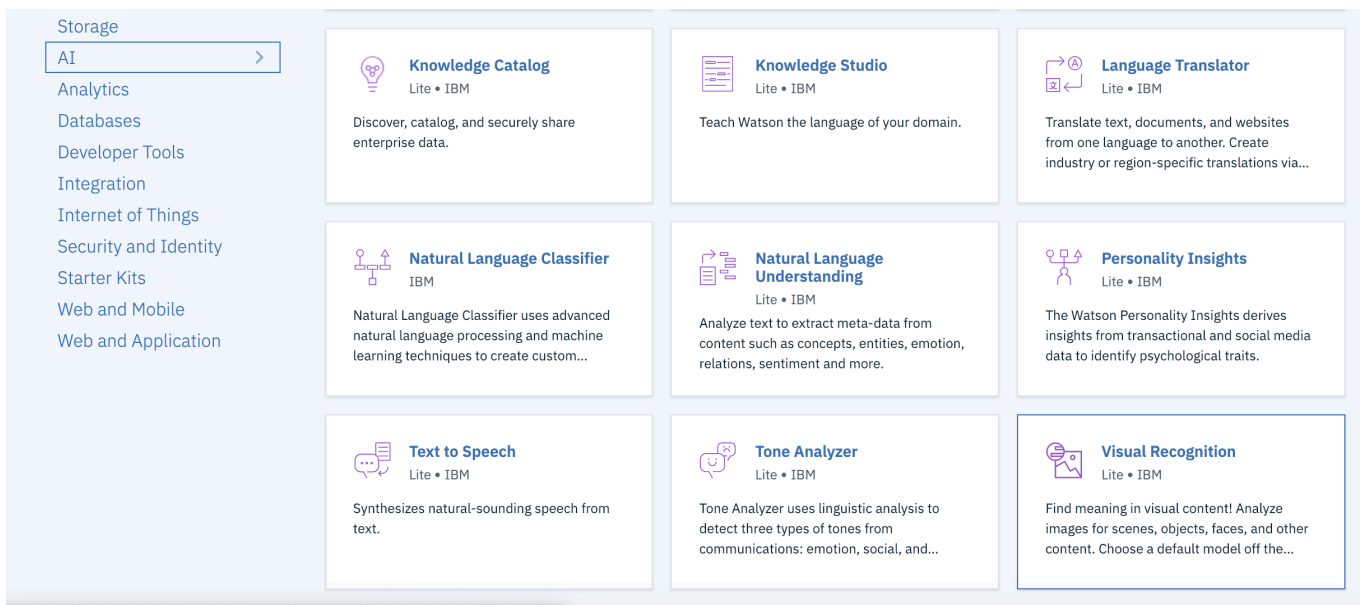
PREREQUISITES FOR THIS LAB

- 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/BdzhQy>

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://ibm.biz/BdzhQy>, and then select the **Create Resource** button in the top right. If you do not see **Create Resource**, you can select **Catalog**.
2. Select **AI** in the left menu, and then select **Visual Recognition**. You can create only one free **lite tier** resource with the free IBM Cloud account. If you already have a visual recognition service, either delete it and follow these steps or you can skip to step 4 to get the credentials.



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

The screenshot shows a form for creating a new service in the IBM Cloud Functions UI. The form is light blue and contains the following fields and buttons:

- Service name:** A text input field containing "Visual Recognition-bmv".
- Choose a region/location to deploy in:** A dropdown menu with "Dallas" selected.
- Select a resource group:** A dropdown menu with "default" selected, accompanied by an information icon (i).
- Tags:** A text input field with an information icon (i) and placeholder text "Examples: env:dev, version-1".
- Buttons:** Two buttons at the bottom right: "Add to estimate" (outlined) and "Create" (solid blue).

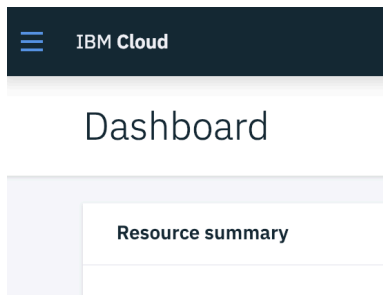
4. Click **Service Credentials** to ensure a set of service credentials were generated for you (you may need to refresh this page).
- a. If not, click **New Credential**, and then **Add**.
5. Click **view credentials** and take note of the **apikey** provided. You will need this later on in the lab,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

CREATE AN ACTION IN THE CLOUD FUNCTIONS UI

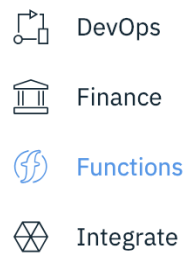
There are two main options to get started with Cloud Functions. Both allow you to work with Cloud Function's basic entities by creating, updating, and deleting actions, triggers, rules and sequences.

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.

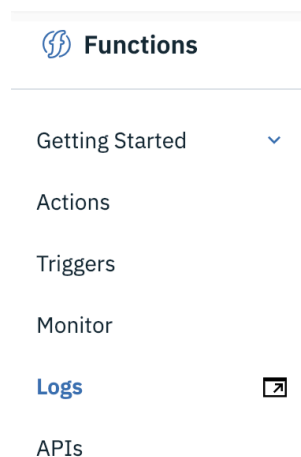
1. Select the hamburger menu in the IBM Cloud header.



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.


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.

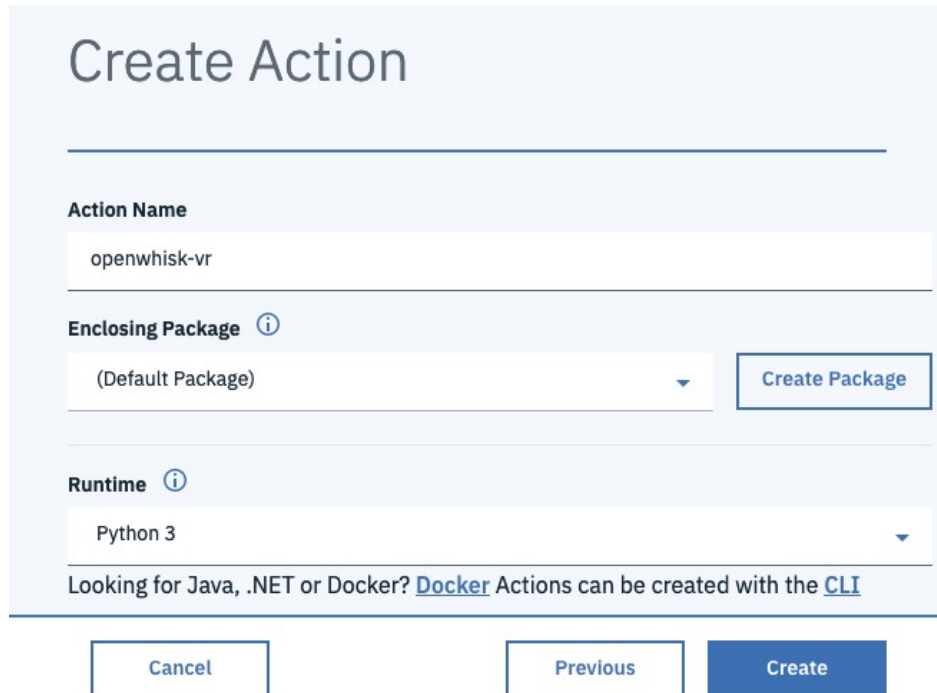


Create Trigger
Triggers receive events from outside IBM Cloud Functions and invoke all connected Actions.



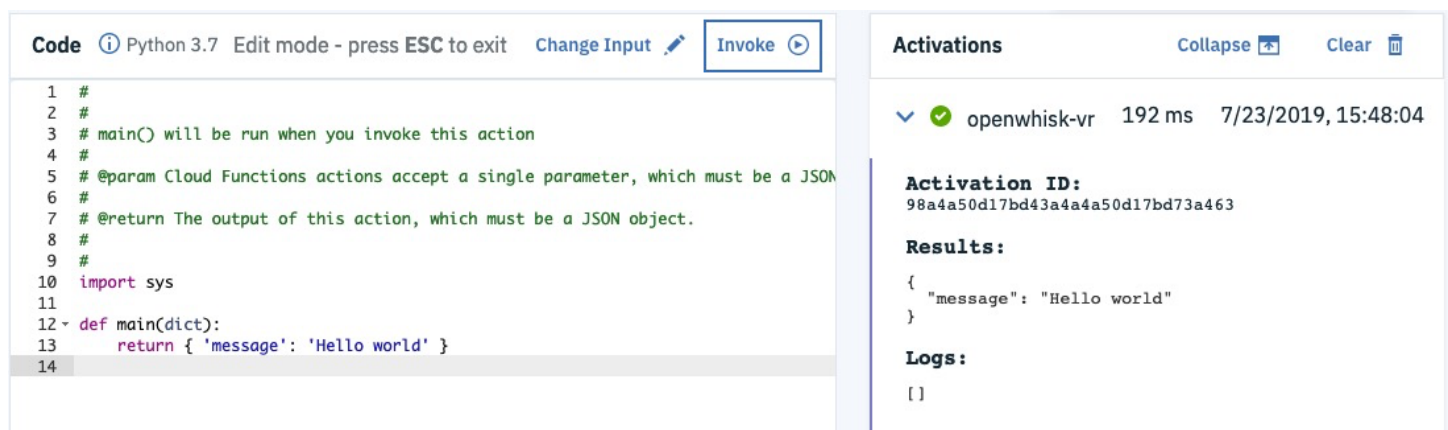
Install Packages
Installing Packages installs reusable Actions into your namespace.

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



The 'Create Action' form is displayed on a light blue background. It features a title 'Create Action' at the top. Below the title is a horizontal line. The form has three main sections: 'Action Name' with a text input field containing 'openwhisk-vr'; 'Enclosing Package' with a dropdown menu showing '(Default Package)' and a 'Create Package' button; and 'Runtime' with a dropdown menu showing 'Python 3'. At the bottom, there are three buttons: 'Cancel', 'Previous', and 'Create'. A link at the bottom reads: 'Looking for Java, .NET or Docker? [Docker](#) Actions can be created with the [CLI](#)'.

- 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.
- Click **Invoke** to test this action directly from within your browser. You should see an Activations panel show up with the result. The result should be “Hello world”



The image shows a code editor interface with a Python 3.7 script and an 'Activations' panel. The code editor has a header with 'Code', 'Python 3.7', 'Edit mode - press ESC to exit', 'Change Input', and an 'Invoke' button. The code is as follows:

```
1 #
2 #
3 # main() will be run when you invoke this action
4 #
5 # @param Cloud Functions actions accept a single parameter, which must be a JSON
6 #
7 # @return The output of this action, which must be a JSON object.
8 #
9 #
10 import sys
11
12 def main(dict):
13     return { 'message': 'Hello world' }
14
```

The 'Activations' panel on the right shows a single activation for 'openwhisk-vr' with a duration of 192 ms and a timestamp of 7/23/2019, 15:48:04. It includes an 'Activation ID' and 'Results'.

Activation ID:
98a4a50d17bd43a4a50d17bd73a463

Results:
{
 "message": "Hello world"
}

Logs:
[]

USE THE BUILT IN VISUAL RECOGNITION SDK FROM YOUR PYTHON ACTION.

Each IBM Cloud Functions runtime comes with some packages already pre-installed to the environment. The Python 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 python-native way.

1. Replace the hello world Python code with the following code, found on the next page. You can copy paste this code from this github gist:

<https://ibm.biz/openwhisk-vr-1>

```
from watson_developer_cloud import VisualRecognitionV3

def main(params):
    # init visual recognition library
    apiKey = params['apiKey']
    version = "2018-03-19"
    visual_recognition = VisualRecognitionV3(version=version, iam_apikey=apiKey)

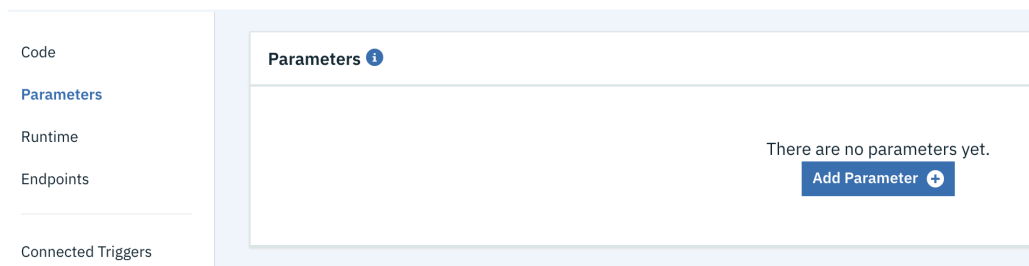
    # get image url from params
    image_url = params['imageUrl']

    # parse visual recognition return data for our tags
    tags = ""
    classifiedImages = visual_recognition.classify(url=image_url).get_result()
    image = classifiedImages['images'][0]
    classes = image['classifiers'][0]['classes']
    for theClass in classes:
        currentTag = theClass['class']
        print(currentTag)
        tags = tags + currentTag + ", "
    result = {'classes': tags}
    return result
```

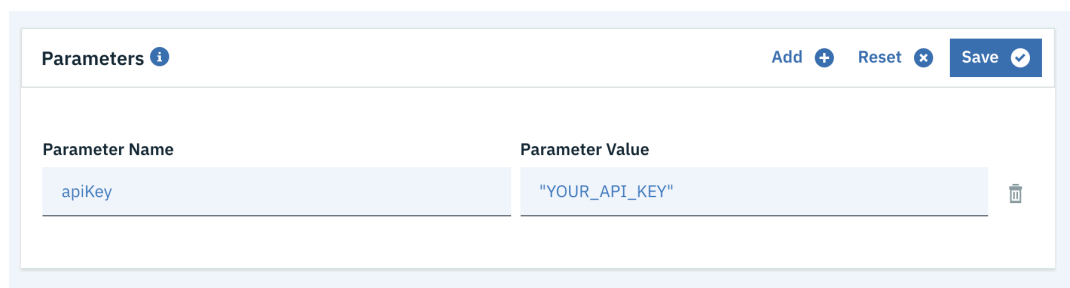
2. Click **Save**. Look over the code. You can see we're importing the **VisualRecognitionV3** SDK as promised. Find where we're instantiating the SDK (around line 7). You can see that we will need the apiKey we saved before. This action expects the apiKey to be passed in as a parameter.'

```
# init visual recognition library
apiKey = params['apiKey']
version = "2018-03-19"
visual_recognition = VisualRecognitionV3(version=version, iam_apikey=apiKey)
```

3. 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 +**.



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



5. Click **Save**.

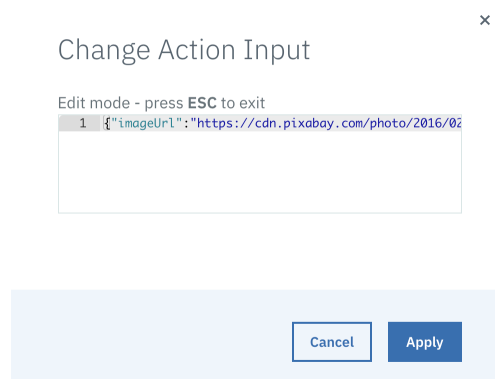
USE THE BUILT IN VISUAL RECOGNITION SERVICE TO CLASSIFY AN IMAGE

1. 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.

```
classifiedImages = visual_recognition.classify(url=image_url).get_result()
image = classifiedImages['images'][0]
classes = image['classifiers'][0]['classes']
for theClass in classes:
    currentTag = theClass['class']
    print(currentTag)
    tags = tags + currentTag + ", "
result = {'classes': tags}
```

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**.

ActivationsCollapseClear

openwhisk-vr

2276 ms

7/23/2019, 16:11:35

Activation ID:
4838cde99053463fb8cde99053b63fef

Results:

```
{
  "classes": "puppy, dog, domestic animal, animal, Labrador retriever dog, retriever dog, golden retriever dog, pale yellow color, light brown 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 Python 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.