Lab work

Basics of Apache OpenWhisk

Actions and triggers

Mathieu Bacou
mathieu.bacou@telecom-sudparis.eu

2024 – 2025
Télécom SudParis
Institut Mines-Télécom & Institut Polytechnique de Paris

Part I. Introduction

1. Overview

The cloud has evolved towards providing a paradigm of cloud-native applications, where the latter are built to be deployed specifically in the cloud. This paradigm brings promises of infinite performance scaling with great money savings.

The first embodiment are microservices, however they still require thinking about server provisioning, with all the constraints and chores that come with it. The answer is to go serverless: the cloud user that wishes to deploy their application should not be burdened with any server-side consideration. Instead, they should focus on writing the core of their application and then configuring the architectural requirements. The application is typically written as elementary functions, and deployed on a Function-as-a-Service (FaaS) platform. The architectural dependencies, usually long-lived server-like features, are provided by a Backend-as-a-Service (BaaS) offering.

Actual servers are obviously still required, but they are managed by the cloud provider; on continuity with the previous labs, we chose Google Cloud Platform (GCP)[1] to provide the resources. However, you will use it only as the support for a Kubernetes cluster, similarly to the first lab, on which you will deploy a FaaS platform: Apache OpenWhisk (OW)[2]. Thus, you will have the point-of-view of both the cloud provider and a cloud client.

This lab will guide you through the installation and setup of OW in part III, and then give you a brief primer on its command-line interface (CLI) in part IV.

2. Prerequisite

Knowledge of using a terminal is assumed. A very basic understanding of Kubernetes's pods, and a good understanding of the serverless notions seen during the lecture are welcome.

The lab makes use of resources from GCP, which is a paid product. For

[1] *Cloud computing services | Google Cloud*. 2011. URL: `https://cloud.google.com/` (visited on 12/06/2020).

[2] *Apache OpenWhisk is a serverless, open source cloud platform*. 2017. URL: `https://openwhisk.apache.org/` (visited on 12/04/2021).

this class, you should have received credits to redeem on GCP using a Google account; on this note, you might prefer creating an account linked to your institutional e-mail address instead of using a personal account.

As for technical requirements, almost all operations take place in the Kubernetes cluster in GCP. Thus, the only prerequisites are to install:

- Google Cloud SDK, and more specifically to configure its `gcloud` command to access your GCP project (instructions can be found on Google Cloud Docs: `https://cloud.google.com/sdk/docs/install#installation_instructions`);
- `kubectl` (it will be configured to access your cluster when it is created);[3]
- `helm`[4], a "package manager" for Kubernetes.[5]

Alternatively, you may opt to use GCP's Cloud Shell (a shell embedded in GCP's web pages): `gcloud`, `kubectl` and `helm` are already available and configured there. You will also be able to install the additional necessary programs to interact with OW.

Warning

You will be working with a paid Google Kubernetes Engine (GKE) from the cloud provider Google Cloud Platform. Underneath, it runs on nodes, i.e., virtual machines (VMs). The pricing model is that you pay for your VM resources as long as the VM, and thus the GKE node, is up, whatever its activity. Thus: *destroy the GKE cluster at the end of the lab.*

Part II. Setup

The steps to deploy OpenWhisk on a GKE cluster are summarized in a script.

Fetch and extract the archive containing the deployment script and its helper script by running the command in listing 1.

[3] You may use your package manager, or follow the instructions at `https://kubernetes.io/fr/docs/tasks/tools/install-kubectl/`.

[4] *Helm*. 2017. URL: `https://helm.sh/` (visited on 12/04/2021).

[5] You may use your package manager, or follow the instructions at `https://helm.sh/docs/intro/install/`.

Warning

For Mac users: the script "make_mycluster.sh" expects GNU/Linux binaries that behave differently from their Mac OS counterparts. The script will thus not work.

You may either:

- follow the lab in Google Cloud Shell;
- use a VM that runs an up-to-date Linux distribution, such as a recent Ubuntu version.

Listing 1: Download base files.

```
curl https://www-inf.telecom-sudparis.eu/COURS/CSC5004/practicals/\
basics-openwhisk/basics-openwhisk.tgz | tar --extract --gzip
```

You should obtain two Shell scripts, "deploy.sh" and "make_mycluster.sh".

Information

While not mandatory, you may want to read and understand the steps taken by "deploy.sh" to deploy OpenWhisk on a GKE cluster.

Part III. Installation of Apache OpenWhisk on a Google Kubernetes Engine cluster

Information

The deployment script will default to the Google Cloud location `europ-west9-c`, a datacenter in Paris.

Run the script "deploy.sh" to deploy OpenWhisk, as shown in listing 2.

Listing 2: Deploy OpenWhisk.

```
./deploy.sh
```

Question 1

The first deployment step undertaken by the script, is to provision a Kubernetes cluster with two nodes of type `e2-medium`.

- What does it mean on the cloud backend side, to provision a cluster on two nodes of the specified machine type?
- Why are virtual machines still used even in a Serverless cloud?

Question 2

Get the list of pods with `kubectl --namespace openwhisk get pods`.

- Match OpenWhisk's components seen in the lecture, with the pods listed by the command above.
  - There are more deployed pods than components in OpenWhisk's architecture: ignore `alarm-provider`, `ingress-nginx-controller`, `init-couchdb` and `wskadmin`.
- How could you make the OpenWhisk platform itself scale, to handle more requests to its services? Be specific to OpenWhisk's components seen in its architecture diagram.

Warning

The certificate of your OpenWhisk deployment, although working to provide HTTPS access, is still self-signed. As indicated at the end of the deployment script, you will have to tell `wsk` to ignore verifying the certificate, with the flag `--insecure` (short-hand `-i`). You are advised to define an alias to avoid typing it every time.

In the next part, you will test the deployment by executing sample functions and using triggers.

Part IV.
Using Apache OpenWhisk: actions and triggers

In this part, you will:

1. test your OpenWhisk deployment by invoking a sample action;
2. create a trigger;
3. link it to an action with a rule to create an automated pipeline.

From this point on, the lab uses OpenWhisk vocabulary:

action a cloud function;
to invoke to run a cloud function;
trigger an event channel that can be used to invoke an action through a rule;
rule a link between a trigger and an action;
activation an invocation of an action;
package a collection of actions or triggers under a namespace.

An object of OpenWhisk (action, trigger, etc.) is kept under one namespace, i.e., a path-like name that always begins with a slash "/".[6] Then, it may be kept under one package, i.e., another path-like name. For example, the fully qualified name of the sample action you invoke in the next section is "/whisk.system/samples/greeting": the namespace is "/whisk.system",[7] the optional package is "samples", and the name of the action is "greeting".

Information

To answer code questions below, find the correct invocation of the `wsk -i` command. It is organized with subcommands that you can list with `wsk --help`; and then you can get help for each subcommand by providing it the `--help` flag.

[6]The namespace may be omitted for your own objects: each user has a default namespace.
[7]This namespace is special because using a period is otherwise forbidden by the platform.

3. Invoking an action as simply as possible

OpenWhisk comes with a catalog of curated actions, under the namespace "/whisk.system". It includes a package "samples" that gathers sample actions.

Coding task 1

Test your deployment: invoke the action "/whisk.system/samples/greeting" in the simplest possible way.

Question 3

Explain what happens when you invoke an action:

- How does the `wsk` command demands the invocation? Re-run the invocation with the added flag `--verbose` to help you answer.
- What components interact with each other?

Coding task 2

The command from the previous question returned immediately with an activation ID.

Watch the activation by *polling* for the arrival of new activation events. When you see the action's activation, quit polling.

Coding task 3

Get information about the activation, as well as its logs, by using its activation ID.

Warning

There is a bug in OpenWhisk that may not actually refresh the activation records: *you may have to run the commands twice*!

Question 4

In the activation record *of this first invocation of the action*, identify:

- the execution time;
- the time spent in the platform;

- the time spent initializing the runtime;
- the result returned by the action.

Question 5

How exactly was the action executed at the Kubernetes cluster level? Look at the list of pods (`kubectl --namespace openwhisk get pods`) for a beginning of explanation.

Congratulations, you invoked your first OpenWhisk action!

Information

The next time you want to get information about the last activation, you can use the `-l` flag instead of providing the activation ID.

4. Invoking an action with parameters

The action was invoked without parameters, so it returned a default message.

Coding task 4

Invoke the action again, but this time providing values to its parameters `name` and `place`. In addition, make `wsk` wait for the result returned by the action and display it, instead of printing the activation ID.

You can see in the result how your parameters were taken into account. Congratulations, you invoked your first OpenWhisk action with parameters!

Question 6

You should have noticed that the following invocations were much faster than the first one.

- To confirm this observation, check the times in the activation data: which one has disappeared?
- What is the name of the mechanism behind this execution time optimization? Confirm your answer by checking the columns “Start” and “Duration” in the list of activations.

- This time, how exactly was the action executed at the Kubernetes cluster level?

5. Using triggers and rules

Cloud functions are best used as automatic responses to events in a channel. For instance, you could have your cloud storage system emit an event when a new file is stored into it; OpenWhisk would in turn invoke an action, with the event's data as parameters.

In OpenWhisk, it is achieved by:

1. creating a trigger: this is the channel endpoint on OpenWhisk's side;
2. linking the trigger to an action: tell OpenWhisk to trigger this action when the trigger is fired by an event.[8]

You will create a trigger representing the arrival of a new person, and set up a rule to invoke "/whisk.system/samples/greeting" upon firing the trigger.

Coding task 5

Create the trigger named "new-person".

Information

In more complex situations, the trigger may be given default parameters, that will be passed to the action invoked by firing the trigger.

The trigger may also be given a *feed action*: this is an action invoked by OpenWhisk to manage the remote source of events, when the user executes operations to manage the trigger itself on the OpenWhisk side.

For instance, in the example of a trigger fired when a new file is created on a cloud storage:

- upon creating the trigger, the feed action is invoked to actually set up the event production by the cloud storage system, so that it sends the events to OpenWhisk's trigger;
- upon deleting the trigger, the feed action is invoked to disable the event production by the cloud storage system.

[8] This architecture means that you can invoke multiple different actions from a single trigger.

Coding task 6

Set up a rule named "say-hello" to invoke "/whisk.system/samples/greeting" when "new-person" is fired.

Coding task 7

Test the trigger "new-person" by firing it manually. You can pass parameters to the trigger when firing it: they will be passed to the invoked action.

Information

As explained above, in a real-world system, the firing is done by the external event source communicating with the OpenWhisk platform to fire the trigger, as configured by the feed action. This most often works via HTTP REST requests, instead of invoking the `wsk` command, that is only the user interface.

Coding task 8

`wsk` answered to firing the trigger with an activation ID.

Confirm the action "/whisk.system/samples/greeting" was invoked with the parameters given when firing the trigger.

Congratulations again! You set up your first OpenWhisk trigger to automatically invoke an action. You can clean up after this lab by:

1. deleting the rule "say-hello";
2. deleting the trigger with "new-person".

Part V. Conclusion

The Google Kubernetes Engine cluster to which you deployed OpenWhisk, being deployed on Compute Engine VMs, will continue to consume credits.

Warning

You will probably work on the next lab, so keep the cluster running. However, if you don't need to use the cluster anymore, i.e., if you will not follow another lab on serverless using it, *destroy the cluster!*

In this lab, you discovered Apache OpenWhisk after deploying it in a Kubernetes cluster. You invoked a sample action, and set up a trigger to invoke it as part of an automated system.

Were you an actual serverless cloud user, you would have directly used Google or another cloud provider's native serverless offering, saving you the burden of managing a server – the GKE cluster and OpenWhisk deployment.

To go further, follow the next lab on scaling a web service with serverless.