Serverless Swift

Apache OpenWhisk for iOS developers

Marek Sadowski
Lennart Frantzell

Apress®

Serverless Swift

Apache OpenWhisk for iOS developers

Marek Sadowski
Lennart Frantzell

Apress®

Serverless Swift: Apache OpenWhisk for iOS developers

Marek Sadowski
Walnut Creek, CA, USA

Lennart Frantzell
Sunnyvale, CA, USA

ISBN-13 (pbk): 978-1-4842-5835-4
ISBN-13 (electronic): 978-1-4842-5836-1
https://doi.org/10.1007/978-1-4842-5836-1

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Aaron Black
Development Editor: James Markham
Coordinating Editor: Jessica Vakili

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-5835-4. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

To Marta, Mikolaj "Nick," Helena, my brother,
and my parents – for your love
To my grandchildren Nessa and Niko
To Raymond Camden, Andrew Trice, Chris Bailey,
and Neil Patterson, without whom we wouldn't dare

Table of Contents

About the Authors

Marek Sadowski is a full-stack developer advocate, a robotics startup founder, and an entrepreneur. Born in Poland, he has about 20 years of experience in consulting large enterprises in America, Europe, Japan, the Middle East, and Africa. As a graduate from the International Space University, Marek pioneered research on VR goggles for the virtual reality system to control robots on Mars in NASA Ames in 1999. He also founded a startup to deliver robotics solutions and services for industries. In 2014, Marek moved to Silicon Valley to promote Edge, IoT, robotics, and mobile solutions driven with AI, APIs, and Cloud native.

Lennart Frantzell is a developer advocate with IBM in San Francisco, focusing on Blockchain and AI. Born in Sweden, Lennart moved to Silicon Valley in the late 1980s to work with AI technology, especially with Expert Systems. He worked on a team that specialized in taking prototypes from IBM Research and productizing them, making them ready for distribution all over the world. When the "AI Winter" put the brakes to development of Expert Systems, Lennart moved to object-oriented programming and from there to the IBM Internet Division, part of the burgeoning Internet and Web movement in the late 1990s.

About the Technical Reviewer

Matt Rutkowski is an STSM and Master Inventor at IBM developing open infrastructure and industry standards along with open source for over 20 years in areas including banking, digital media and entertainment, and security compliance and specializing in Cloud for the last 9+ years. Most recently, he is the IBM lead for and a committer to the Apache OpenWhisk Serverless computing project at Apache Software Foundation (ASF) serving on its Project Management Committee and as a committer. In addition, he has worked on Cloud Orchestration, Security, Audit, and Compliance standards. Specifically, he has chaired and been lead editor for such standards as OASIS Topology Orchestration for Cloud Applications (TOSCA), OASIS CloudID, and DMTF Cloud Auditing (CADF) which he founded. Furthermore, he has contributed to implementations of these standards within communities such as Apache, CNCF, and OpenStack.

Swift Book Forward

Whenever I present on Serverless computing, I often start by affirming that the name itself is a misnomer and joke some clever marketing person coined it. In retrospect, it is actually a disservice to name such a powerful technology by what it takes away. Granted, I am not going to try to attempt to rename it here nor would I want to try to. Instead, let me suggest that Serverless is best viewed by what it attempts to enable which is allowing programmers to write efficient functions that perform some cool task in their favorite language and not care at all about where it runs or how it scales.

In this book, you will be doing just that, learning how to write Swift language functions that implement some of the most popular use cases that drive adoption of Serverless. These use cases should cause light bulbs to go off as they clearly showcase how turning to Serverless for many common programming tasks can garner large savings in terms of compute costs while reducing operational overhead dramatically. It is my hope that each and every developer who finishes the contents of this book will be able to recognize these patterns when tasked to write some new Cloud-based service and choose to do so using Serverless technology.

Notice that I used the word "attempts" in the first paragraph; allow me to explain. After describing this vision of Serverless and my wishes for you to embrace it, we should have a reality check. Serverless is still trying to figure itself out. What I mean is that despite being generally available in some form as a compute technology offered for more than a few years via services such as AWS Lambda, MS Azure, or IBM Cloud Functions, there is no real standard for how it is implemented. Therefore, the programming and deployment models, the programming conventions, the supported

languages, and even the use cases themselves will be presented in different ways (if at all) from different providers. Some Serverless platform providers may not even have a thought-out programming model, a disservice we will discuss, or even include some of the most important features needed to implement the use cases.

Do not despair. In the end, if you write good RESTful functions that are not tightly coupled to proprietary frameworks or APIs, tooling is available to help you easily package them to whichever platform serves your needs the best. The Cloud providers will, in my opinion over time, all evolve to acknowledge the same patterns and strengths and come to support a common programming model based on an event-centric, observer pattern. This book, I am proud to say, chose the Apache OpenWhisk platform which, in my biased opinion, is the gold standard for Serverless. Why do I say that? Because OpenWhisk represents a platform that was originally designed and implemented by many great minds in IBM Research who not only understood the value of creating a highly scalable platform around the observer pattern but also placed the simplicity and usability of the developer as the top priority. After being donated to the Apache Software Foundation (ASF), OpenWhisk only got better as it was battle-hardened and fine-tuned by top-notch developers from around the world. Each of these developers strove to make OpenWhisk code efficient and performant for both their own personal use cases and those of their customers. Even better, they made sure OpenWhisk could be run anywhere and on any Cloud, public, private, or hybrid where you can run containers. Kubernetes became the preferred framework, but experimental deployments have also shown that it could be run on Mesos or OpenShift if needed. I should confess some of my favorite community members actually run OpenWhisk themselves on AWS or IBM Kubernetes services. This is a great way for companies that want to explore hosting their own Serverless platform to test-drive the technology. My colleagues

at Apache have even had great success running minimal configurations of the platform for local development or as part of edge computing platforms, which is a nice segue.

What does the future hold for Serverless? When I said that Serverless is trying to "figure itself out," what I more accurately should say is that those who "get it" are running with it. Those that see the goals of event-driven, reactive programming come to fruition without having to stand up even a service framework being realized, now want to take it further. Truthfully, when I think of and talk about Serverless, I mean Function-as-a-Service (FaaS) because in order for the technology to go to places it needs to go to, it needs to be small, fast, and lean. However, so many of us have had to and are still working hard just to migrate legacy applications to Cloud likely using Containers popularized by Docker. Even some of us, excited to adopt Serverless as the ultimate reactive programming platform, have found that we have "bound" great functional code to proprietary service frameworks. In these cases, Containers may be the only simple option to bring our function, whole stack in tow, with us in order to attempt to advantage Serverless scaling characteristics without spending countless hours decoupling code. But never accept that running functions in containers equates to Serverless or is its culmination simply because you can scale them.

As you can imagine with all the code locked in legacy and proprietary frameworks, the reliance and focus on advantaging Containers will go on for quite some time, but Containers are not the end vehicle that will carry Serverless to these new and exciting places. Even as I write this, people in the Apache OpenWhisk community and other cutting-edge visionaries are trying to see how they optimize the best Serverless use cases by taking them "to the edge." Focusing on running simple functions in response to events, what a good Serverless platform does, such as OpenWhisk, aligns well for processing and analyzing data for most modern data needs. This means using Serverless "under the covers" to process event data generated

by millions upon millions of Internet of Things (IoT) devices or handle requests to rapidly prepare and serve data to mobile devices as part of Content Delivery Networks (CDNs). Additionally, as you will learn in this book, one of the premier use cases for Serverless is the ability to quickly create APIs in Public Clouds where functions can sniff, modify, or enhance data to and from some existing backend services. If you ever write a new Cloud-based service and do not use Serverless APIs for your frontend, you are likely missing out on many advantages and savings.

In fact, if the programmer-oriented vision I laid out earlier for Serverless holds true, there may be a future where the term "Serverless" evaporates entirely as it just becomes "good Cloud Programming". Swift developers specifically, being highly aware of programming efficiently for mobile and wearable devices, may be the best poised to understand and take Serverless where it needs to go. Indeed, I am quite excited to see an uptick in adoption and discussion of Swift in Serverless circles from the readers of this book to help shape its future.

Cheers to you as you take Serverless for a ride, go "off road," and perhaps take it to see places it has not yet been!

Matt Rutkowski
IBM STSM, CTO for Open Serverless Technologies
June 2020

CHAPTER 1

Introducing Serverless

Today "the Cloud" is everywhere, it permeates our lives, and it is impossible to imagine ourselves without it. Without Cloud technology, we wouldn't have globally available applications like Airbnb, Uber, Facebook, Google, IBM Cloud, Netflix, Apple iTunes, Amazon, and Microsoft Azure, to name but a few Cloud-based services.

Cloud services do not depend on dedicated hardware servers, but on ephemeral APIs[1] that can be accessible by everyone, from anywhere, on any type of device, from desktops to smartphones. All it takes is just a computer, like a lightweight laptop, or even a smartphone.

So the Cloud is really thousands and thousands of APIs coupled to swarms of cheap hardware, and into this revolutionary mix we introduce "Serverless," a new programming model that is changing the world.

Serverless: The next generation of Cloud computing

Serverless has become the new Cloud programming model, which only requires the programmer to pass in the Function-as-a-Service to the Cloud API for execution. It involves no operations or any maintenance for you as a Cloud developer.

[1]APIs, a plural of an API – that is a short for an application programming interface – which one would use to access a function of an application.

M. Sadowski and L. Frantzell, *Serverless Swift*,
https://doi.org/10.1007/978-1-4842-5836-1_1

Using the Serverless programming model, you can efficiently deploy your APIs with a minimum effort. And serve your APIs without worry while just paying for actual code usage, since the code is being executed by an API consumer.

If you are a client-side developer for iOS in Swift or other mobile or traditional platforms using other programming languages or if you are a Cloud engineer and you want to get hands-on experience in using Serverless – the latest and greatest Cloud-side technology – this book is for you.

Serverless will help you develop lean, Cloud-based APIs that are as cost-efficient and lightweight as possible for consumer and business APIs alike. Serverless technology has been mainstreamed by well-established enterprises like Amazon, Microsoft, Google, and IBM and embraced by startups like Slack and so on where Serverless effortlessly enriches stand-alone applications almost effortlessly with information sourced in Social Networks, results of big data processing, or supported by Artificial Intelligence (AI) and Machine Learning (ML).

If the Cloud became the social and industrial infrastructure of today, then Serverless technology is the latest generation of Cloud services. Serverless services are superlight thanks to dynamical allocation of machine resources. Those resources are allocated only when needed, instead of forcing the developer to pre-arrange these resources ahead of time in order to be used later for the estimated earlier loads. Furthermore, in order to stay on the safe side, the typical server-based resources are often oversized and rarely saturated. While Serverless technology allows operators to adapt and respond to the given load, the provided computers are being billed for only the gigabyte seconds of the allocation that was actually used by the applications. Moreover, organizations that use Serverless are responsible only for the functions they deploy, while Cloud operators are taking care of all the maintenance and operations of the underlying libraries, operating systems, and hardware.

Note Event-based modern architecture is allowing developers to decrease time to market. This chapter answers the questions what it is, when, why to use it, and how.

Traditional client-server computing has dominated the computer scene since at least the 1980s. Clients connected to beefy dedicated servers, called on-premise (or on-prem servers), which provided the compute and data storage needed for crunching the problems and changing the world. The fall of the Berlin Wall in November 1989 not only proved the superiority of Western democracy but also of the client-server model on which democracy was based.

In the beginning of the third millennium, with newly minted millennials, the Cloud made its entry on the world scene with Amazon's Elastic Compute Cloud. Startup companies realized that they could move their own computer services from their own dedicated data centers to Amazon's Cloud and cut their initial investments and decrease the startup costs dramatically. Instead of paying for hardware upfront, they paid for the access to hardware in time slices.

Servers were cheaper in the Cloud than in each startup's data center, but you still had to pay for maintenance and support on an ongoing basis. What made Serverless revolutionary is that you can now replace monolithic servers with spinning applications on virtual machines (VMs) with Serverless functions. These functions are only invoked by client requests when they are needed. Serverless is a revolution which, going forward, will have repercussions throughout the world. So now the change concerns moving from paying per servers to actual gigabyte seconds the CPUs were spinning for the Serverless functions.

So what is **Serverless** more precisely? Let's consider the following typical scenario for the Serverless function.

A user uploads a picture to her/his Cloud account. As soon as the picture is loaded, it generates an event (the new picture in the folder). As soon as our app detects this event, it triggers a function. The function sends a request to an AI-based Visual Recognition service that tags the pictures. Thanks to provided tags, the app can now update the automatic description and catalogue the picture according to the pre-trained classifiers of Visual Recognition. In such a way, the application may provide a user with suggestions for the image classifications. At the same time, the AI analysis can make the service providers aware if the uploaded content might be Not Suitable for Work (NSFW) due to the explicit or harmful content.

Please see the following simple example implementation of Hello World Serverless function that responds to a simple text input with a simple greeting. As an input, you provide a name in the JSON format:

```
{
    "name": "Marek & Lennart"
}
```

Your Serverless function will respond with the greeting message customized with the provided name. The function itself is written in IBM Cloud in Functions – the IBM implementation of the Apache OpenWhisk project, an open source Serverless platform hosted in IBM Cloud. Since IBM Cloud provides a generous free tier for developers who want to test the examples in this book, they will be based on this flavor of the Open Source project (you might want to use a different provider of Apache OpenWhisk). Your first implementation of a greeting function in Swift will look like this:

```
func main(args: [String:Any]) -> [String:Any] {
  if let name = args["name"] as? String {
    return [ "greeting": "Hello \(name)!" ]
  } else {
    return [ "greeting": "Hello stranger!" ]
  }
}
```

This implementation of the Serverless function is authored for the Apache OpenWhisk Swift 4.2 runtime which is hosted as part of the IBM Cloud Functions service running in IBM Cloud. In addition, this function was written in the built-in, language-aware editor that comes with the IBM Cloud Functions user web interface. Figure 1-1 shows how the Swift function looks in the browser.

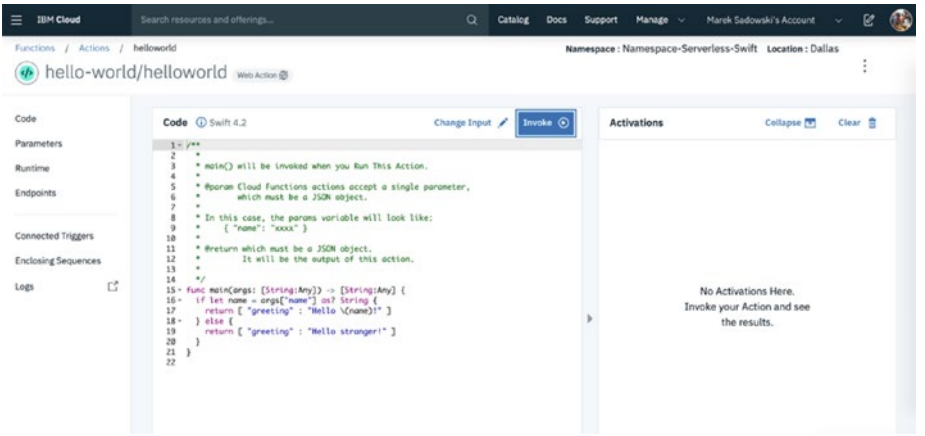

Figure 1-1. *Serverless function implemented in Apache OpenWhisk as it seen in the IBM Cloud Functions browser-based editor*

When your Serverless function is invoked, the result would look like this:

```
Activation ID:
45a4f1af985e4b93a4f1af985e3b9374
Results:
{
  "greeting": "Hello Marek & Lennart!"
}
Logs:
[]
```

The results will appear on the right-hand side of your editor in a browser (Figure 1-2). You might notice the very short time of the Serverless function execution – especially after the function has been "warmed up" (i.e., called and installed in the memory of the Serverless function engine – we will discuss more on "warming up" functions in the following chapters).

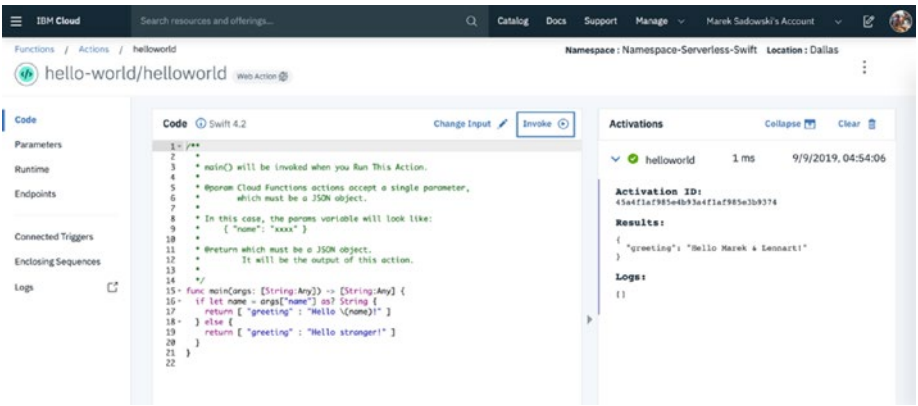

Figure 1-2. *The result of an execution of the Serverless function*

The function was created with the help of the Quickstart Templates of a Hello World function for Swift 4.2 language – please see Figures 1-3 and 1-4 for your reference.

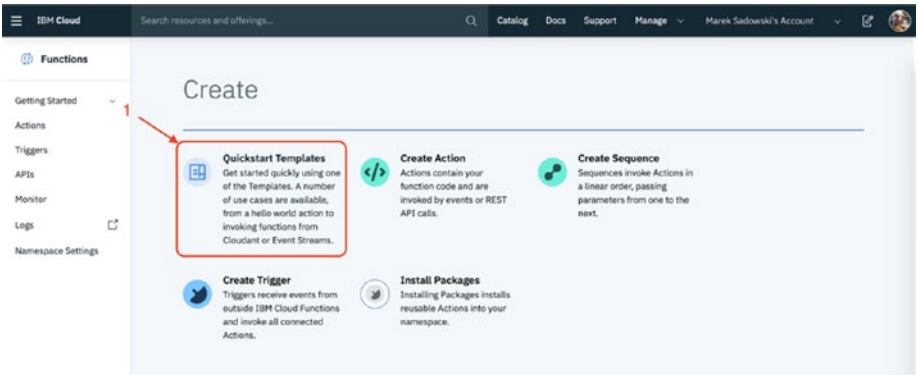

Figure 1-3. *Selecting a Quickstart Templates in IBM Cloud*

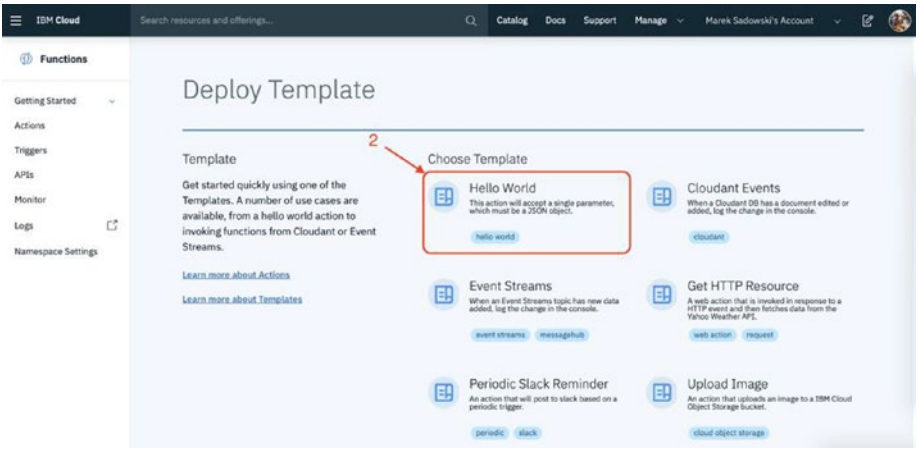

Figure 1-4. *Hello World Serverless template-based example from IBM Cloud*

Finally, the resulting Swift function will appear in your Action library as shown in Figure 1-5.

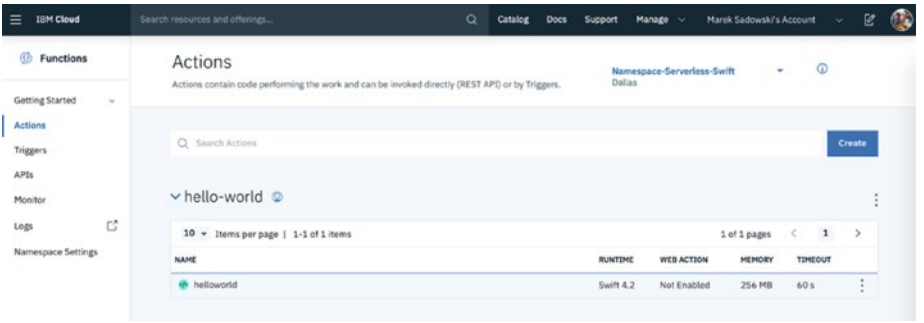

Figure 1-5. *Hello World Serverless function in the Actions library in IBM Cloud*

The Action here represents the basic executable element of Apache OpenWhisk, which when called produces a result in a JSON text format (Figure 1-6).