Towards sustainable ecosystems for cloud functions

Yessica Bogado-Sarubbi, Walter Benitez-Davalos, and Fabio Lopez-Pires

Information and Communication Technology Center
Itaipu Technological Park
Hernandarias, Paraguay
Email: {yessica.bogado,walter.benitez,fabio.lopez}@pti.org.py

Josef Spillner

Service Prototyping Lab
Zurich University of Applied Sciences
Winterthur, Switzerland
Email: josef.spillner@zhaw.ch

Abstract

The main technologies around modern cloud development and deployment paradigms such as Function-as-a-Service (FaaS) environments follow a typical technology life-cycle. Starting with basic code installation and execution environments, they unfold into a complete ecosystem with rich collaborative development and market enablement tools. In this paper, we analyse the growth of such ecosystems, reveal causes of hindrances in previous service-oriented approaches, and present a vision of how an ecosystem with sustainable operation could look like both in general and specifically for cloud functions. We present Function Hub, a partial prototypical implementation to gain first knowledge about the potential operational ecosystem behaviour.

Keywords: Cloud Applications, Serverless Computing, Service Ecosystems, Tangible Microservices, Marketplaces.

1 Problem statement

In the late 1990s and early 2000s, the idea of marketplaces to exchange software development artefacts arose in the research communities for Component-Based Software Engineering (CBSE) and later Service-Oriented Software Engineering (SOSE).

According to the early visions, rich ecosystems would emerge based on centralised exchanges such as registries implementing the Universal Description, Discovery and Integration (UDDI) specification including the Universal Business Registry (UBR) (Saini 2016). Yet, on a large scale, the ideas mostly failed. Instead, several technology-specific exchanges, often decentralised, have emerged in recent years. After several iterations across web technologies, heavy-weight and light-weight cloud technologies and more recently fog computing approaches, a few of them have manifested as useful practical tooling. A brief overview on relevant technology-specific exchanges is presented next in Table 1.

Technology	**Exchange**
Source code	GitHub
Java libraries	Maven
Containers	Docker Hub
Web services	Programmable Web
Cloud services	Open Service Broker API

Table 1: Overview on technology-specific exchanges.

Hence, the initial ideas increasingly become viable albeit often in limited form, focusing on pragmatic productivity gains especially for rapid prototyping of composite applications and services.

In the context of previously described ecosystems, emerging computing paradigms such as Function as a Service (FaaS) (Baldini et al. 2017) taking into account serverless computing architectures and implementations with cloud functions, relevant challenges still remain open as the ones presented as follows.

- Who operates such exchanges (e.g. brokers or marketplaces) in a sustainable way?
- If the operation of these exchanges is not centralised and not linked to concrete business models, how can it be sustained?
- Which future exchanges will emerge for new technologies such as cloud functions in seemingly serverless computing environments?
- How these future exchanges may look like?

Starting by answering the last question, this paper presents our position by outlining an ecosystem vision for exchanging and collaboratively developing software with cloud functions. Additionally, a comparison of the conceptual ecosystem against existing ones is presented, and main reasons about the sustainability aspects are included to give answers to the remaining three questions.

2 Ecosystem analysis

For conceptualisation and contextualisation, an analysis of the actual software ecosystem environment is needed. In this section a definition is presented as well as an analysis of ecosystems current growth and obstacles.

2.1 Ecosystem Definition

In this paper an ecosystem is defined to be an open system consisting of a set of technical elements which enable growth by attracting contributions from participants. Among the elements are service-oriented elements (marketplaces, hubs, brokers), as well as platforms and further provider and consumer tools.

2.2 Growth of ecosystems

Ecosystems can be measured primarily by looking at provider and consumer metrics which cover both the absolute volume and the growth factors. Yu observes the co-evolution of ecosystems on a wider scale from a provider perspective involving hardware, system software and software-implemented applications (Yu 2011). Among his key observations is the slowdown effect which occurs when moving up the stack. The dependent products grow slower with logarithmic relation compared to the independent product. From a consumer perspective, Petsas et al. reveal in the context of mobile application ecosystems that download statistics do not follow a *Zipf distribution* (Petsas et al. 2013). In these systems, there are moreover few alternative system software choices, leading to a comparably huge offering of applications.

2.3 Obstacles and hindrances

A key concern is the stakeholder structure behind the operating entities of ecosystem elements. Multiple popular exchanges are operated by a single commercial player. In case of bankruptcy or change of business model, the exchange may vanish or fade into obscurity quickly. The ownership structure also distorts the analysis of the economic viability. Docker Hub, for instance, is operated by Docker, Inc. but does not itself generate direct revenue. Instead, it is cross-financed by other company operations whose growth is in turn supported by the dominance of the exchange on the market. Another issue is the concentration of providers in ecosystems. For example, images on Docker Hub are prone to several security issues. This situation also affects the officially maintained images from which the issues propagate into derivatives. Any security vulnerability that could be exploited has the potential to affect vast shares of development and deployment workflows in critical applications as a consequence (Shu et al. 2017).
In summary, main identified obstacles and hindrances could be single commercial owners and concentration of providers in ecosystems.

3 Value proposition and position

This paper propose to strive towards sustainable ecosystems for heterogeneous application development artefacts which can be customised for arbitrary domains including cloud and mobile applications and so forth. The sustainability will be achieved by **decentralisation** and **abstraction**. Arguing that by a suitable combination of decentralised ecosystem elements with built-in abstraction capabilities, a long-term growth and sustainable operation can be achieved even in volatile environments with changing technologies and market forces.

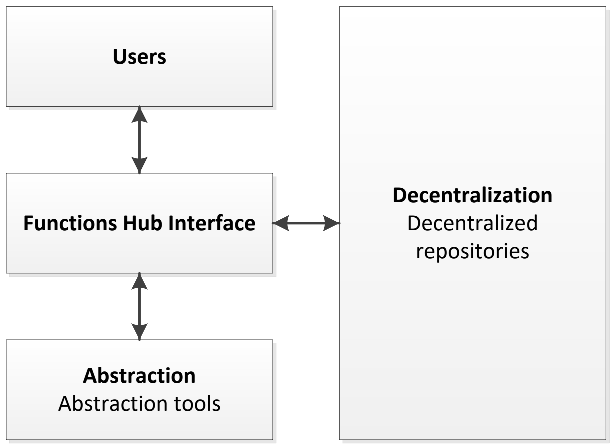

Figure 1: General view of the proposed ecosystem.

The considered arguments are as follows.

- **Decentralisation:** In recent years, several applications on the Internet have become centralised conglomerate services. Yet over longer periods of time, they may not prevail (DeLegge & Wangler 2017). A decentralised ecosystem guarantees that in a worst case even when individual market participants vanish, the system will continue to function in reduced form.

- **Abstraction:** The right level of abstraction is important. While existing ecosystems have only become successful for technologically specialised artefacts, there is a large spectrum between these and fully generic approaches which can be exploited. This exploitation can be partially automated by converting formats and protocols.

To include these characteristics in a serverless ecosystem, the following approach has been establish as shown in Figure 1. For decentralisation, a set of decentralised repositories that allows users to download, test and use cloud functions directly. These repositories will be connected through a marketplace, where developers will be able to share, deploy and even commercialise cloud functions. For abstraction, tools to convert and deploy functions will be used, allowing deployment on a diverse set of cloud providers.

4 Conceptual elements perspective

This section covers detailed design and architecture of the core elements within the proposed ecosystem.

4.1 Marketplaces

The idea of a marketplace is to create an environment where developers could interact with the platform ecosystem in a way that allows them to create, share and trade tools, enabling users to deploy, scale and create functions more easily and efficiently.
In that sense, industry already has some marketplaces from other software ecosystems. For example, GitHub has the GitHub Marketplace, a marketplace where third-party companies create and commercialise integration tools that allow users to work more efficiently with their source code. Additionally, Docker has DockerStore, a place where third-party companies offer plugins and certified containers to docker users. This allows them to access enterprise solutions and create industry-ready applications using the Docker ecosystem. Amazon Web Services (AWS) Marketplace from Amazon, allows both users and third-party companies to trade tools that uses the AWS ecosystem. At the time of this writing, only Amazon has a first prototype of a serverless ecosystem with some still basic features[1].
From the above mentioned example marketplaces, some relevant characteristic that the marketplace should have can be specified:

- Enable third-party companies and/or users to create tools that interact with the ecosystem.

- Enable third-party companies and/or users to commercialise their add-on tools.

- A framework that allows users to collaborate with each other within the ecosystem.

- Integration with other software platforms (Docker, GitHub, AWS, among others).

One characteristic that stands out of these software platforms is marketplace centralisation. This could lead to potential software disruptions in case of change of policies or departure of mayor stakeholders.

[1]https://aws.amazon.com/serverless

4.2 Converters

Even deployment-ready applications have to deal with syntax associated with each cloud platform. For instance, every serverless cloud platform has its own Application Program Interface (API), methods and syntax to deploy functions, generating a vendor lock-in issue, where users have to create functions for specific cloud providers instead of a general one.
A feasible solution is to automatically add wrappers through a converter, allowing this way developers to create code according to its real requirements, without worrying with the code necessary to make it run on a specific cloud environment. With the advent of mobile edge computing, this could be a very interesting approach to federate cloud providers.

4.3 Deployers

To accomplish a total integration with the industrial cloud ecosystem, a flexible tool that allows users to deploy their functions on multiple cloud environments is needed. Most of the tools used in industrial cloud environments are associated to its own platforms. Amazon Cloud Formation, Google Cloud Deployment Manager and Azure Resource Manager are examples of local frameworks that manage the set of resources for its own specific platform.
For cloud multi-tenancy, for example, allowing migration from one cloud provider to another is a key feature. An example of a tool that currently applies this concept is the serverless framework[2], an open source Command Line Interface (CLI) for building serverless architectures and event-driven applications, using specific software resources for deployment of functions in a wide set of cloud platforms.

4.4 Execution environments

To easily create and deploy functions, an execution environment has to be set in place. Each cloud provider focuses its environment in accordance to an aimed developer group or their specific infrastructure. For example, AWS Lambda supports major programming languages such as Java, JavaScript, Python, C# and lately Go. Each of these languages with their most popular runtimes like Node.js, Java 8, Python 2 and 3, but they add extra features allowing users to interact with the infrastructure. On the other hand, Azure Functions supports major development languages for Microsoft like C#, F# , JavaScript or PHP. Nevertheless, the field of the deployment of functions does not belong only to private platforms. In this context, one of the most popular open source cloud platform is Apache OpenWhisk, that execute functions in response to events, supporting programming languages like JavaScript/Node.js, Swift, Python, Java and even implementing Docker logic.

[2]https://github.com/serverless/serverless

4.5 Interaction and serverless ecosystem

In serverless architectures, cloud providers have complete management over the environment in which functions run. These creates high-level abstract environments for the users, where they should not worry about deployment or maintenance and expects it to be fault-tolerant with auto-scaling features (Baldini et al. 2017). This architecture is basically a event-driven computation pattern that promotes loosely couple services and ensures a trigger function execution (Stigler & Stigler 2018).
The interaction between users, interfaces and repositories is given in a way that presents a continuous exchanging of functions and allows the growth of a marketplace. The interface creates the link between all repositories and users. Because each repository is personal, every user could create a new repository and has the option to share theirs or store functions from other users. As a way of allowing abstraction, every function is created as simple as possible, to after that, establish a conversion to match a cloud provider specific requirement.

4.6 Related ecosystems

At the time of this writing, a new serverless ecosystem for sharing functions in AWS Lambda is the AWS Serverless Application Repository. This repository allows users of AWS Lambda to create, share and use functions on that specific environment. The main drawback of this new ecosystem is the vendor lock-in issue, because all components are associated with this specific provider, resulting in a centralised non-abstract ecosystem. Additionally, it only allows registration for users with an associated bank account (credit or debit card), difficulting students to access this repository and associated functions.
In contrast, the new approach considered for the proposed ecosystem, let users create their own repository to share, test and deploy functions. Remembering that functions are created as simple as possible, they could

be converted and deployed in different cloud environments, making it easy to create functions associated with a specific product without having any dependencies over a certain provider.

5 Proof of concept and implementation

For a Proof of Concept (PoC), several ecosystem elements were implemented to gain insights and knowledge about their actual behaviour and effectiveness. For an implementation, it was focused on an event-driven application as a subset of the entire cloud application space, due to their alignment with stateless cloud function processing. Furthermore, the use of decentralised open messaging infrastructure was explored. Figure 2 shows the proposed ecosystem to achieve a **Function Hub** that could help to create a decentralised environments for the serverless ecosystem.

5.1 Function marketplaces

The initial prototype its based on the core idea that defines what and how elements are trade within it. In this case, an ecosystem that allows free exchange of functions between users and generates the required environment for a serverless market to proliferate.
For this purpose a **Function Hub**[3] was designed considering the following characteristics:

- **Decentralisation**, by using an Extensible Messaging and Presence Protocol (XMPP) for communications, without centralised storage.
- **Abstraction**, by using *Snafu*[4] which allows managing cloud functions across provider convention boundaries.

Additionally, the complete **Function Hub** ecosystem is composed by four main components:

- **Users**: Users that access the **Function Hub Interface**[5] to share and get functions according to its needs.
- **Function Hub Interface**: It is a website based on AngularJS which runs on a NginX server. Users can search a function by name or specifying other attributes, having the testing option on the same platform and then download functions according to its requirements. For the interconnection between HTTP and XMPP protocols, a message broker named *MW 2* was developed.
- **Decentralisation**: In this work, decentralisation is achieved through repositories to store functions and serving as an active server for users. A software named *Snafu* (Swiss Army Knife of Serveless Computing), a FaaS host process, was used for this task. *Snafu* was chosen because it fulfils two main characteristics, as an execution environment for functions (active mode) and as a function repository (passive mode). As **Function Hub** does not pretend to use an unique centralised storage, it communicate with other repositories through XMPP. For this reason, each provider has a XMPP Client account associated to its message broker *MW 1* and to its associated XMPP Server for sending data required by users. The use of message brokers allows interconnection between HTTP and XMPP protocols, because both *Function Hub* and *Snafu* use HTTP as a main protocol connector. The implementation of the message brokers (*MW 1* and *MW 2*) was made through flask and nbxmpp libraries in Python.
- **Abstraction**: The abstraction sector is composed by a basic converter that add all the basic wrappers needed to deploy the function, according to most of the main cloud providers and uses the serverless framework (Collins 2015) as a deployment tool.

5.2 Function converters

As an early prototype[6], a converter of Python functions was developed to add wrappers for different modules that the file could have. It works according to these steps:

- It consumes and checks if the function has some run-time code to avoid security issues when importing.
- After checking, depending of the choice of the converter user, it dismantles different modules of the file and adds wrappers with the function for each

[3]https://github.com/serviceprototypinglab/functionshub
[4]https://github.com/serviceprototypinglab/snafu
[5]https://github.com/YessicaBogado/FunHub
[6]https://github.com/walter-bd/faas-converter

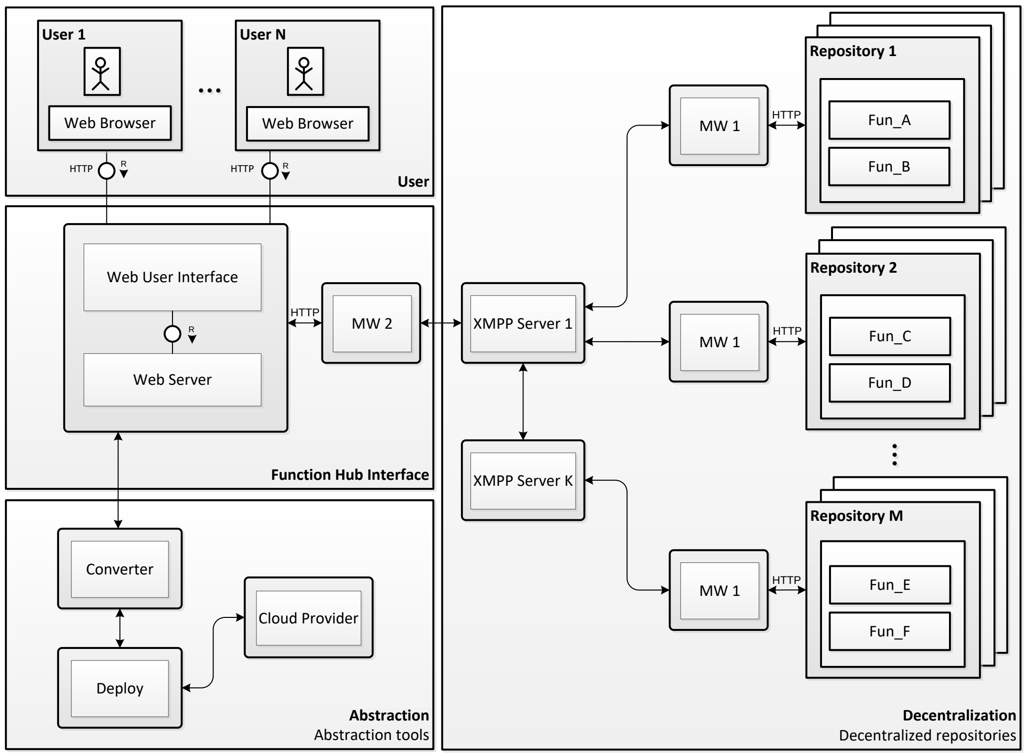

Figure 2: Proposed ecosystem implementation flowchart.

one on different files or it add the wrapper of one of this module at the end of a copy of the file.

- It has option to add wrappers for providers like *AWS*, *OpenWhisk*, *Fission*, *OVH* and *Azure*.

- The user can choose wherever to add the wrappers on different files for each provider or to add it all together in one file but, it has to manually check for some collisions with other wrappers added.

This tool will allow users of the environment to easily create standard function to deploy in different cloud providers.

5.3 Function deployers

For deploying functions into the ecosystem, a new functionality was needed to be added. This will give users the option to upload their functions from their repositories to the **Function Hub** ecosystem; hence, for this purpose a *Snafu* server is used.
To deploy functions from the **Function Hub** to a private cloud provider is intended to use the serverless framework, because this have already options to facilitate the deployment on the most popular cloud providers. Furthermore, a composeless[7] service is being developed to deploy functions on serverless ecosystems or in a container, according to particular needs.

5.4 Function execution environments

The execution environment of the ecosystem is provided by the FaaS host process *Snafu*, and it's support different programming languages like C, Java, JavaScript and Python. *Snafu* is deployed on an Alpine-based docker image, where it has the following run-time environments Python 3.5, Python 2.7, openjdk8, nodejs and for compiling purpose *gcc* and $g++$ for C programs.

5.5 Type of users

In the proposed ecosystem, the interaction is based in four kind of users:

- **Regular users**: They use the ecosystem to look, download, test or upload functions to allowed

[7]https://github.com/serviceprototypinglab/composeless

repositories. To make this possible, a web interface is available, showing the user all the repositories and its respective functions.

- **Exclusive repository user**: For companies that want to create exclusive functions to their products associated with their own infrastructure. **Function Hub** would provide a place where they can share the use of this function, allowing them to decide what to share and what to sell.

- **Passive repository user**: In this case, collaborative users or companies could create a repository so that other users could use it to store its functions.

- **Active repository user**: It would work as a Passive Repository, but also will allow regular users to test the functions or execute it there.

6 Conclusions and future directions

The rapid growth of serverless computing creates a need for an ecosystem in order to bring users necessary tools for a fast and cheap deployment of their software. On that point, Amazon took the first steps, providing to their users a serverless repository with basic functionalities. Also, new open source serverless initiatives, like OpenWhisk and Fission, gives developers freedom to create new tools.
However, and as it was showed in this paper, it is also needed properties like decentralisation and abstraction that allows them to create applications that interact with a diverse cloud ecosystem and take advantage of this diversity according to their needs, without worrying when individual market participant vanish. Pointing on that direction, this paper presented a vision for how ecosystems with a decentralised Function Hub may give to developers a way to share their functions, thinking not only on a specific cloud provider but also considering application itself instead. Further discussion, research and develop is required with the objective of give such ecosystem essential properties like sustainability, scalability and reliability associated with a wider adoption.

References

Baldini, I., Castro, P., Chang, K., Cheng, P., Fink, S., Ishakian, V., Mitchell, N., Muthusamy, V., Rabbah, R., Slominski, A. et al. (2017), Serverless computing: Current trends and open problems, *in* 'Research Advances in Cloud Computing', Springer, pp. 1–20.
URL: *https://doi.org/10.1007/978-981-10-5026-8_1*

Collins, A. (2015), 'Serverless framework', GitHub.
URL: *https://github.com/serverless/serverless*

DeLegge, A. & Wangler, H. (2017), 'Is this the end for facebook? A mathematical analysis', *Applied Mathematics and Computation* **305**, 364–380.
URL: *https://doi.org/10.1016/j.amc.2017.02.014*

Petsas, T., Papadogiannakis, A., Polychronakis, M., Markatos, E. P. & Karagiannis, T. (2013), Rise of the planet of the apps: a systematic study of the mobile app ecosystem, *in* 'Proceedings of the 2013 Internet Measurement Conference, IMC 2013, Barcelona, Spain, October 23-25, 2013', pp. 277–290.
URL: *http://doi.acm.org/10.1145/2504730.2504749*

Saini, A. (2016), An extension to UDDI for the discovery of user driven web services, *in* 'Distributed Computing and Internet Technology - 12th International Conference, ICDCIT 2016, Bhubaneswar, India, January 15-18, 2016, Proceedings', pp. 92–96.
URL: *https://doi.org/10.1007/978-3-319-28034-9_11*

Shu, R., Gu, X. & Enck, W. (2017), A study of security vulnerabilities on docker hub, *in* 'Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY 2017, Scottsdale, AZ, USA, March 22-24, 2017', pp. 269–280.
URL: *http://doi.acm.org/10.1145/3029806.3029832*

Stigler, M. & Stigler, M. (2018), *Beginning Serverless Computing*, Springer.
URL: *https://doi.org/10.1007/978-1-4842-3084-8_1*

Yu, L. (2011), 'Coevolution of information ecosystems: a study of the statistical relations among the growth rates of hardware, system software, and application software', *ACM SIGSOFT Software Engineering Notes* **36**(6), 1–5.
URL: *http://doi.acm.org/10.1145/2047414.2047435*