Practical Tooling for Serverless Computing

Josef Spillner
Zurich University of Applied Sciences
School of Engineering, Service Prototyping Lab (blog.zhaw.ch/icclab/)
Winterthur, Switzerland
josef.spillner@zhaw.ch

ABSTRACT

Cloud applications are increasingly built from a mixture of runtime technologies. Hosted functions and service-oriented web hooks are among the most recent ones which are natively supported by cloud platforms. They are collectively referred to as serverless computing by application engineers due to the transparent on-demand instance activation and microbilling without the need to provision infrastructure explicitly. This half-day tutorial explains the use cases for serverless computing and the drivers and existing software solutions behind the programming and deployment model also known as Function-as-a-Service in the overall cloud computing stack. Furthermore, it presents practical open source tools for deriving functions from legacy code and for the management and execution of functions in private and public clouds.

CCS CONCEPTS

• **Networks** → **Cloud computing**; • **Software and its engineering** → *Automatic programming*; *Software as a service orchestration system*;

KEYWORDS

serverless; microservices; FaaS; hosted functions; tutorial

ACM Reference format:
Josef Spillner. 2017. Practical Tooling for Serverless Computing. In *Proceedings of UCC'17: 10th International Conference on Utility and Cloud Computing, Austin, Texas, USA, December 5–8, 2017 (UCC'17),* 2 pages.
https://doi.org/10.1145/3147213.3149452

1 EMERGENCE OF SERVERLESS COMPUTING

Across the cloud computing industry, there is a large trend to move up the stack. Infrastructure management has become a commodity while in parallel, developer-facing platforms have become numerous and widespread. The ability to deploy applications directly from source without intermediate packaging as virtual machine images or container images increases the rapid service creation and reduces issues related to the configuration of the development environment. Apart from pushing monolithic code trees, the fine-grained deployment of individual functions in Function-as-a-Service (FaaS) environments is becoming more common, in particular for cloud automation, cloud-device coupling, and infrequently used web services [5]. FaaS brings utility computing closer to reality due to real on-demand enablement and microbilling of service invocations without having to pay for idle periods. It also drives the decomposition of applications into microservices, leading to more cloud-native applications. Businesses around the world have thus started to embrace FaaS through the paradigm of seemingly serverless computing.

Academics have started to describe and analyse FaaS through surveys and experiments [1, 9, 10] as well as economic analysis [3, 8, 15] and dedicated workshops (for instance WoSC'17 [4]). Still, not much is known about which tools to use for producing, deploying and running functions. This tutorial consolidates and enhances knowledge about such tools in order to achieve two goals: first, foster more experimental research on the topic; and second, enrich rapid transfer into education with hands-on competences. Fig. 1 puts the relevant tool categories into perspective concerning stacks and roles in the larger cloud computing ecosystem with its XaaS service models. FaaS refines the PaaS layer with a higher degree of abstraction to convey the desired serverless experience to application engineers. In the following paragraphs, the tool categories will be explained along with prototypical research implementations.

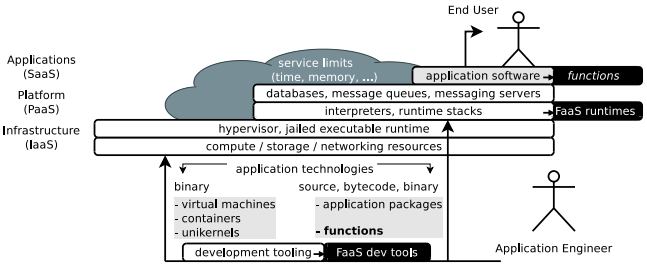

Figure 1: Function development and execution tools in the cloud computing ecosystem.

2 SERVERLESS FOUNDATIONS

From a software engineering perspective, functions in the cloud resemble functions or methods in modelling and programming languages. Functions are engineered in almost arbitrary languages following conventions for function parameters, return values and stateless execution semantics. The engineering is an activity of programming or meta-programming through transformation or transpilation of legacy code, followed by an implementation-dependent build process. Parameters are passed explicitly per request or implicitly through environment variables and other read-only data

UCC'17, December 5–8, 2017, Austin, Texas, USA

ACM ISBN 978-1-4503-5149-2/17/12.
https://doi.org/10.1145/3147213.3149452

spaces. Function implementations are thus tangible microservices in the form of source code, byte code or executables, including containers.

Based on this definition, the foundations then encompass all concepts, terminologies, emerging reference architectures and development and usage patterns for the lifecycle of services implemented as functions. Recent research sheds some light on the challenges associated with this service model. For instance, in contrast to virtual machines it is not possible to provision resource-differentiated function instance types in commercial services [14].

3 TOOLING FOR SERVERLESS CONTEXTS

Two categories of tools can be distinguished: function development, including deployment, testing and debugging, and execution.

3.1 Development Tools

Development tools determine how well and efficiently functions can be created and offered as microservices on the market. Often, the development process requires provider-specific SDKs and APIs, although higher-level programming libraries and cross-provider abstraction frameworks like PyWren and the Serverless Framework exist [2, 7]. A recent trend is the semi-automated decomposition of legacy code into functions through a process called FaaSification. In practice, not all functions can or should be exported this way. Through annotations on the programming level, developers can selectively specify and configure the transformations. The configuration includes the location of the target environment and the execution characteristics. Tool implementations exist for Java and Python, for instance Podilizer, Termite and Lambada [12, 13]. In case functions are to be called from other functions, the function locality needs to be configured as well. The mechanics of such calls involve again provider-specific SDKs, e.g. Boto for Python, and appropriate permissions.

3.2 Execution Tools

Functions are executed in private or public cloud services. Commercial public services mostly depend on proprietary implementations, including Azure Functions, AWS Lambda and Google Cloud Functions. Open source runtime environments such as OpenWhisk, Fission, OpenFaaS or IronFunctions enable private deployments, but are also increasingly used in public clouds such as Bluemix. Furthermore, there are attempts to replicate the runtime characteristics of the proprietary services in open source tools such as Docker-LambCI. Despite being open source, most of the runtimes require an effort-intensive setup and operation. Academic approaches include OpenLambda [6] and Snafu [11], the Swiss Army Knife of Serverless Computing which integrates on demand with other runtimes and thus allows for controlled experiments.

Apart from the runtime, the execution depends on how a function is triggered (e.g. through a network protocol or a timer) and how requests are routed to the runtime. Often, API gateways are instrumental in achieving the necessary degree of automation. Given the stateless nature of functions, stateful services such as blob stores or databases need to be coupled which may introduce additional delays and cost.

3.3 Limitations and Challenges

While many practical tools will become available in the next year or two, resolving many of the initial issues associated with FaaS, the research potential remains high. Researchers need to identify possibilities as well as documented and undocumented limitations through analytical work given specialised tools. Within some years, hybrid orchestrations of microservices with containers and functions are going to become common, yet there are no languages available to express the compositions. Container orchestration (e.g. Docker Compose) and function orchestration (e.g. Step Functions) are still separate also in terms of tools, as well as in terms of ecosystems (e.g. Docker Hub). Another challenge are Deep FaaSification processes which streamline the conversion of legacy code into functions by incorporating a semantic code analysis. Improved debugging, profiling and autotuning of functions is also required. First commercial tools such as X-Ray and Stackdriver exist to introspect function execution, but no research work is known on such tools and their effectiveness remains unknown.

REFERENCES

[1] Ioana Baldini, Paul C. Castro, Kerry Chang, Perry Cheng, Stephen J. Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric M. Rabbah, Aleksander Slominski, and Philippe Suter. 2017. Serverless Computing: Current Trends and Open Problems. *CoRR* abs/1706.03178 (2017). http://arxiv.org/abs/1706.03178

[2] Austen Collins, Eslam Hefnawy, and Philipp Müns. 2017. Serverless - The Serverless Application Framework. online: serverless.com. (2017).

[3] Adam Eivy. 2017. Be Wary of the Economics of "Serverless" Cloud Computing. *IEEE Cloud Computing* 4, 2 (2017), 6–12. https://doi.org/10.1109/MCC.2017.32

[4] Geoffrey C. Fox, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. 2017. Status of Serverless Computing and Function-as-a-Service(FaaS) in Industry and Research. *CoRR* abs/1708.08028 (2017). http://arxiv.org/abs/1708.08028

[5] Alex Glikson, Stefan Nastic, and Schahram Dustdar. 2017. Deviceless edge computing: extending serverless computing to the edge of the network. In *Proceedings of the 10th ACM International Systems and Storage Conference, SYSTOR 2017, Haifa, Israel, May 22-24, 2017*. 28:1. https://doi.org/10.1145/3078468.3078497

[6] Scott Hendrickson, Stephen Sturdevant, Tyler Harter, Venkateshwaran Venkataramani, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. 2016. Serverless Computation with OpenLambda. In *8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. Denver, Colorado, USA.

[7] Eric Jonas, Shivaram Venkataraman, Ion Stoica, and Benjamin Recht. 2017. Occupy the Cloud: Distributed Computing for the 99%. *CoRR* abs/1702.04024 (2017). http://arxiv.org/abs/1702.04024

[8] Philipp Leitner, Jürgen Cito, and Emanuel Stöckli. 2016. Modelling and managing deployment costs of microservice-based cloud applications. In *Proceedings of the 9th International Conference on Utility and Cloud Computing, UCC 2016, Shanghai, China, December 6-9, 2016*. 165–174. https://doi.org/10.1145/2996890.2996901

[9] Maciej Malawski. 2016. Towards Serverless Execution of Scientific Workflows - HyperFlow Case Study. In *Proceedings of the 11th Workshop on Workflows in Support of Large-Scale Science co-located with The International Conference for High Performance Computing, Networking, Storage and Analysis (SC 2016), Salt Lake City, Utah, USA, November 14, 2016*. 25–33. http://ceur-ws.org/Vol-1800/paper4.pdf

[10] Josef Spillner. 2017. Exploiting the Cloud Control Plane for Fun and Profit. arχiv:1701.05945. (January 2017).

[11] Josef Spillner. 2017. Snafu: Function-as-a-Service (FaaS) Runtime Design and Implementation. arχiv:1703.07562. (March 2017).

[12] Josef Spillner. 2017. Transformation of Python Applications into Function-as-a-Service Deployments. arχiv:1705.08169. (May 2017).

[13] Josef Spillner and Serhii Dorodko. 2017. Java Code Analysis and Transformation into AWS Lambda Functions. arχiv:1702.05510. (February 2017).

[14] Josef Spillner, Cristian Mateos, and David A. Monge. 2017. FaaSter, Better, Cheaper: The Prospect of Serverless Scientific Computing and HPC. In *4th Latin American Conference on High Performance Computing (CARLA)*. To appear.

[15] Mario Villamizar, Oscar Garces, Lina Ochoa, Harold Castro, Lorena Salamanca, Mauricio Verano, Rubby Casallas, Santiago Gil, Carlos Valencia, Angee Zambrano, and Mery Lang. 2017. Cost comparison of running web applications in the cloud using monolithic, microservice, and AWS Lambda architectures. *Service Oriented Computing and Applications* 11, 2 (2017), 233–247. https://doi.org/10.1007/s11761-017-0208-y