CHALMERS
UNIVERSITY OF TECHNOLOGY

Transpiling Applications into Optimized Serverless Orchestrations

Citation for the original published paper (version of record):
Scheuner, J., Leitner, P. (2019). Transpiling Applications into Optimized Serverless Orchestrations. Proceedings - 2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems, FAS*W 2019, June 2019: 72-73. http://dx.doi.org/10.1109/FAS-W.2019.00031

Transpiling Applications into Optimized Serverless Orchestrations

Joel Scheuner
Software Engineering Division
Chalmers | University of Gothenburg
Gothenburg, Sweden
scheuner@chalmers.se

Philipp Leitner
Software Engineering Division
Chalmers | University of Gothenburg
Gothenburg, Sweden
philipp.leitner@chalmers.se

***Abstract*—The serverless computing paradigm promises increased development productivity by abstracting the underlying hardware infrastructure and software runtime when building distributed cloud applications. However, composing a serverless application consisting of many tiny functions is still a cumbersome and inflexible process due to the lack of a unified source code view and strong coupling to non-standardized function-level interfaces for code and configuration. In our vision, developers can focus on writing readable source code in a logical structure, which then gets transformed into an optimized multi-function serverless orchestration. Our idea involves transpilation (i.e., source-to-source transformation) based on an optimization model (e.g., cost optimization) by dynamically deciding which set of methods will be grouped into individual deployment units. A successful implementation of our vision would enable a broader range of serverless applications and allow for dynamic deployment optimization based on monitoring runtime metrics. Further, we would expect increased developer productivity by using more familiar abstractions and facilitating clean coding practices and code reuse.**

***Index Terms*—serverless, FaaS, transpiling, orchestration**

I. Introduction

The emerging cloud computing paradigm called *serverless computing* abstracts operational concerns such as autoscaling and exposes an event-driven interface for building scalable distributed applications. Its *Function-as-a-Service (FaaS)* model refers to event-triggered server logic running in ephemeral (compute) containers, which are fully managed by a third party, such as AWS Lambda[1]. FaaS pushes the minimal deployment unit towards small chunks of code encapsulated within individual functions and the minimal billing unit towards tens of milliseconds. However, building complex serverless applications is still cumbersome as mentioned in literature [1], supported by an empirical study [2], and illustrated by calls of practitioners to raise the level of abstraction level (c.f., "I don't care about Lambda [...] but if I can move one layer up where I'm just writing business logic and the code gets split up appropriately, that's real magic.")[2]. Contemporary FaaS approaches mandate developers into non-standardized Application Programming Interfaces (APIs) and proprietary deployment configurations. The challenge in making FaaS more accessible is addressed by countless startups[3], frameworks, libraries, and tools. These efforts are driven by a strong focus on facilitating the definition, deployment, and orchestration of individual FaaS functions. In this paper, we outline a vision where developers are liberated from conforming to particular forms of deployment units, such as individual functions, and FaaS applications are automatically and dynamically transpiled into a set of individually deployed functions.

II. Vision

There exist different alternatives to combine individual functions into a FaaS application. Examples for declarative approaches are AWS StepFunctions[4], which represents workflows as JSON state machine specifications, and Fission Workflows[5], which represents workflows in a YAML-based DSL. An example for an imperative approach is Azure Durable Functions[6], which represents workflows in code as orchestrator functions. We think that Azure Durable Functions is a promising step towards writing FaaS applications in a more natural and flexible way but envision an even more integrated way of defining functional code and orchestration logic.

Figure 1 illustrates the main idea how source-to-source transformation, also called transpilation, can enrich serverless applications while exposing familiar language abstractions and allowing for dynamic deployment optimization. Transpilation can automatically generate boilerplate code and apply interface adjustments for provider API compliance. A platform-independent application could be transpiled into FaaS application variants for multiple providers. Application code can be expressed using familiar language concepts for asynchronous programming in a local environment (e.g., async/await following the ECMAScript 2018 specification) and then transpiled into FaaS-aware implementation as demonstrated in recent work [3]. Furthermore, transpilation allows for dynamically adjusting deployment decisions based on static source code information (e.g., data flow) or dynamically collected monitoring data (e.g., resource footprint). For example, data flow analysis

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

[2]https://read.acloud.guru/serverless-is-eating-the-stack-and-people-are-freaking-out-and-they-should-be-431a9e0db482

[3]https://github.com/anaibol/awesome-serverless

[4]https://aws.amazon.com/step-functions/

[5]https://github.com/fission/fission-workflows

[6]https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-overview

 The definitive Version of Record was published in *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS*W), June 16, 2019, Umeå, Sweden*, https://doi.org/10.1109/FAS-W.2019.00031

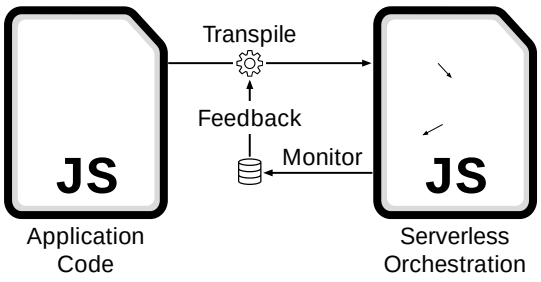

Figure 1. Application to Orchestration Transpilation

could group methods into deployment units to minimize data marshalling between individual FaaS functions, or resource usage (e.g., memory consumption or CPU utilization) could identify optimal function configurations and group methods with similar resource demands together. Thus, this approach allows for informed and dynamic deployment topology decisions whereas the question how large a FaaS function should be is nowadays determined manually by the "gutfeeling" of the developers or guided by FaaS platform restrictions.

Current technical trends and several application scenarios motivate our envisioned approach. The trend towards disaggregated data centers (e.g., even suggested for memory [4]) and increasingly specialized hardware beyond Graphics Processing Units (GPUs) (e.g., Tensor Processing Unit (TPU), or Field-Programmable Gate Array (FPGA)) open a potential for deployment optimization where parts of an applications run on highly optimized hardware. A hypothetical example application could be orchestrated by affordable CPU functions, offload image rendering to GPU hardware, machine learning training to TPU hardware, and encryption to FPGA hardware. Furthermore, code that leverages specialized libraries (e.g., NumPy[7] for Python > 10 MB) or runtimes (e.g., Puppeteer[8] headless Chrome API > 80 MB) with large memory footprints can be isolated without imposing high memory requirements on other parts of an application. Such other parts could then be grouped and multiple code-level functions can be compiled into a single FaaS function for improved latency. These sample scenarios illustrate how this approach can lead to cost savings and better performance.

III. Current Work and Challenges

We are currently working on a prototype to demonstrate the main idea of optimized dynamic deployments. Our prototype targets Javascript, as most commonly used language in FaaS [2], and supports the open source serverless cloud platform Apache OpenWhisk[9]. Transpilation, a widely accepted concept in the Javascript community (e.g., Babel[10]), is implemented as a series of Abstract Syntax Tree (AST) transformations using jscodeshift[11]. For deploying FaaS compositions, we build upon the incubating Apache OpenWhisk Composer[12] but provide a more natural native Javascript code-oriented way of expressing FaaS applications. During transpilation, we dynamically decide whether to fuse source code-level functions together or deploy them as separate FaaS functions coordinated by OpenWhisk conductor actions.

Several challenges may affect the applicability of the envisioned approach. Transpiling applications into orchestrations favors applications written in the same programming language and therefore violates the blackbox constraint of the Serverless Trilemma [5]. However, we think that for many practical reasons (e.g., maintainability, developer skills) it is a fair assumption to promote a main language. Polyglot applications could still be integrated non-intrusively by transpiling remote APIs into library objects [3]. A general challenge when transpiling a program oriented towards local execution semantics into a distributed application are disparities in execution semantics. One major aspect is the handling of side effects and shared state. While we currently discourage unintentional use of side effects and marshall shared state up to certain limits, we could envison compiler-alike warnings to alleviate such issues.

IV. Conclusion and Future Research

We presented a vision where developers are liberated from conforming to particular forms of deployment units, such as individual functions, and FaaS applications are automatically and dynamically transpiled into a set of individually deployed functions. We envision that such an approach would enable a broader range of serverless applications, lead to more flexible cost-performance trade-off decisions, and increase developer productivity by providing a unified source code view. In our future work, we plan to extend our transpilation prototype by integrating and evaluating dynamic deployment options.

Acknowledgment

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, and the Swedish Research Council VR under grant number 2018-04127 (Developer-Targeted Performance Engineering for Immersed Release and Software Engineers).

References

[1] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht, "Occupy the cloud: distributed computing for the 99%," in *Proceedings of the 2017 Symposium on Cloud Computing*, 2017, pp. 445–451.

[2] P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of function-as-a-service software development in industrial practice," *Journal Systems and Software*, vol. 149, pp. 340 – 359, 2019.

[3] K. Kimura, A. Sekiguchi, S. Choudhary, and T. Uehara, "A javascript transpiler for escaping from complicated usage of cloud services and apis," in *25th Asia-Pacific Software Engineering Conf. (APSEC)*, 2018.

[4] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker, "Network requirements for resource disaggregation," in *12th USENIX Symposium OSDI*, 2016, pp. 249–264.

[5] I. Baldini, P. Cheng, S. J. Fink, N. Mitchell, V. Muthusamy, R. Rabbah, P. Suter, and O. Tardieu, "The serverless trilemma: Function composition for serverless computing," in *Onward!*, 2017, pp. 89–103.

[7] https://www.numpy.org/

[8] https://github.com/GoogleChrome/puppeteer

[9] https://openwhisk.apache.org/

[10] https://github.com/babel/babel

[11] https://github.com/facebook/jscodeshift

[12] https://github.com/apache/incubator-openwhisk-composer