Static Call Graph Construction in AWS Lambda Serverless Applications

Matthew Obetz
Rensselaer Polytechnic Institute
obetzm@rpi.edu

Ana Milanova
Rensselaer Polytechnic Institute
milanova@cs.rpi.edu

Stacy Patterson
Rensselaer Polytechnic Institute
sep@cs.rpi.edu

Abstract

This paper discusses challenges to statically constructing call graphs for applications that execute in a serverless cloud. We consider the specific problem of capturing program flows where state passes between lambda functions through event triggers associated with external data stores. To implement our discovered techniques we present _____, a tool for statically constructing call graphs on AWS Lambda applications written in Javascript.

Keywords serverless, static analysis, AWS Lambda

ACM Reference Format:
Matthew Obetz, Ana Milanova, and Stacy Patterson. 2018. Static Call Graph Construction in AWS Lambda Serverless Applications. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages.

1 Introduction

Serverless computing represents a new paradigm for web development in which application logic is divided among lambda functions that can be dynamically distributed across containers in a host-managed cloud. This powerful abstraction delivers an opportunity for software engineers to design applications without consideration for the context in which they will be executed. Instead, programmers write short functions against high-level APIs that expose mechanisms for long term storage and interaction with the serverless environment. These functions, often packaged as small libraries of Python or Javascript, are then assigned events which specify the conditions under which they will execute. Commonly, these events will specify the address of a web gateway or database which is also managed by the host and will accept a subscription that triggers when the associated resource receives new data.

This model, while offering many advantages for applications which require elastic scalability, creates a gap in tooling for debugging and program analysis. Because containers are reused across different applications to maximize performance, the current generation of serverless platforms are inherently stateless, requiring data that is used in more than one lambda function to be passed directly or persisted and retrieved between each lambda. Existing commercial serverless platforms such as AWS Lambda place strict bounds on the running time and available memory for each individual lambda, encouraging developers to create flows that chain several lambdas together to complete a single task [1]. As each stage of this pipeline is a separate context that may even execute on a different machine, stack information that would normally serve a primary mechanism for a developer to identify defective paths does not exist. Likewise, classic methods of static analysis will fail to associate writes to a data store with events that trigger off of that data store, losing critical information about the flow of data between parts of an application.

However, such information remains an invaluable tool for developers [5]. This has prompted platform providers to release products such as Amazon X-Ray[1], which seek to fill this gap by collecting runtime traces on Lambda applications that have already been deployed, then aggregating these traces to generate visualizations that detail runtime performance and graph flows from lambdas to supporting backend services. Though X-Ray and similar tools provide some insight into the structure of Lambda applications, they crucially lack the ability to capture events triggered by these backend services and map a complete path that data may take from initial input.

In response, a tracing-based solution that augments X-Ray with implicit identifiers that are injected into inputs to allow tracing across event triggers has been explored in GammaRay [3]. This work was further generalized in Lowgo to support serverless clouds outside the AWS ecosystem [2]. While these solutions offer a more complete view of causal dependencies in a serverless application, the dynamic nature of the analysis they perform means that infrequently traversed paths may be missed in the service call graph they construct. Additionally, both GammaRay and Lowgo colocate their instrumentation directly in the cloud. This makes obtaining call graph information impractical during the implementation of an application, when this data is most useful to developers who wish to understand the scope of potential changes to an individual lambda. Colocation of runtime instrumentation also has an adverse effect on application performance, introducing as much as 43% overhead in the case of GammaRay [3], which may be unacceptable in performance critical contexts or at the massive scales serverless is designed to accommodate.

Conference'17, July 2017, Washington, DC, USA
2018.

[1] https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html

These drawbacks may be mitigated by using a fully static analysis. However, attempts to implement such an analysis are nontrivial, as serverless applications are highly asynchronous given their event-driven design, often leading to numerous dissociated entrypoints. Extensions to static call graph construction to support event listeners for general NodeJS programs have been proposed previously in Radar [4], but these techniques are insufficiently well-defined for the declarative configuration of event triggers common to serverless platforms.

In developing _____, we implement a model for static call graph construction more suited to the needs of serverless applications. Specifically, we contribute static semantics for generating a service call graph that incorporates flow to and from common backend services used by Javascript applications on the Lambda platform without the use of program traces. We thus extend traditional call graph analysis by providing a means for multiple programs that may even be running in separate execution environments to be integrated into a single extended call graph that provides information about the limited interfaces between which the constituent programs may share state.

2 Motivating Examples

To quantify the set of features commonly used in serverless applications, we analyzed publicly available lambda programs submitted by real users to the AWS Serverless Application Repository[2]. We found

References

[1] Ioana Baldini, Paul Castro, Kerry Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, et al. 2017. Serverless computing: Current trends and open problems. In *Research Advances in Cloud Computing*. Springer, 1–20.

[2] Wei-Tsung Lin, Chandra Krintz, and Rich Wolski. 2018. Tracing Function Dependencies Across Clouds. In *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE, 253–260.

[3] Wei-Tsung Lin, Chandra Krintz, Rich Wolski, Michael Zhang, Xiaogang Cai, Tongjun Li, and Weijin Xu. 2018. Tracking Causal Order in AWS Lambda Applications. In *Cloud Engineering (IC2E), 2018 IEEE International Conference on*. IEEE, 50–60.

[4] Magnus Madsen, Frank Tip, and Ondrej Lhoták. 2015. Static analysis of event-driven Node.js JavaScript applications. In *OOPSLA*.

[5] Mengting Yan, Paul Castro, Perry Cheng, and Vatche Ishakian. 2016. Building a chatbot with serverless computing. In *Proceedings of the 1st International Workshop on Mashups of Things and APIs*. ACM, 5.

[2] https://serverlessrepo.aws.amazon.com/applications