LogLess:
A Logging Paradigm for Serverless Applications

Kushagra Soni
Masters of Computer Science
University Of Illinois
Urbana Champaign, Illinois, USA
soni14@illinois.edu

Abhishek Shinde
Masters of Computer Science
University Of Illinois
Urbana Champaign, Illinois, USA
ashinde2@illinois.edu

Simranjit Bhamra
Masters of Computer Science
University Of Illinois
Urbana Champaign, Illinois, USA
sbhamra2@illinois.edu

Abstract

Logging is a prominent practice followed in software development. Without sufficient and appropriate logging, isolating bugs, debugging problems, and maintaining best security practices become increasingly difficult.

Current logging mechanisms in serverless platforms can sometimes become cumbersome for developers. In case of any failures or debugging, spotting the issues can become difficult due to inadequate logging. Additionally, the auto-generated system-wide logs, by cloud platforms, are not descriptive enough to help debug an application error. There are indeed some logging tools and utilities available in the market, which can be configured to an application's needs. However, they still need developers to write the logging statements apart from the code.

We propose LogLess, a new paradigm in logging, which can automate the logging mechanism. With this approach, the developers will be able to publish logs of their serverless applications without writing the log statements manually within the code. LogLess utilizes the "decorator" objects in a programming language. These decorators will be used to parse the function arguments and every underlying block or line of code within the function. LogLess will categorize every block/line of code based on the function it is performing such as variable assignment, API call request/response, connection to external databases, etc. Upon categorizing these lines of codes, LogLess will assign them to a corresponding "pattern group/sub-group". Once the pattern group has been assigned, it will fetch the corresponding values of all the underlying variables in each block/line of code and generate an appropriate log.

Keywords

Cloud Computing, Distributed Systems, Serverless Computing, Logging, Programming Languages, Decorators

I. Introduction

LogLess is a one-of-a-kind logging model and makes use of the fact that functions are "first-class objects" in programming languages, like Python, JavaScript, Java, Golang, etc. This allows functions to be passed as arguments to decorators which are functions themselves, to be used by LogLess. LogLess advances knowledge by offering a unique perspective and approach to how logging can be produced. In contrast to the traditional methodology of writing time-consuming logging statements in every block of code, this proof of concept offers a *Log-Less* decorator-based approach that will automate a range of logging methods for serverless applications. Ultimately, this idea can provide a quick, customizable, and developer-friendly logging model.

Currently, developers write individual logging statements in code blocks which could lead to missing key statements or heavily duplicate messages across the platform. LogLess would offer a contribution that simplifies logging by enabling developers to focus on their code. LogLess would enforce standardization on statement messages so that they can be reused.

Different development environments may require differing requirements for logging. LogLess will provide a novel contribution in that a logging mode can be selected from a list of modes. A logging mode will vary in terms of the number of log statements, what parameters get logged, and which log levels are supported. In addition, various modes can be configured namely – dev-mode, safe-mode, and prod-mode. Each mode will serve a different purpose in the software development lifecycle and won't need any additional changes to the actual code for debugging and logging. This configuration can provide better logging flexibility and design to applications.

Many serverless platform providers have predefined logging of their cloud services. For example, in AWS, a connection between a lambda function and DynamoDB is logged into CloudWatch. However, this logging is restricted to only AWS-provided services and is not inclusive of any external communications. Moreover, CloudWatch logs do not provide in-depth knowledge of variables and values used within a Lambda application or offer any customization opportunities. LogLess will offer more efficient, granular, and customized logging that will augment such in-built logging tools.

This project can provide a broader benefit to the serverless development community, which includes computer science students, development teams in the workplace, and open-source contributors. While we plan to experiment with LogLess using Python, it can be implemented in other programming languages, thus impacting a larger community.

II. Preliminary Plan

A logging model which will be built in python programming language but is applicable to any modern programming language. It will consist of below mentioned components.

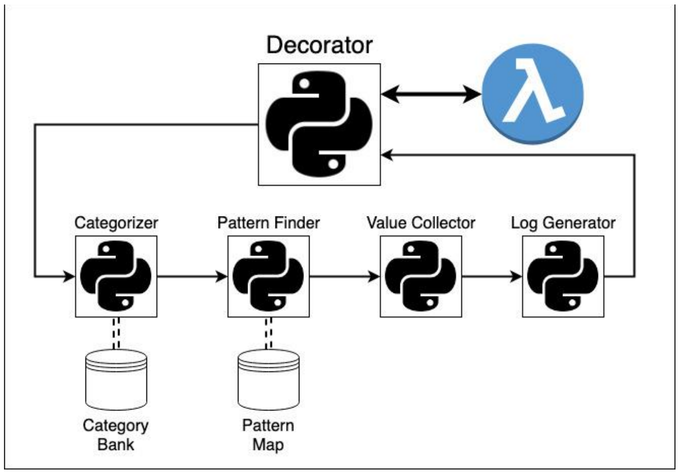

Figure 1: Workflow diagram of LogLess. It shows all the major components of the model

A. *Decorator*

This will be a parsing module, which will parse the functions within the serverless deployed application. It will traverse through every code-block within the function and pass them to the code categorizer. [7].

B. *Code Categorizer*

This module will take each code-block as an argument and categorize the code according to the predefined categories. These categories will be looked up from a code category bank. This code category will then be passed to a pattern finder.

C. *Code Category Bank*

This data bank will consist of various categories of code that might occur in a serverless architecture. This will categorize various programming functionalities like variable assignment, conditionals, loops, etc. Additionally, it will also contain categories defined based on extensive research on the available functionalities across various Serverless providers and their respective FaaS services.

D. *Pattern Finder*

This module will determine the pattern of the categorized code and based on the category and pattern map; it will determine the type of logging needed for that code category.

E. *Pattern-Category Map*

This module will map the code category with its relevant logging pattern. This pattern will be created based on extensive research and experimentation for various types of applications running over serverless platforms.

F. *Value Collector*

This module will store the values of all the variables occurring within a particular line/block of code.

G. *Log Generator*

It takes the pattern and the variables and their corresponding values as an input, will generate an appropriate log, which will be descriptive and informative for developers and code reviewers.

III. EXPERIMATATIONS

For experimentation, we plan to research a variety of applications currently prevalent on serverless platforms for their type of built-in functionalities. Based on that research we plan to create a data bank of all programming concepts and functions which are frequently used in serverless applications. We will also populate the categorization data bank serverless utility-specific methods, and functions.

Once we have a data suite ready, we will have a robust and extensive set of unit tests to verify the functionality of logging and what is expected for each category of code and functions. The team plans to perform black box testing through equivalence partitioning and boundary value analysis. In addition, the team will utilize a code coverage tool to carry out white box testing. In testing the team plans to use tools such as *Pytest* and *Coverage.py*.

IV. PRELIMINARY LITERATURE SURVEY

The project’s primary concern regarding software logging will be log instrumentation. This stage consists of including log statements in the application and maintaining them throughout the software development lifecycle. The logging approach, Logging Utility (LU) integration, and Logging Code (LC) composition are the steps of log instrumentation. The logging approach is the high-level workflow of how logging will be performed. The next component of LU integration involves deciding which tool to adopt (example: SLF4J for Java) and how to configure it. Finally, LC composition requires addressing where logging should occur, what should be logged, and how to log with high quality [1]. Documentation of these steps will be critical for this project.

Software logging with security practices is of paramount importance. To follow and implement best security practices when designing an application, vulnerabilities must be looked at. Insufficient logging is one example that can complicate bug identification and expose security vulnerabilities. The Open Web Application Security Project (OWASP) identified insufficient logging as one of the top ten security risks in 2017 [3,4]. In a serverless architecture, pinpointing bugs can become challenging if there are not enough log statements in the functions. In addition, malicious attackers may have had attempts to steal sensitive information, but it could go unnoticed if the pertinent logging is absent [2]. For this project, understanding this vulnerability will be important to design the logging module; brainstorming the LC composition design effectively can help prevent or mitigate it. If our team plans to test this model in a distributed system consisting of multiple serverless functions, adapting some aspects of a shared log may be helpful. A team from the University of Texas have proposed a serverless runtime, Boki, that outputs a shared log API. This approach consists of building an ordered log that can be accessed concurrently, and is known to achieve scalability, strong consistency, and fault tolerance [6].

[1] A Survey of Software Log Instrumentation https://dl-acm-org.proxy2.library.illinois.edu/doi/pdf/10.1145/3448976

[2] Vulnerabilities and Secure Coding for Serverless Applications on Cloud Computing https://link-springer-com.proxy2.library.illinois.edu/chapter/10.1007/978-3-031-05412-9_10

[3] Automated Modelling of Security Incidents to represent Logging Requirements in Software Systems https://dl.acm.org/doi/pdf/10.1145/3407023.3407081

[4] Serverless Computing Security: Protecting Application Logic https://ieeexplore.ieee.org/abstract/document/9180214

[5] Boki: Stateful Serverless Computing with Shared Logs https://dl.acm.org/doi/abs/10.1145/3477132.3483541

[6] Troubleshooting Serverless functions: a combined monitoring and debugging approach https://link.springer.com/article/10.1007/s00450-019-00398-6

[7] Advanced Python Techniques Decorators - https://www.researchgate.net/publication/272833859_Advanced_Python_Techniques_Decorators