Proceedings of the 11th International Conference on Applied Informatics
Eger, Hungary, January 29–31, 2020, published at http://ceur-ws.org

Stack Traces in Function as a Service Framework*

Ádám Révész, Norbert Pataki

ELTE Eötvös Loránd University, Budapest, Hungary
Faculty of Informatics, 3in Research Group, Martonvásár, Hungary
reveszadam@gmail.com
patakino@elte.hu

Abstract

Containerization has become an essential approach in modern software engineering. Docker is a widely-used solution for separate services (like database, backend, etc.) and run them as a standalone, isolated process instance on the same host kernel. Kubernetes is distributed approach over Docker, it supports multiple hosts for the deployment. Kubeless is a new approach that aims at the functionwise deployment, so every subprogram can be deployed, scaled, operated separately, thererfore a functional programming approach can be realized in a modern, highly distributed realm. In this paper, we present our framework that provides a programming framework over Kubeless. However, many programming development tools are not available in this realm. Stack trace is a well-known construct in programming languages to follow the function calls. We propose a mechanism to retrieve the stack trace for realizing program errors easily.

Keywords: cloud, Kubeless, serverless programming, stack trace

MSC: 68M14 Distributed systems

1. Introduction

Containerization has become an emerging approach in modern software engineering since it enables the shipping of the software products with all required dependencies in a platform-independent way [2]. Containerization eliminates the virtualization costs of not used OS services and the kernel itself per container. Moreover,

*The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications).

containerization supports isolation effectively since the containers are seem to be separate operating systems but they use a shared kernel [9].

The containers are lightweight and they enable the fast and simple deployment and configurations. However, a rather new abstraction is called Function as a Service (FaaS) and this approach eliminates further configuration and deployment cost. This approach provides the deployment of standalone function without launching any virtual machine or container.

We have developed an approach to enable functional programming paradigm over Kubeless. However, this realm does not support the traditional programming development tools for convenient work, such as debuggers. Stack trace is a well-known solution for realizing and detecting runtime problems [10]. Reconstruction of bugs and incorrect program usage highly based on stack traces, therefore convenient programming workflow requires it.

You can see a stack trace received by execution of Java program:

```
Exception in thread "main" java.lang.StackOverflowError
        at java.io.PrintStream.write(PrintStream.java:480)
        at sun.nio.cs.StreamEncoder.writeBytes(StreamEncoder.java:221)
        at sun.nio.cs.StreamEncoder.implFlushBuffer(StreamEncoder.java:291)
        at sun.nio.cs.StreamEncoder.flushBuffer(StreamEncoder.java:104)
        at java.io.OutputStreamWriter.flushBuffer(OutputStreamWriter.java:185)
        at java.io.PrintStream.write(PrintStream.java:527)
        at java.io.PrintStream.print(PrintStream.java:669)
        at java.io.PrintStream.println(PrintStream.java:806)
        at StackOverflowErrorExample.recursivePrint(StackOverflowErrorExample.java:4)
        at StackOverflowErrorExample.recursivePrint(StackOverflowErrorExample.java:9)
        at StackOverflowErrorExample.recursivePrint(StackOverflowErrorExample.java:9)
        at StackOverflowErrorExample.recursivePrint(StackOverflowErrorExample.java:9)
        ...
```

This stack trace illustrates the spreading of a `StackOverflowError` exception. The actual function call chain can be read with filenames and line numbers, so one can determine where the exception appeared first.

However, collecting the stack trace when functions are executed parallelly on many hosts is a difficult task. Unfortunately, the missing stack traces slow down the debugging process significantly.

We argue for a solution in our framework that retrieves stack trace in case of runtime because stack trace is also required when the executed functions are running on different hosts.

In this paper, we present the background of serverless programming in section 2. We introduce our Kubeless-based functional framework in section 3. We present the implementation background of stack traces in section 4. Finally, this paper concludes in section 5.

2. Serverless Programming

Function as a Service (FaaS) is a category of cloud computing services that provides a platform allowing programmers to develop, maintain, operate, scale and manage application functionalities without the complexity of building and maintaining the

infrastructure typically associated with developing and deploying an application. Building an application following this model is one way of achieving a "serverless" architecture [5].

Many cloud providers support serverless programming, for instance, Microsoft Azure Lambda, Google Cloud Functions, etc [7].

Serverless programming is a rather new approach, however, there are real-world applications, for instance, Coca-Cola, Santander Bank and Expedia take advantage of this new paradigm [4].

3. LambdaKube Architecture

Kubernetes is the most popular container orchestration system, Kubeless is a FaaS system over the Kubernetes. We have created a functional approach over Kubeless. The fundamental infrastructural elements are introduced along with the building blocks of composable Kubeless functions discussing challenges, decisions and implementation.

3.1. Kubernetes

Kubernetes is a container orchestration system which manages Docker containers over multiple Docker hosts [8]. This paper does not discuss Kubernetes in details, mentions only the essentials, used by other layers in the LambdaKube architecture.

As an orchestration system, it provides the following services:

- pods and deployments – containers with their meta properties (Docker image, environment variables for example), resource references (e.g. volumes and configmaps) and replication controlling properties
- services – name resolution and load balancing
- configmaps and secrets – storing and serving configurations and secrets
- volumes – provisioning claimable volumes through multiple different volume providers (local volume, cloud provided volumes, and other custom volumes)
- ingress – API gateway definitions for publicly exposed services
- custom resource definitions – supporting user-defined resource templates composed of Kubernetes-native resources

3.2. Kubeless Platform

Kubeless is a serverless function platform implementation. Kubeless is responsible to deploy and run code snippets – functions on Kubernetes in containers.

Provided services:

- code snippet store – by default Kubeless uses config maps for storing code

- runtime for running the code snippets – Docker images for initializing, (optionally) compiling and running the code snippets wrapped into language environment specific wrapper code(s) [1].
- function registration – creating Kubernetes service which ensures service discovery, so other services and functions can call/trigger the function over HTTP
- message triggers – integrates with optionally installed message queue provider (like Kafka, or in our case Nats) and ensures a specified functions trigger on every new message on a specified topic
- function controller – a controller managing function deployments, triggers and scale, an orchestrator over function nodes

3.3. LambdaKube Platform

LambdaKube, the project aims for a functional programming language, a compiler and a platform with runtime frameworks where the compile time generated functions can be deployed. The high-level architecture can be seen on Figure 1.

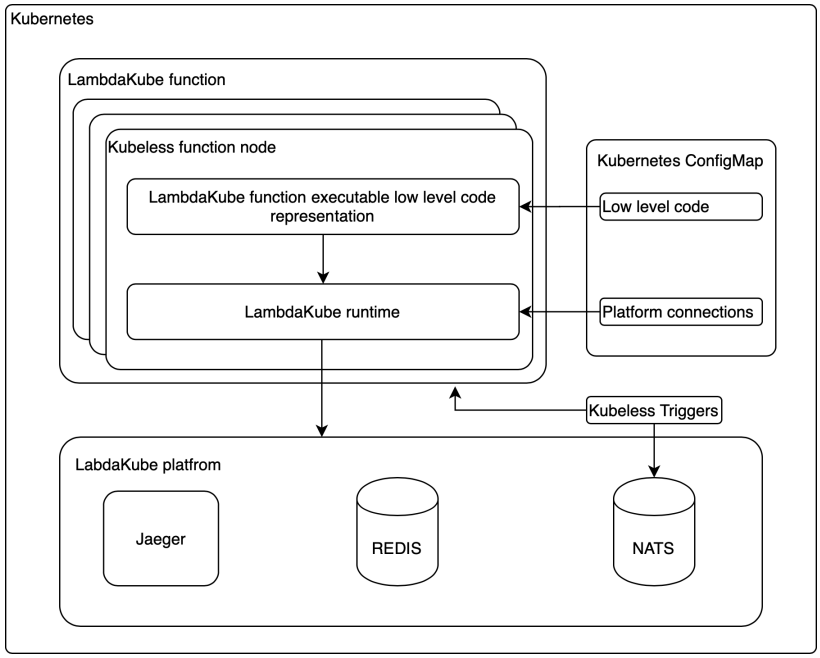

Figure 1: Architecture of LambdaKube

3.3.1. Communication

Since the programming language is functional and the FAAS realm is a distributed environment, the first requirement of the platform seems obvious: the composition. For composition asynchronous communication seemed to be fit, because of the convenient use of unidirectional data flow and the elimination of the complexity timeout introduces in synchronous communication.

For asynchronous communication between services the industry recommends message queues, such as Kafka. Our choice of message queue providers is Nats. Nats is a lightweight message queue implementation fit to development purposes of the platform at this stage and provides similar guarantees on messages (like ordering) like Kafka, so on later stages, when big throughput occurs, bigger scaling required, Nats can be replaced with Kafka with minor changes on the platform (even can be made interchangeable with messaging libraries and dependency injection).

3.3.2. State management

Obviously the programming language layer, as a functional programming language does not handle state. Also the promise of this project and FAAS architecture is the stateless function.

The current goal of the platform is implementing a unidirectional data flow with asynchronous communication and more importantly in an FAAS solution idle time, and active waiting should be avoided due to time based pricing on most providers and the fact of wasting computing resources.

Having the map construction for a case of introducing distributed (parallel) computations the main focus is on the function passed to map but when the function has been evaluated with all elements of the original context, something (lets call it packer service) should wrap it into the new context and send the result to the next stage of the pipeline. The state handling, and active waiting should be avoided, so a state is introduced in LambdaKube platform.

As of now the state storage service is a Redis, in-memory key-value pair database designed for clustered – distributed usage. With this solution the packer service is triggered on every new result from the function passed to map, stores it (with other meta variables) in Redis. If all results arrived, wraps them and sends the result in the new context to the next stage, then stops.

3.3.3. Configurations

Configurations for both Redis and Nats with their corresponding connection strings and secrets are handled by the platform, stored in Kubernetes configmap and secret resources.

3.4. LambdaKube Runtime

LambdaKube now uses Python, generates Python codes as snippets so the current runtime (planned to be interchangeable in the future) is an extended version of

Kubeless provided Python runtime.

LambdaKube runtime layer contains three main elements:

- General dependencies for interacting with the platform services (Redis and Nats).
- Wrapper framework which handles the incoming request from Kubeless controller on trigger, sets up connections to platform services, invokes the function call provided by the mounted code snippet, pushes the result to the next stage.
- a Docker container image packs all elements listed above,

Other elements like init container image and tasks are not discussed in this paper.

On function deployment, the references for configmap and secret entries are injected into the function instance environment.

3.5. LambdaKube low level representation

LambdaKube compilation generates Python codes for the actual functions to be wrapped into the runtime on deployment.

Each function node gets its code generate into a separate service-name.py file. The function names, message topic names and their relations are generated into a descriptor file (picked up by Helm on deployment time [3]).

4. Stack Traces in LambdaKube

4.1. Tracing options

4.1.1. Message queue

With the current LambdaKube platform, a message queue is given. It could seem easy to let the temptation win, and give the developers access to the queues so all the calls, parameters and results are visible for them but this solution has multiple issues:

- Message queue is part of the LambdaKube platform. The developer (except platform developer) should not be interested in platform messages, and hopefully they do not have to debug the platform itself.
- One computation message can be on multiple topics. Querying multiple topics for the same session ID, merging their content into a single time line and representing (visualizing) it in an informative way is a complex task, not comfortable for day-to-day use.

- Developer experience is a metric that a development tool should respect with topmost priority. There are other tools made for exactly this purpose developed and maintained by active developer communities.
- Using the message queue for tracing as well would lead to lock the implementation to message queues, making the support of other composition technologies harder.

4.1.2. Kubernetes level Jaeger

Jaeger is an open source tool supporting OpenTracing API with multiple integration options from Apache Thrift API to REST API [6]. It has API for querying and a developer friendly web UI for browsing and querying traces.

Jaeger can be installed into a Kubernetes cluster with ease, tracing all network calls of the cluster, internal and external as well.

While Jaeger sounds promising, using the same API as the popular Zipkin tracing tool, the maturity level is good. The problem is the Kubernetes level tracing. Kubernetes is the bottom-most level of our architecture (over container level). For a developer who wants to debug a LambdaKube calculation, Kubernetes calls, Kubeless calls are too much noise – in fact, with every function composition, Kubeless controller, the message broker is called.

4.2. LambdaKube level Jaeger

Jaeger seems to be a good tracing implementation. LambdaKube should integrate it.

4.3. Jaeger integration

When integrating tracing into LambdaKube the following have to be kept in mind:

- The generated code should not contain tracing as a boilerplate in all function.
- Runtime framework can inject tracing before and after each function invocation.

With trace messages sending can be injected to the runtime framework, the tracing service – Jaeger – should be placed into the LambdaKube platform layer.

5. Conclusion

The LambdaKube Project has well-defined architectural layers which proven to be extensible with a dynamic development tool, stack tracing. In this paper, multiple options for tracing integration has been evaluated keeping in mind the separation of concerns, respecting boundaries between architectural layers. The decision has been made between a solution based on already used technologies in LambdaKube

and a popular tool which is made for the purpose and fits better into the architecture.

The solution proposed and introduced in this paper supports tracing, the readability of trace messages and generated code as well – by not generating redundant noise into the target code.

References

[1] Balla, D., Maliosz, M., Simon, Cs., Gehberger D., Tuning Runtimes in Open Source FaaS. In Proc. of the Internet of Vehicles. Technologies and Services Toward Smart Cities (IOV 2019), Lecture Notes in Computer Science, Vol. 11894, Springer

[2] Bernstein, D., Containers and cloud: From LXC to Docker to Kubernetes, *IEEE Cloud Computing*, Vol. 1(3) (2014), 81–84.

[3] Buchanan, S., Rangama, J., Bellavance, N., Helm Charts for Azure Kubernetes Service, Introducing Azure Kubernetes Service, Apress, Berkeley, CA, pp. 151–189.

[4] Castro, P., Ishakian, V., Muthusamy, V., Slominski, A., The Rise of Serverless Programming, *Communications of the ACM*, Vol. 62(12) (2019), pp. 44–54.

[5] Castro, P., Ishakian, V., Slominski, A., Serverless Programming (Function as a Service), in Proc. of the 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 2658–2659.

[6] Engel, T., Langermeier, M., Bauer, B., Hofmann, A., Evaluation of Microservice Architectures: A Metric and Tool-Based Approach, In Proc. of the Information Systems in the Big Data Era (CAiSE 2018), Lecture Notes in Business Information Processing, Vol. 317, pp. 74–89.

[7] Kritikos, K., Skrzypek. P., A review of serverless frameworks, in Proc. of 2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC 2018), ACM, pp. 161–168.

[8] Medel, V., Rana, O., Banares, J. Á., Arronategui, U., Modelling Performance and Resource Management in Kubernetes, In Proc. of IEEE/ACM 9th International Conference on Utility and Cloud Computing (UCC 2016), pp. 257–262.

[9] Révész, Á, Pataki, N., Containerized A/B Testing, in Proc. of the Sixth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications (SQAMIA 2017), pp. 14(1)–14(8).

[10] Schröter, A., Bettenburg N., Premraj, R., Do stack traces help developers fix bugs?, In Proc. of 7th IEEE Working Conference on Mining Software Repositories (MSR 2010), pp. 118–121.