Request Log API Pattern
An architecture pattern for a better logging and easy testing of microservices

Marcel Hahn, Albert Zündorf
University of Kassel, Software Engineering Research Group,
Department of Computer Science and Electrical Engineering,
Wilhelmshöher Allee 73, 34121 Kassel, Germany
`[ hahn | zuendorf ]@uni-kassel.de`
`https://seblog.cs.uni-kassel.de/`

Abstract

The practical testing of a microservice is still a big challenge. This paper introduces a special microservice logging API as an architecture pattern that can be used to retrieve additional debug information for each request. This additional information can be used for unit testing and improved logging of microservices.

1 Introduction

The larger and the more distributed a system is, the more important it is to ensure that all requests to the system are fully logged. It is no longer sufficient for each system component to log the requests in its own log file. Rather, in the event of an error, the request that caused the error must be able to be tracked and reproduced across several microservices or components throughout the entire system. Currently, this is usually done using a central logging middleware such as the Elastic Stack[*]. However, this log information cannot easily be used for tests, for example on the production system.

Another challenge is the testing of asynchronous service requests, for example via a message bus. For a synchronous request using HTTP, for example, the server responds directly to a request. When communicating via a bus, the service usually does not respond to a message, but the middleware guarantees delivery. An isolated test would have to monitor all side effects of the microservice, such as messages to other components. In a productive system, these side effects are difficult to isolate and therefore almost impossible to track.

The pattern proposed here offers the possibility to get isolated logging and debugging information for synchronous and asynchronous microservice requests via another API interface.

[*] https://www.elastic.co/

2 Request Log API Pattern

The Request Log API Pattern works in such a way that when a request is made to an API, a log entry with detailed debugging information is not only sent to the logging infrastructure, but can also be retrieved for a limited period of time via a Request Log API at the microservice. Like the Primary API, the Request Log API can be implemented synchronously or asynchronously.

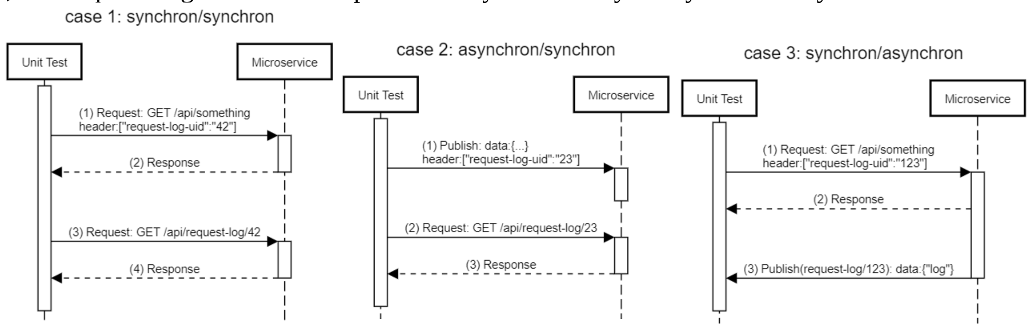
case 1: synchron/synchron
Unit Test
Microservice
(1) Request: GET /api/something header:["request-log-uid":"42"]
(2) Response
(3) Request: GET /api/request-log/42
(4) Response
case 2: asynchron/synchron
Unit Test
Microservice
(1) Publish: data:{...} header:["request-log-uid":"23"]
(2) Request: GET /api/request-log/23
(3) Response
case 3: synchron/asynchron
Unit Test
Microservice
(1) Request: GET /api/something header:["request-log-uid":"123"]
(2) Response
(3) Publish(request-log/123): data:{"log"}

Figure 1 Request Log API Sequence

Figure 1 shows three possible sequences of the pattern. There are four ways to implement the pattern. Variant 1: Both APIs are synchronous. Variant 2: The primary API is asynchronous, and the Request Log API is synchronous. Variant 3: Primary synchronous, Log asynchronous. Variant 4 both asynchronous. In each of the variants, the client adds a unique identifier (UUID) to the header data of its request e.g. under the `request-log-uid` key. If the Request Log API is synchronous, the server then uses the Request Log API + the UUID to provide all related log and debugging information for the request e.g. for 10 seconds. If the Request Log API is implemented asynchronously, the server responds with the log data on the channel `request-log/<UUID>.`

3 Discussion, conclusion and future work

Since using the Request Log API Pattern in our projects, we have been able to significantly improve the quality and testability of our microservices. The pattern enables us to constantly monitor the stability of our APIs in the production system with tests without noticeable performance losses. The overhead could usually be avoided by only providing the request logs to the API if a token is also sent in the header. This is only the case with unit tests. Otherwise, the log message is only sent to the logging infrastructure as it was before the pattern was introduced. Since the logs should already be available, the implementation effort for the introduction of the pattern is manageable. The log message only must be cached in the service for a short period of time or transmitted to another channel.

What the pattern still lacks is an integration formal OpenAPI[†] specification such as Swagger[‡]. Then you could also integrate the log results into a generated API browser and support developers in understanding the API even better.

[†] https://www.openapis.org/
[‡] https://swagger.io