# Modular Monolith: Is This the Trend in Software Architecture?

Ruoyu Su
M3S, University of Oulu
Oulu, Finland
ruoyu.su@.oulu.fi

Xiaozhou Li
M3S, University of Oulu
Oulu, Finland
xiaozhou.li@oulu.fi

## ABSTRACT

Recently, modular monolith architecture has attracted the attention of practitioners, as Google proposed a "Service Weaver" framework to enable developers to write applications as modular monolithic and deploy them as a set of microservices. Google considered it a framework with the best of both worlds, and it seems to be a trend in software architecture. This paper aims to understand the definition of the modular monolith in industry and investigate frameworks and cases building modular monolith architecture. We conducted a systematic grey literature review, and the results show that modular monolith combines the advantages of monoliths with microservices. We found three frameworks and four cases of building modular monolith architecture. In general, the modular monolith is an alternative way to microservices, and it also could be a previous step before systems migrate to microservices.

## KEYWORDS

software engineering, software architecture, modular monolith, microservices, systematic grey literature review

**ACM Reference Format:**
Ruoyu Su and Xiaozhou Li. 2024. Modular Monolith: Is This the Trend in Software Architecture?. In *2024 International Workshop New Trends in Software Architecture (SATrends '24), April 14, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3643657.3643911

## 1 INTRODUCTION

Microservices have been getting increasingly popular in the last few years, and many companies are migrating monolithic applications to microservices [10]. Especially in the industry, companies like Amazon and Netflix are looking to take advantage of microservices such as independent development and deployment to help their systems solve problems [6].

However, several companies did not get the expected benefits from migrating to microservices and fell into difficulty because of issues such as high cost and complexity of microservices [9]. Since AWS Re-Invent 2018 started to mention that blindly migrating to microservices was a mistake [1], more and more practitioners have started discussing the topics of migrating from monolith to microservices. There are examples of switching from microservices back to the monolith like Amazon PrimeVideo [5].In general, monoliths and microservices are not "perfect" and they have their different drawbacks [9].



*SATrends '24, April 14, 2024, Lisbon, Portugal*

ACM ISBN 979-8-4007-0560-1/24/04.
https://doi.org/10.1145/3643657.3643911

Recently, the concept of "Modular Monolith" has attracted the attention of practitioners, as Google proposed the "Service Weaver" framework to enable developers to write applications as modular monolithic and deploy them as a set of microservices [2]. Google explains that it is a framework with the best of both worlds: the development velocity of a monolith, with the scalability, security, and fault-tolerance of microservices [2].

Modularization is not new knowledge; it has been proposed in the last century that modularization is a mechanism for improving the flexibility and comprehensibility of systems [7]. It differs from a monolithic system in that modularization divides the system into separate modules, and independent teams can work on each module so that it reduces product development time and has greater flexibility and comprehensibility [7].

The proposal of the modular monolith is exciting; it has similarities with the traditional monolith and modularization mechanism proposed in the past but is different. It seems to be the "middle ground", the combination of monolith and microservices that addresses the issues of monoliths and microservices. The recently proposed Service Weaver framework has triggered even more discussion among practitioners about modular monolithic architectures.

This paper aims to understand the definition of the modular monolith in industry and investigate frameworks and cases in building modular monolith architecture. Based on our goals, we define the following research questions: $\boldsymbol{RQ_1}$ *What is modular monolith?* $\boldsymbol{RQ_2}$ *Are there any frameworks proposed for building modular monolith architecture?* $\boldsymbol{RQ_3}$ *Are there any cases that use modular monolith architecture?* We conducted a systematic grey literature review, and finally, 64 results were included.

Results show that modular monolith is a software architecture pattern that combines the advantages of monolith with microservices architecture. This architecture organizes systems into loosely coupled modules, each delineating well-defined boundaries and explicit dependencies on other modules. It especially differs from the previously mentioned modularity in that it can be moved or deployed as microservices later if you want. We found three frameworks for building the modular monolith: Service Weaver, Spring Modulith, and Light-hybrid-4j. Four cases are found that use modular monolith rather than microservices: Shopify, Appsmith, Gusto (Time Tracking), and PlayTech (Casino Backend Team). However, they do not use the frameworks proposed above.

The remainder of this paper is organized as follows: Section 2 reports the research method employed to conduct a systematic grey literature review. In Section 3, we analyze the results that address the goal of the study. Finally, Section 4 concludes the paper and outlines our future work.

## 2 METHODOLOGY

In this paper, we aim to understand the definition of the modular monolith in industry and investigate frameworks and cases in building modular monolith architecture. To such an end, we conducted a systematic grey literature review (SGLR) based on the guidelines defined by [3][4]. Herein, we used the following query in the grey literature search based on the identified keywords:

*(serviceweaver OR "service weaver" OR "modular monolith" OR "modular monolithic")*
*AND*
*("microservice" OR "microservices")*

By searching on Google, we obtained 140 results in total. Each result was read and evaluated by two authors and the inclusion criteria is that the studies mention definitions, frameworks or cases on modular monolith. In the case of disagreement, both authors would discuss together and decide whether to include or exclude these results. We also adopted the "snowballing" method in the process to extract more relevant information [11].

## 3 RESULTS

According to the grey literature review, 64 studies that mention the definition, frameworks, and cases of the modular monolith are finally included. The complete paper, with the list of 64 selected studies(Ss), is available at [8].

When dividing these selected 64 studies by the publication year, we can observe the trends of the discussion on modular monolith. As shown in Fig.1, the discussion about the topic of modular monolith started in 2019, and the number of selected studies is stable until 2022. However, its number increases dramatically in 2023. This trend indicates that modular monoliths have become a hot topic and have sparked intense discussion among practitioners.

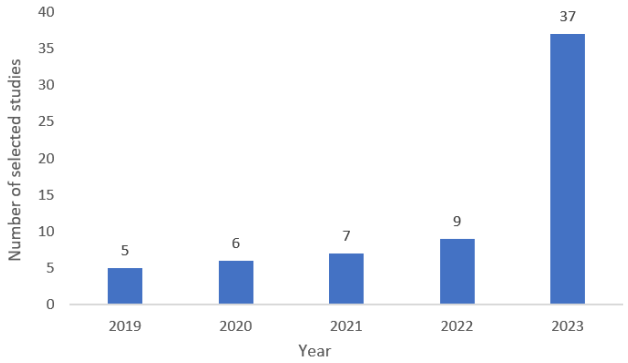


**Figure 1: Selected Studies By Year**

### 3.1 $RQ_1$ What is modular monolith?

Of the included 64 results, 47 studies answered the RQ1. We extracted data with four different parts of the modular monolith: Concept, Characteristics and features, Code structure and Testing.

- **Concept**

Modular Monolith is a software architecture pattern that strategically combines the simplicity of a monolithic structure with the advantages of microservices. In this approach, the system is organized into loosely coupled modules, each delineating well-defined boundaries and explicit dependencies on other modules. The goal is to achieve independence and isolation for each module, allowing them to be worked on independently while still being deployed collectively as a single unit. This architectural style seeks to strike a middle ground between traditional monolithic and microservices architectures. It emphasizes the interchangeability and potential reusability of modules, promoting a clear programming interface between them. The focus on business domains rather than technical layers, along with the vertical stacking of modules, enhances code organization and maintainability. Ultimately, a Modular Monolith represents a holistic and flexible approach to application design, where each module encapsulates specific functionality, fostering ease of development, testing, and deployment. Crucially, it can continue to be migrated to microservices or remain unchanged, which is more convenient than migrating directly from monolith to microservice [S1-S44].

- **Characteristics and features**

We found 6 studies that mention the characteristics, and we summarized 6 characteristics of the modular monolith: (1) Segregation of modules. Each module is independent with its own layers such as Domain, Infrastructure and API. Modules are autonomously developed, tested, and deployed, affording the flexibility to employ diverse database solutions [S18][S24]. (2) Modularity with loose coupling and high cohesion. Modules exhibit loose interdependence and strong internal cohesion. Communication between modules occurs through APIs, preferably adopting loosely coupled asynchronous communication patterns [S1][S18][S24][S46]. (3) Unified Database Schema. The system adheres to a singular database schema, in contrast to microservices, where each microservice necessitates an individual schema [S1]. (4) Monolithic Deployment Structure. All modules within the modular monolith operate within the same Virtual Machine (VM), or each module may run on dedicated VMs. The scale of modules renders them impractical to be encapsulated within containers [S1]. (5) Unified Application Process. The application functions as a singular process, offering a uniform solution applicable to diverse scales of applications. Notably, there exists no rigid data ownership delineation among modules [S22][S24][S26]. (6) Enhanced Maintainability and Scalability. Comparative to traditional monolithic architectures, the modular monolith model demonstrably enhances both maintainability and scalability, underscoring its efficacy in managing complexities and facilitating growth [S24].

We also found 9 features of modular monolith in the selected studies [S4][S5][S11][S17][S62]: (1) A module is never completely independent. It has dependencies with other modules, but these should be minimal; (2) Modules are interchangeable. (3) Code is reusable. (4) Better organization of the dependencies compared to traditional monolithic apps. (5) Easier to maintain and develop new versions than traditional monolithic apps. (6) You can keep the whole project as a single unit, without needing different servers for deployment. (7) More scalable than traditional monolithic apps. (8) Less complex than microservices architecture. (9) Have defined API, allowing access to the logic of each module through public methods, not internal functions and the logic of each one.

- **Code structure**

In modular monolith architecture, the code should contain multiple functional modules, each module having an interface that represents its public definition [S10]. In the selected studies [S10][S12][S13], authors proposed the general modular monolith architecture structure that exposes module interfaces in two ways: Externally, the module offers an API via REST HTTP or GRPC, with API calls managed by a proxy or gateway. Internally, services access the module through an abstracted interface, enabling information retrieval without direct access to the implementation. This upholds a clear separation of concerns, preserving application processes.

- **Testing**

Different from traditional monolithic systems but similar to the microservices systems, the modular monolith should have its own set of unit testing for each module to ensure that its functionality works as expected in isolation because it divides the system into modules [S14]. The interaction between each module should also be verified by performing integration testing. In addition, developers also need to review and refactor the modular monolith on a regular basis to keep the code base neat maintainable and adaptable to changing business requirements [S14].

## 3.2 $RQ_2$ Frameworks proposed in building modular monolith architecture

Of the included 64 results, 17 studies answered the RQ2, and there are three frameworks proposed for building modular monolith architecture: Service Weaver, Spring Modulith, and Light-hybrid-4j.

- **Service Weaver**

Service Weaver is an open-source framework, written by Google, for building and deploying distributed systems. It provides the idea of decoupling the code from how code is deployed [S45][S51][S52][S53][S54][S55]. The framework is currently only available in Go. It allows people to write the application as a modular monolith and deploy it as a set of microservices [S58][S59][S60][S62][S63][S64].

Service Weaver consists of two core pieces: A set of programming libraries that allow developers to write their application as a single modular binary using only native data structures and method calls; A set of deployers that allow developers to configure the runtime topology of their application and deploy it as a set of microservices, either locally or on the cloud of their choice [S51][S54][S59][S60][S64].

The motivation for building Service Weaver is Google found that the expense of maintaining multiple different microservice binaries with separate configuration files, network endpoints, and serializable data formats significantly slowed down the development of microservice-based applications [S51][S59][S60]. Furthermore, microservices heightened the difficulty of cross-binary changes and rendered API modifications very challenging [S54][S59][S60]. Therefore, developers hoped the monolithic binaries would be easy to write, update, and run locally or in a VM [S54][S63][S64].

The goal of Service Weaver is to improve distributed application development velocity and performance [S59][S60][S63][S64]. Its core idea is a modular monolith model that creates a unified binary using native language data structures and methods, organized into modules (components) implemented as native types [S51][S63][S64].

Service Weaver has the following steps: First is to split the application into components written as regular Go interfaces. Secondly, call the components using regular Go method calls and no need for RPCs or HTTP requests. Then, test the application and deploy it to the cloud. Finally, run the components wherever in the same or different process and scale up or down to match load [S52][S54][S64].

Service Weaver has more powerful features compared to traditional modular monoliths. Service Weaver has highly performant in that co-located components communicate via direct method call while remote components communicate using highly efficient custom serialization and RPC protocols [S52][S62]. It has minimal configuration and deploys to the cloud without extensive boilerplate configuration [S52]. Service Weaver provides libraries logging, metrics, and tracing, automatically integrated into the deployed cloud environment [S52][S53]. In addition, it can do the effective sharding that shard requests across various component replicas for optimized performance [S52][S62]. More importantly, Service Weaver has flexible scalability, it can easily scale applications horizontally or vertically on demand, whether as a monolithic application or a distributed microservice [S52].

However, the Service Weaver framework is still in development and now only has the v0.1 release which includes: The core Go libraries used for writing your applications; A number of deployers used for running your applications locally or on GKE; A set of APIs that allow you to write your own deployers for any other platform [S51][S54][S59][S60][S64].

- **Spring Modulith**

Spring Modulith is an experimental Spring project designed for modular monolith applications, with its source code organized based on the module concept [S38][S43][S44][S48]. It provides conventions and APIs for declaring and validating logical modules within the Spring Boot application [S44].

Spring Modulith ensures a modular structure for Spring Beans to grant control over what to expose. The core concept revolves around application modules, representing units of functionality with exposed APIs. Expressing modules can be done through various methods, including organizing domain or business modules as direct sub-packages. Module encapsulation is a key feature, safeguarding internal implementations through sub-packages of the application module's base package. Additionally, it restricts type visibility between modules, allowing access to module content [S38][S43][S44].

The first release of Spring Modulith introduces advanced features, including enhanced package arrangements, flexible module selection for integration tests, a transaction event publication log for seamless integration, automatic developer documentation generation with diagrams, runtime observability at the module level, and Passage of Time Events implementation [S44].

- **Light-hybrid-4j**

The light-hybrid-4j framework is designed to be a modular monolith framework, built on top of the light-4j for modularized monolithic and serverless architecture from the Light platform [S37]. The benefit of using this framework when building a modular monolith system is gives flexibility for deployment and saves production costs [S37].

The framework involves creating a server with integrated third-party dependencies and building multiple services with shared or

specific dependencies. Once services are developed, they are compiled into independent jar files containing only business handlers. These jars are placed in a designated folder on the host running the server, and upon server startup, all services in the folder are loaded. Traffic is then routed to the appropriate handler in the respective service for incoming requests. Developers adhere to the principle of building services into jar files, deploying them to the same Docker container volume, and loading them into the same JVM. Services communicate through interfaces, concealing implementation details. If needed, services experiencing heavy loads can be separated into individual containers for scalable deployment [S37].

### 3.3 $RQ_3$ Cases that use modular monolith architecture

Of the selected 64 studies, 6 studies answered the RQ3, and there are four cases use modular monolith architecture rather than microservices(specifically emphasized this): Shopify, Appsmith, Gusto (Time Tracking) and PlayTech (Casino Backend Team).

**Shopify** is one of the largest Ruby on Rails codebases in existence, worked on for over a decade by more than a thousand developers [S12]. It was initially built as a monolith all the distinct functionalities were built into the same codebase with no boundaries between them. To better development, Shopify chose to change its architecture. Shopify originally considered microservices but finally chose modular monolith to balance the benefits of a single codebase with clear component boundaries. Therefore, Shopify is a great example that has used this approach as an alternative to microservice decomposition [S12][S30].

**Appsmith** is a platform that helps developers build internal apps [S34].They originally considered a microservice-based architecture, However, they rejected this alternative due to many considerations for on-premise deployment of a microservice-based application. Given their core on-premise deployment as a focal point of their business, they deemed it imperative to avoid introducing complexity into the deployment process for end users [S33][S34]. Therefore, they chose modular monolith because it allowed them to maintain one large codebase, deploy a single binary, and balance the interests of their end customers with those of our internal team and open-source community [S33][S34].

**Gusto** wanted to build a new feature named "TimeTracking" and it would be a separate domain that has its own service [S39]. To achieve this feature, they chose modular monolith, because for Gusto, their most important is to have a clear API boundary and make sure the system should not pass rich objects across that boundary [S39]. They did not choose microservices for it has their own sets of challenges in testing and deploying and if got the service boundary wrong, it'd be much more expensive to fix [S39].

**PlayTech (Casino Backend Team)** discovered their company has many departments running microservices and cloud-native applications in the process of continuous refactoring. However, from the perspective of a single team, this maintenance would be unnecessary overhead. Therefore, their team chose a modular monolith as a result [S32].

In general, Shopify, Appsmith, Gusto (Time Tracking) and PlayTech (Casino Backend Team) are great examples that have used modular monolith as an alternative to microservice decomposition. However, they do not use the frameworks proposed in RQ2 above.

## 4 CONCLUSION

The modular monolith architecture has attracted the attention of practitioners for the "Service Weaver" framework proposed by Google. It seems to be a framework with the best of both worlds: the development velocity of a monolith, with the scalability, security, and fault-tolerance of microservices [2]. We conducted a systematic grey literature review, which aims to understand the definition of the modular monolith in industry and investigate frameworks and cases in building modular monolith architecture. Starting from 140 results, 64 related studies were selected. Results show that modular monolith is a software architecture pattern that combines the advantages of monolith with microservices architecture, and in particular, it can be moved or deployed as microservices later if desired. We found three frameworks: Service Weaver, Spring Modulith and Light-hybrid-4j, and four cases: Shopify, Appsmith, Gusto (Time-Tracking) and PlayTech (Casino Backend Team) in building the modular monolith architecture. We conclude that modular monolith is an alternative way to microservices, and it also could be a previous step before systems migrate to microservices. The study helps researchers and practitioners to explore modular monoliths deeply, and provides guidance in software architecture decisions, offering useful directions for future work. In our future work, we shall continue our research on the relationship and migration between modular monoliths and microservices to help organizations enable more effective implementation of architectural changes.

## ACKNOWLEDGMENTS

This work is supported by a grant from the Research Council of Finland (grant n. 349488 - MuFAno) and a grant from Business Finland (grant 24304494 - 6GSoft).

## REFERENCES

[1] 2018. AWS News Blog Category: AWS re:Invent. https://aws.amazon.com/cn/blogs/aws/category/events/reinvent/.
[2] Kucukoglu Ahmet. 2023. What is Modular Monolith? https://serviceweaver.dev/.
[3] Vahid Garousi, Michael Felderer, and Mika V Mäntylä. 2019. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and software technology* 106 (2019), 101–121.
[4] Staffs Keele et al. 2007. Guidelines for performing systematic literature reviews in software engineering.
[5] Marcin Kolny. 2023. Scaling up the Prime Video audio/video monitoring service and reducing costs by 90%. https://www.primevideotech.com/video-streaming/scaling-up-the-prime-video-audio-video-monitoring-service-and-reducing-costs-by-90.
[6] Valentina Lenarduzzi, Francesco Lomio, Nyyti Saarimäki, and Davide Taibi. 2020. Does migrating a monolithic system to microservices decrease the technical debt? *Journal of Systems and Software* 169 (2020), 110710.
[7] David Lorge Parnas. 1972. On the criteria to be used in decomposing systems into modules. *Commun. ACM* 15, 12 (1972), 1053–1058.
[8] Ruoyu Su and Xiaozhou Li. 2024. Modular Monolith: Is This the Trend in Software Architecture? https://doi.org/10.48550/arXiv.2401.11867. arXiv:2401.11867 [cs.SE]
[9] Ruoyu Su, Xiaozhou Li, and Davide Taibi. 2023. Back to the Future: From Microservice to Monolith. *arXiv preprint arXiv:2308.15281* (2023).
[10] Davide Taibi, Valentina Lenarduzzi, and Claus Pahl. 2017. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing* 4, 5 (2017), 22–32.
[11] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 1–10.