

Design and Development of cloud-based microservice architecture for resilient, performant and scalable applications using ADAPT Design Patterns

Mr. Mathieu KERBEL

Senior Software Engineer & Cloud Solutions Architect Creator of the ADAPT Architecture Principle Paris, Île-de-France, France mathieukerbel@gmail.com

The logo consists of two adjacent rounded rectangular boxes. The left box is dark grey and contains the word 'Architecture' in white. The right box is purple and contains the word 'ADAPT' in white.

Figure 1: ADAPT Architecture

Abstract

Microservices have revolutionized software architecture by promoting modularity, scalability, interoperability and resilience. However, traditional design principles like SOLID have shown limitations in addressing the complexities of microservices architectures, more particularly in cloud-based environments. This research paper explores the application of the ADAPT (Asynchronous, Domain-distributed, Abstraction-moderated, Piloted through events, Transparent and observable) architecture principles and design patterns to develop a robust microservices architecture for cloud-based applications.

Introduction

The history of software architecture is continuous struggle against entropy and complexity. As applications grow in size and functionality, the era of monolithic architectures with the logical architecture (how the code is organized) and the physical architecture (how the code is deployed) tightly coupled together has shown its limitations. Microservices architecture emerged as a solution to these challenges, offering a way to break down applications into smaller, independent services that can be developed, deployed, and scaled independently.

If the SOLID principles

\\ continue once results have been achieved.

Literature Review

Methodology

Event-Driven Architecture and DX-based approach

ADAPT Principles and Design Patterns

Asynchronous Communication

“Services must be decoupled, communicating through events rather than direct calls to build resilient and scalable systems.”

First principle of ADAPT architecture [1][2] focuses on asynchronous communication between microservices to enhance scalability and resilience. This section explores various asynchronous communication patterns such as message queues, event streaming, and publish–subscribe models.

Message queues, event streaming, and publish–subscribe models using Apache Kafka

Kafka is a distributed event streaming platform that enables asynchronous communication between microservices. It allows services to publish and subscribe to streams of records, facilitating decoupled interactions. As there are multiple ways to manage message queues, Kafka is one of the most popular solutions for building event-driven architectures, therefore it is used as an example in this section.

Figure 1: Kafka Cluster in an Emitting and Receiving System Component

A few rikes can benefit from using Kafka in microservices architecture include :

- **Decoupling services:** Kafka allows services to communicate without direct dependencies, enhancing flexibility and maintainability.
- **Scalability:** Kafka’s distributed architecture supports high throughput and can handle large volumes of data, its axis of scalability is horizontal due to its distributed nature. As data expand across multiple brokers, Kafka can efficiently manage increased loads by adding more brokers to the cluster.
- **Resilience & Fault Tolerance:** Kafka’s fault-tolerant design ensures that messages are not lost, even in the event of service failures. It achieves this through data replication across multiple brokers and automatic failover mechanisms.
- **Real-time Data Processing:** Kafka enables real-time data streaming and processing, which is essential for applications requiring immediate insights and actions.
- **Event Sourcing:** Kafka can be used to implement event sourcing patterns, where state changes are logged as a sequence of events, allowing for better traceability and recovery.

Domain-Distributed-Driven Design

“The architecture must mirror the business domain, with services organized around business capabilities, not technical layers.”

The second principle of ADAPT architecture [1][2] emphasizes the importance of aligning microservices with business domains. This section discusses Domain-Driven Design (DDD) concepts such as bounded contexts, aggregates, and domain events to structure services around business capabilities.

The ADAPT principle redefines the single responsibility principle from SOLID with a business capability having the complete ownership of its data and logic within a bounded context. This approach ensures that each microservice is responsible for a specific business function, avoiding some common pitfalls of microservices that going against the atomicity of business capabilities.

Abstraction Moderation

Piloted through Events and Configurations

Transparency and Observability through Contacts

Implementation

Case Study: E-commerce Application

To demonstrate the practical application of the ADAPT principles, this chapter details the migration of a reference legacy system to a cloud-based microservices architecture using ADAPT design patterns.

System Architecture

The current baseline architecture of the e-commerce application is structured as a typical legacy e-commerce application with a *layered architecture* with a **Controller Layer**, **Service Layer** and a **Repository Layer**. The application is *monolithic*, with all components *tightly coupled* together.

Technology Stack

The technical stack includes a relational database (*PostgreSQL*), a backend developed in *Java* with *Spring Boot* framework. The current service coupling is high, the **Order Service** has compile-time dependencies on the **Inventory Service** and the **Payment Service**. The Data is managed through a single database schema shared across all services with the **Billing Service** directly accessing the **User Service** database tables.

Benchmarking

The validity of the ADAPT Architecture Principles and Design Patterns is assessed using a suite of quantitative metrics derived from recent software engineering literature. Those metrics replace vague notions of “clean code” and “clean architecture” with measurable criteria and structural attributes.

What to measure?

Derived from Panichella et al. (2021) [15], the structural coupling measures the senity of dependencies between microservices. Lower coupling values indicate better modularity and independence among services. It calcultaed as follows:

$$SC(s_i) = \frac{\sum_{j \in S; j \neq i} dep(s_i, s_j)}{|S|}$$

Where $dep(s_i, s_j)$ indicated a dependency from service i to service j , and $|S|$ is the total number of services in the system. If $SC(s_i) = n$, it means that service s_i depends on n other services.

With microservices architectures, when our services are too granular and too many, there’s a nano-service anti-pattern that can appear, we can measure the Weighted Service Interface Count (WSIC) and the Service Interface Data Cohesion (SIDC).

The WSIC metric [20] measures API surface area and number of interfaces a service exposes, weighted by their complexity.

The SIDC is the metric that quantifies the cohesion of a service based on the cohesiveness of the operations exposed in its interface. It's defined as follows:

$$\begin{aligned}
S &= \text{SOp} (si_s) = \{o_1, \dots, o_n\}, \quad n = |S|. \\
\text{For } 1 \leq i < j \leq n : \\
I_{\text{param}}(o_i, o_j) &= \begin{cases} 1, & \text{ParamTypes}(o_i) \cap \text{ParamTypes}(o_j) \neq \emptyset, \\ 0, & \text{otherwise,} \end{cases} \\
I_{\text{ret}}(o_i, o_j) &= \begin{cases} 1, & \text{Ret}(o_i) = \text{Ret}(o_j), \\ 0, & \text{otherwise.} \end{cases} \\
P &= \binom{n}{2} \quad (\text{number of unordered operation pairs}). \\
\text{Then } SIDC(s) &= \begin{cases} \frac{\sum_{1 \leq i < j \leq n} (I_{\text{param}}(o_i, o_j) + I_{\text{ret}}(o_i, o_j))}{2P}, & P > 0, \\ 0, & P = 0. \end{cases}
\end{aligned}$$

Beyond static code analysis, we also have to consider the operational metrics defined by the DORA (DevOps Research and Assessment) research program, which are widely accepted as key indicators of software delivery performance. Those metrics are the deployment frequency, the change failure rate, the lead time for changes and the mean time to recovery (MTTR).

Performance Testing

Results and Discussion

Scalability

Resilience

Performance

Maintainability

Developer Experience

Future Work

Conclusion

Acknowledgements

Weaknesses

References

- [1] Kerbel, M. (2025). SOLID is killing the micro-services industry — ADAPT saves it. Medium. <https://mathieukerbel.medium.com/solid-is-killing-the-micro-services-industry-adapt-saves-it-7f3f5f4f4f4f>
- [2] Kerbel, M. (2025). The ADAPT Architecture Manifesto. GitHub. <https://github.com/Midekyhub/adapt-architecture-principle/blob/main/README.md>
- [3] Rasheedh, J. A., & Saradha, S. (2022). Design and Development of Resilient Microservices Architecture for Cloud Based Applications Using Hybrid Design Patterns. Indian Journal of Computer Science and Engineering, 13(2), 365–378. PDF: https://www.researchgate.net/profile/J-Rasheedh/publication/360132110_Design_and_Development_of_Resilient_Microservices_Architecture_for_Cloud_Based_Applications_using_Hybrid_Design_Patterns/links/6275381eb1ad9f66c8a72ce1/Design-and-Development-of-Resilient-Microservices-Architecture-for-Cloud-Based-Applications-using-Hybrid-Design-Patterns.pdf
- [4] Bailo, D., Jeffery, K. G., Spinuso, A., & Fiameni, G. (2015). Interoperability Oriented Architecture: The Approach of EPOS for Solid Earth e-Infrastructures. In 2015 IEEE 11th International Conference on e-Science, Munich, Germany. doi:10.1109/eScience.2015.22
- [5] Di Francesco, P., Lago, P., & Malavolta, I. (2019). Architecting with microservices: A systematic mapping study. Journal of Systems and Software, 150, 77–97.
- [6] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2016). Microservices architecture enables DevOps: Migration to a cloud-native architecture. IEEE Software, 33(3), 42–52.
- [7] Newman, S. (2015). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.

- [8] Taibi, D., Lenarduzzi, V., & Pahl, C. (2020). Microservices Anti-patterns: A Taxonomy. In *Microservices* (pp. 111–128). Springer, Cham.
- [9] Soldani, J., Tamburri, D. A., & Van Den Heuvel, W. J. (2018). The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146, 215–232.
- [10] Márquez, G., & Astudillo, H. (2018). Actual use of architectural patterns in microservices-based open source projects. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)* (pp. 31–40). IEEE.
- [11] Lewis, J., & Fowler, M. (2014). Microservices — A definition of this new architectural term. *martinfowler.com*. <https://martinfowler.com/articles/microservices.html>
- [12] Ibryam, B., & Huß, R. (2023). *Kubernetes Patterns: Reusable Elements for Designing Cloud-Native Applications* (2nd ed.). O'Reilly Media.
- [13] Burns, B. (2018). *Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services*. O'Reilly Media.
- [14] Ponugoti, M. (2025). Cloud-Native Platform Engineering: Scalable Design Patterns for Global Enterprise Resilience. *Journal of Computer Science and Technology Studies*, 7(8), 48–59.
- [15] Panichella, S., Rahman, M.I., & Taibi, D. (2021). Structural Coupling for Microservices. *International Conference on Cloud Computing and Services Science*.
- [16] Merson, P. (2020). Principles for microservice design: Think IDEALS, rather than SOLID. *InfoQ*. <https://www.infoq.com/articles/microservices-design-ideals/>
- [17] Martin, R. C. (2017). *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall.
- [18] Millett, S., & Tune, N. (2015). *Patterns, Principles, and Practices of Domain-Driven Design*. Wrox.
- [19] Richardson, C. (2019). Pattern: API Gateway. *microservices.io*. <https://microservices.io/patterns/apigateway.html>
- [20] Restful-ma, C. (2019). Weighted Service Interface Count (WSIC). <https://github.com/restful-ma/rama-cli/blob/master/docs/metrics/WeightedServiceInterfaceCount.md>
- [21] Mateus Gabi Moreira and Breno Bernard Nicolau De França. 2022. Analysis of Microservice Evolution using Cohesion Metrics. In *Proceedings of the 16th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS '22)*. Association for Computing Machinery, New York, NY, USA, 40–49. <https://doi.org/10.1145/3559712.3559716>
- [22] Yiming Z., Tiziano D. M., & Justus B. (2025). How Does Microservice Granularity Impact Energy Consumption and Performance? A Controlled Experiment. <https://arxiv.org/html/2502.00482v1>
- [23] Hirzalla, M., Cleland-Huang, J., Arsanjani, A. (2009). A Metrics Suite for Evaluating Flexibility and Complexity in Service Oriented Architectures. In: Feuerlicht, G., Lamersdorf, W. (eds) *Service-*

Oriented Computing – ICSOC 2008 Workshops. ICSOC 2008. Lecture Notes in Computer Science, vol 5472. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-01247-1_5