Automated Database Schema Evolution in Microservices

Maxime André

Supervised by Prof. Anthony Cleve and Prof. Etienne Rivière
Namur Digital Institute, Computer Science Faculty, University of Namur, Rue Grandgagnage 21, 5000 Namur, Belgium

Abstract

Microservices architecture has emerged as a dominant model for designing cloud-based applications. In this architecture, modular and heterogeneous services, independently deployed and sometimes geo-distributed, dynamically scale and interact with each other to respond to user requests. Typically, each service has its own database(s) and shares information through APIs calls. Facilitating software evolution is one of the main motivations for adopting a microservices architecture. Paradoxically, from a database point of view, recent surveys reveal that database schema evolution remains among the most pressing data management challenges for microservices architecture developers. While there are many tools available in the literature for supporting the evolution of microservices architecture, none of them is specifically designed to address the problem of database schema evolution. To tackle this challenge, the primary objective of this PhD thesis is to provide developers with a set of effective and efficient tools for automatically supporting database schema evolution in microservices architecture, thereby reducing their burden in cloud-based applications evolution.

Keywords

microservice, database schema, automated evolution

1. Introduction

Over the past few years, microservices architectures have gained significant popularity among developers and industry leaders such as Netflix, Google, and Amazon, who leverage this architectural style to design cloud-based applications [1, 2, 3].

A microservices architecture is defined by several key properties. Modularity is the most commonly cited. Within this architectural style, microservices are organized as a collection of small, loosely coupled services, with each service managing its own responsibilities and data, thus respecting separation of concerns [4, 5, 6, 3, 7, 1, 8]. Another property is the use of heterogeneous technologies. Specifically, each microservice may be developed using a diverse set of polyglot and hybrid technologies, including persistence technologies such as relational and NoSQL databases. Each microservice is technologically independent and manages its own codebase and database(s) [5, 8, 3, 6, 7, 9, 2]. Furthermore, microservices should respect deployability property. They are often deployed on public or private cloud infrastructures, accompanied by mechanisms such as autoscaling and geographic distribution. Each microservice and its database(s) can be rapidly and independently redeployed [10, 6, 7, 11]. Due to modularity property, microservices architecture can, on demand, quickly and dynamically scale up and down (autoscaling), without dependency constraints, following pay-as-you-go principle to maintain high availability [10, 5, 6, 3, 11, 12, 13, 1]. For performance reasons, cloud infrastructures are often geographically distributed across multiple regions worldwide, enabling microservices architecture to leverage data partitioning or replication and decentralized processing [10, 6, 9, 5]. Microservices architecture is also renowned for its ability to facilitate collaboration between microservices. This is achieved by exploiting existing lightweight communication protocols strengths, allowing for synchronous or asynchronous exchange of messages and data between microservices via API endpoints that conform to predefined contracts. There are even established patterns for structuring service-to-service calls [4, 5, 8, 6, 3, 13, 1, 9]. Additionally, automation of microservices architecture through DevOps and CI/CD approaches and tools is often claimed as useful at any stage of the project. This property is often associated with deployability property [4, 8, 6, 12, 9]. Finally, evolvability of microservice is a common reason motivating behind its developers adoption. Thanks to modularity, heterogeneity, deployability, and decoupled collaboration properties, combined with automated support tools, DevOps, and CI/CD methodologies, developers can quickly evolve and maintain microservices and databases by detaching from certain constraints [4, 6, 3, 11, 1, 2].

Despite evolutivity property promises, microservices developers continue to face several issues, particularly in regards to database management. Recent surveys reveal that database schema evolution remains one of the most pressing data management challenges for microservices architecture developers [5].

VLDB 2023 PhD Workshop, co-located with the 49th International Conference on Very Large Data Bases (VLDB 2023), August 28, 2023, Vancouver, Canada
maxime.andre@unamur.be (M. André)
https://researchportal.unamur.be/fr/persons/maxime-andre (M. André)
0009-0002-1241-8319 (M. André)

Indeed, practitioners often lack resources to assist them in microservices evolution [8], especially in regards to their databases. Developers have a real need of support tools able to understand, explicit, ensure, visualize and monitor, over time, databases, schemas, data, transactions, and constraints [6, 10, 9]. Particularly, it concerns data replication, implicit constraints such as implicit foreign keys, message structure and database schema change propagation distributed across several microservices [5, 6]. In that context, dealing with CAP theorem [9] and ensuring ACID is a true challenge [8]. This is even more significant considering that modern systems employ hybrid databases composed of heterogeneous database technologies, including NoSQL, particularly known for their absence of fully explicit schemas where consistency is usually delegated to client programs. This complexifies modeling and evolution of database schemas, thereby further justifying the need for support tools [14, 15]. Moreover, even if DevOps gains in popularity, less attention is given to automation of database schema evolution [16]. While microservices are often associated with quick and frequent releases, there is a potential risk of microservices dependencies breaking when introducing a change [6]. This is particularly true when modifications involve substantial database schema changes leading to an expensive cost and even a total downtime [16, 17]. Practitioners are directly affected by those problems while spending precious time in manual operations [9].

In that way, this research aims to answer this main research question: **RQ. How to automate database schema evolution in microservices architecture to reduce developers burden?** The following sub-research questions structure the work in more details. The first four (■) will be the focus of this thesis. Accessorily, the last two (□), to be further refined, will be addressed collectively through collaborative work by the RAINDROP project researchers.

- ■ **SRQ1. [Reverse engineering]**: How to automate reverse engineering of a cross-microservices database schema?
- ■ **SRQ2. [Modeling]**: How to model a cross-microservices database schema?
- ■ **SRQ3. [Generation]**: How to automate generation of artefacts according to a cross-microservice database schema?
- ■ **SRQ4. [Evolution]**: How to automate evolution of a cross-microservice database schema while propagating changes?
- □ **SRQ5. [Visualization]**: How to visualize a cross-microservice database schema?
- □ **SRQ6. [Monitoring]**: How to monitor microservices architecture databases to prepare potential evolution of its schema?

2. Related work

To prepare that research, works related to microservices and databases evolution have been reviewed. Even if they do not focus on both microservices and databases together, they represent good inspiration for reverse engineering, modeling, generation, evolution, visualization, and monitoring of microservice databases.

Reverse engineering. Reverse engineering of microservices aims to recover a comprehensive holistic view of an existing system. Various approaches exist for that purpose. Some authors exploit the version history by mining software repositories to identify changes helping to understand the current version [3, 18]. Others use techniques of static and dynamic analysis. In an offline and white-box approach, static analysis analyzes the source code and API contract specifications. In a runtime and black-box approach, dynamic analysis inspects logs, deployment platform metrics, and any other traces. Both approaches are implemented in tools like MICROLYZE, MicroART, Zipkin, etc. [7, 11, 19, 1, 20, 21]. Techniques and tools operate either at software or at infrastructure level [13]. These approaches primarily examine microservices architecture but lack of persistence perspective in a distributed and heterogeneous context.

Modeling. Modeling in the context of microservices can serve for representing reverse engineering output or simply for designing a new architecture. Additionally to traditional models such as UML, Archimate, and BPMN [19], others are specifically dedicated to microservices. Some propose to combine information from different system layers such as instance, architectural and infrastructure layers [3] or service, infrastructure and interaction layers [13]. Other focuses on distinction between abstract types of components and their concrete deployment instance configurations [12]. Certain models lead to the creation of Domain Specific Language (DSL) [1]. In the same way, creating a modeling language specifically for databases schemas is also studied. For instance, a DSL exists for specifying hybrid polystores schemas [22] and another for abstracting database schema changes coupled to the program code [23]. Finally, in a meta-modeling approach, some research investigates how to design well a modeling language in the context of cloud architecture [24]. Despite interesting directions, no work considers modeling distributed and heterogeneous microservices and databases jointly.

Generation. Accordingly to modeling, DSLs offer capabilities like code generation [24]. For instance, a modeling language representing hybrid polystores schemas plans an extension able to generate data manipulation APIs [22]. Again, no work is specifically dedicated to microservice database artefact generation in a heterogeneous and distributed context.

Evolution. Some evolution approaches suggest reusing a model as a starting point for evaluating changes impact between a current and a future version [3]. Retrospectively and with support of reverse engineering, information recovered is used for understanding architectural changes. Combining modeling and reverse engineering approaches help for decisions making (e.g., splitting or merging microservices). Recovered indicators (e.g., size of a microservice, number of interfaces, coupling index) take part in problem identification and change impact evaluation [13]. Knowing that evolving microservices also means evolving their database(s), it is clear that those techniques are relevant in the context of distributed and heterogeneous microservice database schema evolution. However, no work currently put together these approaches and tools despite the need [16, 17].

Visualization. Many visualizations exist for representing graphically microservices architecture. The most popular take the form of graphs highlighting specifically the interdependencies [11, 25]. Depending on tools, various visualizations are considered such as metaphor-based visualization, 3D visualization, and virtual or augmented reality visualizations. Visualization is closely linked to modeling, sometimes being its graphical representation. Hence, some switch between different views and combination of them like architectural, process, or data views among many others depending on the targeted abstract level [19, 21]. Unfortunately, these views stop at the microservices level and do not go into the details of their distributed and heterogeneous databases. Visualizations of databases exist but none are specifically applied to microservices databases [26].

Monitoring. Microservices monitoring aims to collect real-time data (e.g., runtime metrics) for following up system behaviour. Even if tools and practices appear like reverse engineering, specifically for dynamic analysis, the approach is more centred on the instant. Goals are mainly focused on microservice current state checking and changes detection rather than documentation recovery. Monitoring is essential for preparing any evolution while making decision based on data collected. Those tasks are usually ensured by tools such as OpenTelemetry, Kiali, Zipkin, Jaeger, Prometheus [11, 19, 21]. Regrettably, analyses often ignore the database perspective. There is a need for tools to monitor microservices messages transit through the entire distributed and heterogeneous architecture, from API endpoints to inside the database.

3. Research direction

This PhD study aims to address weaknesses evoked in the related work by designing and developing support tools for microservices database schema evolution.

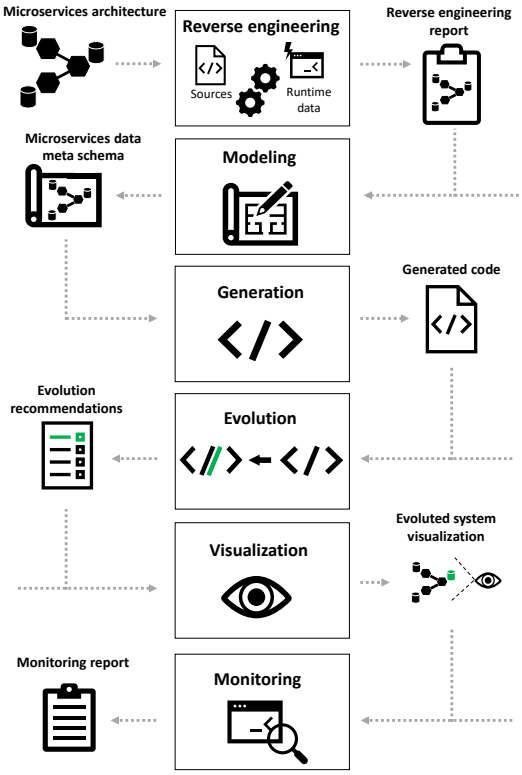

Figure 1: Example of an evolution process supported by envisioned tools.

Figure 1 illustrates an example of evolution process involving all support tools (that could be used independently without any order constraints).

Reverse engineering. The *reverse engineering tool* will help developers to analyze a given heterogeneous and distributed *microservices architecture* (*sources* and *runtime data*) under a data-oriented vision (e.g., concepts, associations, constraints, database accesses, data locations, etc.) to produce a *report* as a basis for the *microservices data meta schema.* The scientific result will be an algorithm based on static and dynamic analyses.

Modeling. The *modeling tool* will help developers to model a microservices architecture, its components and its (cross-)constraints, new or existing, to produce or edit a *microservices data meta schema.* The scientific output will be a DSL defining microservice data meta schemas and evolution operators.

Generation. The *generation tool* will help developers to generate, based on the *microservices data meta schema*, useful *generated code* for microservices database (e.g., database creation script, additional code for cross implicit constraints, etc.). The scientific output will be an algorithm able to generate those artefacts.

Evolution. The *evolution tool* will help developers to evolve a cross microservices database schema based on the *microservices data meta schema*. This process will try to generate, based on evolution operators, what-if scenarios and smells detected, *recommendations* (e.g., implicit constraint materialization, databases split, unused data structure deletion, etc.) responding to developers goals (e.g. consistency, performance, code quality, etc.). The scientific output will be an algorithm able to produce those recommendations and integration in CI/CD tools.

Visualization. The *visualization tool* will help developers to provide *visualizations* of the current or evolved *microservices data meta schema* and its details (e.g., implicit constraints, metrics, changes impact, etc.). The scientific output will be several graphical models customizable on purpose.

Monitoring. The *monitoring* tool will help developers in real-time microservice architecture monitoring while producing a runtime data *report* (e.g., message exchanges, microservices states, critical paths in calls, etc.) specifically related to the *microservices data meta schema*. The scientific output will be several interactive graphical models and monitoring algorithms.

As highlighted, the backbone of this contribution is the *microservices data meta schema*. This representation will model all data-related aspects in a microservice architecture. It will contribute in the whole evolution process of microservices database schema through proposed tools and across intermediary artefacts.

4. Conclusion

In summary, this PhD thesis aims to address challenges related to automated database schema evolution in microservices architecture. The research aims to provide supporting tools reducing developers burden. Specifically, the research will focus on automatic reverse engineering, modeling, generation, evolution, visualization, and monitoring of database schema in microservices. The reverse engineering tool is currently under development with focus on static analysis of JS REST APIs and NoSQL databases.

Acknowledgments

This PhD project is part of the RAINDROP project studying cross-stack adaptations for the edgification of microservices. The project is supported by the subsidy ARC funded by the Wallonia-Brussels Federation (Belgium).

References

[1] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, A. Di Salle, Towards recovering the software architecture of microservice-based systems, in: 2017 IEEE International Conference on Software Architecture Workshops (ICSAW), 2017.

[2] L. P. Tizzei, L. Azevedo, E. Soares, R. Thiago, R. Costa, On the maintenance of a scientific application based on microservices: an experience report, in: 2020 IEEE International Conference on Web Services (ICWS), 2020, pp. 102–109.

[3] A. R. Sampaio, H. Kadiyala, B. Hu, J. Steinbacher, T. Erwin, N. Rosa, I. Beschastnikh, J. Rubin, Supporting microservice evolution, in: 2017 IEEE international conference on software maintenance and evolution (ICSME), IEEE, 2017, pp. 539–543.

[4] J. Lewis, M. Fowler, Microservices: a definition of this new architectural term, MartinFowler. com 25 (2014) 12.

[5] R. Laigner, Y. Zhou, M. A. V. Salles, Y. Liu, M. Kalinowski, Data management in microservices: State of the practice, challenges, and research directions, Proc. VLDB Endow. 14 (2021) 3348–3361.

[6] H. Chawla, H. Kathuria, H. Chawla, H. Kathuria, Evolution of microservices architecture, Building Microservices Applications on Microsoft Azure: Designing, Developing, Deploying, and Monitoring (2019) 1–20.

[7] A. Bakhtin, A. Al Maruf, T. Cerny, D. Taibi, Survey on tools and techniques detecting microservice api patterns, in: 2022 IEEE International Conference on Services Computing (SCC), IEEE, 2022, pp. 31–38.

[8] R. M. Munaf, J. Ahmed, F. Khakwani, T. Rana, Microservices architecture: Challenges and proposed conceptual design, in: 2019 International Conference on Communication Technologies (ComTech), IEEE, 2019, pp. 82–87.

[9] X. Zhou, S. Li, L. Cao, H. Zhang, Z. Jia, C. Zhong, Z. Shan, M. A. Babar, Revisiting the practices and pains of microservice architecture in reality: An industrial inquiry, Journal of Systems and Software 195 (2023) 111521.

[10] D. Abadi, A. Ailamaki, D. Andersen, P. Bailis, M. Balazinska, P. A. Bernstein, P. Boncz, S. Chaudhuri, A. Cheung, A. Doan, et al., The seattle report on database research, Communications of the ACM 65 (2022) 72–79.

[11] V. Bushong, A. S. Abdelfattah, A. A. Maruf, D. Das, A. Lehman, E. Jaroszewski, M. Coffey, T. Cerny, K. Frajtak, P. Tisnovsky, et al., On microservice analysis and architecture evolution: A systematic mapping study, Applied Sciences 11 (2021) 7856.

[12] Y. Wang, D. Conan, S. Chabridon, K. Bojnourdi, J. Ma, Runtime models and evolution graphs for the version management of microservice architectures, in: 2021 28th Asia-Pacific Software Engineering Conference (APSEC), IEEE, 2021, pp. 536–541.

[13] B. Mayer, R. Weinreich, An approach to extract the architecture of microservice-based software systems, in: 2018 IEEE symposium on service-oriented system engineering (SOSE), IEEE, 2018, pp. 21–30.

[14] P. Benats, M. Gobert, L. Meurice, C. Nagy, A. Cleve, An empirical study of (multi-) database models in open-source projects, in: Conceptual Modeling: 40th International Conference, ER 2021, Virtual Event, October 18–21, 2021, Proceedings 40, Springer, 2021, pp. 87–101.

[15] M. Gobert, L. Meurice, A. Cleve, Hydra: A framework for modeling, manipulating and evolving hybrid polystores, in: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2022, pp. 652–656.

[16] M. De Jong, A. Van Deursen, Continuous deployment and schema evolution in sql databases, in: 2015 IEEE/ACM 3rd International Workshop on Release Engineering, IEEE, 2015, pp. 16–19.

[17] A. Maule, W. Emmerich, D. S. Rosenblum, Impact analysis of database schema changes, in: Proceedings of the 30th international conference on Software engineering, 2008, pp. 451–460.

[18] D. A. d'Aragona, L. Pascarella, A. Janes, V. Lenarduzzi, D. Taibi, Microservice logical coupling: A preliminary validation, in: 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C), IEEE, 2023, pp. 81–85.

[19] T. Cerny, A. S. Abdelfattah, V. Bushong, A. Al Maruf, D. Taibi, Microservice architecture reconstruction and visualization techniques: A review, in: 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE), IEEE, 2022, pp. 39–48.

[20] T. Cerny, D. Taibi, Static analysis tools in the era of cloud-native systems (2022).

[21] M. E. Gortney, P. E. Harris, T. Cerny, A. Al Maruf, M. Bures, D. Taibi, P. Tisnovsky, Visualizing microservice architecture in the dynamic perspective: A systematic mapping study, IEEE Access (2022).

[22] M. Gobert, L. Meurice, A. Cleve, Conceptual modeling of hybrid polystores, in: Conceptual Modeling: 40th International Conference, ER 2021, Virtual Event, October 18–21, 2021, Proceedings 40, Springer International Publishing, 2021, pp. 113–122.

[23] S. Scherzinger, W. Mauerer, H. Kondylakis, Debinelle: Semantic patches for coupled database-application evolution, in: 2021 IEEE 37th International Conference on Data Engineering (ICDE), IEEE, 2021, pp. 2697–2700.

[24] A. Alidra, H. Bruneliere, T. Ledoux, A feature-based survey of fog modeling languages, Future Generation Computer Systems (2022).

[25] G. Parker, S. Kim, A. Al Maruf, T. Cerny, K. Frajtak, P. Tisnovsky, D. Taibi, Visualizing anti-patterns in microservices at runtime: A systematic mapping study, IEEE Access (2023).

[26] L. Meurice, A. Cleve, Dahlia 2.0: A visual analyzer of database usage in dynamic and heterogeneous systems, in: 2016 IEEE Working Conference on Software Visualization (VISSOFT), 2016, pp. 76–80.