Managing API Evolution in Microservice Architecture

Alexander Lercher
University of Klagenfurt, Austria
alexander.lercher@aau.at

ABSTRACT

Nowadays, many software systems are split into loosely coupled microservices only communicating via Application Programming Interfaces (APIs) to improve maintainability, scalability, and fault tolerance. However, the loose coupling between microservices provides no immediate feedback on breaking API changes, and consuming services break or exhibit unexpected behavior only after the first actual call to the changed API. Hence, development teams must actively identify and communicate all breaking changes to affected teams to stay compatible. This research addresses this problem with three contributions. First, we identified API evolution strategies and open challenges in practice with an explorative study. Based on the study findings, we formulated two open research directions for evolving publicly accessible APIs, i.e., REpresentational State Transfer (REST) APIs. As the second contribution, we will introduce a REST API change extraction approach to improve the change notification accuracy. We plan experiments on open-source projects to evaluate our approach's accuracy and compare it to openapi-diff for structural changes. Third, we plan to investigate methods for automating communication with affected teams, which will then improve the change notification reliability. Finally, we will evaluate the accuracy and reliability of our notifications with a user study.

KEYWORDS

Microservice Architecture, API Evolution, Web API, REST API

ACM Reference Format:
Alexander Lercher. 2024. Managing API Evolution in Microservice Architecture. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3639478.3639800

1 INTRODUCTION

The Microservice Architecture (MSA) improves the overall maintainability, scalability, and fault-tolerance of a system by replacing a large application with many small and loosely coupled services [16, 24, 26]. Each such microservice manages a subset of the overall system's responsibilities and individual development teams exclusively develop, maintain, and deploy a logical set of microservices. Typically, the MSA employs two means of communication [38]: event-driven for system-internal service communication, e.g., publish-subscribe, and request-response for communicating to external services, e.g., REpresentational State Transfer (REST) [12]. In this paper, we use the term Application Programming Interface (API) for both, as both enable communication between microservices. The communication style in MSA is called "smart endpoints and dumb pipes" and allows the APIs to evolve independently [35].

However, as each microservice API's data structures and behaviors evolve independently, the API evolution process requires more synchronization efforts between the development teams than for a single monolithic code base [24, 39]. Changes in one API could result in unexpected behavior and potentially break consumer services, i.e., services interacting with the API [35]. From the consumer perspective, such breaking changes are undetected at compile time and only manifest during an actual call to the changed API at runtime. Consumer-side contract tests do not mitigate this problem as the API test doubles do not automatically reflect the latest changes. Accordingly, development teams of consumer services rely on the teams providing an API to correctly notify them about all changes in advance. The provider teams can utilize regression testing [2, 15] to detect API contract violations after introducing changes. Still, they must understand the actual API changes leading to these violations. We conducted preliminary discussions with practitioners regarding these challenges. In practice, they manually identify the API changes and notify other teams about them, slowing down the API evolution process and making it susceptible to human error.

2 RELATED WORK

Related works identified the communication and integration of microservices as major challenges in the MSA [1, 32]. Chen et al. [8] conducted a grey literature review on microservice API concerns in practice and identified concerns with API design, API versioning, and API testing. Wang et al. [34] studied REST API change discussions on Stack Overflow and concluded that REST APIs are more change-prone than Java library APIs and the Web Services Description Language (WSDL). Bogart et al. [3] found that developers maintain old API versions to prolong the transition period for consumers but the several separate versions increase the maintenance cost as a result. According to Zdun et al. [37], the microservice API evolution process still misses efficient communication and support for consumers affected by API changes. Similarly, Cerny et al. [6] proposed detecting incompatible API changes and communicating changes to consumer teams as open research challenges. Multiple works focused on detecting and classifying API changes in source code, e.g., in Java libraries [4, 29] and the Android Platform [23]. They automatically identify breaking source code API changes and accordingly update the calls [36] or the API documentation [21]. Related works also investigated the evolution of WSDL [7, 27], but only recently started to target REST APIs [20]. To this end, related works did not discuss the underlying reasons for the API evolution challenges and how to solve them sustainably, and change extraction approaches did not sufficiently target REST API changes.

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0502-1/24/04.
https://doi.org/10.1145/3639478.3639800

3 RESEARCH QUESTIONS

We hypothesize that companies currently do not follow a sustainable microservice API evolution strategy. Further, we hypothesize that automated approaches for reporting structural and behavioral breaking API changes and notifying affected consumers would improve the change notification accuracy and reliability, mitigating human errors and delays with currently used strategies. Hence, we define the following research questions (RQs) to address:

RQ1 Which strategies do developers follow to introduce and communicate API changes in loosely coupled systems, and what challenges do they face?

RQ2 How can structural and behavioral changes of REST APIs be automatically extracted based on the source code?

RQ3 How can relevant REST API changes be automatically communicated to consumer teams affected by these changes?

4 METHODOLOGY AND CONTRIBUTIONS

This section focuses on the research methods to answer the three RQs and the expected contributions.

4.1 Catalogue of API evolution strategies and challenges

To answer **RQ1**, we conducted semi-structured interviews [17] with developers and architects working on developing loosely coupled systems exposing an API, e.g., REST or event-driven, for at least one year. We applied open coding [9] used in grounded theory [14] and analyzed the interview transcripts iteratively. The codebook began to stabilize after 12 interviews, and we did not encounter any new categories for the last three interviews, indicating theoretical saturation [33]. To mitigate threats to validity, we applied investigator triangulation [5] and member checking [28]. We are currently in the process of publishing the results.

In summary [22], we conducted 17 interviews with participants from 11 companies. We identified six microservice API evolution strategies in practice, which we formulated as best practices to follow. Further, we identified six challenges and formulated them as pitfalls when handling API evolution. We drew the relationships between the strategies and challenges and discovered two important problems in the evolution process of publicly accessible APIs, i.e., REST APIs. First, development teams informally communicate REST API changes to affected teams, e.g., through verbal meetings, instant messaging channels, or e-mails, and collaborate closely to avoid or quickly repair unnoticed breaking changes. Second, doubtful or unresponsive consumer teams not migrating to a new REST API version force the provider team to support and maintain outdated versions, continuously increasing technical debt and degrading the overall API design. Based on these two problems, we propose two research directions: a) automating the impact analysis of source code changes on the REST API and b) automating the communication of REST API changes to affected teams.

4.2 Automated REST API change extraction from source code changes

To answer **RQ2**, we plan to propose an approach for extracting REST API changes from source code changes and to evaluate the prototype on Java Spring Boot[1] open-source projects. We selected Java Spring Boot as one of the most popular microservice frameworks offering high flexibility in defining REST APIs [8, 10].

Our approach will consist of a change extractor and a change classifier. The change extractor analyzes the OpenAPI specifications[2] of two REST API versions, generates suitable intermediate models introduced by us, and extracts the structural changes between them using GumTree [11], a popular approach for differencing tree structures [13, 19, 25]. To extract the behavioral changes, we will experiment with the limited behavioral description provided by the OpenAPI specification and supplement it with a lightweight control and data flow analysis of the source code, such as introduced by Hanam et al. [18]. We will also evaluate the feasibility of describing detailed behavioral information by extending the OpenAPI specification. Next, we will investigate existing REST API change type taxonomies [30, 34], currently focusing on structural changes, and will extend them with novel behavioral changes. The change classifier then applies our extended taxonomy on the extracted structural and behavioral changes and finally outputs the corresponding REST API changes. We plan experiments on existing open-source projects to evaluate our approach's applicability and change classification accuracy. Further, we will compare our approach to the state-of-the-art openapi-diff[3], which identifies structural changes between OpenAPI specifications, but does not detect behavioral changes.

4.3 Automated communication of REST API changes to affected teams

To answer **RQ3**, we will build on the findings from our study in Section 4.1 by automating REST API change notifications to affected teams via online collaboration platforms, such as GitHub and GitLab. We will provide GitHub Actions[4] and GitLab CI/CD templates[5] to automatically send REST API change notifications generated from source code with our approach in Section 4.2. We will explore two means of registering for notifications. Consumer teams can enter the relevant service repositories via a web interface or add a dependency file referencing them to their own repository. We will also investigate methods to register private repositories containing proprietary services. Further, we expect that different technical roles prefer different levels of detail and timeliness of notifications. Hence, we will experiment with different granularities and criticality levels of changes and different notification triggers, e.g., opened pull request, completed merge, or passed deployment pipeline. We plan to evaluate our approach with a user study [31]. We will assess its reliability by comparing the timeliness and completeness of our REST API change notifications to manual notifications, and its helpfulness, as done by Frick et al. [13], in terms of understandability compared to manually written or verbal notifications.

In conclusion, the final prototype will incorporate the study insights from Section 4.1, the REST API change extraction approach from Section 4.2, and the automated notification approach. The user study will evaluate our hypothesis that such a tool improves the API change notification accuracy and reliability.

[1]https://spring.io/projects/spring-boot
[2]https://spec.openapis.org/oas/v3.1.0
[3]https://github.com/OpenAPITools/openapi-diff
[4]https://github.com/features/actions
[5]https://docs.gitlab.com/ee/development/cicd/templates.html

REFERENCES

[1] Nuha Alshuqayran, Nour Ali, and Roger Evans. 2016. A Systematic Mapping Study in Microservice Architecture. In *2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*. 44–51.

[2] Swarnendu Biswas, Rajib Mall, Manoranjan Satpathy, and Srihari Sukumaran. 2011. Regression Test Selection Techniques: A Survey. *Informatica (Slovenia)* 35, 3 (2011), 289–321.

[3] Chris Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. 2021. When and How to Make Breaking Changes: Policies and Practices in 18 Open Source Software Ecosystems. *ACM Trans. Softw. Eng. Methodol.* 30, 4, Article 42 (jul 2021), 56 pages. https://doi.org/10.1145/3447245

[4] Aline Brito, Laerte Xavier, Andre Hora, and Marco Tulio Valente. 2018. APIDiff: Detecting API breaking changes. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 507–511. https://doi.org/10.1109/SANER.2018.8330249

[5] John L. Campbell, Charles Quincy, Jordan Osserman, and Ove K. Pedersen. 2013. Coding In-depth Semistructured Interviews: Problems of Unitization and Intercoder Reliability and Agreement. *Sociological Methods & Research* 42, 3 (2013), 294–320. https://doi.org/10.1177/0049124113500475

[6] Tomas Cerny, Michael J. Donahoo, and Michal Trnka. 2018. Contextual Understanding of Microservice Architecture: Current and Future Directions. *SIGAPP Appl. Comput. Rev.* 17, 4 (jan 2018), 29–45.

[7] Animesh Chaturvedi and Dave Binkley. 2021. Web Service Slicing: Intra and Inter-Operational Analysis to Test Changes. *IEEE Transactions on Services Computing* 14, 3 (2021), 930–943. https://doi.org/10.1109/TSC.2018.2821157

[8] Fangwei Chen, Li Zhang, and Xiaoli Lian. 2021. A systematic gray literature review: The technologies and concerns of microservice application programming interfaces. *Software: Practice and Experience* 51, 7 (2021), 1483–1508. https://doi.org/10.1002/spe.2967 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2967

[9] Juliet M. Corbin and Anselm Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative Sociology* 13, 1 (01 Mar 1990), 3–21.

[10] Hai Dinh-Tuan, Maria Mora-Martinez, Felix Beierle, and Sandro Rodriguez Garzon. 2020. Development Frameworks for Microservice-Based Applications: Evaluation and Comparison. In *Proceedings of the 2020 European Symposium on Software Engineering* (Rome, Italy) *(ESSE '20)*. Association for Computing Machinery, New York, NY, USA, 12–20. https://doi.org/10.1145/3393822.3432339

[11] Jean-Rémy Falleri, Floréal Morandat, Xavier Blanc, Matias Martinez, and Martin Monperrus. 2014. Fine-Grained and Accurate Source Code Differencing. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering* (Vasteras, Sweden) *(ASE '14)*. Association for Computing Machinery, New York, NY, USA, 313–324. https://doi.org/10.1145/2642937.2642982

[12] Roy Thomas Fielding. 2000. REST: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California* (2000).

[13] Veit Frick, Thomas Grassauer, Fabian Beck, and Martin Pinzger. 2018. Generating Accurate and Compact Edit Scripts Using Tree Differencing. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 264–274. https://doi.org/10.1109/ICSME.2018.00036

[14] B.G. Glaser and A.L. Strauss. 1967. *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Aldine.

[15] Patrice Godefroid, Daniel Lehmann, and Marina Polishchuk. 2020. Differential Regression Testing for REST APIs. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual Event, USA) *(ISSTA 2020)*. Association for Computing Machinery, New York, NY, USA, 312–323. https://doi.org/10.1145/3395363.3397374

[16] Konrad Gos and Wojciech Zabierowski. 2020. The Comparison of Microservice and Monolithic Architecture. In *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)*. 150–153.

[17] Svetlana Gudkova. 2018. *Interviewing in Qualitative Research.* Springer International Publishing, Cham, 75–96.

[18] Quinn Hanam, Ali Mesbah, and Reid Holmes. 2019. Aiding Code Change Understanding with Semantic Change Impact Analysis. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 202–212. https://doi.org/10.1109/ICSME.2019.00031

[19] Kaifeng Huang, Bihuan Chen, Xin Peng, Daihong Zhou, Ying Wang, Yang Liu, and Wenyun Zhao. 2018. ClDiff: Generating Concise Linked Code Differences. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering* (Montpellier, France) *(ASE 2018)*. Association for Computing Machinery, New York, NY, USA, 679–690. https://doi.org/10.1145/3238147.3238219

[20] Holger Knoche and Wilhelm Hasselbring. 2021. Continuous API Evolution in Heterogenous Enterprise Software Systems. In *2021 IEEE 18th International Conference on Software Architecture (ICSA)*. 58–68. https://doi.org/10.1109/ICSA51549.2021.00014

[21] Seonah Lee, Rongxin Wu, Shing-Chi Cheung, and Sungwon Kang. 2021. Automatic Detection and Update Suggestion for Outdated API Names in Documentation. *IEEE Transactions on Software Engineering* 47, 4 (2021), 653–675. https://doi.org/10.1109/TSE.2019.2901459

[22] Alexander Lercher, Johann Glock, Christian Macho, and Martin Pinzger. 2023. Microservice API Evolution in Practice: A Study on Strategies and Challenges. arXiv:2311.08175 [cs.SE]

[23] Li Li, Tegawendé F. Bissyandé, Haoyu Wang, and Jacques Klein. 2018. CiD: Automating the Detection of API-Related Compatibility Issues in Android Apps. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Amsterdam, Netherlands) *(ISSTA 2018)*. Association for Computing Machinery, New York, NY, USA, 153–163. https://doi.org/10.1145/3213846.3213857

[24] Shang-Pin Ma, Chen-Yuan Fan, Yen Chuang, I-Hsiu Liu, and Ci-Wei Lan. 2019. Graph-based and scenario-driven microservice analysis, retrieval, and testing. *Future Generation Computer Systems* 100 (2019), 724–735.

[25] Junnosuke Matsumoto, Yoshiki Higo, and Shinji Kusumoto. 2019. Beyond GumTree: A Hybrid Approach to Generate Edit Scripts. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. 550–554. https://doi.org/10.1109/MSR.2019.00082

[26] S. Newman. 2019. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith.* O'Reilly Media, Incorporated.

[27] Daniele Romano and Martin Pinzger. 2012. Analyzing the Evolution of Web Services Using Fine-Grained Changes. In *2012 IEEE 19th International Conference on Web Services*. 392–399. https://doi.org/10.1109/ICWS.2012.29

[28] P. Runeson, M. Höst, A. Rainer, and B. Regnell. 2012. *Data Analysis and Interpretation.* John Wiley & Sons, Ltd, Chapter 5, 61–76. https://doi.org/10.1002/9781118181034.ch5

[29] Danilo Silva and Marco Tulio Valente. 2017. RefDiff: Detecting Refactorings in Version Histories. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. 269–279. https://doi.org/10.1109/MSR.2017.14

[30] S.M. Sohan, Craig Anslow, and Frank Maurer. 2015. A Case Study of Web API Evolution. In *2015 IEEE World Congress on Services*. 245–252. https://doi.org/10.1109/SERVICES.2015.43

[31] Klaas-Jan Stol and Brian Fitzgerald. 2020. *Guidelines for Conducting Software Engineering Research.* Springer International Publishing, Cham, 27–62. https://doi.org/10.1007/978-3-030-32489-6_2

[32] Mehmet Söylemez, Bedir Tekinerdogan, and Ayça Kolukısa Tarhan. 2022. Challenges and Solution Directions of Microservice Architectures: A Systematic Literature Review. *Applied Sciences* 12, 11 (2022). https://doi.org/10.3390/app12115507

[33] Frank J. van Rijnsoever. 2017. (I Can't Get No) Saturation: A simulation and guidelines for sample sizes in qualitative research. *PLOS ONE* 12, 7 (07 2017), 1–17. https://doi.org/10.1371/journal.pone.0181689

[34] Shaohua Wang, Iman Keivanloo, and Ying Zou. 2014. How Do Developers React to RESTful API Evolution?. In *Service-Oriented Computing*, Xavier Franch, Aditya K. Ghose, Grace A. Lewis, and Sami Bhiri (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 245–259.

[35] Menghan Wu, Yang Zhang, Jiakun Liu, Shangwen Wang, Zhang Zhang, Xin Xia, and Xinjun Mao. 2022. On the Way to Microservices: Exploring Problems and Solutions from Online Q&A Community. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 432–443. https://doi.org/10.1109/SANER53432.2022.00058

[36] Zhenchang Xing and Eleni Stroulia. 2007. API-Evolution Support with Diff-CatchUp. *IEEE Transactions on Software Engineering* 33, 12 (2007), 818–836. https://doi.org/10.1109/TSE.2007.70747

[37] Uwe Zdun, Erik Wittern, and Philipp Leitner. 2020. Emerging Trends, Challenges, and Experiences in DevOps and Microservice APIs. *IEEE Software* 37, 1 (2020), 87–91. https://doi.org/10.1109/MS.2019.2947982

[38] He Zhang, Shanshan Li, Zijia Jia, Chenxing Zhong, and Cheng Zhang. 2019. Microservice Architecture in Reality: An Industrial Inquiry. In *2019 IEEE International Conference on Software Architecture (ICSA)*. 51–60. https://doi.org/10.1109/ICSA.2019.00014

[39] Olaf Zimmermann, Mirko Stocker, Daniel Lübke, Cesare Pautasso, and Uwe Zdun. 2019. Introduction to Microservice API Patterns (MAP). In *International Conference on Microservices (Microservices 2019)*. https://doi.org/10.4230/OASIcs.Microservices.2017/2019.4