# Documenting Microservice Integration with MSAdoc

Georg-Daniel Schwarz
Technical Faculty
Friedrich-Alexander Universität Erlangen-Nürnberg
Erlangen, Germany
georg.schwarz@fau.de

Dirk Riehle
Technical Faculty
Friedrich-Alexander Universität Erlangen-Nürnberg
Erlangen, Germany
dirk@riehle.org

## Abstract

Microservices are a popular software architectural style that decomposes a large application into smaller services. These microservices integrate at runtime to deliver business value to the users. With an increasing number of microservices, software projects become more difficult to manage. Specifically, maintaining consistent and up-to-date documentation becomes a challenge that can significantly affect the integration efforts in such projects. In this article, we present MSAdoc, an open source tool that helps to prevent documentation from going out-of-date quickly. The tool (1) enables decentralized documentation close to the source code of each microservice and those who have to document it, (2) aggregates documentation centrally across individual microservices to make the documentation accessible in one place and generate higher-order documentation, while (3) supporting technological heterogeneity by relying on the technology-agnostic JSON format. Using a tool like MSAdoc that implements several best practices, practitioners can accommodate the decentralized nature of microservice-based projects and alleviate the problem of maintaining central documentation that quickly becomes outdated.

**ACM Reference Format:**
Georg-Daniel Schwarz and Dirk Riehle. 2025. Documenting Microservice Integration with MSAdoc. In *the 16th International Conference on Internetware (Internetware 2025), June 20–22, 2025, Trondheim, Norway.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3755881.3755980

## 1 Introduction

Microservices have become a widely adopted architectural style for creating robust, scalable software systems that are well-suited for cloud environments [1]. Rather than developing a single large monolithic application, the system is divided into cohesive, loosely coupled units known as microservices, each with its specific responsibilities. Depending on the maturity of the project, microservices can be deployed independently without requiring modifications to other system components, and their development lifecycles can be entirely separate. This loose coupling, both from a domain and organizational perspective, allows multiple teams to work in parallel on large software projects [4].

With an increasing number of microservices, enterprise architecture management (EAM) becomes more difficult. Specifically, maintaining consistent and up-to-date documentation becomes a challenge that can significantly affect the integration efforts in such projects [3]. Enterprise documentation tooling is not tailored to microservice-based projects [3], especially to the distributed nature of such projects. For example, we observed that documentation can be distributed over multiple places and can be handled slightly differently by every team. Additionally, the potential to use different technologies in microservices adds further challenges to documentation tool design.

In this article, we present MSAdoc, a tool that addresses some of the documentation challenges in microservice-based projects by enabling microservice teams to create and maintain documentation in a decentral manner close to the microservice code. At the same time, the tool provides an aggregated view in a central place to browse and read the documentation, including some generated higher-order documentation, like architecture diagrams. The tool implements several techniques we found in prior research, combines them, and, at some points, transfers the mechanism from other fields to documentation.

By using a tool like MSAdoc, practitioners can accommodate the decentral nature of microservice-based projects and alleviate the pain of maintaining central documentation that gets out of date quickly. Future work can extend and use the tool to evaluate their theories, e.g., the underlying techniques the tool is based on.

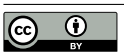
## 2 Microservice Documentation Challenges

We build our work on the findings of Kleehaus and Matthes [3] that conducted an online survey with 58 IT practitioners. Among others, they address the research question "What challenges do Enterprise Architects face in documenting microservice-based IT landscapes?". In the following paragraphs, we discuss a subset of eight challenges. These challenges scored with an average score of 3 or better (1 = fully disagree, 2 = disagree, 3 = agree, 4 = fully agree). We complement them with four further challenges we observed in industry.

*Content-related Challenges (CC).* The main pain point in these challenges is bad data quality. Building documentation on data with low quality might put the whole endeavor on a weak foundation. Root causes are the following challenges:

- CC1: Microservice documentation is mainly created and maintained manually
- CC2: Microservice documentation is wrong or out-of-date
- CC3: Microservice documentation is incomplete

*Tooling Challenges (TC).* These challenges concern the lack of functionality in existing EAM tools. The most prominent tools originate from times before the microservice-architectural style became popular or are not optimized for their special needs:




*Internetware 2025, Trondheim, Norway*

ACM ISBN 979-8-4007-1926-4/25/06
https://doi.org/10.1145/3755881.3755980

- TC1: EAM Tools are not up-to-date with microservice architectures
- TC2: No appropriate visualization for different stakeholders
- TC3: EAM Tools lack of providing runtime KPIs

*Business and Organization Challenges (BC).* Next to the technical view on microservice documentation, we also need to consider an organizational perspective. We will not consider the challenge of budget constraints since it did not reach an agreement score of 3 or higher. The main challenges are the following:

- BC1: Missing motivation and clarification of responsibilities
- BC2: No alignment of superior EA concepts to sub-organizations due to missing standards

In this study, we aim to address these eight most relevant documentation challenges in microservice-based projects to support the successful integration of microservices.

*Additional Observed Challenges (OC).* Further, we observed the following challenges (OC) at our industry partners that we will consider in the tool as well:

- OC1: Documentation is distributed over multiple places, making it difficult to find information.
- OC2: There is a variance in how teams handle documentation, making it difficult to find information.
- OC3: Microservices embrace technical heterogeneity, allowing each microservice to potentially use a different technology stack.
- OC4: In large parts, documentation is project-specific.

## 3 The MSAdoc Tool

MSAdoc is an open source tool[1] to improve microservice documentation and integration by enabling decentralized documentation of microservices close to their source code while aggregating and enhancing it in a central place. The following subsections detail the features of the tool and how they address the previously presented documentation challenges (see Table 1 for an overview).

By building a dedicated tool itself tailored to document microservices (F0), it tackles the challenge of existing EAM tools not being suitable for documenting microservices and their integration (TC1). The upcoming subsections detail the different features and explain how they approach the individual challenges.

### 3.1 Decentralized Documentation (F1)

MSAdoc facilitates decentralized documentation: each team documents its microservice within its code repository instead of in a central place. This fits well with decentralized governance that often accommodates microservice architectures (*T3.4.7 Push more responsibility to teams* in Schwarz et al. [5]). Per convention, there is a *msadoc.json* file in each microservice root directory that documents certain meta-information about the microservice. An example of such a file is shown in Listing 1. At the very least, each microservice needs a unique *name*. It may include links for further meta-data for navigation to project management materials or more fine-granular documentation (*T1.3.2 Document microservice metadata* in Schwarz et al. [5]). Creating and maintaining documentation close to the code aims to reduce the probability of documentation becoming out-of-date (CC2). Co-locating non-application-code artifacts with the code is a practice that finds application in infrastructure-as-code already, where infrastructure files need to change frequently together with different kinds of artifacts to stay in sync [2]. Moving more documentation to the code repository makes it versioned and traceable by default.

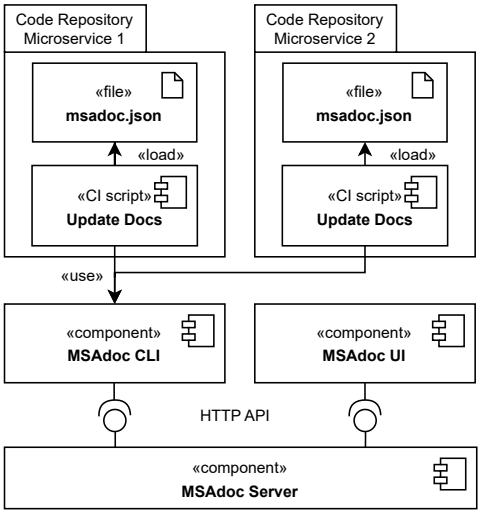



**Figure 1: MSAdoc - Decentralized Documentation**

Continuous integration and deployment (CI/CD) scripts send these *msadoc.json* files to an MSAdoc server instance that aggregates documentation meta-data across all microservices and serves as a user interface to browse the documentation in an aggregated format (see Figure 1). The CI script simply calls an HTTP endpoint, providing a pre-generated API key next to the payload. In this manner, the documentation of the single microservices is aggregated in a central place via standard CI/CD tooling to prevent local and central documentation from deviating, preventing the central documentation from becoming outdated (CC2). CI/CD pipelines are a common technique for continuously integrating and deploying new microservice versions (*T3.3.1 CI/CD for automated deployment* in Schwarz et al. [5]). The centrally aggregated documentation automatically stays up-to-date by adding a phase in the CI/CD pipeline to publish the latest documentation next to the microservice. Aggregating information about the system in a central place is a technique that is facilitated for logging and monitoring data as well (*T3.4.19 Aggregate logging/monitoring information in a central place* in Schwarz et al. [5]).

The documentation can be accessed in a central place to reduce the cognitive load and effort of searching for documentation in multiple places (OC1). The tool may serve as an entry point, linking to further documentation. Having a standard place to start the search for documentation will alleviate the pain of searching in different locations while allowing for the use of different documentation mechanisms at different places (OC2). Standardizing the location of the documentation, or here more lightweight as an entry point,

[1] https://github.com/riehlegroup/msadoc

| Challenge | Feature |
|---|---|
| CC1: Manual documentation efforts | F3: Generation of higher-order documentation,<br>F7: Generation of documentation (OS community) |
| CC2: Wrong/out-of-date documentation | F1: Decentralized Documentation<br>F3: Generation of higher-order documentation |
| CC3: Incomplete documentation | F2: JSON files with schema |
| TC1: EAM tools not suitable | F0: Building a dedicated tool |
| TC2: No stakeholder-specific visualizations | F4: Stakeholder views |
| TC3: Lack of runtime KPIs | F8: API for runtime data (future work) |
| BC1: Lack of clarification of responsibilities | F5: Documentation of responsibilities |
| BC2: Missing alignment with suborganizations | F2: JSON files with schema |
| OC1: Bad discoverability of documentation | F1: Decentralized documentation |
| OC2: Variance of how teams do documentation | F1: Decentralized documentation |
| OC3: Technology heterogeneity | F2: JSON files with schema |
| OC4: Project-specfic Documentation | F6: Documentation Customization |

**Table 1: Mapping of Documentation Challenges to MSAdoc Features**

will support stakeholders by having a default place to search and browse the microservice documentation (*T3.4.1 Standardize location of microservice documentation* in Schwarz et al. [5]).

## 3.2 JSON Files for Configuration (F2)

The file format chosen for documentation is JSON[2]. JSON files for configuration are widespread and well-known among developers while being interoperable with HTTP APIs. Microservices allow for using heterogeneous technologies throughout the system since they are decoupled by APIs [6]. We chose JSON to keep the tool technology agnostic by using a well-designed interface and support technological heterogeneity (OC3)

Further, by specifying a *$schema* field, most editors support auto-completion and schema validation out of the box. Enforcing a certain standard ensures consistent application across all microservices (*T3.4.11 Standardization* in Schwarz et al. [5]). Here, the consistency and completeness of documentation are ensured (CC3), and sub-organizations are enforced to align with the enterprise architecture concepts (BC2).

## 3.3 Generation of Higher-Order Documentation (F3)

Next to meta-data, users can also specify provided and consumed API endpoints and events by the fields *providedAPIs*, *consumedAPIs*, *publishedEvents*, and *subscribedEvents*. The MSAdoc server can link consumed and provided APIs and published and consumed events of each microservice to generate higher-order documentation in the form of architecture diagrams, for instance, an high-level component diagram as shown in Figure 2. By this mechanism, changes in specific microservices are propagated to the high-level documentation, ensuring it stays up-to-date (CC2) as long as the microservice-specific documentation is updated. Generating parts of the documentation requires less manual work to create and maintain documentation (CC1). Such API information can be maintained manually with the danger of getting out-of-sync or generated by community tools (see Section 3.7).

[2]https://www.json.org/json-en.html

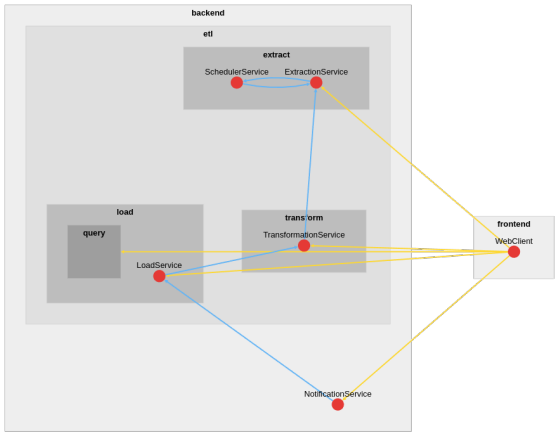



**Figure 2: Generated Architecture Diagram**

## 3.4 Stakeholder Views (F4)

Different stakeholders are interested in different views on the system (TC2). On the one hand, MSAdoc allows specifying a hierarchical *group* field that aggregates microservices into hierarchical groups, e.g., bounded contexts. The generated higher-order documentation (see Section 3.3) allows displaying aggregated groups instead of their internal details (the microservices) to facilitate a birds-eye view over the system and incrementally going deeper level by level.

On the other hand, MSAdoc allows filtering services according to certain criteria, showing only a subset of microservices and groups of services of interest.

## 3.5 Documentation of Responsibilities (F5)

To understand the relationship between microservices and responsible individuals or teams, we capture this information in the fields *responsibles* and *responsibleTeam* (*T3.4.2 Responsibility documentation* in Schwarz et al. [5]) to enable reaching the right stakeholders (BC1). Next to showing this information for each microservice, MSAdoc allows users to browse from teams to their microservices as well.

```
{
  // F2: Schema for auto-completion
  // and validation
  "$schema": "raw.githubusercontent.com/riehlegrou
p/msadoc/main/schemas/service-doc.json",

  // F1: Basic meta information
  "name": "NotificationService",

  // F1: Links to project management resources
  "repository": "github.com/jvalue/ods.git",
  "taskBoard": "github.com/...",

  // F1: Links to detailed documentation
  "developmentDocumentation": "github.com/...",
  "apiDocumentation": "github.com/...",

  // F4: Hierarchical grouping allows
  // different stakeholder views
  "group": "backend.notification"

   // F3: API endpoints, e.g., HTTP endpoints
  "providedAPIs": [
    "/notifications/configs",
    "/notifications/execution-stats"
  ],

  // F3: Events, e.g. RabbitMQ topics
  "publishedEvents": [
    "notification.config.created",
    "notification.config.updated",
    "notification.config.deleted",
  ],
  "subscribedEvents": [
    "load.execution.success",
    "load.execution.failure"
  ],

  // F5: Documentation of responsible people
  // and/or teams
  "responsibles": ["john@doe.org"],
  "responsibleTeam": "notifications",


  // F6: Tags for advanced filtering
  // and search (customization)
  "tags": ["notifications", "backend"],

  // F6: Extensions for customization
  // of documentation
  "extensions": {
    "usedInProducts": ["ProductA", "ProductB"]
  }
}
```

**Listing 1: Example of a msadoc.json file**

### 3.6 Documentation Customization (F6)

To accommodate variations in what projects document (OC4), MSAdoc offers two mechanisms. With the *tags* field, labels can be assigned to microservices, allowing for easy filtering in the aggregated view. Further, the *extensions* field enables users to document further information in a structured way, for example, in which products a microservice is included.

### 3.7 Community Work: Generation of Documentation (F7)

By choosing the technology-agnostic intermediate format of a JSON file (F2), MSAdoc is open for other tools coming from the Open Source community to automatically derive documentation from code to reduce manual documentation efforts (CC1). As a positive side-effect, documentation is less likely to become outdated if generated. We see special merit in tools that utilize API specification languages like OpenAPI[3] or AsyncAPI[4] to generate parts of the *msadoc.json* file.

### 3.8 Future Work: API for runtime data (F8)

At the time of writing, MSAdoc does not support the enrichment of static documentation data with runtime data (TC3). We cannot infer the infrastructure a microservice architecture is running on as we try to keep the tool technology agnostic (OC3). Instead, we plan to provide an API for an agent running on the deployment platform that periodically announces all running microservice instances, e.g., containers, and maps the running instances to their microservices. This way, we can support multiple deployment platforms like cloud providers or Kubernetes while the MSAdoc server stays agnostic to them.

## 4 Conclusion

The documentation of microservices and their integration has many challenges. Among others, microservice documentation can become out-of-date quickly and is distributed over multiple places, making it difficult to find necessary information.

We introduced an open source tool called MSAdoc to approach these challenges by (1) facilitating decentralized documentation close to the source code, (2) aggregating documentation centrally across individual microservices to generate higher-order documentation, and (3) supporting technological heterogeneity by relying on the technology-agnostic JSON format for documentation.

## Availability of Resources

The **open source tool**: https://github.com/riehlegroup/msadoc
The **demonstration video**: https://youtu.be/aUMS5ClehMo
The **pre-configured demo**: https://riehlegroup.github.io/msadoc

## Acknowledgments

We gratefully acknowledge Martin Buchalik for their valuable assistance in developing and testing the tool.

## References

[1] Pooyan Jamshidi, Claus Pahl, Nabor C Mendonça, James Lewis, and Stefan Tilkov. 2018. Microservices: The journey so far and challenges ahead. *IEEE Software* 35, 3 (2018), 24–35.
[2] Yujuan Jiang and Bram Adams. 2015. Co-evolution of infrastructure and source code-an empirical study. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 45–55.
[3] Martin Kleehaus and Florian Matthes. 2019. Challenges in documenting microservice-based IT landscape: A survey from an enterprise architecture management perspective. In *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*. IEEE, 11–20.
[4] Sam Newman. 2021. *Building microservices*. O'Reilly Media".
[5] Georg-Daniel Schwarz, Andreas Bauer, Dirk Riehle, and Nikolay Harutyunyan. 2024. A Taxonomy of Microservice Integration Techniques. Manuscript accepted but not yet published in *Information and Software Technology*.
[6] Georg-Daniel Schwarz, Philip Heltweg, and Dirk Riehle. 2024. Balancing Technology Heterogeneity in Microservice Architectures. Manuscript in submission to *Transactions on Software Engineering and Methodology*.

[3] https://www.openapis.org/
[4] https://www.asyncapi.com/en