Editor: **Cesare Pautasso**
University of Lugano
c.pautasso@ieee.org

Editor: **Olaf Zimmermann**
University of Applied Sciences
of Eastern Switzerland, Rapperswil
ozimmerm@hsr.ch

Emerging Trends, Challenges, and Experiences in DevOps and Microservice APIs

Uwe Zdun, Erik Wittern, and Philipp Leitner

From the Editors

The Vienna Software Seminar (VSS) has established itself as a place to be for those operating at the intersection between software research and practice: here, staff members from Internet firms, software vendors, and consultancies exchange thoughts and experiences with academics across the whole software engineering lifecycle. In this issue's "Insights" department, the VSS 2019 organizers report on program highlights and share findings from open space discussions.

—*Cesare Pautasso and Olaf Zimmermann*

IN AUGUST 2019, we organized the second Vienna Software Seminar (VSS) with the topic "DevOps and Microservice APIs."[1] Embracing the positive reception of its first iteration in 2017,[2] VSS is an opportunity for attendees to discuss recent software technologies, practices, and related research. The seminar's 34 participants included a mix of practitioners and academics, who were invited based on their roles and experiences. The explicit intention of the seminar was to provide ample opportunities for exchange and communication: six themed sessions consisted of one invited keynote and two lightning talks, giving different perspectives on the session's topic and (ideally) sparking ideas for follow-up discussions. After the talks, all participants decided on subtopics for two to three breakout sessions (i.e., informal, self-organized discussions among interested participants). Breakout session topics included microservice security, tooling for application programming interface (API) evolution, serverless programming models, and identification of microservices using domain-driven design. The sessions provided opportunities for detailed discussions and identifying challenges to address in future collaborations. Toward the end of each session, all participants gathered once more to summarize the breakout discussions. Additional opportunities for communication were provided during shared lunch breaks and social events in the evenings.

Focal topics that emerged in this year's iteration of the seminar were how to identify, design, and evolve (micro)services; how to manage service APIs and API ecosystems; how to implement services (for instance, using novel techniques, such as serverless); and how to operate services in a DevOps style. In this article,

Digital Object Identifier 10.1109/MS.2019.2947982
Date of current version: 24 December 2019

we report on emerging trends, challenges, and experiences in DevOps and microservice APIs as they were identified and discussed during the seminar. A graphical overview is provided in Figure 1.

Service Identification and Design

A question that many seminar participants identified as crucial is how to identify constituent services and APIs in a microservice-based application. Conventional wisdom indicates that a good practice is to start with a monolith, and when an actual need arises, start cutting out individual services from this monolith.[3] However, a monolith that has grown naturally (i.e., without explicit planning for future microservice migration) may often require substantial architectural redesign before nontrivial services can be cut out. Consequently, some participants reported on experiences with a structured monolith approach: building a monolithic application in a way to explicitly ease later microservice migration. Unfortunately, experience has shown that such a structured monolithic approach requires considerable upfront architectural investment at a time when it is still unclear if this investment will ever pay off. For example, foreseeing implications on reliability, performance, or security once components start communicating over a network is hard and will eventually still require significant refactoring. Hence, it remains challenging for practitioners to actually design a system with (potential) future microservice migration in mind.

As for how to actually design services, the domain-driven design (DDD) approach received a lot of attention during the seminar. Context Mapper[4] and MDSL[5] were demonstrated as prototypical domain-specific languages (DSLs) for modeling bounded contexts, services, and APIs. The two intertwined DSLs support code generation of service stubs after the domain has been modeled successfully. Service decomposition criteria were discussed as well; in addition to well-known software qualities, for example, consistency, availability, and recoverability, and design principles, including high cohesion within and low coupling between services, economic forces such as costs were identified as key decision drivers during service decomposition. Both development efforts and operational expenses

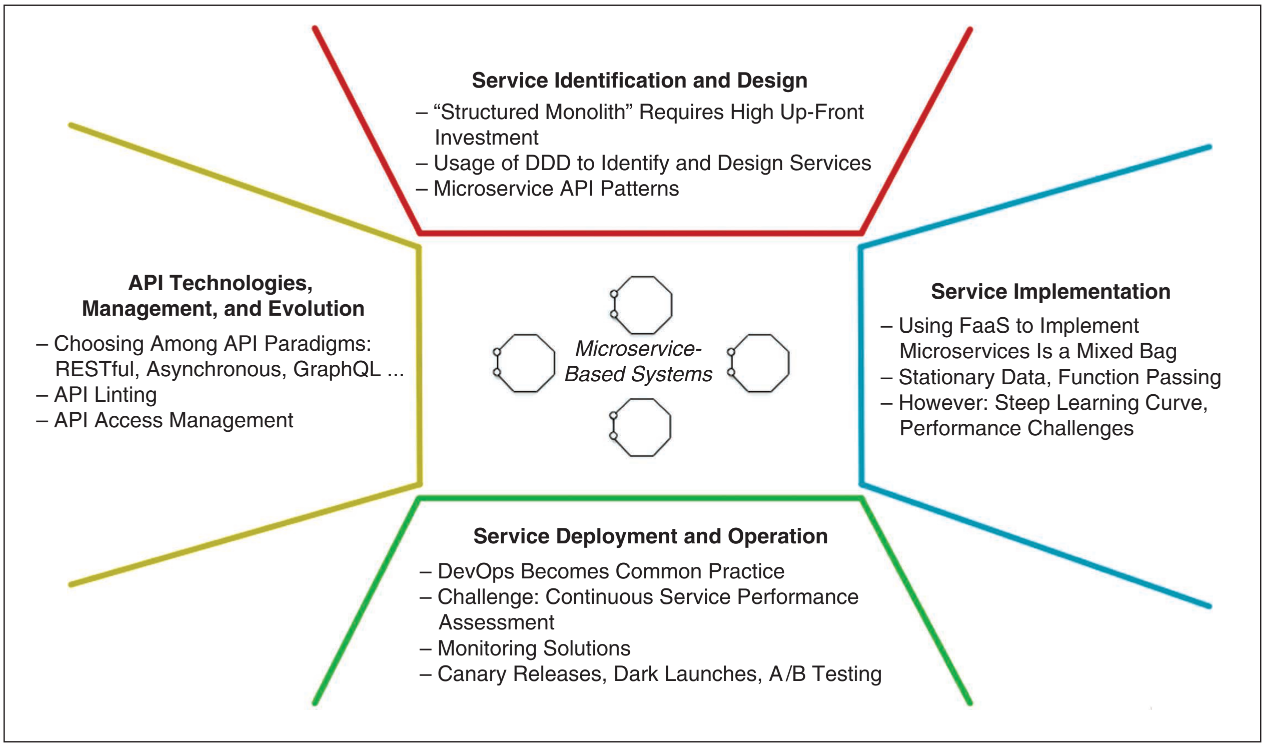
Service Identification and Design
– "Structured Monolith" Requires High Up-Front Investment
– Usage of DDD to Identify and Design Services
– Microservice API Patterns
API Technologies, Management, and Evolution
– Choosing Among API Paradigms: RESTful, Asynchronous, GraphQL ...
– API Linting
– API Access Management
Microservice-Based Systems
Service Implementation
– Using FaaS to Implement Microservices Is a Mixed Bag
– Stationary Data, Function Passing
– However: Steep Learning Curve, Performance Challenges
Service Deployment and Operation
– DevOps Becomes Common Practice
– Challenge: Continuous Service Performance Assessment
– Monitoring Solutions
– Canary Releases, Dark Launches, A/B Testing

FIGURE 1. The emerging challenges of microservice API development and operations. REST: Representational State Transfer; DDD: domain-driven design; FaaS: function-as-a-service.

(i.e., CPU workload and resulting cloud resource fees) add to these costs. Complementing such tools and experiences, we also discussed an emerging catalog of microservice API patterns.[6] This catalog collects knowledge and best practices around a variety of recurring API-related problems (e.g., how to represent complex parameters, how to handle pagination and API keys, or how to best rate limit an API and evolve it in client- and provider-friendly ways) and is particularly useful to teach common knowledge to new services designers or students. Although the catalog is still being actively developed, participant feedback suggests that it already contains a host of useful information.

API Technologies, Management, and Evolution

One focus area of this year's VSS was APIs, as already indicated in the seminar's title. Building on the standard HTTP protocol, publicly accessible resource APIs emerged, enabling the creation of ecosystems of third-party applications. The large number of public APIs attracts empirical research efforts. Recent studies analyze large numbers of APIs, often not by assessing their implementation (which is inaccessible to researchers in most cases) but by analyzing API-related network traffic,[7] developer documentation,[8] or even exposed pricing plans and related business models.[9] Rich sources for further research, for example, making use of interactive visualizations, lie in analyzing API specifications, which are made accessible in user-maintained repositories like APIs.guru.[10] For instance, the previously mentioned microservice API patterns were empirically mined from public and nonpublic APIs.[11] Empirical works describe the current state of APIs and identify commonly used practices. Furthermore, observations from such studies form the basis for prescribing best practices,[6] point to possible pitfalls, and motivate the creation of new development approaches and tooling, or the evolution of standards.

Within large systems, internal APIs have long played a major role to enable network-based communication between distributed components. In the early 2000s, web services using Web Services Description Language and SOAP were among the many technologies used to implement such APIs. In recent years, increasingly, Representational State Transfer [i.e., REST(ful) or REST-like] APIs gained importance as well as new takes on Remote Procedure Call (RPC)-style APIs and asynchronous API technologies.

The seminar participants discussed a number of new API-related technologies, which reflects the increased importance of APIs. GraphQL, e.g., allows clients to request data from servers using statically typed queries, which reduces overfetching and the number of network round trips. By adopting queries, clients can fetch different data without requiring the server to change. Binary serialization formats like protocol buffers optimize message sizes and introduce static typing to messages. Middleware frameworks like gRPC and their extension libraries or plug-ins build on these formats to provide or enable the realization of higher-level functionalities, for instance, message tracing, load balancing, or streaming capabilities. Where REST and REST-like APIs often implement synchronous request–response patterns, event-driven or reactive architectures rely on asynchronous APIs, where clients, e.g., subscribe to topics and then continuously receive messages for these topics. Asynchronous communication protocols, i.e., MQTT, Apache Kafka, and Advanced Message Queuing Protocol, enable more loosely coupled connections between clients and servers, but as the participants acknowledged, require rethinking familiar synchronous development practices. API specification approaches specific for the various API implementation options, for example, OpenAPI[12] for HTTP-based APIs or AsyncAPI[13] for asynchronous APIs, enable better API documentation and thus ease API management.

A question that many seminar participants identified as crucial is how to identify constituent services and APIs in a microservice-based application.

Despite these developments, or maybe as a result of them, many challenges remain in the API space. Many challenges are both technological and organizational in nature. As the seminar participants discussed, organizations struggle to be consistent in how they design APIs, and they require mechanisms to make internal APIs easier to find and reuse. An API linter that checks the adherence of APIs to an organization's design principles came up as an interesting future research topic. API evolution, especially when it comes to breaking changes, requires careful consideration and involves challenges like efficient communication with and support for affected clients. This is especially challenging when considering Hyrum's law, which states that, given sufficient users, all observable behaviors of your system will be depended on by somebody. Access to APIs must be managed (especially for public APIs) to secure private data but also to prevent inadvertent misuse or even malicious attacks. Researchers have identified challenges in consuming APIs, including dealing with varying quality of service, the fact that APIs may sunset and eventually cease to exist, or lack of support for refactoring client code.[14]

Service Implementation

For actually implementing services, the discussion in the seminar centered around the ideas of microservices (as discussed previously) as well as serverless and function-as-a-service (FaaS) clouds. In FaaS, developers build systems as a collection of small, stateless functions, often written in lightweight scripting languages, for instance, JavaScript or Python. Individual functions are triggered by infrastructure events, such as incoming HTTP requests or messages in a message queue. Functions are registered with the cloud provider, and function management, routing, and scaling is handled transparently by the cloud.

The usage of serverless functions to implement APIs was conceived to be a mixed bag in the seminar. They lift operational concerns but at the same time introduce new challenges, i.e., overcoming a steep learning curve, complicating testing, hard-to-predict operating costs, or limited application portability.[15] However, we also discussed upcoming tools, for example, the Nimbus framework,[16] which is designed to address these concerns by providing an abstraction to deploy code across serverless providers and enabling local testing. Furthermore, serverless functions provide opportunities for new programming models, including function passing for fault-tolerant distributed programming using stationary data through serializable, passed functions.

Service Deployment and Operation

In addition to questions of APIs and software development, the seminar raised the issue that the proliferation of microservices and APIs also has implications on operations teams. DevOps is widely understood as an industry standard for API-operating companies, but even in a DevOps team, the frequent release of small, independent services may pose challenges. In particular, various seminar participants raised questions related to noise in continuous integration (CI) builds (e.g., related to flaky tests) and managing nonfunctional service properties (e.g., service performance or stability of API contracts). How, and whether, service performance can be assessed prior to actual deployment (when middleware, such as Istio,[17] can be used to implement canary releases, dark launches, or A/B testing) was discussed extensively. All seminar participants agreed that continuously testing performance (e.g., as part of the CI pipeline) is challenging, if not impossible, for most Internet-scale services. However, for some classes of applications (e.g., middleware, storage solutions, or libraries), the usage of continuous microbenchmarking may make sense. Ultimately, service quality management requires a holistic approach that integrates basic sanity checks within the build system combined with postdeployment techniques, for example, canary releases or chaos engineering, meaning the deliberate introduction of failures to test the resilience of an application.

The field of DevOps and microservice APIs continues to develop a plethora of novel concepts, technologies, and trends in rapid succession and continues to provide exciting new developments that enable many of the major software impacts we can observe today. Each and every one of them promises interesting prospects for the future but at the same time leads to major adoption challenges. Many of the proposed solutions and their corresponding challenges are, as has been observed for software generally,[18] new incarnations of existing best practices, usually with a substantial novel twist, leading to new concepts and technology developments that render the prior generation inferior. Foundational and empirical research is required to carve out the best practices and expected impact on quality attributes at a conceptual level, so that the research results will stand the test of time while new concepts and technologies keep emerging. Such timeless knowledge can train inexperienced DevOps and microservice API developers more effectively and more sustainably than any ephemeral technology incarnation. At the same time, novel approaches have to be developed to cope with the specific challenges that new contexts pose, i.e., the serverless implementation option or the massive scale of cloud applications. Thus, a continuing exchange between researchers (both from academia and companies) and innovative practitioners is required to let research and innovations follow up on the current industrial needs, which are rapidly evolving in bustling fields like DevOps and microservice APIs.

References

1. DevOps and Microservice APIs, 2nd Vienna Software Seminar. Accessed on: Oct. 1, 2019. [Online]. Available: https://vss.swa.univie.ac.at/2019/
2. On the Relation of Software Architecture and DevOps/Continuous Delivery, 1st Vienna Software Seminar. Accessed on: Oct. 1, 2019. [Online]. Available: https://vss.swa.univie.ac.at/2017/
3. M. Fowler, "MonolithFirst," June, 2015. [Online]. Available: https://martinfowler.com/bliki/MonolithFirst.html
4. Context Mapper, "A DSL for context mapping & service decomposition," 2019. [Online]. Available: https://contextmapper.org/
5. GitHub, "MDSL," 2019. [Online]. Available: https://socadk.github.io/MDSL/index
6. Microservice API Patterns. Accessed on: Oct. 1, 2019. [Online]. Available: https://microservice-api-patterns.org
7. C. Rodríguez et al., "REST APIs: A large-scale analysis of compliance with principles and best practices," in *Proc. Int. Conf. Web* Engineering, 2016, pp. 21–39.
8. A. Neumann, N. Laranjeiro, and J. Bernardino, "An analysis of public REST web service APIs," *IEEE Trans. Serv. Comput.*, to be published. doi: 10.1109/TSC.2018.2847344.
9. A. Gamez-Diaz, P. Fernandez, and A. Ruiz-Cortes, "An analysis of RESTful APIs offerings in the industry," in *Proc. Int. Conf. Service-Oriented Computing*, 2017, pp. 589–604.
10. GitHub, "APIs-guru/openapi-directory," 2019. [Online]. Available: https://github.com/APIs-guru/openapi-directory
11. U. Zdun, M. Stocker, O. Zimmermann, C. Pautasso, and D. Lübke. "Guiding architectural decision making on quality aspects of microservice APIs," in *Proc. 16th Int. Conf. Service-Oriented Computing (ICSOC 2018)*, 2018, pp. 73–89.
12. Swagger, "What is OpenAPI?" 2019. [Online]. Available: https://swagger.io/docs/specification/about/
13. AsyncAPI, "Building the future of event-driven architectures," 2019. [Online]. Available: https://www.asyncapi.com/
14. E. Wittern et al., "Opportunities in software engineering research for web API consumption," in *Proc. 1st Int. Workshop API Usage and Evolution*, 2017, pp. 7–10.
15. P. Leitner, E. Wittern, J. Spillner, and W. Hummer, "A mixed-method empirical study of function-as-a-service software development in industrial practice," *J. Syst. Software*, vol. 149, pp. 340–359, Mar. 2019.
16. Nimbus. Accessed on: Oct. 1, 2019. [Online]. Available: https://www.nimbusframework.com
17. Istio. Accessed on: Oct. 1, 2019. [Online]. Available: https://istio.io
18. Z. Obrenovic, "Insights from the past: The IEEE software history experiment," *IEEE Softw.*, vol. 34, no. 4, pp. 71–78, 2017.

ABOUT THE AUTHORS

UWE ZDUN is a full professor of software architecture at the University of Vienna, Faculty of Computer Science, Vienna, Austria. Contact him at uwe.zdun@univie.ac.at.

ERIK WITTERN is a GraphQL lead architect at IBM, Hamburg, Germany. Contact him at Erik.Wittern@ibm.com.

PHILIPP LEITNER is an assistant professor of software engineering at Chalmers University of Technology and the University of Gothenburg. Contact him at philipp.leitner@chalmers.se.