A Dashboard for Microservice Monitoring and Management

Benjamin Mayer and Rainer Weinreich
Johannes Kepler University Linz
Linz, Austria
{benjamin.mayer, rainer.weinreich}@jku.at

***Abstract*—We present an experimental dashboard for microservice monitoring and management. The dashboard can be adapted to different stakeholder needs and it supports the integration of different monitoring infrastructures for collecting microservice runtime data. Aside from runtime information, the dashboard also supports the integration of other information sources for providing static information about microservices and microservice development. We describe the main motivation for developing the dashboard, explain the basic concepts and present important usage scenarios and views currently supported in the dashboard.**

Keywords—Microservices, microservice monitoring, micro–service management, microservice dashboard.

I. Introduction

Microservices are an architectural style largely based on decoupled autonomous services that can be developed, deployed and operated independently of each other [1][2]. Microservices lead to several challenges regarding team organization, development practices and infrastructure [1][3]. In this paper, we highlight two of these challenges, which require support for microservice management and monitoring.

The first is the close relationship between software architecture and team organization. While the relationship between software architecture and team organization is well known (see Conway's law [4]), it is especially manifest in microservice architecture. In microservice architecture, teams operate as independently of each other as possible. One team is usually responsible for the development, deployment and (using DevOps [5]) operation and failure handling of a particular service. This autonomy goes so far that different teams may even select different implementation technologies for different microservices, depending on which best suits the task at hand.

The second is the distributed nature and granularity of microservices. A microservice-based system may consist of a large number of independently developed and deployed services, which interact at runtime. Due to this independent nature of the individual services and service development, the interaction between services and thus the overall system architecture only become evident at runtime. Since services can be independently and continuously deployed and changed over time, the system architecture is also continuously changing and evolving. Thus, information on the system architecture and on service interaction and behavior need to be generated and updated automatically.

The independent nature of service development and evolution in a microservice-based system and its nature as a highly dynamic distributed system require infrastructure support in terms of monitoring and managing such services. Changes to team organization in development approaches such as DevOps additionally require information that can be tailored to a particular stakeholder, be it a developer, tester, operator or any of these in combination.

In this paper, we present an experimental dashboard for microservice monitoring and management, which can provide such information based on stakeholders' needs and which can integrate different backend systems providing both runtime and development information about microservices and microservice-based systems.

This paper is structured as follows: In Section II, we list the requirements for the dashboard we obtained through a combined survey and interview study. In Section III, we describe the basic concepts of the dashboard. In Section IV, we present use cases and views that are supported in the dashboard. Related work is described in Section V. A conclusion is presented in Section VI.

II. Requirements

To obtain the requirements for the dashboard, we conducted a combined survey and interview study on important information for microservice monitoring and management. We received feedback from 15 architects, developers and operations experts from 12 companies in Germany and Austria. About half of the study participants (8) provided information through an online questionnaire. The other half (7) were interviewed either face-to-face or online through Skype based on the same questionnaire. Company domains include financial services, media services, e-commerce, and consulting, with company size ranging between 200 and several thousand employees.

Table 1 shows the questions we asked about the information required for microservice monitoring and management. The study participants considered the following information to be most important:

- Service API
- Service version
- Number of service instances
- System metrics such as CPU and memory consumption response time, workload and the error rate of a particular service
- Service interaction and service dependencies

978-1-5090-4793-2/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSAW.2017.44

TABLE I. QUESTIONS ABOUT THE REQUIRED INFORMATION

Static information
1. Developers, operations experts and other persons responsible for specific microservices
2. Service version
3. Service API
4. Differences between the APIs of different versions
5. Differences between the APIs in different versions regarding the request parameters and response objects
6. Units for grouping services (i.e., ordering process)
7. Used technology stacks (e.g., programming language, frameworks)
Dynamic information
8. System layers where runtime information should be provided (options: service, version, service instance, host)
9. Status of service (alive)
10. Workload of service
11. Relationships between services (call dependencies, APIs used)
12. Request count for specific APIs over time
13. Request count for specific versions over time
14. Response time of specific APIs over time
15. Response time of specific versions over time
16. Composition of response time over a chain of requests
17. Number of successful and failed requests per service
18. Number of specific kinds of failures (e.g., unavailability)
Infrastructure-related information
19. Number of running service instances
20. Hosts where service instances are running
21. Datacenter where service instances are running
22. CPU and memory consumption of hosts

API documentation for microservices was considered to be particularly important for developers. Versioning information was considered to be important for identifying the exact service version causing problems and determining service compatibility. Infrastructure-related information (e.g., number of running service instances) and metrics (e.g., CPU load and memory consumption) were considered to be important for operations to improve the deployment process and detect infrastructure problems. Runtime information on a particular service was mentioned as important for detecting problems. Service interaction at runtime was considered to be important for root cause analysis, since a problem in one microservice can often be traced back to a problem in another service.

We use the use cases identified as important in the survey as the main requirements for the microservice dashboard. Additional requirements were the possibility to adapt the dashboard to different stakeholders and the integration of different information sources for both static and dynamic service information.

III. BASIC CONCEPTS

Dashboards usually provide key metrics on a particular system in one unified view. In our case, the dashboard provides both static and runtime information about a microservice-based software system (see Table 2).

Central concepts that are part of our dashboard are stakeholders, views and visualization components. The different stakeholders of a microservice-based system need different views on the system. For example, developers may be more interested in static system information and the runtime behavior of microservices, while operators may be more interested in resource utilization. Because of developments such as DevOps [5], it should be possible to tailor the information to a particular stakeholder's needs.

To support this required level of customization, we provide different visualization components for the aspects of a microservice-based system. These visualization components can be combined to create stakeholder-specific views, which we call dashboard views (or short dashboards) in our system. Multiple dashboard views to address different concerns and information needs can be created for each stakeholder. Within a dashboard view, different visualization components can be used, which can be individually arranged and resized.

These different visualization components may use different forms of visualization such as tables, lists, line charts and bar charts. Visualization components have been created for each of the requirements identified in Section II. We classified them into visualization components providing mostly static information and visualization components providing runtime information. All in all, we created more than 50 types of visualization components. Table 2 provides an overview of the available kinds of visualization components.

TABLE II. VISUALIZATION COMPONENTS

Static information
• Pie chart comparing the amount of commits per developer
• Table showing all issues for a specific service
• Table showing the latest commits for a service
• Table showing the API provided by a service including HTTP endpoint and description
• Table comparing the provided APIs of two versions of a service
• Table showing an overview of the available services including name, version, contact information and running service instances
• Graphs for visualizing an overview of a system including services, nodes and allocation of services to nodes
Dynamic information
• Line charts and bar charts for displaying service-related runtime information over time, including response time, request count, throughput, failure rate, Apdex score, etc.
• Line charts, bar charts and pie charts for comparing service-related runtime information on different services and different service versions
• Graphs for visualizing service interactions including the frequency of communication between services, failure rate and response time
• Radar chart for comparing the different metrics of different services
• Different charts for showing resource utilization (e.g., CPU performance over time and memory usage)

Each of the visualization components may retrieve its data from different backend systems (using REST-APIs), providing either static or runtime monitoring data. Data is transformed into the format required by the dashboard components using a data adapter service, which can be configured dynamically with a data transformation script. This script is stored as part of the information describing each backend in the dashboard configuration database. Currently, we support *GitHub* [6] as the backend for static and development information, *Dynatrace APM* [7] for providing service runtime and infrastructure-related information and our own custom Spring-based microservice monitoring and management infrastructure for providing both. Since integration is based on REST-APIs and a generic data transformation service, the integration of additional information sources is possible with minimal effort. We will add additional backend systems in the future depending on project partner needs.

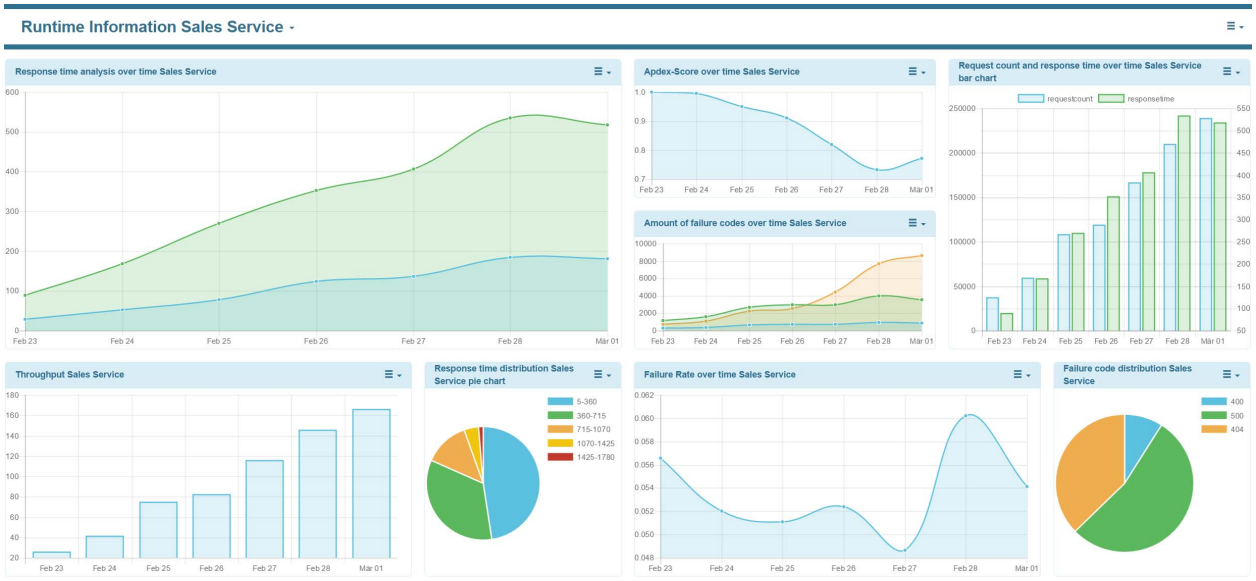

Fig. 2. Runtime information of one microservice

Dashboard views defined by a particular stakeholder can be shared with other stakeholders. In addition, the provided visualization components of the implemented system are parameterizable. This allows us to change the properties of all visualization components within one view. By using this mechanism, various components showing different aspects for a particular service can be easily switched to another service. Similarly, the time interval used to analyze particular information can be changed for all visualization components within a particular dashboard view.

IV. DASHBOARD VIEWS

In the following, we provide some examples of dashboard views that can be configured for particular stakeholders and use cases. In particular, we describe dashboard views for providing a system overview, for providing runtime information for a particular service, for providing development-related information for a particular service, for comparing different services and service versions, and for visualizing service interactions. The described dashboard views are also available in a demo version of the dashboard available online [8].

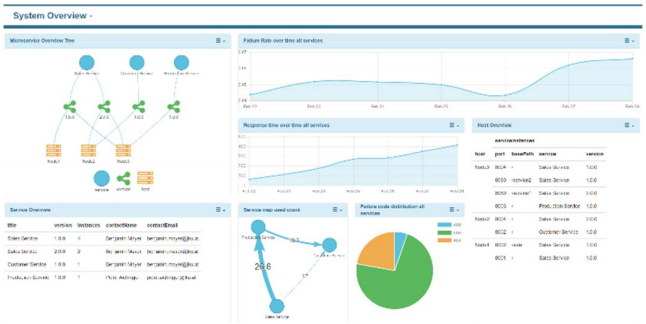

Fig. 1. System overview

The dashboard shown in Fig. 1 provides an overview of the system. It contains a graph showing how services and service versions are allocated to system nodes (top left), charts showing the failure rate and response time of all services over time (top right) and a graph showing service interaction (center bottom) among other information.

The dashboard in Fig. 2 provides runtime information for a specific service (in this case, a *SalesService*) including response time, Apdex score, failure rate and throughput. For example, the chart at the top left in Fig. 1 compares the *SalesService* response time with the response time of the outgoing requests of the *service*. This allows us to analyze how external services contribute to the response time of a particular service.

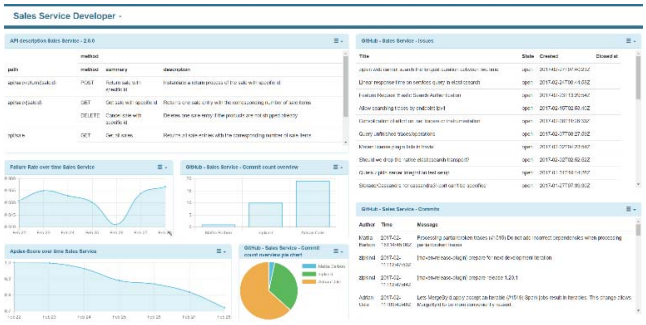

Fig. 3. Developer view for one microservice

The dashboard in Fig. 3 has been customized for service developers. It combines static and dynamic information on service development and runtime behavior and contains visualization components showing the service API (top left), GitHub issues and commits (top and bottom right) and runtime information such as failure rate and Apdex score (bottom left).

The dashboard in Fig. 4 has been configured to display information on service interaction. It shows the API of a service and graphs visualizing the interactions with other services. The interaction graphs at the top and bottom right contain weighted runtime relations in terms of contribution to failure rate, response time and communication frequency. This kind of information is useful for determining the root causes of failures

or high response times. The chart at the bottom left additionally shows the API calls during service interaction, which helps examine service interaction in more detail.

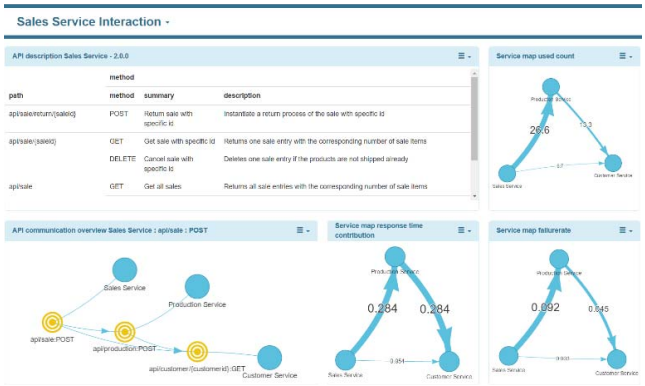

Fig. 4. Service interaction

Fig. 5 shows a dashboard comparing runtime information on two versions of a particular service. Along with the differences between runtime metrics over time, it also shows the API differences between different versions. This can be used to resolve compatibility issues during system evolution.

Fig. 5. Comparison of different versions of one service

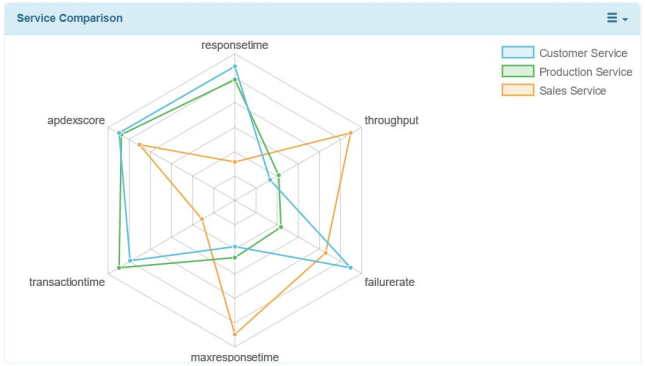

Fig. 6. Comparison of different services

We also provide a dashboard for comparing different services over time. it contains similar visualization components to those shown in Fig. 5 for comparing runtime metrics such as response time and the throughput of different services over time. This makes it possible to detect lower performing services as well as analyze correlations in service runtime behavior. Fig. 6 shows a visualization component, which is also part of this dashboard and which shows the differences between three services with respect to different runtime metrics at a glance.

Finally, we also provide runtime information on the utilization of resources such as CPU and memory in the service runtime environment (see [8]).

V. RELATED WORK

Systems providing similar features to what is presented in this paper are *Dynatrace APM (*dynatrace.com*)*, *NewRelic* (newrelic.com) and *Zipkin* (zipkin.io). Main differences when compared to our approach are that these systems typically focus on short term monitoring (i.e., on fault detection), on runtime information, and on providing one central environment for all monitoring activities, from data collection to presentation. The dashboard presented in this paper aims to support both management and monitoring by combining development and runtime information and by supporting adaptation to different stakeholder needs through configuration. Our approach also supports the integration of different backend systems and thus can be used alongside existing monitoring solutions, since frontend and backend are less coupled than in the aforementioned solutions.

VI. CONCLUSION AND FUTURE WORK

The current dashboard is an experimental system for exploring ideas in microservice monitoring and management. We are especially interested in the documentation aspect of the system and on combining information from different information sources to support different stakeholder needs. The dashboard can be used alongside already existing monitoring environments and other information sources and its views can be tailored to specific stakeholder needs. We are interested in both scientific and commercial collaboration for using or extending the dashboard. We are also looking into integrating approaches to architecture documentation, specifically architecture knowledge management, to enable users of the dashboard to provide the rationale for the observed behavior and encourage them to provide documentation. A demo version of the described system is available at [8]. This provides the described dashboards and additionally allows a user to define and adapt his own dashboards.

Acknowledgment: We want to thank Peter Aichinger for contributing to the survey and the implementation of the dashboard.

REFERENCES

[1] E. Wolff, *Microservices: Flexible Software Architecture*. Addison-Wesley Professional, 2016.
[2] J. Lewis and M. Fowler, “Microservices,” 25-Mar-2014. [Online]. Available: https://martinfowler.com/articles/microservices.html.
[3] S. Newman, *Building Microservices*. O’Reilly Media, Inc., 2015.
[4] M. E. Conway, “How do committees invent,” *Datamat. J.*, vol. 14, no. 4, pp. 28–31, 1968.
[5] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect’s Perspective*. Addison-Wesley Professional, 2015.
[6] “GitHub.” [Online]. Available: https://github.com/. [Accessed: 28-Feb-2017].
[7] “Digital Performance & Application Performance Monitoring | Dynatrace.” [Online]. Available: https://www.dynatrace.com/. [Accessed: 28-Feb-2017].
[8] “Microservice Dashboard | WIN-SE.” [Online]. Available: https://www.se.jku.at/microservice-dashboard/. [Accessed: 28-Feb-2017]. username: demo@msm.se.jku.at, pw: mKxKX8WeEfgg