Applying a Technical Reference Architecture to Implement a Microservices-based Insurance Application

Arne Koschel*
Andreas Hausotter*
*Hochschule Hannover
University of Applied Sciences & Arts Hannover
Faculty IV, Department of Computer Science
Hanover, Germany
Email: arne.koschel@hs-hannover.de
andreas.hausotter@hs-hannover.de
christin.schulze@stud.hs-hannover.de

Henrik Meyer†
Christin Schulze*
†Capgemini
Hanover, Germany
Email: henrik.meyer@capgemini.com

***Abstract*—To overcome the shortcomings of traditional monolithic applications, the Microservices Architecture (MSA) style is playing an increasingly vital role in providing business services. This also applies to the insurance industry, which is facing challenges like cut-throat competition and decreasing customer loyalty. Providing scalable and resilient services of high availability in a flexible and agile manner, which comes with the MSA style, is undoubtedly a competitive advantage. However, the insurance industry's application landscape is characterized by the coexistence of historically grown systems based on different architectural paradigms. Therefore, the integration of microservices with Service-Oriented Architecture (SOA) services or even legacy systems induces additional complexity. A reference architecture may lower the complexity of this integration task by defining an architectural framework of MSA-based applications in a heterogeneous environment. In this contribution, we present a technical reference architecture for our partner insurance companies. The reference architecture is shaped along a cloud-native approach to provide good scalability, short release cycles, and high resilience. As a key feature, a technical microservice supports the integration of SOA services. To demonstrate the applicability of the technical reference architecture, it is used to implement a typical insurance business process in the context of car insurance. The target architecture comprises four business microservices and a SOA service managed by an ESB.**

***Keywords*—Microservices Architecture; Reference Architecture; Cloud Native; SOA; Insurance Industry.**

I. Introduction

In this paper, which is an extension of a previous paper [1], we look at implementing the technical reference architecture in the insurance industry via a cloud-native approach.

A long-lasting trend in software engineering is to divide software into lightweight, independently deployable components. Each component can be implemented using different technologies because they communicate over standardized interfaces. This approach to structuring the system is known as the MSA style [2]. A study from 2019 (see [3]) shows that the MSA style is already established in many industries, such as e-commerce. However, this is rarely the case for the insurance services industry.

Our current research is the most recent work of a long-standing, ongoing applied research–industry cooperation on service-based systems. This includes cooperative work on traditional SOA, Business Rules and Business Process Management (BRM/BPM), SOA-Quality of Service (SOA-QoS), and microservices [4]–[7], between the *Competence Center Information Technology and Management (CC_ITM)* from the University of Applied Sciences and Arts Hanover and two regional, medium-sized German insurance companies. The ultimate goal of our current research is to develop a 'Microservice Reference Architecture for Insurance Companies' (RaMicsV) jointly with our partner companies. This shall allow the building of microservice conformant insurance application systems or at least such system parts.

However, several cornerstones and resulting challenges exist frequently in the German industry domain for this purpose. Insurance companies rarely start development 'in the green field' but must integrate and comply with existing application systems, like Knoche and Hasselbring showed in [8]. This requirement stems from building complex systems over multiple years, or decades even, where a significant bang replacement is too risky and cost-intensive. For example, our partners both operate a SOA and additional 3rd party software, such as SAP systems, which both have significantly different characteristics, for example, for testing, release cycles, versioning, administration etc.

Nowadays, our partners would like to get the promised benefits of microservices, such as improved scalability, both technical and organizational, through parallel execution and also parallel development, significantly faster release cycles, (a few weeks or even days instead of quarters or several months) etc. However, a microservices-based approach to help them must still work well in 'cooperative existence' with their existing systems and SOA services. Thus, improvements or partial replacements of their existing software landscape for particular goals using microservices is fine. Still, a complete migration to the Microservices Architecture style is not a

feasible option. The main goal is to extend a system and not replace it.

In our previous work [9], we already developed RaMicsV, a logical reference architecture considering those insurance industry specifics. Moreover, we explored parts of it, such as logging, monitoring, security, workflows, and choreographies, in more depth (in [9]–[11]).

As the major contribution of this article, we present a technical reference architecture for RaMicsV. Our technical reference architecture (T-RaMicsV) is based on a cloud-native approach for microservices, including, for example, containerization, message-driven communication, and an ESB-wrapper microservice. In addition to the contents of our previous work [1], we also present the application of a typical insurance business process (Car Insurance Process), based on insurance business microservices within our technical reference architecture. In this implementation, we have selected and used a selection of technologies, such as Kubernetes for container orchestration, Apache Kafka for message-driven communication, and Keycloak for security-related topics. This is to be seen as one possible example of implementing our architectural considerations.

We organize the remainder of this article as follows: After discussing related work in Section II, we briefly repeat RaMicsV in Section III. Next, Section IV and Section V contribute our cloud-native approach to the technical reference architecture based on RaMicsV. Some implementation details of microservices within this architecture follow in Section VI. Section VII concludes and looks at future work.

II. Related Work

The foundations of this work relate to the concepts of Microservices Architecture, cloud computing, cloud-native architecture, and practical application within the insurance industry.

Our research builds on authors in the field of microservices, such as the work of Newman [12], as well as Fowler and Lewis [2], as well as Fachat [13]. When designing our reference architecture, we benefit from various microservices patterns discussed by Richardson [14].

To design T-RaMicsV, we use a cloud-native approach that is definitionally based on the descriptions of the Cloud Native Computing Foundation [15]. Furthermore, we supplement our understanding with the aspects of cloud computing introduced by the National Institute of Standards and Technology (*NIST*) [16], as well as containerization, automation, and observability. We also used works by S. Reinheimer to learn the basics of the cloud. [17]

To implement the approach in practice, we are using an exemplary insurance industry business process. For this purpose, we have chosen car insurance, one of the core products of German insurers. The authors [18] provide the basis for the process. Car insurance is compulsory for every car in Germany. For this reason, it is considered particularly important for acquiring new customers. The elaboration refers to the VAA [19] and describes in detail what car insurance is all about and much more. We use Business Process Model and Notation (BPMN) to realize the processes [20], as it is widely used in the insurance industry.

In this contribution, we lay out T-RaMicsV that is derived from RaMicsV [21] as our foundation. An implementation of RaMicsV that demonstrates its technical feasibility, as envisaged in this paper, has not yet taken place.

Regarding its realization, a technical reference architecture must be developed that makes fundamental statements about the technologies used, such as programming languages or infrastructure. In accordance to Angelov, Grefen, and Greefhorst [22] T-RaMicsV can be categorized as a 'semi-concrete type 4' reference architecture, i.e., indicating technology choices to be implemented in one single organization [22].

In this context, the technology-agnostic approach of microservices is broken with, to provide practical specifications that harmonize with the existing system landscape of an insurance company. The cloud-native approach used in this contribution is a conceivable option that seems promising to us to realize RaMicsV technically.

Deriving a technical reference architecture from a logical one, like RaMicsV, seems to be a common practice. Furthermore, there are contributions to reference architecture for microservices for broader enterprise context, e.g., by Yu, Silveira, and Sundaram in [23]. However, this has not yet been done for insurance companies such as our industry partners and their specific requirements.

These operate a historically grown heterogeneous system landscape characterized by an existing SOA. The use of microservices, that is embedded in the cloud-native approach, must be integrated in a cooperative manner, which plays an essential role in our overall architectural considerations.

III. Service-based Reference Architecture for Insurance Companies

This section presents our RaMicsV as initially started in [9].

RaMicsV defines the setting for the architecture and the design of a microservices-based application for our industry partners. The application's architecture will only be shown briefly, as it heavily depends on the specific functional requirements.

When designing RaMicsV, a wide range of restrictions and requirements given by the insurance company's IT management have to be considered. Regarding this contribution, the most relevant are:

- Enterprise Service Bus (ESB): The ESB as part of the SOA must not be questioned. It is part of a successfully operated SOA landscape, which seems suitable for our industry partners for several years. Thus, from their perspective, the Microservices Architecture (MSA) style is only appropriate as an additional enhancement and only

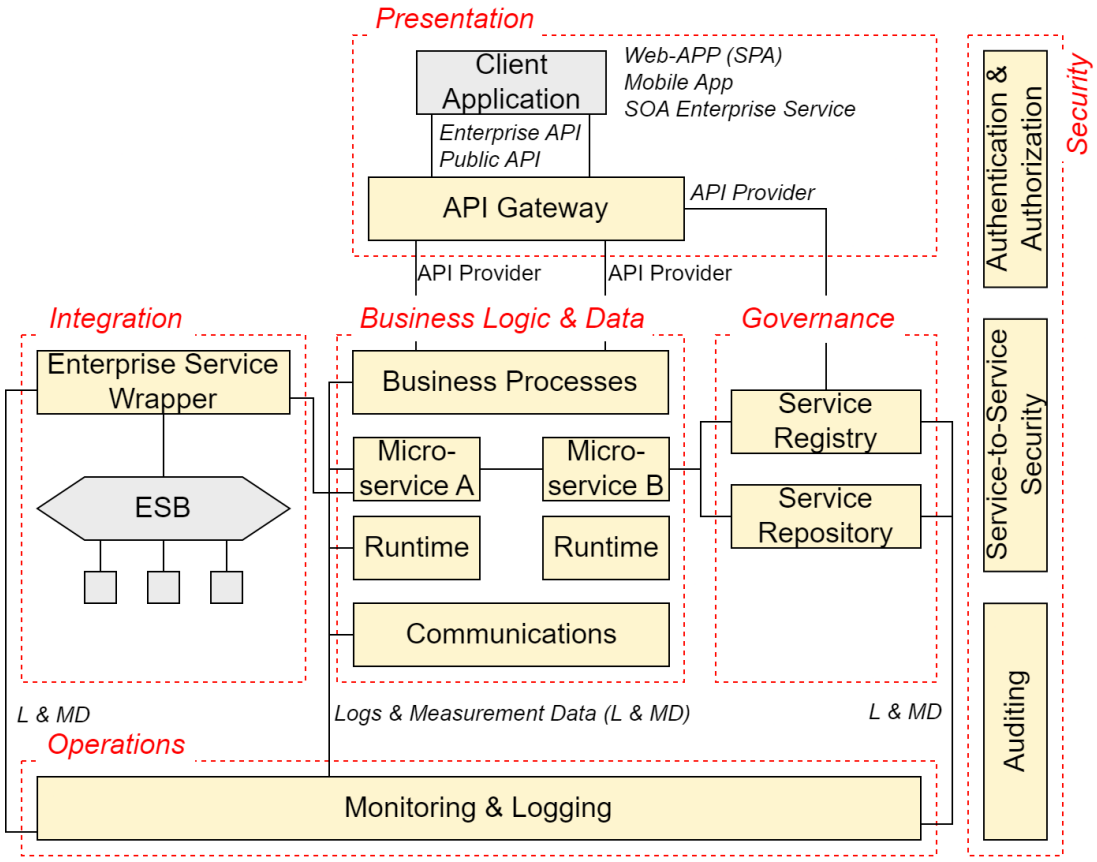

Figure 1. Building Blocks of the Logical Reference Architecture RaMicsV.

a partial replacement of parts from their SOA or other self-developed applications.

- Coexistence: Legacy applications, SOA, and microservices-based applications will be operated in parallel for an extended transition period. This means that RaMicsV must provide approaches for integrating applications from different architecture paradigms – looking at it from a high-level perspective, allowing an 'MSA style best-of-breed' approach at the enterprise architectural level as well.
- Business processes are critical elements in an insurance company's application landscape. To keep their competitive edge, the enterprise must change their processes flexibly and agilely. RaMicsV must, therefore, provide suitable solutions to implement workflows while ensuring the required flexibility and agility.

Figure 1 depicts the building blocks of RaMicsV, which comprises layers, components, interfaces, and communication relationships. Components of the reference architecture are colored yellow; those out of scope are greyed out.

A component may be assigned to one of the following *responsibility areas*:

- **Presentation** includes components for connecting clients and external applications, such as SOA services.
- **Business Logic & Data** deals with the implementation of an insurance company's processes and their mapping to microservices, using various workflow approaches to achieve desired application-specific behavior.
- **Governance** consists of components that contribute to meeting the IT governance requirements of our industrial partners.
- **Integration** contains system components to integrate microservices-based applications into the industrial partner's application landscape.
- **Operations** consist of system components to produce unified monitoring, logging, and tracing, which encloses all systems of the application landscape.
- **Security** consists of components to provide the goals of information security, i.e., confidentiality, integrity, availability, privacy, authenticity & trustworthiness, nonrepudiation, accountability, and auditability.

Components communicate either via HTTP(S) – using a RESTful API, or message-based – using a Message-Oriented Middleware (MOM) or the ESB. The ESB is part of the integration responsibility area, which itself contains a message broker (see Figure 1).

In the next section, we will present our understanding of cloud-native architecture, which represents our approach to developing T-RaMicsV as this paper's contribution.

IV. Cloud-native Architecture

The architectural approach of our contribution is based on the following five aspects of a cloud-native architecture.

- **Cloud Computing** means that an IT service is offered by a cloud service provider and used by a cloud service consumer. The NIST [16] defines Cloud Computing via five **characteristics**: *On-demand self-service*, *broad network access*, *resource pooling*, *rapid elasticity*, and *measured service*. Different **deployment models** (*Public*, *Private*, *Community*, and *Hybrid*) are used to describe ownership and control of the cloud environment, while **service models** describe the level of abstraction from a consumer perspective (*Software-*, *Function-*, *Platform-*, *Container-*, and *Infrastructure as a Service*. They are abbreviated to SaaS, FaaS, PaaS, CaaS, and IaaS, respectively). CaaS and FaaS are often used in the industry.
- **Microservices** are considered – clearly not the only – but the 'most native' architectural style in cloud-native architectures. They grant loose coupling as well as independent scaling and deployment [14]. These services are modeled around business functionality and gain high cohesion in the process.
- **Containerization** is the process of deploying software for a cloud-native architecture as isolated, virtualized units. Containers offer efficient hardware usage and dynamic resource allocation.
- **Automation** in a cloud-native architecture aims at automated deployments of microservices and new functionality. To achieve this goal, concepts like Continuous Integration and Continuous Deployment (CI/CD) [12] and Infrastructure as Code (IaC) [24] play a vital role.
- **Observability** is the aggregation and analysis of data in a system to gain transparency and understanding about the internal components [25]. This supports troubleshooting in a microservices-based system, where data and business logic are distributed [12].

The following section will provide insight into our cloud-native technical reference architecture, designed with our business partners in mind.

V. Cloud-native Technical Reference Architecture for Insurance Companies

This section provides an overview of T-RaMicsV (see Figure 2). It has been designed using a cloud-native approach, which is expected to provide good scalability, short release cycles, and high resilience.

T-RaMicsV provides for private cloud and public cloud as the underlying operating model. The host machines of all elements of the architecture are physical or virtual machines, whereby the former can also include mainframes typical for Insurance Companies. In line with the cloud operating model, the host machines are operated in the company's data center or booked as IaaS solutions from public cloud service providers. A hybrid cloud scenario is also made possible by T-RaMicsV. In the insurance context, for example, it would be conceivable to use only certain IT services in the public cloud while continuing to process highly regulated, sensitive customer data in the company's own data center.

The chosen cloud-native approach implies the operation of microservices as containerized workloads. Thus, a central element of the reference architecture is the container orchestration platform (COP). It is operated by the company itself or purchased as a managed service from a public cloud provider. All microservices and their databases, the Enterprise Service Wrapper (ESW), a MoM for message-driven communication, and the Security Token Service (STS) are operated in containers. These run in a container runtime environment managed by the container orchestration platform, which also distributes them across multiple host machines. A microservice is horizontally scaled in an automated fashion by the platform as a set of containers, where one container corresponds to one service instance.

Furthermore, the platform performs the task of service discovery, i.e., a mechanism by which the microservices of the architecture can find and address each other. In addition, the COP provides routing capabilities and allows load balancing between multiple instances of a microservice.

The MSA includes business-oriented microservices, named A to C, as examples in Figure 2. These mainly communicate in an asynchronous, message-driven fashion via topics or queues provided by the MoM. Depending on persistence needs, a microservice may exclusively use its own database.

For authentication and authorization purposes, microservices exchange JSON Web Tokens (JWT) with each other as security tokens. These are issued by the STS of the architecture, to which the microservices must authenticate themselves. In addition to the use of JWT, T-RaMicsV stipulates that microservices communicate with each other in an mTLS-encrypted form.

To gain transparency in the architecture, metrics, logs, and traces of the services are collected. These can be aggregated, processed, and analyzed in corresponding solutions.

As in correspondence with RaMicsV, the SOA with ESB is understood as an existing system landscape part in T-RaMicsV, in which existing SOA services are operated. The connection to the ESB is a classic use case in the context of insurance. The legacy systems characterized by SOA cannot be replaced by microservices but should instead be extended. The insurance partners are aiming for the coexistence of SOA and microservices.

In the insurance industry, for example, SOAP — or REST-based web services, Enterprise Java Beans (EJB), or SAP systems offered as SOA services are conceivable. The SOA with ESB is run on host machines in the company's own data center but not on the container orchestration platform. The Microservices Architecture on the COP is considered a part of the landscape in which new services can be realized

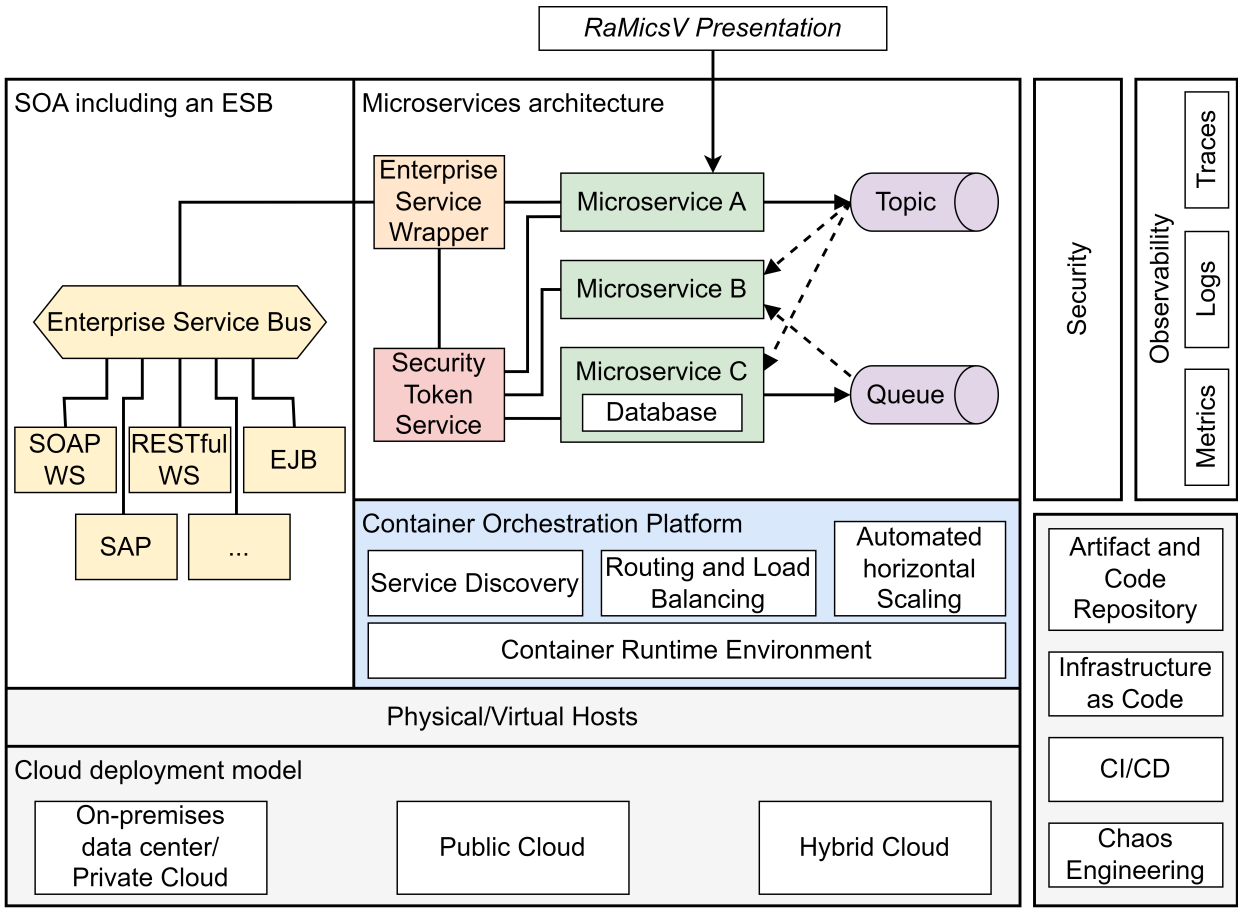

Fig. 2. Building Blocks of the Technical Reference Architecture T-RaMicsV.

as microservices. Existing SOA services could be migrated successively to the Microservices Architecture as required.

T-RaMicsV provides a proposal for the ESW, a component directly derived from RaMicsV. The ESW acts as a central transition from the Microservices Architecture to the ESB of the SOA. It is accessed by one or more microservices via REST when they need to consume a SOA service.

The software delivery process is planned to be automated as much as possible. For this purpose, CI/CD pipelines are used to build, test, and deliver artifacts. Provisioning of virtual infrastructure, especially the one provided by a public cloud service, should be based on the IaC principle. The infrastructure and application code, as well as the software artifacts, are captured and versioned in appropriate repositories. With the aim of increasing the resilience of the Microservices Architecture in production, it may additionally be tested using Chaos Engineering methods.

In the following, we give an overview of a concrete implementation example of our proposed technical reference architecture and shed light on more details.

VI. Cloud-native Microservices

This section presents the more specific design of microservices in T-RaMicsV, together with an overview of an implementation example of the proposed approach (see Figure 3). It was operated in the eduDScloud, a private cloud used for academic and educational purposes [26].

The example implements a part of a typical business process in the insurance industry we modeled, called the *Car Insurance Process* (from [18]). In this process, a customer submits an application to an insurance company to successfully conclude a contract for a car insurance policy. The workflow in this example is realized by the four business-oriented microservices `insurability`, `application`, `premium`, and `policy`.

To achieve high availability of the microservices, asynchronous communication over a MoM is preferred, and synchronous calls are to be reduced to a minimum. For this purpose, a solution like Apache Kafka can be used to implement messaging. The synchronous communication style is implemented as REST calls in combination with JSON as the serialization format. It is a common and well-understood synchronous inter-process communication style between microservices.

The ESW is implemented as a technical microservice (`eswrapper`) that propagates calls from the MSA to the ESB of the SOA. The ESB, in turn, consumes a SOA service, which can be a SOAP web service, as shown in the example (`solvency`). The JWT that the microservices use for authen-

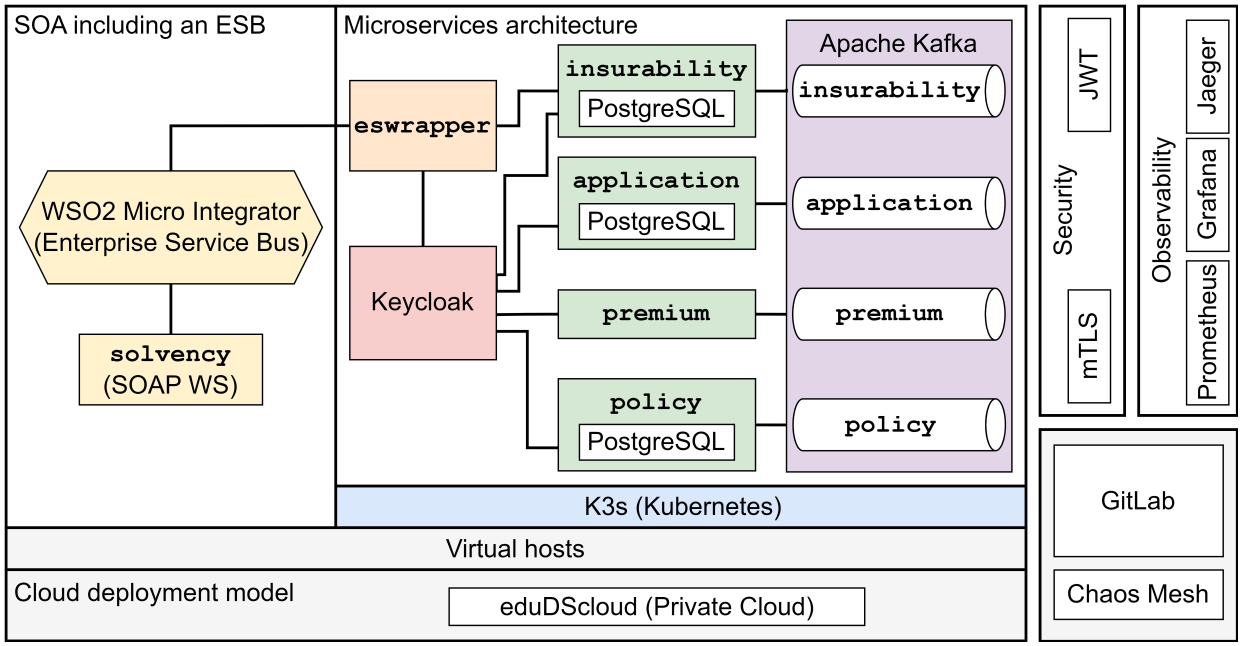

Fig. 3. Technical Architecture for the *Car Insurance Process*.

tication and authorization purposes are issued by Keycloak in the role of the STS.

A. Event-driven Choreography

In continuation of our previous work in [11] and [27], choreography is preferred over orchestration in T-RaMicsV to realize business processes through a microservices-based architecture. Furthermore, event-driven communication is proposed to implement the choreography. This approach supports loose coupling of microservices and enables good scalability of the architecture. The complexity caused by the implicit communication of the microservices of this approach has to be addressed by observability solutions, e.g., Prometheus and Jaeger with Grafana, for visualization purposes.

The publication of events is carried out according to the *publish-subscribe* pattern. A service emits events according to the *fire-and-forget* principle in a topic managed by the MoM. Interested services can subscribe to the topic, receiving the corresponding events, whereas the publishing service has no knowledge or expectation of whether or not the events are processed. The choreography then emerges as an interplay of publishing events and responding to those events.

The MoM only plays the role of a mediator that forwards the events. It is not meant as an orchestrator in the sense of orchestrating services since it does not include any logic to control or monitor the business-oriented correctness of the communication flow.

B. Scalability

The scalability of microservices in T-RaMicsV is considered using Newman's extended scaling cube [12]. Scaling is based on the dimensions of functional decomposition into microservices and horizontal scaling of these services through replication. The functional decomposition of a domain into microservices is derived directly from RaMicsV and is an essential component of the chosen cloud-native approach. Horizontal scaling is achieved by running the microservices in containers.

A container serves as a lightweight delivery unit that provides a consistent, isolated environment for the microservice and can be updated independently of other containers. The containers can be distributed and moved virtually without much effort. This results in the potential for more efficient utilization of hardware resources and higher availability.

Replication of a microservice into multiple instances can be achieved by various provisioning of a corresponding container. This and the management of the instances in the form of the individual containers is handled by the container orchestration platform. The platform also provides the mechanism for load balancing and routing requests to a group of replicas of a microservice. An often used platform for this is Kubernetes, available in different distributions, like K3s, which was used in the example (see Figure 3).

The dimensions of data partitioning and vertical scaling are not considered in a focused manner by T-RaMicsV. In the case of data partitioning, it should be mentioned that concrete DBMS products that can be used in principle may implement internal mechanisms of data partitioning. For example, consider the document-oriented DBMS MongoDB, which implements data partitioning via its sharding technology [28]. The elasticity of the Microservices Architecture is addressed by horizontal scaling of the services.

This must take place in an automated manner, depending on the generated load. This is the responsibility of the container

orchestration platform, which offers a corresponding mechanism for automating the horizontal scaling of the services. Depending on the workload on the existing containers, new ones are started up, or existing ones are terminated again when they run idle.

The availability of a service can be supported by horizontal scaling. A service remains available overall, despite the failure of individual instances, as long as a correspondingly high number of replicas is provided that is ready to respond to queries.

C. Fault tolerance

The distribution of the containers across different virtual or physical host machines managed by the COP is significant for the overall consideration of resilience. If all instances of a microservice were running on the same host machine and that machine was to fail, the service would not be accessible despite replication. The same applies further if the COP is run on virtual machines and these are only deployed on one physical one.

The virtual machines must be distributed across multiple physical ones. The distribution of the virtual host machines to multiple physical ones is outside the responsibility of the container orchestration platform, as it does not differentiate between physical and virtual host machines. This must be handled by the underlying virtualization solution.

D. Delivery

The container orchestration platform is responsible for managing a highly dynamic environment that results from continuous shutdown and startup of different containers. This is done intentionally, e.g., by new releases of services, or unintentionally, e.g., in case of failure.

To be resource-efficient and minimize the time to boot up containers, lightweight software artifacts should be targeted. Thus, a container image of a microservice should always be designed with a 'just as much as necessary' approach in mind. A lightweight operating system and only the packages, libraries, and files required for the service running in the container should be included in the image.

The infrastructure should be provided according to the IaC principle, especially when using public cloud services, and configured as immutable infrastructure only via the code. The infrastructure code can be recorded in version management systems, such as Git, and hosted with solutions like GitHub or GitLab, as used in the example. It thus serves as the *Single source of truth* of the infrastructure and can be versioned like application code. This creates transparency among administrators and developers and enables traceable, repeatedly executable deployment. In particular, in the event of a failure, this approach allows the system to be reliably recovered.

The application and infrastructure code of different microservices should be managed separately in their respective repositories. This can lead to redundant code between the repositories of different microservices. However, to support the autonomy and loose coupling of the microservices, this is recommended, especially regarding different development teams working on each of them. In this context, a platform like GitLab can also be used to implement CI/CD pipelines to automate the delivery process.

VII. Conclusion and Future Work

In this contribution, we present a cloud-native-based technical reference architecture that aims to build compliant microservices-based applications for insurance companies. The architecture adopts our partner's special requirements, such as integrating SOA services.

The reference architecture is applied to the car insurance process to shape an application-specific architecture for this scenario. With the given selection of specific technology, we present an instance of one possible implementation of our technical reference architecture.

The following steps in our research work are the integration of Microsoft Azure as a prominent public cloud solution (see block 'Cloud deployment model,' Figure 2). Again, the Car Insurance Process will be used to build a conforming architecture. The findings may require enhancing or even redesigning our technical reference architecture.

In the following work, we will carry out experiments on the technical reference architecture using chaos engineering. We will focus on requirements such as scalability. We are also investigating the topic of security in the context of microservices using homomorphic encryption within cloud environments.

Another topic is the refinement of the building block Business Processes within the responsibility area Business Logic & Data (see Figure 1). Choreography is the preferred approach to implementing workflows. Based on BPMN Choreography diagrams, we identified and classified frequently occurring patterns in the context of the insurance industry. Our goal is to implement the patterns to make choreography diagrams executable.

References

[1] C. Schulze, H. Meyer, A. Koschel, A. Hausotter, A. Link, and T. van Dorp, "A Technical Reference Architecture for Microservices-based Applications in the Insurance Industry," in *SERVICE COMPUTATION 2024, 15th Intl. Conf. on Advanced Service Computing*. IARIA, ThinkMind, 2024, pp. 1–6, Online. Available: https://www.thinkmind.org/library/SERVICE_COMPUTATION/SERVICE_COMPUTATION_2024/service_computation_2024_1_10_10009.html [retrieved: 09, 2024].

[2] M. Fowler and J. Lewis, "Microservices a definition of this new architectural term," 2014, Online. Available: https://martinfowler.com/articles/microservices.html [retrieved: 04, 2023].

[3] H. Knoche and W. Hasselbring, "Drivers and Barriers for Microservice Adoption–A Survey among Professionals in Germany," *Enterprise Modelling and Information Systems Architectures (EMISAJ)*, vol. 14, p. 10, 2019.

[4] A. Hausotter, C. Kleiner, A. Koschel, D. Zhang, and H. Gehrken, "Always Stay Flexible! WfMS-independent Business Process Controlling in SOA," in *15th IEEE Intl. Enterprise Distributed Object Computing Conference Workshops*. IEEE, 2011, pp. 184–193.

[5] A. Hausotter, A. Koschel, M. Zuch, J. Busch, and J. Seewald, "Components for a SOA with ESB, BPM, and BRM – Decision framework and architectural details," *Intl. Journal On Advances in Intelligent Systems*, vol. 9, no. 3 & 4, pp. 287–297, Dec. 2016, [Online]. Available: https://www.thinkmind.org/index.php?view=article&articleid=intsys_v9_n34_2016_6. [retrieved: 12, 2023].

[6] A. Hausotter, A. Koschel, J. Busch, and M. Zuch, "A Flexible QoS Measurement Platform for Service-based Systems," *Intl. Journal On Advances in Systems and Measurements*, vol. 11, no. 3 & 4, pp. 269–281, Dec. 2018, [Online]. Available: https://www.thinkmind.org/index.php?view=article\&articleid=sysmea_v11_n34_2018_4. [retrieved: 12, 2023].

[7] A. Koschel, A. Hausotter, M. Lange, and P. Howeihe, "Consistency for Microservices - A Legacy Insurance Core Application Migration Example," in *SERVICE COMPUTATION 2019, The Eleventh International Conference on Advanced Service Computing*, Venice, Italy, 2019, [Online]. Available: https://thinkmind.org/index.php?view=article&articleid=service_computation_2019_1_10_18001. [retrieved: 12, 2023].

[8] H. Knoche and W. Hasselbring, "Using Microservices for Legacy Software Modernization," *IEEE Software*, vol. 35, no. 3, pp. 44–49. [Online]. Available: https://ieeexplore.ieee.org/document/8354422/

[9] A. Koschel, A. Hausotter, R. Buchta, A. Grunewald, M. Lange, and P. Niemann, "Towards a Microservice Reference Architecture for Insurance Companies," in *SERVICE COMPUTATION 2021, 13th Intl. Conf. on Advanced Service Computing*. IARIA, ThinkMind, 2021, pp. 5–9, Online. Available: https://www.thinkmind.org/articles/service_computation_2021_1_20_10002.pdf [retrieved: 10, 2023].

[10] A. Hausotter, A. Koschel, M. Zuch, J. Busch, and J. Seewald, "Microservices Authentication and Authorization from a German Insurances Perspective," *Intl. Journal od Advances in Security*, vol. 15, no. 3 & 4, pp. 65–74, 2022, Online. Available: https://www.iariajournals.org/security/tocv15n34.html [retrieved: 01, 2024.

[11] A. Koschel, A. Hausotter, R. Buchta, C. Schulze, P. Niemann, and C. Rust, "Towards the Implementation of Workflows in a Microservices Architecture for Insurance Companies: The Coexistence of Orchestration and Choreography," in *SERVICE COMPUTATION 2022: The Fourteenth International Conference on Advanced Service Computing*, 2022, Online. Available: https://www.thinkmind.org/articles/service_computation_2022_1_10_10002.pdf [retrieved: 12, 2023].

[12] S. Newman, *Building Microservices: Designing Fine-Grained Systems*, 2nd ed. Sebastopol, Kalifornien, USA: O'Reilly Media, Inc., 2021.

[13] André Fachat. Challenges and benefits of the microservice architectural style, part 1. [Online]. Available: https://developer.ibm.com/articles/challenges-and-benefits-of-the-microservice-architectural-style-part-1/

[14] C. Richardson, *Microservices Patterns: With examples in Java*. Shelter Island, New York: Manning Publications, 2018.

[15] The Linux Foundation - The Cloud Native Computing Foundation, "Cloud Native Computing Foundation ("CNCF") Charter," 2023, Online. Available:https://github.com/cncf/foundation/blob/main/charter.md [retrieved: 12, 2023].

[16] P. Mell and T. Grance, *NIST Special Publication 800-145: The NIST Definition of Cloud Computing*, National Institute of Standards and Technology, U.S. Department of Commerce, Gaithersburg, Maryland, USA, 2011, Online. Available: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf [retrieved: 12, 2023].

[17] S. Reinheimer, *Cloud Computing - Die Infrastruktur der Digitalisierung (The infrastructure of digitization)*, 1st ed. Springer Vieweg Wiesbaden. [Online]. Available: https://doi.org/10.1007/978-3-658-20967-4

[18] M. Stadler and U. Gail, *Die Kfz-Versicherung - Grundlagen und Praxis (The car insurance - basics and practice)*. Karlsruhe: VVW GmbH, 2015.

[19] Gesamtverband der Deutschen Versicherungswirtschaft e.V. - General Association o.t. German Insurance Industry, "VAA Final Edition. Das Fachliche Komponentenmodell (VAA Final Edition. The Functional Component Model)," 2001.

[20] OMG, *Business Process Model and Notation (BPMN), Version 2.0*, Object Management Group Std., Rev. 2.0, January 2011, Online. Available: http://www.omg.org/spec/BPMN/2.0 [retrieved: 12, 2023].

[21] A. Koschel, A. Hausotter, R. Buchta, A. Grunewald, M. Lange, and P. Niemann, "Towards a Microservice Reference Architecture for Insurance Companies," in *SERVICE COMPUTATION 2021: The Thirteenth International Conference on Advanced Service Computing*, 2021, pp. 5–9, Online. Available: https://www.thinkmind.org/articles/service_computation_2021_1_20_10002.pdf [retrieved: 12, 2023].

[22] S. Angelov, P. Grefen, and D. Greefhorst, "A classification of software reference architectures: Analyzing their success and effectiveness," in *2009 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, 2009, pp. 141–150.

[23] Y. Yu, H. Silveira, and M. Sundaram, "A microservice based reference architecture model in the context of enterprise architecture," in *2016 IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, 2016, pp. 1856–1860.

[24] K. Morris, *Infrastructure as Code: Dynamic Systems for the Cloud Age*, 2nd ed. Sebastopol, Kalifornien, USA: O'Reilly Media, Inc., 2021.

[25] C. Majors, L. Fong-Jones, and G. Miranda, *Observability Engineering: Achieving Production Excellence*. Sebastopol, Kalifornien, USA: O'Reilly Media, Inc., 2022.

[26] D. Schöner, A. Koschel, and F. Heine, "Teaching Microservices in the Private Cloud by Example of the eduDScloud," in *SERVICE COMPUTATION 2018: The Tenth International Conferences on Advanced Service Computing*, 2018, pp. 36–39, Online. Available: https://thinkmind.org/articles/service_computation_2018_2_30_18003.pdf [retrieved: 12, 2023].

[27] A. Koschel, A. Hausotter, C. Schulze, A. Link, and H. Meyer, "Towards patterns for choreography of microservices-based insurance processes," in *SERVICE COMPUTATION 2023: The Fifteenth International Conference on Advanced Service Computing*, 2023, Online. Available: https://www.thinkmind.org/articles/service_computation_2023_1_10_10003.pdf [retrieved: 12, 2023].

[28] MongoDB Manual, "Sharding," Online. Available: https://www.mongodb.com/docs/manual/sharding/ [retrieved: 01, 2024].