Hitchhiker's Guide for Explainability in Autoscaling

Floriment Klinaku
Institute of Software Engineering, University of Stuttgart
Stuttgart, Germany
floriment.klinaku@iste.uni-stuttgart.de

Sandro Speth
Institute of Software Engineering, University of Stuttgart
Stuttgart, Germany
sandro.speth@iste.uni-stuttgart.de

Markus Zilch
Institute of Software Engineering, University of Stuttgart
Stuttgart, Germany
markus.zilch@iste.uni-stuttgart.de

Steffen Becker
Institute of Software Engineering, University of Stuttgart
Stuttgart, Germany
steffen.becker@iste.uni-stuttgart.de

ABSTRACT

Cloud-native applications force increasingly powerful and complex autoscalers to guarantee the applications' quality of service. For software engineers with operational tasks understanding the autoscalers' behavior and applying appropriate reconfigurations is challenging due to their internal mechanisms, inherent distribution, and decentralized decision-making. Hence, engineers seek appropriate explanations. However, engineers' expectations on feedback and explanations of autoscalers are unclear. In this paper, through a workshop with a representative sample of engineers responsible for operating an autoscaler, we elicit requirements for explainability in autoscaling. Based on the requirements, we propose an evaluation scheme for evaluating explainability as a non-functional property of the autoscaling process and guide software engineers in choosing the best-fitting autoscaler for their scenario. The evaluation scheme is based on a Goal Question Metric approach and contains three goals, nine questions to assess explainability, and metrics to answer these questions. The evaluation scheme should help engineers choose a suitable and explainable autoscaler or guide them in building their own.

CCS CONCEPTS

• **Computer systems organization** → **Cloud computing**; • **Software and its engineering** → *Software performance*; *Software usability*.

KEYWORDS

Cloud, Elasticity, Explainability, Requirements, Evaluation

ACM Reference Format:
Floriment Klinaku, Sandro Speth, Markus Zilch, and Steffen Becker. 2023. Hitchhiker's Guide for Explainability in Autoscaling . In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3578245.3584728

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal

ACM ISBN 979-8-4007-0072-9/23/04…$15.00
https://doi.org/10.1145/3578245.3584728

1 INTRODUCTION

Modern applications often follow a microservice-based architectural style built as cloud-native applications to adapt to the constantly changing workload [15]. These self-adaptive systems use autoscalers to automatically scale the components to guarantee the applications' quality of service, e.g., response time, to ensure site reliability [3]. For example, there are rule-based autoscalers like the Kubernetes Horizontal Pod Autoscaler (HPA) [1], and CAUS [13, 24], or machine learning-based autoscalers like the framework by Toka et al. [20]. Autoscalers are configured using policies and parameters which effectively determine when and how to scale. The behavior of autoscalers is hard to grasp due to the use of complex models and techniques internally, such as queuing models, control theoretic models, or even machine learning models. Furthermore, understanding their effect is challenging because of the distribution and the decentralized decision-making, e.g., each microservice scales through a different mechanism. For example, one downstream microservice scales if the average response time is higher than a certain amount, while the upstream microservice scales if the average CPU load of the replicas is below a threshold. If, in such a scenario, autoscalers are poorly configured, their decisions end up competing and resulting in many contradicting scaling events. Hence, autoscalers are prone to misconfiguration, which is difficult to detect, and the understanding of coordinated and conflicting scaling events is challenging in uncertain environments [14, 18]. Thus, we require self-explanatory autoscalers, which is one of the six challenges for production-ready autoscaling identified by Straesser et al. [19]. However, there are no common expectations known of DevOps engineers for the explainability of such autoscalers. This leads us to the research question guiding us in our work:

> **RQ:** *"Are there common expectations of software engineers with operational tasks on the explainability of autoscalers?"*

In this paper, we present a workshop with engineers from industry and academia to elicit the requirements and expectations of engineers on explainable autoscaling. Furthermore, we describe an evaluation scheme that we built upon the requirements and which should guide engineers in selecting a fitting explainable autoscaler or building their own. The evaluation scheme consists of three goals, nine questions, and three groups of metrics. Therefore, our contributions are (1) requirements and expectations of engineers for explainable autoscaling and (2) an evaluation scheme to check existing autoscalers against their explainability.

2 RELATED WORK

The first related work cluster highlights the problem of autoscaling and elasticity policies in terms of explainability. However, they do not go far enough to form requirements and evaluation criteria for explainability in the context of autoscaling. For example, Straesser et al. [19] identify through experimentation that having self-explanatory autoscalers is one of the six challenges for production-ready autoscaling. They argue that explainability becomes a concern for the whole spectrum of autoscaling methods. From rule-based, where too many rules are present, i.e., the architectural style of microservices with multiple services and many rules, to complex queuing-based models and data-driven machine-learning-based autoscalers. Ghanbari et al. [10] suggest that the decisions for model-based autoscaling, e.g., queuing-based or machine learning, are more difficult to understand than decisions in rule-based systems. The survey from Chen et al. [8] highlights the large design space for achieving self-aware and self-adaptive autoscaling frameworks. Those design decisions affect how the software engineer interacts with the autoscaling framework.

The second cluster contains work in the area of providing explainability for self-adaptive systems through concrete architectural proposals and frameworks. Blumreiter et al. [4] propose the MAB-EX loop (Monitor, Analyse, Build, and Explain) for enabling explainability for cyber-physical systems in general. In the *Monitoring* phase, the framework shall collect runtime data relevant to generating explanations. In the *Analyse* phase, the framework determines the need for the explanation either triggered by situations that need to be explained or by queries from the operator. The explanation is built, in the *Build* phase, from a behavioral model of the system that captures the causal relationships between events and system reactions. Their solution is generic and may apply to autoscalers as well. Another instance is the XSA (eXplainable Self-Adaptations) framework [7] in which authors first define five levels of explainability inspired from [2] and then construct a framework that provides explainability up to the fourth level. The difference to our work is that we focus on better understanding the requirements for explainability from the perspective of software engineers with operational tasks in the context of automated management of resources (which can be viewed as a subset of self-adaptive systems) rather than proposing a concrete solution to achieve explainability.

Explainability as a term is tightly connected with the emergence and popularity of artificial intelligence methods [11]. In this work, we view autoscalers for microservice-based applications as black-box autonomous systems that may or may not use machine learning-based models internally. Our work explores the requirements for explainability at the interface of such systems from the operator's viewpoint, which may cause the derivation of requirements for the internal subsystems that may be AI-based. Hence all the work in explaining particular machine learning-based methods become relevant in the solution space.

Finally, we tend to contribute to converging what explainability entails for autoscalers as a quality attribute and how to quantify and compare alternatives. This way, we help practitioners to choose the most self-explainable alternative. Furthermore, it helps research solutions that do not neglect this quality attribute. In this context, Rosenfeld and Richardson [16] observe that currently, there are no commonly agreed definitions of explainability. They define questions to identify the reason why explanations are needed, who the target is, what interpretation can be generated when the information should be presented, and how explanations can be evaluated. Doshi-Velez and Kim [9] define interpretability as the ability to explain or present something in an understandable way to a human. They propose a taxonomy of evaluation approaches for interpretability. Their taxonomy includes functionally-grounded, human-grounded, and application-grounded evaluations.

3 REQUIREMENTS ENGINEERING

In this section, we describe the questionnaire and workshop we conducted. Furthermore, we elaborate on the demographics and the most important results. Then, we describe the requirements which we elicited based on the questionnaire results. Figure 1 depicts the entire process. The evaluation scheme is presented in Section 4.

3.1 Method

Our objective stakeholders are software engineers with operational tasks, e.g., DevOps engineers, researchers in the area of operations, and operators. Therefore, we decided on a workshop with such engineers of industry and academia to elicit common requirements and expectations on the explainability of autoscalers. First, we created a questionnaire consisting of 15 questions (13 content and 2 closing). The first five questions survey the participant's demographics, i.e., their job, company size, prior experience with autoscalers, and, optional, the company they are working for. The next four questions query which autoscalers the participants already have used, their preferred type (rule-, control-, or ML-based), the metrics the participants use for autoscaling decisions, and their current process to debug and understand the autoscaling behavior. In the last four content-focused questions, the participants could state the explainability questions they want an autoscaler to answer, the kind of autoscaler's interface they are looking for, additional metrics than the ones already stated to understand the autoscaling behavior, and the occasions they need information from the system, e.g., system failure. The questionnaire can be found online[1].

Second, we developed an example scenario based on the T2-Project [17] to create a common understanding of the problem as a basis for the workshop discussions and created the workshop. The example shows a simple architecture with an autoscaler. The workshop was designed for small groups to discuss the explainability of autoscaling behavior based on this example and took 30 minutes for each session. If one participant was only available alone, we discussed it with them. After the discussion, the participants had to fill out the questionnaire.

Furthermore, we analyzed the thesis by Weiler [21] for requirements fitting to explainability in autoscaling. Weiler interviewed five experienced engineers within a single organization to elicit requirements for their autoscaling solution both from a business and technical perspective. Consequently, not all but a few of the elicited requirements fit in our context.

[1] https://zenodo.org/record/7564994

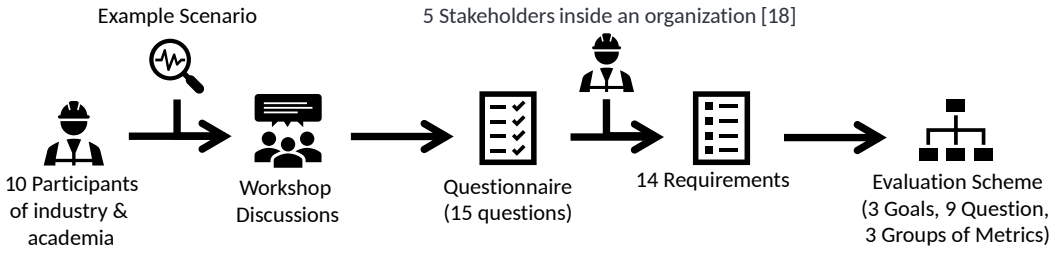

Figure 1: Process to elicit requirements and design the evaluation scheme.

3.2 Questionnaire and Workshop Results

For space reasons, we describe only the most relevant results. The full anonymized results can be found online[1].

Demographics: In total, we had ten participants from Germany and Switzerland, most of them working as DevOps engineers or software architects. Furthermore, the majority work in big-sized companies (more than 250 employees), e.g., Microsoft or Mercedes Benz. The majority actively use and manage autoscalers from time to time or regularly.

Current usage: Nearly all participants used Kubernetes HPA, and two additionally used Azure Autoscale and AWS EC2 Autoscale. Hence, most used rule-based autoscalers, and three participants also used ML-based autoscalers, of which only one participant stated an LSTM-based autoscaler. The most popular metrics are CPU load, response time (latency), RAM load, and the number of messages in a queue. As a current process to debug and understand autoscaling behavior, the participants have either no process or check and compare metrics and HPA events.

Explaining autoscaling behavior: Nearly all of the participants stated similar explainability questions. In total, they are interested in why the autoscaler did or did not scale, how much it scaled and why this amount, and when it scaled. Furthermore, some are interested in forecast and what-if questions. Regarding the provided interface, the participants seem to require a mixed set of interfaces, from graphical over a remote API to a CLI and integration to other tools, e.g., Grafana. Additionally, the participants are interested in Service Level Objectives (SLOs), which incorporate business information, the correlation between scaling and the input values and their weight, and how the autoscaler came to a decision, esp. if they were bad decisions. The participants generally state that they need this information, especially when system failure or SLO violations occur, and more specific scenarios, e.g., if a huge load is expected to ensure correct scaling or when the autoscaler decided to scale too much.

3.3 Elicited Requirements

Based on the questionnaire results and discussions during the workshop, we elicited 34 requirements. Due to space, we aggregate some and focus on the most relevant 11 requirements. All requirements are available in Zilch's master's thesis [24]. Furthermore, we include three additional requirements, which Weiler [21] elicits by interviewing experienced engineers in the context of a single organization. In Table 1, we present the resulting 13 requirements. The first four requirements describe the representation of metrics' data, logs, and scaling decisions. Requirements five and six state that autoscalers should support forecasts, what-if scenarios, and alternative decisions for past events. Requirement seven covers system failures, and SLO violations, as engineers especially try to understand the autoscaler's behavior and decisions in such cases. Potential information could be a failure analysis and metrics for the moment leading to the system failure or SLO violation. Furthermore, autoscalers should detect anomalies in workload and their scaling decisions (R8) to identify unnormal behavior. Additionally, they should allow the management and scaling of different components (R9). Requirements ten and eleven cover the autoscalers interfaces to other tools and the engineer. Additionally, autoscalers should support self-optimizing their scaling rules if thresholds are suboptimal (R12) and allow exceptions and manual overriding of the scaling rules for specific time frames or customers (R13). To fulfill requirement 14, autoscalers should allow a configuration via SLOs.

4 EVALUATION SCHEME

We devise an evaluation scheme that follows the Goal Question Metric [6] plan proposed in Zilch's Master's thesis [24]. The evaluation scheme contains the three goals (G1) *accessibility*, (G2) *configurability*, and (G3) *explainability*. *Explainabaility* explicitly has

#	Description
1	Store input metrics as time series
2	Represent metric's data visually
3	Present actual and historical information and decisions from a timeline
4	Present logs corresponding to scaling decisions
5	Show forecasts for estimated workloads and answer what-if questions
6	Present alternative decisions it could have made at a given time
7	Incorporate details on system failure and SLO violation
8	Support anomaly detection for workloads and scaling decisions
9	Management and scaling of different components
10	Provide a mix of different interfaces, e.g., GUI, REST, and CLI
11	Provide an interface where the user can ask for explanations
12∗	Support self-optimization of scaling rules
13∗	Allow manual overriding for specific time frames or customers
14∗	Allow configuration via SLOs, e.g., guaranteed response times

Table 1: Most relevant requirements. Additional requirements marked with an ∗ are derived from [21]. The requirements are written in the form "The autoscaler should ...".

been stated in R11. Also, other requirements (e.g., R2, R4, or R5) state requirements that aid the engineer in understanding the autoscaler better. *Configurability* is included in the joint evaluation of explainability because for the software engineer operating the autoscaler, applying reconfigurations is the ultimate action that succeeds the understanding of the current workings and performance. The *Configurability* goal is covered in R9, R13, and R14. Similarly, *accessibility* is crucial in the interaction of the engineer and the autoscaler. Requirements R10, R2, R4 cover accessibility. This goes in hand with the definition of explainability given by Camili [7] in the context of self-adaptive systems as "... the ability of the system to make the entire adaptation process **transparent** and **comprehensible** ...". Configurability and accessibility aid the system's transparency and comprehension from the software engineer's point of view. We formulate all the questions as "how well" questions while we are interested to evaluate the overall quality of explainability for an autoscaler. For metrics, in addition to requirements that were mentioned in the workshop, we collect other existing metrics from related work.

4.1 Questions

Explainability. We formulate four questions of interest. The first question (Q1) is on how well the autoscaler explains its scaling decisions in relation to the models and constructs it uses. For example, a threshold-based autoscaler shall provide explanations in relation to the set thresholds and the metrics that triggered an exceed or undercut. The expectation is different for a Reinforcement Learning-based autoscaler which should include information on the current rewards and penalties. The second question (Q2) is on how well the autoscaler provide explanations in relation to the different type of queries: for example, whether it supports explanations on only "why" queries or also "what if" queries. The third question (Q3) determines the quality of explanations related to the environment: the workload and/or the application/component being scaled. The last question (Q4) assesses whether visuals have been used appropriately to aid explainability.

Configurability. For configurability, we devise one question (Q5) on how well does the autoscaler support reconfigurations. Reconfigurations, in our context, are changes by the software engineer through an interface that affect the operation of the autoscaler. One example of such a reconfiguration is changing the scaling threshold of the Horizontal Pod Autoscaler [1] by applying a new configuration through the `kubectl` command. Straesser et al. [19] state that keeping the configuration overhead as small as possible is challenging as many autoscaling approaches offer several parameters that end-users have to adjust. However, at the same time, based on the workshop with experts, they would like to have the right control over the behavior of the autoscaler with appropriate reconfiguration.

Accessibility. For accessibility, we devise two questions of interest from an evaluation perspective: first, determining how accessible (Q6) the autoscaler is. For example, the Horizontal Pod Autoscaler does not have a custom Graphical User Interface. For operational tasks, it relies on the default Kubernetes dashboard and a Command Line Interface (CLI) through `kubectl`. The second question evaluates the integration with observability tools[2] (Q8). Observability tools such as Prometheus or Grafana are by now standard components in a cloud environment.

4.2 Metrics

We create three groups of metrics motivated by the taxonomy of Doshi-Velez et al. [9]: (1) feature-grounded, (2) subjective-human-grounded, and (3) objective-human-grounded group of metrics. In evaluating and comparing two autoscaler concerning the posed questions, one needs to conduct experiments and surveys with human subjects besides feature comparisons. The first group of metrics is feature-grounded metrics, whereas the second and the third are human-grounded metrics. We propose candidate metrics within such groups, and practitioners could use this as a base, enrich it with new metrics and refine the existing ones.

MG1: Feature-grounded Group of Metrics. In feature-grounded metrics, we include various metrics that come from the requirements. Feature-grounded metrics could be obtained by reading the documentation or conducting end-user tests. First, a software engineer can assess whether the autoscaler to evaluate provides or promises any support for *Explainability*. In addition, the number of types of queries supported defines another metric. The categories for the type of queries may include the following: why, why not, what if, counterfactual, etc. (R5). These are two example metrics that aid in addressing question one and question two in metric group one. For *Configurability*, an autoscaler may support different types of reconfigurations. Based on the requirements, software engineers may be interested in reconfiguring different aspects of the autoscaler. One is adjusting model parameters. For example, they adjust hyperparameters for an ML-based or scaling rules for a rule-based autoscaler. Another reconfiguration type is adjusting temporal properties (e.g., cooldowns), which are independent of the internal models autoscalers use. Reconfigurations may also occur as changes to a given objective function (R13, R14). In the *Accessibility* part, we identify the number of interfaces the autoscaler supports. Interfaces to provide input and receive output include the command line interface (CLI), the graphical user interface (GUI), remote application programming interfaces (APIs), conversational interfaces (e.g., chatbot), and logs (R10). Multiple interfaces are relevant for different tasks, e.g., understanding scaling behavior requires a graphical interface, while configuring works best via a remote API. Finally, we could count the number of observability tools for which an integration exists or determine the support for elicited observability tools, e.g., Prometheus (R1-R4).

MG2: Subjective Human-grounded Group of Metrics. For subjective human-grounded metrics, Zhou et al. [22] identify trust, confidence, and preference as three subjective metrics. These metrics also apply in our context to quantify the "how well" questions for *Explainability*. These metrics reflect the subjective perception of the software engineer during an experiment or an empirical study and could be computed independently for questions 1-9. For example, the trust scale by Hoffmann et al. [12] captures the rating by the end-user whether the explainable AI system is predictable, reliable, efficient, and believable. Classical usability scales such as the SUS

[2] https://openapm.io/landscape

Goals	Questions		Metrics
G1 Explainability	Q1	How well does the autoscaler explain its decisions in relation to the models and constructs it uses?	MG1 Feature-grounded Group of Metrics: • Presence of explainability method • Number of supported query types { why, what if, counterfactuals, ... } • Number of supported configuration modes {Machine-learning parameters, temporal properties (e.g., cooldowns), objective function, scaling rules} • Number of interfaces it supports {CLI, GUI, Remote API, Conversational} • Number of observability tools an integration exists MG2 Subjective Human-grounded Group of Metrics • Trust ([12]) • Confidence • Preference MG3 Objective Human-grounded Group of Metrics • Task performance metrics (Time, Correctness) • Behavioral and Physiological Metrics
	Q2	How well does the autoscaler provide explanations in relation to the different types of queries?	
	Q3	How well does the autoscaler provide explanations in relation to the environment (workload and/or managed/scaled application (service))?	
	Q4	How well does the autoscaler use visuals to represent metrics, models, and operations?	
G2	Q5	How well does the autoscaler support reconfigurations?	
G3	Q7	How accessible is the autoscaler in terms of input and output interfaces it supports?	
	Q8	How well is the autoscaler integrated to observability tools?	

Table 2: Goals, questions, and metrics of our evaluation scheme. The goals are G1 (Explainability), G2 (Configurability), and G3 (Accessibility).

score [5] could give complementary preference insights for the system as a whole or for the explainability component specifically.

MG3: Objective Human-grounded Group of Metrics. Objective human-grounded metrics include all metrics that assess human performance and can be computed objectively. For example, measuring time for completing a task in a particular operating scenario determines an objective measure. In addition to time, computing a correctness score for the solution of the task is another metric that can be used to evaluate explainability for the autoscaler objectively. Both time and correctness have been used in various experiments and empirical studies for evaluating different software engineering methods, techniques, and tools. In addition to classical task performance metrics, there are studies that use behavioral and physiological metrics to assess trust and confidence. For example, Zhou et al. [23] use Galvanic Skin Response and Blood Volume Pulse as indicators of user trust.

5 THREATS TO VALIDITY

In this section, we describe threats to the validity of our requirements and evaluation scheme. The requirement's most critical threat to external validity is the number of engineers participating in the workshop and their countries. In total, only 10 engineers from Germany and Switzerland participated, which might not be representative for the entire population. However, they cover a good range of different companies, from big to small sizes. Therefore, their statements and our elicited requirements should be a good indicator of general expectations for explainable autoscaling. Furthermore, the workshop scenario and questionnaire might have led the participants in a specific direction so that other requirements, e.g., reusable configuration for multiple components, might have been overseen, resulting in a threat to construct validity. The evaluation scheme is a work-in-progress proposal, which we iterate twice: the initial proposal in Zilch's thesis [24] and the refinement in this paper. Although there is a link between the goals, questions, and metrics to the requirements, we did not evaluate its validity. However, researchers and practitioners could still use it as a guiding base to evaluate autoscalers w.r.t. explainability.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we outline a workshop with ten software engineers with operational duties from industry and academia to elicit their requirements and expectations on explainable autoscaling frameworks. We present the most relevant requirements and derive an evaluation scheme for autoscaling frameworks based on them using the Goal Question Metrics approach [6]. It consists of three goals, nine questions, and three groups of metrics to evaluate these questions. The evaluation scheme should guide DevOps engineers in selecting the best fitting explainable autoscaling framework for their purpose or building their own one. However, currently, some metrics are quite abstract and should be more detailed. In follow-up work, through a feature-based analysis, we are applying the evaluation scheme to autoscalers that are proposed for the Kubernetes ecosystem and the most popular cloud-provider autoscalers. The results of such an analysis, besides aiding in comparing different alternatives for autoscaling, serve as a validation of the evaluation scheme and will highlight its potential benefits and limitations. Furthermore, we are currently working on a tool that allows the engineer to ask explainability questions for autoscaling, get scaling events, and the autoscaler's reasoning for the scaling decisions visually presented.

REFERENCES

[1] Kubernetes HPA. https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/, 2021. [Online; 2022-12-28].

[2] EU Robotics AISBL. Robotics 2020 multi-annual roadmap for robotics in europe, call 1 ict23–horizon 2020. *Initial Release B*, 15(01), 2014.

[3] Betsy Beyer, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. *Site reliability engineering: How Google runs production systems*. "O'Reilly Media, Inc.", 2016.

[4] Mathias Blumreiter, Joel Greenyer, Francisco Javier Chiyah Garcia, Verena Klös, Maike Schwammberger, Christoph Sommer, Andreas Vogelsang, and Andreas Wortmann. Towards Self-Explainable Cyber-Physical Systems. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019, Munich, Germany, September 15-20, 2019*, pages 543–548. IEEE, 2019.

[5] John Brooke et al. SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7, 1996.

[6] Victor R Basili1 Gianluigi Caldiera and H Dieter Rombach. The goal question metric approach. *Encyclopedia of software engineering*, pages 528–532, 1994.

[7] Matteo Camilli, Raffaela Mirandola, and Patrizia Scandurra. XSA: eXplainable Self-Adaptation. In *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*, pages 189:1–189:5. ACM, 2022.

[8] Tao Chen, Rami Bahsoon, and Xin Yao. A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems. *ACM Comput. Surv.*, 51(3):61:1–61:40, 2018.

[9] Finale Doshi-Velez and Been Kim. Towards A Rigorous Science of Interpretable Machine Learning. 2017.

[10] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, and Gabriel Iszlai. Exploring Alternative Approaches to Implement an Elasticity Policy. In *IEEE International Conference on Cloud Computing, CLOUD 2011, Washington, DC, USA, 4-9 July, 2011*, pages 716–723. IEEE Computer Society, 2011.

[11] David Gunning, Mark Stefik, Jaesik Choi, Timothy Miller, Simone Stumpf, and Guang-Zhong Yang. XAI—Explainable artificial intelligence. *Science Robotics*, 4(37):eaay7120, 2019.

[12] Robert R. Hoffman, Shane T. Mueller, Gary Klein, and Jordan Litman. Metrics for Explainable AI: Challenges and Prospects, 2018.

[13] Floriment Klinaku, Markus Frank, and Steffen Becker. CAUS: An Elasticity Controller for a Containerized Microservice. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pages 93–98, 2018.

[14] Floriment Klinaku, Martina Rapp, Jörg Henss, and Stephan Rhode. Beauty and the beast: A case study on performance prototyping of data-intensive containerized cloud applications. In Dan Feng, Steffen Becker, Nikolas Herbst, Philipp Leitner, and Arthur Kang, editors, *ICPE '22: ACM/SPEC International Conference on Performance Engineering, Bejing, China, April 9 - 13, 2022, Companion Volume*, pages 53–60. ACM, 2022.

[15] Sam Newman. *Building Microservices: Designing Fine-Grained Systems*. O'Reilly, 2nd edition, 2021.

[16] Avi Rosenfeld and Ariella Richardson. Explainability in Human–Agent Systems. *Autonomous Agents and Multi-Agent Systems*, 33(6):673–705, may 2019.

[17] Sandro Speth, Sarah Stieß, and Steffen Becker. A Saga Pattern Microservice Reference Architecture for an Elastic SLO Violation Analysis. In *Companion Proceedings of 19th IEEE International Conference on Software Architecture (ICSA-C 2022)*. IEEE, March 2022.

[18] Sandro Speth, Sarah Stieß, and Steffen Becker. A Vision for Explainability of Coordinated and Conflicting Adaptions in Self-Adaptive Systems. In *Proceedings of 14th Central European Workshop on Services and their Composition (ZEUS 2022)*, pages 16–19. CEUR, February 2022.

[19] Martin Straesser, Johannes Grohmann, Jóakim von Kistowski, Simon Eismann, André Bauer, and Samuel Kounev. Why Is It Not Solved Yet?: Challenges for Production-Ready Autoscaling. In *ICPE '22: ACM/SPEC International Conference on Performance Engineering, Bejing, China, April 9 - 13, 2022*, pages 105–115. ACM, 2022.

[20] Laszlo Toka, Gergely Dobreff, Balazs Fodor, and Balazs Sonkoly. Adaptive AI-based auto-scaling for Kubernetes. In *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, pages 599–608. IEEE, July 2020.

[21] Simon Weiler. Automatic resource scaling in cloud applications - Case study in cooperation with AEB SE, 2021.

[22] Jianlong Zhou, Amir H. Gandomi, Fang Chen, and Andreas Holzinger. Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics. *Electronics*, 10(5):593, mar 2021.

[23] Jianlong Zhou, Huaiwen Hu, Zhidong Li, Kun Yu, and Fang Chen. Physiological Indicators for User Trust in Machine Learning with Influence Enhanced Fact-Checking. In *Lecture Notes in Computer Science*, pages 94–113. Springer International Publishing, 2019.

[24] Markus Zilch. Evaluation of Explainability in Autoscaling Frameworks. Master's thesis, University of Stuttgart, September 2022.