Received February 6, 2021, accepted February 26, 2021, date of publication March 9, 2021, date of current version March 26, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3064867

Quality-Aware DevOps Research: Where Do We Stand?

AHMAD ALNAFESSAH, ALIM UL GIAS, RUNAN WANG, LULAI ZHU, GIULIANO CASALE, AND ANTONIO FILIERI
Department of Computing, Imperial College London, London SW7 2AZ, U.K.

Corresponding author: Giuliano Casale (g.casale@imperial.ac.uk)

The work of Ahmad Alnafessah was supported in part by the National Center for Artificial Intelligence and Big Data Technologies, King Abdulaziz City for Science and Technology (KACST), Saudi Arabia. The work of Alim Ul Gias was supported by the Commonwealth Scholarship Commission in the U.K. The work of Ahmad Alnafessah, Lulai Zhu and Giuliano Casale was supported in part by the RADON project, funded by the European Union's Horizon 2020 Research and Innovation Program under Grant 825040.

ABSTRACT DevOps is an emerging paradigm that reduces the barriers between developers and operations teams to offer continuous fast delivery and enable quick responses to changing requirements within the software life cycle. A significant volume of activity has been carried out in recent years with the aim of coupling DevOps stages with tools and methods to improve the quality of the produced software and the underpinning delivery methodology. While the research community has produced a sustained effort by conducting numerous studies and innovative development tools to support quality analyses within DevOps, there is still a limited cohesion between the research themes in this domain and a shortage of surveys that holistically examine quality engineering work within DevOps. In this paper, we address the gap by comprehensively surveying existing efforts in this area, categorizing them according to the stage of the DevOps lifecycle to which they primarily contribute. The survey holistically spans across all the DevOps stages, identify research efforts to improve architectural design, modeling and infrastructure-as-code, continuous-integration/continuous-delivery (CI/CD), testing and verification, and runtime management. Our analysis also outlines possible directions for future work in quality-aware DevOps, looking in particular at *AI for DevOps* and *DevOps for AI software*.

INDEX TERMS DevOps, CI/CD, infrastructure as code, testing, artificial intelligence, verification.

I. INTRODUCTION

The rapid evolution of cloud and virtualization technologies over the last 15 years has brought to software vendors the ability to easily and programmatically control a broad set of computing resources in the execution environment of a software system. This development has then paved the way to increased levels of automation in the way software applications are delivered to production, for example by enabling continuous integration of new version of the application code. The resulting delivery paradigm, which places more attention towards continuous re-release, unified tooling and organizational processes, is often referred to as *DevOps*. Common DevOps advances include for example continuous-integration/continuous-delivery (CI/CD) pipelines, and highly-automated orchestration and configuration solutions for the runtime environment [1], [2].

DevOps tools and methods have also reduced the cultural and methodological divide between developers and operators [3], leading to the formation of many new organizational structures within software vendors, such as virtual teams composed of both developers and operators, and the establishment of new professional figures often referred to as DevOps engineers, who center their activity on tooling and automation across the whole application lifecycle.

A methodology that releases application versions at a faster pace than traditional methods is effective only if coupled with testing tools that can reduce the likelihood of failures in production. For this reason, quality assurance in DevOps is often a synonym of continuous *functional* testing methods to check the correctness of application prior to deployment. However, to accelerate the pace of delivery, quite often DevOps

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.

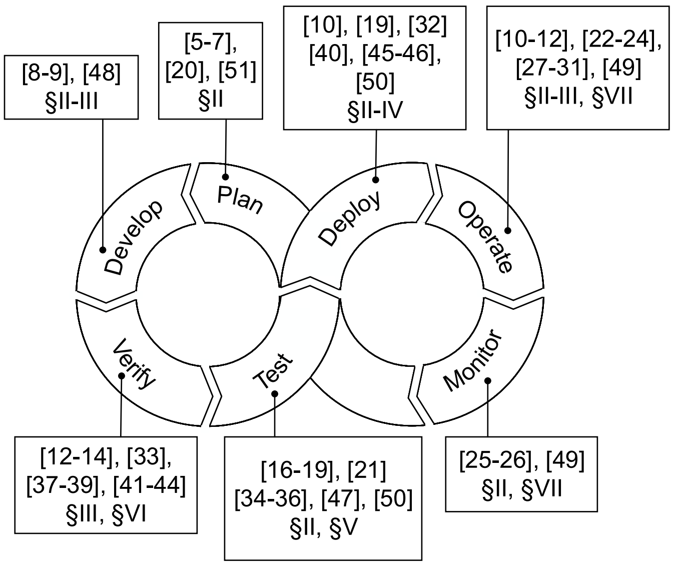

FIGURE 1. **Stages of the DevOps Cycle. The citations indicate works that mainly perform research in the context of that stage. Relevant sections of this paper are also indicated.**

testing methods lead to restrict the depth of scrutiny of the software system, leaving rooms for significant defects and bugs to still emerge in production. Several defects are related to properties other than correctness, such as performance, reliability, or cost. This problem has raised the attention of many research groups, leading to several research works that attempt to couple DevOps methods with novel forms of rapid and automated quality assurance, centered on a broader range of quality characteristics (*e.g.*, performance, reliability, availability, scalability, ...).

While the research community has sustained this effort by publishing numerous studies and innovative tools and methods to support quality analyses within DevOps, there is still a limited cohesion between the research themes in this domain and a shortage of surveys that holistically examine quality engineering work within DevOps. In this work, we address this gap by offering a survey of recent efforts in the area of *quality-aware* DevOps. Our work focuses on research efforts that aim at coupling the rapid delivery of DevOps with techniques to ensure that software artifacts also meet quality expectations on non-functional properties. Our analysis covers several tens of papers, categorizing the different contributions according to the software engineering area they mainly contribute to including architectural design, modeling, continuous integration and delivery, infrastructure, testing, verification, CI/CD and infrastructure-as-code, runtime management. We look also at the positioning of these works within the DevOps lifecycle stages. The paper, in particular, reveals that a highly-diverse body of work has been published on the subject, which yet leaves ample margins to carry out further investigations in areas that are systematically under-investigated from a quality angle, e.g., CI/CD & IaC and architectural design. We further look at the state-of-the-art on two emerging trends: *AI for DevOps* and *DevOps for AI software*, as these are expected to dominate the DevOps landscape in the years to come, and survey early works in these areas.

A. THE DevOps CYCLE

The reference stages of the DevOps lifecycle we consider are illustrated in Figure 1. The figure lists the bibliographic references that we classify as doing research in the context of that stage. Note that a paper may touch upon multiple stages, in which case it is listed on all relevant stages. The stage definitions we adopt are as follows:

- *Plan*: This stage aims at defining the objectives and requirements of the software production, along with the initial plan for updates and release across iterations.
- *Develop*: Based on the plan, developers focus on development and reviewing of software code and/or IaC. Typically, in this phase the code undergoes frequent commits on code repositories as well as integration and unit tests based on build automation tools.
- *Verify*: Verification is the process to evaluate the correctness of software artifacts in terms of the requirements.
- *Test*: In this phase, automation testing will be performed continuously to ensure the quality of the software artifact. Contrary to traditional tests, this phase can include the release of trial versions to part of the end user base, by means of canary testing.
- *Deploy*: This stage focuses on continuously re-deployment of the software in the production environment. This phase entails the problem of configuration management of the target platforms and resources.
- *Operate*: Operation in DevOps cycles deals with configuration and management of the software application

after deployment, e.g., resource provisioning and auto-scaling. Orchestrators and other runtime methods can be used to automatically instantiate and adapt at run-time the application topology and components.

- *Monitor*: Monitor the performance of deployed applications by collecting and analyzing usage data, which can help to detect and identify exceptions and provide feedback to iteratively improve the software. Continuous tracing and diagnostics of production problems is important to guide the evolution of the application across release cycles.

The above phases are qualitatively similar to those carried out in traditional software engineering methodologies, with the main difference being that the process of releasing the software artifact is continuous and highly-automated. We point the reader to books such as [4] for additional details on the above methodological phases.

B. METHODOLOGY

Our review methodology is as follows. We examine computer science peer-reviewed journals and conference papers written in English between 2015 and 2020. Papers are obtained with systematic searches using Google scholar for search strings always including ''DevOps'', one quality term between ''performance'', ''scalability'', ''quality'', ''quality-aware'', ''reliability'', ''availability'', or ''survivability''; and a third term matching the title of the sections of this paper (e.g., ''verification'', ''CI/CD'', etc). Books, presentations, thesis, technical reports, white papers, and patents are excluded from this study. After collecting the pool of paper, due to space limitations we have narrowed down the list to around 10 in each section. This has been done with manual screening of each paper, trying to identify a subset of papers that was representative of the whole category, as our goal is to illustrate different research challenges and approaches, rather than exhaustively list every individual contribution.

We have aligned these collected pool of papers with different stages of the DevOps lifecycle and presented in Figure 1. We also present a mapping of these papers with the considered quality attributes in Table 1. The table also classifies the papers based on their methodology. We considered the following methodologies:''Model-based'' - if the authors emphasize modeling abstractions, ''Empirical'' - if the authors designed a model-free approach and their decision process is based on the collected static or runtime data, and ''Hybrid/Other'' - if the authors used a combination of model-based and empirical approach or other methods.

C. CONTRIBUTIONS AND ORGANIZATION

Summarizing the core contributions of this paper are as follow:

- We survey recent works in the area of quality-aware DevOps, outlining the main contributions and comparatively position them to other DevOps works within the same field.
- We organize the surveyed papers into different categories, both globally across the survey and locally within each research area, offering a better qualitative understanding of the areas of main interest and current research gaps.
- For each category, we identify open research challenges, offering several ideas for further exploration by researchers in upcoming years.
- We outline open research directions and ongoing work in emerging DevOps trends related to the use of AI technology, which we expect to foster novel solutions in the quality engineering space in the near future.

The rest of the papers is organized as follows. Sections 2-8 survey recent research work across the considered areas, namely: architecture design (§2), model-based DevOps (§3), CI/CD (§4), testing (§5), verification (§6), and runtime management (§7). Each section outlines context, summarizing research papers, and giving guidance for future work. Section 8 discusses ongoing work in *DevOps for AI* and in *AI for DevOps*. Section 9 draws conclusions.

II. ARCHITECTURE DESIGN

Context: The architecture design of today's software systems, particularly cloud applications, allows for a rapid extension of features and functions with minor modifications to existing implementations. This requires increased communication and collaboration between development and operation teams to achieve a strong integration of coding, building, testing, packaging, releasing, configuring and monitoring activities. Therefore, designing the software architecture has profound implications not just on the software, but also on the overall DevOps delivery process. Recent research on architecture design in the context of DevOps centers around the following challenges:

AD1 Refactoring monolithic applications into microservices;
AD2 Modeling the architectures of cloud-native applications;
AD3 Deciding architecture design variants through testing or experimentation;
AD4 Adopting new architectural styles with best practices and tactics.

Quality-Aware DevOps Research: By migrating a mobile-app backend to microservices, Balalaie *et al.* [5] show how the microservices architecture could be beneficial, especially in shipping new features and providing built-in scalability. They also report on architectural patterns observed in migration projects, which can help practitioners and consultants to address **AD1** with a DevOps methodology.

Two papers are found to target **AD2**. Di Nitto *et al.* [48] introduce SQUID, a framework that provides DevOps-ready software architecture descriptions through model-based documentation of software architectures and their quality properties in DevOps scenarios. Meanwhile, Heinrich *et al.* [49] propose iObserve, an approach to enriching and updating

TABLE 1. Mapping of the reviewed paper with the quality attributes. The corresponding section of the papers are also provided.

Paradigm / Quality Attr.	**Model-based**	**Empirical**	**Hybrid/Other**
Maintainability / Transparency / Evolution		§II: [5]–[7]	§III: [8], [9]
Performance / Scalability	§III: [10]–[14] §IV: [15] §V: [16]–[18] §VII: [12], [27]–[31]	§II: [5]–[7], [19], [20] §V: [21] §VII: [22]–[24]	§VII: [25], [26]
Reliability / Fault-tolerance	§III: [11], [32], [33] §V: [34]–[36]	§II: [6]	
Safety / Correctness	§VI: [37], [38]	§VI: [39]	§IV: [40] §VI: [41]–[44]
Survivability / Security	§IV: [45] §V: [47]	§IV: [46]	
Cost of ownership	§III: [10], [11]	§II: [20]	
Any quality attribute	§II: [48]–[50]	§II: [51]	

the architectural development models of cloud-based software applications with operational observations so that the resulting architectural runtime models are usable during the operation phase.

In order to tackle **AD3**, Avritzer *et al.* [19] introduce an approach to automatically assessing the scalability of configuration alternatives for the microservices architecture through load testing. This approach provides a domain-based metric that can be used to make informed decisions about which configuration alternative to select. By contrast, Jiménez *et al.* [50] proposes a framework for quality-driven adaptive continuous experimentation. This framework dedicates three feedback loops to control the satisfaction of high-level quality goals through experiment design and conduct experimental trials for infrastructure configuration and architecture design variants.

Reference [6], [20], [51] and [7] provide solutions to **AD4** for the microservices architectural style by practicing DevOps in industrial use cases. Using OpenStack as case study, the authors of [20] compares the efficiency of DevOps in container-based and VM-based deployments and explores the scalability of stateless and stateful containerized components. Reference [51] discusses DevOps practices and architecting tactics for developing large-scale systems, like a Neo-Metropolis BDaaS platform. The authors of [6] show how the properties of the microservices architecture facilitate the scalability, agility and reliability of e-commerce applications. Differently, [7] advocates the use of microservices for software development in connected car business and proposes a suitable team setup for establishing a DevOps culture.

Analysis of Open Challenges: DevOps architectural work has focused on microservices, but novel research opportunities arise to extend this research, for example by including in architecture design also Function as a service (FaaS) elements. FaaS refers to a novel serverless computing paradigm that may radically evolve the landscape of software architectures. It enables software engineers to virtualize the business logic of an application as individual functions registered in the cloud. Because of advantages brought by the serverless FaaS paradigm, software vendors tend to migrate their existing products onto FaaS platforms, e.g., AWS Lambda and OpenFaaS. There is however a lack of approaches available in the literature to automatically decomposing monolithic applications into architectures containing serverless functions. Moreover, the choice of a suitable architectural granularity is an open problem, e.g., when to prefer a serverless function to a microservice.

III. MODEL-BASED DevOps

Context: Modeling provides a flexible and efficient means to study the qualitative and quantitative properties of a given system in an abstract language, thus being widely applied in support of various development and operation activities. As aforementioned, it can help with the architecture design of modern software systems [48], [49]. Models can either take the form of mathematical models or code in a domain-specific language to declare properties. Within this trend, models may refer to the system or its environment. In the former case, they provide abstractions for the software inter-dependencies or dynamic behavior. In the latter case, they provide abstractions to specify and automate the configuration of a target deployment environment.

This section focuses on model-based DevOps frameworks and methods that cope with the following challenges:

- **MD1** Assessing the quality of systems under development;
- **MD2** Optimizing system configurations in a cloud environment;
- **MD3** Specifying the required underlying infrastructure as code.

Quality-Aware DevOps Research: As a requisite for quality assurance, **MD1** receives continuous attention from the research community. Gorbenko *et al.* [33] provide a time-probabilistic failure model for distributed systems that follows the service-oriented paradigm to define interaction with clients over the Internet and clouds. A three-layer queueing network is proposed by Barna *et al.* [12] for developing autonomic management systems. This model is proven to be robust and accurate in predicting system performance under a variety of workloads and topologies. Take a tax fraud detection system as an example, Perez-Palacin *et al.* [13]

TABLE 2. Overview of publications on architecture design.

Ref. \ Aim	Refactoring	Modeling	Decision	Adoption
Balalaie et al. [5]	✓			
Di Nitto et al. [48]		✓		
Heinrich et al. [49]		✓		
Avritzer et al. [19]			✓	
Jiménez et al. [50]			✓	
Kang et al. [20]				✓
Chen et al. [51]				✓
Hasselbring et al. [6]				✓
Schneider [7]				✓

TABLE 3. Summary of publications on model-based DevOps.

Ref. \ Aim	Assessment	Optimization	Specification
Anatoliy et al. [33]	✓		
Barna et al. [12]	✓		
Perez-Palacin et al. [13]	✓		
Peuster and Karl [14]	✓		
Guerriero et al. [10]		✓	
Suk et al. [32]		✓	
Sun et al. [11]		✓	
Rahman et al. [8]			✓
Tamburri et al. [9]			✓

show the use of Petri nets for the performance analysis of data-intensive applications. Peuster and Karl [14] present an automatable and platform-agnostic modeling approach that can profile the performance of an entire service function chain at once.

MD2 is a common issue that needs to be addressed in the deployment and operation phases so as to adapt DevOps for a cloud environment. Guerriero *et al.* [10] propose SPACE4Cloud, an integrated framework for the deployment optimization and resource allocation of cloud applications represented as PCM models. A proactive application placement algorithm is introduced by Suk *et al.* [32]. This algorithm uses failure indexes evaluated by modeling application turnover and infrastructure failure as stochastic processes. Sun *et al.* [11] present a stochastic model and an optimization method to minimize the completion time, availability degradation, and monetary cost of the rolling upgrade procedure through appropriate parameterization.

The emergence of Infrastructure-as-Code (IaC) is a response to **MD3**. IaC often relies on textual resource models to configure the application environment. It is especially useful to increase the repeatability of configuration tasks in distributed architectures, where many dependencies exist between software components and virtualized resources. Comparatively to other areas surveyed in this paper, IaC quality research is in its infancy and relatively few works exist. Two representative examples are [8] and [9]. In [8], the authors discuss a qualitative analysis of over 1700 IaC scripts to identify code smells in IaC code, i.e., code snippets that are indicative of some deeper violation of design principles or best practices. The paper considers in particular security smells related to cryptography, authentication and hard-coded secrets, among others. The authors of [9] explore the use of intent modeling as a way to ensure the correctness of IaC. This is based on the idea of specifying IaC in terms of the high-level final state/goal that needs to be reached, operating at a higher level of abstraction than detailed sub-activities. The paper focuses on the standardized TOSCA language, which offers an implementation of this approach. TOSCA models are also compatible with the execution of IaC scripts in languages such as Ansible, effectively offering polyglot IaC. Rahman *et al.* [52] provides a systematic survey of quality in IaC, noting its current underdevelopment in the research literature. They carry out an analysis of 32 publications. Within this paper, IaC work insists primarily on quality dimensions such as the reliability, repair, testing, idempotency of IaC scripts.

Analysis of Open Challenges: Although the serverless FaaS paradigm simplifies user involvement in resource allocation at runtime and saves operating costs by billing executions at the function level. It is often difficult to decide the optimal configuration of serverless functions comprising an application, which minimizes the operating costs while satisfying the performance requirements. This raises the needs for models that can accurately predict the performance of FaaS-based applications as well as approaches that can effectively optimize their deployment. In the literature, no work seems to have been carried out for either.

Our survey also reveals that IaC research is still at an early stage, and thus many outlets for research exist in developing tools to increase the quality of IaC artifacts. Because IaC scripts can be specified using model-based declarative languages (e.g., TOSCA) or be written with specialized languages (e.g., Ansible), a research question is how to develop holistic and polyglot defect prediction and debugging environments for IaC. Another potential aspect to consider in IaC involves the quantification of costs associated to maintenance and configuration operations. At present such costs can only be indirectly estimated from execution logs, but there is practical business value in estimating these figures from code or software artifacts. Lastly, the relative merits of different IaC technical approaches used in industry are currently not well understood in the literature. More systematic investigations on these matters may be relevant to researchers.

IV. CONTINUOUS INTEGRATION AND DELIVERY (CI/CD)

Context: CI/CD pipelines provide the technical means to automate recurring tasks related to deployment, testing, and orchestration of cloud native applications. Market solutions such as Jenkins, CircleCI, Trevis, and several others can be used to coordinate delivery and quality checks on the application source code and associated software artifacts, prior to their production use. Several books and papers overview the general properties of CI/CD, e.g., [1] discusses the broader applicability and benefits of CI/CD in industrial context. More recently, [2] provides a large-scale empirical study on the impact of continuous integration on software development practice. Strategies to concretely adopt CI/CD in organizations are exemplified in [53].

TABLE 4. Comparing publications on CI/CD.

Attrb. / Ref.	SLA-aware	Security-aware	Data-aware
R. Meissner et al. [40]			✓
M. Mazkatli et al. [46]		✓	
F. Boyer et al. [15]	✓		
N. Ferry et al. [45]		✓	

Quality-Aware DevOps Research: It is well known that CI/CD finds immediate application to quality assurance via unit testing of functional properties. For example, the practitioner interviews in [54] reveal that deployability via CI/CD and architectural design to improve test quality are relevant dimensions in DevOps. We point the reader to a broader overview of related testing research in Section V. We instead here focus on innovative uses of CI/CD in the context of quality assurance, which offer novel outlets for research. Research in quality-aware CI/CD has centered on the following challenges:

CC1 Performance-aware CI/CD
CC2 Data-aware CI/CD
CC3 Secure CI/CD

An example of work that addresses **CC1** is [15], which uses CI/CI to ease updates while releasing new versions of microservices. The authors propose an architecture-based CI/CD approach, rather than using scripts, to update the microservices while in production. They define templates for different architectural models based on which an application can be updated to a target architecture using simple commands. In addition, they incorporate common update strategies, such CleanRedeploy, BlueGreen, Canary, etc., from which an appropriate strategy can be selected to satisfy specific SLA requirements. They demonstrate the effectiveness of the approach updating an application in production.

Another example of work in the context of **CC1** is [46], where the authors propose a roadmap to apply CI/CD to incrementally maintain and parameterize application performance models. The approach entails to react to changes in the application source code and then apply targeted monitoring and statistical estimation methods to update resource demands, probabilities of selecting particular code branches, loop execution numbers, and other relevant parameters. Such updates are essential to continuously evolve the quality-aware toolchain analysis synchronously with code commits.

An instance of work that tackles **CC2** is as follows. Data stores based on query languages such as RDF can be directly stored on systems such as Github, allowing to coordinate the publishing of data with CI/CD pipelines. This triggers the question on what is the inter-play between CI/CD and data quality engineering. [40] provides an overview of tools that can be integrated in CI/CD pipelines to continuously meet quality requirements on data. The include utilities for RDF serialization quality checks, ontology validation tools, data anti-patterns, linked open quality data assessment. An example of CI pipeline to holistically coordinate the surveyed data quality assurance tools is described in the paper.

The recent work in [45] illustrates another approach to quality-aware DevOps, where the goal of the study is to continuously integrate and orchestrate a system so to ensure security and privacy. This aligns to challenge **CC3**. Model-driven engineering methods are coupled with secure DevOps practices to allow continuous changes in the deployment environment. This is based on a so-called ‘‘models@runtime’’ approach, where the application model is evolved directly in the production environment in response to dynamic events that occur therein.

Analysis of Open Challenges: The above papers exemplify novel trends in CI/CD towards integrating in the CI/CD pipeline the specification of data services. It is possible to envision that similar needs will arise in connection with AI/ML services, which require a continuous evolution of data pipelines, learning and training services alongside the application. CI/CD support specific to such kind of services offers outlet for novel research.

V. DevOps TESTING

Context: DevOps has been widely adopted in enterprises, which leads to shortened development cycles and involvement of automation. With speedy iterations, the risk and cost of quality assurance increase at the same time. Testing is of great importance to ensure the quality of software in DevOps practice. In particular, automating the testing process enables continuous testing of the frequent code changes occurring throughout the development cycle. The following challenges are highlighted in the literature:

TS1 Automatic workload selection and specification of target services.
TS2 Test automation frameworks to enable automatic execution within DevOps cycles.
TS3 Employing testing strategies to adapt to frequent changes in DevOps.

Quality-Aware DevOps Research: In the phase of test specification, a central goal is to maximize the coverage of new changes and specify unit tests aiming at specific and identifiable target services or functions, to make test results quickly actionable for developers. Besides common functional testing, which is not specific of DevOps, automated load testing allows spotting possible performance issues during the integration phase, preventing them from manifesting in production. In [16], Schulz *et al.* propose an approach of load testing selection based on contextual information that focuses on **TS1**. Workloads can be automatically selected according to monitoring data, target services, along with testing requirements in the proposed load testing process.

The authors in [17] focus on representative workload models for load testing of individual microservices in session-based systems. Two algorithms are proposed to enable extracting specific workload for the microservices under testing and consequent adjustments of workload models. Such an approach aims at only target microservices and their

TABLE 5. Comparing publications on DevOps testing.

Phase / Ref.	Specification	Modeling	Execution
Schulz et al. [16]	✓	✓	✓
Schulz et al. [17]	✓	✓	
Schulz et al. [18]	✓	✓	
Ferme et al. [34]	✓	✓	
Pietrantuono et al. [35]			✓
Schermann et al. [21]			✓
Schermann et al. [36]	✓		✓
Dullmann et al. [47]		✓	✓

dependencies so that they can reduce the testing cost, addressing the issue of **TS1**.

In [18], the authors also propose to solve **TS1** by introducing a behavior-driven load testing language (BDLT), which is designed to describe performance concerns in natural language that can be easily adopted by users. Based on BDLT, testing workloads can be automatically generated with the method in [16].

To enable the automated execution of tests for the purpose of faster integration and delivery, several test automation frameworks and tools have been proposed to address the problem of **TS2**. For example, in [34], the authors address automated testing workflows in the process of continuous integration, focusing on unit tests and integration tests. Pietrantuono *et al.* in [35] present a continuous software reliability testing approach called DevOpRET. This approach mainly involves usage monitoring and operational profile estimation and updating. By monitoring the endpoint users, estimated operational profile is able to be updated with the actual user profile. At each DevOps cycle, the reliability testing can be executed based on the continuously updated operational profile.

In addition, testing techniques such as canary releases and shadow/dark launches are being increasingly adopted as strategies to automate testing execution in **TS3**. In an empirical study on continuous experimentation [21], the authors provide an overview of continuous experimentation practices that contain canary releases, dark launches and A/B testing in both research and practice of DevOps. In [36], Schermann *et al.* proposed a live testing model and implemented a middleware, Bifrost, to specify testing strategies and execute tests through traffic routing. Bifrost is able to describe release techniques with multiple phases including canary releasing, dark launching, A/B testing and gradual rollout in YAML-based language. The authors of [47] address the issues of dependability and security of CD pipelines, proposing involving testing strategies, such as canary releases and A/B testing, into building and integration pipelines execution.

Analysis of Open Challenges: Rapid changes bring new challenges to test specification and execution in practice. Learning and analyzing the internal and external dependencies between components at specification stages could also inform test specifications, enabling a more effective identification of tests that need to be re-executed after a change, trading off test complexity for coverage. In addition, test automation needs to meet the dynamics of iterations in DevOps cycles. For example, the objective of each iteration may change, which will lead to involve different test strategies into the respective iteration to meet the QA requirements.

VI. VERIFICATION IN DevOps

Context: Verification complements testing in software quality assurance processes. Compared to testing, it leverages mathematical abstractions and code/model semantics to prove the (partial) correctness of artifacts with respect to a variety of properties. While still in their infancy, verification methods tailored to DevOps are gaining traction in both industry and academia for their potential to deliver stronger quality guarantees than testing. This section reviews a selection of paper relevant to verification in DevOps (Table 6). The main open challenges discussed through this section are summarized in the following points:

VE1 Develop diff-time verification methods for prompt and localized feedback to keep developers engaged
VE2 Increase compositionality and incrementality to support the analysis of large, rapidly changing code bases
VE3 Feed information from design time to runtime and viceversa to improve runtime tasks and verification

Quality-Aware DevOps Research: Despite not as widely adopted as testing, verification is applied at several stages of a DevOps cycle, including, in order: design, build, diff, land, and production times [55].

Design-time methods analyze pre-implementation software artifacts, including goal or architectural models from the DevOps plan and create phases. User-provided abstractions, e.g., statecharts or unambiguous dialects of UML, are automatically translated into formal models to verify artifacts' properties. For example, analytical models of performance and reliability obtained from higher-level modeling languages like the Palladio Component Model [56] or OASIS TOSCA can be analyzed with numerical routines or probabilistic model checking [57].

Build-time methods are usually embedded within compilers and IDEs, providing quick feedback to the developers about the module they are implementing. These are usually light-weight static analyses performed with tools like Valgrind [41] and ASan to detect buffer overflows or dangling pointers and profile C/C++ artifacts, or FindBugs to localize several classes of bugs in Java artifacts [42]. While most of these methods are not specific to it, DevOps needs started to push for adapting them into staged analyses, where static code information computed during build time are carried on to later development stages and runtime to enable subsequent analyses [58]. For example, in [39], Beigi-Mohammadi *et al.* exploit control flow analysis to extract security-related predicates to be checked during operation, enabling automatic adaptation actions for early countering potential attacks.

Diff-time is the gatekeeping at the end of code creation, when submitted code waits for review and approval. Verification methods in this phase usually completes within a few

TABLE 6. Verification and static analysis through DevOps phases.

Ref. \ Phase	Design/ build-time	Diff time	Land time	Runtime
Koziolek et al. [56]	✓			
Brunnert et al. [57]	✓			
Nethercote et al. [41]	✓			
Sadowski et al. [42]	✓	✓		
O'Hearn [55]		✓		
Backes et al. [37]		✓		
Larus et al. [44]		✓	✓	
Harman et al. [43]		✓	✓	
Beigi-Mohammadi et al. [39]	✓			✓
Filieri et al. [38]				✓

tens of minutes [55] to allow their reports to complement human code reviews. The peculiarity of this phase is its intrinsic incrementality: only portions of an artifact change since the last run of the analysis, and they can be identified by the diff. While few academic tools specialize for diff-time analysis, notable industrial contributions include Facebook's Infer [43], Amazon's s2n [37], and Microsoft' Prefast [44].

Land-time occurs after a diff is approved and before release to production. This phase is allowed longer execution time (typically from hours to overnight) and can operate from built and executable modules, which can be analyzed both statically and dynamically [43]. Microsoft Prefix [44] is an example of tools used in this stage.

Finally, in production, *runtime* verification methods can be used to detect requirements violations as they happen. These methods require the instrumentation of the application with monitors and probes to measure specific quantities or detect the violation of safety/security predicates [39], [57]. Methods based on partial evaluation compute surrogate model of the system that enable efficient verification at runtime, after current monitoring information are gathered (e.g., [38] verifies probabilistic properties).

Analysis of Open Challenges: Broadly speaking, verification requires formal models and analysis algorithms. Model-driven processes exploit human ingenuity to produce semantically richer models of an application and its environment. While these models allow the use of established model checking algorithms, keeping the models consistent with the application code may be challenging in all but the few domains where fully automated code generation is possible. Nonetheless, where available, even partial design models should be used in the future to improve the effectiveness of later-stage methods. This includes, for example, the contextualization of build-time verification within realistic usage profiles specified by the designers, as well as using design models to narrow down the relevant scenarios for diff-time and runtime analysis, reducing the relevant search space to cut verification time (V3). To improve diff-time verification, research has to focus on compositional methods which enable incremental re-analysis of only the changed parts of a codebase [55]. This overall addresses challenge **VE1**. Academic research largely underestimated so far the importance of prompt and localized developer feedback, preferring detailed verification reports produced overnight at land-time. However, empirical evidence from industry suggests that diff-time verification is more effective for bug fixing and keeps developers more engaged [42]. Developing compositional verification algorithms often requires to reduce the expressiveness of verifiable properties, which may nonetheless allow to intercept problems before they reach production, which falls under challenge **VE2**. Adequate design-time models can also help to narrow down the state space to be verified at later process stages, bringing a global view hard to infer from lower level artifacts like code or binaries. Finally, runtime verification methods, whether measurement/probing based or model-based, have the potential of observing the application within its actual execution environment, which may differ from design-time assumptions or land-time simulations. This addresses challenge **VE3**. The ability to promptly detect issue while the application is running can reduce the exposition time to a bug, but also enable automatic adaptation actions to self-protect an application or its infrastructure.

VII. RUNTIME SERVICE MANAGEMENT

Context: Runtime service management particularly concerns dynamic resource scheduling of microservices. Microservices is one of the core DevOps practices. It is a design principle to build applications with fine-grained services. One of the benefits arising from this is the ability to manage each service individually. However, regardless of this benefit, managing microservices at runtime is not trivial. This involves multiple research challenges, regarding monitoring, configuration options, decision making, etc. In a nutshell, the following challenges were highlighted by the researchers:

RM1 Monitoring microservices
RM2 Container placement strategy
RM3 Autoscaling microservices

Quality-Aware DevOps Research: **RM1** can be considered as an ensemble of complex sub-problems. Researchers often focus on these sub-problems rather than the overall issue. For example, Noor *et al.* [25] focused on the issue of collecting data from heterogeneous virtualization architecture. They have developed a framework M3 using the SIGAR library and RESTful API that collects both the system and process level metrics through separate agents. On the other hand, Miglierina and Tamburri [26] have focused on reducing the complexity of monitoring configuration management. They proposed Omnia that addresses this issue through Monitoring Configuration as Code. This is realized by defining a set of vocabulary and protocols, which are used to setup and update the monitoring configurations for popular tools like InfluxDB, Prometheus, Grafana, etc.

RM2 is often addressed based on the scale of the computing environment. For example, in [22], Boza *et al.* proposed kube-scheduler to address RM2 in a typical multi-server setup. kube-scheduler is a performance-aware orchestrator, based on Kubernetes, that take container placement decisions by considering the number of available CPUs in the host machine and how they effect the runtime and initialization time performance. However, considering a geo-distributed environment like edge or fog, the placement approach needs to be adapted to incorporate information on heterogeneous computing environment. Rossi *et al.* [27] focused on this issue and proposed ge-kube. Along with the placement issue, ge-kube focuses on the elasticity problem as well, thus addressing **RM3**. They resolved the placement issue by formulating it as an optimization problem and the elasticity issue is resolved using model-based Reinforcement Learning (RL).

Recently, compared to RM1 and RM2, **RM3** has gained more attention from the researchers. Multiple autoscalers have been proposed to address this issue, each differing on how the problem is perceived. In [23], Kwan *et al.* propose an autoscaler, HyScale, that focuses on the performance trade-offs between horizontal and vertical scaling. HyScale's principle is to scale vertically if the resources are available. If not, it performs horizontal scaling. Rossi *et al.* [28] also emphasize the use of both horizontal and vertical scaling. However, in contrast to [23], they adopted a model-based approach. It is based on a novel Reinforcement Learning model that relies on approximations (state transition probabilities and the associated costs) from monitoring data. They realized the so-called Elastic Docker Swarm (EDS) by integrating their method with Docker Swarm.

Another context in **RM3** is coordinated scaling to solve the bottleneck shift problem. Bauer *et al.* [29] present Chamulteon that focuses on that issue. It is based on queueing models which are used to forecast system performance and taking a coordinated scaling actions. Chamulteon also includes a workload forecasting component, which makes it proactive. Barna *et al.* [12] propose an Autonomic Management System (AMS) based on Layered Queueing Network (LQN) that inherently offers coordinated autoscaling. However, Chamulteon and AMS both do not consider vertical scaling. In [30], Gias *et al.* present an autoscaler, ATOM, that supports both horizontal and vertical scaling along with coordinated autoscaling. Similar to AMS, ATOM is based on LQN models but it considers both a microservice CPU share (vertical scaling) and the number of its replicas (horizontal scaling) during performance forecasting.

In [24], Qiu et. al present FIRM that focuses on fine grain resource management of microservices considering resources like cache, network bandwidth, CPU, memory, etc. However, unlike most of the approaches, it opted for a model-free method. Their approach relies on a combination of support vector machine and Reinforcement Learning to identify and allocate resources to bottleneck microservices. Rossi *et al.* highlight another important issue - decentralizing the autoscaler components and propose a hierarchical autoscaler me-Kube [31]. Such decentralization makes the autoscaler more scalable when deployed in a large cluster. Although me-Kube uses queueing models, it does not support coordinated scaling as they model each microservice separately rather than the overall application.

TABLE 7. Comparing different autoscalers for microservices.

Attrb. Ref.	Model	Vertical	Proactive	Coordinated
Kwan et al. [23]		✓		
Rossi et al. [28]	✓	✓		
Bauer et al. [29]	✓		✓	✓
Barna et al. [12]	✓			✓
Gias et al. [30]	✓	✓		✓
Rossi et al. [27]	✓	✓		
Qiu et. al. [24]		✓		✓
Rossi et al. [31]	✓		✓	

A comparison of these autoscalers, based on different attributes (model-based, supporting vertical scaling, proactive, coordinated scaling), are presented in Table 7.

Analysis of Open Challenges: Regardless of the progress made, there are still multiple research challenges concerning runtime service management. A major challenge concerning **RM1** is providing support for the model-based approaches. Thus, a monitoring framework for microservices should focus on providing metrics related to queueing or machine learning models, like queue length, arrival rates, transition probabilities, etc., to improve the estimates of different model-based runtime controllers. In addition, they can also leverage machine learning techniques to provide insight of a system architecture such that a model can be automatically generated.

For a runtime controller, focusing on **RM2**, a major challenge is to forecast the performance of container groups rather than a single container. A container group can represent a chain of microservices. Considering a single container alone will only provide a partial view of performance in that particular cluster node. On the other hand, the controllers focusing on **RM3**, particularly the model-based ones, should emphasize faster decision making. This issue can be solved by being proactive but that requires a huge volume of data for accurate forecasting. Thus, researchers should investigate the effectiveness of hybrid autoscalers that combines proactive, simple reactive and model-based reactive approaches.

VIII. EMERGING TRENDS: BRIDGING AI/ML, BIG DATA, AND DevOps

Artificial Intelligence (AI) and machine learning (ML) algorithms are being increasingly used by industry for monitoring and development to boost performance. These techniques offer the ability to quickly learn the pattern of baseline performance from a large space of performance metrics to diagnose system issues. AI/ML can play a crucial role in accelerating DevOps efficiency for today's dynamic and distributed data-intensive environment. The future of DevOps will be AI, ML, data-intensive driven, which offer potential benefits to enhance functionality and transform how system developers and administrators can design, test, deploy, and maintain systems. Monitoring the modern DevOps environment involves a high level of complexity that AI/ML techniques can alleviate. Dealing with Exabytes of data to investigate the root causes analysis using conventional DevOps solutions may lead to unexpectedly long time to identify the reason of failures within complex distributed systems.

AI/ML solutions for DevOps are utilized to play a significant role in automating and enhancing DevOps processes. Sun *et al.* [59] propose a non-intrusive automated fault diagnosis for public cloud (such as Amazon Web Service) and rolling upgrade DevOps operation using system logs and machine learning algorithm. They use performance metrics that are collected during monitoring time to train the ML classifier for detecting issues and to expect behavior over every time interval within the system. Their proposed approach achieves on average 90% for recall and precision. The study in [59] demonstrates that using ML for fault diagnoses within DevOps operations (such as rolling upgrade) is promising.

Real challenges for building AIOps solutions are presented by Dang *et al.* [60] based on practices within Microsoft. The term AIOps comes from Gartner to address the challenges of DevOps using AI. Dang *et al.* [60] mention that AIOps is about enabling software and system engineers to operate services efficiently using ML and AI solutions. The added values of AIOps includes: ensuring high service quality, offering high service intelligence, increasing engineering productivity, and decreasing operational cost. Around 60% of firms will adopt AI and ML analytics for DevOps by 2024 to accelerate service delivery, improve performance, and secure systems [60].

Nogueira *et al.* [61] review existent research that applies ML to optimize the quality of process within DevOps pipeline. ML techniques have the ability to provide insight into specific IT processes to effectively assist stakeholders in recognizing improvements that are needed within the software development life cycle. Kumar *et al.* [62] present Sankie, which is an AI Platform for Azure DevOps which is a scalable and general service that is developed to assist and impact all stages of the modern software development life cycle. The proposed AI platform can provide smart and actionable recommendations to system developers and administrators, which include training, recommending, explaining, and evaluating. The proposed platform is used at Microsoft and is enabled for over 50 repositories internally.

There are some DevOps solutions for AI/ML. While AL/ML offers valuable benefits to DevOps, there are some existing DevOps solutions for AL/ML stakeholders that help in developing continuous efficient AL/ML services. Ciucu *et al.* [63] develop a software architecture solution that can ensure the continuous development of computer vision applications. They examine high-performance computing and GPU resource management for model implementation within data centers to enhance the integration process and performance optimization. The integration covers software services and microservices to orchestrate the containers within systems to high availability services.

Palacin *et al.* [13] present a DevOps industrial application that focuses on software quality evaluation tools for tax fraud detection in the context of improving the quality and reliability of Big Data. During development iterations, the impact of quality assessment is reported with a particular focus on the accomplishment of performance requirements during the continuous adding of new functionalities to systems. The authors in [13] target applications that manage billions of invoice records. The evaluation is conducted using simulation (SimTool), which is developed by DICE European project developers for quality analysis. The goal is to reduce the number of DevOps iterations.

TABLE 8. Taxonomy for AI/ML, Big Data, and DevOps.

Target / Ref.	AI for DevOps	DevOps for AI	Big Data
Sun *et al.* [59]	✓		✓
Dang *et al.* [60]	✓		✓
Nogueira *et al.* [61]	✓		
Kumar *et al.* [62]	✓		
Ciucu *et al.* [63]		✓	
Palacin *et al.* [13]			✓
Fox *et al.* [65]			✓
Di Nitto *et al.* [48]			✓
Chen *et al.* [64]			✓

Regarding software architecture, Di Nitto *et al.* [48] investigate concerns and obstacles, which are needed to be tackled in DevOps scenarios. The authors in [48] present Specification Quality In DevOps (SQUID), which is a framework for software architecture. The proposed framework is evaluated in the Big Data domain. SQUID is evaluated on a real industrial DevOps scenario, to find SQUID's pros and limitations.

Chen *et al.* [64] contribute to the field of Big Data and DevOps by presenting a methodology revolve around architecture-centric Agile Big data Analytics (AABA), which is evaluated on many Big Data analytics projects in security, cloud-based mobile, healthcare, etc. The authors [64] conclude that architecture agility has a significant impact on the rapid continuous delivery within Big data intensive applications. Finally, it is obvious that AI/ML will play a crucial role in improving DevOps productivity for future dynamic and distributed data-intensive systems. The future of DevOps will be AI, ML, data-intensive driven that offer potential advantages to improve functionality and transform how system developers and administrators can design, test, deploy, and maintain systems.

Analysis of Open Challenges: While AI/ML clearly provides valuable benefits to DevOps, there are some potential challenges that may arise in the future. This is because they are fundamentally different from conventional applications, and it is crucial to take into account that they have a different development lifecycle. Another well-known challenge of AI/ML is the availability of sufficient real-world datasets to build, train, and test the model before deploying it into the real production environment. In addition, the characteristic of the system may continuously change, which make the AI/ML model fail to be generalized from datasets that are used for training purposes. Therefore, AI/ML requires continuous model evaluating, tuning, retraining, and retesting.

A team from Microsoft illustrates that the data used in AI systems are large, specific for each context, and complicated for explaining and becoming a burden. These factors

TABLE 9. Summary of challenges and current contributions.

Challenges	Reference	Domain	Scope / Results
AD1	Balalaie et al. [5]	methodology	Benefits of the microservice architecture and architectural patterns for migration.
AD2	Di Nitto et al. [48]	specification	`SQUID` - A framework for modeling software architectures and their quality properties.
	Heinrich et al. [49]		`iObserve` - An approach to reusing architectural development models during the operation phase.
AD3	Avritzer et al. [19]	configuration	An approach to the scalability assessment of the microservices architecture.
	Heinrich et al. [49]		A framework that enables quality-driven adaptive continuous experimentation.
AD4	Kang et al. [20]	exercise	Solutions to design cloud infrastructure based on containerized microservices.
	Chen et al. [51]		DevOps practices and architecting tactics for developing large-scale systems.
	Hasselbring et al. [6]		A use case of the microservices architecture in e-commerce platforms.
	Schneider [7]		A use case of the microservices architecture in connected cars .
MD1	Gorbenko et al. [33]	analysis	A time-probabilistic failure model for service-oriented distributed systems.
	Barna et al. [12]		A three-layer queueing network for autonomic management systems.
	Perez-Palacin et al. [13]		Petri nets to analyze the performance of data-intensive applications.
	Peuster and Karl [14]		An approach to profiling the performance of an entire service function chain.
MD2	Guerriero et al. [10]	configuration	`SPACE4Cloud` - An framework that enables deployment optimization and resource allocation on clouds.
	Suk et al. [32]		An algorithm for application placement in a high turnover environment.
	Sun et al. [11]		A method for optimizing the parameters of the rolling upgrade procedure.
MD3	Rahman et al. [8]	automation	An analysis of security-related code smells over massive IaC scripts.
	Tamburri et al. [9]		An approach to ensuring the correctness of IaC code through intent modeling.
CC1	Mazkatli and Koziolek [46]	integration	An approach in an agile development process for integrating application models with CI/CD.
CC2	Meissner and Junghanns [40]		An overview of tools that can be integrated in CI/CD pipelines to continuously meet quality requirements on data.
CC3	Ferry et al. [45]		Model-driven approach to continuously integrate and orchestrate a system so to ensure security and privacy.
TS1	Schulz et al. [16]	specification	Representative workload models and automatic workload selection based on contextual information.
	Schulz et al. [17]		Specific workload selection for target microservices in load testing.
	Schulz et al. [18]		`BDLT` - Behavior-driven load testing language to describe performance concerns.
TS2	Ferme et al. [34]	automation	Automated testing workflows for CI/CD.
	Pietrantuono et al. [35]		`DevOpRET`- A continuous software reliability testing approach.
TS3	Schermann et al. [21]	execution	Continuous experimentation practices with testing strategies.
	Schermann et al. [36]		`Bifrost` - A living test model with testing strategies and automatic execution.
	Dullmann et al. [47]		Canary releases and A/B testing strategies in CD pipelines.
VE1	Sadowski et al. [42]	development / verification	Diff-time verification methods to include formal methods within code-review processes, providing developers with local and contextualized feedback.
VE2	O'Hearn [55]	verification	Compositional verification to reuse evidence and lemmas computed from previous versions of the application to achieve efficiency via incrementality on new versions
VE3	Filieri et al. [38]	verification / monitoring	Improved verification efficiency at runtime via design-time partial evaluation and pre-processing
RM1	Noor et al. [25]	monitoring	`M3` - A decentralized monitoring framework for heterogeneous cloud.
	Miglierina et al. [26]		`Omnia` - A monitoring configuration approach based on the concepts of IaC.
RM2	Boza et al. [22]	placement	`kube-scheduler` - A performance-aware container orchestrator based on Kubernetes.
	Rossi et al. [27]		`ge-kube` - A container orchestrator for geographically distributed clusters
RM3	Kwan et al. [23]	autoscaling	`HyScale` - A threshold-based autoscaler that considers both horizontal and vertical scaling.
	Rossi et al. [28]		`EDS` - An autoscaler based on model-based RL for Docker Swarm.
	Gias et al. [30]		`ATOM` - An LQN-based autoscaler that aims to optimize the system throughput.
	Qiu et. al. [24]		`FIRM` - An autosaler supporting fine-grained resource management using model-free RL.
	Bauer et al. [29]		`Chamulteon` - A queueing model based horizontal autoscaler that includes workload forecasting.
	Barna et al. [12]		`AMS` - A horizontal autoscaler for containerized applications based on LQN.
	Rossi et al. [31]		`me-kube` - A hierarchical autoscaler based on model-free RL.

make it challenging to integrate AI model on a large scale and distributed system. Therefore, system engineers need to carefully collect and preprocess datasets before training and tuning AI algorithms to gain high accuracy performance. In addition, the collected data has to be efficiently stored and updated continuously with a predefined schema. Another challenge is that datasets' schema may frequently change in a real-time, which need to be resilient with continuous developments when new data is ingested from large scale systems with changeable performance characteristics [66].

When developing a large scale DevOps project, it is challenging to maintain modularity. This is because the AI models are developed in a separated and isolated environment to ensure the prevention of interference among systems components. These separated AI subsystems are developed by different engineering teams, where AI services communicate with other systems in non-obvious ways using a controlled API that has to be precisely controlled [67], [68]. This kind of challenge may cause error to be propagated among system and impact the overall performance of services. There is a need for a more advanced researches and effective solutions to continuous update of AI models and discover the unseen misconceptions among system components while taring data characteristics are changing.

IX. CONCLUSION

DevOps methods have reduced the cultural and methodological gap between developers and operators, which lead to the formation of many new organizational structures, such as virtual teams working on both development and operation tasks. This motivates the establishment of new professional figures, often referred to as DevOps engineers, who center their activity on tooling and automation across the whole application lifecycle. DevOps paradigm allocates more attention towards continuous re-release, unified tooling and organizational processes. Common DevOps advances include, for example, continuous-integration/continuous-delivery (CI/CD) pipelines and highly-automated orchestration solutions for the run-time environment.

This survey reviews recent research to support DevOps with quality-aware software engineering tools. The paper reviews the context in which research was carried out and reveals some gaps in areas such as continuous-integration/continuous-delivery (CI/CD), incremental verification, and infrastructure-as-code (IaC). Table 9 provides a summary of challenges and current contributions with the domain of quality-aware DevOps. Initial activity on the upcoming *AI for DevOps* and *DevOps for AI software* as also been surveyed, outlining possible directions for further work.

ACKNOWLEDGMENT

(Ahmad Alnafessah, Alim Ul Gias, Runan Wang, and Lulai Zhu contributed equally to this work.)

REFERENCES

[1] M. Virmani, "Understanding DevOps & bridging the gap from continuous integration to continuous delivery," in *Proc. INTECH*, May 2015, pp. 78–82. [Online]. Available: http://ieeexplore.ieee.org/document/7173368/

[2] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu, "The impact of continuous integration on other software development practices: A large-scale empirical study," in *Proc. ASE*. Urbana, IL, USA: IEEE, Oct. 2017, pp. 60–71. [Online]. Available: http://ieeexplore.ieee.org/document/8115619/

[3] J. Wettinger, U. Breitenbücher, and F. Leymann, "Devopslang-bridging the gap between development and operations," in *Proc. Eur. Conf. Service-Oriented Cloud Comput.* Berlin, Germany: Springer, Sep. 2014, pp. 108–122.

[4] L. J. Bass, I. M. Weber, and L. Zhu, *DevOps—A Software Architect's Perspective* (SEI Series in Software Engineering). Reading, MA, USA: Addison-Wesley, 2015. [Online]. Available: http://bookshop.pearson.de/devops.html?productid=208463

[5] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May 2016.

[6] W. Hasselbring and G. Steinacker, "Microservice architectures for scalability, agility and reliability in e-commerce," in *Proc. ICSAW*, Apr. 2017, pp. 243–246.

[7] T. Schneider, "Achieving cloud scalability with microservices and DevOps in the connected car domain," in *Proc. Soft. Eng. CEUR*, 2016, pp. 138–141.

[8] A. Rahman, C. Parnin, and L. Williams, "The seven sins: Security smells in infrastructure as code scripts," in *Proc. ICSE*, 2019, pp. 164–175.

[9] D. A. Tamburri, W.-J. Van den Heuvel, C. Lauwers, P. Lipton, D. Palma, and M. Rutkowski, "TOSCA-based intent modelling: Goal-modelling for infrastructure-as-code," *Softw.-Intensive Cyber-Phys. Syst.*, vol. 34, nos. 2–3, pp. 163–172, Jun. 2019.

[10] M. Guerriero, M. Ciavotta, G. P. Gibilisco, and D. Ardagna, "A model-driven DevOps framework for QoS-aware cloud applications," in *Proc. SYNASC*, Sep. 2015, pp. 345–351.

[11] D. Sun, S. Chen, G. Li, Y. Zhang, and M. Atif, "Multi-objective optimisation of online distributed software update for DevOps in clouds," *ACM Trans. Internet Technol.*, vol. 19, no. 3, pp. 1–20, Nov. 2019.

[12] C. Barna, H. Khazaei, M. Fokaefs, and M. Litoiu, "Delivering elastic containerized cloud applications to enable DevOps," in *Proc. SEAMS*, May 2017, pp. 65–75.

[13] D. Perez-Palacin, Y. Ridene, and J. Merseguer, "Quality assessment in DevOps: Automated analysis of a tax fraud detection system," in *Proc. 8th ACM/SPEC ICPE*, 2017, pp. 133–138.

[14] M. Peuster and H. Karl, "Profile your chains, not functions: Automated network service profiling in DevOps environments," in *Proc. NFV-SDN*, 2017, pp. 1–6.

[15] F. Boyer, X. Etchevers, N. De Palma, and X. Tao, "Architecture-based automated updates of distributed microservices," in *Proc. Int. Conf. Service-Oriented Comput.* Cham, Switzerland: Springer, Nov. 2018, pp. 21–36.

[16] H. Schulz, T. Angerstein, and A. van Hoorn, "Towards automating representative load testing in continuous software engineering," in *Proc. ICPE (Companion)*, 2018, pp. 123–126.

[17] H. Schulz, T. Angerstein, D. Okanović, and A. van Hoorn, "Microservice-tailored generation of session-based workload models for representative load testing," in *Proc. MASCOTS*, 2019, pp. 323–335.

[18] H. Schulz, D. Okanović, A. van Hoorn, V. Ferme, and C. Pautasso, "Behavior-driven load testing using contextual knowledge-approach and experiences," in *Proc. ICPE*, 2019, pp. 265–272.

[19] A. Avritzer, V. Ferme, A. Janes, B. Russo, A. V. Hoorn, H. Schulz, D. Menasché, and V. Rufino, "Scalability assessment of microservice architecture deployment configurations: A domain-based approach leveraging operational profiles and load tests," *J. Syst. Softw.*, vol. 165, Jul. 2020, Art. no. 110564.

[20] H. Kang, M. Le, and S. Tao, "Container and microservice driven design for cloud infrastructure DevOps," in *Proc. IC2E*, 2016, pp. 202–211.

[21] G. Schermann, J. Cito, P. Leitner, U. Zdun, and H. C. Gall, "We're doing it live: A multi-method empirical study on continuous experimentation," *Inf. Softw. Technol.*, vol. 99, pp. 41–57, Jul. 2018.

[22] E. F. Boza, C. L. Abad, S. P. Narayanan, B. Balasubramanian, and M. Jang, "A case for performance-aware deployment of containers," in *Proc. WOC*, 2019, pp. 25–30.

[23] A. Kwan, J. Wong, H. Jacobsen, and V. Muthusamy, "HyScale: Hybrid and network scaling of dockerized microservices in cloud data centres," in *Proc. ICDCS*, 2019, pp. 80–90.

[24] H. Qiu, S. S. Banerjee, S. Jha, Z. T. Kalbarczyk, and R. K. Iyer, "FIRM: An intelligent fine-grained resource management framework for SLO-oriented microservices," in *Proc. OSDI*. Berkeley, CA, USA: USENIX, 2020, pp. 805–825.

[25] A. Noor, D. N. Jha, K. Mitra, P. P. Jayaraman, A. Souza, R. Ranjan, and S. Dustdar, "A framework for monitoring microservice-oriented cloud applications in heterogeneous virtualization environments," in *Proc. CLOUD*, 2019, pp. 156–163.

[26] M. Miglierina and D. A. Tamburri, "Towards omnia: A monitoring factory for quality-aware DevOps," in *Proc. ICPE ACM/SPEC*, 2017, pp. 145–150.

[27] F. Rossi, V. Cardellini, F. L. Presti, and M. Nardelli, "Geo-distributed efficient deployment of containers with Kubernetes," *Comput. Commun.*, vol. 159, pp. 161–174, Jun. 2020.

[28] F. Rossi, M. Nardelli, and V. Cardellini, "Horizontal and vertical scaling of container-based applications using reinforcement learning," in *Proc. CLOUD*, 2019, pp. 329–338.
[29] A. Bauer, V. Lesch, L. Versluis, A. Ilyushkin, N. Herbst, and S. Kounev, "Chamulteon: Coordinated auto-scaling of micro-services," in *Proc. ICDCS*, 2019, pp. 2015–2025.
[30] A. U. Gias, G. Casale, and M. Woodside, "ATOM: Model-driven autoscaling for microservices," in *Proc. ICDCS*, Jul. 2019, pp. 1994–2004.
[31] F. Rossi, V. Cardellini, and F. L. Presti, "Hierarchical scaling of microservices in kubernetes," in *Proc. ACSOS*, 2020, pp. 28–37.
[32] T. Suk, J. Hwang, M. F. Bulut, and Z. Zeng, "Failure-aware application placement modeling and optimization in high turnover DevOps environment," in *Proc. CLOUD*, 2019, pp. 115–123.
[33] A. Gorbenko, A. Romanovsky, and O. Tarasyuk, "Fault tolerant Internet computing: Benchmarking and modelling trade-offs between availability, latency and consistency," *J. Netw. Comput. Appl.*, vol. 146, Nov. 2019, Art. no. 102412.
[34] V. Ferme and C. Pautasso, "Towards holistic continuous software performance assessment," in *Proc. ICPE (Companion)*, 2017, pp. 159–164.
[35] R. Pietrantuono, A. Bertolino, G. De Angelis, B. Miranda, and S. Russo, "Towards continuous software reliability testing in DevOps," in *Proc. AST*, 2019, pp. 21–27.
[36] G. Schermann, D. Schöni, P. Leitner, and H. C. Gall, "Bifrost: Supporting continuous deployment with automated enactment of multi-phase live testing strategies," in *Proc. ACM Middleware*, 2016, pp. 1–14.
[37] J. Backes, C. Varming, M. Whalen, P. Bolignano, B. Cook, A. Gacek, K. S. Luckow, N. Rungta, M. Schaef, C. Schlesinger, and R. Tanash, "One-click formal methods," *IEEE Softw.*, vol. 36, no. 6, pp. 61–65, Nov. 2019.
[38] A. Filieri, G. Tamburrelli, and C. Ghezzi, "Supporting self-adaptation via quantitative verification and sensitivity analysis at run time," *IEEE Trans. Softw. Eng.*, vol. 42, no. 1, pp. 75–99, Jan. 2016.
[39] N. Beigi-Mohammadi, M. Litoiu, M. Emami-Taba, L. Tahvildari, M. Fokaefs, E. Merlo, and I. V. Onut, "A DevOps framework for quality-driven self-protection in Web software systems," in *Proc. CASCON*, 2018, pp. 270–274.
[40] R. Meissner and K. Junghanns, "Using DevOps principles to continuously monitor RDF data quality," in *Proc. SEMANTiCS*. Leipzig, Germany: ACM, 2016, pp. 189–192. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2993318.2993351
[41] N. Nethercote and J. Seward, "Valgrind: A framework for heavyweight dynamic binary instrumentation," in *Proc. PLDI*, 2007, pp. 89–100.
[42] C. Sadowski, E. Aftandilian, A. Eagle, L. Miller-Cushon, and C. Jaspan, "Lessons from building static analysis tools at Google," *Commun. ACM*, vol. 61, no. 4, pp. 58–66, 2018, doi: 10.1145/3188720.
[43] M. Harman and P. O'Hearn, "From start-ups to scale-ups: Opportunities and open problems for static and dynamic program analysis," in *Proc. SCAM*, 2018, pp. 1–23.
[44] J. R. Larus, T. Ball, M. Das, R. DeLine, M. Fahndrich, J. Pincus, S. K. Rajamani, and R. Venkatapathy, "Righting software," *IEEE Softw.*, vol. 21, no. 3, pp. 92–100, May 2004.
[45] N. Ferry, P. H. Nguyen, H. Song, E. Rios, E. Iturbe, S. Martinez, and A. Rego, "Continuous deployment of trustworthy smart IoT systems," *J. Object Technol*, vol. 19, no. 2, pp. 16:1–23:1, 2020.
[46] M. Mazkatli and A. Koziolek, "Continuous integration of performance model," in *Proc. ICPE (Companion)*. Berlin Germany: ACM, Apr. 2018, pp. 153–158. [Online]. Available: https://dl.acm.org/doi/10.1145/3185768.3186285
[47] T. F. Düllmann, C. Paule, and A. van Hoorn, "Exploiting DevOps practices for dependable and secure continuous delivery pipelines," in *Proc. RCoSE*, 2018, pp. 27–30.
[48] E. Di Nitto, P. Jamshidi, M. Guerriero, I. Spais, and D. A. Tamburri, "A software architecture framework for quality-aware DevOps," in *Proc. QUDOS*, 2016, pp. 12–17.
[49] R. Heinrich, R. Jung, C. Zirkelbach, W. Hasselbring, and R. Reussner, "An architectural model-based approach to quality-aware DevOps in cloud applications," in *Software Architecture for Big Data and the Cloud*. Amsterdam, The Netherlands: Elsevier, 2017, pp. 69–89.
[50] M. Jiménez, L. F. Rivera, N. M. Villegas, G. Tamura, H. A. Müller, and N. Bencomo, "An architectural framework for quality-driven adaptive continuous experimentation," in *Proc. RCoSE/DDrEE*, 2019, pp. 20–23.
[51] H.-M. Chen, R. Kazman, S. Haziyev, V. Kropov, and D. Chtchourov, "Architectural support for DevOps in a neo-metropolis BDaaS platform," in *Proc. SRDSW*, 2015, pp. 25–30.
[52] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," *Inf. Softw. Technol.*, vol. 108, pp. 65–77, Apr. 2019. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0950584918302507
[53] L. Chen, "Continuous delivery: Overcoming adoption challenges," *J. Syst. Softw.*, vol. 128, pp. 72–86, Jun. 2017.
[54] M. Shahin, M. A. Babar, and L. Zhu, "The intersection of continuous deployment and architecting process: Practitioners' perspectives," in *Proc. ESEM*. Ciudad Real, Spain: ACM, 2016, pp. 1–10. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2961111.2962587
[55] P. W. O'Hearn, "Continuous reasoning: Scaling the impact of formal methods," in *Proc. LICS*, 2018, pp. 13–25, doi: 10.1145/3209108.3209109.
[56] H. Koziolek and R. Reussner, "A model transformation from the palladio component model to layered queueing networks," in *Proc. SPEC Int. Perform. Eval. Workshop*. Berlin, Germany: Springer, Jun. 2008, pp. 58–78.
[57] A. Brunnert *et al.*, "Performance-oriented DevOps: A research agenda," 2015, *arXiv:1508.04752*. [Online]. Available: https://arxiv.org/abs/1508.04752
[58] E. Di Nitto, P. Jamshidi, M. Guerriero, I. Spais, and D. A. Tamburri, "A software architecture framework for quality-aware DevOps," in *Proc. QUDOS*. New York, NY, USA: Association for Computing Machinery, 2016, pp. 12–17, doi: 10.1145/2945408.2945411.
[59] D. Sun, M. Fu, L. Zhu, G. Li, and Q. Lu, "Non-intrusive anomaly detection with streaming performance metrics and logs for DevOps in public clouds: A case study in AWS," *IEEE Trans. Emerg. Topics Comput.*, vol. 4, no. 2, pp. 278–289, Apr. 2016.
[60] Y. Dang, Q. Lin, and P. Huang, "AIOPS: Real-world challenges and research innovations," in *Proc. IEEE/ACM ICSE (Companion)*, 2019, pp. 4–5.
[61] A. F. Nogueira, J. C. Ribeiro, M. Zenha-Rela, and A. Craske, "Improving La Redoute's CI/CD pipeline and DevOps processes by applying machine learning techniques," in *Proc. QUATIC*, 2018, pp. 282–286.
[62] R. Kumar, C. Bansal, C. Maddila, N. Sharma, S. Martelock, and R. Bhargava, "Building Sankie: An AI platform for DevOps," in *Proc. IEEE/ACM BotSE*, 2019, pp. 48–53.
[63] R. Ciucu, F. C. Adochiei, I.-R. Adochiei, F. Argatu, G. C. Seri an, B. Enache, S. Grigorescu, and V. V. Argatu, "Innovative DevOps for artificial intelligence," *Sci. Bull. Electr. Eng. Fac.*, vol. 19, no. 1, pp. 58–63, Apr. 2019.
[64] H.-M. Chen, R. Kazman, and S. Haziyev, "Agile big data analytics development: An architecture-centric approach," in *Proc. HICSS*, 2016, pp. 5378–5387.
[65] G. C. Fox, J. Qiu, S. Kamburugamuve, S. Jha, and A. Luckow, "HPC-ABDS high performance computing enhanced apache big data stack," in *Proc. IEEE/ACM CCGRID*, Jun. 2015, pp. 1057–1066.
[66] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng., Softw. Eng. Pract. (ICSE-SEIP)*, Jun. 2019, pp. 291–300.
[67] C. R. de Souza, D. Redmiles, L.-T. Cheng, D. Millen, and J. Patterson, "Sometimes you need to see through walls: A field study of application programming interfaces," in *Proc. ACM Conf. Comput. Supported Cooperat. Work*, 2004, pp. 63–71.
[68] C. R. De Souza, D. Redmiles, and P. Dourish, "'Breaking the code' moving between private and public work in collaborative software development," in *Proc. Int. ACM SIGGROUP Conf. Supporting Group Work*, 2003, pp. 105–114.

AHMAD ALNAFESSAH is currently pursuing the Ph.D. degree with the Department of Computing, Imperial College London. He was a Senior Academic Researcher with the National Centre for AI and Big Data Technologies, KACST, from 2012 to 2017. His research focuses on performance engineering for big data systems and AI, with a specific focus on in-memory platforms. He is interested in big data systems, AI, IoT, HPC, complex distributed systems, and cloud computing.

ALIM UL GIAS received the bachelor's degree in information technology (major in software engineering) from the Institute of Information Technology (IIT), University of Dhaka (DU), and the M.Sc. degree in software engineering from IIT, DU, with thesis on adaptive software performance testing. He is currently pursuing the Ph.D. degree with Imperial College London, U.K. His current research interest includes software performance engineering using stochastic process modeling and machine learning.

RUNAN WANG received the B.Eng. and M.Sc. degrees in software engineering from the Beijing Institute of Technology (BIT), China, in 2017 and 2019, respectively. She is currently pursuing the Ph.D. degree with the Department of Computing, Imperial College London. Her current research focuses on performance models, program engineering, and machine learning.

LULAI ZHU received the B.Eng. and M.Eng. degrees in measurement technology and instruments from Sichuan University, China, in 2008 and 2011, respectively, and the M.Sc. degree in computing science from Imperial College London, U.K., in 2016. He is currently pursuing the Ph.D. degree with the Department of Computing. He is currently an RA with the Department of Computing. His current research focuses on performance models, such as queuing networks and Petri nets, and their applications to distributed systems and cloud computing.

GIULIANO CASALE joined the Department of Computing, Imperial College London, in 2010, where he is currently a Senior Lecturer in modeling and simulation. Previously, he worked as a Scientist at SAP Research, U.K., and as a Consultant in the capacity planning industry. He teaches and does research in performance engineering, cloud computing, and big data, topics on which he has published more than 100 refereed articles. He has served on the technical program committee of over 80 conferences and workshops and as co-chair for several conferences in the area of performance engineering, such as ACM SIGMETRICS/Performance. He was a recipient of multiple awards, recently the Best Paper Award at ACM SIGMETRICS in 2017. He serves on the Editorial Board of IEEE Transactions on Network and Service Management (TNSM) and ACM TOMPECS and as the Chair of ACM SIGMETRICS.

ANTONIO FILIERI is an Assistant Professor with Imperial College London, U.K. His main interests are in the application of mathematical methods for software engineering, in particular, probability, statistics, logic, and control theory. His recent work includes exact and approximate methods for probabilistic symbolic execution, incremental verification, quantitative software modeling and verification at runtime, and control-theoretical software adaptation. www.antonio.filieri.name.