Compliance Management of IaC-Based Cloud Deployments During Runtime

Ghareeb Falazi
Lukas Harzenetter
Kálmán Képes
Frank Leymann
Institute of Architecture of Application Systems,
University of Stuttgart
Germany
{firstname.lastname}@iaas.uni-stuttgart.de

Uwe Breitenbücher
Reutlingen University
Germany
uwe.breitenbuecher@reutlingen-university.de

Evangelos Ntentos
Uwe Zdun
Research Group Software Architecture, Faculty of
Computer Science, University of Vienna
Austria
{firstname.lastname}@univie.ac.at

Martin Becker
Elena Heldwein
IBM Deutschland
Germany
martin.becker@de.ibm.com
elena.heldwein1@ibm.com

ABSTRACT

Modern cloud applications increasingly depend on Infrastructure-as-Code (IaC) practices for infrastructure automation to help manage the complexity of deploying large-scale architectures. Additionally, the deployment of cloud applications is commonly subject to compliance rules. Moreover, designing compliant IaC-based cloud deployments is not enough since runtime changes to the infrastructure or the configuration of individual components may introduce compliance violations. Often, the process of checking and fixing such violations is done manually, which is time-consuming and error-prone. Therefore, this work aims to define and implement a method for runtime IaC compliance management that reduces the complexity, effort, and uncertainty of checking and enforcing compliance rules against IaC-based cloud deployments at runtime. To this end, we follow the design-science research methodology to design and implement (i) the Runtime IaC Compliance Management (RICMa) method and (ii) the IaC Compliance Management Framework (IaCMF) that supports the execution of the RICMa method. We prototypically implement IaCMF and evaluate it using a qualitative interview study with industry experts.

CCS CONCEPTS

• **Computer systems organization** → *Cloud computing*; • **Applied computing** → **IT governance**; • **General and reference** → *Verification.*

KEYWORDS

Infrastructure-as-Code, IaC, IT Compliance, Cloud Compliance, Compliance Management

ACM Reference Format:
Ghareeb Falazi, Lukas Harzenetter, Kálmán Képes, Frank Leymann, Uwe Breitenbücher, Evangelos Ntentos, Uwe Zdun, Martin Becker, and Elena Heldwein. 2023. Compliance Management of IaC-Based Cloud Deployments During Runtime. In *Proceedings of The 16th IEEE/ACM International Conference on Utility and Cloud Computing (UCC '23).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Modern cloud applications increasingly depend on microservice-based architectures. However, the development and deployment of microservice systems have become more complex due to the need for frequent releases and the large number of different cloud technologies required to host the application's components, e.g., PaaS and FaaS offerings as well as container-based deployment technologies such as Kubernetes. Additionally, cloud applications are released to production frequently, sometimes multiple times per day, leading to frequent changes regarding the infrastructure components used for hosting [17]. Therefore, many organizations adopted *Infrastructure-as-Code (IaC)*, which are practices involving the use of reusable scripts and deployment models, referred to as *IaC code*, to manage and provision IT infrastructure. Thereby, the term *infrastructure* refers to all services, components, and platforms that are used to host an application's components [23]. IaC aims to keep the provisioned environment and its intended configuration consistent [2, 23]. Furthermore, implementing IaC can also improve security, and reduce errors and manual configuration effort [2]. Many IaC tools exist to support deploying and configuring modern cloud applications and cloud-based systems, e.g., Ansible, Terraform, and Kubernetes. However, the configuration and host environments for such deployments are typically subject to a set of domain-specific guidelines, regional laws, and internally enforced policies, which we collectively refer to as *compliance rules*. For example, an enterprise might require all containers to follow a specific security benchmark. Thus, when using IaC for infrastructure automation, enterprises need to ensure that the IaC code and its resulting effects on the actual host environment adhere to such compliance rules. We call such deployments that are provisioned and/or managed by IaC tools *IaC-based cloud deployments*.

Obviously, it is important to design compliant IaC code. However, even if compliance is ensured during design time, IaC-based cloud deployments might still suffer from compliance violations during runtime. For example, Figure 1 shows the conceptual architecture of a cloud application that is deployed and managed by an IaC tool such as OpenTOSCA [3] or Terraform. The architecture contains several software components that host a simple three-tier application. The backend is a Java application hosted on a Docker container. The Docker engine is hosted on an AWS EC2 instance. In addition, another container runs on the same engine and hosts an NGINX server with an Angular application on top constituting the frontend. The backend persists data in a MySQL DB hosted on an on-premise stack. During the lifetime of this cloud application, ad-hoc changes to its components might result in compliance violations. For example, the system administrator might decide to introduce temporary changes to facilitate unforeseen or repetitive tasks such as adding a new *User X* to the DB to allow temporary access by a data analysis application or changing the security configuration of the operating system to allow quicker user access by permitting empty user passwords. Such changes introduce security vulnerabilities and violate common compliance guidelines such as STIGs [7] and NIST SP 800-53 [24]. Furthermore, a carefree system administrator may introduce privately used components that consume infrastructure resources and constitute security threats. For example, Figure 1 shows a Docker container (*Container X*) that is privately used, e.g., to mine Bitcoin using the company's infrastructure.

Multiple studies demonstrate how to check the architectural compliance of IaC-based cloud applications at **design-time** [19, 25, 26] and at **deployment-time** [38]. However, the compliance violations presented in the previous example can neither be checked during design time nor during deployment time, but only at **runtime**. Industrial solutions, e.g., Prisma Cloud [27] and VMware Aria Automation for Secure Hosts [34] focus on runtime compliance violations that pertain to the configuration of individual software components, but as shown in the previous example, runtime IaC compliance violations can also be architectural in nature, i.e., resulting from the non-compliant addition or removal of software components during runtime. Therefore, the current process of checking and fixing such compliance violations is typically done manually by experts who have the domain knowledge to interpret the corresponding compliance rules and determine how to fix the possible violations for different IaC tool types without affecting the integrity of IaC-based cloud deployments. This is a repetitive task that is complex, time-consuming, and error-prone due to the uncertainty of how to interpret compliance rules and fix compliance violations.

Hence, the *goal* of this research is *to define and implement a compliance management process that reduces the complexity, effort, and uncertainty of checking compliance rules against IaC-based cloud deployments at runtime and fixing the possible violations.* Specifically, we tackle the following research questions:

- **RQ1** How can compliance rules be modeled, checked, and enforced against IaC-based cloud deployments at runtime while minimizing the associated effort, complexity, and uncertainty?
- **RQ2** How can a framework support performing these tasks?

To answer these questions, (i) we introduced the *Runtime IaC Compliance Management (RICMa) method* that describes how to model, check, and enforce compliance for IaC-based cloud deployments during runtime. (ii) we designed the *IaC Compliance Management Framework (IaCMF)*, which supports the execution of the RICMa method. (iii) we prototypically implemented IaCMF, and (iv) we conducted a qualitative user interview study to evaluate the usefulness of the proposed concepts. A partial design for the *RICMa method* was introduced in a previous work [8] without technical details. In this work, the *RICMa method* is significantly extended and equipped with technical details.

The paper is structured as follows: we discuss related work in Section 2 and the research method in Section 3. We describe the RICMa method in Section 4 and IaCMF in Section 5. Finally, we present a prototypical implementation of IaCMF and a use case in Section 6, evaluate the developed concepts qualitatively in Section 7, discuss the threats to validity in Section 8, and conclude in Section 9.

2 RELATED WORK

In this section, we present relevant academic and industrial works and compare our approach with them.

2.1 Related Work on IaC Best Practices and Design-Time IaC Compliance

With the increasing adoption of IaC practices in industry, there is a growing body of scientific research that collects and systematizes IaC-related patterns and practices. For example, Kumara et al. [20] compile a catalog encompassing language-agnostic and language-specific best and bad practices that address implementation issues, design problems, and violations of IaC principles, while Sharma et al. [31] propose language-specific best practices for Puppet. In contrast to our work, these works merely introduce compliance rules and not a comprehensive compliance management method.

Moreover, multiple studies propose tools and metrics to assess the quality of IaC code. Dalla Palma et al. [5, 6] introduce a catalog of 46 quality metrics specifically tailored to Ansible scripts, which enable the identification of IaC-related properties and demonstrating their application in analyzing IaC code. Kumara et al. [21] propose a tool-based approach for detecting smells in TOSCA models. Sotiropoulos et al. [32] develop a tool-based approach that identifies issues related to dependencies by analyzing Puppet manifests and their system call trace. Van der Bent et al. [33] define metrics that encompass best practices for assessing Puppet code quality. In contrast to our approach, none of these tools allows for modeling custom compliance rules and resolving architectural compliance violations. Moreover, in the works [9, 19], an approach is presented for automatically verifying the compliance of declarative deployment models during design-time. The approach uses custom compliance rules in the form of deployment model fragments that describe the desired design of a specific part of the deployment model. The comparison between the model fragment and the deployment model is performed using subgraph isomorphism.

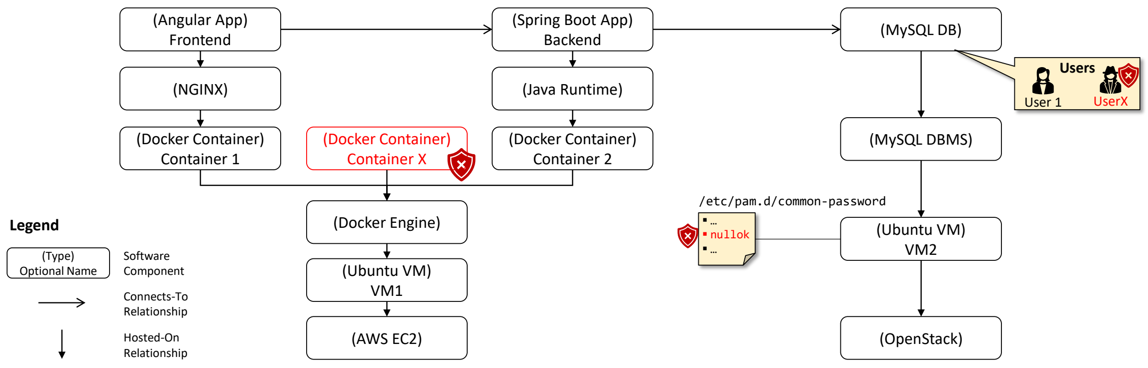


Figure 1: Example IaC-based cloud deployment that has three runtime compliance violations.

However, detected compliance violations are not resolved and only TOSCA-based deployment models are supported. We build upon this approach by (i) including compliance enforcement, integrity validation, and reporting, (ii) focusing on runtime compliance, and (iii) enhancing extensibility and support for heterogeneous systems through plugins.

2.2 Related Work on Behavioral Compliance for Cloud Deployments

Several approaches exist for runtime behavioral compliance checking. For example, Krieger et al. [18] propose an approach to check application system conformance to common architectural patterns of behavioral nature during runtime. Their approach depends on complex event processing via a dedicated compliance monitoring component. Moreover, Havelund [14] presents a rule-based runtime verification approach to monitor the behavioral compliance of application systems. The approach utilizes SCALA to define compliance rules and an AI-based algorithm to check them against system traces. However, these and similar approaches are aimed at checking the behavioral compliance of application systems, i.e., how well application systems behave in accordance with pre-defined rules. In contrast, our approach checks the architectural compliance of the *infrastructure* of IaC-based cloud deployments during runtime. Behavioral compliance is outside the scope of this work.

2.3 Related Industrial Approaches

A set of industrial platforms deal with IT compliance management. For example, *Google Cloud Security Command Center (GCD SCC)* [11] is a built-in security and risk management solution for Google Cloud. It provides continuous design-time and runtime checking of cloud resource configuration against a predefined set of compliance rules. Furthermore, it suggests manual fixing steps for found vulnerabilities. Nonetheless, this solution is not able to detect structural compliance violations and is only restricted to Google Cloud.

Azure Policy [22] is a solution that allows users of Azure cloud to enforce organizational standards and assess compliance policies for cloud deployments. It introduces a domain-specific language for defining compliance policies. These policies allow to check the configuration of cloud resources both at design-time and at runtime. All Azure-based resources are supported. Additionally, Azure Policy supports a predetermined set of resource types whose instances are hosted outside of Azure, e.g., Linux Servers, Kubernetes Clusters, etc. Furthermore, it allows to fix resource configuration violations at runtime. However, this solution does not support checking or enforcing structural IaC compliance rules and does not allow extending its capabilities, e.g., through plugins.

AWS Config [1] facilitates live monitoring of resources hosted on AWS cloud. With this tool, the current configuration of individual cloud resources can be queried using predefined and customizable *inventories*, i.e., checks that collect information about a target set of managed cloud resources. The collected information can also include details about the architecture of the cloud deployment, e.g., which applications are hosted on a given EC2 instance. Compliance checks can then be executed against the collected information by defining corresponding compliance rules. A rule defines the desired configuration setting of a specific type of cloud resource in addition to the remediation action to be taken if the rule is violated. AWS Config provides a repository of predefined compliance rules and allows for the creation of custom rules. However, like Azure Policy, this approach does not support extending its capabilities through plugins. Additionally, it only supports resources hosted on AWS.

Red Hat (RH) OpenShift Container Platform [29] provides compliance checking and enforcement for containers running in OpenShift. A preset of rules is provided, which can be customized. Scans can be executed at runtime and if remediation recommendations exist, they can be automatically applied. A limitation of this tool is its exclusive integration with RH OpenShift.

Palo Alto Network's Prisma Cloud [27] is an API-based service for cloud environments that includes compliance management. It allows retrieving cloud resource configuration details from a variety of cloud providers, e.g., AWS, GCP, Azure, etc. This information can be used to check IaC configuration compliance at build-time and runtime. Beyond provided default policies and compliance standards, it also allows for the creation of custom policies and standards. Issues found at runtime can be automatically remediated using a limited number of cloud CLI commands. Although Prisma Cloud supports a wide range of cloud providers, it is not extensible and the aspects that can be checked and remediated are restricted to the respective cloud API's functionalities.

Lastly, *VMware Aria Automation for Secure Hosts* [34] is a compliance and vulnerability management tool. It allows the checking

Table 1: Characteristics of related industrial approaches. For comparison, our approach is presented in the last row.

Approach	Runtime Compliance Rule Checking		Custom Rules	Autom. Fixing	Hetero-geneity	Exten-sibility
	Config.	Structure				
GCD SCC [11]	Yes	No	No	No	No	No
Azure Policy [22]	Yes	No	Yes	Yes	Yes	No
AWS Config [1]	Yes	Yes	Yes	Yes	No	No
RH OpenShift [29]	Yes	No	Yes	Yes	No	No
Prisma Cloud [27]	Yes	No	Yes	Yes	Yes	No
VMWare Aria [34]	Yes	No	Yes	Yes	Yes	No
RICMa/IaCMF	Yes	Yes	Yes	Yes	Yes	Yes

of infrastructure compliance using either default or custom compliance policies. An SDK is provided for custom assessment and remediation. However, the tool only allows integrating systems that can be managed through VMware Aria Automation and does not support checking or enforcing architectural compliance rules.

Table 1 summarizes the characteristics of the discussed industrial approaches. The table shows whether a given approach supports checking IaC compliance rules that address the configuration of cloud resources in addition to rules of an architectural nature. Additionally, it indicates whether the approach supports creating custom compliance rules and introducing automatic remediation of compliance violations. Finally, it shows if the approach supports heterogeneous environments and IaC tools, and if its functionality and the spectrum of supported tools and environments can be extended, e.g., through plugins. Compared to our approach, industrial solutions have limited scope in terms of the supported IaC tools and cloud platforms. Additionally, they do not support extensibility through plugins. Hence, they are limited regarding heterogeneous environments and legacy systems, and risk vendor lock-in. Finally, they put only little focus on compliance rules of architectural nature. Nonetheless, they constitute a good reference for distilling industry-accepted requirements for compliance management.

3 RESEARCH METHOD

In this work, we employ the design-science research method [15, 36]. In software engineering research, design science creates and evaluates *IT artifacts* intended to solve problems identified within organizations [15]. We created and evaluated two IT artifacts: (i) a method that facilitates the compliance management of IaC-based cloud deployments during runtime, the *RICMa method*, and (ii) an extensible framework, *IaCMF*, that facilitates this method. To this end, we use the methodology proposed by Wieringa [36], in which one or more *design cycles* are conducted each with three activities: (i) *problem investigation*, in which the stakeholders and goals are identified and the state-of-the-art is studied, (ii) *treatment design*, in which the artifacts are designed, and (iii) *treatment validation*, in which the artifacts are assessed for their capability to "treat" the problem. Figure 2 summarizes the steps we followed and highlights how they relate to the steps of the design cycle. After setting the goal of the research, we conducted two design cycles. In cycle 1, we analyzed the current state-of-the-art in the domain of IT compliance management. Next, we created an initial design for the RICMa method and IaCMF, and validated these initial artifacts by discussing them in a focus group comprising six industry experts from a large technology enterprise. The experts suggested enhancements to the RICMa

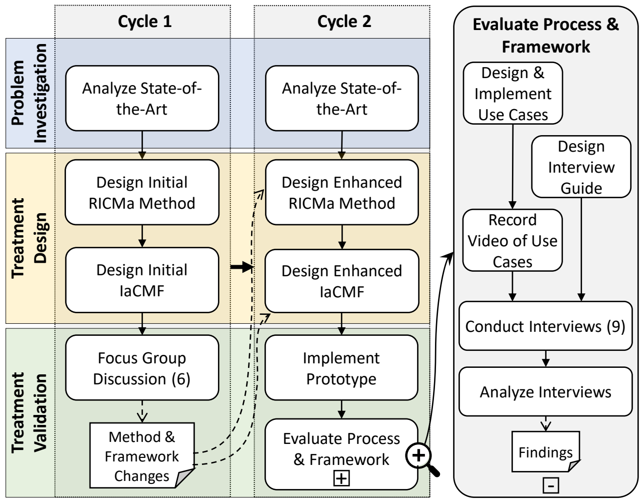


Figure 2: Our research method based on [36]: two design cycles ending in a qualitative interview study.

method and IaCMF focusing on introducing the ability to validate the integrity of the deployments after applying automatic fixes and supporting a class of compliance rules relevant to large enterprises.

This triggered a second design cycle, in which we extended our problem investigation in accordance with the new requirements suggested by the focus group. Next, we enhanced our previous designs for the RICMa method and IaCMF. To validate these enhanced artifacts, we created a prototypical implementation of the framework, used it to handle three example compliance use cases, and conducted a qualitative interview study with nine industry experts from six different companies.

The initial RICMa method, designed as part of the first cycle, was introduced in a previous work [8], in which we presented a high-level vision without any technical details. In this work, we conducted all the other steps discussed above, which resulted in a significant enhancement of the approach. In Section 4 and Section 5, we discuss the enhanced RICMa method and IaCMF respectively, and in Section 6, we discuss the implemented prototype and one of the three example use cases. Finally, in Section 7, we discuss the qualitative interview study.

4 RUNTIME IAC-BASED COMPLIANCE MANAGEMENT METHOD

In this section, we introduce the RICMa method, which allows to model, check, and enforce compliance rules during runtime while maintaining application instance integrity. The method is depicted in Figure 3 and is split into the *Compliance Rule Modeling Process*, and the *Compliance Checking and Enforcement Process*. Note that since the same IaC code can be used to instantiate an application multiple times, in the following, we refer to a specific deployment thereof as an *application instance*.

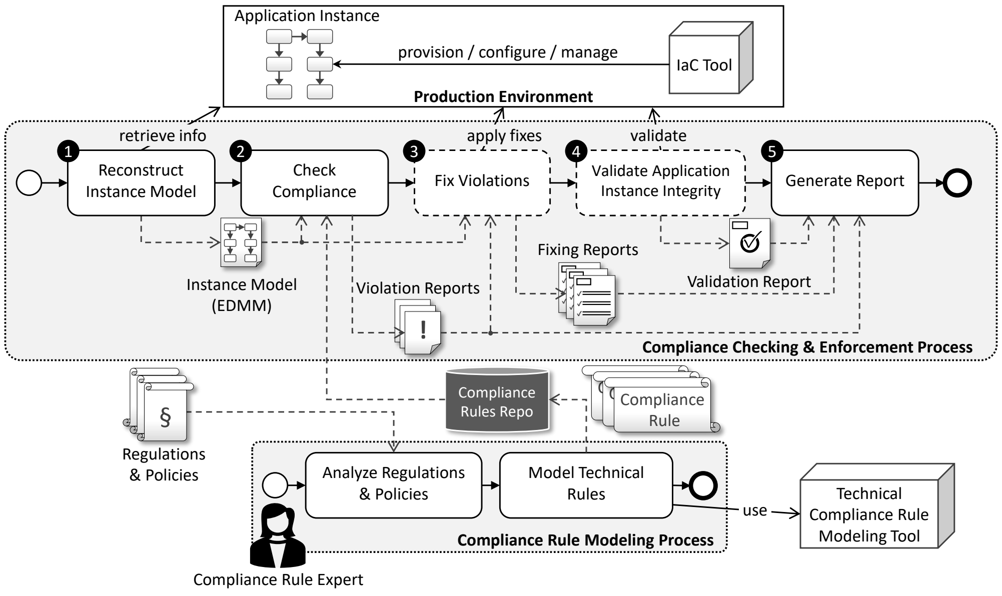


Figure 3: The RICMa method. Dotted tasks are optional. A high-level concept of steps 1–3 was presented in a previous work [8] without technical details.

4.1 Compliance Rule Modeling Process

In this process, all applicable compliance requirements that stem from enterprise-external regulations and enterprise-internal policies are turned into technical compliance rules that can be automatically checked by IaCMF: A *Compliance Rule Expert* analyzes the documents that describe the applicable compliance rules and uses a suitable *Technical Compliance Rule Modeling Tool* to implement corresponding machine-readable technical compliance rules. The implemented technical compliance rules are then stored in a *Compliance Rule Repository*, which makes them accessible to the Compliance Checking and Enforcement Process. Since these rules are machine-readable and embed expert knowledge, they reduce the uncertainty associated with interpreting the corresponding policies and guidelines. The format of the technical compliance rules depends on how the *Check Compliance* step of the Compliance Checking and Enforcement Process is implemented. Section 6.2 presents a technical example.

4.2 Compliance Checking and Enforcement Process

The primary objective of this process is the execution of compliance jobs. A *compliance job* is a user-defined entity that represents a set of interrelated compliance rules that apply to a specific application instance and defines the information necessary to fix possible violations and subsequently validate the application instance. The process, which is further detailed below, is repeated for each compliance job separately. The provided details include the requirements for IaCMF, which implements this process.

4.2.1 ❶ *Reconstruct Instance Model.* In this step, the framework creates a model that represents the architecture of the running application instance under consideration, which we refer to as *instance model*. We use the approach proposed by Harzenetter et al. [12, 13]: An initial instance model is created based on information retrieved from the IaC tool(s) that manage the application instance. To achieve this, every supported IaC tool must have a corresponding *instance model creation plugin*. The initial instance model contains general information about the application instance as known to the IaC tool(s) including the involved software components and (some of) their relations. However, IaC tools use their own languages with their own syntax and expressiveness to describe the IT resources they manage. Therefore, to facilitate extensibility and universality, we decided to base our approach on the *Essential Deployment Metamodel (EDMM)* [37], which is a technology-agnostic common denominator for the features of common IaC tools obtained by systematically analyzing them. EDMM supports the specification of typed software components that are interconnected using typed relations. Components and relations are further described using properties. Thus, we require all instance model creation plugins that extract application instance details from IaC tools to generate corresponding instance models using EDMM.

The initial instance model may not be sufficient for the execution of most compliance jobs since certain compliance rules require detailed information about specific software components as we have seen in Figure 1. For example, to check that the operating system of a given VM is securely configured, we need information about the contents of the corresponding configuration files. However, such detailed information is typically unknown to the IaC tool. Furthermore, some architectural changes might be introduced to the application instance at runtime. These changes are not known to the IaC tool and, therefore, cannot be part of the initial instance model. Hence, after the initial instance model is created, it is refined with information directly retrieved from the components of the application instance by letting IaCMF sequentially apply *architectural refinement plugins* that connect to the corresponding software components, retrieve the necessary information and process it, and introduce changes to the instance model accordingly. These changes could be, e.g., the addition of newly discovered software components or new properties added to existing components. Architectural refinement plugins also consume and generate EDMM instance models. This effectively decouples them from each other and from instance model creation plugins, since each plugin expects a technology-agnostic instance model as input regardless of which and how many previous plugins operated on it. Finally, it is important to note that the goal of this step is not to generate a comprehensive instance model but rather only to include the information necessary to successfully check the compliance rules included in the job under consideration. Thus, IaCMF will only execute the instance model refinement plugins that provide the information required by the compliance job.

4.2.2 ❷ *Check Compliance.* In this step, all compliance rules included in the compliance job are checked and violations are identified. To this end, the EDMM instance model from the previous step is evaluated against the relevant compliance rules retrieved from the Compliance Rule Repository. For each identified violation, a *Violation Report* is generated that refers to the corresponding compliance rule and to the elements of the instance model that cause the violation. There are many approaches that check the conformance of graph-based models, e.g., approaches based on subgraph-matching [9, 19, 38], approaches based on logic programming [30], and approaches based on evaluating predefined metrics [25, 26]. Therefore, to support extensibility, this step depends on *compliance checking plugins* that utilize any compliance checking approach of choice providing that EDMM instance models are accepted as input and properly formatted Violation Reports are produced as output.

Note that certain compliance rules, such as rules that pertain to legal or ethical requirements, may require human interpretation. In such a case, the corresponding technical compliance rule is only able to give indications that it *might* be violated by a certain production system. Then, a suitable compliance checking plugin waits for *human input* before deciding to produce a corresponding Violation Report or to consider the finding as a false positive.

4.2.3 ❸ *Fix Violations.* The goal of this optional step is to automatically fix the compliance violations reported in step ❷. Obviously, there are many ways compliance violations may be addressed, which we categorize into three groups: (i) *Compliance violations that can be fixed automatically by executing the original IaC code.* Certain IaC tools, such as Terraform, maintain a representation of the state of the running cloud applications they manage. If these applications are altered in an ad-hoc manner, they deviate from the tool-maintained representation. Therefore, such tools usually provide a mechanism, e.g., the Terraform `refresh` command, to "refresh" their internal representation of the managed application instances. Providing that the IaC code is compliant, runtime compliance violations can then be fixed by requesting a redeployment of the IaC code. Accordingly, the tool introduces changes to the running application instance including its infrastructure and configuration to match the original IaC code, thus enforcing compliance again. Ensuring that a deployment model adheres to compliance rules is outside the scope of this work and can be done using multiple existing approaches (see Section 2). (ii) *Compliance violations that can be fixed automatically by applying changes to software components.* In certain cases, the used tool, e.g., Ansible, does not maintain a representation of the state of the managed application instances, or the compliance violations may correspond to aspects of the application instance that cannot be affected by the IaC tool. In these cases, fixing compliance violations can be accomplished by directly accessing the affected software components and applying changes to them, e.g., assigning a specific value to a given configuration file record in a VM. (iii) *Compliance violations that require manual intervention.* Certain compliance violations are dangerous to fix automatically. For example, a violation related to a mission-critical DB requires human oversight to be fixed safely.

Accordingly, this process step supports framework extensibility via *violation fixing plugins* suitable for different kinds of compliance violations and different IaC tools. These plugins take the Violation Reports and the reconstructed instance model as input and apply changes to the application instance directly or with the help of the IaC tool. As output, the plugins generate a *Fixing Report* for every violation they attempt to fix. In case automatic fixing is not desired, this step is skipped and the reported violations are directly forwarded to human operators in step ❺ to handle them.

4.2.4 ❹ *Validate Application Instance.* In this optional step, the application instance is validated to ensure that the automatic fixes possibly applied in the previous step have not affected its integrity. This can be done, for example, by executing some or all of the integration tests that are normally part of the CI/CD pipeline. As output, a *Validation Report* is generated and propagated to the next step. Finally, to support different kinds of toolings for integration testing, this step uses corresponding *validation plugins.*

4.2.5 ❺ *Generate Report.* In this step, the compliance job execution is reported, which is done, e.g., by sending an email to the manager with a summary of the execution, or by sending a message to a pub/sub topic of a connected Message-Oriented Middleware that can be consumed by external systems to further perform operations related to the compliance management process. Therefore, this step provides a chance to extend the RICMa method with new integration possibilities. Hence, IaCMF needs *reporting plugins* that take the Violation Reports, the Fixing Reports, and the Validation Report as input, and perform the intended external reporting.

5 IAC-BASED APPLICATION COMPLIANCE MANAGEMENT FRAMEWORK

In this section, we present IaCMF, an extensible framework that realizes the method described in Section 4. The conceptual architecture of IaCMF consists of three layers and is presented in Figure 4. At the top is the *API Layer*, which allows client applications to utilize the functionality of IaCMF. In the middle, we have the *Domain Logic Layer*, which is responsible for implementing the RICMa method and facilitating its extensibility. This layer has the following components: (i) The *Execution Orchestrator* component, which is responsible for orchestrating the execution of the Compliance Checking and Enforcement Process (see Section 4.2) by invoking the components that implement the different process steps and passing

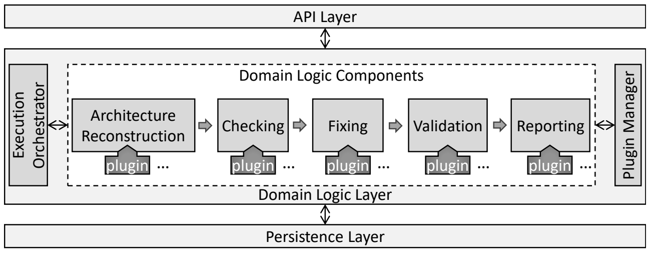


Figure 4: The conceptual architecture of IaCMF.

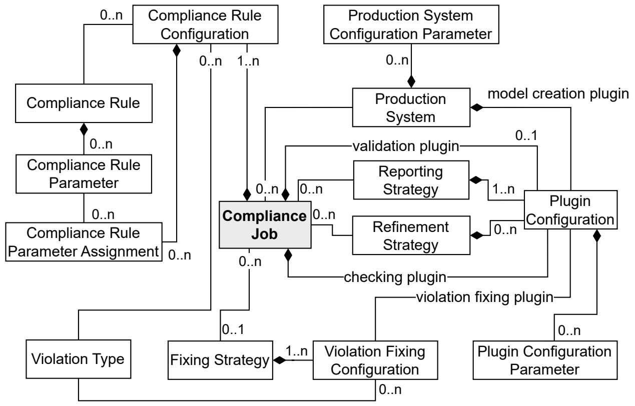


Figure 5: Class diagram for the IaCMF domain entities.

data between them. It is also responsible for integrating the Compliance Rule Modeling Process and importing the modeled technical compliance rules during the execution of the corresponding compliance jobs from the repo. (ii) The *Execution Components* are the components that implement the different steps of the Compliance Checking and Enforcement Process. Each of these components is extensible with plugins that realize concrete approaches. (iii) The *Plugin Manager* component allows deploying new plugins for the Execution Components. At the bottom, we have the *Persistence Layer*, which is responsible for storing all the domain entities necessary to model and execute compliance jobs according to the RICMa method. Specifically, Figure 5 depicts a simplified class diagram of these entities. Entity properties are omitted for brevity. We explain the domain entities in the following:

A *Compliance Rule* is the framework's representation of a technical compliance rule created using the Compliance Rule Modeling Process, and it can have *Compliance Rule Parameters*, which facilitate instantiating customizable compliance rules for different scenarios. Furthermore, a *Production System* is a reusable entity that allows the framework to access an existing application instance via an IaC tool. To this end, it refers to a set of *Production System Configuration Parameters* that contain the information necessary to communicate with the IaC tool and identify the target application instance. A *Compliance Job* represents a set of interrelated *Compliance Rules* that apply to a specific *Production System*. It also defines the information necessary to fix possible violations and validate the corresponding application instance. A *Compliance Rule Configuration* connects an existing *Compliance Rule* to a *Compliance Job*, specifies concrete values for the rule's parameters using *Compliance Rule Parameter Assignments*, and determines the *Violation Type* to report if the rule is violated.

Finally, IaCMF supports multiple types of customizable plugins. In order for the framework to use a specific plugin, a *Plugin Configuration* entity is needed, which represents an instance of the plugin and the set of *Plugin Configuration Parameters* to configure it. Moreover, we can specify how the different steps of the Compliance Checking and Enforcement Process will be performed as follows: A *Production System* refers to a model creation plugin suitable for the specific IaC tool at hand, e.g., Kubernetes. Furthermore, each *Compliance Job* refers to a reusable *Refinement Strategy* that specifies the sequence of architectural refinement plugins that will refine the initial instance model. It also refers to a compliance checking plugin, a validation plugin, a *Fixing Strategy*, and a *Reporting Strategy*. A *Fixing Strategy* is a set of *Violation Fixing Configurations* that map possible *Violation Types* to violation fixing plugins, and a *Reporting Strategy* is set of reporting plugins triggered at the end of the process. Refer to Section 6.2 for technical examples.

6 VALIDATION

In this section, we validate IaCMF by introducing a prototypical implementation thereof and describing how it can be used in a concrete compliance use case.

6.1 Prototypical Implementation

We developed a prototypical implementation of IaCMF in the form of a Spring Boot application as a backend that exposes a REST API, and an Angular application as a UI frontend that consumes this API. The frontend allows the users to configure and trigger compliance jobs, and examine the results. The backend stores user-configured domain entities and compliance job execution results in a MySQL DB. The implementation supports configuring multiple compliance jobs running against different application instances and executing these jobs in parallel. To demonstrate the extensibility and flexibility of the RICMa method, we implemented eleven IaCMF plugins[1], which are described in the following: We created three *instance model creation plugins*: (i) The `opentosca-container-model-creation-plugin`, which allows creating an instance model for cloud applications deployed using OpenTOSCA Container [3]. (ii) The `kubernetes-model-creation-plugin`, which allows for the creation of an instance model for cloud applications managed by the Kubernetes container orchestration technology. (iii) The `manual-model-creation-plugin`, which allows retrieving an existing EDMM instance model created manually using an external tool, e.g., text editor or Winery[2]. This plugin is helpful for creating an initial instance model if the used IaC tool does not have an internal representation of the state of the managed application instances.

Furthermore, we created three *instance model refinement plugins*: (i) The `docker-refinement-plugin`, which updates the instance model with information about all the Docker containers hosted on any of the Docker engines present in the input instance model. For example, this plugin is useful for identifying "rogue" containers that

[1]The source code for IaCMF and all the plugins is available in the online dataset: https://doi.org/10.5281/zenodo.8252989 and on Github: https://github.com/IAC2-Project
[2]https://winery.readthedocs.io/en/latest/

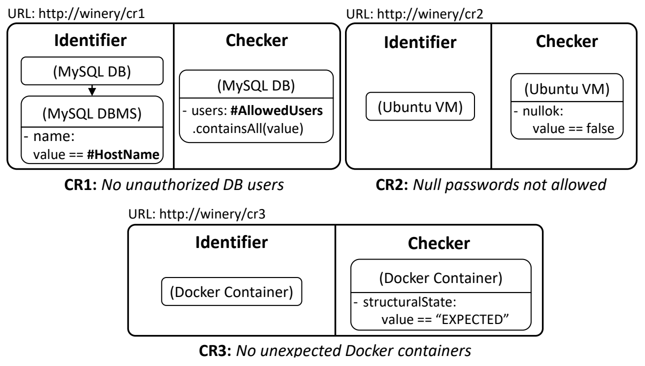


Figure 6: Technical compliance rules that detect the violations depicted in Figure 1.

are unknown to the IaC tool. (ii) The `mysql-db-model-refinement-plugin`, which updates the instance model with information about all DB users of every MySQL DBMS present in the instance model. (iii) The `bash-refinement-plugin`, which allows running a customizable `bash` script over `ssh` on any Linux VM component represented in the input instance model. The script retrieves and processes information from the accessed component and stores the resulting value in a customizable property in the corresponding node in the EDMM instance model. For example, this plugin is useful for retrieving information on the security configuration of a Linux VM.

Moreover, we created three *violation fixing plugins*: (i) The `docker-container-issue-fixing-plugin`, which allows fixing violations related to having unexpected Docker containers running in the application instance. (ii) The `remove-mysql-db-users-fixing-plugin`, which allows fixing violations related to having unauthorized users for MySQL DBs. (iii) The `bash-fixing-plugin`, which allows running a customizable `bash` script over `ssh` on a Linux VM to fix violations related to it.

Finally, we created an *execution reporting plugin* named `smtp-email-sending-plugin` that sends a human-readable compliance job execution report to a customizable email address using SMTP, and we created a *compliance checking plugin* named `subgraph-matching-checking-plugin` that extends the approach proposed by Krieger et al. [19]. We provide details about this plugin in the next section.

6.2 Demonstrative Use Case

We show how to model a technical compliance rule that detects a compliance violation introduced in Section 1, in which an unexpected user is given access to a MySQL DB that is a part of an IaC-based cloud deployment. We also discuss how to configure a corresponding *Compliance Job*, which is accomplished using the UI frontend of the IaCMF prototype. To model the technical compliance rule, we use and extend the approach proposed by Krieger et al. [19]. In this approach, a technical compliance rule is modeled as a pair of EDMM graphs, an *Identifier* and a *Checker*. The Identifier is used to find matching subgraphs in the instance model. Each of the matches represents a set of software components that are subject to the compliance rule. The Checker describes the conditions that make these software components compliant according to the rule.

```
---
complianceRules:
- complianceRuleId: CR1
  complianceRuleFormat: subgraph-
    matching
  location: http://winery/cr1
  complianceRuleParams:
  - parameterId: HostNameParam
    name: HostName
    type: String
  - parameterId: AllowedUsersParam
    name: AllowedUsers
    type: StringList

productionSystems:
- productionSystemId: >-
    MySimple3TierApp
  iacToolType: OpenTOSCA
  modelCreationPluginConfig:
    pluginId: >-
      opentosca-container-
      model-creation-plugin
    pluginConfigParams: []
  prodSystemConfigParams:
  - name: OpenToscaEngineIp
    value: 192.168.1.61
  - name: appInstanceId
    value: 155
```

```
complianceJobs:
- complianceJobId: MySqlUserAuthChecks
  complianceRuleConfigurations:
  - complianceRuleId: CR1
    violationType: UNAUTHORIZED_DB_USER
    parameterAssignements:
    - parameterId: HostNameParam
      value: financeDB
    - parameterId: AllowedUsersParam
      value: financeSysUser
  productionSystemId: MySimple3TierApp
  refinementStrategyId: refinementStrategy1
  checkingPluginConfug:
    pluginId: >-
      subgraph-matching-checking-plugin
    pluginConfigParams: []
  fixingStrategyId: fixingStrategy1
```

Listing 1: Snippet from a *Compliance Job* configured to use the compliance rule CR1.

Figure 6 depicts three technical compliance rules designed to detect the compliance violations shown in Figure 1. We focus on rule *CR1*, which requires that every MySQL DB we find when matching the EDMM instance model with the Identifier must only have users that are within the list of expected users. Specifically, the Identifier is modeled as a MySQL DB node hosted on a MySQL DBMS node that has a customizable hostname (`#HostName`).When applied to the EDMM instance model, this will match with any MySQL DB that is hosted on a MySQL DBMS with this specific hostname. Furthermore, the Checker is modeled as a single MySQL DB node that poses a requirement on the set of users allowed to have access to it via the `#AllowedUsers` parameter.

The approach, which is implemented by the `subgraph-matching-checking-plugin`, uses the VF2 algorithm for subgraph isomorphism [4]. During matching, two software components are considered to be equal if their types match according to a predefined type hierarchy *and* their property values match. We enhance the original approach [19] by allowing the use of Boolean expressions, e.g., `value == #HostName`, as property values in the Identifier and Checker, thus facilitating more expressive matching with instance models. The expressions use the *Spring Expression Language* [35] and may have parameters, e.g., `#HostName`, that get a concrete value when the rule is used in a *Compliance Job*. We extended Winery, a TOSCA and EDMM modeling tool, to support modeling such compliance rules and accessing them via a REST API.

As discussed in Section 5, in order to check and enforce technical compliance rules against a given application instance, a framework user needs to use the frontend UI to configure a set of entities that tell the framework how to run the Compliance Checking and Enforcement Process. In Listing 1, we see a YAML snippet of the configuration needed to check and enforce the technical compliance rule CR1 discussed above. Specifically, the user starts with configuring a reusable *Compliance Rule* entity that represents CR1. The entity indicates that the technical compliance rule can be retrieved from Eclipse Winery using the URL "http://winery/cr1" and that the rule's format is "subgraph-matching", which corresponds to the format presented in Figure 6. Furthermore, the entity declares two customizable parameters: "HostName" and "AllowedUsers". Next, the user configures a reusable *Production System* entity specifying OpenTOSCA Container as the IaC tool used to manage the application instance under consideration, named "MySimple3TierApp", and provides information on how to communicate with it using the `opentosca-container-model-creation-plugin`.

Finally, the user creates a *Compliance Job* entity that points to the *Compliance Rule* and *Production System* previously created. Additionally, it specifies concrete values for the parameters of CR1 stating that the hostname of the considered DBMS is "financeDB" and the only user allowed to access the DB is "financeSysUser". Furthermore, it specifies that if CR1 is found to be violated, a Violation Type named "UNAUTHORIZED_DB_USER" will be reported. Lastly, the job specifies that the plugin used to parse the included technical compliance rule and execute the compliance checking step is `subgraph-matching-checking-plugin`, which corresponds to the subgraph isomorphism-based approach discussed earlier. For brevity, we skip describing the configuration of the other steps of the Compliance Checking and Enforcement Process (see Section 4.2). To demonstrate the usage of the framework and to assist reproducibility, we recorded a video[3] showing how to use IaCMF to configure and execute the *Compliance Jobs* that correspond to all the compliance rules shown in Figure 6.

7 EVALUATION

In Section 3, we discussed the research method used in this work, which follows the design-science methodology [36]. In this section, we give further details about how we evaluated the resulting IT artifacts, namely, the RICMa method and IaCMF, which is depicted as the last step in Figure 2. Specifically, we conducted a qualitative interview study. We started by designing the compliance use cases we presented in Section 1. Furthermore, we designed an *interview guide*, which comprises the questions to be asked during interviews and their order, the criteria used for selecting candidate participants, and a *fundamentals document* containing background information about the RICMa method and IaCMF. Next, we recorded a video showing the implementation of two of the three compliance use cases using the framework. Then we invited potential participants and requested them to watch the video and read the fundamentals document before conducting the interviews. We held online individual interviews with the participants that lasted 40–60 minutes. We recorded the interviews, transcribed them using the Whisper speech recognition tool [28], manually refined the transcripts, anonymized them, and analyzed them by finding common and interesting opinions among the participants. A dataset containing the interview guide and all transcripts is available online[4].

The interview included 24 questions, which focused on demographic information (summarized in Table 2) and on the following expected attributes associated with using the RICMa method and IaCMF: (i) the *reduction of effort*, (ii) the *reduction of complexity*, (iii) the *reduction of uncertainty*, and (iv) *novelty*.

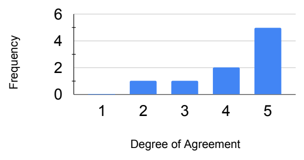
[3]Available at: https://clipchamp.com/watch/Geqe70aPkjh
[4]Available at: https://doi.org/10.5281/zenodo.8252989

Table 2: Demographic information for interview participants.

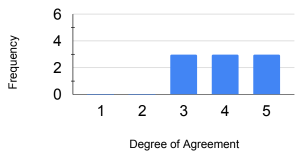
Participant	Role	Years of IaC Experience	Company	Domain	Employees
P1	Senior Software Developer	11	C1	Technology	<50
P2	Software Developer	4			
P3	CTO	10			
P4	System Manager	8	C2	Technology	>100k
P5	System Admin	6			
P6	Researcher	3	C3	Automotive	>100k
P7	Software Architect	12	C4	Telecom.	50k .. 100k
P8	Software Architect	9	C5	Technology	>100k
P9	Network Admin	6	C6	IT Management	250 .. 2k

Five participants have reported that they define lists of free-text compliance rules to be followed in their current practices. For example *P1* stated, "We have a kind of semi-comprehensive sheet that lists common security issues when it comes to running software in the cloud or over the internet." Furthermore, six participants have reported that checking compliance rules against running application instances is done manually. For example, *P2* disclosed, "On the infrastructure levels of Kubernetes, we don't have automatic checkings: so, this is done by us by manual checking and looking." Hence, it is evident that defining compliance rules currently is not a complex task, but checking them is. Therefore, all participants agreed that using the framework to define compliance rules takes more effort and is more complex than their current practices, but when the rules exist, *checking them becomes very simple and effortless*. For example, *P2* argued, "I think once you have your rule, really asking (the framework) to (perform) checking, is no effort (...) you shift the complexity to defining your rules. But when you define them, there is no complexity during checking, because you have your rules defined as machine-readable instructions." Moreover, seven participants agreed that using well-defined, machine-readable models for compliance rules *reduces the uncertainty associated with interpreting them*. For example, *P6* emphasized, "Maybe it's (manageable) for smaller systems, but if you reach a certain level of complexity, I think you need some language or model you can rely on. For large distributed systems. It's difficult to do things without models." Nonetheless, *P7* argued that uncertainty may still arise since the quality of modeled rules is not ensured: "How do I check the compliance rules for quality? (...) I (should be able to) inspect and improve the rules."

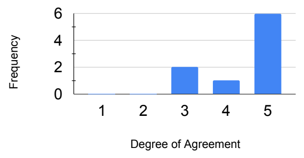
Violation fixing is a step in the RICMa method. Some participants said that they already applied semi-automatic fixes to address compliance violations. This is done by manually changing the IaC code and automatically redeploying it using an existing CI/CD pipeline. For example, *P4* stated, "Once we know which (of the infrastructure resources) is affected, then we need to define an action plan: Can we reinstall it? Can we test it? How will this affect all other applications? (The changes) will be tested in the staging environment, and then rolled out to production." Participants *P1*, *P2*, *P3*, and *P5* use a similar practice. Nonetheless, fixing compliance violations related



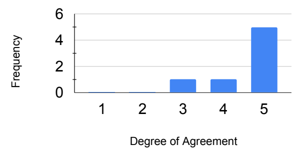
(a) Using IaCMF reduces the effort associated with defining and checking compliance rules.



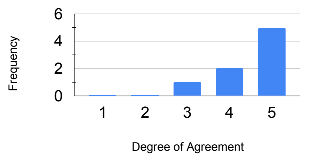
(b) Using IaCMF reduces the complexity associated with defining and checking compliance rules.



(c) Using well-defined models for compliance rules reduces the uncertainty associated with interpreting them.



(d) Using IaCMF reduces the effort associated with fixing compliance violations (2 participants did not give concrete answers).



(e) Having well-defined models for compliance jobs reduces the uncertainty associated with handling detected violations (1 participant did not give a concrete answer).

Figure 7: Responses to Likert-style interview questions (1 represents "totally disagree" and 5 represents "totally agree").

to the parts of the infrastructure not (yet) incorporated into the CI/CD pipeline is done manually. For example, *P1* acknowledged, "For stuff like managed services, we apply manual configurations." The other participants stated they currently only employ manual violation fixing. All participants agreed that using the framework *would reduce the effort associated with fixing compliance violations* to some degree, but only if a suitable plugin exists. For example, *P6* argued, "If you do have the plugins suitable for your use case, (then) the effort is like almost no effort." Furthermore, most participants agreed that using *well-defined compliance jobs reduces the uncertainty associated with handling detected compliance violations.* For example, *P7* stated, "A compliance job provides structure because it provides a set of interrelated compliance rules in the same application. One has some feeling of the application."

Moreover, all participants agreed that the RICMa method is *novel* in that it combines all compliance management tasks in one process. For example, *P2* claimed, "At least for me it was something new. So you detect it and you fix it all at once." Furthermore, *P7* highlighted that being domain-specific for compliance checks gives IaCMF an advantage over using CI tools for compliance management: "(Using CI for compliance management) is the most possible general framework and I have to have the mental mapping from the CI checks to the compliance rules, and IaCMF does that for me, and then I can also have more coding-far-out people, e.g., security experts, using this framework." Finally, Figure 7 summarizes the responses to five Likert-style questions that asked how much each participant agreed to the corresponding statements.

8 THREATS TO VALIDITY

In this section, we discuss aspects that might threaten the validity of our approach and the qualitative interview study.

External Validity: This type of validity pertains to the generalizability of the results. Starting with the approach, one threat to external validity is handling *legacy systems* that do not utilize IaC tools. Of course, if there are individual legacy systems that cannot be recognized with the common instance model creation plugins we use for modern applications and for which the effort would be too high to write dedicated plugins, we recommend creating instance models manually, e.g., using the `manual-model-creation-plugin` introduced in Section 6.1. Note that checking the compliance of these systems works similarly to other systems due to the usage of a technology-agnostic language to represent instance models, i.e., EDMM (see Section 4.2.1). Another potential threat to external validity is the *performance* of IaCMF. If the execution of the Compliance Checking and Enforcement Process lacks sufficient performance, usability will be greatly affected. However, the performance of IaCMF is mainly driven by the used plugins rather than the framework itself. For example, if human intervention is needed for certain plugins, performance will inevitably be affected. Lastly, if the *accuracy* of compliance checking is low, usability will also be affected. Nonetheless, this is predominantly decided by the accuracy of the modeled compliance rules and not IaCMF itself.

A potential threat to the external validity of the interview study pertains to the non-random selection of interviewees, as we specifically reached out to certain experts. This affects the extent to which our findings can be applied to practitioners in different companies. To address this threat, we made sure to select participants with different roles that work for companies with varying sizes and domains. Additionally, the number of interviews conducted with practitioners is insufficient for statistical generalization.

Internal Validity: The main concern here is how we select the interview participants and the potential bias that may arise. To mitigate this threat, specific decisions were made. For example, to ensure that the evaluation was conducted by individuals with relevant expertise, all selected participants were practitioners with extensive experience in software engineering and a solid understanding of IaC technologies stemming from an industrial background rather than a pure academic background.

Construct Validity: During individual interviews, practitioners were given the opportunity to freely express their opinions without any interruption from the researchers. To ensure this environment, we adhered to established guidelines, such as those outlined in a reputable source like [16] for conducting interviews with practitioners. These guidelines served as a reference to maintain a respectful and open atmosphere that encouraged practitioners to share their thoughts and perspectives without any interference.

9 CONCLUSION AND FUTURE WORK

In this work, we have tackled the problem of defining a method to check compliance rules against IaC-based cloud deployments at runtime and fix the possible violations while reducing the associated complexity, effort, and uncertainty. To this end, we followed the design-science methodology to design and develop (i) the RICMa method, which is capable of solving the aforementioned research problem, and (ii) IaCMF, an extensible framework that supports executing the RICMa method.

We evaluated the usefulness of the RICMa method and IaCMF by conducting nine interviews with industry experts. One interesting outcome is that a careful cost-benefit analysis is needed before adopting IaCMF since using it entails significant effort for defining technical compliance rules and specialized plugins for specific use cases. This makes the framework more beneficial for companies managing a large number of IaC-based cloud deployments. Furthermore, adoption can be facilitated by providing large, publicly accessible repositories of customizable plugins and technical compliance rules for well-known catalogs, e.g., DISA STIGs [7].

A possible direction for future research is answering the question of how to integrate the RICMa method with standard DevOps pipelines, which was explicitly asked by most interview participants. Another question to be answered in future research is how we can enhance the method to ensure the quality of the authored technical compliance rules, e.g., by detecting conflicting rules when applied to the same production system, and how we can incorporate the Common Vulnerability Scoring System (CVSS) [10], which is commonly used in industry.

ACKNOWLEDGMENTS

This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Project number 314720630, and FWF (Austrian Science Fund) project IAC^2: I 4731-N.

REFERENCES

[1] Amazon Web Services. 2023. AWS Config Documentation. https://docs.aws.amazon.com/config/

[2] Matej Artac, Tadej Borovssak, Elisabetta Di Nitto, Michele Guerriero, and Damian Andrew Tamburri. 2017. DevOps: Introducing Infrastructure-as-Code. In *IEEE/ACM ICSE-C'17*. 497–498.

[3] Tobias Binz, Uwe Breitenbücher, Florian Haupt, Oliver Kopp, Frank Leymann, Alexander Nowak, and Sebastian Wagner. 2013. OpenTOSCA – A Runtime for TOSCA-based Cloud Applications. In *ICSOC'13*, Vol. 8274. Springer, 692–695.

[4] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (Sub) Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 26, 10 (2004), 1367–1372.

[5] Stefano Dalla Palma, Dario Di Nucci, Fabio Palomba, and Damian Andrew Tamburri. 2020. Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software* 170 (2020), 110726.

[6] Stefano Dalla Palma, Dario Di Nucci, and Damian A. Tamburri. 2020. Ansible-Metrics: A Python library for measuring Infrastructure-as-Code blueprints in Ansible. *SoftwareX* 12 (2020), 100633.

[7] Defense Information Systems Agency. 2023. Security Technical Implementation Guides (STIGs). https://public.cyber.mil/stigs/

[8] Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, Miles Stötzner, Evangelos Ntentos, Uwe Zdun, Martin Becker, and Elena Heldwein. 2022. On Unifying the Compliance Management of Applications Based on IaC Automation. In *IEEE ICSA-C'22*. IEEE, 226–229.

[9] Markus Fischer, Uwe Breitenbücher, Kálmán Képes, and Frank Leymann. 2017. Towards an Approach for Automatically Checking Compliance Rules in Deployment Models. In *SECURWARE'17*. Xpert Publishing Services (XPS), 150–153.

[10] Forum of Incident Response and Security Teams, Inc. 2019. Common Vulnerability Scoring System version 3.1: User Guide. https://www.first.org/cvss/user-guide

[11] Google Cloud. 2023. Security Command Center. https://cloud.google.com/security-command-center

[12] Lukas Harzenetter, Tobias Binz, Uwe Breitenbücher, Frank Leymann, and Michael Wurster. 2021. Automated Generation of Management Workflows for Running Applications by Deriving and Enriching Instance Models. In *CLOSER'21*. SciTePress, 99–110.

[13] Lukas Harzenetter, Uwe Breitenbücher, Tobias Binz, and Frank Leymann. 2023. An Integrated Management System for Composed Applications Deployed by Different Deployment Automation Technologies. *SN Computer Science* 4, 370 (2023), 1–16.

[14] Klaus Havelund. 2014. Rule-based runtime verification revisited. *International Journal on Software Tools for Technology Transfer* 17, 2 (April 2014), 143–170.

[15] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. 2004. Design Science in Information Systems Research. *MIS Quarterly* 28, 1 (2004), 75–105.

[16] S.E. Hove and B. Anda. 2005. Experiences from conducting semi-structured interviews in empirical software engineering research. In *METRICS'05*. 10–23.

[17] Jez Humble and David Farley. 2010. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Addison-Wesley Professional.

[18] Christoph Krieger, Uwe Breitenbücher, Michael Falkenthal, Frank Leymann, Vladimir Yussupov, and Uwe Zdun. 2020. Monitoring Behavioral Compliance with Architectural Patterns Based on Complex Event Processing. In *Proceedings of the 8th European Conference on Service-Oriented and Cloud Computing (ESOCC 2020)*. Springer International Publishing, 125–140.

[19] Christoph Krieger, Uwe Breitenbücher, Kálmán Képes, and Frank Leymann. 2018. An Approach to Automatically Check the Compliance of Declarative Deployment Models. In *SummerSoC'18*. IBM Research Division, 76–89.

[20] Indika Kumara, Martín Garriga, Angel Urbano Romeu, Dario Di Nucci, Fabio Palomba, Damian Andrew Tamburri, and Willem-Jan van den Heuvel. 2021. The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and Software Technology* 137 (2021), 106593.

[21] Indika Kumara, Zoe Vasileiou, Georgios Meditskos, Damian A. Tamburri, Willem-Jan Van Den Heuvel, Anastasios Karakostas, Stefanos Vrochidis, and Ioannis Kompatsiaris. 2020. Towards Semantic Detection of Smells in Cloud Infrastructure Code. In *WIMS'20* (Biarritz, France). ACM, 63–67.

[22] Microsoft. 2023. Azure Policy Documentation. https://learn.microsoft.com/en-us/azure/governance/policy/

[23] Kief Morris. 2020. *Infrastructure as Code: Dynamic Systems for the Cloud*. Vol. 2. O'Reilly.

[24] National Institute of Standards and Technology. 2020. *Security and Privacy Controls for Information Systems and Organizations*. Technical Report. https://doi.org/10.6028/nist.sp.800-53r5

[25] Evangelos Ntentos, Uwe Zdun, Konstantinos Plakidas, Sebastian Meixner, and Sebastian Geiger. 2020. Assessing Architecture Conformance to Coupling-Related Patterns and Practices in Microservices. In *ECSA'20*.

[26] Evangelos Ntentos, Uwe Zdun, Konstantinos Plakidas, Sebastian Meixner, and Sebastian Geiger. 2020. Metrics for Assessing Architecture Conformance to Microservice Architecture Patterns and Practices. In *ICSOC'20*.

[27] Palo Alto Networks, Inc. 2023. Prisma™ Cloud Administrator's Guide. https://docs.paloaltonetworks.com/prisma/prisma-cloud/prisma-cloud-admin

[28] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust Speech Recognition via Large-Scale Weak Supervision. arXiv:2212.04356

[29] Red Hat OpenShift Documentation Team. 2023. Security and Compliance OpenShift Container Platform. https://access.redhat.com/documentation/en-us/openshift_container_platform/4.13/html/security_and_compliance/

[30] Karoline Saatkamp, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. 2019. An approach to automatically detect problems in restructured deployment models based on formalizing architecture and design patterns. *SICS Software-Intensive Cyber-Physical Systems* (Feb. 2019), 1–13.

[31] Tushar Sharma, Marios Fragkoulis, and Diomidis Spinellis. 2016. Does Your Configuration Code Smell?. In *MSR'16* (Austin, Texas). ACM, 189–200.

[32] Thodoris Sotiropoulos, Dimitris Mitropoulos, and Diomidis Spinellis. 2020. Practical Fault Detection in Puppet Programs. In *ICSE'20*. ACM, 26–37.

[33] Eduard van der Bent, Jurriaan Hage, Joost Visser, and Georgios Gousios. 2018. How good is your puppet? An empirically defined and validated quality model for puppet. In *IEEE SANER'17*. 164–174.

[34] VMware Aria Automation SaaS. 2023. Using Automation for Secure Hosts. https://docs.vmware.com/en/VMware-Aria-Automation/SaaS/using-automation-secure-hosts.pdf

[35] VMWare, Inc. 2023. Spring Expression Language. https://docs.spring.io/spring-framework/reference/core/expressions.html

[36] Roel J. Wieringa. 2014. *Design Science Methodology for Information Systems and Software Engineering*. Springer. https://doi.org/10.1007/978-3-662-43839-8

[37] Michael Wurster, Uwe Breitenbücher, Michael Falkenthal, Christoph Krieger, Frank Leymann, Karoline Saatkamp, and Jacopo Soldani. 2019. The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies. *SICS* 35 (2019), 63–75.

[38] Michael Zimmermann, Uwe Breitenbücher, Christoph Krieger, and Frank Leymann. 2018. Deployment Enforcement Rules for TOSCA-based Applications. In *SECURWARE'18*. XPS, 114–121.