On Unifying the Compliance Management of Applications Based on IaC Automation

Ghareeb Falazi, Uwe Breitenbücher, Frank Leymann, Miles Stötzner
Institute of Architecture of Application Systems, University of Stuttgart
Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

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

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

***Abstract*—Infrastructure-as-Code (IaC) technologies are used to automate the deployment of cloud applications. They promote the usage of code to define and configure the IT infrastructure of cloud applications allowing them to benefit from conventional software development practices, which facilitates the rapid deployment of new versions of application infrastructures without sacrificing quality or stability. On the other hand, enterprise applications need to conform to compliance regarding external regulations and internal policies. Many of these compliance rules affect the application architecture on which IaC code operates. However, managing the architectural compliance of IaC-based application deployments faces a number of challenges, such as configuration drift and the heterogeneity of IaC technologies. Therefore, in this work, we present a vision on how to uniformly manage the compliance of the infrastructure of applications that utilize heterogeneous IaC technologies for deployment automation. To this end, we introduce an initial design for the IaC-based Architectural Compliance Management Framework and discuss how it addresses the corresponding challenges.**

***Index Terms*—Compliance, Software Architecture, Software Infrastructure, Infrastructure-as-Code.**

I. Introduction

Infrastructure-as-Code (IaC) is a deployment automation approach that emphasizes the repeatability and consistency of application infrastructure provisioning and configuration [1]. IaC technologies, such as Terraform, CloudFormation, and Ansible, use code to deploy, configure, and manage the infrastructure of software applications. Depending on the technology, *IaC code* takes either the form of *declarative models* that describe the desired end-state, or *imperative scripts* that describe the specific sequence of operations to be taken by the tool. Using IaC code instead of, for example, directly issuing commands to a CLI, has the inherent benefits of conventional software development, such as the option to use version control, static code analysis tools, and so on [1]. Furthermore, common software development practices, such as *Continuous Integration* and *Continuous Delivery*, can be applied to IaC. This enables rolling out new versions of the infrastructure of a software application in rapid iterations without sacrificing quality and stability [2]. This is especially important in the cloud domain in which changes to the infrastructure of applications are common and frequently needed [3].

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

However, besides these benefits of IaC, the IT infrastructure of enterprise applications is commonly subject to a set of regulations and policies that must be adhered to. Specifically, enterprise application infrastructures might be subject to: (i) *externally enforced compliance rules*, e.g., branch-specific guidelines and regional laws, such as the *European General Data Protection Regulation (GDPR)* [4], and (ii) *internally enforced compliance rules*; e.g., in the form of policies that must be followed by all DevOps teams, such as requiring all mission-critical data to be hosted on-premise rather than on a public cloud. Thus, an enterprise that utilizes IaC technologies for deployment automation needs to ensure that the corresponding IaC code adheres to such compliance rules. However, this faces a number of challenges: First, ensuring the compliance of IaC code only during design-time is not enough due to *configuration drift*, i.e., possible ad-hoc infrastructure changes that happen during runtime [1]. Second, representing IaC compliance rules in a uniform and machine-readable format that facilitates automatic checking and enforcement is difficult because the various IaC technologies offer heterogeneous sets of features and use different formats; thus, the corresponding IaC scripts and models to be checked are highly polyglot. Finally, introducing fixes to non-compliant IaC code is also difficult since it requires substantial technical expertise, especially if multiple IaC technologies are used.

Therefore, the goal of this research is to ***present a concept on how to uniformly manage the design-time and runtime compliance of the infrastructures of software applications that utilize heterogeneous IaC technologies for deployment automation.*** To this end, we propose a *uniform compliance rule format* capable of handling the heterogeneity of IaC technologies. We also introduce an initial design for the *IaC-Based Architectural Compliance Management Framework (IACMF)*, which facilitates: (i) the management of compliance rules in terms of modeling, updating, deleting, etc., (ii) checking and enforcement of compliance rules both at design-time and at runtime, and (iii) the continuous measurement of the enhancement or degradation of the compliance of the software application infrastructure as changes to IaC code occur.

II. Background and Research Challenges

In this paper, we define the term *compliance* as the state in which a considered IT system follows a set of predefined rules. Therefore, *compliance management* is the process of ensuring compliance in a certain domain, which is done by identifying the applicable rules and transforming them into technical measures that determine if a system is compliant or not. Then, automated or manual corrective actions are performed if necessary and a report is generated for auditing purposes [5]. In this work, we address *IaC-based architectural compliance*, which checks the compliance of the IT resources provisioned and configured by IaC technologies for hosting an application system, e.g., compute, storage, and networking resources. Moreover, it is often necessary to use a complementary set of IaC technologies together, since some of them are specialized in the provisioning of resources, while others focus on configuration management. Thus, IaC technologies often need to be combined—especially in large-scale application systems that run in different environments [6].

Therefore, *IaC-Based Architectural Compliance Management* faces several challenges that need to be addressed by the envisioned framework: (**C1) IaC technologies are heterogeneous**, because they are not uniform in terms of their purpose and they employ different scripting languages, modelling languages, data formats, and interfaces, i.e., different APIs and CLIs [7]. Nonetheless, as it is often necessary to utilize multiple of these technologies simultaneously, a framework designed to manage IaC-based architectural compliance has to handle this heterogeneity. (**C2) Application infrastructure and compliance rules may change at runtime.** IaC code is delivered into production via a delivery pipeline that ensures proper testing and integration. However, ad-hoc changes directly made to the running infrastructure of the application might still take place using tools such as Ansible, which is known as *configuration drift* [1]. Furthermore, changes in the regulations or internal policies may require changes to corresponding compliance rules. For both reasons, even if the *designed deployment* of an infrastructure is compliant, the *running infrastructure* might be non-compliant due to ad-hoc alterations during runtime or because the compliance rules changed after deployment. Therefore, the proposed framework should check compliance periodically at runtime. Note that design-time checks are still necessary, because fixing architectural compliance violations of the infrastructure during runtime may be very costly for running applications. Therefore, preventive measures in the form of design-time compliance checking are preferable. (**C3)** If the designed or running infrastructure of an application is found to violate one or more compliance rules, it must be brought back into a compliant state. However, **correcting IaC scripts and models or running infrastructure instances due to compliance violations requires substantial technical expertise**. Thus, the framework should automatically suggest corrections and, for critical rule violations, it should support a mode that directly enforces the changes.

III. Related Work and Previous Work

Plenty of approaches address the **detection of specific characteristics in IaC code**, such as code smells. For example, Kumara et al. [8] introduce an approach for the semantic detection of smells and anti-patterns in deployment models of cloud applications. Palma et al. [9] propose a catalog of metrics to understand and improve the quality of Ansible scripts. Sotiropoulos et al. [10] introduce an approach to detect faults in Puppet models involving ordering violations and notifiers by analyzing the system call traces produced when executing these models. All of these approaches aim to detect predefined characteristics in IaC code, but none of them introduces a general purpose compliance management framework that detects and enforces user-defined rules. On the other hand, many **business process compliance management approaches** exist (summarized in [11]), but none of them addresses the compliance of the IT infrastructure layer.

One goal of our proposed approach is supporting **compliance rule modeling and checking**. Existing approaches solve this problem in different ways. For example, in a previous work [12], we introduced the concept of a *Deployment Compliance Rule*, which consists of two constructs: the *Detector*, which is a subgraph used to detect areas in the deployment model that are subject to the rule, and the *Required Structure*, which is another subgraph used to verify that the rule is satisfied. Matching these constructs with the deployment model utilizes subgraph isomorphism. Another possibility, is to use first-order logic instead of subgraph isomorphism by modeling compliance rules as Prolog or Datalog rules as suggested in [13]–[15]. These approaches entail increased difficulty in rule modeling, and an extra step to transform all relevant elements of the deployment model into first-order logic facts, so rules can be evaluated. However, they allow defining rules that indicate, e.g., the absence of undesired model structures, which is not possible in the subgraph-based approach.

Checking architectural compliance, both at design-time and at runtime, requires **reconstructing the system's architecture** in the form of a deployment model. There are multiple approaches that can be utilized for this purpose. For example, a previous approach [16] uses technology-specific *discovery plugins* that crawl running instances of the application system for the automated reconstruction and maintenance of *Enterprise Topology Graphs*. An enhancement of this approach by Harzenetter et al. [17] starts from an initial deployment model that is (re)constructed using plugins specific to the IaC technologies used to manage the running application instance. Then technology-specific plugins (similar to the ones from the previous approach) are used to complete the deployment model with technology-specific information. Another approach [18] utilizes reusable detectors that analyze the code base looking for specific characteristics that help to discover components and their relations. This approach is less versatile than the previous two, but is able to accurately find more relations between the discovered components.

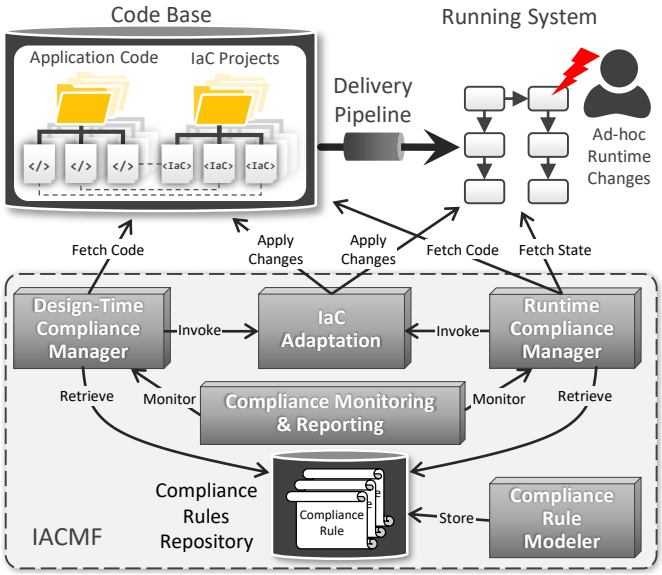

Fig. 1. Delivery pipeline (top) and architecture of the IACMF (bottom)

IV. A Framework for Uniform IaC-Based Architectural Compliance Management

At the top of fig. 1, we see a delivery pipeline for applications that use IaC to define and configure their infrastructures. The code base is split into code projects that represent application logic, and projects that include IaC code, i.e., IaC models and scripts used to (i) define the IT infrastructure of a software application, e.g., using Terraform models, and (ii) to configure it, e.g., using Ansible scripts. These projects can reference each other, and such *cross-project references* (in addition to references to external libraries) represent dependencies that are resolved in the delivery pipeline using dedicated integration tools. Other tasks done in the delivery pipeline include building projects with tools such as Jenkins, and testing.

At the bottom of fig. 1, we see our proposed framework IACMF, which first facilitates the **modeling and storage of generic reusable *IaC Architectural Compliance Rules***. Furthermore, it facilitates **design-time compliance management** that ensures all relevant compliance rules are checked against the involved code projects. In addition, the framework supports **runtime compliance management**, ensuring the continuous checking of compliance rules while the system is running. Besides, IaC-based compliance management, both at design-time and at runtime, involves **adapting non-compliant IaC scripts or models or even IaC-managed infrastructures** so that they are brought back into a compliant state. Finally, the framework allows the **measurement and reporting of compliance enhancement or degradation** based on changes introduced to the code base. Next, we briefly sketch our vision on how to achieve these features while addressing the entailed challenges, which we refer to using the same identifiers presented in section II, i.e., (**C1**), (**C2**), and (**C3**).

A. Handling the Heterogeneity of IaC Technologies

For automating the deployment of applications, often multiple IaC technologies need to be combined [6], which constitutes a challenge for architectural compliance management frameworks (**C1**). We propose to solve this problem via a **plugin-based approach that generates IaC technology-agnostic deployment models**. In a previous work [7], we introduced the *Essential Deployment Metamodel (EDMM)*, which was derived by systematically analyzing deployment technologies and distilling a common denominator of their features: An *EDMM model* represents the application and infrastructure components as a graph-based deployment model, which consists of typed *components* (vertices) that are connected via typed *relations* (edges). Components and relations can be enriched with *properties*, which describe their current or desired state. To solve the IaC heterogeneity problem, we propose to transform technology-specific IaC scripts and models into an EDMM model, e.g., by extending our previous work [19] that utilizes technology-specific plugins. Based on this IaC technology-agnostic model, architectural compliance rules can be defined more easily by limiting their lexicon only to the model elements defined by EDMM rather than the heterogeneous set of features of IaC technologies. This isolates the complexity into the technology-specific plugins, allowing to create generic and future-proof compliance rules.

B. IaC Architectural Compliance Rule Modeling

An IaC Architectural Compliance Rule encodes one or more regulations or policies that the infrastructure must adhere to. For example, a rule could dictate that web applications and the databases connected to them must be hosted in the same region so that sensitive data do not move across regions, or it could dictate that no mission-critical databases are hosted on a public cloud. We reuse the idea that such rules contain two parts: a *detector* and an *evaluator* [12]. An evaluator allows the framework to determine if the architecture fulfills the rule. Moreover, it is foreseeable that not all defined rules are applicable to the current system and that the evaluation of certain rules could be time- or resource-intensive. Therefore, the detector is introduced as a lightweight mechanism to allow the framework to determine if a rule has to be evaluated for the architecture or not. Detectors and evaluators are generic concepts that can be implemented using various techniques, e.g., using subgraph isomorphism [12] or Prolog rules [13] as shown in previous works. We only mandate that they work on EDMM models, since EDMM is IaC technology-agnostic, and thereby increases the reusability and portability of the rules.

C. Design-Time Architectural Compliance Management

The *Design-Time Compliance Manager* checks the architectural compliance of the input IaC code during design-time, following the process described in fig. 2. First, the application architecture including its infrastructure is (re)constructed as an EDMM model. However, since architectural reconstruction approaches have trade-offs (see section III), a combination of them is often required, which we support using **plugins**.

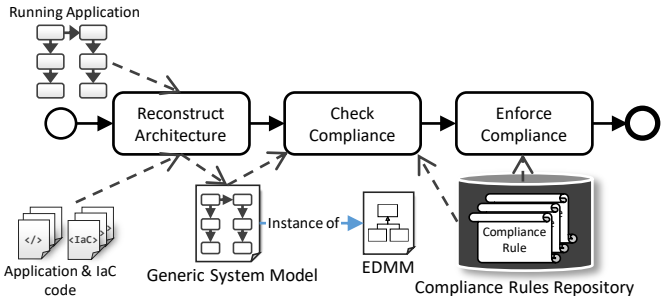

Fig. 2. Architectural compliance checking and enforcement process.

After generating the EDMM model, relevant compliance rules are detected and evaluated also using plugins since different implementations that correspond to different modeling approaches are possible, e.g., using subgraphs or first-order logic (see section IV-B). If a violation of certain rules are detected, corrective measures are taken by the *IaC Adaptation* component **(C3)**. Depending on the severity of the violation, this component either automatically enforces changes to the IaC code, or suggests them to a human operator. Generating these fixes also requires **technology-specific plugins**.

D. Runtime Architectural Compliance Management

The *Runtime Compliance Manager* component ensures compliance of running application instances. It also executes the process depicted in fig. 2 with two differences from the design-time variant: (i) Since running application instances might be modified at runtime in an ad-hoc manner **(C2)**, the (re)construction of their architecture in Step 1 depends on both the code projects used in the deployment pipeline and on information about the running instance. We discuss suitable approaches for this in section III. However, further research is required to enable the identification of arbitrary communication relations between components, which typically requires combining various reconstruction techniques. (ii) In Step 3, the *IaC Adaptation* component handles fixing detected compliance violations **(C3)** in one of two ways: it either makes the original IaC code compliant and then re-executes the delivery pipeline to guarantee that the ad-hoc changes applied to the running instance are revoked making it compliant agian, or if re-executing the pipeline is not a viable option, e.g., due to possible interruptions to the production environment [1], fixes are delivered directly to the running infrastructure, e.g., via invocations to the APIs of the used IaC technologies.

E. Continuous Architectural Compliance Measurement

Detecting compliance violations enables IaC architects to get feedback on possible improvements and degradation of the quality of the IaC code and architecture. The IACMF supports the continuous measurement of a system's IaC compliance by measuring, e.g., the number of rule violations and the impact of different IaC changes on this number. Such metrics will be automatically monitored and reported back to architects by the *Compliance Monitoring and Reporting* component.

V. Conclusion and Future Work

We proposed a framework for managing the architectural compliance of IaC-based deployments despite the heterogeneity of IaC technologies by generating IaC technology-agnostic models and using an IaC technology-independent compliance rule format. The envisioned framework also enables continuosly checking the compliance of running applications and takes corrective measures if required, e.g., for configuration drift. Apart from implementing the IACMF and evaluating it, we also plan to research ways to formalize and incorporate compliance rules about the IaC-based deployment delivery pipelines themselves, thus covering the IaC compliance of all lifecycle stages.

References

[1] K. Morris, *Infrastructure as Code*, 2nd ed. O'Reilly Media, Inc., 2020.

[2] R. Modi, "CI/CD with Terraform," in *Deep-Dive Terraform on Azure*. Berkeley, CA: Apress, 2021, pp. 163–190.

[3] D. Farley, *Continuous delivery : a handbook for building, deploying, testing and releasing software*. Addison-Wesley Professional, 2010.

[4] (2018, May) General Data Protection Regulation. European Commission. [Online]. Available: https://gdpr-info.eu/

[5] F. Koetter, M. Kochanowski, T. Renner, C. Fehling, and F. Leymann, "Unifying Compliance Management in Adaptive Environments through Variability Descriptors (Short Paper)," in *SOCA 2013*. IEEE, 2013.

[6] M. Wurster *et al.*, "Automating the Deployment of Distributed Applications by Combining Multiple Deployment Technologies," in *CLOSER 2021*. SciTePress, May 2021, pp. 178–189.

[7] ——, "The Essential Deployment Metamodel: A Systematic Review of Deployment Automation Technologies," *SICS Software-Intensive Cyber-Physical Systems*, vol. 35, pp. 63–75, Aug. 2019.

[8] I. Kumara *et al.*, "Towards Semantic Detection of Smells in Cloud Infrastructure Code," in *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*. ACM, jun 2020, pp. 63–67.

[9] S. Dalla Palma, D. Di Nucci, F. Palomba, and D. A. Tamburri, "Toward a catalog of software quality metrics for infrastructure code," *Journal of Systems and Software*, vol. 170, p. 110726, dec 2020.

[10] T. Sotiropoulos, D. Mitropoulos, and D. Spinellis, "Practical fault detection in puppet programs," in *ACM/IEEE ICSE 2020*. ACM, 2020.

[11] S. Saralaya, V. Saralaya, and R. D'Souza, "Compliance Management in Business Processes," 2019, pp. 53–91.

[12] C. Krieger, U. Breitenbücher, K. Képes, and F. Leymann, "An Approach to Automatically Check the Compliance of Declarative Deployment Models," in *Papers from the 12th Advanced Summer School on Service-Oriented Computing*. IBM Research Division, 2018.

[13] K. Saatkamp, U. Breitenbücher, O. Kopp, and F. Leymann, "An approach to automatically detect problems in restructured deployment models based on formalizing architecture and design patterns," *SICS Software-Intensive Cyber-Physical Systems*, pp. 1–13, Feb. 2019.

[14] C. Deiters, P. Dohrmann, S. Herold, and A. Rausch, "Rule-Based Architectural Compliance Checks for Enterprise Architecture Management," in *EDOC 2009*. IEEE, 2009, pp. 183–192.

[15] M. Montanari, E. Chan, K. Larson, W. Yoo, and R. H. Campbell, "Distributed security policy conformance," *Computers & Security*, vol. 33, pp. 28–40, mar 2013.

[16] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, "Automated Discovery and Maintenance of Enterprise Topology Graphs," in *Proceedings of the 6th IEEE International Conference on Service Oriented Computing and Applications (SOCA 2013)*. IEEE, Dec. 2013, pp. 126–134.

[17] L. Harzenetter *et al.*, "Automated Generation of Management Workflows for Running Applications by Deriving and Enriching Instance Models," in *CLOSER 2021*. SciTePress, May 2021, pp. 99–110.

[18] E. Ntentos *et al.*, "Detector-based component model abstraction for microservice-based systems," *Computing*, vol. 103, no. 11, pp. 2521–2551, nov 2021.

[19] C. Endres, U. Breitenbücher, F. Leymann, and J. Wettinger, "Anything to Topology - A Method and System Architecture to Topologize Technology-Specific Application Deployment Artifacts," in *CLOSER 2017*. SciTePress, Apr. 2017, pp. 180–190.