Assessing Architecture Conformance to Security-Related Practices in Infrastructure as Code Based Deployments

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

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

***Abstract*—Infrastructure as Code (IaC) enables developers and operations teams to automatically deploy and manage an IT infrastructure via software. Among other uses, IaC is widely used in the context of continuously released deployments such as those of microservice and other cloud-based systems. Although IaC-based deployments have been utilized by many companies, there are no approaches on checking their conformance to architectural aspects yet. In this paper, we focus on security-related practices including observability, access control, and traffic control in IaC-based deployments. While best practices for this topic have been documented in some gray literature sources such as practitioners' blogs and public repositories, approaches enabling automated checking of conformance to such best practices do not yet exist. We propose a model-based approach based on generic, technology-independent metrics, tied to typical architectural design decisions on IaC-based deployments. With this approach, we can measure conformance to security-related practices. We demonstrate and assess the validity and appropriateness of these metrics in assessing a system's conformance to practices through regression analysis.**

***Index Terms*—Infrastructure as code, metrics, software architecture, modeling, best practices**

I. Introduction

Cloud computing has significantly increased the number of infrastructure nodes that a system requires [1]. Moreover, nowadays systems are being released to production more rapidly, often many times a day, resulting in more and more frequent changes in the infrastructure [2], [1]. *Infrastructure as Code (IaC)* is the management and provisioning of the infrastructure using reusable scripts instead of manual processes [3]. IaC can ensure that a provisioned environment remains the same every time it is deployed in the same configuration, and configuration files contain infrastructure specifications making the process of editing and distributing configurations easier [3], [4]. IaC can also contribute to improving consistency, security, avoiding errors, and reducing manual configuration effort. Without an IaC practice in place, it becomes increasingly difficult to manage the scale of current systems' components and infrastructure.

IaC technologies (e.g., Ansible, Terraform) enable to provision, deploy, and configure arbitrary application architectures. Thus, developers and operators, respectively, can model any desired deployment. This freedom quickly results in severe problems if security-related aspects are not taken into account. For example, vulnerabilities in IaC deployment models (e.g. weak access and traffic control) could allow attackers to access procedures and run code to hack the application.

The focus of this work is to investigate security-related practices in deployment architectures that can be configured and managed via IaC scripts. In this context, we formulate a number of *Architectural Design Decisions (ADDs)* with corresponding decisions options that have been documented as best practices in the gray literature, i.e. informal guidelines for practitioners, blog posts, public repositories, and so on. Based on this architectural knowledge, we aim to provide an automatic assessment of architecture conformance to these practices in the IaC deployment models.

So far, not many architectural patterns or formal guidelines that reflect security best practices in the context of IaC have been documented. Currently, the literature seems to rather focus on specific code-level practices. In combination with the fact that industry-scale systems support multiple such architectural practices at once and different implementations of them, this makes it difficult to assess whether an IaC deployment model that implicitly describes also the application's architecture is conforming to recommended best practices or not. In modern cloud-based architectures, such as microservice architectures [5] and other frequently released systems [1], an automatic assessment method would vastly improve cost-effectiveness and produce more accurate results. For instance, this could be applied in the context of continuous delivery practices employed in these systems requiring the automated setup of infrastructures in usually every run of the delivery pipeline [2]; for example, consider production environments and identical test environments. Therefore, this paper aims at answering the following research questions:

- **RQ1** How can conformance to IaC architecture security-

related decision options (i.e. patterns or practices) be automatically assessed?
- **RQ2** What kinds of measures can be applied to asses this conformance and how well do they perform?
- **RQ3** What is the minimal set of modeling elements required in an IaC deployment model to compute these measures?

To address these research questions, we introduced a set of metrics for different security-related architectural decisions to cover all their possible decision options. We derived a ground truth from a manual assessment of a set of models that evaluates their conformance to all considered decision options and their combinations. In particular, the decisions focus on *Security Observability*, *Security Access Control*, and *Security Traffic Control*. To create the ground truth, we first objectively assessed whether each decision option is supported in a system and to which extent. We derived an ordinal rating scheme to distinguish different levels of security support, and If security flaws were indeed found, the rating scheme indicates how far the conformance to best practices is nonetheless supported. Both, the rating scheme and the ground truth assessment, have been reviewed and discussed with two industrial experts (both having experience in security of microservice systems) until a consensus was reached. We then used the ground truth data to assess how well the defined metrics can predict the ground truth assessment by performing an ordinal regression analysis.

In this paper, we propose a deployment model-based approach, which uses only modeling elements that can be derived from the typical scripts used by IaC technologies. A deployment model implicitly also describes the architecture of the application to be deployed, thus, enabling assessing the conformance to architectural decisions. For this reason, it is important to be able to work with a minimal set of elements, else it might be difficult to parse them from the IaC scripts.

This paper is structured as follows: Section II compares related work. Next, we describe the research methods and the tools we have applied in our study in Section III. In Section IV we explain the decisions considered in this paper and the related patterns and practices. In Section V we report how the ground truth data for each decision is calculated. Section VI introduces our hypothesized metrics. Section VII describes the metrics calculations results for our models and the results of the ordinal regression analysis. Section VIII discusses the RQs regarding the evaluation results and analyses the threats to validity. Finally, in Section IX we draw conclusions and discuss future work.

II. Related Work

A. Related Works on Best Practices and Patterns

As IaC practices are becoming more and more popular and widely adopted by the industry, there is also more scientific research into collecting and systematizing IaC-related patterns and practices. Kumara et al. [6] present a broad catalog of best and bad practices, both language-agnostic and language-specific, that reflect implementation issues, design issues, and violations of essential IaC principles. Morris [3] presents a collection of guidance on how to manage Infrastructure as Code. In his book, there is a detailed description of technologies related to IaC-based practices and a broad catalog of patterns and practices embodied in several categories. Language-specific practices have been proposed by Sharma et al. [7] who present a catalog of design and implementation smells for Puppet. Schwarz et al. [8] present a catalog of smells for Chef. Our work also follows IaC-specific recommendations described in AWS [9], OWASP [10], [11], [12] and the Cloud Security Alliance [13], [14]. In contrast to our work, many of these works are less focused on architectural decisions in the deployment architecture of the systems to be deployed. Furthermore, these works do not support conformance checking as suggested in our work.

B. Related works on Frameworks and Metrics

a) Tool-based and Network Metrics Approaches: There are several studies that propose tools and metrics for assessing and improving the quality of IaC deployment models. Dalla Palma et al. [15], [16] propose a catalog of 46 quality metrics focusing on Ansible scripts to identify IaC-related properties and show how to use them in analyzing IaC scripts. Wurster et al. [17] present TOSCA Lightning, an integrated toolchain for specifying multi-service applications with TOSCA Light and transforming them into different production-ready deployment technologies. They also present a case study on the toolchain's effectiveness based on a third-party application and Kubernetes. A tool-based approach for detecting smells in TOSCA models is proposed by Kumara et al. [18]. Sotiropoulos et al. [19] develop a tool-based approach that identifies dependencies-related issues by analyzing Puppet manifests and their system call trace. Van der Bent et al. [20] define metrics that reflect also the best practices to assess Puppet code quality. Pendleton et al. [21] present a comprehensive survey on security metrics. It focuses on how a system security state can evolve as an outcome of cyber-attack defense interactions. They then propose a security metrics framework for measuring system-level security.

Although some of these works focus on quality assurance of IaC systems, none of them address and focus specifically on security critical concerns and measures in IaC deployment models, which is the case in our work.

b) Software Architecture Conformance Checking: In [22], [23], an approach for automatically checking the compliance of declarative deployment models during design time is introduced. The approach allows modeling compliance rules in the form of a pair of deployment model fragments. One of the fragments represents a detector subgraph that determines whether the rule applies to a given deployment model or not. If a compliance rule is found to be applicable, the second fragment determines a desired structure that the deployment model must contain. Comparing the model fragments to a given deployment model happens using subgraph isomorphism. In contrast to our study, this approach is generic and does not introduce specific compliance rules, e.g. for the security domain, and assumes the rule modeler is capable of translating best practices into compliance rules of the expected format. Moreover, it only provides a Boolean outcome indicating whether a compliance rule is violated or not, rather than indicating to what degree it is violated.

Our approach can be considered as a metrics-based, IaC-specific approach for deployment architecture conformance checking. Many approaches related to architecture conformance checking are usually based on automated extraction techniques [24], [25]. Conformance to architecture patterns [24], [26] or other kinds of architectural rules [25] can often be checked by such approaches. As proposed in our work, here techniques that are based on various interrelated metrics of IaC-related metrics to cover security related features and practices do not yet exist. Furthermore, none of these approaches treat deployment architectures as a set of nodes and connectors i.e. a deployment architecture. Also, none are able to produce assessments that combine different assessment parameters (i.e. metrics). Such metrics, if automatically computed, can be utilized as part of larger assessment models/frameworks during design and development time.

III. Research and Modeling Methods

A. Overview

Figure 1 shows the research steps followed in this study. We first studied knowledge sources related to IaC-specific security best practices from industry organizations, practitioners blogs and scientific literature [14], [13], [12], [11], [10], [27], [9], [6], [3], [8], [7]. We then analyzed the data collected using qualitative methods based on Grounded Theory [28] coding methods, such as open and axial coding, and extracted the three core IaC security-related decisions described in section IV along with their corresponding decision drivers and impacts. Based on our dataset, which contains generated models, we derived a ground truth as well as a set of metrics to measure the proportion of the support of each practice. Next, we defined a rating scheme on support or violation of best practices and refined the ground truth assessment. The generated models, the rating scheme, and ground truth assessment were independently analyzed by two industrial security experts, each with around 5 years of experience in developing and architecting cloud-based systems. We discussed each assessment with the experts until a consensus was reached and used the ground truth data to assess how well the hypothesized metrics can possibly predict the ground truth data by performing an ordinal regression analysis..

B. Model Selection Methods

This study focuses on conformance to security-related features and practices in IaC deployment models. To be able to study this, we first performed an iterative study of a variety of IaC knowledge sources. Next, we refined a meta-model, which contains all the required elements to help us reconstruct the actual architecture that gets deployed by the IaC model. For problem investigation and as an evaluation model set for eventually creating a ground truth for our study, we gathered a number of IaC systems, summarized in Table I. Each of them is taken directly from a system published by practitioners (on GitHub and/or practitioner blogs) from 9 independent sources. We mostly used search engines to find all systems used in this study. Searching in such engines might lead to selection bias and to different types of data [29]. For avoiding such a selection bias we initially started our research with keywords taken from AWS [9], OWASP [10], [11], [12] and the Cloud Security Alliance [13], [14] such as *Infrastructure Architecture Security*, *Infrastructure Hardening*, *Monitoring and Logging* and *Security Groups and Traffic Control* etc. We used major search engines (e.g. Google, Bing, DuckDuckGo) and topic portals (e.g. InfoQ, DZone) to find relevant practitioner texts.

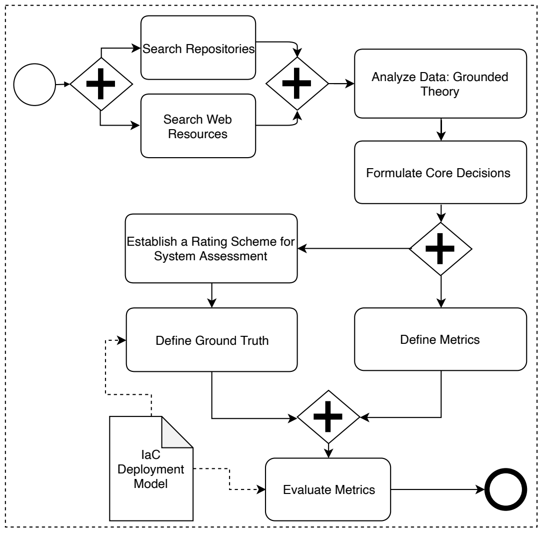

Fig. 1: Overview of the research method followed in this study

The systems we found were developed by practitioners with relevant experience, which justifies the assumption that they provide a good representation of the IaC security-related best practices summarized in Section IV. We performed a fully manual static code analysis for the IaC models that are in the repos together with the application source code. For model creation, we used our existing modeling tool Codeable Models[1], a Python implementation for the precise specification of meta-models, models, and model instances in code. The result is a set of precisely modeled component models of the software systems. Variations were modeled to cover the complete design space, including also the bad practices that can cause a violation, of our three ADDs and described in Section IV, according to the referenced practitioner sources. Apart from the variations described in Table I, all other system aspects remained the same as in the base models. This resulted in a total of 21 models summarized in Table I. We assume that our evaluation models are close to models used in practice and real-world practical needs. As many of them are open source systems with the purpose of demonstrating practices, they are

[1]https://github.com/uzdun/CodeableModels

at most of medium size, though.

C. Metrics Definition, Ground Truth Calculation, and Statistical Evaluation Methods

We defined a set of metrics for each ADD to measure conformance to the respective practices in the design decisions from Section IV, i.e. at least one metric per major feature/practice. Based on the manual assessment of the models from Table I, we derived a ground truth for our study (the ground truth and its calculation rules are described in Section V). The ground truth is established by assessing whether each decision option is supported, partially supported, or not supported. We combined the outcome of all decision's options and then derived an ordinal assessment on how well the decision is supported in each model, using the scale:

- $++$: very well supported;
- $+$: well supported, but aspects of the solution could be improved;
- $\sim$: serious flaws in the security design, but substantial support can already be found in the system;
- $-$: serious flaws in the security design, but initial support can already be found in the system;
- $--$: no support for the security tactic can be found in the system.

We discussed this assessment scheme with the two industrial security experts until a consensus was reached. The authors assessed all the system models and the variants for conformance to each of the ADDs, and the assessments are again reviewed by the two industrial security experts.

Our scale does not assume equal distances, but it assumes the given order. We then used the ground truth data to assess how well the defined metrics can predict the ground truth data by performing an ordinal regression analysis.

Ordinal regression is a widely used method for modeling an ordinal response's dependence on a set of independent predictors. For the ordinal regression analysis we used the *lrm* function from the *rms* package in R [30].

D. Methods for IaC Architectural Reconstruction

From an abstract point of view, an IaC system is composed of nodes and connectors with a set of nodes types and a set of connector types. Our paper has the goal to automate metrics calculation and assessment based on the node model of an IaC system. That is, if the system is manually modeled or the model can be derived automatically from the IaC scripts, our approach is applicable. For modeling IaC deployment models we followed the method reported in our previous work [31]. All the code and models used in and produced as part of this study have been made available online for reproducibility[2].

In the context of this approach, and based on our work [32], we have introduced a number of detectors in order to automatically reconstruct the IaC architecture from the source code. Combining the automatic reconstruction with the automatic metrics calculation, the overall assessment process can be improved. So far we have done this only for one system and for one technology (S4 in Table I) fully and are developing extensions for the other systems at the moment. A detector is a piece of code that looks for specific characteristics in the IaC deployment model and produces architectural information. The idea is that by having many of these, all lightweight and possibly looking at different languages (e.g. Ansible, Terraform), then deployment architectures can be extracted in an automatic way. Figure 3 shows the resulting model after applying our detectors. Moreover, from the grounded theory analysis, we defined deployment meta-models, in which nodes are extended by the stereotype nodes types and connectors by the stereotype connector types, to reconstruct model instances for the deployments.

[2]https://doi.org/10.5281/zenodo.6559385

E. The Tool Flow of the Approach

Figure 2 illustrates the general tool flow architecture on how the building blocks interact. Using the detectors it's possible to generate models from our modeling tool Codeable Models. As Figure 2 shows, the user can be either a developer or an architect. The architect specifies the architecture abstraction specification for an IaC system, while the developer implements it in the IaC scripts potentially of different technologies. However, both can be involved in the development of detectors. The user can use the architecture specification, IaC scripts and the detector toolchain to generate the IaC deployment model. The architecture specification and the IaC scripts are the inputs of the detector component. The detector performs the detections, and, if successful, uses a code generator to generate the Codeable Models code. The generator uses the deployment meta-model to generate the IaC deployment model. It also contains a visualization generator for generating PlantUML diagrams such as the one in Figure 3.

IV. IaC Security-Related ADDs

In this section, we briefly introduce the three security-related decisions along with their decision options (i.e. the relevant patterns and practices) which we studied for this paper. We also discuss the impact on relevant security aspects, which we later on use as an argumentation for our manual ground truth assessment in Section V.

a) **Security Observability***:* An important aspect in deployment architectures is to be able to identify and respond to what is happening within a system, what resources need to be observed, and inspect what is causing a possible issue. Using observability practices to collect, aggregate, and analyze log data and metrics is a key for establishing and maintaining more secure, flexible, and comprehensive systems [13]. Moreover, collecting and analysing information improves the detection of suspicious system behavior or unauthorised system changes on the network and can facilitate the definition of different behavior types, in which an alert should be triggered. A crucial decision in infrastructure observability is *Server Monitoring* which is an essential process of observing the activity on servers, either physical or virtual. A single server can support hundreds or even thousands of requests simultaneously. As

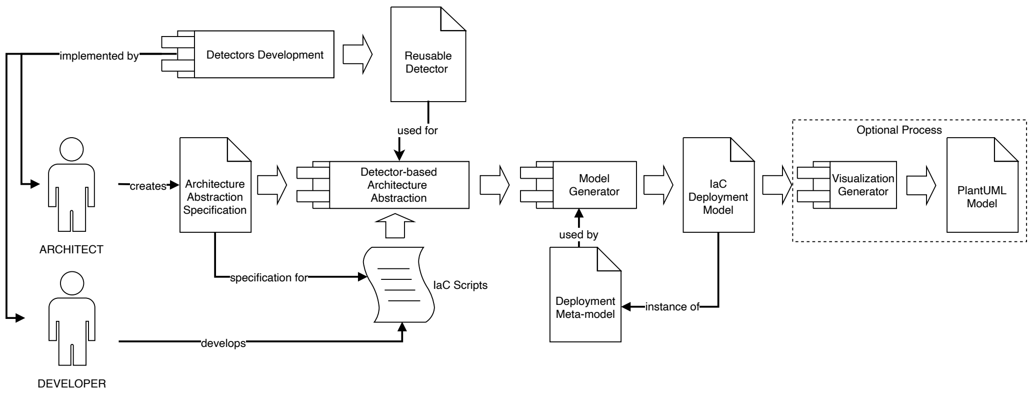

Fig. 2: Tool Flow Architecture of the Proposed System

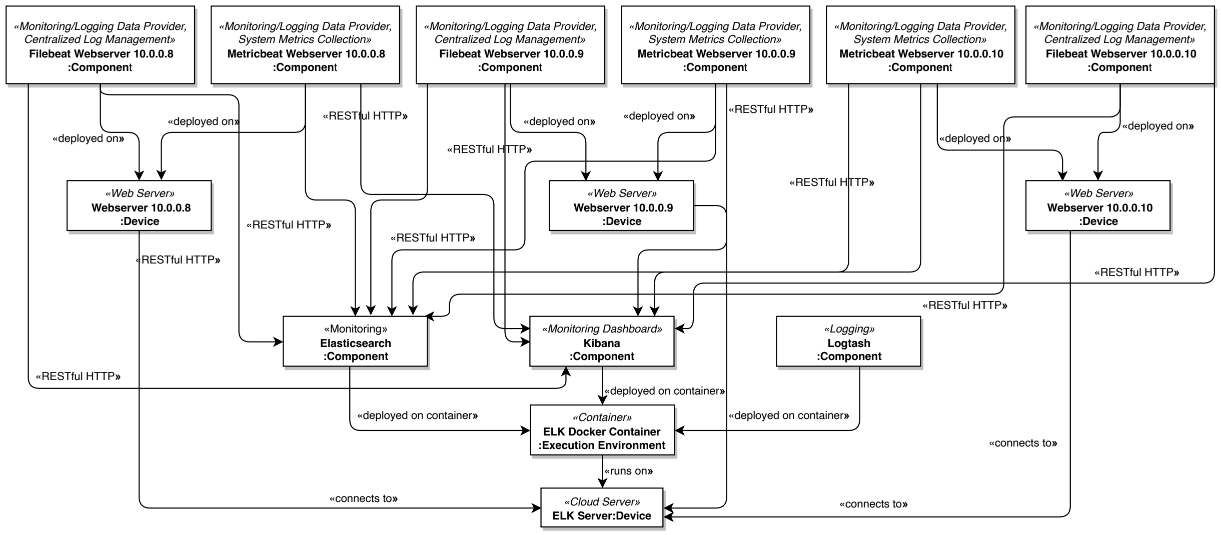
«Monitoring/Logging Data Provider, Centralized Log Management» Filebeat Webserver 10.0.0.8 :Component
«Monitoring/Logging Data Provider, System Metrics Collection» Metricbeat Webserver 10.0.0.8 :Component
«Monitoring/Logging Data Provider, Centralized Log Management» Filebeat Webserver 10.0.0.9 :Component
«Monitoring/Logging Data Provider, System Metrics Collection» Metricbeat Webserver 10.0.0.9 :Component
«Monitoring/Logging Data Provider, System Metrics Collection» Metricbeat Webserver 10.0.0.10 :Component
«Monitoring/Logging Data Provider, Centralized Log Management» Filebeat Webserver 10.0.0.10 :Component
«Web Server» Webserver 10.0.0.8 :Device
«Web Server» Webserver 10.0.0.9 :Device
«Web Server» Webserver 10.0.0.10 :Device
«Monitoring» Elasticsearch :Component
«Monitoring Dashboard» Kibana :Component
«Logging» Logtash :Component
«Container» ELK Docker Container :Execution Environment
«Cloud Server» ELK Server:Device
«deployed on»
«RESTful HTTP»
«deployed on container»
«runs on»
«connects to»

Fig. 3: Overview of a reconstructed model (Model S4 in Table I).

such, ensuring that all of the servers are operating according to expectations is a critical part of managing an infrastructure. Another equally critical decision is *Application Monitoring* which is a process of collecting log data to support aspects such as track availability, bugs, resource use, and changes to performance in an application. Moreover, features such as *Metrics Collection, Services Availability, Centralized Log Management* and *Monitoring and Performance Analytics Support* would additionally improve a system's security.

b) **Security Access Control***:* A critical security factor in cloud-based systems is how stable, verifiable, and secure the interactions between a user and a cloud-application are. For this, a secure authentication practice would address most of the possible issues. Authentication is the process of determining a user's identity. Moreover, authorization practices provide access control for systems by checking if a user's credentials match the credentials of an authorized user or in a data authentication server. Also, it assures secure systems, secure processes and enterprise information security [10]. There are a number of ways authentication can be achieved. At the level of deployment architectures, one strongly recommended practice is the *SSL Protocol-Based Authentication* [33] in which a cryptographic protocol (SSL/TLS) encrypts the data that is exchanged between a web server and a user [11] and provides means for authentication of the connection. An alternative practice is *Token-Based Authentication* [34], a protocol which allows users to verify their identity, and in return receive a unique access token for a specified time period. A similar practice but without granting tokens for a limited time period is *API Keys* based authentication [27], which utilizes a unique identifier to authenticate a user or a calling program to an API. However, they are typically used to authenticate a project with the API rather than a human user. A less secure practice and not recommended for security critical interactions is *Plaintext Authentication* (or Shared Secret Based Authentication), where the user name and password are submitted to the server in plaintext, being easily visible in any intermediate router on the Internet. An authentication practice that can be implemented

Model ID	*Model Size*	*Description / Source*
S1	12 nodes 13 connectors	Consul-based application with specified security groups (from https://github.com/apiacademy/microservices-deployment).
S2	12 nodes 13 connectors	Variant of S1 which uses API keys authentication practice.
S3	8 nodes 9 connectors	Variant of S1 which uses Plaintext authentication practice.
S4	14 nodes 25 connectors	Elasticsearch, Logstash, and Kibana (ELK)-based system using metrics collection and log management tools (from https://github.com/babtunee/azure-cloud-security-architecture).
S5	14 nodes 25 connectors	Variant of S4 using Ingress Traffic Control and partial support of SSL-Based Authentication.
S6	14 nodes 25 connectors	Variant of S4 using Ingress Traffic Control and full support of Token-Based Authentication.
S7	10 nodes 17 connectors	ELK-based system using metrics collection and application monitoring related tools (from https://github.com/deviantony/docker-elk).
S8	10 nodes 17 connectors	Variant of S7 with additional servers and Plaintext Authentication.
S9	17 nodes 48 connectors	Variant of S8 using additional tools for centralized log management, service availability and performance analytics as well as Ingress and Egress traffic control, Token-Based Authentication and Single Sign-On authentication.
S10	8 nodes 11 connectors	ELK-based system using metrics collection related tools (from https://github.com/ManuelMourato25/elastic-stack-architecture-example).
S11	10 nodes 14 connectors	Variant of S10 using Application monitoring, Ingress traffic control and Security groups
S12	16 nodes 30 connectors	ELK-based system using metrics collection and log management tools as well as security groups (from https://github.com/frahmeto/Elk-Stack-Project-1).
S13	19 nodes 57 connectors	Variant of S12 using additional Application monitoring, service availability, performance analytics and API keys authentication.
S14	10 nodes 15 connectors	ELK-TLS-based system using metrics collection, log management, performance analytics and endpoint security tools (from https://github.com/swimlane/elk-tls-docker).
S15	14 nodes 35 connectors	Variant of S14 using additional servers and Application monitoring, performance analytics and SSL-Based Authentication, ingress traffic control and Single Sign-On authentication
S16	8 nodes 11 connectors	Java ELK-based system with service availability and log management tools (from https://github.com/twogg-git/java-elk).
S17	6 nodes 5 connectors	Variant of S16 using only Plaintext authentication and no additional tool to support other features.
S18	14 nodes 21 connectors	Variant of S16 which uses application monitoring, Ingress and Egress traffic control and security groups.
S19	9 nodes 8 connectors	Openshift-based application with application monitoring and metrics collection tools (from https://github.com/redhat-helloworld-msa/k).
S20	9 nodes 8 connectors	Variant of S19 with SSL-based and Single Sign-On authentication.
S21	8 nodes 7 connectors	Terraform application deployed in AWS with metric collection tool and security groups (from https://github.com/ryanmcdermott/terraform-microservices-example).

TABLE I: Selected IaC Deployments Models: Size, Details, and Sources

additionally is the *Single Sign-On (SSO)* [35]. This is a method that can allow users to log into one application and gain access to multiple applications. The goal of SSO is to make it unnecessary for users to have numerous kinds of credentials also benefits them because it allows them to log-off from all system components that use SSO with a single request. In this way, SSO can enable users to improve passwords by getting rid of the need to remember and use them for every single application, offering the best combination of simplicity and security for users.

c) **Security Traffic Control***:* Controlling incoming and outgoing traffic in a system can significantly improve the overall security. Two common practices in this field are *Ingress* and *Egress Traffic Control*. Optimally, a system should fully support both practices. *Egress Traffic Control* [36] refers to traffic that exits a network boundary, while *Ingress Traffic Control* [37] refers to traffic that enters the boundary of a network. The ability to control what is entering a system is of significant importance for security assurance, since it can prevent possible attacks from outside of the network, where many possible attacks originate. Furthermore, it is important to reduce the vulnerability as much as possible and prevent the attackers from using a cluster for further attacks on external services or systems outside of the cluster. This requires securing control of egress traffic. Both practices can be specified by security rules implementing security groups [9] that act as a virtual firewall for a system.

V. Ground Truth Calculations for the Study

In this section, we report for each of the ADDs from Section IV how the ground truth data is calculated based on manual assessment whether each of the relevant practices is either Supported (**S**), Partially Supported (**P**), or Not-Supported (**N**) in Table II. Following the information taken from the description of the impacts of the various decision options in Section IV, we combined the outcome of all decision options to derive an ordinal assessment on how well the decision as a whole is supported in each model, using the ordinal scale in Section III-C. This was done according to best practices documented in literature and experts assessment. For instance, following the ordinal scale the assessment for the model S4 is $\sim$: serious flaws in the security design, but substantial support can already be found in the system, since the practices *ServicesAvailability Bugsand Performance Management* and *ApplicationMonitoring* are not supported. The ordinal results of assessments are reported in the Assessments rows of Table II.

Following the argumentation, for the *Security Observability* related practices, we can derive the following scoring scheme for our ground truth assessment of this decision:

- $++$: All server nodes support *Server Monitoring*, *Application Monitoring*, *Centralized Log Management*, *System Metrics Collection*, *Services Availability* and *Monitoring and Performance Analytics*.
- $+$: All server nodes support *Server Monitoring* and *Application Monitoring* and one or more of the practices *Centralized Log Management*, *System Metrics Collection*, *Services Availability* and *Monitoring and Performance Analytics* is supported.
- $\sim$: The majority of the server nodes support *Server Monitoring* and *Application Monitoring* and one or more of the practices *Centralized Log Management*, *System Metrics Collection*, *Services Availability* and *Monitoring and Performance Analytics* is supported.
- $-$: Some of the server nodes support *Server Monitoring* and/or *Application Monitoring*.
- $--$: None of the server nodes support monitoring.

From the argumentation for the *Security Access Control* decision, we can derive the following scoring scheme for our ground truth assessment:

- $++$: All server nodes support *SSL-Based Authentication* or *Token-Based Authentication* and *Single Sign-On authentication*.
- $+$: All server nodes support *Token-Based Authentication* or *SSL-Based Authentication*.
- $\sim$: All server nodes are authenticated and some of those only support *API Keys-Based Authentication*.
- $-$: All server nodes are authenticated and some of those only support *Plaintext-Based Authentication*.
- $--$: None of the server nodes support authentication.

Finally, from the argumentation for the *Security Traffic Control* decision, we can derive the following scoring scheme for our ground truth assessment:

- $++$: All server nodes support *Ingress Traffic Control* and *Egress Traffic Control*.
- $+$: All server nodes support *Ingress Traffic Control* and *Egress Traffic Control* to the majority of system nodes.
- $\sim$: All server nodes support *Ingress Traffic Control* and *Egress Traffic Control* not to the majority of system nodes.
- $-$: The majority of server nodes support *Ingress Traffic Control*.
- $--$: No traffic control is supported.

VI. Metrics

In this section, we describe the metrics we have hypothesized for each of the decisions described in Section IV. They are deliberately rather simple, as to represent each decision point in our design decisions. The metrics, are a continuous value with range from 0 to 1, with 1 representing the optimal case where a set of patterns is fully supported, and 0 the worst-case scenario where it is completely absent, except *Plaintext Authentication utilization metric* in which the scale is reversed in comparison to the others, because here we detect the presence of an anti-pattern: the optimal result of our metrics is 0, and 1 is the worst-case result.. Using the model computed in the ordinal regression analysis below, we then provide more complex metrics per decision in section IV.

A. Metrics for the Security Observability Decision

Server Monitoring metric (SEM). This metric returns the proportion of *Server Nodes* that support server monitoring.

$$SEM = \frac{\textit{Server Monitoring Support}}{\textit{Number of Server Nodes}}$$

Application Monitoring Support metric (AMS). This metric measures the proportion of servers that support *Application Monitoring*.

$$AMS = \frac{\textit{Number of Application Monitoring Links}}{\textit{Number of Server Nodes}}$$

System Metrics Collection Support metric (SMC). This metric measures the proportion of servers that support *System Metrics Collection* tools.

$$SMC = \frac{\textit{Number of System Metrics Collection Links}}{\textit{Number of Server Nodes}}$$

Centralized Log Management Support metric (CLM). This metric measures the proportion of servers that support *Centralized Log Management* tools.

$$CLM = \frac{\textit{Number of Centralized Log Management Links}}{\textit{Number of Server Nodes}}$$

Service Availability Support metric (SAS). This metric measures the proportion of servers that support *Service Availability* tools.

$$SAS = \frac{\textit{Number of Service Availability Links}}{\textit{Number of Server Nodes}}$$

Monitoring and Performance Analytics Support metric (PAS). This metric measures the proportion of servers that support *Monitoring and Performance Analytics* tools.

$$PAS = \frac{\textit{Number of Monitoring and Performance Analytics Links}}{\textit{Number of Server Nodes}}$$

Security Observability																					
	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**	**S14**	**S15**	**S16**	**S17**	**S18**	**S19**	**S20**	**S21**
Server Monitoring	S	S	P	S	S	S	S	S	S	S	N	S	S	S	P	S	N	S	N	S	S
Application Monitoring	N	N	N	N	S	N	S	P	S	N	N	N	S	N	P	N	N	S	S	S	N
System Metrics Collection	N	N	N	S	P	N	S	P	S	S	N	S	S	S	S	N	N	N	S	N	S
Centralized Log Management	N	N	N	S	S	N	N	N	S	N	N	S	S	S	S	S	N	S	N	N	N
Services Availability	N	N	N	N	N	N	N	N	S	N	N	N	S	N	N	S	N	S	N	N	N
Bugs and Performance Management	N	S	P	N	N	N	N	N	S	N	N	N	S	S	S	N	N	N	N	N	N
Assessments	-	∼	∼	∼	+	∼	+	o	++	o	--	∼	++	∼	-	∼	–	+	-	+	∼
Security Access Control																					
	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**	**S14**	**S15**	**S16**	**S17**	**S18**	**S19**	**S20**	**S21**
SSL Protocol-Based Authentication	N	N	N	N	P	N	N	S	N	N	N	N	N	S	S	N	N	N	N	S	N
Token-Based Authentication	S	N	N	N	N	S	N	N	S	N	N	N	N	N	N	N	N	N	N	N	N
Plaintext Authentication	N	N	S	S	N	N	S	N	N	N	N	N	N	N	N	N	S	N	N	N	S
API Keys	N	S	N	N	N	N	N	N	N	N	N	N	S	N	N	N	N	N	S	N	N
Single Sign-On (SSO)	N	N	N	N	N	N	N	N	S	N	N	N	N	N	S	N	N	N	N	S	N
Assessments	+	∼	-	-	∼	+	-	+	++	--	--	--	∼	+	++	--	-	--	∼	++	-
Security Traffic Control																					
	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**	**S11**	**S12**	**S13**	**S14**	**S15**	**S16**	**S17**	**S18**	**S19**	**S20**	**S21**
Ingress Traffic Control	S	P	N	N	S	S	N	S	S	N	N	S	P	N	P	N	S	S	N	P	S
Egress Traffic Control	S	N	N	N	P	N	N	N	P	N	N	S	P	N	N	N	N	S	N	N	S
Assessments	++	-	--	--	+	∼	--	∼	+	--	--	++	-	--	-	--	∼	++	--	-	++

TABLE II: Ground Truth Data

B. Metrics for Security Access Control Decision

SSL Protocol-based Authentication utilization metric (SSLA). We defined this metric to measure the proportion of servers that support *SSL Protocol-based Authentication*.

$$SSLA = \frac{\textit{SSL Protocol-based Authentication Support}}{\textit{Number of Server Nodes}}$$

Token-Based Authentication utilization metric (TBA). This metric measures the proportion of servers that support *Token-Based Authentication*.

$$TBA = \frac{\textit{Token-Based Authentication Support}}{\textit{Number of Server Nodes}}$$

API Keys utilization metric (API). This metric measures the proportion of servers that support *API Keys*.

$$API = \frac{\textit{API Keys Support}}{\textit{Number of Server Nodes}}$$

Plaintext Authentication utilization metric (PLA). This metric measures the proportion of servers that support *Plaintext Authentication*.

$$PLA = \frac{\textit{Plaintext Authentication Support}}{\textit{Number of Server Nodes}}$$

Single Sign-On Authentication utilization metric (SSO). This metric measures the proportion of servers that support *Single Sign-On Authentication*.

$$SSO = \frac{\textit{Single Sign-On Authentication Support}}{\textit{Number of Server Nodes}}$$

C. Metrics for Security Traffic Control Decision

Ingress Traffic Control utilization metric (ING). We defined this metric to measure the proportion of servers that support *Ingress Traffic Control*.

$$ING = \frac{\textit{Ingress Traffic Control Support}}{\textit{Number of Server Nodes}}$$

Egress Traffic Control utilization metric (EGR). We defined this metric to measure the proportion of servers that support *Egress Traffic Control*.

$$EGR = \frac{\textit{Egress Traffic Control Support}}{\textit{Number of Server Nodes}}$$

VII. Evaluation of our Approach

In this section, we present and discuss the results of the metrics calculations for our models as well as the results of the ordinal regression analysis. The metrics calculations for each model per each decision metric are presented in Table III. The dependent outcome variables are the ground truth assessments for each decision, as described in Section V and summarized in Table II. The metrics defined in Section VI and summarized in Table III are used as the independent predictor variables. The ground truth assessments are ordinal variables, while all the independent variables are measured on a scale from 0.0 to 1.0. The objective of the analysis is to predict the likelihood of the dependent outcome variable for each of the decisions by using the relevant metrics for each decision.

Each resulting regression model consists of a *baseline intercept* and the independent variables multiplied by *coefficients*. There are different intercepts for each of the value transitions of the dependent variable ($\geq$ [-]: *serious flaws in the security design, but initial support can already be found in the system*, $\geq$[$\sim$]: *serious flaws in the security design, but substantial support can already be found in the system*, $\geq$ [+]: *well supported, but aspects of the solution could be improved*, $\geq$ [++]: *very well supported*), while the coefficients reflect the impact of each independent variable on the outcome.

The statistical significance of each regression model is assessed by the p-value; the smaller the p-value, the stronger the model is. A p-value smaller than 0.05 is generally considered statistically significant [38]. In Table IV, we report the p-

TABLE III: Metrics Calculation Results

Metrics	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20	S21
Security Observability																					
SEM	1.00	1.00	0.75	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.00	1.00	1.00	1.00	0.40	1.00	0.00	1.00	0.00	1.00	1.00
APM	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.66	1.00	0.00	0.00	0.00	1.00	0.00	0.40	0.00	0.00	1.00	1.00	1.00	0.00
SMC	0.00	0.00	0.00	1.00	0.66	1.00	1.00	0.33	1.00	1.00	0.00	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	1.00	1.00
CLM	0.00	0.00	0.00	1.00	1.00	0.66	0.00	0.00	1.00	0.00	0.00	1.00	1.00	1.00	1.00	1.00	0.00	1.00	0.00	0.00	0.00
SAS	0.00	0.00	0.00	0.00	0.00	0.33	0.00	0.00	1.00	0.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	1.00	0.00	0.00	0.00
PAS	0.75	1.00	0.75	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00
Security Access Control																					
SSLA	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00	0.00
TBA	1.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
API	0.00	1.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00
PLA	0.00	0.00	1.00	1.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00	0.00	1.00
SSO	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	0.00	0.00	1.00	0.00
Security Traffic Control																					
ING	1.00	0.75	0.00	0.00	1.00	1.00	0.00	1.00	1.00	0.00	0.00	1.00	0.33	0.00	0.80	0.00	1.00	1.00	0.00	0.50	1.00
EGR	1.00	0.00	0.00	0.00	0.66	0.00	0.00	0.00	0.66	0.00	0.00	1.00	0.33	0.00	0.00	0.00	0.00	1.00	0.00	0.00	1.00

values for the resulting models, which in all cases are very low, indicating that the sets of metrics we have defined are able to predict the ground truth assessment for each decision with a high level of accuracy.

Often, the C-index, which is also called concordance index and is equivalent to the area under the Receiver Operating Characteristic (ROC) curve, is reported in the statistical literature as a measure of the predictive power of ordinal regression models [39]. The C-index is a metric to evaluate the predictions made by an algorithm. Values over 0.7 indicate a good model, whereas values over 0.8 indicate a strong model. A value of 1 means that the model perfectly predicts those group members who will experience a certain outcome and those who will not. As Table IV shows, the values of C-indexes are generally larger than 0.8 indicating a good enough model for predicting the outcomes of individuals. Harrell [30] suggests bootstrapping as a method for obtaining nearly unbiased estimates of a model's future performance based on re-sampling. We used lrm's *validate* function to perform bootstrapping and calculate the bias-corrected C-index.

TABLE IV: Regression Analysis Results

Intercepts/Coefficients	*Value*
Security Observability	
Intercept: $\geq$ [-]	*-1.0154*
Intercept: $\geq$ [$\sim$]	*-4.0071*
Intercept: $\geq$ [+]	*-7.9753*
Intercept: $\geq$ [++]	*-10.956*
Metric Coefficient (SEM)	*4.4224*
Metric Coefficient (APM)	*2.7670*
Metric Coefficient (SMC)	*1.7050*
Metric Coefficient (CLM)	*0.2630*
Metric Coefficient (SAS)	*2.1536*
Metric Coefficient (PAS)	*0.3730*
Model p-value	*1.153036e-06*
Model C-index (original)	*0.929*
Model C-index (bias-corrected)	*0.8934091*
Security Access Control	
Intercept: $\geq$ [-]	*0.1688*
Intercept: $\geq$ [$\sim$]	*-2.0267*
Intercept: $\geq$ [+]	*-3.6306*
Intercept: $\geq$ [++]	*-6.6639*
Metric Coefficient (SSLA)	*4.3872*
Metric Coefficient (TBA)	*4.4533*
Metric Coefficient (API)	*1.2695*
Metric Coefficient (PLA)	*2.5834*
Metric Coefficient (SSO)	*3.8701*
Model p-value	*3.871694e-07*
Model C-index (original)	*0.951*
Model C-index (bias-corrected)	*0.9456178*
Security Traffic Control	
Intercept: $\geq$ [-]	*-29.792*
Intercept: $\geq$ [$\sim$]	*-69.340*
Intercept: $\geq$ [+]	*-90.643*
Intercept: $\geq$ [++]	*-120.88*
Metric Coefficient (ING)	*77.7199*
Metric Coefficient (EGR)	*52.539*
Model p-value	*2.031708e-14*
Model C-index (original)	*0.904*
Model C-index (bias-corrected)	*0.8827108*

VIII. Discussion

A. Discussion of Research Questions

To answer **RQ1** and **RQ2**, we proposed a set of generic, technology-independent metrics for each IaC decision, and to each decision option corresponds at least one metric. We established the ground truth to objectively assessed how well patterns and/or practices are supported in each model, and extrapolated this to how well the broader decision is supported. We defined a set of generic, technology independent metrics to automatically and numerically assess each pattern's implementation in each model, and performed an ordinal regression analysis using these metrics as independent variables to predict the ground truth assessment. Our results show that every set of decision-related metrics can predict with high accuracy our objectively evaluated assessment. This suggests that an automatic metrics-based assessment of a system's conformance to ADD's options is possible with a high degree of confidence.

Here, we make the assumption that the infrastructure code can be mapped to the models used in our work. For enabling this, we used rather simplistic modeling means, which can easily be mapped from a specific code to the models

Regarding **RQ3**, we can assess that our deployment metamodel has no need for major extensions and is easy to map to existing modeling practices. More specifically, in order to fully model our evaluation model set, we needed to introduce 13 device type nodes and 11 execution environment nodes types such as *Cloud Server* and *Virtual Machine* respectively, and 6 deployment relation types and 4 deployment node relations. Furthermore, we also introduced a deployment node metamodel to cover all the additional nodes of our decisions, such as *Centralized Log Management*, which is a subclass of node type. The decisions in *Security Observability* require modeling several elements such as the *Web Server*, *Container*, and *Cloud Server* nodes types and technology-related connector types (e.g. *RESTful HTTP*) as well as deployment-related connector types (e.g. *Runs on, Deployed in Container*). For the *Security Access Control* and *Security Traffic Control* decisions, we have introduced additional attributes in the system nodes (e.g. in *Web Server*) to specify whether an authentication practice is supported or not, and which specific practice is used (e.g. *SSL Protocol-Based Authentication*). Based on these considerations, we can assess that our decisions and the corresponding model elements can easily be extracted automatically, e.g. with an approach as described in Section III-E.

B. Threats to Validity

We mostly relied on third-party systems as the basis for our study in order to increase the internal validity and thus avoid distorting the system composition and structure. It is possible that our search procedures have resulted in some kind of unconscious exclusion of certain sources; we have mitigated this by putting together a team of authors with years of experience in this field and doing a very general and broad search. As our search was not exhaustive and most of the systems we found were created for demonstration purposes and they were relatively small in size, meaning that some potential architectural elements were not included in our metamodel. Moreover, this contains a potential threat to the external validity of generalization to other, more complex systems. However, we are confident that the documented systems represent a representative cross-section of current practices in this area. Another potential risk is the fact that the system variants were developed by the team of authors themselves. However, this was done in accordance with the best practices documented in the literature. We made sure to only change certain aspects in one variant and to keep all other aspects stable.

Another possible source of compromising internal validity is the modeling process. The team of authors has considerable experience with similar methodologies, and the models of the systems have been repeatedly and independently cross-checked, but the possibility of some interpretive bias remains: other researchers might have coded or modeled differently, resulting in different models. Since our aim was only to find a model that could specify all observed phenomena, and this was achieved, we do not consider this risk to be particularly problematic for our study. The assessment of the basic truth could also be interpreted differently by different practitioners. The individual metrics that are used to evaluate the presence of the individual patterns were deliberately kept as simple as possible in order to avoid false positive results and to enable an evaluation that is independent of technology.

IX. Conclusions and Future Work

In this work we have studied how far it is possible to develop a method to automatically assess security practices in architectural design decisions in the scope of an IaC deployment model. We have shown that this is possible for IaC security-related decisions that contain patterns and practices as decision options. In our approach, we modeled the key aspects of the decision options using a minimal set of deployment model elements, which means that it is possible to automatically extract them from the IaC scripts. Then we defined a number of metrics to cover all the possible decision options and utilized the generated models as a ground truth. We then used statistical methods (ordinal regression analysis) to derive a prediction model. The results show that our metrics set can predict the ground truth assessment with a high level of accuracy.

So far, for security aspects of IaC deployment models, no generic, technology-independent metrics have been studied in depth. According to the discussion in Section II, there are studies which are focused on quality assurance of IaC systems, but none specifically address security critical measures. Furthermore, our approach treats deployment architectures as set of nodes and connectors factoring out the technologies used, which is not the case for other studies. Our approach's goal is a continuous assessment, considering the impact of continuous delivery practices, in which the metrics are assessed in a continuous manner, indicating improvements, stability, and security of deployment architecture conformance.

As future work, we plan to study more decisions and related metrics, and to create a larger data set, thus better supporting tasks such as early architecture assessment in a project.

X. Acknowledgments

This work was supported by: FWF (Austrian Science Fund) project IAC^2: I 4731-N.; Our work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 952647 (AssureMOSS project).

References

[1] M. Nygard, *Release It! Design and Deploy Production-Ready Software*. Pragmatic Bookshelf, 2007.

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

[3] K. Morris, *Infrastructure as Code: Dynamic Systems for the Cloud*. O'Reilly, 2015.

[4] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D. A. Tamburri, "Devops: Introducing infrastructure-as-code," in *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, pp. 497–498.

[5] S. Newman, *Building Microservices: Designing Fine-Grained Systems*. O'Reilly, 2015.

[6] I. Kumara, M. Garriga, A. U. Romeu, D. Di Nucci, F. Palomba, D. A. Tamburri, and W.-J. van den Heuvel, "The do's and don'ts of infrastructure code: A systematic gray literature review," *Information and Software Technology*, vol. 137, p. 106593, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584921000720

[7] T. Sharma, M. Fragkoulis, and D. Spinellis, "Does your configuration code smell?" in *Proceedings of the 13th International Conference on Mining Software Repositories*, ser. MSR '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 189–200. [Online]. Available: https://doi.org/10.1145/2901739.2901761

[8] J. Schwarz, A. Steffens, and H. Lichter, "Code smells in infrastructure as code," in *2018 11th International Conference on the Quality of Information and Communications Technology (QUATIC)*, 2018, pp. 220–228.

[9] AWS Documentation, "Security groups for your vpc," https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html, 2021.

[10] OWASP Cheat Sheet Series, "Authentication cheat sheet," https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html#logging-and-monitoring, 2021.

[11] ——, "Transport layer protection cheat sheett," https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html#ssl-vs-tls, 2021.

[12] ——, "Infrastructure as code security cheatsheet," https://cheatsheetseries.owasp.org/cheatsheets/Infrastructure_as_Code_Security_Cheat_Sheet.html, 2021.

[13] Cloud Security Alliance, "Continuous monitoring in the cloud," https://cloudsecurityalliance.org/blog/2018/06/11/continuous-monitoring-in-the-cloud/, 2018.

[14] ——, "Five approaches for securing identity in cloud infrastructure," https://cloudsecurityalliance.org/blog/2021/05/20/five-approaches-for-securing-identity-in-cloud-infrastructure/, 2021.

[15] 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, 2020.

[16] S. Dalla Palma, D. Di Nucci, and D. A. Tamburri, "Ansiblemetrics: A python library for measuring infrastructure-as-code blueprints in ansible," *SoftwareX*, vol. 12, p. 100633, 2020.

[17] M. Wurster, U. Breitenbücher, L. Harzenetter, F. Leymann, and J. Soldani, "Tosca lightning: An integrated toolchain for transforming tosca light into production-ready deployment technologies," in *Advanced Information Systems Engineering*, N. Herbaut and M. La Rosa, Eds. Cham: Springer International Publishing, 2020, pp. 138–146.

[18] I. Kumara, Z. Vasileiou, G. Meditskos, D. A. Tamburri, W.-J. Van Den Heuvel, A. Karakostas, S. Vrochidis, and I. Kompatsiaris, "Towards semantic detection of smells in cloud infrastructure code," in *Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics*, ser. WIMS 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 63–67. [Online]. Available: https://doi.org/10.1145/3405962.3405979

[19] T. Sotiropoulos, D. Mitropoulos, and D. Spinellis, "Practical fault detection in puppet programs," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 26–37. [Online]. Available: https://doi.org/10.1145/3377811.3380384

[20] E. van der Bent, J. Hage, J. Visser, and G. Gousios, "How good is your puppet? an empirically defined and validated quality model for puppet," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 164–174.

[21] M. Pendleton, R. Garcia-Lebron, J.-H. Cho, and S. Xu, "A survey on systems security metrics," *ACM Comput. Surv.*, vol. 49, no. 4, Dec. 2016. [Online]. Available: https://doi.org/10.1145/3005714

[22] M. P. Fischer, U. Breitenbücher, K. Képes, and F. Leymann, "Towards an approach for automatically checking compliance rules in deployment models," in *Proceedings of The Eleventh International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*. Xpert Publishing Services (XPS), 2017, pp. 150–153.

[23] 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 (SummerSoC 2018)*. IBM Research Division, Oktober 2018, Konferenz-Beitrag, pp. 76–89.

[24] G. Y. Guo, J. M. Atlee, and R. Kazman, "A software architecture reconstruction method," in *Software Architecture*. Springer, 1999, pp. 15–33.

[25] A. Van Deursen, C. Hofmeister, R. Koschke, L. Moonen, and C. Riva, "Symphony: View-driven software architecture reconstruction," in *4th Working IEEE/IFIP Conf. on Software Architecturen(WICSA 2004)*. IEEE, 2004, pp. 122–132.

[26] T. Haitzer and U. Zdun, "Semi-automated architectural abstraction specifications for supporting software evolution," *Science of Computer Programming*, vol. 90, pp. 135–160, 2014.

[27] Google Cloud, "Using api keys," https://cloud.google.com/docs/authentication/api-keys, 2021.

[28] J. Corbin and A. L. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative Sociology*, vol. 13, pp. 3–20, 1990.

[29] W. M. Piasecki, J. and V. Dranseika, "Google search as an additional source in systematic reviews," *Sci Eng Ethics 24*, vol. 55, no. 4, p. 809–810, 2018.

[30] J. Frank E. Harrell, *Regression Modeling Strategies: With Applications to Linear Models, Logistic and Ordinal Regression, and Survival Analysis*, 2nd ed. Springer, 2015.

[31] U. Zdun, E. Navarro, and F. Leymann, "Ensuring and assessing architecture conformance to microservice decomposition patterns," in *Service-Oriented Computing*, M. Maximilien, A. Vallecillo, J. Wang, and M. Oriol, Eds. Cham: Springer International Publishing, 2017, pp. 411–429.

[32] E. Ntentos, U. Zdun, K. Plakidas, P. Genfer, S. Geiger, S. Meixner, and W. Hasselbring, "Detector-based component model abstraction for microservice-based systems," *Computing*, vol. 103, pp. 2521–2551, August 2021. [Online]. Available: http://eprints.cs.univie.ac.at/6926/

[33] A. K. Ranjan, V. Kumar, and M. Hussain, "Security analysis of tls authentication," in *2014 International Conference on Contemporary Computing and Informatics (IC3I)*, 2014, pp. 1356–1360.

[34] Okta, "Token-based authentication," https://www.okta.com/identity-101/what-is-token-based-authentication/, 2021.

[35] auth0Docs, "Single sign-on (sso)," https://auth0.com/docs/authenticate/single-sign-on, 2021.

[36] The Security Skeptic, "Firewall best practices - egress traffic filtering," https://securityskeptic.typepad.com/the-security-skeptic/firewall-best-practices-egress-traffic-filtering.html, 2021.

[37] Kubernetes Documentation, "Ingress traffic control," https://kubernetes.io/docs/concepts/services-networking/ingress/, 2021.

[38] C. D. Michael Cowles, "On the origins of the .05 level of statistical significance," in *American Psychologist, 37(5), 553–558.*

[39] A. Airola, T. Pahikkala, W. Waegeman, B. De Baets, and T. Salakoski, "An experimental comparison of cross-validation techniques for estimating the area under the roc curve," *Computational Statistics & Data Analysis*, vol. 55, no. 4, pp. 1828–1844, 2011.