Research Article GU J Sci, Part A, 10(4): 452-471 (2023) 10.54287/gujsa.1373305

Gazi University
Journal of Science
PART A: ENGINEERING AND INNOVATION
http://dergipark.org.tr/gujsa

Systematic Analysis of Infrastructure as Code Technologies

Erdal ÖZDOĞAN[1*] Onur CERAN[1] Mutlu Tahsin ÜSTÜNDAĞ[2]

[1] IT Department, Gazi University, Ankara, Türkiye
[2] Distance Education Application and Research Center, Gazi University, Ankara, Türkiye

Keywords

Infrastructure as Code
Network Automation
IaC Tools
Configuration Management

Abstract

"Infrastructure as Code" technologies are the network automation concept used in configuring network devices, allocating network resources, and deploying developed applications. By using machine-readable codes, various tasks that previously required time and effort can now be done dynamically with infrastructure as code tools. Although Infrastructure as Code is a technology that brings many advantages and is still at the beginning of its popularity, there are not enough resource in the literature. In this study, the key concepts of Infrastructure as Code technologies are discussed and infrastructure as code tools are systematically examined. The six most used Infrastructure as Code tools were examined in terms of management, language, data representation, code approach, stateful and stateless, architectural perspectives. Also, they were compared over these key concepts. The main purpose of this article is to define, classify, and elucidate the emerging infrastructure as code tools.

Cite

Ozdogan, E., Ceran, O., & Ustundag, M.T (2023). Systematic Analysis of Infrastructure as Code Technologies. *GU J Sci, Part A, 10*(4), 452-471. doi:10.54287/gujsa.1373305

Author ID (ORCID Number)		Article Process	
0000-0002-3339-0493	Erdal OZDOGAN	**Submission Date**	09.10.2023
0000-0003-2147-0506	Onur CERAN	**Revision Date**	30.10.2023
0000-0001-6198-2819	Mutlu Tahsin USTUNDAG	**Accepted Date**	14.11.2023
		Published Date	12.12.2023

1. INTRODUCTION

Today's Information Technologies (IT) have fundamental needs for both consumers of IT services and the software development sector that provides these services, such as speed, consistency, and security. With the support of virtualization and cloud computing technologies, IT resources have begun to be utilized more efficiently, and the needs of service providers and users have been relatively met. Increasing consumer demands and new technological trends have necessitated cloud-based resource provisioning to be much faster and error-resistant. This requirement has led to the use of automation in the sharing and provisioning of infrastructure resources such as storage, processors, memory, and networking, giving rise to the concept of Infrastructure as Code (IaC). On the other hand, the growing popularity of cloud technologies has particularly led to the development of new types of applications targeting cloud environments or efficient operation of cloud-based applications or network applications, the automation of various infrastructures and the dynamization of the software development process are required (Tankov et al., 2021). The use of Infrastructure as Code, which is a software engineering tactic that reduces the technical and organizational distance between software development and infrastructure provisioning processes, has become quite widespread in the IT sector (Artac et al., 2017).

IaC is a set of applications that uses "code" instead of manually entered commands to set up virtual machines, networks, and software packages, and to configure the environments that applications require (Patni et al., 2020). In IaC technologies, tasks such as provisioning infrastructure and installing software are managed through automation using code. In cloud service providers, meeting users' varying demands at different times using traditional methods can be time-consuming and error prone. However, thanks to IaC, efficient

*Corresponding Author, e-mail: erdal.ozdogan@gazi.edu.tr distribution and sharing of infrastructure resources can be achieved in much shorter periods. Tasks such as resource sharing, device configuration, preparation of application development and testing environments, application deployment, and resource management, as depicted in Figure 1, can be accomplished using various tools within the IaC framework.

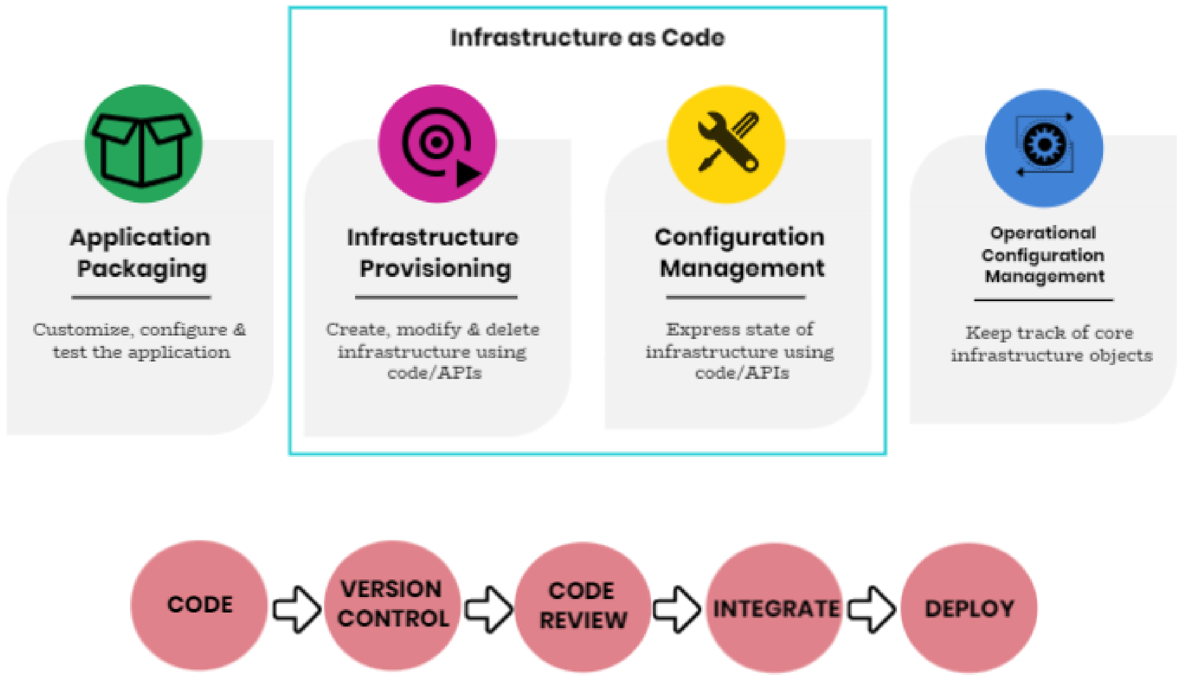
Infrastructure as Code
Application Packaging
Customize, configure & test the application
Infrastructure Provisioning
Create, modify & delete infrastructure using code/APIs
Configuration Management
Express state of infrastructure using code/APIs
Operational Configuration Management
Keep track of core infrastructure objects
CODE
VERSION CONTROL
CODE REVIEW
INTEGRATE
DEPLOY

Figure 1. *Structure of Infrastructure as Code*

Infrastructure as Code, a technology that offers many advantages such as rapid resource sharing, efficient scalability, and low probability of errors, is still at the early stages of its popularity. Many organizations providing cloud-based services in the IT market are utilizing IaC technologies. However, there is still limited understanding of how the code underlying IaC applications can be maintained and developed in a measurable manner (Artac et al., 2017; Guerriero et al., 2019; Dalla Palma et al., 2020). Also there are challenges encountered in its implementation (Chen et al., 2018; Sandobalin et al., 2019). While there have been various research efforts in the literature regarding the widespread adoption of IaC technologies and tools in the industry, there still exists a shortage of resources on the subject (Kumara et al., 2021; Falazi et al., 2022; Alonso et al., 2023).

Infrastructure as Code has become a frequently employed tool by software developers with the aim of facilitating the rapid delivery of DevOps applications and services to end-users. In order to systematically assess studies related to IaC, research domains for IaC were identified in a mapping study conducted by Rahman et al. (2019).

Due to the early stage of research in the field of IaC, there is a limited amount of academic literature available on the subject. Artac et al. (2017) have discussed key elements and abstractions in the Topology and Orchestration Specification for Cloud Applications (TOSCA) standards, pertaining to IaC. The study summarizes the specialized TOSCA standard for IaC as a tactic to expedite Development and Operations (DevOps) based lifecycles, aiming to accelerate the cycle of development and operations activities. In the study by Sandobalin et al. (2017), a tool is presented that supports the management of IaC based DevOps tools. This tool enables modeling the current state of infrastructure provisioning in the cloud and facilitates the creation of scripts. In another work by the same authors, a model-based approach is proposed for infrastructure provisioning (Sandobalin et al., 2019).

Dalla Palma et al. (2020) have pointed out that software quality metrics developed for general-purpose programming languages may not be suitable for IaC. In their relevant study, they focused on a specific IaC tool and developed a new catalog comprising 46 metrics.

In the study by Opdebeeck et al. (2020) the role transition in Ansible, which is one of the Infrastructure as Code tools, was analyzed. The authors designed a structural model for Ansible roles and developed a unique algorithm to extract structural changes between two versions of a role. In a study focused on comparing Infrastructure as Code tools, an analysis was conducted regarding features that do not lead to runtime errors in the code but require improvement (known as code smells). The study compared two IaC tools based on these aspects (Schwarz et al., 2018). The authors categorized deficiencies and features that need improvement in the code as technology-dependent and technology-independent in their study. In another study related to "code smells" in Infrastructure as Code, three IaC tools were examined from a security perspective (Rahman & Williams, 2021).

In Infrastructure as Code applications, very little is still known about sustainability and usability. As a result, some literature studies have directly focused on this issue. Indeed, in a study conducted by Guerriero et al., (2019), semi-structured interviews were conducted with senior developers from various companies. This study highlighted the state of implementation and the fundamental challenges in software engineering in relation to IaC adoption, exploring the reasons behind its adoption or non-adoption. The study clearly emphasizes the need for further research in this field. Furthermore, in the same study, it is noted that the support provided by existing tools is still limited.

In the study by Shvetcova et al. (2019) a method is proposed for the unified description and deployment of infrastructures, including hardware and software requirements. In the study, the authors describe an improved Infrastructure as Code tool's (Ansible) module that deploys the necessary infrastructure in a cloud environment based on specific descriptions. In the work by Tankov et al. (2021), a novel approach is presented that significantly simplifies the development of local applications in the cloud, enabling developers to create infrastructure without needing expert-level knowledge about a specific cloud platform. In the study, a direct code-based approach was developed for provisioning infrastructure, eliminating the need to create manifest files for infrastructure allocation. In a study that can be considered an advancement in Infrastructure as Code application, the author aims to enhance the quality of IaC scripts by identifying the characteristics of the development process associated with scripts and errors that may compromise security and privacy (Rahman, 2018).

Infrastructure as Code is extensively used in modern times for critical software's code reviews, testing environments, and development processes. Consequently, the techniques employed while writing code can also be applied when defining infrastructure (Heap, 2016). The experimental study aimed to enhance the quality of Infrastructure as Code through practitioner assistance. It aimed to assist practitioners in identifying the source code characteristics of erroneous IaC scripts (Rahman et al., 2019). In another study focused on identifying code errors, Dalla Palma et al. (2020) developed an innovative method for error prediction using three different techniques. In the study by Chen et al. (2018), a machine learning-based approach is proposed to address frequently occurring errors in Infrastructure as Code.

Considering the studies conducted in the field of Infrastructure as Code, it is evident that development processes are still ongoing and there is a need for systematic analyses of IaC and IaC tool usage. In our study, we conduct a static analysis by identifying key concepts used to define Infrastructure as Code tools in both the literature and grey literature. We explain the features of these tools and compare them. Additionally, within the framework of these key concepts, we conduct a static examination of these tools, revealing which tools are used for specific purposes. This study explores the concepts of IaC technology to provide insights and facilitate future research endeavors. The tools utilized in IaC technology are discussed, and a systematic approach is employed to analyze key concepts.

The study distinguishes itself by incorporating insights from the latest articles and gray literature sources, ensuring that the findings are not only grounded in established knowledge but also reflect the most current trends and developments in the field of IaC technologies. By capturing cutting-edge practices and emerging challenges, the research offers a forward-looking perspective, contributing to a more nuanced understanding of the evolving landscape of IaC tools. Going beyond traditional literature reviews, this study constructs a recommendation tree for the strategic usage of IaC tools. This structured framework provides practical guidance to decision-makers and practitioners, aiding them in navigating the complexities of IaC implementation. By outlining tailored strategies based on specific project requirements, organizational goals, and integration needs, the research offers actionable insights that can be directly applied in real-world scenarios.

The rest of the article is organized as follows:

In the methodology section, the IaC tools are examined through key concepts, aligning with the primary objective of this study. In the third section, an evaluation was conducted based on key concepts, and a recommendation tree for the strategic usage of IaC tools was generated. The results and discussion section presents the findings acquired and offers recommendations believed to be beneficial for prospective research in the realm of IaC technologies. In the final section, the contributions of the study have been presented, and the article concludes with future potential research directions.

2. METHODOLOGY

This section includes the fundamental concepts that will be used for evaluating Infrastructure as Code tools, along with explanations of these concepts. Understanding the meanings of these key concepts accurately is essential for comprehending the efficiency of the tools. The correct interpretation of these key concepts enables us to grasp the effectiveness of the tools.

The method we followed in the study is shown in Figure 2.

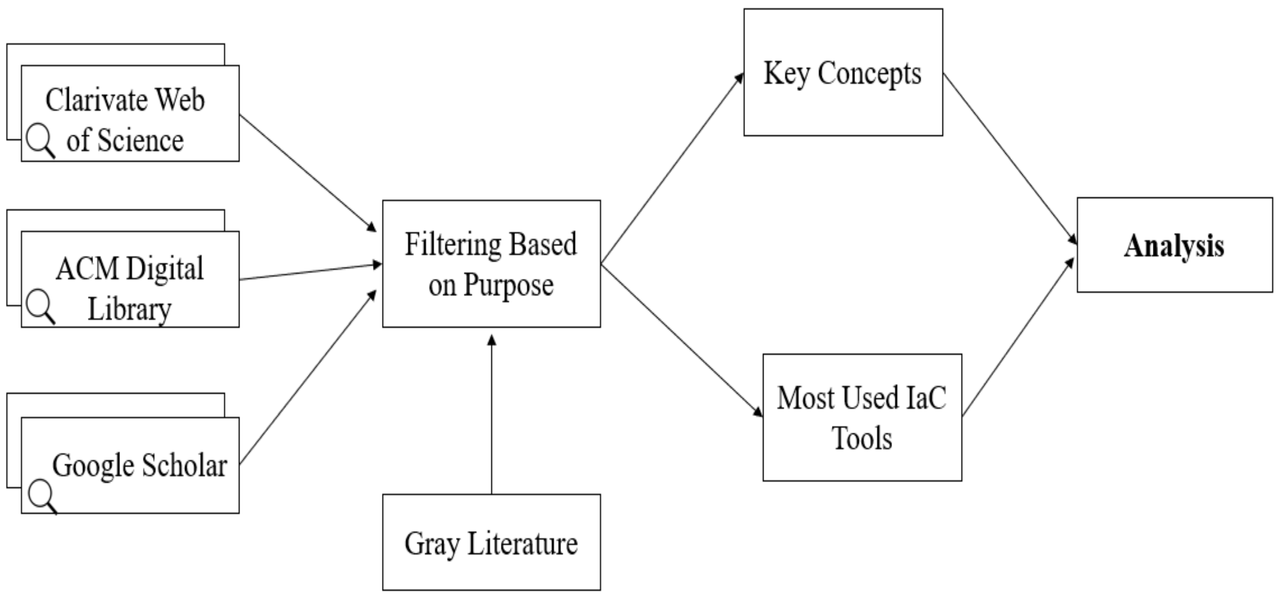

Figure 2*. The workflow we followed in this study*

Firstly, to identify the most used Infrastructure as Code tools, we conducted searches in ACM Digital Library, Clarivate Web of Science and Google Scholar for studies published in the last two years using the keywords "*Infrastructure as Code* $\vee$ *IaC* $\vee$ *IaC Management* v *IaC Tools*". The studies we have examined are presented in Table 1. Subsequently, we identified the IaC tools discussed in the studies and extracted the key concepts used in their definitions.

***Table 1**. Studies published within the last two years within the scope of the article*

No	The Examined Studies
1	Automated Application Deployment on Multi-Access Edge Computing: A Survey (Santos et al., 2023)
2	Static Analysis of Infrastructure as Code: a Survey (Chiari et al., 2022)
3	Practitioner Perceptions of Ansible Test Smells (Zhang et al., 2023)
4	Infrastructure as Code for Dynamic Deployments (Sokolowski, 2022)
5	Comparison of infrastructure as code frameworks from a developer perspective (Karlsson, 2023)
6	Infrastructure-as-Code Ecosystems (Opdebeeck et al., 2023a)
7	The SODALITE Model-Driven Approach (Gorroñogoitia et al., 2022)
8	DevOps and IaC to Automate the Delivery of Hands-On Software Lab Exams (Sorour & Hamdy, 2022)
9	Embracing IaC Through the DevSecOps Philosophy (Alonso et al., 2023)
10	IEM: A Unified Lifecycle Orchestrator for Multilingual IaC Deployments (Diaz-De-Arcaya et al., 2023)
11	Extensible Testing for Infrastructure as Code (Spielmann et al., 2023)
12	Control and Data Flow in Security Smell Detection for Infrastructure as Code: Is It Worth the Effort? (Opdebeeck et al., 2023b)
13	Cybercompetitions: A survey of competitions, tools, and systems to support cybersecurity education (Balon & Baggili, 2023)
14	Towards Reliable Infrastructure as Code (Sokolowski & Salvaneschi, 2023)
15	Decentralizing Infrastructure as Code (Sokolowski et al., 2023)
16	Formal Verification of Infrastructure as Code (De Pascalis, 2022)
17	A Structured Literature Review Approach to Define Serverless Computing and Function as a Service (Manner, 2023)
18	“Through the looking-glass." An Empirical Study on Blob Infrastructure Blueprints in TOSCA (Dalla Palma et al., 2023)
19	Building an IT Security Laboratory for Complex Teaching Scenarios Using ’Infrastructure as Code’ (Soll et al., 2023)
20	Provisioning Secure Cloud Environment Using Policy-as-code and Infrastructure-as-code (Tripathi, 2023)
21	DevSecOps: A Security Model for Infrastructure as Code Over the Cloud (Ibrahim et al., 2022)
22	A comparison between Terraform and Ansible on their impact upon the lifecycle and security management for modifiable cloud infrastructures in OpenStack (Gurbatov, 2022)
23	Ansible in different cloud environments (Witt & Westling, 2023)
24	Ansible: A Reliable Tool for Automation (Daffalla Elradi, 2023)
25	DevOps and Tools Used: A Systematic Review (Raj et al., 2022)
26	Adoption of Infrastructure as Code (IaC) in Real World (Murphy, 2022)

Within the scope of this study, the works that define, examine, or utilize IaC tools are listed in Table 2. The table illustrates which tools are addressed in each study and the percentage weight of their presence across all studies.

According to Table.1, the most frequently used tools are Ansible, Puppet, Chef, Terraform, CloudFormation, and SaltStack. For these tools mentioned in the literature, we determined the most widely used IaC tools for cloud configuration purposes in 2023, based on Statista (2023). Accordingly, CloudFormation has a usage rate of 51%, Terraform: 30%, Ansible: 20%, Chef: 14%, Puppet: 15%, and SaltStack: 10%. The usage statistics of IaC tools covered in this article, which have been the subject of academic studies in the last two years, are also provided in Table 2.

***Table 2**. IaC tools are addressed in each study and the percentage weight of their presence*

IaC Tool Name	References	Weight %
Ansible	[1],[2],[3],[4],[6],[7],[8],[9],[10],[12],[13],[14],[16],[18],[19],[21],[22],[23],[24],[25],[26]	81%
Chef	[1],[2],[4],[7],[9],[10],[12],[13],[14],[16],[18],[19],[21],[22],[23],[24],[25],[26]	69%
CloudFormation	[2],[4],[5],[9],[15],[16],[18],[19],[22],[26]	38%
Puppet	[1],[2],[4],[7],[10],[12],[13],[14],[21],[22],[23],[24],[26]	50%
SaltStack	[10],[13],[16],[21],[24]	19%
Terraform	[1],[2],[4],[8],[9],[10],[11],[13],[14],[15],[16],[19],[20],[21],[22],[23],[24],[26]	69%

We examined these tools statically. We identified the keywords that authors used in defining IaC, explaining IaC tools, or conducting comparisons. Additionally, we browsed the websites of the most frequently used tools and conducted a gray literature review. This allowed us to identify not only the key concepts from previous studies but also new concepts (Rahman et al., 2020). These key concepts are provided in Table 3.

***Table 3**. Key concepts used in definitions*

Key Concept	References
Approach	[1],[2],[4],[5],[13],[16],[21],[22],[23],[26]
Architecture	[1],[7],[10],[13],[21],[22],[23],[24]
Idempotency	[1],[2],[6],[8],[16],[22]
Programming Language	[4],[5],[14],[15],[23],[24]
Data Format	[1],[4],[5],[12],[14],[21],[22],[23],[24],[26]
Scalability	[1],[15],[19],[22],[24]
Infrastructure State	[6],[15],[22]
Management Purpose	[1],[2],[6],[7],[8],[9],[10],[11],[12],[14],[15],[16],[20],[22],[26]
Security & Code Smells	[2],[12],[19],[24]
Cloud Integration	[5],[21],[22]
Application Deployment & Distribution	[1],[2],[6],[7],[9],[19]
Usability	[2],[5],[7],[12],[13],[14],[19],[24],[26]

When considering the concepts expressed in the studies, it is observed that the most used terms in defining IaC tools are management purpose, data format, approach, architecture, and usability.

2.1 IaC Tools

Infrastructure as Code tools use scripts to define, update, and execute the creation of cloud infrastructure. Each IaC tool has its own scripting language to describe the infrastructure and processes to be carried out. Commonly used IaC tools (Achar, 2021).

Ansible: Ansible is an IT (Information Technology) automation engine that automates cloud applications (Singh et al., 2016). With both open source and paid versions available, Ansible is primarily utilized as a configuration management tool for managing resources (Artac et al., 2018). The Ansible management tool works on almost all Linux machines running Python 2 or 3. Management of infrastructure resources is achieved through SSH connections established via the node where this configuration management tool operates as seen in Figure 3. This enables the execution of scripts on remote hardware, the removal of executed scripts, and the installation of Python libraries. Similarly, configuration can also be performed on remote hardware through REST APIs (Ning, 2023).

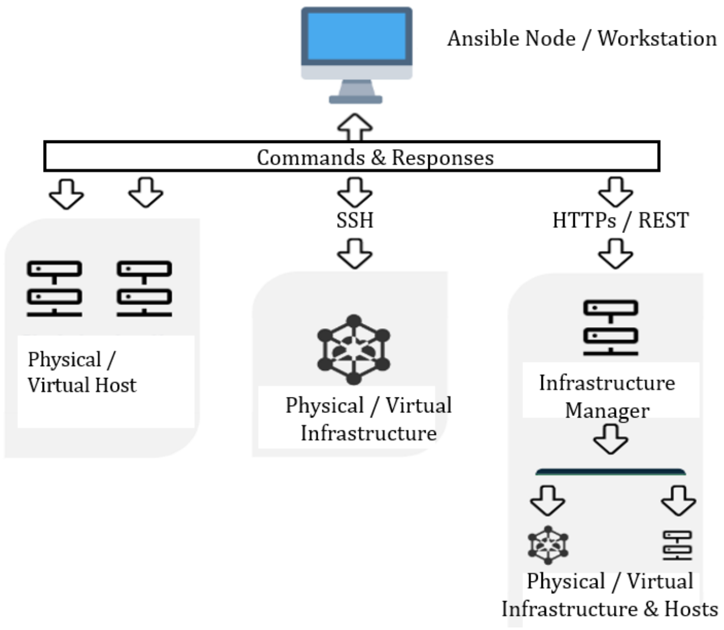
Ansible Node / Workstation
Commands & Responses
SSH
HTTPs / REST
Physical / Virtual Host
Physical / Virtual Infrastructure
Infrastructure Manager
Physical / Virtual Infrastructure & Hosts

***Figure 3**. The architecture of Ansible*

Chef: Chef is an open-source IaC configuration management tool that enables developers and operations teams to automate the process of configuring and deploying infrastructure and applications (L'Esteve, 2023). It serves as a significant IaC tool for managing complex environments. With Chef, users can define infrastructure as code using a domain-specific language (DSL) called Chef DSL (Mustafa, 2023). This allows for the desired state of infrastructure components and applications to be defined, including software packages, system configurations, user accounts, and security settings.

Chef operates by breaking down the configuration process into small, reusable components called "recipes" and collections of relevant recipes known as "cookbooks" (Figure 4). Recipes are individual components of the configuration code, while cookbooks are collections of related recipes (Surianarayanan & Chelliah, 2023). Users can share and reuse cookbooks to create their own infrastructure and applications, thereby facilitating the management of larger, complex environments.

As seen in Figure 4, Chef also includes a powerful tool called "Chef Server," which serves as a centralized repository for configuration data and provides a way to manage configuration changes across multiple nodes. This server ensures that all nodes are consistently configured and aids in simplifying the management of large, complex environments.

Puppet: Puppet is an IaC configuration management tool that was founded as an open-source solution in 2005 but later evolved into a commercial platform. The Puppet Server, also known as Puppet Master, is the core component, which includes the information gathering service called Facter, and components like PuppetDB that store events, node catalogs, current configurations, and historical data.

Puppet consists of both client and server components. The client, also known as the Agent, is securely installed, and configured on target machines. The client and server authenticate each other using self-signed certificates. The Agent collects events under the control of the Facter service and applies configuration changes as directed by the Puppet Server (Bessghaier et al., 2023). Puppet includes modules that enable connectivity for Cloud APIs and hardware that cannot run agents. User interactions are typically conducted through SSH and the command line (Figure 5).

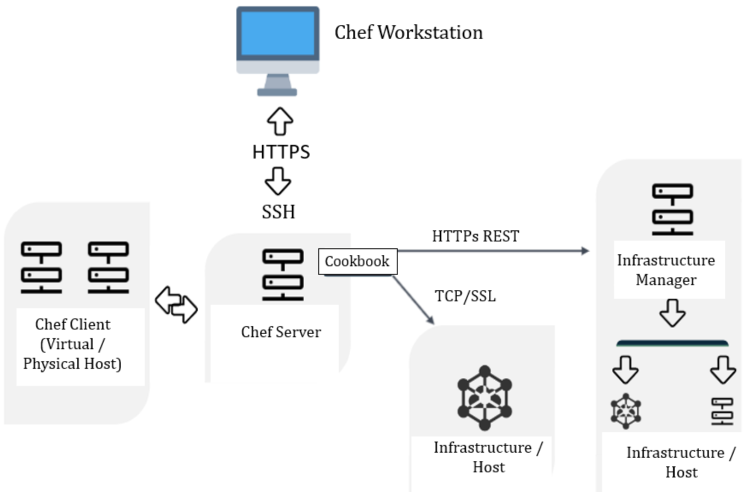

***Figure 4**. The Chef tool architecture*

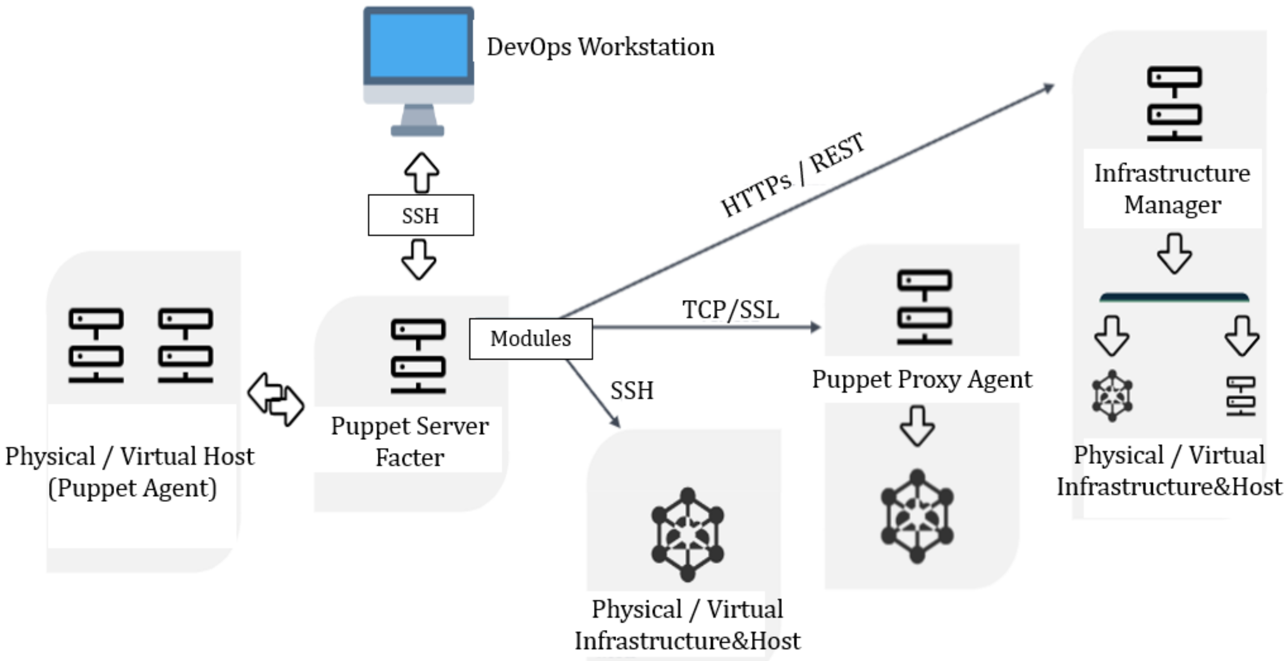
DevOps Workstation
SSH
HTTPs / REST
Infrastructure Manager
TCP/SSL
Modules
Physical / Virtual Host (Puppet Agent)
Puppet Server Facter
SSH
Puppet Proxy Agent
Physical / Virtual Infrastructure&Host
Physical / Virtual Infrastructure&Host

***Figure 5**. The architecture of Puppet tool*

After the Puppet server is operational, the Puppet Agent tool can be installed on the desired client to manage it. Server configuration information is stored in the puppet.conf file on the client. The server can now collect information from the client and update its status with any configuration changes. Notifications to be configured are stored in manifest files. Notification files typically have a .pp extension and are written in a declarative language resembling Ruby.

SaltStack: SaltStack is a decentralized model-based Infrastructure as Code tool that operates through a server (Zadka, 2019). As seen in Figure 6, it consists of a central server called Salt Master and clients known as Salt Minions, which act as agents on nodes. One of its notable features is the use of Python scripts and the YAML language (Ning, 2023). Configuration commands are sent to clients via the SSH protocol as events. In the SaltStack architecture, clients and configuration templates are organized in groups, allowing for easier management of the environment. Salt servers can operate redundantly. In case one server becomes unavailable, clients can seek support from another server. This means that multiple master servers can be utilized.

Terraform: Terraform is an open-source IaC orchestration tool developed by HashiCorp (Gupta et al., 2021). It is used to create, manage, and update server infrastructure, storage, networking, and various other services using written configuration files (Terraform, 2023). Terraform enables users to manage infrastructure using explicitly defined configuration files. With Terraform, users can create and configure resources on various platforms such as cloud service providers and on-premises virtual machines. Users utilize a language called HashiCorp Configuration Language (HCL) to define infrastructure in Terraform. HCL allows users to define infrastructure resources in a human-readable and writable format. In Figure 7, the Terraform architecture is depicted.

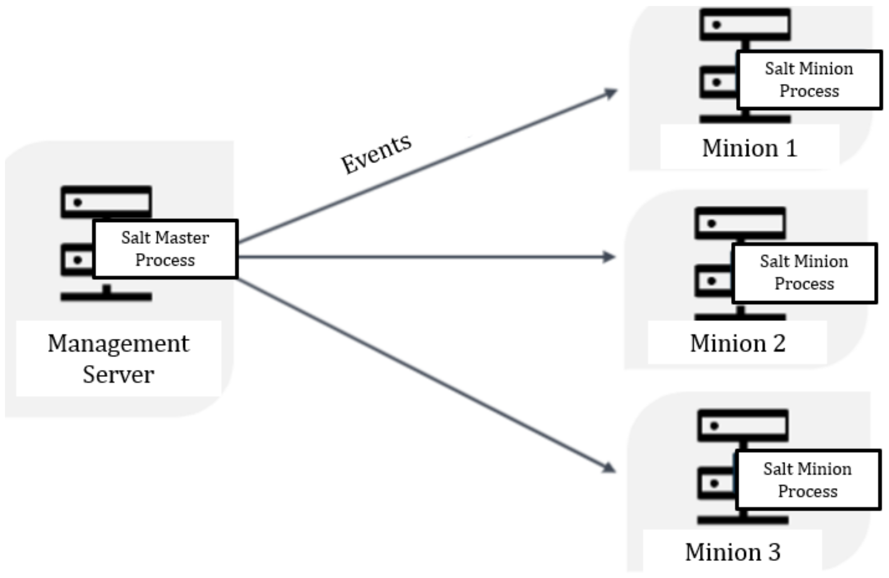

Figure 6*. The architecture of Saltstack*

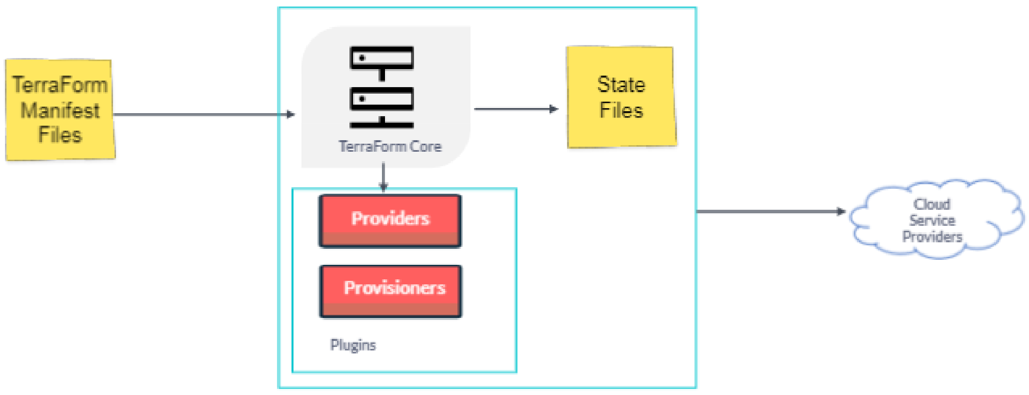

Figure 7*. The architecture of Terraform*

Terraform projects are written in a specialized language called HCL. This language enables the definition and configuration of infrastructure and is formatted to be readable and writable by humans. The command-line interface of Terraform allows users to execute Terraform commands. These commands perform various tasks such as planning, applying, and managing infrastructure.

Terraform can manage various infrastructures, including different cloud providers or on-premises infrastructure providers. Each infrastructure provider includes a custom "provider" plugin that enables Terraform to communicate with these platforms. Terraform's operational state is stored in a 'state file.' This file is used to track the current state of the managed infrastructure and changes made by Terraform. It is crucial to securely store this state file as it represents the actual state of the infrastructure. Terraform modules are used to define reusable infrastructure components. Modules are independent blocks representing different sections of the infrastructure and can be customized with parameters. Configuration Files: Terraform projects typically include configuration files such as 'main.tf,' 'variables.tf,' and 'outputs.tf.' These files determine how the infrastructure is defined and configured. Terraform creates and updates resources as defined, ensuring a specific state, making it a tool suitable for continuous integration and continuous deployment (CI/CD) processes without manual intervention.

CloudFormation: CloudFormation is a service provided by Amazon Web Services (AWS) that allows automatic creation and management of infrastructure resources (AWS, 2023). It is a managed AWS service that enables users to define AWS infrastructure using template files in JSON or YAML format, known as templates. These template files assist users in specifying the resources, relationships, and configurations they want to create. CloudFormation enables users to deploy their infrastructure quickly and in a repeatable manner. These templates can include resources such as virtual private clouds (VPCs), servers, databases, storage solutions, and other AWS services. In Figure 8, the CloudFormation architecture is depicted.

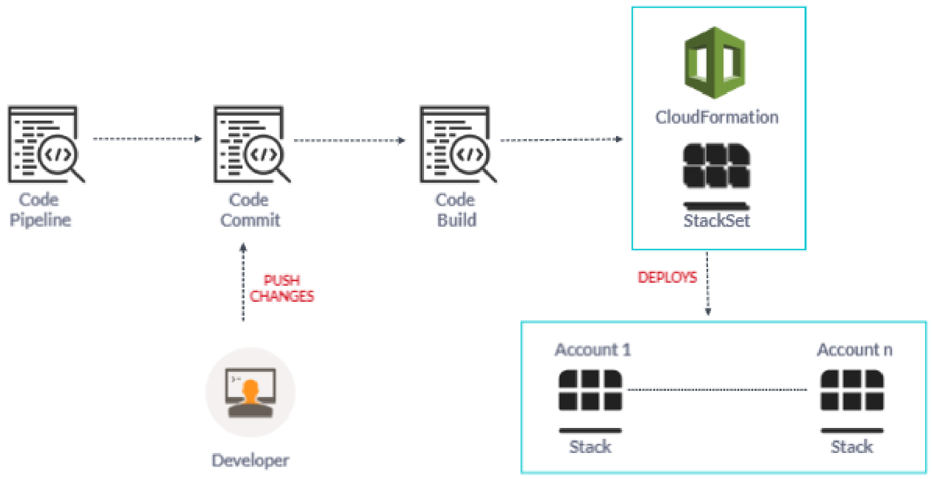

Figure 8. *The architecture of CloudFormation*

Code Pipeline is a fully managed continuous delivery service that automates the build, test, and deployment phases. Code Commit is a fully managed source control service that hosts secure and scalable Git repositories. CodeBuild is a fully managed build service that compiles source code, runs tests, and produces software packages that are ready to deploy. CloudFormation is a service that allows to define and provision AWS infrastructure as code using a template. The template is a JSON or YAML file that describes the resources needed and their configurations.

In summary, the user creates a template and pushes it to CloudFormation. CloudFormation then provides provisioning based on the specified resources. CloudFormation automates tasks such as tracking infrastructure changes, configuring security settings, and documenting the infrastructure. This allows users to manage their infrastructures and utilize their resources.

2.2. Key Concepts for Analyzing IaC Tools

Various concepts need to be considered when provisioning network infrastructure and providing a software development platform. It can be said that these concepts play a key role in the examination and evaluation of

IaC tools. In this section, these concepts that will be used in the analysis are discussed within the framework of IaC.

Idempotency: It is a concept that refers to producing the same result every time a software or script is executed (Rahman et al., 2020). Both in infrastructure provisioning and the delivery of software platforms, the presence of multiple components such as memory, bandwidth, and platforms are required within a collection structure. When scripts are executed, it is expected that these collections yield the same results consistently. The concept of idempotence is related to the situation where a script, even if it has been run before, can be executed again without causing any harm to the intended collections being created. Within this context, idempotence allows for the creation of new, consistent, and standard infrastructure while also facilitating the easy generation of collection components. Another advantage provided by idempotence is the ability to revert to a correctly functioning configuration to rectify errors stemming from incorrect configurations. When an IaC tool with idempotence feature is used, it allows for the reversal of erroneous changes and quick reconstruction within a short period. Idempotence also offers advantages in dynamic structures like cloud computing resource provisioning. It makes it easier to securely fix infrastructure-related issues, perform gradual upgrades, modify configurations, or manage scaling (Salonen, 2020). In essence, idempotence ensures that scripts, like code, reliably achieve the desired end goal on a collection of interconnected resources or components.

Approach: In both software development and infrastructure provisioning processes, the desired outcome is to create an idempotent collection of resources. IaC tools utilize two different approaches to achieve this: procedural and declarative methods (Bellendorf & Mann, 2020; Vladusic & Radolovic, 2020). The procedural approach involves preparing automation scripts by performing specific steps each time. In this approach, scaling creates more management overhead. However, existing configuration scripts are usually more understandable, making it generally an easier approach. In the declarative approach model, the desired infrastructure collection is expressed statically through declarations. In IaC tools that use this model, the existing conditions are examined to provide configurations dynamically, ensuring compatibility by calculating differences to achieve the desired infrastructure. Ansible and Puppet are declarative-based, while Chef is a procedural IaC tool (Muthoni et al., 2021).

Stateless: This concept refers to not storing session information or additional details about the used tool on the server or target systems within the IaC configuration tool. Automation yields the best results when applications are made stateless. In stateless approaches, each configuration request is treated as an independent request. This provides advantages in terms of speed and resource consumption (Salonen, 2020).

Architecture: This refers to whether IaC tools require code running as an agent on the target system to perform the configuration process on the target system (Alonso et al., 2023). Some IaC tools operate in an agent-based manner, while others work without requiring an agent. The installation of an agent onto the target system adds an extra process and burden. Therefore, it can be said that the agentless approach is more flexible (Hasbi et al., 2022). Agent-based approaches often utilize a client/server architecture.

*Infrastructure Stat*e: The process of changing production infrastructure components while all services or applications continue to run normally is known as mutable infrastructure. Such infrastructures bring together and organize components and resources to create a fully functional service or application. If any component, service, or configuration needs a change, it is updated by redeployment without any editing or modification. The old version is stopped, releasing resources for reuse, while the new version is compiled, tested, verified, and deployed. Patching and reconfiguration processes are not performed. One of the significant advantages of mutable infrastructures is the ability to quickly revert to a previous version when needed. On the other hand, immutable infrastructures can simplify configuration management by reducing the server space that needs to be managed by definition files (Johann, 2017).

Programming Language: IaC tools have been developed in various programming languages. The capabilities of these programming languages naturally determine the capabilities of the IaC tool as well (Rahman et al., 2021). Library support and module support, as language-specific capabilities, enhance the usability of an IaC tool. The modular structure provided by the development language assists in ease of maintenance, readability, and familiarity with the language. Furthermore, it allows changes to be applied incrementally and independently (Shvetcova et al., 2019). Therefore, the language of an IaC tool being a widely-used language or closely related to a high-level programming language is an important advantage.

Data Format: Configuration commands executed by IaC tools need to be in a machine-readable format. For this purpose, XML, YAML, and JSON data types are commonly used data representation formats (Quattrocchi & Tamburri, 2023). Having configuration parameters or commands easily extractable in a readable form enhances the capabilities of an IaC tool. While XML is a self-descriptive data representation format, the process of extracting parameters for automation processes by machines can be more complex (Wąsowski & Berger, 2023). JSON is a more commonly used data representation format that expresses data in key-value pairs. Compared to XML, it is easier to read, and key-value pairs can be extracted more easily. Additionally, its similarity to the dictionary data format in the Python programming language has provided it with a broader range of applications. Another data representation format, YAML, consists of a simple structure of key-value pairs. As a result, it is commonly used for automating device configurations.

Management Purpose: IaC tools are used in the realms of provisioning, configuration, deployment, and orchestration from a managerial perspective. Provisioning refers to the acquisition of real or virtual computing, storage, and network infrastructure, enabling communication, bringing services online, and preparing them for use by operators and developers (Sandobalin et al., 2019). Configuration involves performing the necessary tasks, processes, and tests to set up fundamental applications and services, as well as preparing a low-level platform to deploy applications or a higher-level platform. Deployment typically refers to the creation, arrangement, integration, and preparation of multi-component applications or higher-level platforms across multiple nodes (Achar, 2021). Orchestration refers to the processes or workflows that connect automation tasks together to manage workload lifecycles in container environments, dynamically respond to changing conditions, and provide business advantages such as self-service. This is particularly relevant in container environments where various tasks are coordinated to achieve efficient and effective management (Artac et al., 2017).

Scalability: Scalability refers to the ability to manage infrastructure resources in a manner that aligns with the size, complexity, and requirements of an organization or project. In the context of Infrastructure as Code tools, scalability indicates the capability to effectively operate in larger and more complex systems. Scalable IaC solutions are noteworthy for their ability to adapt to factors such as increased workloads, user numbers, or data volumes. The ability of Infrastructure as Code tools to integrate with the cloud is also related to scalability. (Patni et al., 2020).

Security & Code Smells: Security in IaC refers to the built-in mechanisms and practices that help ensure the security of the deployed infrastructure. These features are essential for protecting sensitive data, preventing unauthorized access, and maintaining the overall security posture of the system. IaC tools often integrate with secrets management systems to securely store and manage sensitive information such as API keys, passwords, and certificates. These secrets are accessed programmatically by the IaC scripts without exposing them in the configuration files (Petrović et al., 2022). IaC tools can integrate with vulnerability scanning tools to identify security weaknesses in the deployed infrastructure. Automated scans help detect vulnerabilities, misconfigurations, and potential security threats, enabling timely remediation.

Cloud Integration: Cloud integration capability for IaC tools refers to the ability of these tools to seamlessly interact and integrate with various cloud service providers' APIs (Application Programming Interfaces) to provision, manage, and configure cloud resources (Diaz-De-Arcaya et al., 2023). IaC tools with cloud integration capabilities enable users to automate the deployment and management of cloud-based infrastructures using code.

3. RESULTS AND DISCUSSION

Infrastructure as Code is an application that defines complex processes, often cloud-based deployments, and configurations, through machine-readable code. It encompasses tasks such as configuration, resource allocation, application distribution, and sharing, achieved through various IaC tools. Despite its growing popularity, this study focuses on fundamental key concepts of IaC tools, as presented in Table 4. The same table also includes features that reflect the conceptual differences among IaC tools.

***Table 4**. Conceptual differences of IaC tools*

	Ansible	**Puppet**	**Chef**	**SaltStack**	**TerraForm**	**CloudFormation**
Code Approach	Procedural	Declarative	Procedural	Declarative	Declarative	Declarative
Architecture	Agentless	Client/Server	Client/Server	Client/Server	Agentless, Client-only	Agentless, Client-Server
Language	Python	Ruby	Ruby	Python	Go	AWS Lambda
Data Format	YAML	Embedded DSL	DSL / JSON	YAML	HCL	JSON, YAML
Configuration Tool	Playbook	Recipes / Cookbook	Cookbook	States	Modules, Resources	Template
Infrastructure	Mutable	Mutable	Mutable	Mutable	Mutable / Immutable	Mutable / Immutable
Management Purpose	Orchestration, Deployment, Provisioning	System Management, Code Management, Configuration Automation, Reporting	Developer Based Infrastructure Automation, Automatized workload deployment	System Management, Orchestration, Deployment	Orchestration, Provisioning, Configuration Management, Deployment	Provisioning, Configuration Management, Deployment
Model	Push	Pull	Pull	Pull	Push / Pull	Push/Pull
Ease of Use	Easy	Medium	Medium	Easy	Hard	Medium / Hard
Dependencies	Minimal	Medium	Medium	Medium	Minimal/ Medium	High
Cloud Integration	Multi Cloud	Multi Cloud	Multi Cloud	Multi Cloud	Multi Cloud	AWS Cloud

Idempotency has not been treated as a key value in the comparison and has not been shown in the table, as it is a goal for all IaC technologies. Puppet is a declarative IaC tool, whereas Ansible and Chef are procedural ones. The chef, being procedural, requires creating code step by step to specify how to reach an intended end state. Additionally, a Chef Client agent is needed on each server to be configured. Terraform and CloudFormation are declarative, allowing users to define the desired state of their infrastructure without specifying the step-by-step procedures to reach that state.

From an architectural perspective, when examined, the statelessness of IaC tools provides ease of use and flexibility. In the event of a potential server failure, Salt offers redundancy, making it highly advantageous. Puppet achieves the same effect through an alternative server, while Chef utilizes a backup server. Similarly, Ansible, operating without an agent, becomes a preferred configuration tool. Terraform follows a client-only architecture. It operates as a standalone command-line tool without the need for a central server or agent. CloudFormation operates using a centralized service architecture. CloudFormation uses a client-server model where the client (user or automation tool) sends requests to the CloudFormation service, which then takes care of coordinating the deployment and management.

The languages in which IaC tools are developed contribute to the tool's development and ensuring long-term support. Ansible is a tool written in the Python language, which is why it has found a broader range of applications. When examined in terms of data representation, YAML is more akin to natural language and is considered an easier language compared to XML, JSON, and DSLs due to its higher level of human-readability. Ansible can be considered more successful in this regard when compared to other tools. However, there is variability in syntax integrity within playbooks and other components, resulting in differences from one product to another. Terraform uses the HashiCorp Configuration Language. HCL is designed to be human-readable and easy to write. It uses a simple syntax with a focus on readability. CloudFormation templates are written in JSON or YAML. Terraform operates on a different paradigm. Instead of Cookbooks and Recipes, Terraform uses the concept of modules and resources. Modules are collections of Terraform configurations and resources represent the infrastructure components. CloudFormation uses a template-based approach in which the entire infrastructure is defined as code in a single document.

Configuration management tools like Chef, Puppet, Ansible, and SaltStack inherently follow a mutable infrastructure paradigm. For example, when a new version of an application needs to be deployed, the software update runs on existing servers, and changes occur on the nodes. However, over time, as more updates are performed, each server accumulates a distinct and unique change history. This can lead to difficult-to-diagnose, subtle configuration errors. SaltStack uses a declarative language called Salt State Language to define how systems should be configured. The States in SaltStack are similar in concept to Cookbooks in Chef, but the terminology and approach are different. States define the desired state of a system, and the SaltStack system then applies those states to manage and configure the infrastructure. Both Terraform and CloudFormation provide flexibility in managing both mutable and immutable infrastructure.

When viewed from the perspective of server configuration, Ansible, Chef, Puppet, and SaltStack are tools specifically designed to configure servers using the infrastructure-as-code approach. They utilize configuration definition files with a Domain Specific Language designed for server configuration. The IaC tool reads definitions from these files and applies the relevant configuration to a server. Many server configuration tools use an agent installed on each server. Both Chef and Puppet are designed to work in this manner by default. The concepts of Pull and Push are related to how changes in infrastructure are applied and managed. Ansible uses push model to configure. By default, it uses SSH keys to connect to servers and execute commands. It benefits from not requiring configuration agents installed on managed servers, yet SSH usage can slow down large-scale networks. Furthermore, Ansible is more focused on orchestration rather than just configuration management. Puppet, Saltstack and Chef use pull model. Terraform is more explicitly pull-oriented, both Terraform and CloudFormation can be integrated into CI/CD workflows, allowing for automation and collaboration in a push-centric or pull-centric manner, depending on the use case and preferences.

Cloud integration is an important feature for IaC tools. While Ansible, SaltStack, Chef, Puppet, and Terraform support multi-cloud systems such as Google Cloud, AWS, and Azure, CloudFormation is specific to AWS. Ansible uses modules to interact with cloud APIs, allowing users to manage cloud resources alongside other infrastructure components. Puppet modules and tasks can be used to manage cloud resources, and there are specific modules for different cloud providers.

From a development environment perspective, Ansible and SaltStack are perceived as more advantageous management tools compared to others. These tools are more oriented towards system operators. Conversely, Puppet and Chef tools are arguably more developer focused. The user-friendliness of Ansible makes it ideal for entry-level operations. However, Puppet and Terraform, requiring knowledge of Domain Specific Language, is more developer-oriented in terms of usability when compared to other tools. CloudFormation utilizes AWS Lambda, a serverless computing service that enables users to execute code without the need to explicitly provision or manage servers.

When looking at ease of use, various factors such as the structure of tasks, the prevalence of the programming language, scalability, user knowledge, and code complexity affect usability. In this regard, although Ansible and SaltStack are perceived as more user-friendly and practical, they can still pose challenges for individuals like everyday users or professors who lack highly technical expertise. Conversely, Terraform takes it a step further by introducing its own configuration language, thereby adding another layer of complexity.

Another factor affecting ease of use is the dependencies of the tools being used. Another factor affecting ease of use is the dependencies of the tools being used. Ansible requires minimal dependencies on the managed nodes. It communicates using SSH for Unix-based systems and Windows. Therefore, it doesn't necessitate agent installation on target machines. Terraform is a standalone binary that doesn't require installation on the target systems. It communicates with APIs of cloud providers and other infrastructure services. While it has minimal dependencies, the necessary provider plugins need to be available. Chef requires the installation of a Chef client on each node that it manages. This client communicates with the Chef server. The server, in turn, stores configuration data and cookbooks. SaltStack uses a master-minion architecture. The Salt Minion needs to be installed on target nodes, and they communicate with the Salt Master. While this introduces some dependencies, SaltStack is known for its flexibility and scalability. Puppet follows a master-agent architecture similar to SaltStack. The Puppet agent needs to be installed on managed nodes, communicating with the Puppet Master. It also requires the installation of the Puppet server. CloudFormation is specific to AWS and is primarily used within the AWS ecosystem. Users need to have AWS credentials and permissions. Dependencies are managed by AWS itself.

In light of all these key concepts, we developed a recommendation tree regarding which tool to use. This tree structure is provided in Figure 9.

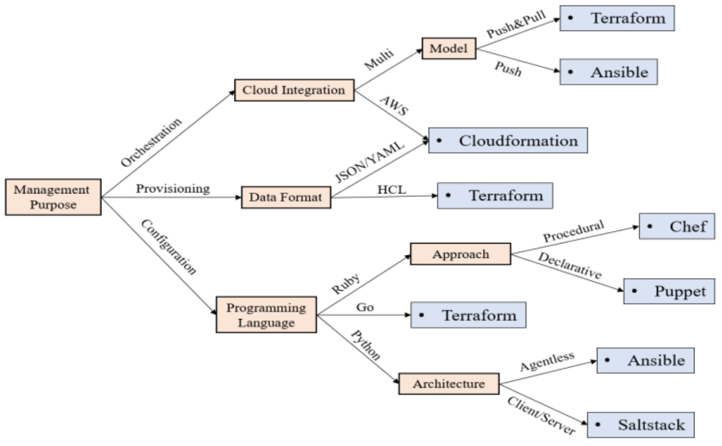

Figure 9. *Differentiation based on the features of IaC tools*

Provisioning, orchestration, and configuration applications can be implemented using Infrastructure as Code (IaC) tools. Depending on the purpose, a network automation or cloud application can be developed by following the tree structure shown in Figure 9. For example, if the goal is to provide provisioning and one is familiar with which data format (HCL, YAML, JSON), options like CloudFormation or Terraform can be chosen. If an orchestration process needs to be performed, the choice between Ansible and Terraform depends on whether Cloud integration is multi-cloud supported or specific to AWS. To make the selection more specific, choosing between a Push model or a push&pull model can be determined, and a choice between these two IaC tools can be made. Relative concepts such as ease of use were not taken into consideration when constructing the tree structure.

This tree structure has been created to clarify the specific use of Infrastructure as Code tools. Objectives, architecture, programming language, data format, and cloud integration capability, among other distinguishing features, assist in focusing on each IaC tool's particular strengths. For instance, for the purpose of provisioning, tools like Cloudformation and Terraform are recommended, while a choice between Ansible and Terraform may be necessary for orchestration processes. Additionally, taking into account factors such as compatibility with a specific cloud infrastructure and ease of use, this tree structure guides the selection of IaC tools tailored to specific use cases. Consequently, users can make more informed and effective decisions when choosing the most suitable IaC tool for their needs.

4. CONCLUSION

In this study, a comprehensive systematic review of Infrastructure as Code tools has been presented with the aim of assessing the current state, highlighting potential future trends, and emphasizing existing challenges. This study significantly contributes to the existing literature by conducting a comprehensive and systematic review of Infrastructure as Code tools. By thoroughly examining IaC tools through key concepts and fundamental aspects such as code approach, architecture, language, data format, configuration tool, infrastructure management, model, and ease of use, this research provides valuable insights into the current state of IaC technologies. The study not only identifies the characteristics and advantages of these tools but also emphasizes the positive impact of integrating multiple IaC tools within the same environment on network automation processes.

In addition to the comprehensive analysis of Infrastructure as Code tools, this study stands out by focusing on the latest updates from articles and gray literature sources. By extracting key concepts from these recent publications, the research ensures that the findings are not only grounded in established knowledge but also reflect the most current trends and developments in the field of IaC technologies. This approach provides a forward-looking perspective, allowing for a nuanced understanding of the evolving landscape of IaC tools. By incorporating insights from the latest articles and gray literature, the study captures cutting-edge practices and emerging challenges, enriching the overall analysis, and contributing to a more holistic view of the subject matter. This approach not only strengthens the validity of the research findings but also positions the study as a valuable resource for both scholars and practitioners seeking up-to-date and relevant information on IaC tools. This study goes a step further by constructing a recommendation tree for the strategic usage of IaC tools. By creating this structured framework, the research provides practical guidance to decision-makers and practitioners in navigating the complexities of IaC implementation. The recommendation tree outlines tailored strategies based on the specific requirements and goals of different projects and organizations.

This structured approach not only helps in the selection of appropriate IaC tools but also guides users on their optimal utilization, ensuring alignment with organizational objectives, scalability, security, and efficiency. The recommendation tree serves as a valuable roadmap, aiding in the decision-making process and enhancing the strategic deployment of IaC tools within diverse contexts. By integrating this strategic perspective, the study not only enriches the scholarly discourse but also offers actionable insights that can be directly applied in real-world scenarios, making it a valuable resource for both academic research and practical implementations in the field.

This study contributes to our understanding of the overall performance and utilization of IaC tools, potentially helping shape future developments more effectively. In addition, aspects such as security, resource utilization, and user-friendliness were not considered within the scope of this study. Conducting research that evaluates IaC tools based on these features would lead to more effective benefits from IaC technologies. Similarly, when considering scenarios involving the simultaneous use of multiple IaC tools in the same environment, conducting analysis and studies on the integration and interoperability of these tools would provide positive contributions to network automation processes.

While aspects like security, resource utilization, and user-friendliness were not within the scope of this study, the research highlights the need for future investigations in these areas. By evaluating IaC tools based on these features and conducting analysis on the integration and interoperability of multiple tools in the same environment, future studies can further enhance the effectiveness and benefits of IaC technologies, contributing significantly to the advancement of network automation processes.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

REFERENCES

Achar, S. (2021). Enterprise SaaS Workloads on New-Generation Infrastructure-as-Code (IaC) on Multi-Cloud Platforms. *Global Disclosure of Economics and Business*, *10*(2), 55-74. https://www.doi.org/10.18034/gdeb.v10i2.652

Alonso, J., Piliszek, R., & Cankar, M. (2023). Embracing IaC Through the DevSecOps Philosophy: Concepts, Challenges, and a Reference Framework. *IEEE Software*, *40*(1), 56-62. https://www.doi.org/10.1109/MS.2022.3212194

Artac, M., Borovssak, T., Di Nitto, E., Guerriero, M., & Tamburri, D. A. (2017, May 20-28). *DevOps: Introducing infrastructure-as-code*. In: Proceedings of the IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C 2017), (pp. 497-498). https://www.doi.org/10.1109/ICSE-C.2017.162

Artac, M., Borovsak, T., Di Nitto, E., Guerriero, M., Perez-Palacin, D., & Tamburri, D. A. (2018, April 30 - May 4). *Infrastructure-as-Code for Data-Intensive Architectures: A Model-Driven Development Approach*.

In: Proceedings of the IEEE 15th International Conference on Software Architecture (ICSA 2018), (pp. 156-165). https://www.doi.org/10.1109/ICSA.2018.00025

AWS Architecture Blog. (2023). AWS Cloudformation (Accesed:13/11/2023) URL

Balon, T., & Baggili, I. (Abe). (2023). Cybercompetitions: A survey of competitions, tools, and systems to support cybersecurity education. *Education and Information Technologies*, *28(9)*, 11759-11791. https://www.doi.org/10.1007/s10639-022-11451-4

Bellendorf, J., & Mann, Z. Á. (2020). Specification of cloud topologies and orchestration using TOSCA: a survey. *Computing*, *102*(8), 1793-1815. https://www.doi.org/10.1007/s00607-019-00750-3

Bessghaier, N., Sayagh, M., Ouni, A., & Mkaouer, M. W. (2023). What Constitutes the Deployment and Run-Time Configuration System? An Empirical Study on OpenStack Projects. *ACM Transactions on Software Engineering and Methodology*, *33*(1), 1-37. https://www.doi.org/10.1145/3607186

Chen, W., Wu, G., & Wei, J. (2018, October 15-18). *An Approach to Identifying Error Patterns for Infrastructure as Code*. In: Proceedings of the 29th IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW 2018), (pp. 124-129). https://www.doi.org/10.1109/ISSREW.2018.00-19

Chiari, M., De Pascalis, M., & Pradella, M. (2022, March 12-15). *Static Analysis of Infrastructure as Code: a Survey*. In: Proceedings of the IEEE 19th International Conference on Software Architecture Companion (ICSA-C), (pp. 218-225). https://www.doi.org/10.1109/ICSA-C54293.2022.00049

Daffalla Elradi, M. (2023). Ansible: A Reliable Tool for Automation. *Electrical and Computer Engineering Studies*, *2*(1), 1-10. https://www.doi.org/10.58396/eces020104

Dalla Palma, S., van Asseldonk, C., Catolino, G., Di Nucci, D., Palomba, F., & Tamburri, D. A. (2023). "Through the looking-glass …" An empirical study on blob infrastructure blueprints in the Topology and Orchestration Specification for Cloud Applications. *Journal of Software: Evolution and Process*, 1-22. https://www.doi.org/10.1002/smr.2533

Dalla Palma, S., Di Nucci, D., Palomba, F., & Tamburri, D. A. (2020). Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software*, *170*, 110726. https://www.doi.org/10.1016/j.jss.2020.110726

De Pascalis, M. (2022). *Formal verification of infrastructure as code*. MSc Thesis, Polytechnic University of Milan.

Diaz-De-Arcaya, J., Osaba, E., Benguria, G., Etxaniz, I., Lobo, J. L., Alonso, J., Torre-Bastida, A. I., & Almeida, A. (2023, April 15-19). *IEM: A Unified Lifecycle Orchestrator for Multilingual IaC Deployments*. In: Proceedings of the Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE 2023), (pp. 195-199). https://www.doi.org/10.1145/3578245.3584938

Falazi, G., Breitenbucher, U., Leymann, F., Stotzner, M., Ntentos, E., Zdun, U., Becker, M., & Heldwein, E. (2022, March 12-15). *On Unifying the Compliance Management of Applications Based on IaC Automation*. In: Proceedings of the IEEE 19th International Conference on Software Architecture Companion (ICSA-C 2022), (pp. 226-229). https://www.doi.org/10.1109/ICSA-C54293.2022.00050

Gorroñogoitia, J., Radolović, D., Vasileiou, Z., Meditskos, G., Karakostas, A., Vrochidis, S., & Bachras, M. (2022). The SODALITE Model-Driven Approach. In: E. Di Nitto, J. Gorroñogoitia Cruz, I. Kumara, D. Radolović, K. Tokmakov, & Z. Vasileiou (Eds.), *Deployment and Operation of Complex Software in Heterogeneous Execution Environments* (pp. 23-52). SpringerBriefs in Applied Sciences and Technology, Springer, Cham. https://www.doi.org/10.1007/978-3-031-04961-3_3

Guerriero, M., Garriga, M., Tamburri, D. A., & Palomba, F. (2019, September 29 - October 4). *Adoption, Support, and Challenges of Infrastructure-as-Code: Insights from Industry*. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution (ICSME 2019), (pp. 580-589). https://www.doi.org/10.1109/ICSME.2019.00092

Gupta, M., Chowdary, M. N., Bussa, S., & Chowdary, C. K. (2021, October 22-23). *Deploying Hadoop Architecture Using Ansible and Terraform*. In: Proceedings of the 5th International Conference on Information

Systems and Computer Networks (ISCON), (pp. 1-6). https://www.doi.org/10.1109/ISCON52037.2021.9702299

Gurbatov, G. (2022). *A comparison between Terraform and Ansible on their impact upon the lifecycle and security management for modifiable cloud infrastructures in OpenStack*. MSc Thesis, Blekinge Institute of Technology.

Hasbi, M., Reza Aristiadi Nurwa, A., Febriyan Priambodo, D., Riski Aulia Putra, W., Sinar Nusantara, S., & Siber dan Sandi Negara, P. (2022). Infrastructure as Code for Security Automation and Network Infrastructure Monitoring. *Teknik Informatika Dan Rekayasa Komputer*, *22*(1), 203-217. https://www.doi.org/10.30812/matrik.v22i1.2471

Heap, M. (2016). *Ansible: from beginner to pro*. Apress. https://www.doi.org/10.1007/978-1-4842-1659-0

Ibrahim, A., Yousef, A. H., & Medhat, W. (2022, May 8-9). *DevSecOps: A Security Model for Infrastructure as Code over the Cloud*. In: Proceedings of the 2nd International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC 2022), (pp. 284-288). https://www.doi.org/10.1109/MIUCC55081.2022.9781709

Johann, S. (2017). Kief Morris on Infrastructure as Code. *IEEE Software*, *34*(1), 117-120, https://www.doi.org/10.1109/MS.2017.13

Karlsson, D. (2023). *Comparison of infrastructure as code frameworks from a developer perspective*. MSc Thesis, Linköping University.

Kumara, I., Garriga, M., Romeu, A. U., Di Nucci, D., Palomba, F., Tamburri, D. A., & van den Heuvel, W. J. (2021). The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and Software Technology*, *137*(March), 106593. https://www.doi.org/10.1016/j.infsof.2021.106593

L'Esteve, R. C. (2023). Applying DevOps. In: R. C. L'Esteve (Eds.), *The Cloud Leader's Handbook: Strategically Innovate, Transform, and Scale Organizations* (pp. 105-122). Apress. https://www.doi.org/10.1007/978-1-4842-9526-7_7

Manner, J. (2023, July 2-8). *A Structured Literature Review Approach to Define Serverless Computing and Function as a Service*. In: Proceedings of the IEEE International Conference on Cloud Computing, (pp. 516-522). https://www.doi.org/10.1109/CLOUD60044.2023.00068

Murphy, O. (2022). *Adoption of Infrastructure as Code (IaC) in Real World Lessons and practices from industry*. MSc Thesis, JAMK University of Applied Sciences.

Mustafa, O. (2023). Understanding DevOps Concepts. In: *A Complete Guide to DevOps with AWS: Deploy, Build, and Scale Services with AWS Tools and Techniques* (pp. 37-78). Apress. https://www.doi.org/10.1007/978-1-4842-9303-4_2

Muthoni, S., Okeyo, G., & Chemwa, G. (2021, December 9-10). *Infrastructure as Code for Business Continuity in Institutions of Higher Learning*. In: Proceedings of the International Conference on Electrical, Computer and Energy Technologies (ICECET), (pp. 1-6). https://www.doi.org/10.1109/ICECET52533.2021.9698544

Ning, A. (2023, February 24-26). *An Ansible-based Distributed Application Architecture Rapid Deployment Scheme*. In: Proceedings of the IEEE 2nd International Conference on Electrical Engineering, Big Data and Algorithms (EEBDA), (pp. 972-975). https://www.doi.org/10.1109/EEBDA56825.2023.10090753

Opdebeeck, R., Zerouali, A., Velazquez-Rodriguez, C., & Roover, C. De. (2020, September 28 - October 2). *Does Infrastructure as Code Adhere to Semantic Versioning? An Analysis of Ansible Role Evolution*. In: Proceedings of the 20th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2020), (pp. 238-248). https://www.doi.org/10.1109/SCAM51674.2020.00032

Opdebeeck, R., Zerouali, A., & Roover, C. De. (2023a). Infrastructure-as-Code Ecosystems. In: T. Mens, C. De Roover, & A. Cleve (Eds.), *Software Ecosystems: Tooling and Analytics* (pp. 215-245). Springer International Publishing. https://www.doi.org/10.1007/978-3-031-36060-2_9

Opdebeeck, R., Zerouali, A., & De Roover, C. (2023b, May 15-16). *Control and Data Flow in Security Smell Detection for Infrastructure as Code: Is It Worth the Effort?*. In: Proceedings of the IEEE/ACM 20th International Conference on Mining Software Repositories (MSR 2023), (pp. 534-545). https://www.doi.org/10.1109/MSR59073.2023.00079

Patni, J. C., Banerjee, S., & Tiwari, D. (2020, July 2-4). *Infrastructure as a Code (IaC) to Software Defined Infrastructure using Azure Resource Manager (ARM)*. In: Proceedings of the International Conference on Computational Performance Evaluation (ComPE 2020), (pp. 575-578). https://www.doi.org/10.1109/ComPE49325.2020.9200030

Petrović, N., Cankar, M., & Luzar, A. (2022, November 15-16). *Automated Approach to IaC Code Inspection Using Python-Based DevSecOps Tool*. In: Proceedings of the 30th Telecommunications Forum (TELFOR), (pp. 1-4). https://www.doi.org/10.1109/TELFOR56187.2022.9983681

Quattrocchi, G., & Tamburri, D. A. (2023). Infrastructure as Code. *IEEE Software*, *40*(1), 37-40. https://www.doi.org/10.1109/MS.2022.3212034

Rahman, A. (2018, May 27 - June 3). *Characteristics of defective infrastructure as code scripts in DevOps*. In: Proceedings of the International Conference on Software Engineering, (pp. 476-479). https://www.doi.org/10.1145/3183440.3183452

Rahman, A., Barsha, F. L., & Morrison, P. (2021, October 18-20). *Shhh: 12 Practices for Secret Management in Infrastructure as Code*. In: Proceedings of the IEEE Secure Development Conference (SecDev 2021), (pp. 56-62). https://www.doi.org/10.1109/SecDev51306.2021.00024

Rahman, A., Farhana, E., Parnin, C., & Williams, L. (2020, June 27 - July 19). *Gang of eight: A defect taxonomy for infrastructure as code scripts*. In: Proceedings of the International Conference on Software Engineering, (pp. 752-764). https://www.doi.org/10.1145/3377811.3380409

Rahman, A., Parnin, C., & Williams, L. (2019, May 25-31). *The Seven Sins: Security Smells in Infrastructure as Code Scripts*. In: Proceedings of the International Conference on Software Engineering, (pp. 164-175). https://www.doi.org/10.1109/ICSE.2019.00033

Rahman, A., & Williams, L. (2021). Different Kind of Smells: Security Smells in Infrastructure as Code Scripts. *IEEE Security and Privacy*, *19*(3), 33-41. https://www.doi.org/10.1109/MSEC.2021.3065190

Raj, K. A., Anand, A., & Sahana, V. (2022). DevOps and Tools Used: A Systematic Review. (Accessed: 12/11/2023) URL

Salonen, E. (2020). *Software Project Services using Infrastructure-as-Code*. MSc Thesis, University of Vaasa.

Sandobalin, J., Insfran, E., & Abrahao, S. (2017, June 25-30). *An Infrastructure Modelling Tool for Cloud Provisioning*. In: Proceedings of the IEEE 14th International Conference on Services Computing (SCC 2017), (pp. 354-361). https://www.doi.org/10.1109/SCC.2017.52

Sandobalin, J., Insfran, E., & Abrahao, S. (2019, September 15-20). *ARGON: A model-driven infrastructure provisioning tool*. In: Proceedings of the ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C 2019), (pp. 738-742). https://www.doi.org/10.1109/MODELS-C.2019.00114

Santos, A., Bernardino, J., & Correia, N. (2023). Automated Application Deployment on Multi-Access Edge Computing: A Survey. *IEEE Access*, *11*(July), 89393-89408. https://www.doi.org/10.1109/ACCESS.2023.3307023

Schwarz, J., Steffens, A., & Lichter, H. (2018, September 4-7). *Code smells in infrastructure as code*. In: Proceedings of the International Conference on the Quality of Information and Communications Technology (QUATIC 2018), (pp. 220-228). https://www.doi.org/10.1109/QUATIC.2018.00040

Shvetcova, V., Borisenko, O., & Polischuk, M. (2019, September 13-14). *Domain-Specific Language for Infrastructure as Code*. In: Proceedings of the Ivannikov Memorial Workshop (IVMEM 2019), (pp. 39-45). https://www.doi.org/10.1109/IVMEM.2019.00012

Singh, N. K., Thakur, S., Chaurasiya, H., & Nagdev, H. (2016, September 4-5). *Automated provisioning of application in IAAS cloud using Ansible configuration management*. In: Proceedings of the 1st International Conference on Next Generation Computing Technologies (NGCT 2015, September), (pp. 81-85). https://www.doi.org/10.1109/NGCT.2015.7375087

Sokolowski, D. (2022, November 14-18). *Infrastructure as code for dynamic deployments*. In: Proceedings of the 30th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering, (pp. 1775-1779). https://www.doi.org/10.1145/3540250.3558912

Sokolowski, D., & Salvaneschi, G. (2023, March 13-17). *Towards Reliable Infrastructure as Code*. In: Proceedings of the IEEE 20th International Conference on Software Architecture Companion (ICSA-C 2023), (pp. 318-321). https://www.doi.org/10.1109/ICSA-C57050.2023.00072

Sokolowski, D., Weisenburger, P., & Salvaneschi, G. (2023). Decentralizing Infrastructure as Code. *IEEE Software*, *40*(1), 50-55. https://www.doi.org/10.1109/MS.2022.3192968

Soll, M., Helmken, H., Belde, M., Schimpfhauser, S., Nguyen, F., & Versick, D. (2023, May 1-4). *Building an IT Security Laboratory for Complex Teaching Scenarios Using "Infrastructure as Code."* In: Proceedings of the IEEE Global Engineering Education Conference (EDUCON, 2023-May), (pp. 1-8). https://www.doi.org/10.1109/EDUCON54358.2023.10125250

Sorour, A., & Hamdy, A. (2022, July 21-23). *DevOps and IaC to Automate the Delivery of Hands-On Software Lab Exams*. In: Proceedings of the 6th International Conference on Computer, Software and Modeling (ICCSM), (pp. 28-35). https://www.doi.org/10.1109/ICCSM57214.2022.00012

Spielmann, D., Sokolowski, D., & Salvaneschi, G. (2023, October 22-27). *Extensible Testing for Infrastructure as Code*. In: Proceedings of the Companion Proceedings of the 2023 ACM SIGPLAN International Conference on Systems, Programming Languages, and Applications: Software for Humanity, (pp. 58-60). https://www.doi.org/10.1145/3618305.3623607

Statista (2023) Usage of cloud configuration tools worldwide in 2023, current and planned. (Accessed: 13/11/2023) URL

Surianarayanan, C., & Chelliah, P. R. (2023). Cloud Integration and Orchestration. In: *Essentials of Cloud Computing: A Holistic, Cloud-Native Perspective* (2nd ed., pp. 305-319). Springer International Publishing. https://www.doi.org/10.1007/978-3-031-32044-6_11

Tankov, V., Valchuk, D., Golubev, Y., & Bryksin, T. (2021). *Infrastructure in Code: Towards Developer-Friendly Cloud Applications*. In: 36th IEEE/ACM International Conference on Automated Software Engineering (ASE 2021), (pp. 1166-1170). https://www.doi.org/10.1109/ASE51524.2021.9678943

Terraform (2023). Terrafform documentation. (Accessed: 10/11/2023) URL

Tripathi, A. (2023). *Provisioning Secure Cloud Environment Using Policy-as-code and Infrastructure-as-code*. MSc Thesis, School of Computing National College of Ireland.

Vladusic, D., & Radolovic, D. (2020, September 1-4). *Infrastructure as Code for Heterogeneous Computing*. In: Proceedings of the 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), (pp. 1-2). https://www.doi.org/10.1109/SYNASC51798.2020.00011

Wąsowski, A., & Berger, T. (2023). Concrete Syntax. In: *Domain-Specific Languages: Effective Modeling, Automation, and Reuse* (pp. 87-142). Springer International Publishing. https://www.doi.org/10.1007/978-3-031-23669-3_4

Witt, A., & Westling, S. (2023). *Ansible In Different Cloud Environments*. MSc Thesis, Mälardalen University.

Zadka, M. (2019). Salt Stack. In: *DevOps in Python: Infrastructure as Python* (pp. 121-137). Apress. https://www.doi.org/10.1007/978-1-4842-4433-3_10

Zhang, Y., Wu, F., & Rahman, A. (2023, March 13-17). *Practitioner Perceptions of Ansible Test Smells*. In: Proceedings of the IEEE 20th International Conference on Software Architecture Companion (ICSA-C 2023), (pp. 325-327). https://www.doi.org/10.1109/ICSA-C57050.2023.00074