IEM: A Unified Lifecycle Orchestrator for Multilingual IaC Deployments

Josu Diaz-de-Arcaya
josu.diazdearcaya@tecnalia.com
Tecnalia, Basque Research & Technology Alliance (BRTA)
Miñano, Álava, Spain

Eneko Osaba
eneko.osaba@tecnalia.com
Tecnalia, Basque Research & Technology Alliance (BRTA)
Derio, Biscay, Spain

Gorka Benguria
gorka.benguria@tecnalia.com
Tecnalia, Basque Research & Technology Alliance (BRTA)
Derio, Biscay, Spain

Iñaki Etxaniz
inaki.etxaniz@tecnalia.com
Tecnalia, Basque Research & Technology Alliance (BRTA)
Derio, Biscay, Spain

Jesus L. Lobo
jesus.lopez@tecnalia.com
Tecnalia, Basque Research & Technology Alliance (BRTA)
Derio, Biscay, Spain

Juncal Alonso
juncal.alonso@tecnalia.com
Tecnalia, Basque Research & Technology Alliance (BRTA)
Derio, Biscay, Spain

Ana I. Torre-Bastida
isabel.torre@tecnalia.com
Tecnalia, Basque Research & Technology Alliance (BRTA)
Derio, Biscay, Spain

Aitor Almeida
aitor.almeida@deusto.es
DeustoTech, University of Deusto
Bilbao, Biscay, Spain

ABSTRACT

Over the last few years, DevOps methodologies have promoted a more streamlined operationalization of software components in production environments. Infrastructure as Code (IaC) technologies play a key role in the lifecycle management of applications, as they promote the delivery of the infrastructural elements alongside the application components. This way, IaC technologies aspire to minimize the problems associated with the environment by providing a repeatable and traceable process. However, there are a large variety of IaC frameworks, each of them focusing on a different phase of the operationalization lifecycle, hence the necessity to master numerous technologies. In this research, we present the IaC Execution Manager (IEM), a tool devoted to providing a unified framework for the operationalization of software components that encompasses the various stages and technologies involved in the application lifecycle. We analyze an industrial use case to improve the current approach and conclude the IEM is a suitable tool for solving the problem as it promotes automation, while reducing the learning curve associated with the required IaC technologies.

CCS CONCEPTS

• **Information systems → Computing platforms**; Data centers.

KEYWORDS

Infrastructure as Code, IaC, cloud continuum, cloud, edge, DevOps, DevSecOps

ICPE '23 Companion, April 15–19, 2023, Coimbra, Portugal
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0072-9/23/04.
https://doi.org/10.1145/3578245.3584938

ACM Reference Format:
Josu Diaz-de-Arcaya, Eneko Osaba, Gorka Benguria, Iñaki Etxaniz, Jesus L. Lobo, Juncal Alonso, Ana I. Torre-Bastida, and Aitor Almeida. 2023. IEM: A Unified Lifecycle Orchestrator for Multilingual IaC Deployments. In *Companion of the 2023 ACM/SPEC International Conference on Performance Engineering (ICPE '23 Companion), April 15–19, 2023, Coimbra, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3578245.3584938

1 INTRODUCTION

Traditionally, industries have relied on their premises for running their desired workloads. This has the advantage that once the entire infrastructure is up and running, the cost of executing the computation is negligible. However, there are some major drawbacks such us the upfront investment that is required for the acquisition of the servers, the cumbersome integration with the authentication and authorization mechanisms already in use, the significant investment required for a dedicated data center, or the need for having a qualified professional in charge of the maintenance. Over the last few years, big technology companies proposed a change in this paradigm by proposing the cloud computing paradigm, in which storage, processing, networking, and analytics services are offered as a service, instead of using the traditional local services. Cloud computing services are usually offered on the following forms: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS) [21]; and the benefits of this paradigm are: cost savings due to only paying what is consumed, increased efficiency, and unparalleled scalability. On the other hand, the rise of more powerful and feature rich infrastructural devices, including Internet of Things (IoT) devices, has led to the conception of the edge computing paradigm. It promotes moving parts of the computation to the edge of the network, where data can be processed and analyzed reducing this way the amount of data that needs to transmit and persisted over the network. This paradigm benefits from the lower latencies of edge devices, hence improving the performance

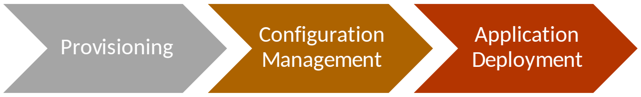

Figure 1: The application lifecycle supported by Infrastructure as Code technologies.

and responsiveness of the applications. In addition, it emphasizes the reliability and security of the architecture by keeping the processing closer to the data source. In this regard, microservices and containerization technologies, which promote the development and management of large applications in self-contained, loosely coupled elements, have come a long way in facilitating the ubiquitous operationalization of such services. This extension of the cloud entities to the edge of the network that provide analysis, processing, storage, and data generation has been coined as the cloud continuum [4].

In this context, methodologies such as DevOps aspire to narrow the existing gap between the development and IT operations teams and improve their communication and collaboration. To do this, it emphasizes the necessity to automatize the various stages of software development and operationalization and release the product in the production environment more often, and with fewer errors. The enormous success of this methodology has led to organizations wanting to replicate this with other methodologies such as Machine Learning Operations (MLOps) [16], Artificial Intelligence for IT Operations (AIOps) [29], and DevSecOps, which aspires to integrate modernized security methods in DevOps [23]. On the other hand, there are various domain specific languages that can be beneficial for the implementation of the cloud continuum such as DOML [9], TOSCA [3], or PADL [10].

The main goal of this manuscript is to present the IEM tool, which aspires to provide a unified interface for the deployment of multilingual IaC projects for the provisioning, configuration management, and application lifecycle of DevSecOps projects. The rest of this paper is structured as follows: Section 2 covers the background that has led to the development of this tool. The IEM itself is described in detail in Section 3. In Section 4, the importance of the IEM in an industrial use case is showcased. Finally, the conclusions and future work are summarized in Section 5.

2 BACKGROUND

Infrastructure as Code (IaC) is the practice to provision local and remote infrastructural devices and configure their system dependencies [27]. As opposed to low-code development platforms, which promote the use of visual environments by stakeholders with little or lack of programming background [31], IaC technologies benefit from applications and methodologies widely used in the DevOps community such as source code management tools [17], continuous integration and deployment pipelines [19], or quality metrics [8]. This upsurge in IaC tools enables an increased coverage of the application lifecycle, which is depicted in Figure 1.

Firstly, the provisioning of the infrastructural devices refers to the process of bringing up the required instances in private or public cloud providers. Each of the major cloud providers have aired their particular IaC solutions for interacting with their services. For instance, AWS CloudFormation [2], Azure Resource Manager [28], and Google Cloud Deployment Manager [30] enable developers and operations engineers to create cloud resources in an orderly manner. Even though they all are appropriate solutions for managing the infrastructural elements of an organization, they force engineers into a single cloud provider as switching from one to another can be a cumbersome process. Due to this, cloud agnostic IaC tools such as Terraform [14] and Pulumi [25] are becoming increasingly popular. The former proposes the use of a domain specific language for the definition of the various infrastructural elements of a project or organization and offers various providers for interacting with the different Cloud services. On the other hand, Pulumi is conceived because of the rapidly evolving cloud necessities of the organizations and provides various APIs for interacting with cloud services from a programmatic approach. Secondly, the configuration management phase refers to the process of programmatically installing the requirements and libraries of an application on a given infrastructural device. There are two distinct patterns in this type of solution: one that interacts directly with the devices, and another one that relies on the use of agents for installation and configuration of the requirements. Technologies such as Chef [7] and Puppet [26] are extremely popular when adopting the client-server approach to configuration management. On the other hand, Ansible [15] and Saltstack [32] promote a more lightweight interaction with the infrastructure by avoiding the use of agents. Finally, the application deployment phase oversees the operationalization and maintenance of the various elements in the application layer. Recently, containerization has become the reigning approach to develop and deploy application in heterogeneous environments, as they represent a lightweight form of providing isolation and resource management [12]. There are various platforms that provide containerization capabilities such as LXC, LXD, and Docker with varying performance overheads [22]. In addition, various projects offer orchestration capabilities over the edge and cloud computation layers. In [11] the authors leverage AIOps methodologies for the deployment of distributed analytical pipelines. Next, the authors offer an MLOps framework for the deployment and orchestration of analytical pipelines using containerization technologies [20]. In [35], a lightweight orchestrator for TOSCA is presented. Yet another orchestrator supporting TOSCA is Cloudify [18], which allows applications to efficiently run across multiple clouds and data centers. Finally, the authors present a platform for the dynamic management of virtual infrastructures [5].

The conceptualization of the IEM has leveraged these methodologies and technologies to provide a unified IaC orchestrator. It aspires to be compatible with the major cloud providers by utilizing an initial set of provisioning and configuration management technologies. In addition, it has been conceived to be an extensible tool so it can support the full lifecycle of DevSecOps projects.

3 THE IAC EXECUTION MANAGER

The PIACERE Project[1], which has been leading the development of the IEM[2], aspires to make the creation of IaC more accessible to designers, developers, and operators; while increasing the quality, security, trustworthiness and evolvability of IaC code [1]. The IEM

[1] https://www.piacere-project.eu/
[2] https://doi.org/10.5281/zenodo.7310937

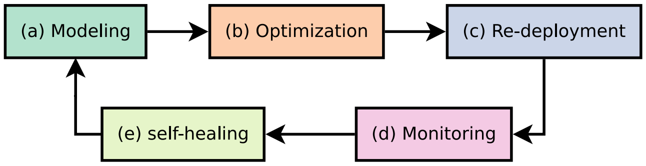

Figure 2: The application lifecycle supported by the IEM.

is the component in charge of operationalization of the various use cases generated by the PIACERE ecosystem. The existing tools already utilized in industry and research focus on one stage of the application lifecycle management. The IEM was envisioned to orchestrate the provisioning, configuration management, and application deployment into one single component. In addition, the projects within PIACERE are required to adhere to the application lifecycle depicted in Figure 2, and specific tools have been designed for each of the depicted stages. First, (a) the modeling of the solution is accomplished via an Eclipse-based IDE[3]. Next, (b) the optimization stage of the solution is conducted by the IOP, which provides DevSecOps teams with the most appropriate deployment configuration subject to a set of predefined constraints [24]. Then, (c) the re-deployment of the optimized solution is exclusively handled by the IEM. Finally, the (d) monitoring and (e) self-healing are conducted by the runtime monitoring component[4].

The IEM relies on the implementation of various engines to orchestrate the lifecycle of the PIACERE use cases. This engine interface has been designed with the goal of being an extensible mechanism so that further implementations can be added in the future. Even tools that are not yet publicly available or designed could be added seamlessly by implementing this interface. In its current state, the two tools that are currently in use are Terraform and Ansible, which are explained in further detail in Section 2. There are several reasons for choosing Terraform as the infrastructure provisioning tool to be used by the IEM: i) it provides an abstraction layer over a myriad of public and private cloud providers [13] including AWS, Azure, or Google Cloud Platform; ii) it is an open source tool with a large community and enterprise support; and iii) it has a declarative syntax meaning that failing deployments can be retriggered and has built-in support for partial redeployments. Next, Ansible has been selected as the configuration management tool for the following reasons: i) it is agentless meaning that it can interact straightaway with the provisioned infrastructural devices, ii) it is also open-source with both business and community support, and iii) a failure during and execution can be solved just by retriggering the failing task. However, these technologies have been designed to be used in isolation, and being able to glue them together into one single tool comes at a cost. In this regard, the IEM provides a unified interface for the utilization of these tools in the form of a REST API.

Projects implemented with Terraform often incur failures due to the lack of resources in the cloud provider, or the stringent security policies imposed by the account owners. This problem is especially acute in private cloud providers such as OpenStack [33], in which the number of resources tends to be more limited than in their public cloud counterparts, which are not that limited in this regard. On the other hand, Ansible tends to fail such as when the provisioned infrastructural devices linger for too long in the creation state and do not react to the provided commands in time. These problems in the provisioning and configuration management stages are not particularly complex to solve when the tools are utilized in isolation from one another. However, given that the IEM aspires to provide a seamless experience to the user, they need to be considered and appropriate mechanisms have been implemented for alleviating their impact.

```
{
  "deployment_id": "string",
  "repository": "string",
  "commit": "string",
  "credentials": {
    "aws": {
      "access_key_id": "string",
      "secret_access_key": "string"
    },
    "azure": {
      "arm_client_id": "string",
      "arm_client_secret": "string",
      "arm_subscription_id": "string",
      "arm_tenant_id": "string"
    },
    "openstack": {
      "user_name": "string",
      "password": "string",
      "auth_url": "string",
      "project_name": "string",
      "region_name": "string",
      "domain_name": "string",
      "project_domain_name": "string",
      "user_domain_name": "string"
    }
  }
}
```

Listing 1: The IEM schema for credentials handling when triggering a deployment.

The first challenge while developing this orchestrator is credential handling. The IEM is designed to be agnostic to the cloud provider, and yet the distinct credentials to be used by the different cloud providers need to be fed into the orchestrator. The schema to be used when performing a deployment or undeployment action on a given project is depicted in Listing 1. At its current stage, the IEM orchestrator offers support for the AWS and Azure public cloud providers and the OpenStack private one, which enables its users to leverage deployments on some of the most common infrastructures and even hybrid ones. The necessary fields for interacting with the different cloud providers have been abstracted and are fed into the IEM to interact with them. In addition, this payload has been designed to be extensible so additional providers not being considered in the first place can be incorporated in further iterations. It is worth noting that these credentials are treated as session variables internally, which provides an isolated environment for the different requests of the systems. Due to this, cloud credentials are never persisted on disk, hence the IEM is safe to be used by distinct users at the same time.

Next, one of the biggest feats has been the implementation of a mechanism so that the different technologies the IEM leverages are glued together seamlessly. This way the user does not need to

[3] https://doi.org/10.5281/zenodo.6821671
[4] https://doi.org/10.5281/zenodo.6821765

dive into the nuances of the underlying technologies and manages the projects from one unified interface. To do this, we have relied on simple YAML files, an example of which is depicted in Listing 2. A typical project is comprised of various folders, each of them represents a different stage of the deployment lifecycle. For the IEM to understand the structure, a configuration file is placed on each folder, including the root of the project. This first configuration file is slightly different than the rest, as it contains the order in which the separate phases must be executed. A stage is only executed should the previous one be finished successfully. The IEM tries to recover from existing failures, but an unrecoverable failure would result in the whole project ending up in a bad state. The rest of the configuration files follow the structure depicted in Listing 2. First, the inputs that are fed into that stage, followed by the outputs of that stage, and the engine to be used. A complete example of such a project can be found in GitLab [34] for further details.

```
---
input:
  - OS_USERNAME
  - OS_PASSWORD
  - OS_AUTH_URL
  - OS_PROJECT_NAME
output:
  - instance_ip
engine: terraform
...
```

Listing 2: Snippet of a configuration file utilized within the IEM.

Finally, the IEM component must provide means for the self-healing strategy of the PIACERE ecosystem, which encompasses an embedded predictive module that tries to know beforehand the behavior of the system based on the incremental learning of the monitored variables. In case some of these variables exceed their threshold, this module warns the self-healing component, and an event may be raised. Should an event occur, the IEM offers a series of endpoints for triggering the appropriate self-healing strategy for the event. These self-healing strategies are automatically scheduled, and the underlying technologies comply with the various engines implemented within the IEM.

In summary, the overarching goal of the IEM is to provide a unified component for the seamless orchestration of the application lifecycle within PIACERE. However, its use expands beyond the project in which it has been conceived, and it can be used independently of the rest of the ecosystem. An in-depth description of the IEM with further instructions on its use is publicly available in Zenodo [6].

4 AN IEM ENHANCED USE CASE

The validation of the IEM has been designed with the following use case: an organization has decided to transition from their manual deployment, which was both time-consuming and error prone, to a more agile continuous delivery. They aspire to reduce the drawbacks they have been experiencing when performing even tiny modifications to their application and reduce the fear of operationalizing software. Their ecosystem is composed of several microservices that are deployed on virtual machines, which they have been deploying on an OpenStack they already own within their premises. Currently, they provision three virtual machines utilizing the OpenStack dashboard, where they manually select the desired resources. Then, they build the required artifacts and copy them over to each of the virtual machines. Finally, they spin up the processes and make sure each service is running smoothly. There are major drawbacks with this process, as failing to properly execute one of the stages can lead to a failed deployment, and a lengthy downtime. Due to this, the organization has decided to pursue the automatization of this process as they believe it can solve many of their problems, and they have been advised to do so by embracing IaC technologies. However, there is a daunting number of frameworks that can cover various stages of the operationalization of their software components, and they do not have the resources to learn them, nor the time to utilize them efficiently. Due to this, they reckon the IEM can be a practical solution because it unifies different IaC engines into one single application and can be utilized free of charge due to being open source.

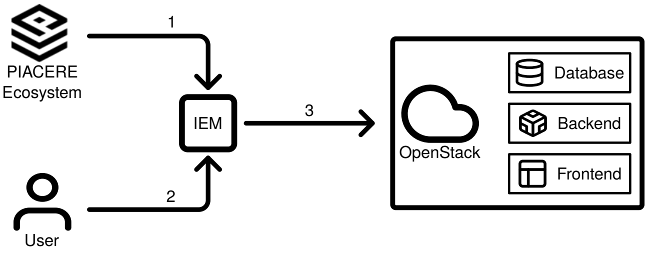

Figure 3: An IEM enhanced use case.

The new scenario for this particular use case is depicted in Figure 3. The IEM can work as part of the PIACERE ecosystem (1) or as an independent entity (2). In this particular use case, the organization has decided to go down this second path, as it does not require the complexity of the whole PIACERE ecosystem. This is one of the benefits of its design, as pure IaC technologies can be fed into its interface, and the user can define the integration between them with the simple configuration files defined in Section 3. First, the user designs its deployment locally, and (2) feeds into the IEM with the appropriate credentials. In this scenario the organization is using an OpenStack private cloud providers the way is depicted in Listing 1. A combination of Terraform and Ansible scripts suffices to get the job done. At this point, the IEM oversees the (3) communication with the OpenStack first, for provisioning each of the required virtual machines with the specified characteristics. Then, it moves to the configuration management stage in which the requirements for each of the microservices are handled and automatically configured. Finally, the actual deployment and execution takes place.

These steps are executed sequentially, and the user does not need to be aware of the specifics of each tool, nor does it need to install their clients. In addition, common problems and pitfalls are handled automatically by the IEM, hence lightening the burden of solving common communication and execution problems from the user. Finally, a set of healing strategies has been defined. For instance, the backend is using a library that even though it is important for the organization has some unfixed memory problems. Should an out of memory problem occur, the IEM provides means for the triggering of healing strategies such as rebooting the application.

In summary, the IEM is an appropriate asset for this organization as it reduces the errors during the operationalization, minimizes the downtime, promotes the automation of the ecosystem, and reduces the learning curve for the integration of the various IaC technologies.

5 CONCLUSIONS AND FUTURE WORK

Recent computing paradigms such as cloud and edge computing require innovative methodologies like cloud continuum. In addition, the success of DevOps practices has paved the way to solutions such as MLOps and DevSecOps. In this research, we present the IEM, a framework devoted to the orchestration of multilingual IaC projects. It excels at seamlessly integrating the provisioning, configuration management, and application lifecycle frameworks into one unified tool. In addition, the proposed use case demonstrates the suitability of the IEM in traditional organizations. It lightens the burden of utilizing multiple IaC technologies and stages off the user and eases the automation of the deployments.

As for the future work, the development of the IEM is tightly coupled with the requirements of the PIACERE Consortium, and its use cases. One of the main goals is to provide functionalities not only with the OpenStack provider as demonstrated in this research, but also with Azure, AWS, and multicloud deployments. In addition, we aspire to offer a better coverage of the application deployment stage depicted in Figure 1 by leveraging technologies such as Docker, Swarm, and Kubernetes.

6 ACKNOWLEDGMENTS

This research was funded by the European project PIACERE (Horizon 2020 research and innovation Program, under grant agreement no 101000162).

REFERENCES

[1] Juncal Alonso, Christophe Joubert, Leire Orue-Echevarria, Matteo Pradella, Daniel Vladušič, et al. 2021. Piacere: Programming trustworthy infrastructure as code in a secure framework. In *CEUR Workshop Proceedings*. CEUR-WS.

[2] Inc Amazon Web Services. 2022. AWS CloudFormation. (2022). https://docs.aws.amazon.com/whitepapers/latest/introduction-devops-aws/aws-cloudformation.html Last accessed 20 December 2022.

[3] Tobias Binz, Uwe Breitenbücher, Oliver Kopp, and Frank Leymann. 2014. TOSCA: portable automated deployment and management of cloud applications. In *Advanced Web Services*. Springer, 527–549.

[4] Luiz Bittencourt, Roger Immich, Rizos Sakellariou, Nelson Fonseca, Edmundo Madeira, Marilia Curado, Leandro Villas, Luiz DaSilva, Craig Lee, and Omer Rana. 2018. The internet of things, fog and cloud continuum: Integration and challenges. *Internet of Things* 3 (2018), 134–155.

[5] Miguel Caballer, Ignacio Blanquer, Germán Moltó, and Carlos de Alfonso. 2015. Dynamic management of virtual infrastructures. *Journal of Grid Computing* 13, 1 (2015), 53–70.

[6] PIACERE Consortium. 2022. D5.1 - IaC Execution platform prototype- v1.1. (Nov. 2022). https://doi.org/10.5281/zenodo.7310937

[7] Progress Software Corporation. 2022. Chef Software DevOps Automation Solutions. (2022). https://www.chef.io/ Last accessed 20 December 2022.

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

[9] Politecnico di Milano/Polimi. 2022. D3.1 PIACERE Abstractions, DOML and DOML-E - v1.0. (July 2022). https://doi.org/10.5281/zenodo.7386060

[10] Josu Díaz-de Arcaya, Raúl Miñón, Ana I Torre-Bastida, Javier Del Ser, and Aitor Almeida. 2020. PADL: A modeling and deployment language for advanced analytical services. *Sensors* 20, 23 (2020), 6712.

[11] Josu Díaz-de Arcaya, Ana I Torre-Bastida, Raúl Miñón, and Aitor Almeida. 2023. Orfeon: An AIOps framework for the goal-driven operationalization of distributed analytical pipelines. *Future Generation Computer Systems* 140 (2023), 18–35.

[12] Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia. 2014. Virtualization vs containerization to support paas. In *2014 IEEE International Conference on Cloud Engineering*. IEEE, 610–614.

[13] Inc. HashiCorp. 2022. Terraform providers. (2022). https://registry.terraform.io/browse/providers Last accessed 1 December 2022.

[14] Inc. HashiCorp. 2022. Terraform public repository. (2022). https://github.com/hashicorp/terraform Last accessed 1 December 2022.

[15] Red Hat Inc. 2022. Ansible public repository. (2022). https://github.com/ansible/ansible Last accessed 1 December 2022.

[16] Dominik Kreuzberger, Niklas Kühl, and Sebastian Hirschl. 2022. Machine Learning Operations (MLOps): Overview, Definition, and Architecture. *arXiv preprint arXiv:2205.02302* (2022).

[17] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A survey of DevOps concepts and challenges. *ACM Computing Surveys (CSUR)* 52, 6 (2019), 1–35.

[18] Cloudify Platform Ltd. 2023. Cloudify DevOps Automation & Orchestration Platform, Multi Cloud. (2023). https://cloudify.co/ Last accessed 15 February 2023.

[19] Ruth W Macarthy and Julian M Bass. 2020. An empirical taxonomy of DevOps in practice. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 221–228.

[20] Raúl Miñón, Josu Diaz-de Arcaya, Ana I Torre-Bastida, and Philipp Hartlieb. 2022. Pangea: An MLOps Tool for Automatically Generating Infrastructure and Deploying Analytic Pipelines in Edge, Fog and Cloud Layers. *Sensors* 22, 12 (2022), 4425.

[21] Chnar Mustafa Mohammed, Subhi RM Zeebaree, et al. 2021. Sufficient comparison among cloud computing services: IaaS, PaaS, and SaaS: A review. *International Journal of Science and Business* 5, 2 (2021), 17–30.

[22] Marek Moravcik, Pavel Segec, Martin Kontsek, Jana Uramova, and Jozef Papan. 2020. Comparison of lxc and docker technologies. In *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*. IEEE, 481–486.

[23] Håvard Myrbakken and Ricardo Colomo-Palacios. 2017. DevSecOps: a multivocal literature review. In *International Conference on Software Process Improvement and Capability Determination*. Springer, 17–29.

[24] Eneko Osaba, Josu Diaz-de Arcaya, Leire Orue-Echevarria, Juncal Alonso, Jesus L Lobo, Gorka Benguria, and Iñaki Etxaniz. 2022. PIACERE project: description and prototype for optimizing infrastructure as code deployment configurations. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 71–72.

[25] Pulumi. 2022. Pulumi. (2022). https://www.pulumi.com/ Last accessed 20 December 2022.

[26] a Perforce company Puppet, Inc. 2022. Puppet - Powerful infrastructure automation and delivery. (2022). https://puppet.com/ Last accessed 20 December 2022.

[27] Akond Rahman, Rezvan Mahdavi-Hezaveh, and Laurie Williams. 2019. A systematic mapping study of infrastructure as code research. *Information and Software Technology* 108 (2019), 65–77.

[28] David Rendón. 2022. Azure Resource Manager. In *Building Applications with Azure Resource Manager (ARM)*. Springer, 9–17.

[29] Laxmi Rijal, Ricardo Colomo-Palacios, and Mary Sánchez-Gordón. 2022. Aiops: A multivocal literature review. *Artificial Intelligence for Cloud and Edge Computing* (2022), 31–50.

[30] Navin Sabharwal and Piyush Pandey. 2021. Getting Started with Google Cloud Deployment Manager. In *Pro Google Cloud Automation*. Springer, 23–70.

[31] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. 2020. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 171–178.

[32] Craig Sebenik and Thomas Hatch. 2015. *Salt Essentials: Getting Started with Automation at Scale.* " O'Reilly Media, Inc.".

[33] Omar Sefraoui, Mohammed Aissaoui, Mohsine Eleuldj, et al. 2012. OpenStack: toward an open-source solution for cloud computing. *International Journal of Computer Applications* 55, 3 (2012), 38–42.

[34] Tecnalia. 2023. IEM public repository. (2023). https://git.code.tecnalia.com/piacere/public/the-platform/iem Last accessed 9 January 2023.

[35] XLAB. 2022. xOpera TOSCA orchestrator. (2022). https://github.com/xlab-si/xopera-opera Last accessed 21 December 2022.