POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Fallacies and Pitfalls of Large-Scale DevOps Pipelines: an Industrial Proof-of-Concept Study

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Marco Tonnarelli**

Student ID: 976686
Advisor: Prof. Damian Andrew Tamburri
Co-advisors: Prof. Giovanni Quattrocchi
Academic Year: 2022-23

Abstract

DevOps is practice that enforces collaboration and automation in software development, guaranteeing high product quality and reliability. This discipline covers many aspects of software development life-cycle, and it leverages on a wide variety of technologies. Among these, Infrastructure as Code is the most prominent, as it allows to automatically configure system dependencies to provision local and remote instances. The adoption of this practices pushes to another level automation in Continuous Integration/Continuous Deployment environments. In the past recent years there have been a great interest in these disciplines, both from academy and industry. We conducted a Systematic Literature Review to understand what are the challenges these practices are currently facing, with particular attention for standards: we looked for the state-of-the-art of standards-based tools, in particular the TOSCA standard. Among these, we found the RADON tool-chain. We conducted a proof-of-concept in an industrial environment to verify the benefits and drawbacks of adopting this framework. In particular, we tested a novel approach to support the modeling and deployment of function orchestrators in a serverless environment with the support of BPMN and TOSCA standards.

Keywords: DevOps, Infrastructure as Code, TOSCA, automation, standards

DevOps è una pratica che spinge verso la collaborazione e l'automazione nello sviluppo software, garantendo un'alta qualità del prodotto e affidabilità. Questa disciplina copre molte fasi del ciclo di vita dello sviluppo software, e fa affidamento su una grande varietà di tecnologie. Tra queste, l'*Infrastructure as Code* è la più prominente, dal momento che consente di configurare automaticamente dipendenze di sistema per la gestione e l'approvigionamento di istanze remote e locali. L'adozione di queste pratiche spinge ad un altro livello l'automazione in ambienti che adottano CI/CD (*Continuous Integration/Continuous Deployment*). Negli ultimi anni c'è stato un grande interesse per queste discipline, sia dall'ambiente accademico sia da quello industriale. Abbiamo condotto una revisione sistematica della letteratura per comprendere quali sono le sfide che queste pratiche stanno affrontando, con particolare interesse per gli standard: abbiamo studiato lo stato dell'arte dei tool basati su standard, in particolare lo standard TOSCA. Tra

questi, abbiamo trovato la RADON tool-chain. Abbiamo condotto una dimostrazione di fattibilità in un ambiente industriale per verificare i vantaggi e gli svantaggi dell'adozione di questo framework. In particolare, abbiamo testano un nuovo approccio che supporta la modellazione e il *deployment* di *orchestrator* di funzioni in un ambiente serverless con il supporto degli standard BPMN e TOSCA.

Parole chiave: DevOps, Infrastructure as Code, TOSCA, automazione, standard

Contents

Introduction

Software engineering is a discipline that came officially into existence - even though most of its practices existed before - at the 1968 NATO Conference held in Germany [115], where the methods, challenges and difficulties of designing complex software system were explored and, for the first time in History, standards and practices were grouped and categorized, covering all the aspects of software development, with the intent of improving and clarify how software should be developed: software engineering was born.

Most of the results obtained by this conference, and many others that followed in the next years, are referenced to as classical Software Engineering (SE) and the proposed methodologies mostly derive from the traditional industry, where state-of-the-art methods to design and manufacture goods of all kinds were already well-defined. In particular, the most evident dependency with other industries is to be found in the so-called plan-driven - or heavy-weight - methodologies, which require upfront requirements definition, documentation and well-structured project plans [54].

It quickly became clear how this traditional manufacturing way of working was not always well-suited for software development: the concepts of detailed plans, project and delivery deadline were obsolete. With the passing years, software systems became more and more complex thanks to new technological advancements: the ever-increasing computational power followed by a decreased cost of hardware components, faster storage systems and the advent of the World Wide Web, have introduced a level of complexity in software systems that could not be reached by applying the classical SE methodologies; in the 1980s, a frequent phenomenon was having dissatisfied customers due to late deliveries [44].

It is in this scenario that some terms, like *agile*, *frequent delivery* and *extreme programming*, begun to gain popularity within industry leading companies: it became clear that it was necessary to adopt a different paradigm to keep-up with the novel complex systems required by customers. In particular, software products started also to change in shape, as the *classical* monolithic application was not anymore the most common product, but it started to get replaced by novel concepts, such as software services, or services delivered

by software [115].

Indeed, looking back at History, we can find that the need of an *agile* development process is something that can be traced back to the 1930s, when the Bell Labs company used this practice to improve product quality [54].

In the traditional way of working, separate teams work on a product and each team plays its own role during development, with poor communication with the others: this is at the basis of the so-called Waterfall Model [107]. The Agile model, instead, breaks this compartmentalization, by enhancing communication among teams, reducing development time and increasing product quality.

As we can observe, the transition from classical SE to Agile methodologies has been quite slow, since not only in some cases Agile was performing worse than the classical method, but also because applying a novel methodology to production lines is difficult, due to operational and technical deficiencies, but also we have to consider that at the time, i.e., before the 1990s, computing was not a mainstream discipline yet [54].

The latest evolution of Agile is DevOps. DevOps is a discipline that covers two main aspects of software manufacturing: development and operations. DevOps has been adopted by a paramount number of industries, and its benefits have been largely disclosed.

Nonetheless, the adoption of DevOps brings several challenges and drawbacks, which sometimes do not overrule the benefits. For this reason, in this study we want to shed light on the most recent challenges this discipline is facing, at two different levels.

First, we will systematically review the literature produced in the past recent years, to get an insight from the academic community on some specific subjects that DevOps covers, but also we will explore in a real world scenario how these challenges are faced by the industry, providing also a proof-of-concept for a novel tool-chain and methodology developed to address many DevOps-related issues.

However, in order to understand the obtained results, the reader should have a clear understanding of the main underlying concepts, that is, DevOps, Infrastructure as Code and TOSCA standard. In Chapter 1 we will describe how our study has been designed, providing the research questions we want to address. Then, to avoid any misalignment, in the first chapter of our study we provide a brief summary of the aforementioned concepts, matched by a summary of our research and a discussion on the related work.

1 | Study Design

Our research consists of an investigation in the DevOps culture, with two main objectives: (i) discover what are the difficulties, pitfalls and challenges the discipline is currently facing; (ii) verify how these challenges impact software development in a real-word scenario, by reporting the experiments we made in collaboration with a company, using a novel tool.

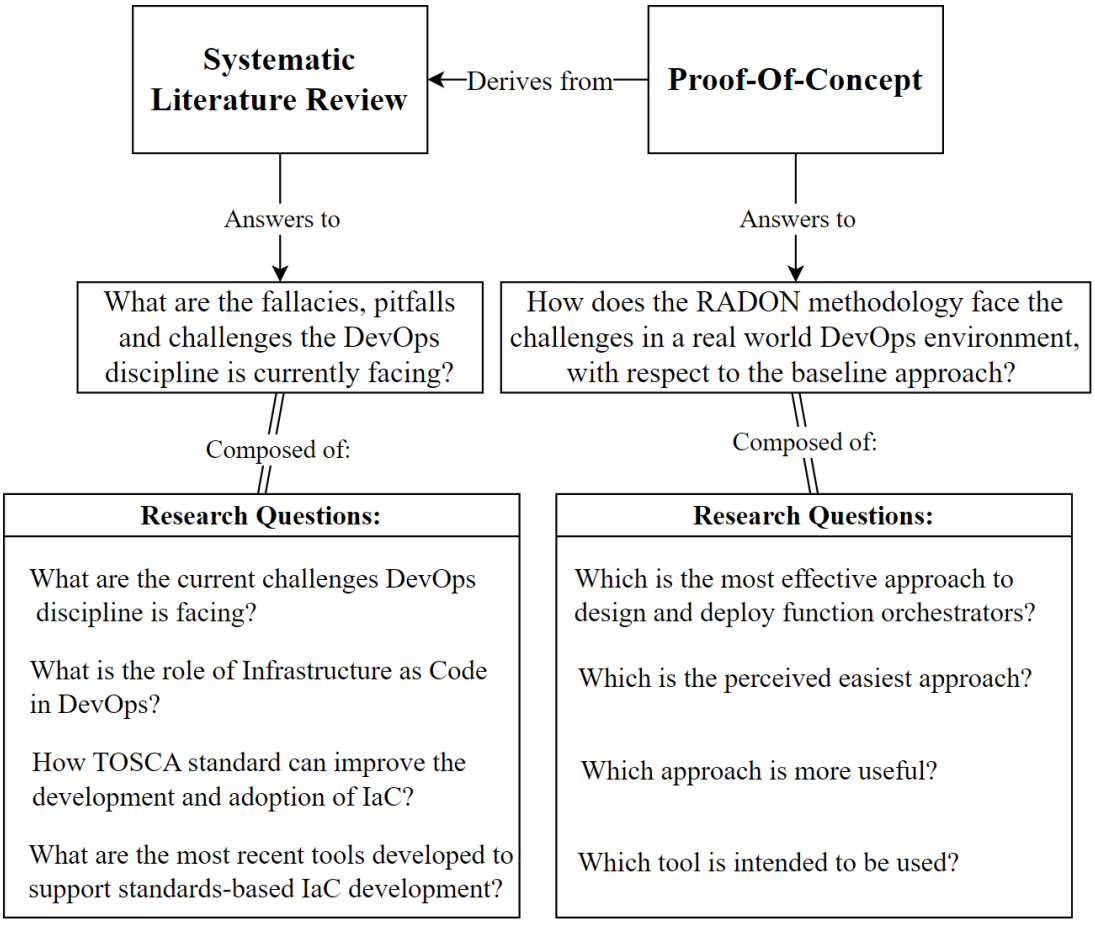

Figure 1.1: Study design overview.

Figure 1.1 summarizes how our study is structured, highlighting what are the research questions each part of the study addresses. We identified two main research questions,

one for each objective, and we derived four specific research questions from each of them.

To achieve the first objective, we conducted a Systematic Literature Review (SLR), analyzing the latest studies regarding DevOps published since 2019: in particular, our focus was on some DevOps sub-topics, namely Infrastructure as Code (IaC), standardization (TOSCA) and tools. The goal of the SLR is to answer the following research questions:

- **RQ1.1.** What are the current challenges DevOps discipline is facing?
- **RQ1.2.** What is the role of Infrastructure as Code in DevOps?
- **RQ1.3.** How TOSCA standard can improve the development and adoption of IaC?
- **RQ1.4.** What are the most recent tools developed to support standards-based IaC code development?

To provide a sound and complete answer, after discussing the motivation of our research, (i) we describe our study design, highlighting our background and explaining the research questions and describing the methodology we employed to provide consistent results; (ii) we define the study selection criteria and the quality concepts we adopted in order to include or exclude papers in our result set; (iii) we define the research queries we crafted to obtain the results, using a specific search engine, justifying why some choices were made; (iv) we evaluate the resulting set, providing insights on the distribution of publications over the years, on the authors, and we also define a DevOps taxonomy that we used to classify the papers according to the man topics they investigate; (v) we discuss the results obtained, producing an in-depth analysis of the papers found classified according to the taxonomy and answering the research question mentioned above; (vi) we present what are the most relevant threats to validity of our study and finally (vii) we conclude the SLR. All of these steps are described in Chapter 4. Our SLR comprises in the final set an amount of 71 papers, each of the classified according to our taxonomy.

Our second objective, instead, has been achieved thanks to a collaboration with a semiconductor company, namely NXP Semiconductors[1], which allowed us to verify the challenges highlighted in our SLR in the DevOps field, permitting us to conduct a proof-of-concept to verify the efficacy of a novel framework created to support the development of serverless function in a DevOps environment, in a team whose main objective is to develop, maintain and monitor data ingestion pipelines that retrieve data from many different sources and load them into a Data Lake. The framework we used, named RADON, is the result of the European Union founded project RADON h-2020[2], it provides a tool-

[1] https://www.nxp.com
[2] https://radon-h2020.eu

chain and a methodology to develop server-less functions and it is based on the Topology Orchestration and Specification for Cloud Application (TOSCA[3]) standard, to be used in conjunction with state-of-the-art cloud services vendors. Thanks to our investigation, we witnessed how the challenges and pitfalls of DevOps impact on a team, and we make another step towards the adoption of standards-based development of IaC scripts: our results confirm the benefits claimed by RADON, but they also show the challenges and limits of the framework. This part of research is described in Chapter 5. The goal of our proof-of-concept is to answer to the following research questions:

- **RQ2.1**: Which is the most effective approach to design and deploy function orchestrators?
- **RQ2.2**: Which is the perceived easiest approach?
- **RQ2.3**: Which approach is more useful?
- **RQ2.4**: Which tool is intended to be used?

Achieving this goal has been possible by adopting the following strategy to conduct our study: (i) we describe and motivate why we have chosen a semiconductor company to conduct our experiments, (ii) giving some insight on the most recent development in the industry and (iii) describing why Data, which is the main asset involved in our scenario, are so valuable; then, (iv) we show the environment we worked in, (v) highlighting the goals of the team we collaborated with; Finally, (vi) we present our proof-of-concept, providing the motivations of our study, the issues the team is facing and the solutions found, and (vii) we compare the performance of the RADON methodology versus the Baseline methodology, that is the standard way of working the team is currently adopting. The results obtained (viii) are discussed and (ix) threats to the validity of our research are presented and (x) conclusions are drawn.

Before diving into our SLR and Proof-of-Concept, however, we introduce the reader to the main concepts tackled by our research: in the next chapter we provide (i) a brief overview of the DevOps discipline, (ii) a description of Infrastructure as Code and what role plays in DevOps, (iii) a summary of the TOSCA standard main characteristics.

[3]https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca

2 | Background

This chapter represents a short and necessary introduction to the main topics which are the subject of our research. We present the key concepts at the basis of the DevOps culture, we show why Infrastructure as Code has become so important within this discipline and we explain what are the most relevant characteristics of the TOSCA standard.

2.1. DevOps Overview

DevOps is the latest evolution of the Agile[1] method for developing software. It is defined as *a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability* [39]. DevOps is a term coined in 2008 [32] and derived from the combination of two words: development and operations. It represents the fusion of these two fundamentals aspects of software life-cycle, by blending in one team tasks that are traditionally delegated to two or more different teams. The integration is done not only by physically merging separated teams, but also by improving communication among other teams in the organizations and combining the tools employed to support the various stages of product development.

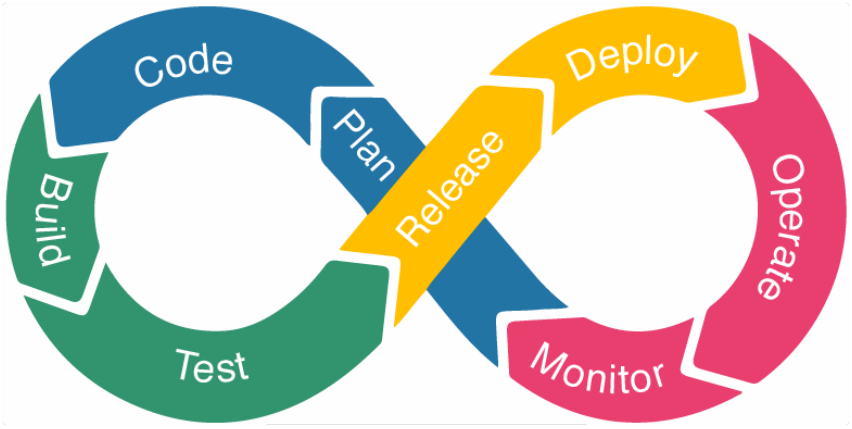

Figure 2.1: DevOps tool-chain stages [120].

[1] https://www.agilealliance.org/agile101/

Figure 2.1 show us how DevOps changes the approach in software development with respect to other traditional ways of working such as, for instance, the waterfall model. Without going deep in the details of each step, here we just want to highlight how DevOps abandons a linear-sequential life-cycle model, adopting instead a circular model.

A key concept of DevOps is automation: it has been shown how automation, applied to different levels and stages of software development, improves software quality and reduces delivery time. Automation is always matched by another concept, that is CI/CD - standing for Continuous Integration / Continuous Deployment - which we will discuss later: however, we stress the fact that in DevOps many concepts rely on each other and this, in combination with a circular model, witnesses dependencies among different practices, creating a relationship loop [64].

From a *process* point of view, DevOps aims to reach business objectives by minimizing risks and costs but at the same time improves product quality by employing a frequent and reliable release process. This process relies on Agile concepts, short feedback life-cycle and continuous improvement. DevOps abandons the rigorous and hierarchical traditional approval schema by replacing it with agile and lean principles.

By look at DevOps from the *people*'s perspective, as we stated before DevOps merges both developers and operators through the culture of collaboration. Collaboration enables sharing knowledge, tools and practices, but at the same time each team achieves a higher level of autonomy: cross-functional teams are a key element of the DevOps culture, as they enhance collaboration and communication.

The *delivery*'s perspective, instead, highlights how Continuous Delivery and Continuous Deployment are at the basis of the whole framework, and they rely on may different factors, such as static analysis, testing automation, deployment pipelines and a wide variety of tools. In the end, all these elements contribute to increase automation, enabling the shift towards micro-services instead of big and batch applications. All of that is matched by a frequent and reliable release process.

Finally, the *run-time* point of view enables to notice how DevOps copes with performance, availability, scalability, resilience and reliability, relying on continuous run-time monitoring and alerting. In the long-run, this also improve the security and the stability of the environment.

2.2. Infrastructure as Code

Infrastructure as Code is one of the main DevOps pillars [74]. IaC is technique that, deriving from practices in software deployment, allow to automatically configure the definition of system dependencies and provision local and remote instances. IaC is a core component of Continuous Deployment. It allows to consistently and repeatably configure routines to configure a system.

The main goals of infrastructure as code are the following [74]:

- dynamically change IT infrastructure;
- make changes to a system a routine, rather than an obstacle;
- define, provision and manage resources independently;
- quick and easy recover from failures.

IaC strongly enforces effortless and reliable build and rebuild of every element of the declared infrastructure. This is because, thanks to Dynamic Infrastructure, which is one of the groundings elements of IaC, it is possible to easily create, modify, delete and move each part of the architecture, independently; this allows making effortless improvements and fixes to the infrastructure. Furthermore, Infrastructure as Code allows delivering multiple and consistent deployments of the same system: this is due to the fact that Infrastructure as Code declaration scripts are reproducible [106]. This is enough to understand why CI/CD practices strictly relies on IaC.

In the past recent years, many tools and technologies have been developed to support IaC, such as Chef[2], Puppet[3] and Ansible[4], which are very popular and as such are used to provision cloud-based resources. For this reason, IaC gained a lot of interest both amongst practitioners and researchers: this however, does not mean that IaC is a fully explored topic, but, as opposite, at the current state it is under-explored.

Limitations and issues has been found regarding many different aspects IaC. Defects and security flaws in IaC scripts can cause serious issues, and for this reason many studies have been conducted to find ways to detect and predict code smells and bad practices: indeed, we will see in our SLR chapter the large amount of tools existing to support this topic. In general, we can state that the situation is made even worse by the fact that a very large variety of tools is available on the market, and each of these tools adopt a

[2]https://www.chef.io/products/chef-infrastructure-management
[3]https://www.puppet.com/products/puppet-enterprise/continuous-delivery
[4]https://ibm.github.io/cloud-enterprise-examples/iac-conf-mgmt/ansible/

different programming paradigm and adopts peculiar design choice.

To address all these issues, researches are pushing towards standardization: in fact, there is evidence to support the thesis that adopting standards for the development of IaC scripts reduces security-related issues and improves the quality of the scripts. In addition, it can reduce the effort of practitioners, who have to learn many different languages and tools, most of the time even in within the same team. Amongst the available standards, the OASIS Topology Orchestration and Specification for Cloud Application Standard is the most prominent one. In addition, the adoption of standards-based DevOps methodologies, in general, might solve another important issue that arises in this context.

Cloud systems are usually provided by a set of vendors that on one side provide an ever-increasing amount of proprietary and open source state-of-the-art services, but on the other side they may introduce an issue, known as vendor lock-in: this is the problem of becoming dependent to a particular vendor, and the migration towards another becomes so expensive that it is better to stick with the older one, even if it provides less services or it is more expensive to maintain. In this perspective, vendor lock-in is a problem solvable, again, by adopting standardization.

2.3. Standards: TOSCA

Lack of interoperability between tools, the sheer amount of components, dependencies and constraints have made deployment and management tasks more complex than ever before. The Organization for the Advancement of Structured Information Standards (OASIS) in 2013 the Topology Orchestration and Specification for Cloud Application standard to support automated deployment and management of cloud applications [105].

TOSCA enables to declare the description of the topology of a cloud application through a graph that is made of two components: nodes, that represent each element composing the application, and relationships, which define the dependencies among different components [51]. Nodes are also able to provide management operations definition, such as creating or configuring an element. Since TOSCA allows reusing components, it represents nodes and relationships in templates: node templates and relationship templates.

TOSCA supports two types of processing: imperative processing and declarative processing [15]. In the first case, a management plan explicitly defines the operations and the execution order of them. In the second one, instead, there is no definition of management plan: the application topology is interpreted at run-time and the management of operations are inferred by the structure of the topology and the definition of its components.

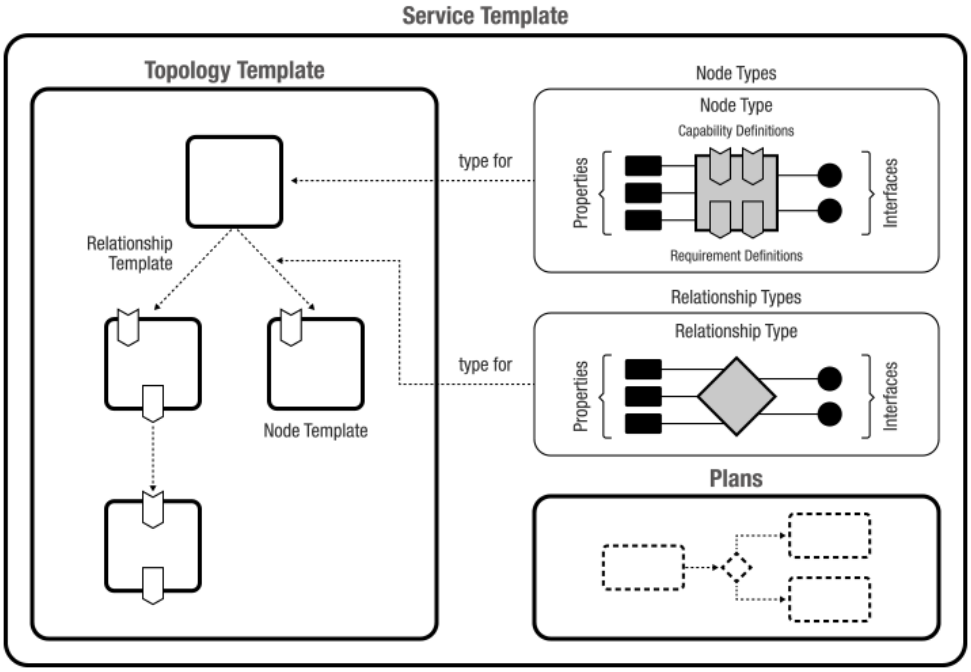

Figure 2.2: OASIS TOSCA Service Template overview.

Figure 2.2 shows a high-level representation[5] of how nodes and relationships interact to create the so-called topology templates, that represent the deployment model specified for a specific application, but that can be re-used in other deployments. All together, nodes, relationships and topology template create a service template.

In the past years, the TOSCA standard has been extended to support more and more policies, that represent the way nodes are defined. Also, there are several methodologies for processing TOSCA models, which can also enable automated topology completion and grouping nodes by matching several ones, to simplify the topology complexity. Other methodologies support the combination of declarative and imperative processing. In general, TOSCA is a very powerful and malleable standard that can be easily adapted to various usages.

Several tools exists to describe, deploy and instantiate TOSCA-compliant cloud applications, to be used in conjunction with the most advanced cloud services available. In our proof-of-concept chapter, we will use some tools that are at the basis of the RADON h-2020 project, which is based on TOSCA. These tools are Eclipse Winery, that is a web-based topology modeling tool, and xOpera, that is a deployment engine capable of executing the defined management plans and deploy the application on the specific target architecture.

[5]http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html

3 | Related Work

Our study is composed of two main parts: the one where we present the Systematic Literature Review, and the one where we discuss our proof-of-concept. For this reason, in this chapter we bring to the attention of the reader the related works we found, according to each argument in our research. In this section, we first discuss related works to our SLR, and then we move to the proof-of-concept. It is necessary to discuss them separately because, even if they are strictly related, they are two completely different kind of studies which apply different methodologies.

3.1. SLR

In the past recent years, there have been an increasing amount of interest in DevOps related topic, and thus the number of publications regarding this research area has always been raising. For this reason, many SLRs and surveys have been conducted to constantly keep-up with novel publications, by classifying them and analyzing the current research direction and making suggestions on which ones might be the next.

The first study we want to mention is a comprehensive survey on DevOps concepts and challenges [64]. This impressive research provide a complete review of the challenges DevOps is currently facing. It provides a conceptual map of the overall DevOps discipline, identifying 4 main concepts: two *Dev* related and two *Ops* related; delivery and process, run-time and people, respectively. These four dimensions reflect the definition of DevOps: *DevOps is a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability [39].* This definition is the one we used as a reference in our research. This study also provides a review of the state-of-the-art tools, discussing also the implications of adopting DevOps practices for engineers, managers and researchers and providing an impressive discussion of the DevOps unresolved challenges.

We consider another extensive study which covers another big topic which is strictly connected to DevOps: serverless computing. This study [50] cover a wide variety of

aspects related to serverless computing, which can be divided in three main categories: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). By exploring these three categories, this survey investigates the challenges in these research areas. In particular, it brings to the attention of the reader a wide variety of issues that still need to be solved such as, for instance, vendor lock-in, resource limits, performance and many others, but tools are also analyzed. Furthermore, it is also tackles another important topic, that is the migration of monolithic applications to serverless computing, a very complex task that is still under investigation.

On the same topics, with particular attention for FaaS, we have found a literature review [46] that provides an important set of results. In particular, it is discussed a comparison of the performance and scalability differences among the different Function as a Service (FaaS) providers; a comparison is also made regarding the constraints associated with FaaS vendors, regarding resources, integration and language run-times. Pricing is another hot topic, and a comprehensive analysis of run-time costs differences among vendors is missing. The last argument brought by this literature review is the level of integration provided by each vendor between FaaS and the other services. Similar topics are also captured by another study [42] , which is not properly a survey but has a similar structure, as it discusses what are the main challenges and limitations of implementing serverless applications.

Moving towards Infrastructure as Code territory, we can find a survey that explores the challenges of adopting Infrastructure as Code in a DevOps environment [47]. In particular, it explores the state-of-the-art of tools for configuration management and frameworks, considering them from a practical point of view. Furthermore, a short but comprehensive research regarding the latest developments of the research around IaC is given by Staron et al. [106], as they describe what are the most recent tools developed in this field to support IaC deployments.

By looking at what are the challenges that arise during the adoption and the support of Infrastructure as Code, we fond the work by Guerriero et al. [47], where they conduct a survey discussing these topics from the point of view of the industry. The shed light on what are the best practices in infrastructure as code development, they discuss what are the available tools, and they witness what are the challenges practitioners are facing. Their findings are similar to the ones we previously discussed: they found out that the main issue is currently related to the fact that there are too many tools, languages and methodologies to support IaC development, and there is the need of standardization. The TOSCA standard is mentioned as one of the possible solutions.

Around IaC we discovered other surveys and SLRs. One survey [48] analyzes a large amount of papers to find six best practices for testing IaC scripts, they are: avoiding coding anti-patterns, behavior-focused test coverage, remote testing, sandbox testing, testing every IaC change and use of automation. An SLR by Nedeltcheva et al. [75] explores what modeling approaches ti generate IaC, concluding that there is no one-size-fits-all solution and identifying key gap ares that needs to be further investigated.

Regarding IaC, another hot topic is related to defects, code smells and security smells in IaC scripts. We mention three extensive surveys which bring a classification of these issues. The Seven Sins paper [86] identifies what are seven most recurring security smells in IaC, and practices to detect them. They are: admin by default, empty password, hard-coded secret, invalid IP address binding, suspicious comment, use of HTTP without TLS and use of weak cryptography algorithms. The Gang of Eight paper [87] provides instead a taxonomy for categorizing IaC defects, which are eight categories: conditional, configuration data, dependency, documentation, idempotency, security, service and syntax. A similar work [89] is proposed by the same authors, which is at the basis of the one mentioned above.

Finally, we mention the work of Bellendorf et al. [11], the most complete survey we found regarding the TOSCA standard. They provide a taxonomy used to classify paper regarding the contribution in TOSCA. In particular, the main taxonomy field they found are: tools, extension of language, methodologies for processing TOSCA models, relation to other solutions, usage of TOSCA and TOSCA introduction. For each of these categories there are sub-categories, and for each of them are classified and analyzed several papers. This survey has been fundamental to our study, since it provides a previous background on the state-of-the-art of TOSCA. In the survey not only are discussed the studies found, but are also suggested some areas that need further exploration.

3.2. Proof-of-Concept

Since the RADON tool is quite recent and not yet massively adopted, few works we have found related to the RADON IDE, and they are all produced by the team that worked in the development of the tool itself. Nonetheless, here we mention some related works that, even if do not necessary apply directly to RADON itself, they are worth mentioning as they share some similarities with our study.

The 2019 study by Sandobalin et al [93] is strictly related to our work as, even if it does not tackle RADON or similar methodologies, it gave us a model to follow to conduct our experiments, that we will discuss in Chapter 5. It is however worth mentioning that they

compare two tools to support IaC, one model-driven and the other code-centric. These two tools are very popular, and they are used in a wide variety of studies, especially the ones devoted to classify, detect and predict code smells.

It is worth mentioning also an empirical study by Leitner et al. [65], which provides interesting insight on the development of Function as a Service from the industry. This study also provides a systematic literature review and a web-based quantitative survey. In this studies are explored the challenges and issues practitioners are facing in the industry, and we used this as a reference during our experiments.

We then consider another study [27], where the Rational Decomposition and Orchestration for serverless computing, namely RADON, is presented: here the main objectives and early results are presented, and we mention it in the related work not for the kind of study but rather because it represents a quick introduction to the overall RADON methodology, which is the basis of our study.

Other related studies are the ones by Yussupov et al. [117, 118] where they extend TOSCA specifications to support the modeling and deployment of function orchestrators in a serverless environment, by using two tools that compose the RADON tool-chain, namely Eclipse Winery and xOpera, in combination with BPMN modeling language. In these study is presented the actual tool-chain and approach we tested in our study: it not the full RADON approach, but a subset of it. The password here is automation: their objective is to extend the radon methodology to support a wider variety of applications. We will discuss each tool and the overall idea more in detail in Chapter 5. The same approach, which we can refer to as BPMN and TOSCA modeling, is presented also in another paper [21] by different authors, namely Calcaterra et al. which worked also with Yussupov et al. in other studies, and they contributed to the RADON project.

BPMN and TOSCA modeling is also enhanced by the same authors in another study [38] where they introduce the Continuous Testing Tool, which is part of RADON IDE and it is based on their previous works. The tools is envisioned for three main use cases: define test cases, execute test cases and maintain test cases. In this study, the underlying architecture is described in detail and the efficacy of the tool is shown.

A methodology and tool-chain similar to RADON is presented in another study [36], where the SODALITE[1] environment is presented. SODALITE provides a set of tools to address issues related to complex task configuration, deployment and operations, enabling faster development of IaC.

[1]https://www.sodalite.eu

We finally mention other two studies by Sokolowski et al. [98, 99] that, even if do not directly related to proof-of-concepts around the RADON tool-chain, they investigate the automation of serverless deployments in DevOps teams. We mention them since we worked in a DevOps team, and they were suffering from some of the issues these studies try to address, in particular dependencies from other teams and dealing with continuous changes in IaC deployments.

4 | Systematic Literature Review

As described in the previous chapters, at the basis of the current evolution of DevOps practices resides Infrastructure as Code (IaC), which is a principle used to manage issues related to configuration, integration and deployment in an automated fashion of infrastructural resources.

Although literature in this field is vast and is evolving really fast, especially in the past recent years, it is necessary to shed light on the most critical research areas. In particular, albeit IaC is a well-studied subject and many surveys and SLRs have been produced, it is still matter of concern which is the current state of the art in this field.

We conducted our literature review following Barbara Kitchenham's systematic methodology [58, 59], which defines a set of guidelines to conduct a research such that the method is rigorous and the results are reproducible. The idea behind these guidelines is the following process: starting from a pilot study, where the key concepts are analyzed and a set of research questions is produced, follow a precise sequence of steps to get, given an initial set of sources, a polished unit of sources to work with.

Our research started with a pilot study, which was fundamental to identify the research questions. The pilot study was conducted by searching for publications related to DevOps and IaC, reaching not only the well-know research venues managed by, for instance, the Association of Computer Machinery (ACM) and the Institute of Electrical and Electronics Engineers (IEEE), but also grey literature articles and blogs. The reason why we made this choice, i.e., consulting a broad kind of resources, is that we wanted to capture the most different voices: in fact, it was immediately clear how the rigorous way of conducting research is not always matched by a novelty of result and, sometimes it is possible to find interesting and novel insights on grey literature rather than on the most popular and well known academic venues.

Before introducing the study design, the search and selection protocol and how papers have been classified, we want to make and important premise. As we stated in the previous paragraph, we follow Kitchenham's guidelines for SLRs. The method, which we provide a brief description in the following section, foresees the involvement of a

small group of people during the selection of papers and evaluation process. This is necessary to produce results which not only are more reliable and complete, but also less biased towards the feelings of the specific person who is performing the review. For instance, it might be possible that one author finds a publication useful for the scope of the research, while another researcher might not agree; or else a situation might occur where one author classifies a paper belonging to a specific category, while another author classifies the publication in another category. So, to avoid these unfortunate situations, the methodology suggests not to work alone, but with a small group of people. In this way, a certain level of objectivity is assured. In our case, this was not possible, since the research and review processes have been conducted by one individual. This is a threat to the validity of this work: nonetheless, we are very confident that this study reaches a good quality level, given that for all the other steps of the methodology we followed the most scientific rigor.

In addition to this important premise, we state that, in the next section, a comprehensive analysis of the internal and external threats to the validity of this study is provided, so that the reader is aware of what are the limits of this research.

4.1. Study design: background and research questions

In the previous chapters of this work, we discussed how DevOps changed the paradigm for software development, thanks to the gradual introduction of new techniques, practices and tools aimed to emphasize the collaboration and the connection among different IT professional figures, both developers and operators, while automating the delivery process and at the same time enhancing reliability.

DevOps is a culture, a combination of many factors which contribute to the most modern way of thinking software development. In particular, we found interesting how Infrastructure as Code has been able to bring the whole process to another level, allowing developers to benefit of the advantages of traditional programming, in a Continuous Deliver/Continuous Integration (CI/CD) environment. Nowadays, Infrastructure as Code is supported by many languages and tools which mostly derive from traditional programming: although this is an advantage, because it allows developers to work in a familiar framework, it also brings challenges to overcome and limitations which might affect the effectiveness of IaC. For this reason, in the past recent years there have been an increasing interest in this field.

The study is structured as follows: as we anticipated in the previous sections, we conducted a pilot study to assess which are the possible paths to follow and identify the research questions. After that, we defined the research queries, identifying the most important keywords. On top of that, we defined the sources selection criteria, describing how a paper was included in the final set and why some were not. Also, it was fundamental to define some quality concepts to assess whether a study was good enough to be considered in the final discussion or not.

In order to properly define these criteria, since the review was conducted by only one person, we compared our initial intuitions and ideas with other papers that we classified as milestones. We consider a paper as a milestone if it is a review conducted scientifically - i.e., explicitly follows some SLRs methodology, be it Kitchenham's or some other technique - and if it produces a well documented result. However, this alone does not grant the rank of milestone to a paper, so in addition we considered that a milestone has to be cited by the majority of the papers we encountered during our research. In the end, we end-up with few papers that we classify as milestones.

	First author	Title	Year
[11]	Bellendorf J.	Specification of cloud topologies and orchestration using TOSCA: a survey	2019
[64]	Leite L.	A Survey of DevOps Concepts and Challenges	2018
[86]	Rahman A.	The Seven Sins: Security Smells in Infrastructure as Code Scripts	2019
[41]	Elazhary. O	Uncovering the Benefits and Challenges of Continuous Integration Practices	2021
[62]	Kumara, I.	The do's and don'ts of infrastructure code: A systematic gray literature review	2021
[87]	Rahman, A.	Gang of Eight: A Defect Taxonomy for Infrastructure as Code Scripts	2020
[85]	Rahman, A.	A systematic mapping study of infrastructure as code research	2018

Table 4.1: List of milestones.

A publication, in order to properly fall in the milestone classification group, needs to meet some criteria. These criteria are defined as a larger subset of the criteria we used to include

studies in our research. The quality and selection criteria will be discussed in the detail in the next section and for this reason here we will just point out the main differences and addition made to these criteria to build the milestone class. First of all, all the paper classified as milestone are among the most cited and reviewed in their respective field. This is a fundamental metric as it gives the publication stronger credibility and authority to a specific study. On top of this preliminary requirement, we also included in the criteria the fact that the paper must be a survey, a literature review, a taxonomy or a classification regarding challenges and discussing the state of state-of-the art of a specific subject. Furthermore, the survey or Literature Review it should no be just a recap of studies made before, but must provide new insights and a string critic to a topic. Finally, the study must be recent as the other ones included in the study: only paper from 2019 have been considered. However, there are just two exceptions: there are two papers published in 2018. The reason why are included is that they are so influential and cover topics so important for our research that we found relevant to include them.

Table 4.1 shows the papers we used as comparison and verification for the criteria employed in our survey: the milestones. As we can see, they are all surveys, systematic literature reviews or taxonomies. They cover specific topics, with well-defined research questions, and they justify every decision made during the research; but, must importantly, they were conducted by more than one person, granting authority and trust on them.

After the definition of the selection criteria and quality concepts, we started the classification of the obtained results, categorizing papers, making statistics on them, assessing their quality according to the previously defined criteria.

Subsequently, we discussed our results, highlighting the most important concepts we extracted by the sources. With the considerations we made we were able to conduct a case study - discussed in detail in the next chapter - related to the most interesting topic we found: standardization, in particular the OASIS Topology and Orchestration Specification for Cloud Applications standard (TOSCA) and the tools developed around it.

We also devote a small sub-section to explain the threats to the validity of our study, discuss both internal and external validity, to clarify what are the aspect we consider less rigorous in our SLR which might compromise the prerogative of the study of being scientific.

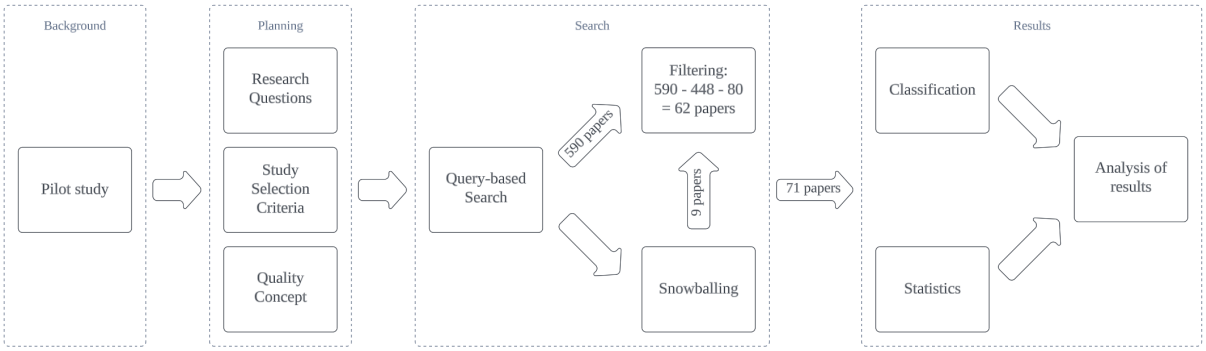

Figure 4.1: Overview of Systematic Literature Review methodology.

Figure 4.1 shows an overview of the previously discussed methodology, that has been applied throughout the study. One important remark we did not previously mention is that, besides the query-based search, we also applied the so-called "snowballing" technique, that is, we enriched our data-set by looking for other sources cited by the main findings of the query-based search. This step of the search process has been critical, since not only the outcome could have - negatively or positively - influenced the discussion of the results, but also we had to be particularly careful in the selection criteria of new sources. For this reasons we established, on top of the study selection criteria baseline, an additional set of rules to strengthen the selection procedure in this delicate phase.

The research questions we want to address in our study are the following:

- **RQ1.** What are the current challenges DevOps discipline is facing?
- **RQ2.** What is the role of Infrastructure as Code in DevOps?
- **RQ3.** How TOSCA standard can improve the development and adoption of Infrastructure as Code?
- **RQ4.** What are the most recent tools developed to support standards-based IaC code development?

With **RQ1** we want to clarify the current state-of-the-art of the DevOps discipline, shedding light on the current challenges this discipline is facing, with particular interest in some of the other topics it encompasses, such as Infrastructure as Code and standardization. On the other end, with **RQ2** we want to make a clear statement on what is the role of IaC in the DevOps field, why it is so important for its application and what are the best techniques available to adopt it. Related to this, **RQ3** has been derived by our pilot study, were we immediately found the OASIS TOSCA standard as the most advance and mature application of standardization to IaC development. For this reason, we want to

investigate why is gaining popularity and what are the advantages of its adoption. Finally, with **RQ4** we further investigate the standards-based infrastructural code development by searching for the most recent and advanced tools available in this field.

4.2. Study selection criteria and quality concept definition

In order to find papers, we used Elvasier's Scopus Search Engine: the reason why we used only one search engine, and we did not use other well-know libraries, such as ACM Digital Library, Google Scholar and IEEE Explore, is to be found in the limitations we encountered in our research; the most important is that it has been conducted by one person. Using multiple search engines would have produce an incredible amount of results, which of course would have produce a more complete study, but it would have not been possible to classify all of them, considering that by using only Scopus we obtain an initial set of slightly less than 600 papers. Furthermore, all the other engines we mentioned do no provide a straightforward way of putting the results in a database, such as a CSV file: third-party tools would have been needed, and this would have increased the time necessary to conduct the study. Another important reason is that Google Scholar provides result by searching on many different sources, and it also includes grey literature and does not provide a fine-grained research.

For all these reasons, in the end we decided to used Scopus since it was the most convenient for our study, allowing exporting results directly in a CSV file. Given the result set obtained by the search - we will discuss the search queries used in the next section - we designed a set of inclusion/exclusion criteria, to filter out non-relevant papers.

Table 4.2 summarizes all the criteria we used in our filtering step. First of all, a study published before 2019 is not included: we only want to consider studies published from the COVID-19 outbreak, as we previously mentioned. Second, we excluded papers that do not have among the Title, Keywords and Abstract fields one of the following keywords: IaC (also not abbreviated), DevOps, Data Ingestion, Framework, Tool, Standard, TOSCA; this was made to avoid non-pertinent papers. Included papers must also be at least three pages long, since shorter papers are usually shorter version of other papers or just presentation of a work, and they do not go in depth in the explanation of results. We also included only peer reviewed papers, published in journals, conferences and workshops, since this helps in considering only papers that have already been analyzed by the scientific community.

Criteria	Include	Exclude
Year	Papers must be published between 2019 and 2023.	Papers published before 2019 are not considered.
Relevance	Relevant for DevOps, IaC, TOSCA: these keywords must appear in Title, Keywords and Abstract.	Papers that do not have relevant keywords in their Title, Keywords and Abstract.
Quality	Peer reviewed papers in Journals, Conferences and Workshops.	Non peer reviewed papers.
Rigor	Scientific research and validation methods must be use to demonstrate results.	Paper that do not use scientific research or have a superficial approach.
Language	The language used by a paper must be English.	Publication not written in English.
Length	Papers must have at least three pages.	Papers shorter than three pages are not considered.

Table 4.2: Design of our experiments.

Another important criterion is that a paper must use clear validation methods and must adopt a scientific rigor to conduct the study and assess the results: this is important to improve the quality of our results. Finally, language is a criterion, since we only considered studies written in English.

As we mentioned in the previous section, where we present the papers classified as milestones, we discussed how these criteria are a subset of the ones used to classy a paper as relevant or not. However, we also highlighted how just two papers do not meet the Year requirement, as we classified two papers as milestones even if were published before 2019: they were published in 2018. Nonetheless, we already depicted why we made this decision.

4.3. Research queries

The crucial step of a systematic literature review consist of defining one (or more) query(s): each query is based on the criteria defined in the previous section. However, building a query is not a strait-forward process, since the combination of keywords used might result

in very different outcomes. Indeed, the query is the formal representation of the criteria, and the keywords represent each basic concept. So, first of all we introduce all the keywords we used in our queries, listed in table 4.3.

	Keywords	Priority level
Infrastructure as Code	infrastructure as code, infrastructural code, iac	MEDIUM
DevOps	devops, mlops, etl pipeline, data ingestion,	MEDIUM
Specific Topics	challenges, standards, standardization, tool, tools, standard, framework, tosca,	LOW
Publication year	pubyear $>$ 2018	HIGH

Table 4.3: Keywords list with their priority level in the queries and, grouped by topics.

For convenience, keywords have been divided in categories which, however, do not affect the query itself: Infrastructure as Code, DevOps, Specific Topics and Publication year. For Infrastructure as Code we found for main keywords to cover the three possible notations which the concept might be found in literature. For DevOps instead, we not only cover a concept itself, but also other related to it, such as Extract, Transform, Load (ETL) pipeline and Data Ingestion, two important fields in which DevOps has been applied and crucial of our SLR, but also important for the case study presented in the next chapter. Then, Specific Topics identifies all the possible sub-topics of IaC that are relevant for our research, but that do not need to appear all together in the same study: in particular, we cared about standardization and challenges in the IaC field. Finally, Publication year is the keyword used to the fine from which year we want to consider studies, based on the considerations made in the previous sections.

We can proceed now to the presentation of the queries used in our research. Actually, for the final result we used one unique query which, however, is the product of an iterative process where, starting from a simple query, each time we added complexity to gain more and more precision. This is due to the fact that initially, with the simplest query composed by only few keywords, the obtained results where very broad and the amount of found sources was that big that it was impossible to conduct a proper review. Also, we obtained a lot of studies completely out of the score of this study: the number of results where initially around 1000-2000 papers. A number too large to be analyzed by one person

alone.

For these reasons, we started build up more and more complex queries, until the final one was obtained.

```
ALL(((infrastructure AND as AND code) OR (infrastructural AND code)
OR iac) AND (devops OR mlops OR (etl AND pipeline) OR (data AND
ingestion)) AND (challenges OR standards OR standardization OR tool
OR tools OR standard OR framework OR tosca)) AND PUBYEAR > 2018
```

The above query is makes use of all the keywords listed in Table 4.3, according to their priority, i.e., to the place they have in the levels of the brackets. In addition to these keywords, the query is structured in a way to be compliant to Elvasier's Scopus literature search engine, the one eventually used for this study. In particular, at the very beginning, the "ALL" keyword is used to allow the engine to search the for the keywords in many different fields of publications, such as the title, the abstract and the keywords. The reason why we decided to search over all the possible fields is that, at some point during the query refinement process, we noticed that we were obtaining very scarce results: this is due to the fact that in literature some of the keywords may appear under different names, and so we did that to avoid loosing important papers, being unable to identify all the possible versions in which keywords might appear. So, to guarantee an even outcome we allowed a broader search over all the fields, considering also the risk to obtain results that might not be useful for the research: in the end, it was crucial the refinement step, where we filtered the final set of the query outcome according to the inclusion and exclusion criteria which we were unable to model using just a query alone.

4.4. Sources result and classification

The resulting set from the final query discussed above is composed by 590 papers. The set comprises publications from January 2019 to March 2023 (time at which the query has been used on the search portal). After filtering out unwanted papers by reading title and keywords - according to the rules and criteria previously defined - we obtained a refined set of 142 papers: indeed, few papers were completely out of the scope of the research, while others were covering similar topics but not the specific ones covered by our research questions.

With this intermediate set of 142, we started reading abstracts and, when needed, the whole content. Following the exclusion criteria, we obtained 62 papers in the final set.

As we can see in Figure 4.2, we can see how the publications are spread among the years.

In particular, most of the publications fell in 2022, highlighting a big different with 2021 and 2020. We interpret this as a consequence of the COVID-19 pandemic: according to most of the paper we included in this review, there have been an ever-increasing interest in the past recent years, and each year the number of publications seems to be growing; however, the pandemic outbreak affected many aspects of human life, including the research community. This lead to a sudden stop in the research activities, slowing down the process and reducing the number of publications. What we want to say is that, even there has been a growing interest in the subjects of our research, we can not state that the sudden increase in publications of the 2022 year represents what would have happened in normal times, i.e., without the pandemic. Nonetheless, we can state that, now that the COVID-19 outbreak seems to have reached the end, the research community is working again at full speed. In addition to that, we have to consider also that some publications might be recorded according to their last version, rather than the day they were published.

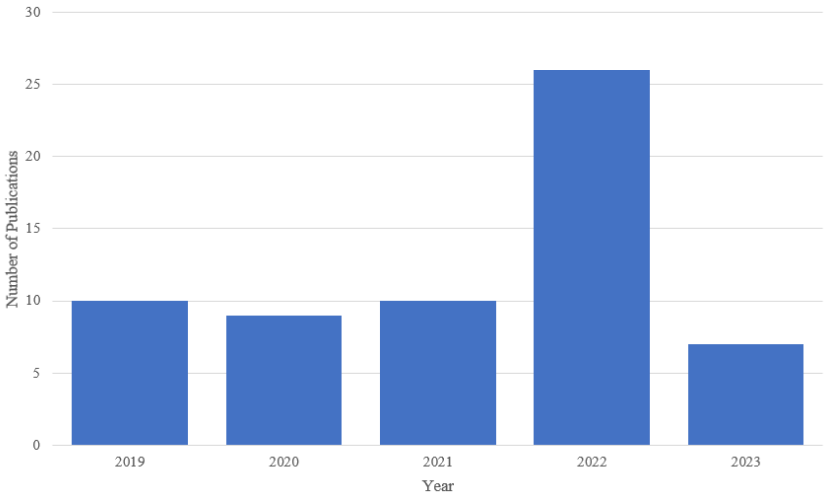

Figure 4.2: Publications distribution per year from 2019 to 2023.

On top of these papers, we found other interesting ones by applying the "snowballing" technique, i.e., we looked at the references of papers, and we included in our research some of the most cited ones, typically appearing in more than one paper's references. Again, also for this part we applied the inclusion and exclusion criteria previously defined, although in this step might rise more subjectivity in the choice with respect to the other steps.

Figure 4.3 shows the most prolific authors found in our research. Out of the 13 authors displayed, 7 are Italian. Also, most of them are co-authors in the same papers, such as Tamburri, Di Nucci and Dalla Palma, who worked together in many publications related to tools, TOSCA, prediction and detection. Other relevant collaborations are the ones

between Rahman and Williams, who are the most prolific authors in the security smells and bad practices research areas. However, we will discuss more in detail the topics tackled by the papers in the following sections.

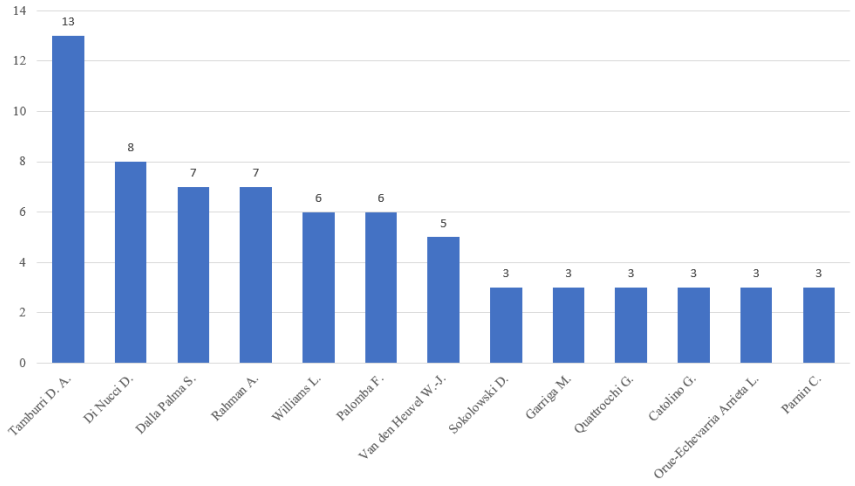

Figure 4.3: Bar chart depicting the authors with most publications.

In addition to the papers found using the Scopus Search in combination with the search query, we also considered some other paper through the so-called snowballing process. Table 4.4 lists all these papers, which in total are 9. They are all related to IaC and TOSCA standard, as they are they main investigation topics of research. They all fall within the time period specified by our query, except for [89] and [117]: we included them anyways, as they tackle fundamental concepts that are at the basis of many other papers, especially from the same authors.

So, overall, in our review we consider 62 papers from the query-based search and 9 from the snowballing process, bringing the total number of papers to 71. It is important to notice that, however, in the classification process and statistical analysis we only considered the 62 initial papers, as we did not want to alter the result of our findings since the snowballing process might be tricky and might not follow the rigor of the query-based search. Nonetheless, they are considered in the final discussion.

	First author	Title	Year
[106]	Staron	Recent Research Into Infrastructure as Code	2022
[89]	Rahman A.	Categorizing Defects in Infrastructure as Code	2018
[100]	Sokolowski D.	Dependencies in DevOps Survey 2021	2021
[13]	Borovitz N.	DeepIaC: deep learning-based linguistic anti-pattern detection in IaC	2020
[5]	Aviv I.	Infrastructure From Code: The Next Generation of Cloud Lifecycle Automation	2023
[24]	Dai, Ting	Automatically Detecting Risky Scripts in Infrastructure Code	2020
[118]	Yussupov V.	Standards-based modeling and deployment of serverless function orchestrations using BPMN and TOSCA	2022
[117]	Wurster, M.	Modeling and Automated Deployment of Serverless Applications Using TOSCA	2018
[38]	Düllmann, T.	CTT: Load Test Automation for TOSCA-based Cloud Applications	2022

Table 4.4: List of snowballed papers.

4.4.1. DevOps Taxonomy

We present now the classification we made. First, we defined a set of categories where each paper might fall. Categories are not mutual-exclusive: a paper that fall in a category might also fall in another one. Indeed, a category represents different aspects of a paper, main topic, secondary topics and type of paper. For instance, a paper can be a Survey or a SLR, and can cover some topics such as IaC and TOSCA.

The taxonomy depicted in Figure 4.4 has been used, as previously discussed, to classify the papers found in our research. This taxonomy has been built by analyzing the key concepts mentioned in the papers tackling DevOps, with the support of other taxonomies and conceptual diagrams provided by some important studies [53, 87, 113]: however, as our findings are reliable, it is not present in literature a complete and organic taxonomy covering all the DevOps concepts.

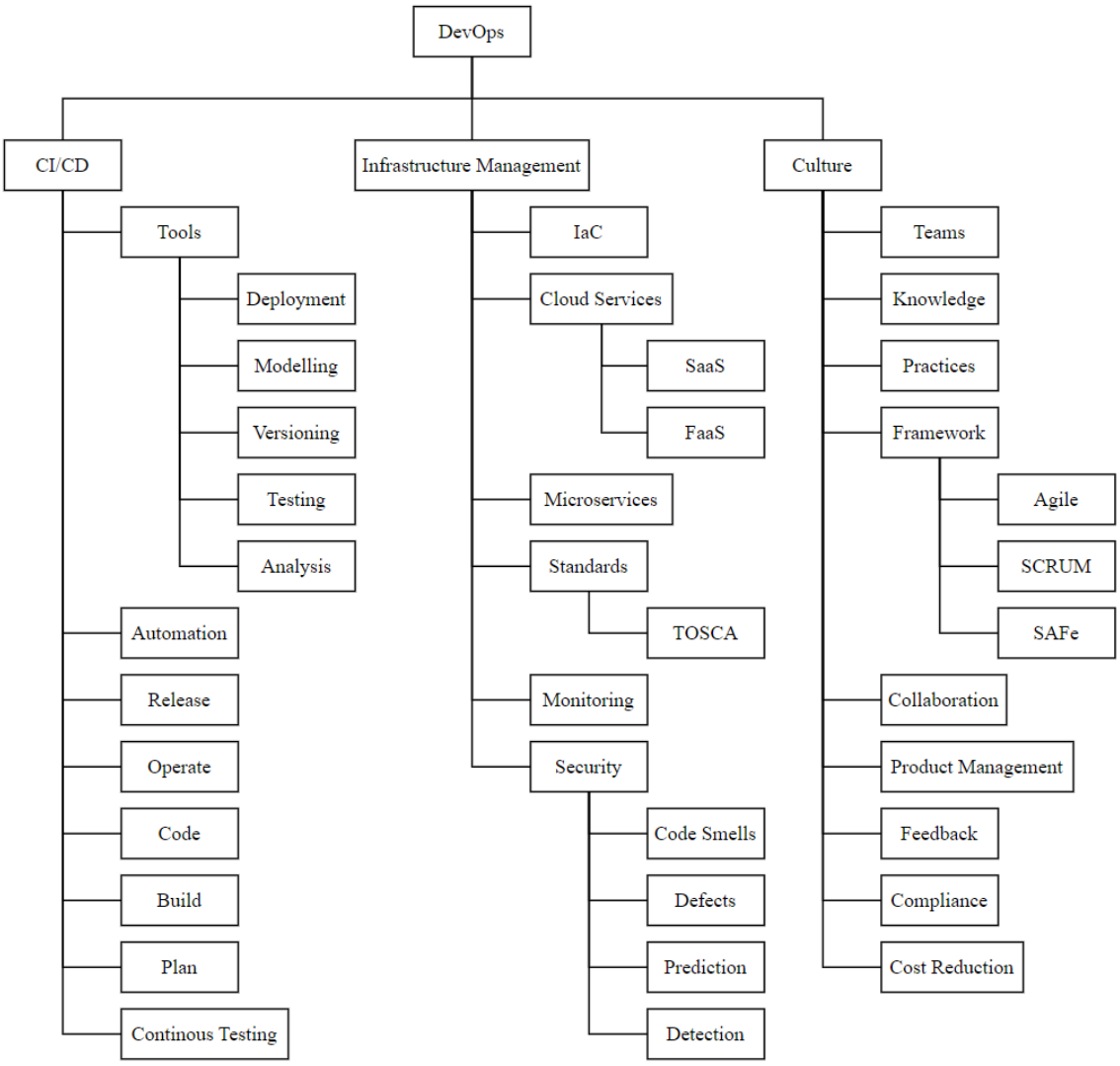

Figure 4.4: DevOps taxonomy and key concepts.

Some taxonomies [11, 64, 87] exist, but they cover only small subsets of the whole DevOps culture: indeed, we can consider our proposal as a grounding for all the others, since they can be appended to our as they provide a more in-depth overview on some specific topics, such as TOSCA or Tools in general. We want to highlight that the taxonomy does not show the relationship between keywords, just their hierarchy. A nice example of the relationships that exist between the different concepts has been proposed by [11, 64].

We identified three main aspects that define DevOps: Continuous Integration/Continuous Deployment (CI/CD), Infrastructure Management and Culture; each of them cover a face of the DevOps working framework.

CI/CD comprises all the aspects which are proper of this modern way of developing software, spreading from Tools, Automation, Release, Code, and all the other activities

related also to traditional software development. CI/CD represents the core of DevOps, as it represent the what in practice has to be done to build software in a way that does not require relying on traditional project deadlines and developing stages. However, the means and the environment in which CI/CD is used are also crucial for its applicability: Infrastructure Management and Culture cover these aspects, respectively. We want to stress that each of the three main categories is not to be considered as independent, but each of them relies on each other. DevOps is a complex combination of many different practices, and it shows its full potential only when it is applied to every level of software development and operations. Also, it is actually difficult to isolate keyword in the taxonomy: in practice, most of them overlap as the cover many topics in a transversal fashion. The same holds for the paper we found: as we will discuss later, most of them usually cover many topics spreading around the three main categories. This shows how DevOps is actually a multi-faced discipline which involves a lot of different resources.

Infrastructure Management represent the practical means through which software is shaped in a DevOps environment. The most relevant keywords we identify are Infrastructure as Code, Cloud Services, Microservices, Standards (where TOSCA falls in), Monitoring and Security. The meaning of these keywords will become clearer in the next sub-section, where we discuss the results. However, we want to highlight two aspects. First, IaC does not represent a simple keyword, but we consider it as a pillar component of DevOps. More and more companies are using this way of programming [47], and it also represents one of the core topics we investigate in our review. Second, Security is a very broad topic: we identified four sub-topics in it, which define how security is related not only to the security of the environment in which software is developed, but also to bad practices, code smells, defects in code which is necessary to detect and, even better, predict, in order to build even better code. The final aspect we want to underline is standardization: this is another core topic we investigate in our research, and in the past recent years there has been an ever-increasing interest in this topic, in particular for the TOSCA-related standards.

Finally, with Culture we identify all the aspects related to the people, both developers and operators, involved in a DevOps environment. So, we consider as fundamental concepts Teams, Knowledge (sharing), Practices, Frameworks (for which we identified Agile, SCRUM and SAFe [52, 98, 99], but many others exist), Collaboration (within the team and between teams), Product Management (with respect to the traditional way of working, i.e., Project Management), Feedback (strictly related to, for instance, the Agile framework), Compliance (w.r.t. regulations defined in a DevOps team) and Cost Reduction. Indeed, the greatest benefit of DevOps is, from a business perspective, cost

reduction [46, 64].

4.4.2. Classification

With this taxonomy in mind, we classified the papers found according to it. However, the taxonomy is related to the DevOps topics, not to the type of paper: with type of paper we identify what is the contribution of the paper at a scientific level or how the study has been conducted. The different categories of papers are: Survey, SLR, Case Study, Comparison, Interviews, Technical Insight, Challenges. In particular, survey and SLR are very similar, and in order to classify papers in these two categories, although we relied on the definitions of the two, we also followed the definitions provided by paper's authors themselves: if a publication is explicitly presented as a survey, we classified it as a survey; instead, for instance, if a paper is presented as a SLR but it does not follow any particular scientific rigor, then we preferred to classify it as a survey. Case Study represents the category of papers that bring practical applications of theoretical concepts, while Interviews is the category of publications which core resides in interviews reports. Challenges is a type of papers which tackles challenges related to particular topics and it can be explicitly stated by the authors or inferred by the content of the publication. Finally, Technical insight represents the broader category and, indeed, is the one where most of the papers fall (as we will see later in this section): this category identifies papers that shed light on, usually, a particular topic providing new ideas, models, approaches or tools; they consist of actual scientific investigations, as they make use of the scientific method to inquire over particular topics.

Combining these two parameters, i.e., the taxonomy and paper type, we built a classification matrix, where each paper can occupy one or more fields: in fact, a paper might bring on or more contributions and at the same time it might tackle one or more topics depicted in the taxonomy. However, in the classification we only considered the main topic(s) faced by a paper: it might happen that a paper also covers a little of other topics.

In Table 4.5 it is depicted how papers have been classified, according to the types of publications explained in the previous paragraphs and the taxonomy in Fig. 4.4. The main challenge of this step was not only to identify the exact publication type, but also the main topics covered, that is, the one explicitly addressed by the papers, typically explained in the abstract and introductory part with research questions and scope. However, many papers also tackle a lot of topics since, in general, DevOps is a broad discipline with many interconnected areas. For these reasons, the vast majority of papers have been classified in more than one category: this was our initial intent, as we envisioned the table as a way

			SLR	Survey	Case Study	Comparison	Interviews	Technical Insight	Challenges
CI/CD				[41], [50], [64]		[121]	[41]	[29], [52], [77]	[41], [64], [55]
	Tools		[90], [110], [85]	[22], [19], [44]	[19]	[95], [90], [110], [6], [22], [86], [93]		[30], [29], [34], [98], [82], [77], [28], [33], [109], [101], [63], [27], [3]	
		Deployment						[21], [29], [98], [77]	[55]
		Modelling				[7]		[21], [29], [77], [33], [109], [14]	
		Versioning		[64]				[29], [77]	[64]
		Testing						[29], [77]	[48]
		Analysis						[29], [78], [82], [77], [101], [26]	
	Automation							[30], [80], [20], [33], [99], [6]	
	Release							[48]	
	Operate		[65]				[65]	[29], [52], [97], [63]	
	Code							[29], [60]	
	Build							[63], [60]	
	Plan							[52], [97]	
	Continuous Testing		[85]					[29]	[48]
Infrastructure Management									
	IaC		[85], [75], [62]	[110], [22], [88]	[104], [102], [56]	[104], [7], [96], [22], [75], [86]	[47]	[84],[21], [25], [76], [30], [34], [80], [20], [98], [82], [92], [77], [28], [33], [68], [14], [83], [60], [27], [68]	[96], [47], [87], [48], [26]
	Cloud Services			[50]				[57]	
		SaaS		[50]		[6]		[69], [6]	
		FaaS	[65], [46]	[42], [50]		[42], [95]	[65]		
	Microservices			[50]					
	Standards							[92]	
		TOSCA		[11]	[68]	[7]		[21], [29], [25], [20], [68], [63], [14], [109], [118]	[25], [34], [78]
	Monitoring							[57]	
	Security			[88]		[86]		[76], [80], [92], [83]	[25]
		Code Smells		[88]		[86]		[84], [25], [78], [82], [92], [83]	
		Defects		[88]				[84], [82], [28], [121]	[87]
		Prediction						[78], [82], [28]	
		Detection						[78], [80], [92]	
Culture			[85]	[64]				[29], [99]	[64]
	Teams								
	Knowledge		[85]					[52], [99]	
	Practices		[85], [62]	[41]		[93]		[29], [52], [76], [78], [121], [81], [57]	[48]
	Framework		[85]	[44]	[68]			[52], [20], [99], [68], [97], [83], [3]	
		Agile							
		SCRUM						[52]	
		SAFe							
	Collaboration							[29], [109], [99], [97]	
	Product Management							[81]	
	Feedback							[99]	
	Compliance							[109]	
	Cost Reduction							[69], [101]	

Table 4.5: Papers classification according to publication types and topics.

to find papers that cover particular topics, in general.

Most of the publications found are Technical Insights, followed by Comparisons and SLRs. Indeed, most of the paper present new techniques or tools, explaining how they work and most of the time they also provide an application example. However, since we could not classify it as a proper case study, these kinds of papers they do not fall in the Case Study category, although they bring practical insights: in fact, the papers that have been classified as Case Study are usually just that, or sometimes are very long papers that provide some Comparisons between tools and have a very large chapter for a case study presentation.

The least common type of paper is the one that provide only or for most of its content interviews reports, and usually they are related to challenges regarding some DevOps topics. The most common topic, instead, is IaC: the reason why it is much more popular than the others is that not only the research query used was explicitly looking for IaC-related papers, but also because IaC is a very general keyword, and almost every paper found covers it, in more or less depth. This is an expected result, as it proves the effectiveness of our query. In addition to this, TOSCA is a very common topic with papers that tackles it usually also fall in the monitoring topic, as TOSCA is strictly related to modeling tools. In general, The tools category is very well populated by all the different kinds of papers: this holds because not only it is another general topic, but also because we explicitly looked for tools keywords in our query.

Some categories, usually very specific ones such as Agile and Plan, are not populated at all or very few papers are present: this is due to the fact that these categories are at the very limit of the scope of our research, and for these reason very few papers about them have been considered. This implies that Table 4.5 should not be intended as a picture of DevOps literature of the past recent years, but rather how our results fit in the whole DevOps taxonomy. This become even clearer if we take a look at the graph shown in Figure 4.5, as they help us visualize the focus of our research over the DevOps spectrum. In general, we consider Table 4.5 as a proof of the effectiveness and precision of our query, as it shows the results we expected to obtain, and also reflects the connection between topics in the DevOps fields, as the same paper usually appears in at least two of the three main categories: CI/CD, Infrastructure Management and Culture.

Figure 4.5: Percentage of topic share among all the papers found.

Graph in Fig. 4.5, as previously mentioned, describes to which extent, given the DevOps taxonomy presented above, each topic has been tackled by the papers we found. Since a paper might have been assigned to more than one category, the chart represents the share of interest a particular topic has among all the papers found: the total, i.e., 100%, do not correspond to 62 (the number of papers found); this is correct, since the graph should be read not by the point of view of a paper, but from the perspective of each keyword. For instance, the 43% of papers tackles, in general, Infrastructure Management, and this implies that 57% do not mention it. The same holds for all the other categories. This helps us visualize how much interest a specific topic has in the research field, and which ones need more attention. Here we just present the graph, so that the reader could get a better idea of how the papers we found are in relation with the keywords, but a deeper

discussion about this will follow in the next section.

With respect to Fig. 4.3, where we show the most prolific authors, we also want to point out how they are related to each other and also to the keywords of our taxonomy. To help us visualize this, Fig. 4.6 shows a graph where the orange nodes are the publications - identified by (part of) the title - and the blue nodes are the authors. It is possible to recognize four main clusters.

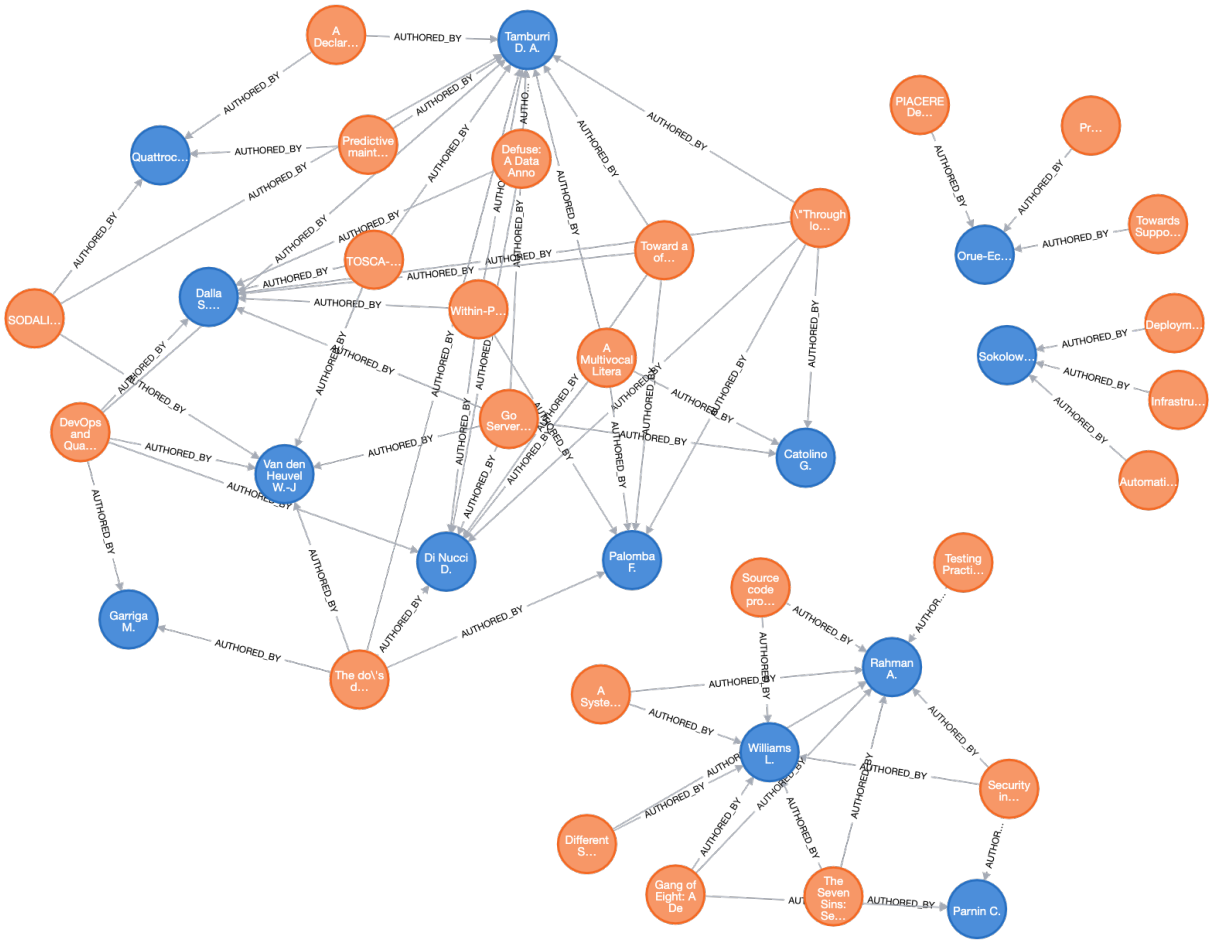

Figure 4.6: Top authors with relationship among publications and other authors.

The size of each cluster perfectly follows the distribution of papers among the different topics, even if it is a small set of authors and publications: indeed, the larger cluster is the one where the most prolific authors, such as Tamburri and Di Nucci, and it is composed by papers related to Tools, IaC and TOSCA, which are the most popular topics, as we can see by looking to the outer shell of the graph in Fig. 4.5. Then, follows the second largest cluster that represent the authors Williams, Rahman A. and Parrin working together in the Security and Code Smells fields, which are also very popular. The last two clusters

at the top-right corner represent instead other author with their publications related to, again, Tools, Security and IaC. Of course, this graph does not represent the whole result set, but it shows us from a different perspective our initial considerations, and also how authors worked together in many different studies.

4.5. Results discussion

In this section we discuss the obtained results, by bringing examples of papers belonging to each of the most popular categories of topics in the DevOps world and also by making some considerations on the state of the research in these fields.

4.5.1. CI/CD

For what concerns the big topic of CI/CD, we begin with two important papers [41, 64] that we also classified as milestones in our previous sections. Leite et al. [64] bring us a very nice study, well documented and well-structured, were they investigate the DevOps challenges from very different perspective, both from researchers and engineers. Most importantly, they propose a DevOps conceptual map, which was very useful to build or taxonomy. Elazhary at al. [41] identify ten practices continuous integration practices, turning on both practitioners and researchers, showing how these practices could be beneficial in project contexts, discussing also the challenges related to the differences on how CI implementations vary.

Della Palma et al. [29] propose a novel tool, namely RADON, to support the different phases of the DevOps life-cycle, depicting the model and showing its effectiveness in the various DevOps steps. This particular paper, and in general all the ones related to this tool, will pop-up in many different topics since they cover much of the DevOps culture. Similarly, to RADON, the PIACERE project [77] covers the implementation, deployment and operation of Infrastructure as Code.

Hassan at al. [50] propose a very large systematic literature review on server-less computing, counting over 270 papers. They cover a broad field, since they answer several research questions regarding concepts, platforms and usage of server-less computing. Hence, they also tackle many topics related to CI/CD, in general, providing also a nice taxonomy in this field.

In their research, Zampetti et al. [121] provide a very nice overview of the whole CI/CD process, with particular attention to code smells and bad practices. Indeed, their research spans over many different topics, especially related to security and culture: their study

is very broad, and they not only make a survey on these concepts, but they also conduct several interviews with processional developers, in order to build a catalog of CI bad smells, with 79 individual smells divided in 7 categories. Another paper that discusses challenges related to CI/CD and deployment of large applications in the DevOps environment is the one of Jiménez et al. [55]: however, they characterize more general challenges with respect to Zampetti, since they cover topics such as needs for deployment notations and tool support for deployment and design.

Finally, another paper that tackles CI/CD in general is the one authored by Helwani et al. [70] they provide a framework, which we actually classified as belonging to the Culture keyword in our taxonomy, to support the CI/CD process: this framework is an extension of the well-known SCRUM methodology.

These were the publication found related to CI/CD in general, but also to other topics. Now we will briefly depict the publications related to the CI/CD subtopics, highlighting the most relevant ones:

- **Tools**. Rahaman et al., in the study that we classified as milestone [85], they make an SLR to four main IaC-related topics, and among these fall tools for IaC. this study has been used as basis for our research, since it covers most of the literature until 2018. They also make important considerations on which are the most relevant research areas of IaC in general. Similarly, another smaller study [22] brings us a survey regarding Tools, with particular attention for the ones to perform static analysis of IaC. Gokarna et al. [44] provide an historical review of DevOps, shedding lights on some popular tools. Although focusing on MLOps tools, we also included a study [90] were a multi-vocal literature review is performed, tackling MLOps stools which, however, have been proved to be very effective in the CI/CD process, since CI/CD is at the basis of both DevOps and MLOps, considering also that the second discipline is the most recent derivation from DevOps. Tools to support CI/CD are also listed and discussed in another survey, by Teppan et al. [110], as also by Buttar et al. [19] who, however, focus on cost reduction and optimization. Another study [6], where it is proposed a method supported by a tool capable of dynamically orchestrate a cloud environment, reducing costs, focuses on cost reduction. The same does Splice [101], a tool capable of reducing costs of cloud resources, by using Neural Networks. Finally, Fireworks [95] is a tool-framework developed to cope with long start-up times in the cloud environment, security risks and memory efficiency, the main goal of reducing costs and run-time environment usage. The "Muse" tool is proposed by Sokolowski [98], which is used to help not only the CI/CD process, but also the collaboration between teams, since, as claimed

by the author, in order to speed-up the CI/CD process it is necessary to decentralize deployment coordination: the proposed tool helps in this direction as Dalvi [30] with its approach. In general, tools to support IaC have usually been adopting one of the following strategies: they can be either model-driven or code-centric. Sandobalin et al. [93] compare the two approaches. They prove that the model-driven approach, is more effective than the code-centric one. The RADON tool is proposed and discussed by Dalla Palama et al. [27–29]. this is actually a framework, composed of an integrated methodology - hence covering also the culture part of the DevOps discipline - and a tool-chain to support the design, development and deployment of serverless function used in data pipelines. TOSCAdata is proposed by Dehury et al. [33], which is a TOSCA standard extension and also a RADON extension that focuses on modeling data pipeline-based cloud applications. Another tool used for similar purposes is SODALITE [63], which focuses more on IaC for cloud-edge resource modeling. Similarly, The PIACERE project [3, 77] provides a set of tools, methods and techniques for IaC, with particular interest in quality and security of infrastructural code (more on that in the security section). Still related to security, we cite also Rahman et al. [86], a paper that we classified as a milestone, were in given particular attention to security smells and tools used to detect them. Quattrocchi et al. [82] worked on a similar topic, since they explored defects in IaC scripts to build defect prediction models (not proper tools, but still an artifact that can be used directly on IaC scripts) to improve state-of-the-art defect prediction tools. MiCADO is proposed by DesLauriers et al. [34] as a tool capable of generating IaC from metadata. Another way of modeling IaC, i.e.e, TOSCA-based intent modeling, is proposed by Tamburri et al [109], as they investigate the novel modeling approach by producing a simple industrial tool featuring the TOSCA language. Intent-based modeling is also tackled by another paper [68], which however does not present a tool but rather a proof of concept.

- **Deployment**. Calcaterra et al [21] propose a methodology which involves TOSCA, BPMN and some tools capable of automatically generate holistic management workflows, which spans from modeling to deployment. We have already mentioned some studies that fall in this category, such as the RADON tool [29], PIACERE Project [77], the Dynamic IaC solution [98], which augments static IaC programs for dynamic deployments, and the analysis of deployment specification challenges [55].
- **Modeling**. Quattrocchi et al. [7] propose a novel declarative modeling framework, particularly directed towards blockchain applications. Another paper

that falls in this category is the one of Brabra et al. [14], which tackles modeling concepts related to orchestration for cloud resources. Again, RADON and PIACERE [29, 33, 77] are mentioned in this category since, as we already stated, are tools that covers multiple steps of the CI/CD lifecycle. The same hold for the methodology proposed by Caltaterra et al. [21], which focuses mostly on deployment and modeling based on TOSCA standard, and TOSCA/based intent modeling proposed by Tamburri et al. [109].

  - **Versioning**. RADON and PIACERE [29, 77] are covering also this area, since they can be beneficial also in the versioning process. Within the challenges investigated by Leite et al. [64] there are also some interesting insights on versioning.

  - **Testing**. In this category we can find a very well documented study performed by Hasan et al. [48] , which identifies six best practices for testing infrastructural code, in addition to the previously mentioned tools: RADON and PIACERE.

  - **Analysis**. Large project such as RADON and PIACERE fall again is this category, as they provide some tools and functionalities able to perform a certain level of analysis, by detecting and predicting the presence of code smells, similar to the defect prediction model proposed by Quattrocchi et al. [82]. In this category also falls Defuse [78], an extension integrated in the RADON IDE capable of predict software defects. Indeed, we also include in this category another study [26], which even tough does not discuss or propose a novel tool, it explains the concepts at the basis of RADON and Defuse, and for this reason we think it is useful to include it in this section, as it deepens the explanation of important concepts. These were tools to be applied to infrastructural code, while Splice [101] is a different concept, since it is a tool meant to analyze resource usage and reduce cloud costs.

- **Automation**. The TOSCAdata approach [33] takes a further step towards automation, since it is capable to produce infrastructural code from metadata, reducing the human effort to develop IaC scripts. A new Python-based tool is proposed by Petrovic et al. [80], which enable the automation of static code analysis to achieve quality and security in IaC scripts, with a web-based interface. Calcaterra et al. [20] propose a framework similar to the one of Yussupov et a. [118] described as approach based on BPMN modeling language and TOSCA standard to automatically generate management workflows in the cloud, demonstrating the viability of

the solution. The already mentioned "Muse" [99] improves deployment automation by decentralizing the coordination in IaC projects between teams, developers and resources. As we discussed above, a similar system have been proposed [30], which focuses on the automation of CCoE requests by introducing a self-service mechanism. Finally, the Neural Network-based system proposed by Bahadori improves the automatic orchestration system present on the Amazon Web Services (AWS) for cloud resources.

- **Release**. The only paper found that explicitly tackles this topic is the previously mentioned "Testing Practices for Infrastructure as Code" by Hasan et al. [48] which presents a set of good testing practices and, as a consequence, better quality IaC releases.

- **Operate**. SODALITE [63] and RADON [29] are also present in this topic since, overall, they support IaC script development operation by not only providing tools but also a methodology and a culture. ACIA [52] is a methodology which, even if mainly focuses on extending the Agile framework and, in particular, SCRUM, brings also support to operations by identifying design patterns in Continuous Integration. Also, Sokolowski [97] proposes a tool that enables truly independent operations in DevOps teams. The Operate topic is also tackled by Leitner et al. [65], as they conduct a survey about serverless computing, covering not only frameworks and collaboration among teams, but operations in general.

- **Code**. Related to this topic, we point out the work presented by Kokuryo et al. [60], where they investigate the usages of the imperative modules in an IaC configuration management tool. Furthermore, we mention, again, RADON [29], as it faces directly coding challenges in infrastructural code.

- **Build**. Build topic is also covered by the aforementioned study [60], as it is investigated not only why are imperative modules used, but also how they are used. Finally, SODALITE [63] project partly covers Build of IaC scripts.

- **Plan**. We include here the study conducted by Helwani et al. [52], as ACIA methodology directly involves planning during the continuous integration process. Similarly, the Muse tool by Sokolowski [97] involves planning since it helps in the coordination among DevOps teams.

- **Continuous Testing**. The extensive survey published in 2019 [85], previously classified as a milestone, covers a variety of topics in the IaC fields, in particular tools and challenges. However, it also gives interesting insights on the state-of-the-art of

tools and techniques for continuous testing, commenting on novel frameworks developed in the past recent years. Similarly, another study [48] discusses the challenges and the practices related to infrastructural code testing: it brings a set of six - continuous - testing practices, applicable to IaC but also to general purpose scripts.

4.5.2. Infrastructure Management

As mentioned in the previous sections, Infrastructure Management keyword identifies a variety of concepts that are used in the DevOps discipline to support the implementation of services and applications. In this field, we recognize IaC as the most relevant concept, as it is also the one with the highest number of papers. However, Many paper that we classified within the IaC keyword usually are not just related to this general concepts, but also to more specific ones related not only to infrastructure management but also to CI/CD and Culture. Among the other concepts that fall in this category we can find Cloud Services, such as Software as a Service (SaaS) and Function as a Services (FaaS), which are crucial in today's world, where companies are moving to the cloud adopting services such as AWS. On the same page are microservices, as they are a key element at the basis of the Serverless way of working, were, as we will see, it is strongly suggested developing small pieces of code, namely serverless functions, which allows the developers to adopt IaC and CI/CD. Also, Security is a very important topic in this field - with its sub-categories such as Code Smells, Defects, Prediction and Detection - as there has been, in the past recent years, a strong research path towards this topic, considering that bad coding practices tend to develop unstable IaC scripts, which, as a consequence, produce less reliable application: for this reasons, tools and techniques have been developed to detect as soon as possible defects in code, and predict the presence of them. In this field, TOSCA standard is of paramount importance, as it has been used in many different applications - for instance, it is at the basis of extensive project such as RADON IDE and SODALITE - thanks to its versatility and applicability to IaC field. Monitoring is the final topic on our list: unfortunately, no paper has been found directly tackling monitoring; nonetheless, monitoring has been mentioned by some papers, especially the previously mentioned ones that cover tools and techniques for cost-reduction. This does not mean that the topic is irrelevant but, actually, the opposite: research in this direction should be improved, since in the Serverless setting monitoring is crucial but, unfortunately, it is one of the last thoughts when building cloud applications, as we will see in the next chapter.

- **IaC**. Many of the papers we cited in the CI/CD section also fall in this category. To avoid being repetitive, here we just report the new ones, and among the ones we already cited only the ones that have IaC as the main topic, such as survey or

presentations of tools to support IaC development. We begin by mentioning a study [96] which compares the traditional way of developing code with infrastructure as code, shedding light over the most common challenges that rise when adopting the IaC approach. Another practical insight is given by Sorour et al. [102] as they apply IaC and the DevOps discipline to demonstrate the level of automation that it is possible to reach with these approaches. Two case studies are also discussed by Souza et al. [104] in order to prove how infrastructure as code is improving the maturity level of DevOps both in the Communications industry and Health-Care. A new way of developing IaC code is proposed by Mascarenhas et al. [68], which discuss a proof of concept for Int2IT, an intent-based TOSCA infrastructure management platform based on concepts similar to the ones discussed by Tamburri et al. [109]. A systematic literature review has been conducted by Nedeltcheva et al. [75], were they explore IaC modeling tools, comparing different solutions and assessing which one is the best for different scenarios, identifying research areas that stills need to be further inquired. Another SLR is brought by Kumara et al. [62], which investigates the best and worst practices to - not - apply to IaC development, producing a taxonomy and practical considerations. Particular attention to Security smells in Iac scripts is given by Rahman et al. [88] in a paper devoted to practitioners, in order to help them to avoid insecure practices while developing IaC scripts. Several interviews have been conducted by Guerriero et al. [47] to investigate the challenges of adopting IaC in the industry, discussing the advantages and the disadvantages for practitioners dealing with this concept. We cite also here, although the focus of the paper is on Code Smells, the research conducted by Rahman et al. [84] were they analyze repositories to classify the most common security smells in IaC. Also Dalla Palma et al. [25] cover the topic of Code Smells and Bad practices, discussing Machine Learning algorithms and rules to detect them in TOSCA compliant scripts; similarly,Saavedra et al. [92] developed GLITCH, which is a tool that exploits the power of Machine Learning to detect security smells in IaC. Another paper [76] proposes an architecture and a method to assess architecture conformance according to security-related practices. Covering the topic of IaC there also another extensive study by Rahman et al. [83] which is at the basis of their other work [84], as they analyze infrastructure as code scripts to identify a set of ten source code properties that correlate with defective scripts. Similarly, in a later work [87] Rahman et al. they categorize defects in IaC scripts, identifying eight categories of them, producing also an extensive analysis on their frequency and consequences. Related to security we can also find the inspection script presented by Petrovic et al. [80], capable of automating the static analysis of IaC scripts, in particular, Terraform scripts. Of

course, this kind of papers will be discussed later in the security-related categories. The RADON approach is introduced by Dalla Palma et al. [27], that is the methodology at the basis of the RADON project based on several tools which make use of Machine Learning techniques, such as the decomposition tool, the defect prediction tool, and the testing tool. Among the papers we cited in the previous sections, we recall the SLR by Teppan et al. [110], where they explore the state-of-the-art for IaC tools and solutions; we mention also the extensive study conducted by Rahman et al. [85] that covers a variety of DevOps topics, in particular the ones related to the current status of IaC research. Chiari et al. [22] instead go deeper in the same direction, by focusing on the most relevant tools for static analysis of IaC. To conclude this category, we mention a paper that discusses [48] the challenges of related to IaC testing, identifying a total of six best practices overcoming them.

- **Cloud Services**. An important contribution in this field is given by Hassan et al. survey on serverless computing [50], as they cover a variety of topics which related to cloud services; in particular they make a review on the state-of-the-art of serverless computing, comparing platform and tools but also analyzing the benefits and issues of this topic. A piratical insight is given by Keskin et al. [57], where it is proposed an extension for the Cloud Monitor service, which is a bench-marking tool for clouds, in order to make it real-time adaptive.
  - **SaaS**. The survey on serverless computing [50] we just cited directly covers also Software as a Service, which is a key concept of Cloud Services. However, we want to focus here on another paper [6] that which highlights the importance of Machine Learning in SaaS field -and, in general, in the Cloud - as it enable the deployment of sound and effective solutions: in particular, they discuss an experimental implementation of an Elastic Container Services (ECS) hosted on AWS, enriched with the ML dynamic orchestration system. Another study [69], instead, inspects the costs of services provided by AWS: AWS StepFunctions; they make an extensive study where they discuss the best way of using StepFunctions in order to reduce costs.
  - **FaaS**. Among this topic we can find several papers which typically provide comparisons and surveys regarding tools or services. For instance, Leitner et al. [65] investigate the serverless state-of-the-art research with an extensive SLR, with particular attention for Function-as-a-Services, discussing when it is used, which are the benefits and the drawbacks, matched by a set of interviews. On the same level we can find another survey [46] that precisely investigates the current research status of Function-as-a-Service, by discussing

not only the challenges of this technique but also which are the current tools available, comparing them with regard to costs and performance. A survey is also proposed by Eskandani et al. [42] which gives us another point of view over the state-of-the-art of FaaS services. Another work, by Shin et al. [95], explores a new tool, namely Fireworks, capable of optimizing FaaS resource utilization and reducing execution time and thus costs.

- **Microservices**. The only paper found which explicitly mention Microservices is the one titled "Survey on Serverless Computing" [50], which mentions the current status of the research among the microservices topic. However, this does not mean that Microservices are not relevant in the DevOps discipline: actually, the are of paramount importance, especially when we deal with IaC; indeed, this topic is indirectly mentioned by most of the paper we previously cited, although it is not the core topic of these papers.
- **Standards & TOSCA**. For what concerns Standards, we will mention only TOSCA: this happens because in literature, recently, there have been a great attention for standardization, and TOSCA has raised as the most important one. Actually, we can find a lot of de-facto standards but TOSCA is the only one that has been developed explicitly with standardization in mind. We introduce this topic by citing the paper we classified as a milestone [11], that is a survey about TOSCA, covering many aspects surrounding this topics, such as tools and applications, producing a taxonomy used to categorize the papers analyzed in the survey. TOSCA has been used in the past recent years at the basis of different tools, frameworks and approaches that, in the end, have been used as a basis for the current state-of-the-art TOSCA-based frameworks, such as RADON IDE [29, 78] and SODALITE [63]. We already mentioned this tools, but we want to underline how the use similar techniques to use and adapt the TOSCA standard to the IaC scripts development application. Intent-based TOSCA modeling is a novel concepts studied by two papers [68, 109], where in the latter one the authors, belonging to the TOSCA technical committee, propose the new way of producing automated scripts, with the new approach that represents the evolution of goal-based modelling; the former one, instead, produced a proof of concepts of the theoretical concepts explained in the other paper. An in-dept analysis regarding code smells and bad practices in TOSCA compliant scripts is conducted by [25], where they also propose Machine Learning algorithms and rules to detect this kind of issues as earlier as possible. In general, paper related to TOSCA topic cover the importance of finding ways to automatically generate IaC scripts. For instance, we mention again TOSCAdata, which is a RADON extension

providing TOSCA models to ease the design technology-specific cloud applications. On the other hand, a 2022 publication [34] suggests a new methodology to generate IaC scripts by metadata, using the TOSCA standard. Based on TOSCA, a new model-driven orchestration approach is proposed by Brabra et al. [14] which leverages TOSCA to describe cloud resource artifact and translate them into technology specific artifacts. Two other papers [20, 118] produce, instead, a similar framework, composed by a set of tools and techniques, to automatically generate application management workflows with BPMN and TOSCA standards, which are at the basis of RADON IDE. Finally, a case study [7], already mentioned, applies TOSCA based modeling approach to blockchain applications.

- **Monitoring**. The only paper [57] that explicitly faces monitoring issues has been already mentioned, as it extends the capabilities of an already existing tool, Cloud Monitor, to enhance its capabilities to Real-Time scenarios.

- **Security**. In their extensive study, Rahman et al. [88] deeply analyze the security risks in IaC scripts by listing the most common ones and creating a tool, namely SLAC, capable of detect them. The same author, in collaboration with other ones, explores the same topic in a more general manner in another study [86], by making a comparison of the most common smells: in particular, they identify a set of seven common security smells. Rahman et al. also investigate these issues in another study [83], where they identify ten source code properties that correlate with defective scripts, but applying it to IaC scripts in general. The previously mentioned study by Dalla Palma et al. [25], similarly to the work conducted by Rahman in many of is studies, investigates the security-related issues - in particular, code smells -, with particular attention to the technology-agnostic standard, TOSCA. It is also presented a comparison between ML-based and metrics-based detectors for such security smells. An approach to measure conformance to security-related practices is proposed by Ntentos et al. [76], which is based on a set of metrics that are used in conjunction with some statistical methods, so that it is possible to assess with a high level of accuracy the quality of IaC scripts. We can also find few security-related tools - besides the ones we previously cited in other sections such as RADON IDE - such as GLITCH [92] and IaC Scan Runner [80].

  - **Code Smells**. The survey [88] and comparison [86] cited above, although covering security within IaC script in general, they are particularly directed towards security smells related issues; the same holds for the technical insight given by Dalla Palma et al. [25] where the challenges of facing security smells in IaC scripts are also discussed. Rahman et al. [25, 83], the most prolific author

in this field, as we mentioned before conducted some studies in this field, giving particular focus towards Security Smells in IaC scripts, producing taxonomies and discussing the implications of the presence of these smells in the code. DEFUSE [78] and GLITCH [92] are the two tools that we mentioned before in other categories, which find their application in the context of detecting and predicting code smells. Finally, an extensive study [82] regarding metrics to assess the quality of IaC scripts and applying them to improve the state-of-the-art defect prediction models.

- **Defects**. A classification of bad practices in proposed by Zampetti et al. [121], where are also explored the consequences of adopting such practices in IaC scripts development and the defects they introduce in code. However, a more comprehensive classification of defects is proposed by Rahman et al. [87], as they produce a taxonomy of eight defect categories as a consequence of the extensive qualitative analysis they perform over fourteen-hundred IaC scripts. The same author uses that study as the basis for another research [86], where similarly security smells are classified. We also mention here the research conducted by Quattrocchi et al. [82], as they cover a wide variety of defects present in IaC scripts. To conclude, we briefly mention another publication by Dalla Palma et al. [28], considering that they apply a set of metrics to predict the presence of defects in IaC; however, we will discuss this later in the next topics.

- **Prediction**. In this category, we can find three studies [28, 78, 82] that are conducted around the same authors - in particular, Tamburri - where they explore not only which are the most common smells, but also what are the best metrics that we can use to predict the presence of such smell in many IaC scripts. In the end, they produce a tool, namely DEFUSE, which is the part of the RADON IDE tool-set capable indeed to predict the presence of defect in IaC scripts.

- **Detection**. We conclude this section by discussing the last three papers. DEFUSE [78] is not only capable of predicting the presence of IaC defects, but also to detect it, and the same holds for GLITCH [92]. Finally, IaC Scan Runner [80] is meant to detect the presence of code smells in IaC code.

4.5.3. Culture

As discussed during the presentation of our taxonomy, Culture refers to all the elements related to the human aspect of the DevOps discipline, in particulars the ones that define the interaction among managers, developers and teams. It is a core aspect of a DevOps environment, and it affects, directly or indirectly, all the others topics discussed in the previous sections. We keep stressing to highlight how, in the DevOps discipline, interaction is a core element. This is due to the fact that the continuous interchange of feedback, knowledge, collaboration and practices are the very first elements that started the advent of the CI/CD practices and IaC methodology in the industry. Among the subtopics we can find here, the most populated are Practices and Framework. Of course, this does not mean that research in the other categories is not developed, or they are considered of low relevance, in particular, we could not find teams-related papers: instead, most of the topics in this area are out of the scope of our research. However, many tools we found strongly affect, in general, culture and the way of working in teams and, so, we will find some papers discussing tools also in the Practices and Framework topics.

Regarding Culture, in general, we can find four publications among our set. The first one the the SLR by Rahman et al. [85], that we already found in other topics such as Tools and IaC, since it covers many aspects of the DevOps discipline: in particular, it covers some tools and practices that are helpful to coordinate the work of people in this field. Another survey [64] discusses the challenges of many DevOps areas, such as CI/CD, but it also focuses on the challenges related to the collaboration among developers and operators. However, the most relevant paper we found in this are in the one by Sokolowski et al. [99], where they investigate how it is possible to reduce delays caused by slow coordination by teams fin a serverless environment, by suggesting a new decentralized tool capable of reducing the dependency on manual coordination - i.e., via email. phone, etc - between people. We conclude by mentioning the RADON approach [29]: we have already seen how this tool-set cover most of the topics we identified in the DevOps discipline, and it also covers the cultural aspect since it provides a sound and complete methodology for people to follow in order to reduce the effort while developing FaaS products.

- **Knowledge**. Among the research paper we found, few of them - partly - cover the Knowledge topic. ACIA [52] is a methodology introduced as a SCRUM extension, i.e., the evolution of the Agile Framework, which is capable of improving the sharing of knowledge among team members: in fact, with a set of small changes to the SCRUM method is it possible to reach a higher level of communication within the team when implementing new design solution, improving the tasks' templates and

the feedback on design solutions. We also mention "Muse" [99], already discussed in other sections, which enable automatic coordination across teams by leveraging decentralized communication, in a self-service manner, so that team members can ask for resources or services by expressing wishes and, if available, they are automatically assigned to them so that there is no need to wait for manual intervention. The previously mentioned extensive SLR conducted by Rahman et al. [85] also falls in this category, as it mentions the challenges related to the lack of knowledge when practitioners are introduced to IaC.

- **Practices**. In the Practices topic we can found a wide variety of paper, most of them already seen in the other categories, that discuss which are the good and the bad practices when developing IaC solutions. Practices are usually related to security, as if bad practices are adopted they have negative consequences in infrastructural code. Among these papers we have the two studies involving Rahamn et al. [48, 85], where in the former are discussed what the state-of-the-art good practices are when developing IaC, while in the latter great attention is given towards testing practices, showing how the adoption of good practices can avoid or mitigate the presence of defection in IaC scripts deployed in production environments. Three studies [29, 62, 78] involving Tamburri et al. directly cover this topic - although others are also indirectly connected to it - as the first one consist of an SLR where, given insights from industry, it is produced a catalog of best and bad practices to - not - adopt when developing IaC code, mostly deriving from practitioners in this field. The other two studies instead involve the RADON project, which embodies all the theoretical results regarding best and bad practices produced by many different authors in the past recent years. Two other publications [41, 121] discuss the most common practices adopted in the DevOps culture, in particular with respect to CI/CD for the first paper, where are also given interesting insights from industry, and to bad practices that generate defect in IaC scripts for the second one. A practical comparison on two practices, i.e., the adoption of model-driven or code-centric tools, is given by Sandobalin et al. [93]. They compare two tools, in order to provide empirical effectiveness of the two different approaches. Bad practices are investigated by Ntentos et al. [76], in particular the ones that as a consequence produce security issues in IaC deployments; furthermore, it is also presented a methodology, based on metrics, to assess whether or not a script follows good or bad practices. New practices are also presented for what concern Cloud Services by two studies [57, 81] to enhance automation when dealing with cloud architectures. The first study in particular proposes an evolution of Cloud Monitor

service, while the second publication proposes a set of strategies to improve the AWS Cloud adoption.

- **Framework**. Some relevant frameworks we cite here are PIACERE [3], Int2IT [68], RADON-related frameworks [20], Muse [97, 99] and ACIA [52]. We have already discusses most of them, as they cover several aspects of the DevOps discipline, although some of them will be also discusses in the remaining topics, as they are specific to particular aspects of the DevOps Culture. Frameworks are also discussed by some surveys and SLRs [44, 83, 85] which we have already previously seen, where they investigate the stet-of-the-art of tools and framework developed to support DevOps operations and development stages.
- **Collaboration**. Collaboration is a core investigation topic of Sokolowki et al., as they discuss in two papers [97, 99] what are the current limitation to teams collaboration in DevOps, and how it is possible to improve in this direction by proposing a novel, decentralized tool, Muse, meant to avoid manual coordination across teams. Also RADON [29] has some insights on collaboration, as in the methodology it is forseen the used of the tool by different team members, with specific assigned roles. Finally, Tamburri et al. [109] indirectly intimate collaboration in their new intent-based TOSCA modeling way of working.
- **Product Management**. Here we only cite a publication [81] where a set of stages are discussed, as they provide a guidance when managing product migration from on-premises solutions to AWS cloud.
- **Feedback**. In this section we briefly cite again the Muse tool [99] that, as we previously said, enhances communication and feedback among cross-functional teams, improving deployment automation.
- **Compliance**. Although this topic is not the main focus of our research, we mention a study by Tamburri et al. [109]. In this study it is presented a new way of working with IaC products, by declaring intents - goal - using TOSCA models, which are constructs capable of enhancing compliance to models and allowing to better describe and orchestrate cloud service applications.
- **Cost Reduction**. Cost reduction is a very important topic when we talk about DevOps and Serverless computing. Although out of the scope of our research, we found two papers covering this topic. One [101] proposes a novel platform, namely SPLICE, capable of improving performance and cost-reduction when blending multiple services in the Cloud environment. By introducing a small amount of overhead,

this ML-based platform claims to reach savings up to 30%. The other study [69] extensively explores the costs of AWS Step Functions, by discussing which are the factors that affect the raise in costs when adopting this technology.

4.5.4. Final Discussion: Answering To The Research Questions

In the previous sub-sections we discussed the contents of the papers we classified by category. That was a mandatory step in order to get sound and complete answers to our initial research questions.

Before doing that, however, we want to analyse what are the most investigated areas in the DevOps discipline, and which ones need to be further explored. At a first glance, our classification, with the related graphs and statistics, clearly depicts a scenario where some topics such as CI/CD, Tools, IaC, TOSCA and Practices are deeply investigated by researchers and scholars, while some other topics such as Microservices, Monitoring, Plan, Build are mostly never considered. However, this interpretation is misleading and does not take into account one crucial aspect.

The taxonomy made in the previous sections should be interpreted as something that holds and it is true anyways, also outside our research, as it is intended as being valid for the DevOps discipline independently of the paper we classified. Of course, it is still open to expansion, as we previously said, since it has been derived by the contents of the papers we found, and it is possible that few topics were never mentioned.

With this in mind, we can affirm that DevOps, in general, is a very hot topic. The increase amount of interest towards this discipline can be ascribed to the ever-increasing adoption of DevOps and its methodologies in the industry: indeed DevOps is not that recent invention if we think in terms of technology, as methodologies, tools, and techniques evolve every year more rapidly than before. However, its mass-scale adoption happened in the past recent years: this pushed again interest of researchers in the field, who focused the attention in specific sub-topics, such as IaC and Tools, to improve the current technology and promote even more its adoption. We can affirm that all the researches we found have in common one underlying concept, which is the following.

The DevOps culture relies on many different technologies and methodologies. Most of the fall under the umbrella of the word "Cloud", which covers, among the others, concepts like FaaS, SaaS, Serverless. These theoretical concepts can be found in the industry under an uncountable number of vendors, proprietary technologies and languages. This variety of instruments created, over the years, a divergence between technologies, as each vendor created more and more different ways of implementing a specific technology, increasing

even more the differences in an already fragmented world. Nowadays this has been seen by researchers as an issue that is necessary to fix or mitigate: developers, operators, managers and practitioners in general are covered by an immense amount of technologies, and this not only slows down development when new techniques are adopted, since they need to learn new practices from scratch, but also the heterogeneity causes an issue when, for instance, one company substitutes the adoption of one platform, sold by a vendor, to another one sold by another vendor. Indeed, the consequence of this issues is the so-called vendor lock-in: a threat to companies, since teams might remain bounded to a specific service, even if they want to abandon it, because changing to another one is too much expensive, not only in terms of money, but also time and resources, most of them used to train practitioner for the new technology.

From our investigation we state that, not only there is a solution to this problem from a theoretical perspective, but physical tools and practices already exists and have been developed to address the issues. There is one underlying concept among all the most advanced and promising solutions: standardization. In particular, the OASIS TOSCA standard, born to describe the topologies of cloud applications, has been applied to automate the development and deployment of IaC. It is present in most of the state-of-the-art tools, such as the ones we mentioned before, namely RADON IDE, SODALITE, PIACERE and other frameworks.

However, the discussion above only scratches the surface of a deeper rabbit hole: these were just the high-level issues, but for each step of DevOps practice many other have been found, and as much solutions have been proposed to solve them.

One of the main topics we found that gained a great interest in scientific literature is Code Smells, or in general, bad practices and security-related issues in IaC. Practitioners may unintentionally develop IaC scripts introducing security smells, which lead to security weaknesses that can be dangerous for the whole system, producing security breaches. This kind of security smells, but also bad practices, as stated by many papers we have cited in the previous sections, are mostly the same we can find in traditional code development; however, new type of smells have been found which were not present before due to the nature of IaC. Indeed, a lot of work has been made to collect, identify and classify the most common smells we can find in IaC scripts, to help practitioners gain awareness on the issue and provide them a reference they can use to avoid this kind of issue when developing IaC scripts. Taxonomies, challenges and guides have been written in this field, and they mostly converge to the same results, although we have to admit that most of the paper found involved the same few researchers, as one study is typically the evolution of the previous one. Many of the papers do not produce a practical guide that practitioners can

use to avoid this kind of issues, but other papers, where present new tools or frameworks, use their results to produce useful tool to automate the detection and prediction of these security smells.

Indeed, we found a lot of papers presenting tools and frameworks which not only help to detect and predict the code smells, such as Defuse, now a RADON IDE plug-in, but also tools capable of improving automation, reliability and, in general, quality in the whole IaC scripts development process. Besides independent tools developed by smalls teams or methodologies, algorithms and code created to address very specific issues, such as cost-reduction heuristics, defect detection or prediction - we have seen how ML and AI, in particular Deep Neural Networks, have been used in this field -, most of the tools we have seen are based on the TOSCA standard, and they are meant to support the development of IaC scripts by applying standards-based modeling.

We mentioned also how cost-reduction is a topic of paramount importance when dealing with serverless applications, and although very few papers have been found in this field - this was not a crucial topic for our research - we recognize how cost-reduction cannot be put aside, since DevOps main objective is to increase the Return of Investment by reducing development time and increasing software quality. For this reason, we found interesting how research are trying to improve methods to reduce costs, by means of Artificial Intelligence, which is capable of outperform the solutions provided by vendors. It is also worth mentioning that costs is a really hot topic for companies, and cloud services vendors usually do not provide clear ways of describing how costs are evaluated and managed.

The research topics we just mentioned gravitate around one other main topic, that is culture, which is also important to discuss since it indirectly touches all the other discipline involved in DevOps. In particular, we have seen how speed in developing serverless application is strictly dependent on the ability of teams and practitioners to coordinate with stakeholders and other teams. Indeed, this is the first are where DevOps found its application. For this reason, multiple studies we found tackle new ways for improving communication, coordination and planning, by introducing new methodologies, frameworks, or extending new ones. We can asses that feedback is a very important tool practitioners should use to improve in this field. Many tools support that, by automating the communication between teams - such as Muse - avoiding manual coordination, and other tools provide ways of works that have been proven to increase quality and throughput in software development.

We now give a clear answer to the research questions of this SLR:

- **RQ1.** *What are the current challenges DevOps discipline is facing?* We state that currently DevOps is facing challenges under many aspects, from Culture to Tools. Culture needs improvement in automating the communication among team, stakeholders and practitioners. Many tools and new ways of working have been recently developed, but the main issues remains their adoption, since companies are usually skeptical in adopting new, open source, tools. Also another challenge is related to IaC adoption and standardization: IaC is a very powerful technology; however, to gain some improvements by adopting it, is necessary to follow good practices. In this sense, standardization - namely TOSCA - can be of paramount help since it has been proven several times how applying standards to IaC development can be beneficial. However, this is a recent topic of research and the technology is still not mature enough to be adopted at industry level. The last challenge we want to mention is the way it the reduction of code smells, bad practices and security issues in the IaC field. IaC is well-known to companies and many of them adopt it but, unfortunately, issues that have been known for years in traditional software development have been found also in IaC. Detect code smells is of paramount importance in IaC scripts, since a faulty definition of the infrastructure can lead to catastrophic issues: tools have been and still are under development, but it is necessary to improve in this direction, since their efficacy needs to be proven also in the industry.
- **RQ2.** *What is the role of Infrastructure as Code in DevOps?* Infrastructure as Code in a quite novel but yet popular way of defining the infrastructure necessary to deploy serverless solutions. Indeed, its role is critical in serverless environments, or it is necessary even in on-premises solutions, when teams want to avoid to directly manage the underlying hardware that is necessary to run a specific application. However, it is able to abstract the underlying hardware infrastructure to code developers, allowing them to take every single piece of application, be it a function, a database, a VM, and declare how they need to be deployed: there is no need to specify what it is actually needed to run them, and for this reason a big chunk of work is removed in the development process. However, IaC is not only an abstraction, but it is actually the latest evolution of automation in CI/CD pipelines: in fact, with IaC it is possible to gain another level of automation, since it is possible to speed-up CI/CD pipelines by, every time, producing a new version of IaC scripts and deploy only the new changes completely automatically.
- **RQ3.** *How TOSCA standard can improve the development and adoption of Infrastructure as Code?* Many technologies exist to support IaC and this is an issue: they are too many. Although this is a good think since having many technologies

competing between each others leads to better products, this can be a problem for practitioners since, even in the same company, might need to learn more than one tool for doing the same thing. This is of course related to the fact that each technology adopts peculiar paradigms that might be missing in another technology. For this reason is of paramount importance to support reach for standardization in this field, since we have seen how TOSCA have been able to abstract constructs present in many different technologies. TOSCA standard is the most prominent in this field and it has been proven to provide great benefits in automating the development and deployment of serverless applications. TOSCA standard is at the basis of many projects we have mentioned, such as PIACERE, RADON and SODALITE. In these applications, TOSCA is the basis for complete automation of design, deployment, testing and versioning of IaC.

- **RQ4.** *What are the most recent tools developed to support standards-based IaC script development?* In our discussion we have seen that many different tools have been developed to support standards-based IaC development. They all rely on TOSCA, and they usually cover the whole development process, following the DevOps Culture. RADON, PIACERE and SODALITE are the most prominent in this field, since they are the most complete: they are born thanks to the support of European Union funding. They all provide a state-of-the-art tool-set to model, verify, simulate and monitor IaC solutions. The most relevant aspect of these tools is that they comprehensively help to solve most of the challenges we discussed in the previous sections, from security smell detection - since they make use of algorithms, typically ML-based, to detect and predict faulty scripts - but also they provide an abstraction to develop technology-agnostic models for IaC solutions without replacing the services offered by the most popular vendors, but enriching them.

4.6. Threats to Validity

In this section we discuss what are the threats to validity of our systematic literature review.

For what concerns the internal validity, several threats can be found which, however, we tried to mitigate as much as possible since we followed popular guidelines for SLRs. The main internal of our study resides in the way the DevOps taxonomy has been developed. As we previously said, before starting to make the taxonomy we looked at other papers, not necessarily related to DevOps, which developed some taxonomies in order to have a better clue on how to properly make one. Most of these papers followed an iterative

process where, considering the topics and keywords tackled by each publication found in the search process, continuously optimize the taxonomy, starting from basic concepts, in order to obtain, at each iteration, a more refined taxonomy. This is the process we followed. However, the main difference between the other studies is that in our case the taxonomy is valid also outside the SLR, in the sense that we tried to build an overall taxonomy of DevOps concepts, and then we tried to fit our findings to it: the reason why wanted to reason in this way is that we think that this taxonomy can be reused and extended by other publications. Indeed, the main problem of our taxonomy is that it might be incomplete. Also, other important threats to validity are the criteria used to include papers in the study: in fact, given the the study has been conducted by just one person, it is obvious that, even if we tried to stick to well-known guidelines, the subjective component cannot be neglected: some relevant papers might have been excluded. Furthermore, this applies also to the choice of snowballed papers, which choice suffers even more to the author's bias.

A threat to the external validity, still related to the fact that the SLR has been conducted by one person, is the choice of the search engine: Elsevier's Scopus. The choice of this engine was made because, among all the others, it seemed to be the easiest to use, and also it allows to export the results directly in a Comma-Separated Values (CSV) file without the need of external tools: this allowed us to speed-up the review process. The problem here is that the engine might have excluded some important references. Still related to this problem, the way research has been conducted, i.e., through the use of a research query, might have influenced the obtained results and important papers might have been excluded. As mentioned in the previous sections, the research query has been built in an iterative process, each time adding or removing keywords and modifying the query structure. Although each time we compared the results and checked the kind of paper were included in the results, we cannot be sure that the final query we used is the best one, but we can be sure that is the one that produced the best results set we could have worked on.

As we can see, although this study is not immune by threats, if we consider that the research has been conducted by one person, we can assert that it has been reached a reasonable level of objectivity.

4.7. Conclusions

In this chapter we presented a systematic literature review over a variety of topics, in particular addressed towards IaC, Standardization and TOSCA, covering, in general, a

wide spectrum of the DevOps culture. We produce a threefold contribution with our work: (a) a structured literature review of the most recent publications, addressing the lack of state-of-the-art reviews regarding tools and standards in the IaC field; (b) a taxonomy of the DevOps culture, which does not exist yet and helps to clarify the DevOps topics and how they interact between each other; (c) an extensive analysis of the current challenges of DevOps, with particular interest in Security, IaC, Tools and TOSCA. According to the statistics we made in our research, we think that the main topics we addressed will remain relevant for the near future in the research community, as they need further exploration and as it is witnessed by the authors of the papers we found. In addition, our systematic literature review helps to reduce the barrier for novice developers and researchers for entering the very complex and multi-faced DevOps field. Finally, our findings can be used as grounding for future studies, stimulating the research in IaC and standardization.

5 | Proof-of-Concept: The RADON methodology and the Semiconductors Industry

In the previous chapters we shed light on many topics related to the DevOps methodology, with particular interest in tools and standardization: within these two main research fields, the OASIS Topology and Orchestration Specification for Cloud Applications standard is the one that in the past recent years has gained a lot of attention by the research community. Based on this standard, many tools have been proposed to help practitioners develop IaC architectures in the various phases of the DevOps life-cycle, such as the RADON IDE, PIACERE Project, SODALITE Project and other mentioned in the SLR chapter. In particular, the RADON IDE project arouse our interest, as it provides a set of different tools, combined in a unique Integrated Development Environment, which are able to help developers in modeling, designing, deploying applications, but also to predict and detect code smells: it covers a wide range of topics, and for this reasons we want to investigate how it works, how easy it easy to use, what are its advantages and disadvantages in a real word scenario. We propose in this section a case study, were we apply the RADON IDE within a data producer team of a semiconductors company: NXP Semiconductors. Before diving in our case study, we want to make a short introduction to the semiconductor industry, and explain why we decided to chose that kind of company.

5.1. Semiconductor Industry Outlook

The semiconductor industry is one of the fastest growing markets which, in the past recent years, have seen companies facing an ever-increasing amount of demand for their products. Although this sounds like very good news for the industry, it is also source of many problems which are catalyzed by many different factors, both internal and externals to the industries themselves.

Chip shortages of the past recent years and the COVID-19 pandemic - that elicited even

more the demand - have worsened an already critical situation [8, 16, 17], leading to a deep crisis affecting the entire industry, not only semiconductor value chains - from wafer factories and chip design to PCB assembly - but also all the industries strictly dependent to chip supply; in particular, the automotive market is the most affected by these shortages [16]. Nonetheless, this situation was not caused only by COVID-19, which instead acted as a catalyst in an already compromised situation, but was made even worse also by the USA-China trade war, an intense competition within the industry itself, fabs operating at full capacity and lack of semiconductors [8, 16, 17]: these are just few of the causes of this crisis, and we are scratching the surface of a very complex problem.

Chip shortage has been caused also by structural deficiencies and bottlenecks in the semiconductor industry: there are not just external causes, but also internal ones [8, 18]. Technology leadership, long-term R&D, resilience, talent, ecosystem capabilities and greater capacity are the six critical areas [17] which suffer the most and that can be severely improved. In general, semiconductor companies, and in particular integrated device manufacturers, fall behind in each of these areas in comparison with other industries [17, 18]: talent acquisition and retention has been highlighted as the most critical area [17, 18].

All these problems are expected to get even worse, since it is expected to be a higher demand in the next years, considering also new incoming technologies, which will stress chip industry even more [10].

The solution to chip shortage would be to increase fabs output but, however, not all the companies would benefit from it since each of them is different and costs to built new plants are incredibly high - even for big companies - not considering the long times to build them, which would not solve the problem in the short term [17]. Cheaper and faster solutions are necessary to be found.

Business intelligence and competitive intelligence are two processes that encompass many technologies such as analytics, dashboards, data mining and reporting to analyze data and get information to support making critical and strategic decisions. These techniques use an immense amount of data, stored in data warehouse, data marts and data lakes. In addition to this, many operations running at chip-design level and hardware design processes run on cloud infrastructure, which is stressed by an immense amount of structured ad unstructured data and operations flowing through it. This is one area that requires further investigation and improvements.

Analyzing this phenomenon alone would require too much time and effort, but most importantly it is out of the scope of our work. However, it is important being aware of

the context we are working in and the challenges the industry is facing.

5.2. The Value of Data

Considering the discussion above, we can state that, even if it is not the direct subject of our research, data are a fundamental underlying asset: they are the most valuable asset of the last decade and it will be even more valuable in the future [9]. But why it is so valuable? The answer to this question is not that straightforward, since it is not a topic systematically evaluated.

Even if some common elements can be found when discussing why data are so valuable, in order to give a clear answer we have to consider the context in which they are used. We can say, however, that data are an important asset that, if used properly, can bring a huge impact to a company's business.

(Big) Data represent an opportunity for companies, and the ability of effectively managing this opportunity is a key advantage over the competition. If used properly, data represent a new factor on the same level of human capital and hard assets. If we consider the manufacturing sector, factories equipped with the proper technologies can achieve improvements in design, production and product quality: an industry adopting these technologies is called Industry 4.0 [119].

Data value is generated in many different ways, and it has repercussions in a variety of fields. Figure 5.1 represent the dimensions of the data value ecosystem. As we can observe, there is a large variety of variables involved around big data. First of all, a high level of expertise is necessary to bring value from data, as well as strong domain knowledge. The right technologies must be carefully chosen to overcome the issues related to acquiring heterogeneous data sources, store a huge amount of data, real-time data processing and interoperability. Furthermore, applications are necessary to deliver data to customers to support many different aspect of production and actively benefit from the value of data. Business models should also be employed to simulate growth in economic activity. Finally, it is also necessary to consider the social implications of big data and the legal aspects, such as privacy and regulations.

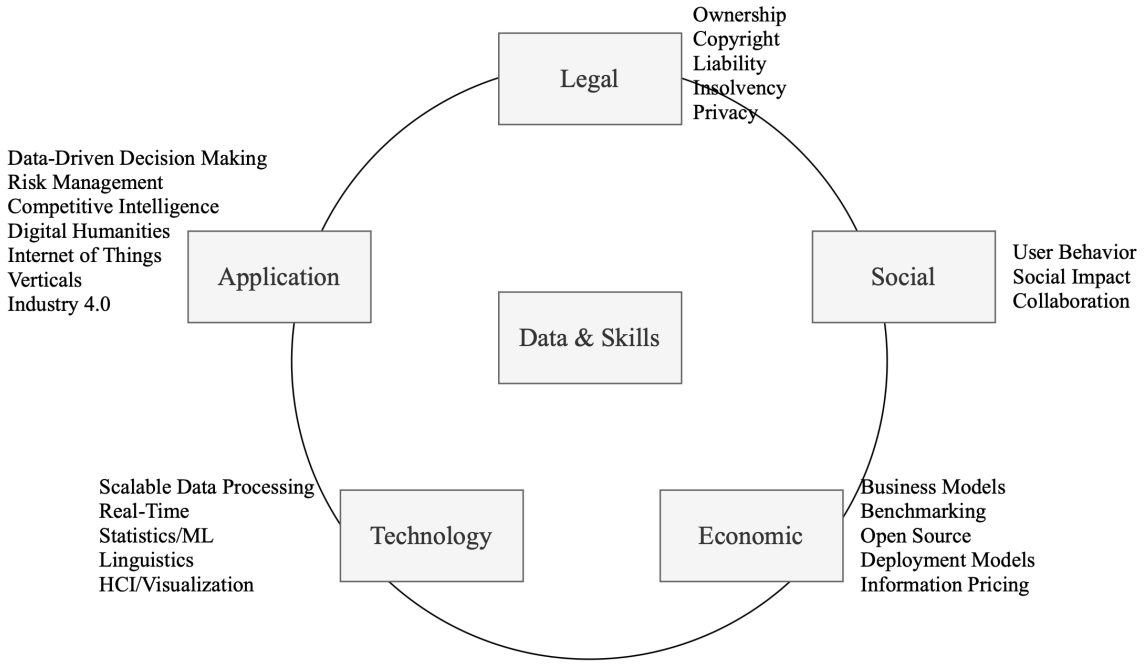

Figure 5.1: Big Data Value Ecosystem [9].

Data are an asset, they are information. For this reason, to define value we can refer to the Seven Information Laws proposed by Moody & Walsh in 1999 [73], who for the first time identified the laws that govern information behavior as an economic resource:

- Information is infinitely shareable;
- The value of information increases with use;
- Information is perishable;
- The value of information increases with accuracy;
- The value of information increases when combined with other information;
- More is not necessarily better;
- Information is not depletable.

These laws helps us to reach the objective of defining value. However, determine value is not possible if we do not define a purpose, as the interests in information define how valuable actually is.

If we consider the semiconductor industry, data can be used not only to improve the quality of the final products, enhancing the production line and increase the quality controls, but also have extreme value for business decisions. For instance, hardware engineers use

Electronic Design Automation (EDA) tools to create and verify chip designs: these tools use a huge amount of resources, and they are computationally intensive. Semiconductor companies rely on very large High Performance Computing (HPC) systems that provide all the necessary resources to the designers. However, resources are limited and some EDA jobs occupy a large amount of resources. It might be necessary to kill jobs that occupy compute slots for long time, idling. This example falls in the category of resource optimization, and in order to do so, it is necessary to collect data through a Real-Time Monitoring (RTM) system capable of collecting data of HPC nodes usage. By using historical data, it is possible to predict the computational load of resources and manage them properly, to avoid resource waste [119].

This is just an example of why data can be so valuable, but many other applications exist where data are an important resource that can be used to make the right decisions. This process is called data-driven decision making, and business intelligence is the discipline that regulates it. It is in this context that we worked.

5.3. Introducing the Company's Environment

Considering all the previous insights on the current status of the semiconductor industry and the importance of data in the business intelligence decision-making process, we conducted a study in collaboration with NXP Semiconductors (NXP from now on), one of the largest semiconductors companies in the world. Their focus is the design and the production of silicon technologies for - mainly - the automotive industry and smartphone industry.

In the past recent years, NXP adopted the DevOps strategy to support the development of the infrastructure used to retrieve data from different sources and load them in their many data lakes, in order to support a wide range of governance decisions. In particular, they use the well-known SCRUM methodology, scaled-up for big companies: the SAFe methodology.

Both of these methods are the latest evolution of the Agile framework, which is, as we have seen in our SLR, one of the pillars of the DevOps culture. SAFe allows teams and stakeholders to align to a shared mission and vision. At the basis of the whole process resides the following concept: *The most efficient and effective method of conveying information to and within a development team is a face-to-face conversation*[1]. As we have previously discussed, communication is the most important element of DevOps.

[1]https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/

Daily meetings, weekly review sessions and planning weeks are there to enhance the collaboration withing team members and different teams.

In our specific case, we worked with a data producer team, which is in charge of developing and maintaining the platform used to collect data from different sources, clean them and load them on the data lake. These data are used by other teams, as they apply machine learning and statistical algorithms to retrieve useful information to support business and governance decisions across the organization.

We worked with the team for a period of 5 months.Throughout this period, the team experienced fluctuations in its resources. The core team consisted of six engineers, excluding ourselves and other students. This number falls exactly in the suggested size for DevOps teams, as it has been proved that small teams - that is, ten or fewer individuals[2] - suite better in the Agile framework.

For the whole period we worked with the company, the team was dealing with the problem of maintaining a data ingestion infrastructure, while at the same time migrating from an on-premises environment to a cloud platform. The company adopts a wide variety of cloud tools provided by different vendors. The NXP employs various tools to facilitate their daily operations, including log- and metric-collection, alerting, and notification via ticketing-systems to detect and resolve operational issues. These tasks align with the typical responsibilities of a DevOps team.

5.4. The Team and Its Goal

The SCRUM[3] methodology suggests dividing the work into sprints, i.e., short periods of time, were are defined a set of user stories. Each user story represents a specific topic on which a team member should work, typically belonging to a bigger topic. Which stories will be done on a specific sprint and how big a Story is, it is decided during the planning sessions, where each team member can vote the amount of effort a Story requires and, according to the team's capacity (the working power), a certain amount of stories is chosen.

During the 5 months period of collaboration with NXP, the team was focusing on improving the system developed in the past months, by introducing higher level of monitoring and optimizing resource usage. However, to better understand the main objectives of the team, here follows a list of all the team's activities:

[2]https://scaledagileframework.com/agile-teams/
[3]https://www.scrum.org

- Build data ETL pipelines: ETL pipelines are necessary to ingest data from data sources and load them on the Data Lake;
- Monitor data streams: ETL data streams must be monitored in order to be aware of the status of the system and act in case of failures or other issues;
- Data gap detection: among the issues the team deals with, data gaps is the most common one and detect it is compulsory;
- Re-ingestion to fix gaps: in case data gaps are detected, it is required to re-ingest the data and fix the gaps;
- Data quality checking: data quality is another important activity carried out by the team, since data loaded on the data lake must reach a certain level of quality;
- Notify data consumers: data consumers must always be aware of status of the ingestion pipelines of their data;
- Clean data: among the data quality checks, data cleaning is the most common practice;
- De-duplicate data: data de-duplication is a compulsory activity to remove duplicate record: only one unique instance of data is kept;
- Share with consumers: consumers that used specific data must always be able to access them;
- Manage cloud platform: the maintenance of the cloud platform is fully managed by the team.

All of these activities are related to the environment depicted in Figure 5.2. As we can see, we are dealing with 12 different data sources, which can be of 3 different kinds: files, logs and data streams: in general, we can say that these data are related to time sheets, project names, licenses, jobs, tools usage and project storage accounts.

Given the diversity in data sources, not only in the content but also in the type of sources, we are dealing with a very complex environment. On each of these data sources an ingestion process is performed, where data are cleaned and prepared for the data lake.

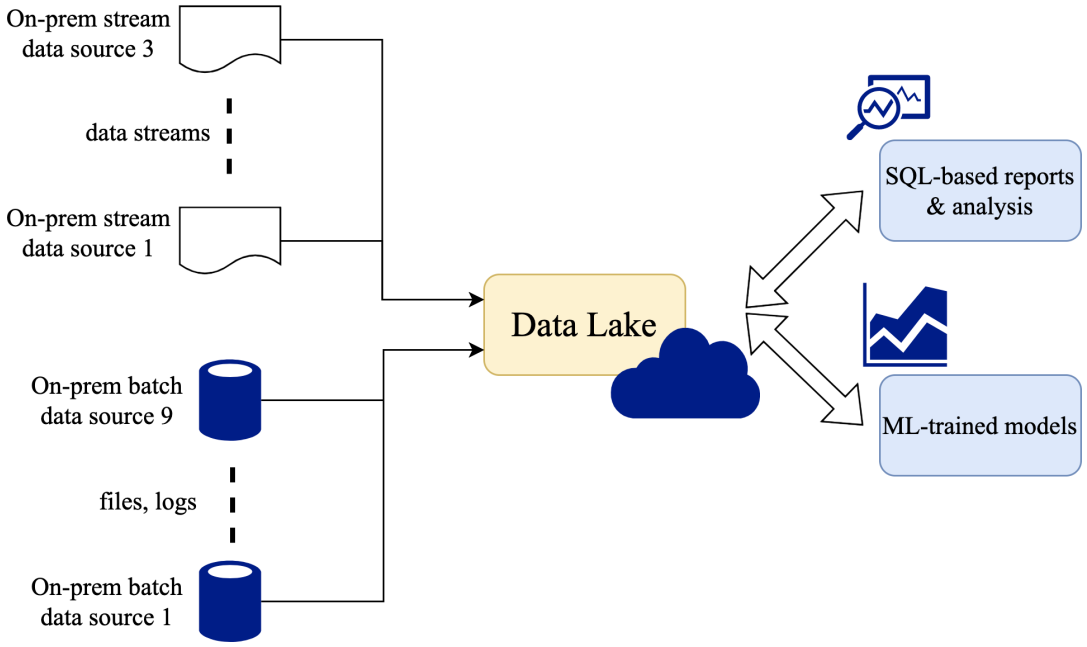

Figure 5.2: Team activities overview.

In the picture it is not shown the process of ingesting data - as we will discuss it later - but it is show how data are used by the stakeholders: mostly, reports and analyses are performed to improve efficiency and in the end, cost reduction, typically by the help of ML-trained models.

5.5. The Proof-of-Concept

Given all the previous considerations, in this section we introduce the reader to the work done in collaboration with NXP Semiconductors.

5.5.1. Introduction

In their cloud environment, the team is dealing with around 200 ETL pipelines. ETL jobs are deployed through a serverless data integration service, that allows the creation of complex ETL pipelines to load data into data lakes. The service is based on Apache Spark[4], the open-source multi-language unified analytics engine for executing large scale data processing.

Most of the jobs configured in the deployment environment are used to retrieve data from different on-premises data sources and load them, after several quality checks, on

[4]https://spark.apache.org/

the team's data lake. For this reason, these jobs, among other configuration settings, need a Virtual Private Cloud (VPC) connection, as they need to connect through several sub-nets to the local database instances.

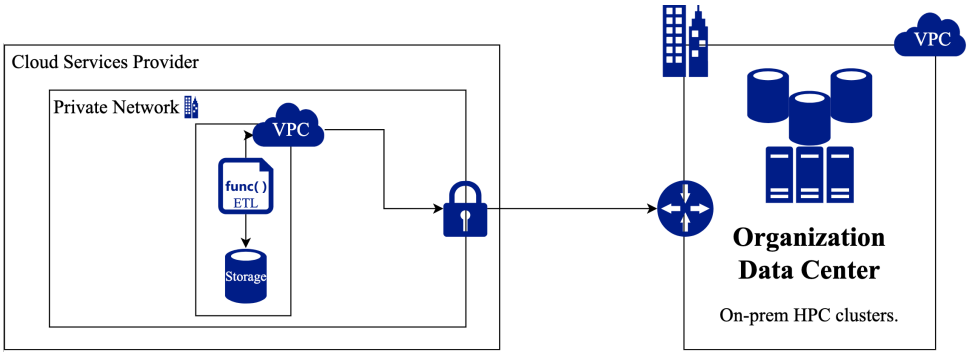

Figure 5.3: Architecture overview.

ETL jobs are configured to retrieve data from a wide variety of sources, and in order to do so they rely on other services provided by the cloud vendors, which are storage services and NoSQL databases. The important aspect to consider is that they have been chosen to benefit from scalability support, data availability, security and performance offered by the fully-managed services.

Figure 5.3 depicts the high-level architecture of the system we are dealing with. As we can see, jobs need to connect to a VPC in order to access the on-premises HPC Clusters, where most of the data sources reside. Supposing that no issues happen during this process, each ETL job accesses a storage unit to load the ingested data. This is an over-simplified view of the real architecture, and there are some issues that needs to be solved, as we will see in the next section.

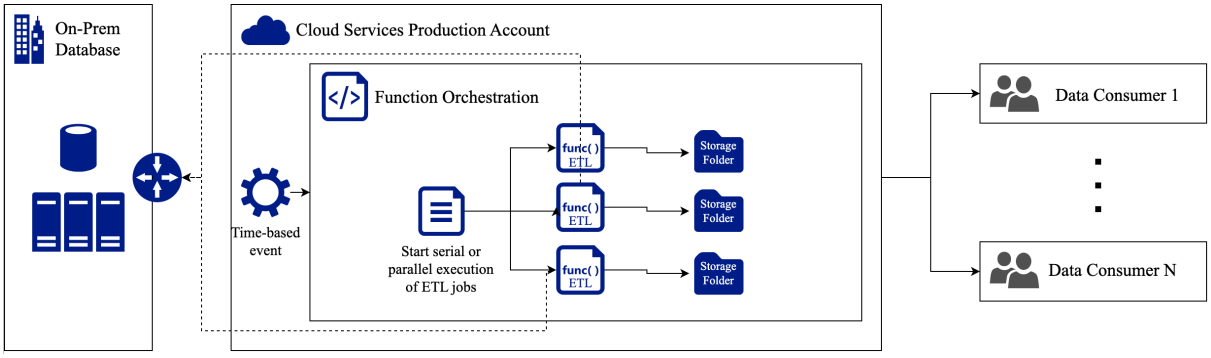

Figure 5.4: ETL pipelines architecture, with data source and target.

Figure 5.4 gives us further details regarding the high-level architecture of the system we are dealing with. In particular, we can see how ETL jobs are scheduled in the team's

cloud environment. Time-based events trigger a workflow, also known as an orchestrator or state machine. Each step in a workflow is called a state: a unit of work, with inputs and outputs that a cloud service performs. This service is used to start the serial or parallel execution of a predefined set of ETL pipelines within the function orchestrator: this is the place were issues might arise due to the connection to Virtual Private Cloud required by the ETL jobs. Finally, the ingestion pipelines write the processed and raw data to their target storage folder. From here, data consumers accounts can read data, which are shared with them, to made further processing and evaluation.

5.5.2. The issues

The architecture discussed above, although it satisfies the preliminary requirements for a basic working of the ingestion pipelines, shows some defects, mainly related to lack of complex control flow and readiness checks. In this section we explain which are the main issues that the team is aware of and needs to solve.

We begin by discussing a resource limitation: the number of IP addresses available. This kind of resources are provided by the team through requests presented to the Cloud Center of Excellence (CCoE) which is a centralized governance function, a task force built to ensure the correct adoption of cloud across the organization.

Following the diagram (Figure 5.5), we can observe the problem. At peak load, jobs can fail due to IP address exhaustion (the pool of unallocated IPv4 addresses is drained, causing job launch failures).

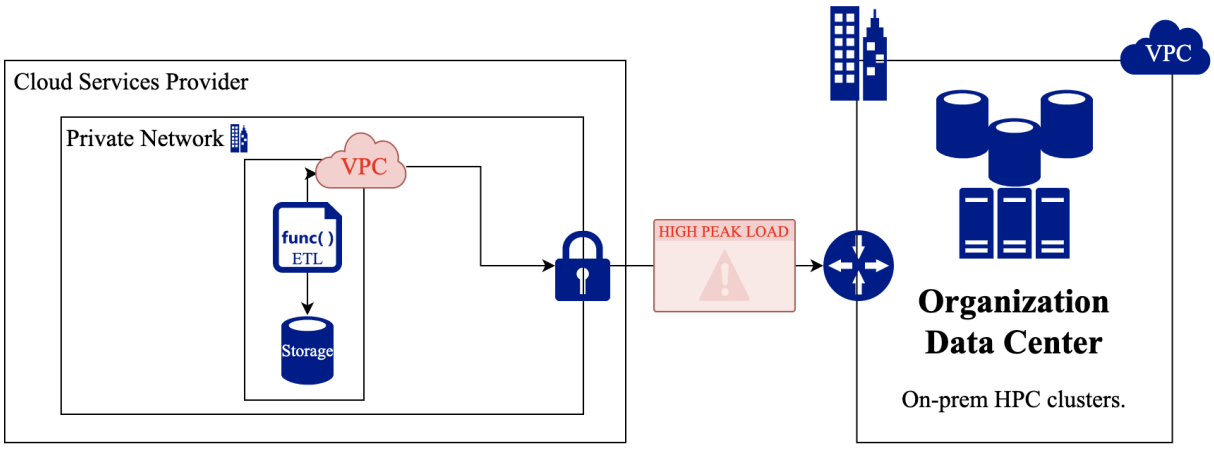

Figure 5.5: High-level architecture overview with IP limitation issue.

The issue we just mentioned is related to resources limitation. However, there are other known issues within the adopted architecture. In order to explain which are the other

problems, we will use as a reference example presented in Figure 5.6. It represents the high level logic of one of the orchestrators created to manage the complex application workflows. In the cloud environment, ETL jobs are grouped by data source type, and for each data source an orchestrator regulates the execution of the related pipelines. Executions can be serial or parallel, depending on the type of jobs involved, i.e., incremental or batch, and the resource type: for instance, serial execution is preferred when large batch jobs need to be executed and to avoid overloading the data source with multiple parallel jobs. With the current status of the system, it is not possible to recover from possible errors that might rise during a pipeline execution since, as we can see, there is no try-catch mechanism implemented.

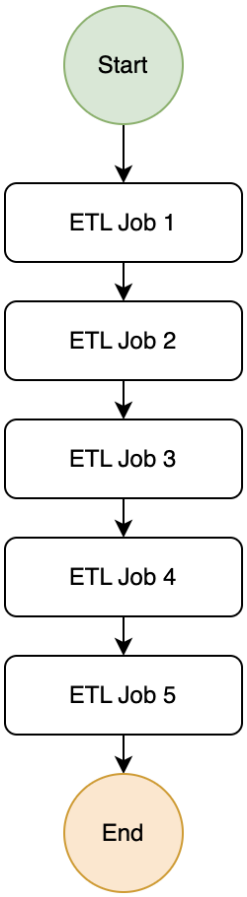

Figure 5.6: Function orchestrator example with serial execution of different ETL Jobs.

This implies that, if a job fails to execute, the whole orchestrator fails and, as a consequence, all the other jobs in the same sequence do not execute.

Missing the try-catch feature implies that it is not possible to implement some kind of retry mechanism: in principle, it is possible to set, during the configuration of a function

orchestrator, the number of times a step execution can be attempted; however, this does not allow reporting what cause the execution to fail. Furthermore, there are some jobs, i.e., the incremental ones (also named snapshot jobs) that have higher priority on the batch-type jobs: so, supposing to have a retry mechanism, it is need a check that assess the type of job, so that only in case it is a snapshot one the execution is attempted again.

On top of these issues, there is lack of monitoring. As we have seen in our SLR chapter, we have found very few papers tackling monitoring, and this does not mean that the topic is not relevant. It is the opposite: without monitoring, there is no insight over the status of the system, and there is no way to know what are the failing jobs and why. Also monitoring is necessary on the resources, since in order to run ETL jobs it is necessary to have enough IP addresses available. For these reason, in the past few months the team worked hard to achieve a certain level of monitoring, by collecting metrics and creating dashboard and alerts. For instance, ETL pipelines are monitored and metrics related to the execution status, outcome and cause of failure are collected in a unique dashboard, so that the team can now visualize the status of jobs and evaluate statistics such how often jobs fail and what are the most common causes. This step was crucial in order to design, later, a solution to address the most common problems, as we will see in the next section. Also, a monitoring system has been design to get insight on the status of the IP subnets usage.

To conclude this section, we just highlight how without these two monitoring systems would have not been possible to design a solution to address these problems. Having eyes on what happens in a system it is a necessary step to non only understand why errors arise, but also to properly solve them, as stated by all the team members.

5.5.3. The solution

In order to solve the issues mentioned in the previous sections, we envisioned the design and implementation of a new state-of-the-art job orchestrator. The baseline idea consists of adding several steps to the current orchestrators in use, in order to introduce readiness checks and more complete logic. At design stage there have been several discussions on which could be the most sound and complete tool to use, as developers did not know yet if the one offered by a vendor is capable of not of addressing all the issues rather than the service offered by a different vendor. Thus, during the design stage there was not only doubt regarding how to address each of the aforementioned issues, but also the final tool to be used.

Now we present the model of the solution proposed. However, we will not discuss here

how we reached to the model or what are the tools used to model and deploy it: this is due to the fact that in the next sections we will discuss two methodologies to reach this goal, designing experiments in order to understand which of the two methodologies is better. So, here we will just discuss the logic lying behind the model. Before discussing the model depicted in Figure 5.7, we explain what is the language model used, and why we used it.

The Business Process Modeling Notation[5] (BPMN) is a popular standard for modeling enterprise process workflows, providing a visual notation, capable of capture business processes in a clear and consistent way. BPMN process are composed by two main elements, activities and events, connected together by sequence flows. Tasks represent atomic units of work, and they can be either atomic or compound (sub-processes): they are both represented by rounded rectangles, with the difference that sub-processes contain a process inside. Events on the other hand are represented by circles and depict an occurrence of a fact: for instance, in Fig. 5.7 we have a start and an end event. Finally, we have gateways, which are capable of expressing divergence of convergence of sequence flow: they are diamond-shaped with a symbol that describe the type of gateway that is, for instance exclusive or the logical AND. In Fig. 5.7 we can observe exclusive gateways, with the purpose of representing a choice, similar to an if-else statement.

Now we discuss the logic behind the model in Fig. 5.7. Right after the start event - right now is not relevant what kind of event triggers the execution, we will discuss it in the next chapters - we have a task that is meant to read from a table and a compound task, *parallelSetExecution*, that executes in parallel each set of jobs that needs to be run. For each set of jobs, in the compound task we have at the very beginning a task that transforms the dataset in a list of jobs, so that it is possible to execute each job in the list in parallel, thanks the sub-process named *parallelJobExecution*. Inside this task we can find the actual logic that handles the execution of each job: the *retryExecutionLoop*.

[5]https://www.bpmn.org/

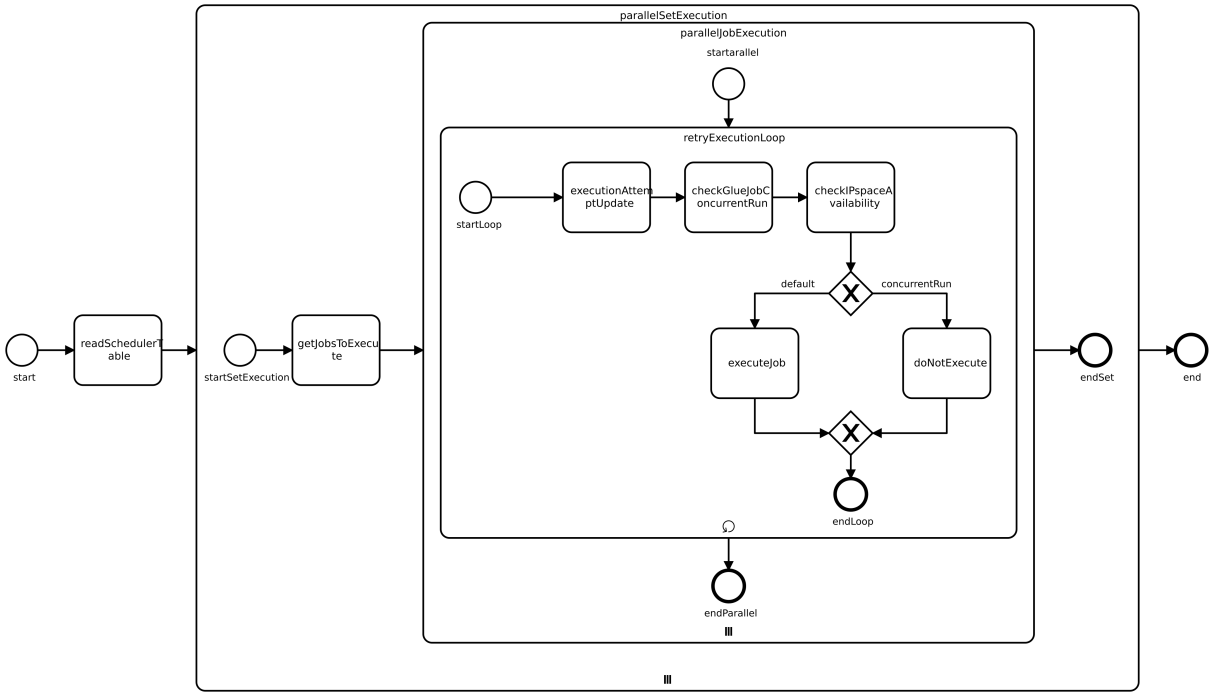

Figure 5.7: BPMN model of the proposed solution.

In the loop, the following issues are addressed: concurrency of a job, IP space availability and retry execution of incremental (snapshot) jobs. In principle, also a try-catch statement is foreseen for the *executeJob*; however, it is not possible to see it in the BPMN model. At the beginning of the loop, the execution attempt counter of the current job is updated: the loop keeps going at most for three attempts. It is subsequently checked if another instance of the same job is run, to avoid concurrency issues. Then, IP addresses availability is checked: if not enough IPs are available or another instance of the job is running, we do not execute the job; otherwise, the job is executed.

The jobs executions are usually parallel. However, for certain jobs might be serial. For each ETL job, the loop ends if there is a successful execution or if the maximum attempts for a job are reached (three at most).

In the following chapters we show what are the two approaches we compare in order to model, develop and deploy this job orchestrator solution, by modeling two kinds of experiments, collecting data and discussing the results, in order to assess the pros and cons of both methodologies.

5.6. Two Approaches: RADON and Baseline

In this study we want to compare the RADON approach to model, develop and deploy FaaS solutions, with the baseline approach, i.e., design and deploy a solution by using directly the tools provided by the cloud services provider. At the basis of both approaches we assume a design phase where all the team members discussed the requirements that the new job orchestrator should satisfy, as we have seen in the previous section.

5.6.1. The Baseline Approach

The baseline approach represents the case where, in order to design and deploy a new job orchestrator solution, we did not rely on the support of any extra tool or tool-set, but only on the tools provided by the cloud services vendor. With this approach we only manually developed all the necessary steps with the different kind of services needed, without the need of any extra abstraction layer, which instead provided by RADON. In order to model the job orchestrator, we directly relied on the function orchestrator modeling tools provided by the cloud vendor. Also, we used many serverless functions, in order to properly make readiness checks, which were triggered by the function orchestrator, since the latter one is used to specify the execution flow and not the content of the steps itself.

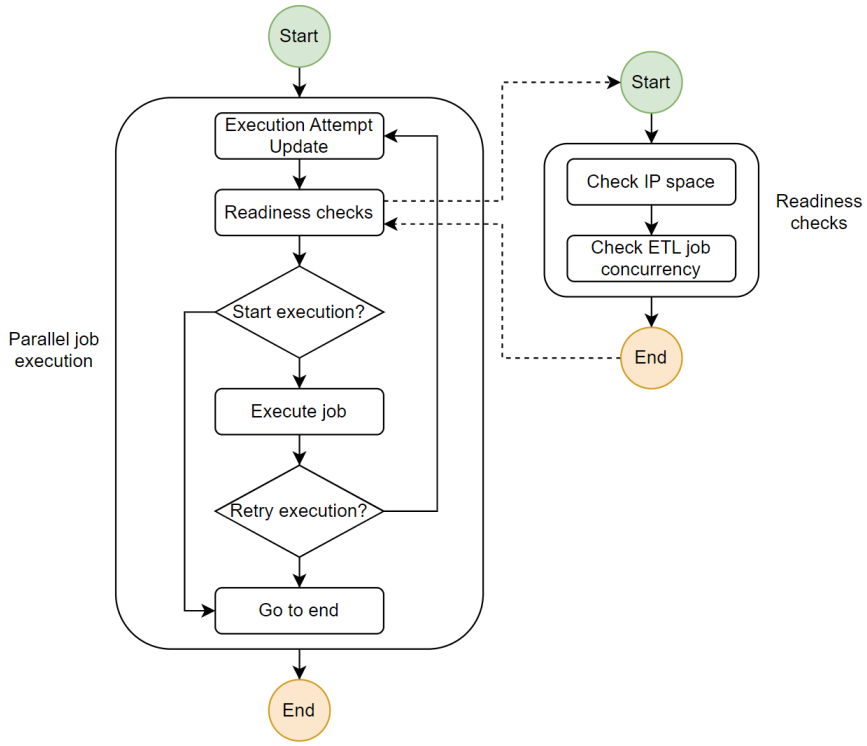

Figure 5.8: Function orchestrator graph: final solution.

Figure 5.8 shows the graph of the final function orchestrator deployed. We discuss now the steps, and what they represents. However, we do not discuss here what is the though process behind the decision taken to design the function orchestrator, since we will do it in the next chapter, where we compare this approach with the RADON approach, and the results of the experiments.

In order to understand the image, we assume that every minute a serverless function is triggered. The serverless function reads from a key-value database table the list of jobs that needs to be executed at a given time, and creates an instance of the function orchestrator on the left of the image for each set of at most five ETL jobs. Suppose that at 12:00 13 jobs needs to be executed: then, the serverless function creates sets of five jobs; in this case, we have three sets, two composed of five jobs and one of three jobs. For each of these sets, a function orchestrator instance is executed. The orchestrator then calls, in a series of steps, for each ETL pipeline in the set, different serverless functions which are in charge of: updating the execution attempt of an ETL job, perform readiness checks (in a different function orchestrator) such as IP space availability and job concurrency, and if these checks are good, then the execution of the job starts; otherwise, if the job is of type snapshot, the execution is attempted again at most three times.

5.6.2. The RADON Approach

As we have seen in the previous section, the Baseline approach requires the direct use of different technologies offered on a cloud platform, with no need of an extra abstraction layer on top of them. However, most of the technologies offered by one vendor are not available on another platform, and if a company wants to adopt another cloud services provider, there is the need to learn to use their specific equivalent services, when they exist.

A novel tool-set and methodology has been developed to address these issues: RADON. RADON is a framework that not only provides and end-to-end application life-cycle management methodology and tools, by envisioning a model-based approach to manage various aspects of serverless application development, such as verification, decomposition, defect prediction, continuous testing, monitoring and CI/CD, compatible with a variety of cloud platform. Indeed, the RADON IDE has to be intended has an extra layer that is build on top of the most popular cloud services, and it does not represent a replacement to the proprietary tools offered by these vendors, but it is a methodology which aims to reduce workload and effort on practitioners working with this kind of technologies. The RADON methodology has been developed by applying method engineering and the

Topology and Orchestration Specification for Cloud Application (TOSCA) as the baseline for models definition.

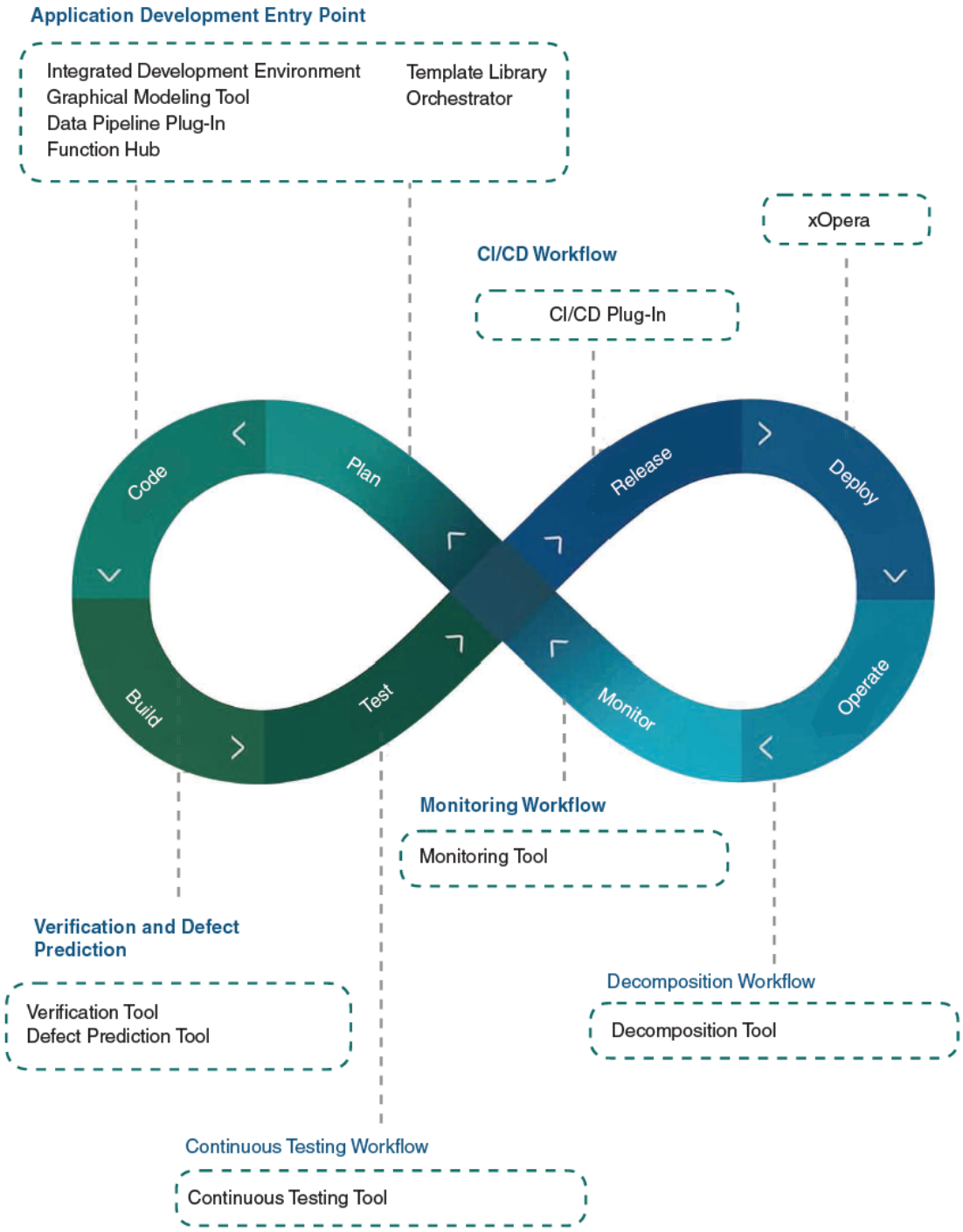

Figure 5.9: RADON DevOps model [29].

Figure 5.9 shows the model of the RADON DevOps life-cycle. This schema reflects

the structure of the traditional DevOps life-cycle, but it adds, for each phase, the tools RADON uses to apply the theoretical concepts. As we can observe, many tools are present within the RADON framework but, however, this is not a proper novelty since, for instance, similar capabilities are reached by other services. RADON, is open-source and offers a unique characteristics: it offers multi-cloud portability, by combining graphical modeling and TOSCA-based templates to abstract deployment characteristics. For these reasons, RADON provides a valuable support to DevOps teams working with cloud services.

In our research, we will focus on a subset of technologies adopted by the RADON framework, all of them open source, described in detail by [29], where the TOSCA language is extended to support the development of serverless function orchestrators. In particular, our focus is directed towards the Graphical Modeling Tool (GMT), namely Eclipse Winery, the deployment tool, xOpera - both TOSCA compliant - and another tool developed by the authors of the cited study, that is, BPMN4FO. In the next subsections we provide a brief description of the purpose and capabilities of each tool.

Workflow overview

RADON, as is in the current development status, only supports the development of FaaS microservices, hosted on cloud VMs, provided by different vendors. Yussupov et al., in their study, extend the capabilities of TOSCA by creating specific node types and relationships to support the modeling and deployment of serverless function orchestrators.

The proposed workflow is structured as follows: first, Business Process Model and Notation (BPMN) is used to model serverless function orchestration and then transform it in target orchestration formats. Then, the TOSCA-based deployment modeling approach (Eclipse Winery, based on TOSCA) is extended to support function orchestration and finally xOpera, the deployment automation technology, is used to deploy the model on the target service.

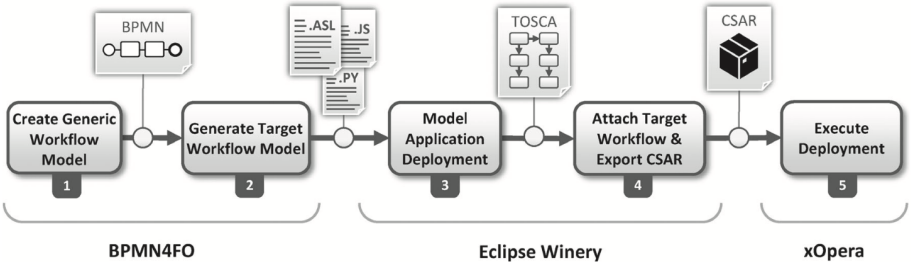

Figure 5.10: Tool-chain for modeling and deployment of function orchestrations [118].

Figure 5.10 shows the tools used to implement the concepts previously described. All the tools used are open-source and are based on well-known standards: BPMN and TOSCA. The first tool is BPMN4FO, which is used to graphically create the function orchestrator model and generate the target model. Then, Eclipse Winery, used in the RADON project and here used on top of a TOSCA extended version, is used to model the application deployment and generated the IaC code in a compressed format (CSAR file), which is then used by the last tool, xOpera, to execute the deployment on the target cloud technology.

BPMN4FO

BPMN for Function Orchestration (BPMN4FO[6]), currently at prototype stage, is used to design serverless function orchestration models. It is based on bpmn-js[7], which is a graphical BPMN-compliant editor adapted such that it is possible to create technology-agnostic function orchestration models in a way that, if the model is compliant to the supported technology, it is possible to generate the target-technology models. In order to obtain these capabilities, the technology-specific constructs have been uniformed and applied to the BPMN elements. For this reason, only the common constructs have been modeled, while the ones which are present only in one of the supported technology must be manually added at a later stage; however, the authors claim to extend the support also to these constructs. This tool is capable of exporting both the BPMN model of the function orchestrator and the technology-specific model.

Eclipse Winery

Eclipse Winery[8] is a web-based environment, included in the RADON project, which provides a GUI for not only topology modeling, but also to define TOSCA constructs. Indeed, in the study a set of new constructs is presented, which is capable of support the deployment modeling on the aforementioned technologies. The most important construct is the Orchestrator Node Type - which is slightly different for each provider - that represents the orchestrator technology and is meant to store the orchestrator model generated by BPMN4FO. The Orchestrator node then is connected to the different other nodes through the Orchestrates relationship, and each node represent a BPMN Task, that is, a serverless function. Of course, these constructs are compatible with the all the other TOSCA constructs provided by RADON. The resulting model can be exported as a Cloud Service ARchive (CSAR file), which is the compressed IaC code, with all the constructs

[6]https://github.com/iaas-splab/matoswo
[7]https://bpmn.io/toolkit/bpmn-js/
[8]https://github.com/eclipse/winery

and artifacts to deploy the model. In order to conduct our experiment, we used the docker container provided by the RADON project (RADON GMT).

xOpera

xOpera[9] is a TOSCA compliant orchestrator to automate deployment of TOSCA-compliant models. In our case, it is the CSAR file provided by Eclipse Winery. Assuming all the target account info is provided in the CSAR file, xOpera automatically deploys the model. xOpera is available is different versions, such as CLI or SaaS. For our experiments, we used xOpera CLI 0.6.5 in a Python 3.8 virtual environment.

5.7. Design of the Experiments

In order to conduct our study, we defined a family of experiments, that is, a set of experiments that share the same goal so that it is possible to combine the results in order to enhance their maturity. With that in mind, we followed the guidance of Santos et al. [94], where they suggest having at least three experiments to compose a family. Each experiment is composed of one factor, that is, the methodology employed to design and develop the job scheduler, and two treatments, that is, the RADON approach and the Baseline approach.

5.7.1. Goal and Research Questions

Our family of experiments share the same goal: analyze the design and deployment of a function orchestrator in a serverless environment, with and without using RADON, with the purpose of estimating them with respect to their effectiveness, efficiency, perceived ease of use, perceived usefulness, and intention of use from the viewpoint of novice practitioners in the context of DevOps team. By chasing this goal, we address the following research questions:

- **RQ1**: Which is the most effective approach to design and deploy function orchestrators?
- **RQ2**: Which is the perceived easiest approach?
- **RQ3**: Which approach is more useful?
- **RQ4**: Which tool is intended to be used?

[9]https://github.com/xlab-si/xopera-opera

5.7.2. Context

The context in which the experiments have been conducted has already been described in detail in the previous sections; here we summarize it. The context of this research is the design and deployment of function orchestrators on a cloud platform. In particular, the context is defined by (i) the infrastructure resources, that is, the cloud services platform used by the DevOps team: in our case, we used the AWS, among the many platforms available; (ii) the methods selected, that is RADON and baseline, extensively described in the previous sections; (iii) and the selection of participants to the experiments.

5.7.3. Participants

According to Kitchenham et al [58, 59] students are the most interesting participants when conducting experiments, since they are next generation of practitioners. In particular, well-trained final-year students can be considered as valid experimental subjects. In particular the experiments have been conducted by the authors of this study, with the support of trained practitioners, i.e., the other team members, who have experience that spans from a minimum of two year to several decades in the IT field. We included them in the study since they supported the main participants in making assessments and reasoned judgments.

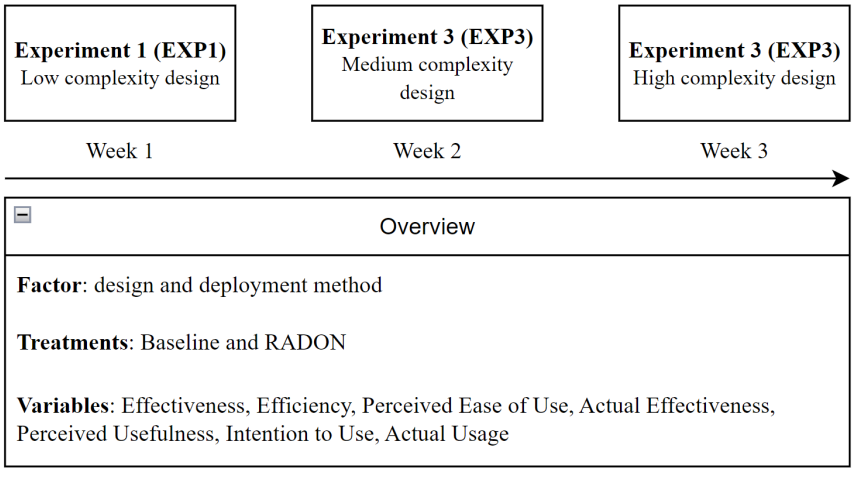

Figure 5.11: Sequence of experiments with factor, treatments and variables involved.

5.7.4. Experiments

The three experiments we conducted are similar, in the sense that they involve the same participants in the same context.

However, they differ on one aspect: for each experiment has been designed and deployed a different version of the job scheduler, and each experiment introduced more complexity in the solution.

This was made to have, in the end, a combination of results that are not biased by the complexity of the function orchestrator designed and deployed. The complexity of each job scheduler solution has been measured by counting the number of steps involved in the orchestrator and the number of branches in the model; in the end, we obtain three different complexity levels: low, medium and high, as we can see in Table 5.4.

Figure 5.11 shows the sequence of experiments performed in the chronological order, highlighting the complexity of the job scheduler solution involved.

We adopted the Method Evaluation Model in conjunction with factorial as theoretical reference for our experiments: in the following sections, we describe in detail how we structured our work.

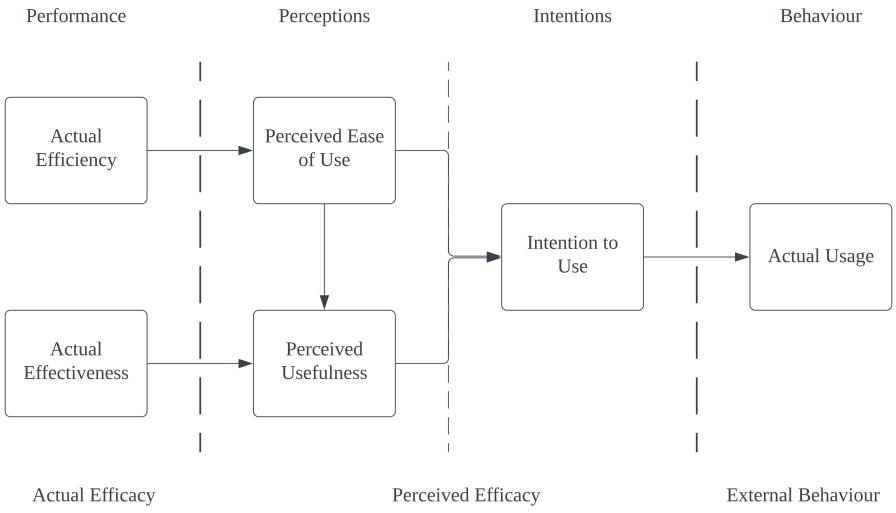

Figure 5.12: Method Evaluation Model.

5.7.5. Variables selection

The Method Evaluation Model (MEM) [72] is a method that derives from the Technology Acceptance Model (TAM) [31]; the MEM defines the theoretical approach to follow for the evaluation of our two methods.

It assumes two different dimensions: actual efficacy and adoption in practices. It makes uses of several constructs, defined in Figure 5.12.

In particular, to measure performance are envisioned two constructs: Actual Efficiency and Actual Effectiveness, which represent the effort required to apply a method and the degree to which a method achieves its objectives, respectively. Perceived Ease of Use represents the degree to which a person feels comfortable using a method, while Perceived Usefulness represents how a person feels that a method is useful to achieve the intended objectives. The Intention to Use, instead, represents the extent to which a person is willing to use a particular method. Finally, Actual Usage is the extent to which a method is used in practice. In our experiments we will use all of these constructs as dependent variables. Table 5.1 summarizes all the variables used in our experiments.

Name	Measure	Scale
Effectiveness (EFCT)	$\frac{\text{Number of Requirements Satisfied}}{\text{Total Number of Requirements}}$	Ratio
Efficiency (EFFC)	$\frac{\text{Effectiveness}}{\text{Time}}$	Ratio
Perceived Ease of Use (PEOU)	5-point Likert Scale	Ordinal
Perceived Usefulness (PU)	5-point Likert Scale	Ordinal
Intention to Use (ITU)	5-point Likert Scale	Ordinal

Table 5.1: List of dependent variables.

The first two variables are performance-based: Effectiveness is defined as the number of requirements satisfied by designing and deploying the function orchestrator, given the experiment, over the total number of requirements. The efficiency instead is Effectiveness over time, since it represent how long a participant took to perform the task. To each satisfied requirement has been assigned an All-Or-Nothing-Metrics, that is, only two possible values are usable: 1 for satisfied requirement, 0 for non satisfied requirement. All the other variables are formulated as a 5-point Likert scale, with opposite-statement question format.

Variable	Statement
Perceived Ease of Use	This method was easy to learn and to use.
Perceived Usefulness	This method is effective in supporting the development of high quality software, with low effort and time.
Intention to Use	If I am working in a company, I would suggest to use this IaC development method.

Table 5.2: List of dependent variables.

Table 5.2 provides a summary with all the statements we used for our 5-point Likert Scale, relative to the variables Perceived Ease of Use, Perceived Usefulness and Intention to Use.

5.7.6. Null Hypotheses

For each of the three experiments, the same Null Hypotheses hold. The Null Hypotheses have been formulated directly from the dependent variables. The Null Hypotheses of our experiments are:

- **H1$_0$**: EFCT (RADON) = EFCT (Baseline);
- **H2$_0$**: EFFC (RADON) = EFFC (Baseline);
- **H3$_0$**: PEOU (RADON) = PEOU (Baseline);
- **H5$_0$**: PU (RADON) = PU (Baseline);
- **H6$_0$**: ITU (RADON) = ITU (Baseline);
- **H7$_0$**: AU (RADON) = AU (Baseline);

The goal is to reject the null hypotheses and accept alternative ones.

5.7.7. Design

To properly design our experiments, we make use of 3 x 2 factorial design, capable of addressing the issue of small sample sizes, increasing sensitivity of experiments. We also used this technique because, given the limitations related to the number of participants, it was impossible to adopt other experiment design such as AB/BA crossover design, which requires the definition of two group to which assign factors and treatments. So, with Table 5.3 we show how we combined the different complexities of the Job Orchestrator design with respect to the two treatments, that is, RADON and BASELINE. For

each combination we considered, we show how they are assigned to each Week-Period of experimentation.

Design complexity	RADON	BASELINE
LOW	Week 1	Week 1
MEDIUM	Week 2	Week 2
HIGH	Week 3	Week 3

Table 5.3: Design of our experiments.

As we mentioned in the subsection where we describe the variables involved during the experiments, Effectiveness and Efficiency are based on the requirements satisfied. Each period of experimentation refers to a different level of complexity in the design of the orchestrator which, in the end, directly depends on the number of requirements the solution should satisfy.

Table 5.4 summarizes the requirements that the Job Orchestrator solution must satisfy, for each level of complexity. The idea is that the lowest complexity solution must be able to manage the *basic* features to orchestrate all the services to execute a set of predetermined ETL Jobs in series, that is, the orchestrator gets as input a set of ETL job names, and for each of them must check if another instance is running, to avoid concurrency issues and, in case the execution of a job fails, catch the error for reporting the issue. The reason why the execution of multiple jobs is serial resides in the fact that the team does not want to overload the data source, and thus a synchronized serial execution avoids this issue. Synchronized means that the orchestrator not only is in charge on starting the execution of an ETL pipeline, but also waits until the execution ends to start a new one. The medium complexity requirements comprise the ones of the low complexity case, but they also add the need of implementing other two features, that is, IP space availability checks before executing a job, and a retry mechanism for snapshot (incremental) jobs, that have a higher priority over the other ones, that is, batch jobs. Finally, the high complexity design gets the same features as the medium ones, except for requirement 7: indeed, in this case the jobs that needs to be executed is no more fixed, but dynamic. It is foreseen a table, where a list of jobs is stored with all the information necessary to run them at a given schedule; the orchestrator should be able to read the table every minute to check if, at a given time, jobs needs to be run, and run them serially. In the first two cases, instead, the assumption is that the hard-coded jobs have the same schedule, and the orchestrator is triggered according to that schedule.

Req. ID	Req. Description	LOW Compl.	MEDIUM Compl.	HIGH Compl.
R1	The Scheduler Table must always be available to the Job Scheduler			X
R2	The Scheduler Table must be updated every time an ETL job is selected for execution			X
R3	The Scheduler Table must contain all the fields necessary to the Job Scheduler for deciding which Job must be executed at a given time			X
R4	The Job Orchestrator must read the scheduler table every minute			X
R5	Snapshot Jobs require a retry mechanism if the execution fails		X	X
R6	A Snapshot Job can retry the execution at most 3 times		X	X
R7	The list of jobs the orchestrator must execute are hard-coded in the input schema	X	X	
R8	The Job Orchestrator must check if IP addresses are available before executing a job		X	X
R9	The Job Orchestrator must, before executing a Job, must check if another instance of that Job is running (Concurrency check)	X	X	X
R10	The Job Orchestrator must implement a try-catch mechanism when executing an ETL Job	X	X	X
R11	The execution of multiple ETL pipelines must be serial	X	X	X
R12	The execution of multiple ETL pipelines must be synchronized	X	X	X

Table 5.4: Requirements for each design complexity.

5.7.8. Experimental material and assumptions

In order to properly conduct the experiments and avoid biasing the considerations made towards one of the two approaches, we made some assumptions. At the beginning of this chapter we described how the goal is to design, develop and deploy through IaC an ETL job scheduler, with increasing complexity levels. Both approaches, however, have some preliminary steps in common, which we consider among our assumptions: the scheduler has been implemented, in every phase of the experiments, through the combination of function orchestrators and step serverless functions provided by the supported cloud platforms. Although the relationship between each step and between the components vary for each experiment and method employed, the code of the steps in the serverless functions is the same across all the phases. For this reason, we assume that the code of each serverless function has been already developed and tested, and provided as artifact at the beginning of each experiment. This however does not include slight modifications that might be or might not be necessary to adapt the code to slightly different deployments, due to the different capabilities of the methodology employed and the different vendors.

Furthermore, in our experiments we do not consider the time needed to read the documentation of each tool employed by both methodologies. However, we consider within our metrics the time used to solve possible issues and errors related to lack of documentation for a given tool or unforeseen events that slowed down the development process that are not attributable to a deficiency of the person conducting the experiment.

We relied on using the official documentation provided by each tool and service involved.

5.8. Results

In this section we present the empirical evidences collected during our experiments, according to the hypotheses and considerations stated in the previous sections.

Table 5.5 presents the list of the values collected for the variables involved in the experiments, averaged among the experiments. We added the *duration* variable that, even if we did not mention it in the previous sections since it is a variable which value is not directly assigned by the participants, we found important to include in the list since it gives an additional important information regarding the experiments and helps to the discussion of the results. Indeed, time is a factor that indirectly or not influences the values assigned to all the other variables.

Variable	RADON	BASELINE
Effectiveness	1	1
Efficiency	0.077	0.69
Duration	15.33h	14.66h
Perceived Ease of Use	2.66	3.66
Perceived Usefulness	4.33	4
Intention to Use	2.67	4.33

Table 5.5: Collected results: all values represent the average over the experiments; values are dimensionless, except for duration which unit is hours.

The most important observation about time is the following: in principle, one could state that time should increase as the complexity of the solution to be deployed increases. However, this is not true, since one aspect to take into account is the reuse of components. Indeed, in both methodologies is it possible to used part of the artifacts generated by one solution for another. Although this would seem to alter the results of our experiments, because reusing components among each other would imply to make each experiment dependent from another, it actually does not: the reuse of a component is a crucial aspect of Infrastructure as Code, and not taking advantage of it would make the results less reliable. Indeed, since we did it for both methodologies, the results are coherent.

With the collected experience, we can make the following considerations, that we will use later to give an answer to our research questions.

5.8.1. Effectiveness

Effectiveness score is high in both approaches: in particular, the maximum score has been reached by the Baseline approach, with the RADON approach scoring lower. The Baseline approach, indeed, offer higher granularity and more precision while developing the infrastructure, since it exploits directly the capabilities of the tools offered by cloud platforms, that on the other side are used over an abstraction layer, namely through TOSCA constructs, in the RADON approach.

Using directly the cloud services gives the developer of higher level of control on each component, giving the possibility to specify some behaviors and deployment settings that are not available (yet) with the RADON approach. Nonetheless, this does not mean that the RADON approach is a limitation: actually, it is an enhancement since it gives the

opportunity to a developer to have a graphical overview of the deployment model, which helps to better visualize all the variables involved in the application, an aspect that is crucial in DevOps since development and operations stages continuously interchange.

With BPMN and Eclipse Winery combined, it has been possible to continuously update team members, including product owners and SCRUM master, and make everybody to understand what the objective of the project was, even to the ones with less technical expertise. This is an aspect that should not be neglected, since we view it as an opportunity for managers to have a better understanding on the status of issues and stories (speaking in SCRUM terms).

However, we have to highlight few limitations in the RADON method that caused concern during the development process: first of all, BPMN4FO translates BPMN tasks to target-specific serverless functions; if it is needed to use other function orchestrator target-specific states, it is necessary to make a more complex serverless functions to address the lack of states. In principle, this is not a big issues, but in the Baseline Approach it is possible to directly exploit this functionality, due to the state-of-the-art support provided by cloud vendors.

Another limitation reside in the fact that sometimes it is not possible to set some *advanced* function orchestrator settings: for instance, in order to model a *parallel execution* state in BPMN4FO is is necessary to create a sub-process task and add the parallel flag; however, it is not possible to set it as a serial execution. In our case, in fact, it was not possible to execute ETL jobs sequentially by using the model created on BPMN4FO.

Finally, one big limitation is to be found in the lack of debugging info in the graphical interface of BPMN4FO: at the very beginning, when learning to use the tool, this is very frustrating since it can make the learning curve very stiff, leading to trial and error. Nonetheless, we have to mention that the version used is a prototype, and it is possible that in the future this issue will be addressed.

5.8.2. Efficiency

The next metric we analyze is efficiency. By looking at Table 5.5, we can see how RADON have been recorded to be more efficient than the baseline approach. This result is a consequence of how the RADON method is structured, with respect to the baseline approach. Indeed, with the baseline methodology it is necessary to first model and create the components in each cloud service involved, from serverless functions to function orchestrators.

Then, it is necessary to create the infrastructural code: even by using proprietary tech-

nologies, that in some cases allow to create the infrastructure using common languages such as Python, this two-steps process can be quite slow. However, very experienced programmers can develop applications directly through them; nonetheless, this was not our case. BPMN4FO, Eclipse Winery and xOpera tool-chain results being more efficient since it provides a well-defined sequence of steps which leads to the deployment of the infrastructure with very little effort, once the user learns to use each tool in a proficient way.

The tool-chain provided by the RADON approach, moreover, improves efficiency since with the Eclipse Winery blueprints it is possible to re-use deployment models and adapt them to new requirements with very little effort, and everything is done through a graphical interface. On the other hand, on the Baseline approach of course you can reuse every single component created before, but it is more difficult to have a clearer view of the infrastructure since blueprints and graphical tools are - most of the time - not available.

5.8.3. Perceived Ease of Use

Perceived ease of use is the first variable we find that drives points in favour of the baseline approach: indeed, learning to use the RADON tool-chain and using is has been perceived to be much more difficult than the Baseline approach.

First, learning to use each cloud service necessary for the solution is much easier due to better documentation and also greater support from the community and the vendors. Also, cloud vendors offer a dedicated support service that, when some suggestions or reviews are necessary, is very helpful. On the other side, the whole RADON tool-chain is open source, it is at prototype stage and does not have a huge user base. This implies that all the information a user needs can only be found in the official documentation which, however, is still unripe.

For this reason, learning to use every tool, especially BPMN4FO and Eclipse Winery, can be time consuming and frustrating: in addition, there is no user-friendly debugger in both cases, and this leads to unfortunate situations were the user must learn by trial and error the root cause of issues. For instance, in BPMN4FO sometimes it is difficult to understand why the tool does not generate the target declaration of function orchestrator, since does not show where the model does not comply with the standard ways of modeling. Also, Eclipse Winery, besides the fact that is prone to crashes and errors while running, does not provide a automatic way of assessing whether or not a blueprint is ready for deployment: if something is missing, in order to detect it, it is sometimes necessary to export the CSAR file and verify the deployment on xOpera, then go back on Winery and

fix the issues.

5.8.4. Perceived Usefulness

Considering just the requirements that needs to be satisfied by the deployed application, both approaches are perceived similarly by user. Indeed, both of them are are capable of achieving the intended objectives. However, it is interesting to notice that if we take into account the fact that one approach - RADON - offers the possibility to migrate to another platform vendor with little or no effort.

However, this is not the only thing to consider. Indeed, RADON offers two levels of standardization. One the one hand, it relies on the TOSCA standard and thus it is able to create a deployment model compliant to a state-of-the-art standard which, in the end, offers the aforementioned capability of developing application in a technology-agnostic manner, but also it offers a standard to another level: the way of working.

Even if the team we worked with has well-defined ways of working and guidelines are offered to schedule each step of the development process of a new feature in the cloud platform, it is not always the case that these rules are followed. It happened multiple times, also outside of our experiments, that team members adapted the way of working according to the specific task they wanted to accomplish: this is an issue, since it violates all the rules the tam imposed to itself to achieve better software quality and control.

RADON solves this issue by defining a structured way of working which is not possible to violate since each step is depended from the previous one and each step relies on a different tool. This creates a reliable way of working which assures a level of software quality that is not achievable otherwise, and this is where we can find the strong perceived usefulness.

5.8.5. Intention to Use

The Intention to Use is the variable that contains the most uncertainty and doubt, if we consider the RADON approach.

Concerning the Baseline approach, we have the maximum adoption score since it is not only the standard way of working the team we worked with adopted, but also it is the common approach adopted by many other teams in the company, and it is of course the way of working suggested by the cloud service providers' support. This is also the method the team will most probably work with in the future.

On the other hand, the RADON approach is very appealing due to its obvious benefits

which spans over different levels of the DevOps culture. Nonetheless, due to the lack of support and the early stage of development, it is difficult to assess whether the tool is to be used in the future. For this reason, the score for the RADON approach is low.

However, this is not the only reason why the intention to use is low. There is also to consider that practitioners are familiar with the most well-known platforms' environment, and it would be necessary to spend resources and time in making the team members learn to use the tool, on top of the others: this is a grey area, since, although the benefits of adopting the tool are known, it is not clear if they cover the costs spent to use to learn the tool, due to lack of documentation in this topic.

5.8.6. Hypotheses Testing

Considering the low-level expertise of the participant and the discussion made in the previous sections, we test the Hypotheses. In particular, we found that:

- $\mathbf{H1_1}$: EFCT (RADON) = EFCT (Baseline);
- $\mathbf{H2_1}$: EFFC (RADON) > EFFC (Baseline);
- $\mathbf{H3_1}$: PEOU (RADON) < PEOU (Baseline);
- $\mathbf{H4_1}$: PU (RADON) > PU (Baseline);
- $\mathbf{H5_1}$: ITU (RADON) < ITU (Baseline);

The new Hypotheses Hn_1 show in a clearer way what we obtained by our experiments. In particular, for the Effectiveness variable we can say that, although reached in different ways, all the objectives and requirements were satisfied in both cases. For what concerns the Efficiency, which is measured as effectiveness over time, given the same level of Effectiveness reached in both methods, time was higher for the Baseline approach and the two-step process necessary to deploy the IaC, while RADON provides a seamless tool-chain which improves the delivery times. However, this does not mean that the Perceived Ease of Use is lower for RADON but, as opposite, it is lower for the Baseline approach: as we explained before, however, this is due to the fact that the tool-chain we used in the RADON methodology is mostly at prototype stage and it has been found difficult to use. However, the Perceived Usefulness is higher for RADON since the benefits have been recognized to be important. Nonetheless, the Intention to Use is still lower for RADON since it is unclear how it will be improved in the future and to what extent will be supported by the community.

5.8.7. Answering the Research Questions

We now answer to the research questions we declared at the beginning of this study:

- **RQ1.** *Which is the most effective approach to design and deploy function orchestrators?* With the evidence we produced, we can assess that the most effective approach is RADON. As we already mentioned, the RADON approach, if correctly implemented and completely learned, can introduce a level of efficiency which is not reachable otherwise. The tool-chain is a powerful element that rigorously defines a way of working that cannot be broken, and it also leverages the standardization such that teams become no longer dependent to a specific service they use, and teams are also capable of presenting the projects they bring on to people with little technical knowledge.

- **RQ2.** *Which is the perceived easiest approach?* Even if RADON is the most effective, for sure it is not the easiest. Again, the lack of documentation and support from the community play a crucial role here, as opposite to the Baseline approach that, even is less organic and fragmented, results is an easier use thanks to the extensive documentation provided by cloud platform vendors and the large amount of projects and community members available on the web.

- **RQ3.** *Which approach is more useful?* We can answer to this question on two different levels. If we just consider the objectives the methods need to reach, both are on the same level. They reach the objective using different paths and techniques, but in the end the result is very similar and the difference cannot be noticed. However, as soon as we introduce the possibility of migrating from a vendor to another, it is clear how the usefulness of RADON approach out-ranges the Baseline approach. Overall, we can say that an approach like RADON is extremely useful when the possibility of migrating from one system to another is a real possibility, or if the same part of infrastructure are often reused within the same team or among teams, since it offers the opportunity to reuse blueprints. Another aspect to consider is also the design stage, as the BPMN4FO tool offers a great support in this phase.

- **RQ4.** *Which tool is intended to be used?* Although RADON has been welcomed with great enthusiasm by many team members and although the benefits of RADON has been recognized, it does not seem the case that RADON will be adopted any soon, at least in the current status. The adoption of a new tool represents a sort of threat to the efficiency of teams, given that they might be an obstacle at the very beginning that can significantly slow down the operations: learning a new tool is always a critical aspect to be considered.

5.9. Conclusions

In this chapter we brought testimony of the numerous challenges we have depicted in the SLR chapter, in an industrial environment, adopting the DevOps culture. Our goal was to provide a proof of concept to verify what are the potentialities of using a novel tool-chain, composed of BPMN4FO, Eclipse Winery and xOpera - to be used in conjunction with well-known cloud services vendors - by comparing it the baseline approach used by the team we worked with.

Before presenting the results, we first described why we have chosen a semiconductor company, NXP, as theater for our experiments, shedding light on the current difficulties and challenges companies in this sector are facing. In particular, we have seen how the chip shortage crisis have been made even worse by the COVID-19 pandemic, and we have pointed out how it is necessary to found novel solutions and methodologies to improve chip production without the need of building new fabs.

Then, we have described why data are so valuable, especially in the semiconductor industry, bringing some examples of valuable data in this industry and describing what are the origins of the data value definition.

We moved, therefore, to the description of the environment we worked with, that is a small team responsible for building and maintaining a data lake collecting business data from multiple data sources. The team provides data to multiple consumers by using cloud platforms, in an DevOps environment which makes use of the SCRUM methodology and its scaled-up evolution, SAFe. We described in detail what are the goals of the team and the challenges it is facing.

Just before moving to the description of the experiments we conducted, we made a brief digression on what are the most recent developments in the ETL pipelines: ETL pipelines are the tool used by the team to get the data from each source and load them to the data lake. We have seen how ETL pipelines evolved to cope with streaming data, and how ML can be used to improve their monitoring.

Finally, we presented our proof of concept, by describing the issues we wanted to address and how we applied the novel tool-chain we wanted to test. We compared this approach with the baselines by building a ETL job scheduler, at different levels of complexity, using both methods and comparing the results according to different metrics used.

We found out that practitioners can gain great benefits from the adoption of the RADON tool-chain and methodology, in conjunction with the cloud services providers. Indeed,

although we verified the effectiveness of a subset of the whole tool-chain, RADON covers most of the areas in the DevOps life-cycle that present criticality. We have focus on the design and modeling of function orchestrators in a standardized way and their deployment: using BMPN modeling standard, is it possible to define the logic of function orchestrators in a technology-agnostic environment, and then produce the target model in the corresponding format; at the moment, the supported target technologies are offered by AWS, Microsoft Azure, Apache Openwhisk Composer and IBM Composer. In this way, a team can reach a higher level of independence from the actual target platform, but also enable to team to work and produce application components in a standardized way, across the whole organization. The other benefit we found by adopting this novel technology was that Eclipse Winery allow to create deployment model templates. This means that it is possible to re-use the same topology for many different purposes, and adapt it to the needs of the specific application. On top of that, it is possible to create new node types and relationships through the graphical interface, allowing enhancing even more the capabilities of the TOSCA standard, which Winery is based on. Another important benefit of the adoption of tool-chain is the effortless deployment of the application model, which is done automatically by xOpera.

Nevertheless, although we found a lot of benefits, there are also some drawbacks. They are mostly related to the fact that the tool-chain is not mature enough to be adopted in an industrial environment. Documentation is not clear, and most of the time practitioners have to find out themselves why something went wrong and fix compatibility issues between the versions of the tools. Also, the tools do not provide a comprehensive debugging support, and this means that sometimes it is necessary to adopt a trial and error strategy to cope with problems in the model. Even though RADON is provided also in a SaaS package, to be easily deployed on, for instance, a container hosted on one of the supported cloud providers, the company we worked in was concerned about possible security issues, and the tools' documentation itself explicitly warns the users about this. If the tools were more mature, the team we worked with was actually prone to consider their adoption.

6 | Conclusions and future developments

DevOps is a culture that is gaining more and more popularity among practitioners and many companies are adopting it due to its proven benefits. This discipline covers a wide range of topics regarding software development life-cycle. Among these, Infrastructure as Code is a rather recent process created to manage and provision data center infrastructure through code instead of manual process. There are many tools and techniques developed to support this process, but there are some areas still under research.

Among the many areas DevOps covers, we found that standardizing Infrastructure as Code development has paramount importance in this fields, since there is evidence to state that the adoption of standards-based strategies could lead to great benefits both in terms of resource reduction and cuttings costs, since standards-based procedures can reduce the amount of time to deliver new deployments and reduce the complexity of large scale applications.

Many technologies are currently available on the market to support IaC development, and this is a problem: the wide variety of tools has an impact on practitioners, who have to deal with many different tools even within the same team. This is another reason why standardization is necessary.

The standard we focused on is the OASIS TOSCA, the most popular one and used by a wide spectrum of tools and frameworks. Among these, we found RADON as one of the most advanced and mature tool-chains capable not only of supporting standards-based IaC development, but it provides a complete methodology to design, model and deploy serverless functions.

By looking for related works, we found that, even though these are very popular topics among the research community, they need a deeper investigation, also because there are relatively recent with respect to other similar topics: for instance, the TOSCA standard, which was presented in 2013, gained popularity among IaC tool only around 2018.

With these considerations, we conducted two studies to go into details of these issues both from a theoretical perspective and a practical one. Our work provides two main contributions: a systematic literature review and a proof-of-concept.

6.1. Systematic Literature Review

In our SLR we provided a set of 71 publications, which were analyzed and classified to answer four research questions. These questions, and the answers, are summarized in Table 6.1.

	Research Question	Result
RQ1	What are the current challenges DevOps discipline is facing?	Sheer amount of tools, techniques and languages introduce a high level of fragmentation; monitoring, IaC automation and standardization are critical areas.
RQ2	What is the role of Infrastructure as Code in DevOps?	IaC allows developers to reach a higher level of automation in CI/CD pipelines and reduces the responsibilities of team members in infrastructure maintenance.
RQ3	How TOSCA standard can improve the development and adoption of Infrastructure as Code?	The adoption of TOSCA-based tools reduces the risk of platform dependency, automates deployment and reduces development times.
RQ4	What are the most recent tools developed to support standards-based IaC code development?	RADON, PIACERE, SODALITE, GLITCH, xOpera, Eclipse Winery, BPMN4FO are some examples.

Table 6.1: SLR: research questions and answers summary.

With the SLR we provide the following contributions: (i) with RQ1 we produce an investigation on what are the current challenges, difficulties and pitfall of DevOps disciplines from a high-level perspective, with particular focus for IaC and standardization; (ii) we then shed light on why IaC is so important in the most recent adoptions of DevOps, with RQ2, and how it pushes automation on the next level; also, we highlighted the challenges IaC is currently facing; (iii) subsequently we analyzed, by answering RQ3, why the

TOSCA standard is so important in this field and what are the benefits of adopting it during IaC development; (iv) With the last research question (RQ4) we conclude by providing the state-of-the-art for what concerns tools which adopt and support the TOSCA standard.

On top of these contributions, which strictly derive from the research questions, we also have produced an analysis of how publications are distributed over the years, which are the most prolific authors on particular subjects and finally we produced a taxonomy regarding DevOps concepts. This taxonomy, that has been derived from the concepts we found in the publications we analyzed, might not be complete, but as far as our knowledge goes, we could not find any other similar work. Indeed, we found taxonomies regarding specific DevOps subjects, such as TOSCA and code smells or bad practices, but we could not find a comprehensive and updated taxonomy of the main concepts of the DevOps culture. The only work we found, which we based on, is the set of conceptual maps provided by Leite et al. [64]: this is the most complete work we found, however it does not provide an actual taxonomy but rather a conceptual map. For these reasons we think that, even if it might not be complete, our taxonomy is a great contribution for researcher who start digging in this field.

6.2. Proof-of-Concept

The second main contribution of our study is the proof-of-concept we worked on in collaboration with NXP Semiconductors. Thanks to our work, we made a step forward in understanding the capabilities, positive and negative aspects of adopting standards-based tools for IaC development and, in particular, the RADON methodology: RADON is one of the most recent tool-chains developed in the past recent years, and we found it during our SLR research. The choice of this particular tool was due to the fact that this framework has not yet reached a level of maturity so high to be adopted by a large company, and for this reason we wanted to contribute by providing evident of its effectiveness. We tested only a small portion of the whole RADON IDE, the one devoted to the design, modeling and deployment of serverless functions.

In addition to that, we also witnessed all the challenges that teams face in a DevOps environment, reflecting the challenges theoretically described in theory by most of the papers found in the SLR, and the actions taken by team members to overcome them.

The goal of our proof-of-concept was to answer to four research questions, which are summarized in Table 6.2.

	Research Question	Result
RQ1	Which is the most effective approach to design and deploy function orchestrators?	RADON tool-chain is more effective than the Baseline approach.
RQ2	Which is the perceived easiest approach?	Due to the low maturity of the tool-chain, RADON is more difficult to use than the Baseline method.
RQ3	Which approach is more useful?	Depending on the environment, the RADON approach might be very useful.
RQ4	Which tool is intended to be used?	Due to lack of support and scarce documentation, we do not see intention to adopt the RADON approach in the near future.

Table 6.2: Proof-of-Concept: research questions and answers summary.

The contributions of our proof-of-concept can be summarized as follows, according to the research questions: (i) we applied three tools of the RADON framework - BMPN4FO, Eclipse Winery and xOpera - in an industrial environment - in conjunction with cloud services - producing another evidence for the applicability of the tool-chain; (ii) we compared the RADON approach to the baseline approach, i.e., the traditional way of working that the team we worked with adopted before; (iii) based on the comparison we produced a discussion regarding the pros and cons of adopting that approach, highlighting what are its main limitations and pitfalls; (iv) we also shared another contribution by applying the RADON approach to serverless function orchestration deployments, which is a recent addition to the original RADON methodology; (v) we discussed what solution we envisioned to solve some common issues related to ETL pipelines.

Our findings show that the RADON methodology has a lot of capabilities and it is able to solve most of the issues related to IaC deployment, if adopted correctly. Standardization has been proven again to be, at the moment, the real solution to many problems practitioners are currently facing when dealing with serverless applications deployment in a DevOps environment. However, there is no free lunch. The adoption of RADON in a small team would require, at the current development stage, a lot of initial effort: documentation is not clear, and the tools we tested lack of a complete debugging interface, which means that sometimes, to understand if deployed topologies are TOSCA compliant

or not, trial and error is needed. Even if this initial effort is surmountable, the tools lack support from community and so maintaining them could be quite expensive.

6.3. Future Developments

With the previous considerations we can find many different paths to which direct future research, not only by expanding and improving our work, but also by investigating in some of the fields we found that need further exploration in our SLR.

For instance, considering our experience at NXP and also the few papers we found regarding monitoring in a serverless cloud environment, we find that a state-of-the-art review in this sense is needed: SLRs and Surveys regarding monitoring could be helpful to asses what are the most recent developments in this area, and with recognize that although theoretical studies are necessary, insights from industry are also important since they provide a practical view of the issues.

There is also a lot of interest in code smells and bad practices regarding infrastructure as code development: however, according to our findings, most of these studies related to this topic are produced by the same circle of authors around A. Rahaman. On one side, this is good news since we have very high experienced researched dealing with a non-trivial topic but, however, this implies that the view over these issues in confined to the one of this rather small set of authors. It would be interesting to verify their results from another perspective, and maybe expand their findings by applying them to other IaC tools different from the most traditional ones.

On this topic, we also consider that, even if a wide variety of tools have been developed to detect and also predict these kind of smells and defects in IaC scripts, the evidence of their efficacy is mostly related to smalls sets and open source ones: to have further details and more evident, it would be interesting to see how these tools perform in a real worlds scenario.

However, considering our specific work, we would suggest continuing on reviewing literature by deepening the finding on the topics which were not the main target of our research: by doing this, it would be possible to verify our results and also to expand the DevOps taxonomy we produced, that we know it might not be completed to the limitations we encountered during our research.

For what concerns our proof-of-concept and RADON, instead, we have some aspects that we would like to investigate even more in detail. In particular, our study was limited by time and number of participants. It would be interesting if the same results could

be obtained by involving a broader range of participants; another aspect we were not able to verify was also the following: in our experiments we tested only three tools - xOpera, Winery and BPMN4FO - which do not represent the full potential of RADON. Indeed, another possible expansion of our work would be to verify the efficacy of the whole RADON IDE and its methodology, by applying it to the development of a complex serverless application.

As an example, RADON provides, among the other tools, DEFUSE, which is capable of detecting defects in IaC scripts. Very the efficacy of this tools in a real world scenario would contribute to prove its usefulness and applicability: this connects to the other development we suggested above regarding testing other security related tools.

The final suggestion would be to collaborate with the RADON project itself and expand the support of TOSCA node and relationships to support the deployment of even more complex applications, since at the moment it officially supports serverless function deployment or function orchestrators. Also, the approach we used support the deployment of function orchestrators provided by a limited number of vendors: we suggest expanding the support to other kind of orchestrators, such as Apache Airflow.

Bibliography

[1] D. Abadi, A. Ailamaki, D. Andersen, P. Bailis, M. Balazinska, P. A. Bernstein, P. Boncz, S. Chaudhuri, A. Cheung, A. Doan, L. Dong, M. J. Franklin, J. Freire, A. Halevy, J. M. Hellerstein, S. Idreos, D. Kossmann, T. Kraska, S. Krishnamurthy, V. Markl, S. Melnik, T. Milo, C. Mohan, T. Neumann, B. C. Ooi, F. Ozcan, J. Patel, A. Pavlo, R. Popa, R. Ramakrishnan, C. Re, M. Stonebraker, and D. Suciu. The seattle report on database research. *Commun. ACM*, 65(8):72–79, jul 2022. ISSN 0001-0782. doi: 10.1145/3524284. URL `https://doi.org/10.1145/3524284`.

[2] D. J. Abadi, R. Agrawal, A. Ailamaki, M. Balazinska, P. A. Bernstein, M. J. Carey, S. Chaudhuri, J. Dean, A. Doan, M. J. Franklin, J. Gehrke, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, D. Kossmann, S. Madden, S. Mehrotra, T. Milo, J. F. Naughton, R. Ramakrishnan, V. Markl, C. Olston, B. C. Ooi, C. Ré, D. Suciu, M. Stonebraker, T. Walter, and J. Widom. The beckman report on database research. *Communications of the ACM*, 59:92 – 99, 2016.

[3] J. Alonso, C. Joubert, L. Orue-Echevarria, M. Pradella, and D. Vladusic. Piacere: Programming trustworthy infrastructure as code in a secure framework. 2021.

[4] F. Aqlan and J. Nwokeji. Big data etl implementation approaches: A systematic literature review. 07 2018. doi: 10.18293/SEKE2018-152.

[5] I. Aviv, R. Gafni, S. Sherman, B. Aviv, A. Sterkin, and E. Bega. Infrastructure from code: The next generation of cloud lifecycle automation. *IEEE Software*, 40 (1):42–49, 2023. doi: 10.1109/MS.2022.3209958.

[6] K. Bahadori and T. Vardanega. Devops meets dynamic orchestration. In J.-M. Bruel, M. Mazzara, and B. Meyer, editors, *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, pages 142–154, Cham, 2019. Springer International Publishing. ISBN 978-3-030-06019-0.

[7] L. Baresi, G. Quattrocchi, D. A. Tamburri, and L. Terracciano. A declarative modelling framework for the deployment and management of blockchain applica-

tions. In *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, MODELS '22, page 311–321, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394666. doi: 10.1145/3550355.3552417. URL https://doi.org/10.1145/3550355.3552417.

[8] H. Bauer, O. Burkacky, P. Kenevan, A. Mahindroo, and M. Patel. How the semiconductor industry can emerge stronger after the covid-19 crisis. *Electronic document available at https://www.mckinsey.com/industries/industrials-and-electronics/our-insights/how-the-semiconductor-industry-can-emerge-stronger-after-the-covid-19-crisis*, 2020.

[9] T. Becker, E. Curry, A. Jentzsch, and W. Palmetshofer. *New Horizons for a Data-Driven Economy: Roadmaps and Action Plans for Technology, Businesses, Policy, and Society*, pages 277–291. Springer International Publishing, Cham, 2016. ISBN 978-3-319-21569-3. doi: 10.1007/978-3-319-21569-3_16. URL https://doi.org/10.1007/978-3-319-21569-3_16.

[10] K. Beckmann. Semicondutor industry challenges in 2022: supply chains, data, and sustainability. *Electronic document available at https://www.merckgroup.com/en/the-future-transformation/semiconductor-industry-challenges-2022.html*, 2022.

[11] J. Bellendorf and Z. Á. Mann. Specification of cloud topologies and orchestration using tosca: a survey. *Computing*, 102:1793 – 1815, 2019.

[12] B. Bilalli, A. Abelló, T. Aluja-Banet, and R. Wrembel. Intelligent assistance for data pre-processing. *Computer Standards & Interfaces*, 57:101–109, 2018. ISSN 0920-5489. doi: https://doi.org/10.1016/j.csi.2017.05.004. URL https://www.sciencedirect.com/science/article/pii/S0920548916302306.

[13] N. Borovits, I. Kumara, P. Krishnan, S. D. Palma, D. Di Nucci, F. Palomba, D. A. Tamburri, and W.-J. van den Heuvel. Deepiac: Deep learning-based linguistic anti-pattern detection in iac. In *Proceedings of the 4th ACM SIGSOFT International Workshop on Machine-Learning Techniques for Software-Quality Evaluation*, MaLTeSQuE 2020, page 7–12, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381246. doi: 10.1145/3416505.3423564. URL https://doi.org/10.1145/3416505.3423564.

[14] H. Brabra, A. Mtibaa, W. Gaaloul, B. Benatallah, and F. Gargouri. Model-driven orchestration for cloud resources. In *2019 IEEE 12th International Conference on*

Cloud Computing (CLOUD), pages 422–429, 2019. doi: 10.1109/CLOUD.2019.00074.

[15] U. Breitenbücher, T. Binz, K. Képes, O. Kopp, F. Leymann, and J. Wettinger. Combining declarative and imperative cloud application provisioning based on tosca. In *2014 IEEE International Conference on Cloud Engineering*, pages 87–96, 2014. doi: 10.1109/IC2E.2014.56.

[16] O. Burkacky, M. de Jong, A. Mittal, and N. Verma. Value creation: How can the semiconductor industry keep outperforming? *Electronic document available at https://www.mckinsey.com/industries/semiconductors/our-insights/value-creation-how-can-the-semiconductor-industry-keep-outperforming*, 2021.

[17] O. Burkacky, M. de Jong, and J. Dragon. Strategies to lead in the semiconductor world. *Electronic document available at https://www.mckinsey.com/industries/semiconductors/our-insights/strategies-to-lead-in-the-semiconductor-world*, 2022.

[18] O. Burkacky, U. Kingsbury, A. Pedroni, G. P. nad Matt Schrimper, and B. Weddle. How semiconductor makers can turn a talent challenge into a competitive advantage. *Electronic document available at https://www.mckinsey.com/industries/semiconductors/our-insights/how-semiconductor-makers-can-turn-a-talent-challenge-into-a-competitive-advantage*, 2022.

[19] A. M. Buttar, A. Khalid, M. Alenezi, M. A. Akbar, S. Rafi, A. H. Gumaei, and M. T. Riaz. Optimization of devops transformation for cloud-based applications. *Electronics*, 12(2), 2023. ISSN 2079-9292. doi: 10.3390/electronics12020357. URL `https://www.mdpi.com/2079-9292/12/2/357`.

[20] D. Calcaterra and O. Tomarchio. Automated generation of application management workflows using tosca policies. pages 97–108, 01 2022. doi: 10.5220/0011096200003200.

[21] D. Calcaterra and O. Tomarchio. Policy-based holistic application management with bpmn and tosca. *SN Computer Science*, 4, 02 2023. doi: 10.1007/s42979-022-01616-w.

[22] M. Chiari, M. De Pascalis, and M. Pradella. Static analysis of infrastructure as code: a survey. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pages 218–225, 2022. doi: 10.1109/ICSA-C54293.2022.00049.

[23] C. Cichy and S. Rass. An overview of data quality frameworks. *IEEE Access*, 7: 24634–24648, 2019. doi: 10.1109/ACCESS.2019.2899751.

[24] T. Dai, A. Karve, G. Koper, and S. Zeng. Automatically detecting risky scripts in infrastructure code. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, SoCC '20, page 358–371, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381376. doi: 10.1145/3419111.3421303. URL `https://doi.org/10.1145/3419111.3421303`.

[25] S. Dalla Palma, C. van Asseldonk, G. Catolino, D. Di Nucci, F. Palomba, and D. A. Tamburri. "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*, n/a(n/a):e2533. doi: https://doi.org/10.1002/smr.2533. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2533`.

[26] S. Dalla Palma, D. Di Nucci, F. Palomba, and D. A. Tamburri. Toward a catalog of software quality metrics for infrastructure code. *Journal of Systems and Software*, 170:110726, 2020. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2020.110726. URL `https://www.sciencedirect.com/science/article/pii/S0164121220301618`.

[27] S. Dalla Palma, M. Garriga, D. Di Nucci, D. A. Tamburri, and W.-J. Van Den Heuvel. Devops and quality management in serverless computing: The radon approach. In C. Zirpins, I. Paraskakis, V. Andrikopoulos, N. Kratzke, C. Pahl, N. El Ioini, A. S. Andreou, G. Feuerlicht, W. Lamersdorf, G. Ortiz, W.-J. Van den Heuvel, J. Soldani, M. Villari, G. Casale, and P. Plebani, editors, *Advances in Service-Oriented and Cloud Computing*, pages 155–160, Cham, 2021. Springer International Publishing. ISBN 978-3-030-71906-7.

[28] S. Dalla Palma, D. Di Nucci, F. Palomba, and D. A. Tamburri. Within-project defect prediction of infrastructure-as-code using product and process metrics. *IEEE Transactions on Software Engineering*, 48(6):2086–2104, 2022. doi: 10.1109/TSE.2021.3051492.

[29] S. Dalla Palma, G. Catolino, D. Di Nucci, D. A. Tamburri, and W.-J. van den Heuvel. Go serverless with radon! a practical devops experience report. *IEEE Software*, 40(2):80–89, 2023. doi: 10.1109/MS.2022.3170153.

[30] A. Dalvi. Cloud infrastructure self service delivery system using infrastructure as code. In *2022 International Conference on Computing, Communication, and*

Intelligent Systems (ICCCIS), pages 1–6, 2022. doi: 10.1109/ICCCIS56430.2022.10037603.

[31] F. D. Davis. *A technology acceptance model for empirically testing new end-user information systems: Theory and results*. PhD thesis, Massachusetts Institute of Technology, 1985.

[32] P. Debois. Agile infrastructure operations. 2008. doi: http://www.jedi.be/presentations/agile-infrastructure-agile-2008.pdf.

[33] C. K. Dehury, P. Jakovits, S. N. Srirama, G. Giotis, and G. Garg. Toscadata: Modeling data pipeline applications in tosca. *Journal of Systems and Software*, 186: 111164, 2022. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2021.111164. URL `https://www.sciencedirect.com/science/article/pii/S0164121221002508`.

[34] J. DesLauriers, J. Kovacs, and T. Kiss. Abstractions of abstractions: Metadata to infrastructure-as-code. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pages 230–232, 2022. doi: 10.1109/ICSA-C54293.2022.00051.

[35] A. Dhaouadi, K. Bousselmi, M. M. Gammoudi, S. Monnet, and S. Hammoudi. Data warehousing process modeling from classical approaches to new trends: Main features and comparisons. *Data*, 7(8), 2022. ISSN 2306-5729. doi: 10.3390/data7080113. URL `https://www.mdpi.com/2306-5729/7/8/113`.

[36] P. Di Nitto. Piacere: Programming trustworthy infrastructure as code in a secure framework. In *Short Papers Proceedings of the First SWForum workshop on Trustworthy Software and Open Source 2021 (TSOS 2021), Virtual Conference, March 23-25, 2021*, volume 2878, pages 8–15, 2021. doi: https://ceur-ws.org/Vol-2878/paper4.pdf.

[37] J. Dsouza and S. Velan. Preventive maintenance for fault detection in transfer nodes using machine learning. In *2019 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE)*, pages 401–404, 2019. doi: 10.1109/ICCIKE47802.2019.9004230.

[38] T. F. Düllmann, A. van Hoorn, V. Yussupov, P. Jakovits, and M. Adhikari. Ctt: Load test automation for tosca-based cloud applications. In *Companion of the 2022 ACM/SPEC International Conference on Performance Engineering*, ICPE '22, page 89–96, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391597. doi: 10.1145/3491204.3527484. URL `https://doi.org/10.1145/3491204.3527484`.

[39] A. Dyck, R. Penners, and H. Lichter. Towards definitions for release engineering and devops. *2015 IEEE/ACM 3rd International Workshop on Release Engineering*, pages 3–3, 2015.

[40] Z. El Akkaoui., A. Vaisman., and E. Zimányi. A quality-based etl design evaluation framework. In *Proceedings of the 21st International Conference on Enterprise Information Systems - Volume 1: ICEIS,*, pages 249–257. INSTICC, SciTePress, 2019. ISBN 978-989-758-372-8. doi: 10.5220/0007786502490257.

[41] O. Elazhary, C. Werner, Z. Li, D. Lowlind, N. Ernst, and M.-A. Storey. Uncovering the benefits and challenges of continuous integration practices. *IEEE Transactions on Software Engineering*, PP:1–1, 03 2021. doi: 10.1109/TSE.2021.3064953.

[42] N. Eskandani and G. Salvaneschi. The uphill journey of faas in the open-source community. *Journal of Systems and Software*, 198:111589, 2023. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2022.111589. URL `https://www.sciencedirect.com/science/article/pii/S0164121222002655`.

[43] S. Glen. Devs don't trust ai in software testing. *Electronic document available at https://www.techtarget.com/searchsoftwarequality/news/252523596/Devs-dont-trust-AI-in-software-testing*, 2022.

[44] M. Gokarna and R. Singh. Devops: A historical review and future works. In *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pages 366–371, 2021. doi: 10.1109/ICCCIS51004.2021.9397235.

[45] S. Gorhe. Etl in near-real-time environment: A review of challenges and possible solutions. 04 2020.

[46] J. Grogan, C. Mulready, J. McDermott, M. Urbanavicius, M. Yilmaz, Y. Abgaz, A. Mccarren, S. Macmahon, V. Garousi, P. Elger, and P. Clarke. *A Multivocal Literature Review of Function-as-a-Service (FaaS) Infrastructures and Implications for Software Developers*, pages 58–75. 08 2020. ISBN 978-3-030-56440-7. doi: 10.1007/978-3-030-56441-4_5.

[47] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba. Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 580–589, 2019. doi: 10.1109/ICSME.2019.00092.

[48] M. M. Hasan, F. A. Bhuiyan, and A. Rahman. Testing practices for infrastructure as code. In *Proceedings of the 1st ACM SIGSOFT International Work-*

shop on Languages and Tools for Next-Generation Testing, LANGETI 2020, page 7–12, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450381239. doi: 10.1145/3416504.3424334. URL https://doi.org/10.1145/3416504.3424334.

[49] C. Hashemi-Pour. How ai and automation play a role in itops. *Electronic document available at https://www.techtarget.com/searchenterpriseai/feature/How-AI-and-automation-play-a-role-in-ITOps*, 2022.

[50] H. Hassan, S. Barakat, and Q. Sarhan. Survey on serverless computing. *Journal of Cloud Computing*, 10, 07 2021. doi: 10.1186/s13677-021-00253-7.

[51] F. Haupt, F. Leymann, A. Nowak, and S. Wagner. Lego4tosca: Composable building blocks for cloud applications. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 160–167, 2014. doi: 10.1109/CLOUD.2014.31.

[52] F. Helwani and J. Jahić. Acia: A methodology for identification of architectural design patterns that support continuous integration based on continuous assessment. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pages 198–205, 2022. doi: 10.1109/ICSA-C54293.2022.00046.

[53] M. Isakov, M. Currier, E. del Rosario, S. Madireddy, P. Balaprakash, P. Carns, R. B. Ross, G. K. Lockwood, and M. A. Kinsy. A taxonomy of error sources in hpc i/o machine learning models, 2022.

[54] L. Jiang and A. Eberlein. An analysis of the history of classical software development and agile development. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 3733–3738, 2009. doi: 10.1109/ICSMC.2009.5346888.

[55] M. Jiménez, N. M. Villegas, G. Tamura, and H. A. Müller. Deployment specification challenges in the context of large scale systems. In *Proceedings of the 27th Annual International Conference on Computer Science and Software Engineering*, CASCON '17, page 220–226, USA, 2017. IBM Corp.

[56] V. Kartheeyayini, S. Madhumitha, G. Lalitha, C. Jackulin, and K. Subramanian. AWS cloud computing platforms deployment of landing zone - Infrastructure as a code. *AIP Conference Proceedings*, 2393(1), 05 2022. ISSN 0094-243X. doi: 10.1063/5.0079757. URL https://doi.org/10.1063/5.0079757. 020175.

[57] R. S. Keskin and P. Pileggi. Making the cloud monitor real-time adaptive. In *2022 IEEE Cloud Summit*, pages 69–74, 2022. doi: 10.1109/CloudSummit54781.2022.00017.

[58] B. Kitchenham and P. Brereton. A systematic review of systematic review process research in software engineering. *Information and Software Technology*, 55: 2049–2075, 12 2013. doi: 10.1016/j.infsof.2013.07.010.

[59] B. Kitchenham and S. Charters. Guidelines for performing systematic literature reviews in software engineering. 2, 01 2007.

[60] S. Kokuryo, M. Kondo, and O. Mizuno. An empirical study of utilization of imperative modules in ansible. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS)*, pages 442–449, 2020. doi: 10.1109/QRS51102.2020.00063.

[61] G. S. S. Kumar and M. R. Kumar. Dimensions of automated etl management: A contemporary literature review. In *2022 International Conference on Automation, Computing and Renewable Systems (ICACRS)*, pages 1292–1297, 2022. doi: 10.1109/ICACRS55517.2022.10029274.

[62] I. Kumara, M. Garriga, A. U. Romeu, D. Di Nucci, F. Palomba, D. A. Tamburri, and W.-J. van den Heuvel. The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and Software Technology*, 137:106593, 2021. ISSN 0950-5849. doi: https://doi.org/10.1016/j.infsof.2021.106593. URL `https://www.sciencedirect.com/science/article/pii/S0950584921000720`.

[63] I. Kumara, P. Mundt, K. Tokmakov, D. Radolović, A. Maslennikov, R. González, J. Fernández Fabeiro, G. Quattrocchi, K. Meth, E. Nitto, D. Tamburri, W.-J. Heuvel, and G. Meditskos. Sodalite@rt: Orchestrating applications on cloud-edge infrastructures. *Journal of Grid Computing*, 19:29, 09 2021. doi: 10.1007/s10723-021-09572-0.

[64] L. Leite, C. Rocha, F. Kon, D. Milojicic, and P. Meirelles. A survey of devops concepts and challenges. *ACM Comput. Surv.*, 52(6), nov 2019. ISSN 0360-0300. doi: 10.1145/3359981. URL `https://doi.org/10.1145/3359981`.

[65] P. Leitner, E. Wittern, J. Spillner, and W. Hummer. A mixed-method empirical study of function-as-a-service software development in industrial practice. *Journal of Systems and Software*, 149:340–359, 2019. ISSN 0164-1212. doi: https://doi.org/10.1016/j.jss.2018.12.013. URL `https://www.sciencedirect.com/science/article/pii/S0164121218302735`.

[66] A. Levi. 4 ways ai-driven etl monitoring can help avoid glitches. *Electronic document available at https://www.techopedia.com/4-ways-ai-driven-etl-monitoring-can-help-avoid-glitches/2/33969*, 2019.

[67] G. Machado, I. Cunha, A. Pereira, and L. Oliveira. Dod-etl: distributed on-demand etl for near real-time business intelligence. *Journal of Internet Services and Applications*, 10, 12 2019. doi: 10.1186/s13174-019-0121-z.

[68] M. D. Mascarenhas and R. S. Cruz. Int2it: An intent-based tosca it infrastructure management platform. In *2022 17th Iberian Conference on Information Systems and Technologies (CISTI)*, pages 1–7, 2022. doi: 10.23919/CISTI54924.2022.toscaintent.

[69] A. Mathew, V. Andrikopoulos, and F. J. Blaauw. Exploring the cost and performance benefits of aws step functions using a data processing pipeline. In *Proceedings of the 14th IEEE/ACM International Conference on Utility and Cloud Computing*, UCC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385640. doi: 10.1145/3468737.3494084. URL `https://doi.org/10.1145/3468737.3494084`.

[70] S. McCarthy, A. McCarren, and M. Roantree. A method for automated transformation and validation of online datasets. In *2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC)*, pages 183–189, 2019. doi: 10.1109/EDOC.2019.00030.

[71] K. C. Mondal, N. Biswas, and S. Saha. Role of machine learning in etl automation. In *Proceedings of the 21st International Conference on Distributed Computing and Networking*, ICDCN '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450377515. doi: 10.1145/3369740.3372778. URL `https://doi.org/10.1145/3369740.3372778`.

[72] D. Moody. The method evaluation model: A theoretical model for validating information systems design methods. pages 1327–1336, 01 2003.

[73] D. L. Moody and P. Walsh. Measuring the value of information - an asset valuation approach. In *European Conference on Information Systems*, 1999.

[74] K. Morris. *Infrastructure as Code: Managing Servers in the Cloud*. O'Reilly Media, Inc., 1st edition, 2016. ISBN 1491924357.

[75] G. Novakova Nedeltcheva, A. De La Fuente Ruiz, L. Orue-Echevarria Arrieta, N. Bat, and L. Blasi. Towards supporting the generation of infrastructure as code through modelling approaches - systematic literature review. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*, pages 210–217, 2022. doi: 10.1109/ICSA-C54293.2022.00048.

[76] E. Ntentos, U. Zdun, G. Falazi, U. Breitenbücher, and F. Leymann. Assessing archi-

tecture conformance to security-related practices in infrastructure as code based deployments. In *2022 IEEE International Conference on Services Computing (SCC)*, pages 123–133, 2022. doi: 10.1109/SCC55611.2022.00029.

[77] E. Osaba, J. Diaz-de Arcaya, L. Orue-Echevarria, J. Alonso, J. L. Lobo, G. Benguria, and I. n. Etxaniz. Piacere project: Description and prototype for optimizing infrastructure as code deployment configurations. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 71–72, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392686. doi: 10.1145/3520304.3533938. URL https://doi.org/10.1145/3520304.3533938.

[78] S. D. Palma, D. Di Nucci, and D. Tamburri. Defuse: A data annotator and model builder for software defect prediction. In *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 479–483, 2022. doi: 10.1109/ICSME55016.2022.00063.

[79] A. Pareek, B. Khaladkar, R. Sen, B. Onat, V. Nadimpalli, and M. Lakshminarayanan. Real-time etl in striim. In *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics*, BIRTE '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450366076. doi: 10.1145/3242153.3242157. URL https://doi.org/10.1145/3242153.3242157.

[80] N. Petrović, M. Cankar, and A. Luzar. Automated approach to iac code inspection using python-based devsecops tool. In *2022 30th Telecommunications Forum (TELFOR)*, pages 1–4, 2022. doi: 10.1109/TELFOR56187.2022.9983681.

[81] R. Pushpaleela, S. Sankar, K. Viswanathan, and S. A. Kumar. Application modernization strategies for aws cloud. In *2022 1st International Conference on Computational Science and Technology (ICCST)*, pages 108–110, 2022. doi: 10.1109/ICCST55948.2022.10040356.

[82] G. Quattrocchi and D. A. Tamburri. Predictive maintenance of infrastructure code using "fluid" datasets: An exploratory study on ansible defect proneness. *Journal of Software: Evolution and Process*, 34(11):e2480, 2022. doi: https://doi.org/10.1002/smr.2480. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.2480.

[83] A. Rahman and L. Williams. Source code properties of defective infrastructure as code scripts. *Information and Software Technology*, 112:148–163, 2019. ISSN

0950-5849. doi: https://doi.org/10.1016/j.infsof.2019.04.013. URL `https://www.sciencedirect.com/science/article/pii/S0950584919300965`.

[84] A. Rahman and L. Williams. Different kind of smells: Security smells in infrastructure as code scripts. *IEEE Security & Privacy*, 19(3):33–41, 2021. doi: 10.1109/MSEC.2021.3065190.

[85] A. Rahman, R. Mahdavi-hezaveh, and L. Williams. A systematic mapping study of infrastructure as code research. *Information and Software Technology*, 108, 12 2018. doi: 10.1016/j.infsof.2018.12.004.

[86] A. Rahman, C. Parnin, and L. Williams. The seven sins: Security smells in infrastructure as code scripts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 164–175, 2019. doi: 10.1109/ICSE.2019.00033.

[87] A. Rahman, E. Farhana, C. Parnin, and L. Williams. Gang of eight: A defect taxonomy for infrastructure as code scripts. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 752–764, 2020. doi: 10.1145/3377811.3380409.

[88] A. Rahman, M. R. Rahman, C. Parnin, and L. Williams. Security smells in ansible and chef scripts: A replication study. *ACM Trans. Softw. Eng. Methodol.*, 30(1), jan 2021. ISSN 1049-331X. doi: 10.1145/3408897. URL `https://doi.org/10.1145/3408897`.

[89] A. A. U. Rahman, S. Elder, F. H. Shezan, V. Frost, J. Stallings, and L. A. Williams. Categorizing defects in infrastructure as code. *ArXiv*, abs/1809.07937, 2018.

[90] G. Recupito, F. Pecorelli, G. Catolino, S. Moreschini, D. D. Nucci, F. Palomba, and D. A. Tamburri. A multivocal literature review of mlops tools and features. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 84–91, 2022. doi: 10.1109/SEAA56994.2022.00021.

[91] O. Romero and R. Wrembel. Data engineering for data science: Two sides of the same coin. In M. Song, I.-Y. Song, G. Kotsis, A. M. Tjoa, and I. Khalil, editors, *Big Data Analytics and Knowledge Discovery*, pages 157–166, Cham, 2020. Springer International Publishing. ISBN 978-3-030-59065-9.

[92] N. Saavedra and J. Ferreira. Glitch: Automated polyglot security smell detection in infrastructure as code. pages 1–12, 01 2023. doi: 10.1145/3551349.3556945.

[93] J. Sandobalín, E. Insfran, and S. Abrahão. On the effectiveness of tools to support

infrastructure as code: Model-driven versus code-centric. *IEEE Access*, 8:17734–17761, 2020. doi: 10.1109/ACCESS.2020.2966597.

[94] A. Santos, O. Gómez, and N. Juristo. Analyzing families of experiments in se: A systematic mapping study. *IEEE Transactions on Software Engineering*, 46(5): 566–583, 2020. doi: 10.1109/TSE.2018.2864633.

[95] W. Shin, W.-H. Kim, and C. Min. Fireworks: A fast, efficient, and safe serverless framework using vm-level post-jit snapshot. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 663–677, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391627. doi: 10.1145/3492321.3519581. URL `https://doi.org/10.1145/3492321.3519581`.

[96] C. Siebra, R. Lacerda, J. Quintino, I. Cerqueira, F. Silva, and A. Santos. From theory to practice: The challenges of a devops infrastructure as code implementation. 07 2018. doi: 10.5220/0006826104270436.

[97] D. Sokolowski. Deployment coordination for cross-functional devops teams. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, page 1630–1634, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385626. doi: 10.1145/3468264.3473101. URL `https://doi.org/10.1145/3468264.3473101`.

[98] D. Sokolowski. Infrastructure as code for dynamic deployments. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 1775–1779, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450394130. doi: 10.1145/3540250.3558912. URL `https://doi.org/10.1145/3540250.3558912`.

[99] D. Sokolowski, P. Weisenburger, and G. Salvaneschi. Automating serverless deployments for devops organizations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, page 57–69, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385626. doi: 10.1145/3468264.3468575. URL `https://doi.org/10.1145/3468264.3468575`.

[100] D. Sokolowski, P. Weisenburger, and G. Salvaneschi. Dependencies in devops survey 2021. Apr. 2022. doi: 10.5281/zenodo.6372120. URL `https://doi.org/10.5281/zenodo.6372120`.

[101] M. Son, S. Mohanty, J. R. Gunasekaran, A. Jain, M. T. Kandemir, G. Kesidis,

and B. Urgaonkar. Splice: An automated framework for cost-and performance-aware blending of cloud services. In *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, pages 119–128, 2022. doi: 10.1109/CCGrid54584.2022.00021.

[102] A. Sorour and A. Hamdy. Devops and iac to automate the delivery of hands-on software lab exams. In *2022 6th International Conference on Computer, Software and Modeling (ICCSM)*, pages 28–35, 2022. doi: 10.1109/ICCSM57214.2022.00012.

[103] M. Souibgui, F. Atigui, S. Zammali, S. Cherfi, and S. B. Yahia. Data quality in etl process: A preliminary study. *Procedia Computer Science*, 159:676–687, 2019. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2019.09.223. URL `https://www.sciencedirect.com/science/article/pii/S1877050919314097`. Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 23rd International Conference KES2019.

[104] I. S. e. Souza, D. P. Franco, and J. P. S. G. Silva. Infrastructure as code as a foundational technique for increasing the devops maturity level: Two case studies. *IEEE Software*, 40(1):63–68, 2023. doi: 10.1109/MS.2022.3213228.

[105] O. Standard. Topology and orchestration specification for cloud applications version 1.0. 2013. doi: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.

[106] M. Staron, S. Abrahão, B. Penzenstadler, and L. Hochstein. Recent research into infrastructure as code. *IEEE Software*, 40(1):86–88, 2023. doi: 10.1109/MS.2022.3212035.

[107] U. STATES. Symposium on advanced programming methods for digital computers: Washington, d.c., june 28, 29, 1956. 1956.

[108] A. Suleykin and P. Panfilov. Metadata-driven industrial-grade etl system. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 2433–2442, 2020. doi: 10.1109/BigData50022.2020.9378367.

[109] D. Tamburri, W.-J. Heuvel, C. Lauwers, P. Lipton, D. Palma, and M. Rutkowski. Tosca-based intent modelling: goal-modelling for infrastructure-as-code. *SICS Software-Intensive Cyber-Physical Systems*, 34, 06 2019. doi: 10.1007/s00450-019-00404-x.

[110] H. Teppan, L. H. Flå, and M. G. Jaatun. A survey on infrastructure-as-code solutions for cloud development. In *2022 IEEE International Conference on*

Cloud Computing Technology and Science (CloudCom), pages 60–65, 2022. doi: 10.1109/CloudCom55334.2022.00019.

[111] V. Theodorou, A. Abelló, W. Lehner, and M. Thiele. Quality measures for etl processes: from goals to implementation. *Concurrency and Computation: Practice and Experience*, 28(15):3969–3993, 2016. doi: https://doi.org/10.1002/cpe.3729. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3729`.

[112] P. Vassiliadis, A. Simitsis, and E. Baikousi. A taxonomy of etl activities. In *Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP*, DOLAP '09, page 25–32, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605588018. doi: 10.1145/1651291.1651297. URL `https://doi.org/10.1145/1651291.1651297`.

[113] P. Vassiliadis, A. Simitsis, and E. Baikousi. A taxonomy of etl activities. pages 25–32, 11 2009. doi: 10.1145/1651291.1651297.

[114] B. Violino. Defining enterprise ai: From etl to modern ai infrastructure. *Electronic document available at https://www.techtarget.com/searchenterpriseai/feature/Defining-enterprise-AI-From-ETL-to-modern-AI-infrastructure*, 2019.

[115] N. Wirth. A brief history of software engineering. *IEEE Annals of the History of Computing*, 30(3):32–39, 2008. doi: 10.1109/MAHC.2008.33.

[116] R. Wrembel. Data integration, cleaning, and deduplication: Research versus industrial projects. In E. Pardede, P. Delir Haghighi, I. Khalil, and G. Kotsis, editors, *Information Integration and Web Intelligence*, pages 3–17, Cham, 2022. Springer Nature Switzerland. ISBN 978-3-031-21047-1.

[117] M. Wurster, U. Breitenbücher, K. Képes, F. Leymann, and V. Yussupov. Modeling and automated deployment of serverless applications using tosca. In *2018 IEEE 11th Conference on Service-Oriented Computing and Applications (SOCA)*, pages 73–80, 2018. doi: 10.1109/SOCA.2018.00017.

[118] V. Yussupov, J. Soldani, U. Breitenbücher, and F. Leymann. Standards-based modeling and deployment of serverless function orchestrations using bpmn and tosca. *Software: Practice and Experience*, 52(6):1454–1495, 2022. doi: https://doi.org/10.1002/spe.3073. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.3073`.

[119] Z. Yusufi, S. J. Preis, D. Kraus, U. Kruschwitz, and B. Ludwig. Data value as-

sessment in semiconductor production: An empirical study to define and quantify the value of data. In *Proceedings of the 6th International Conference on E-Commerce, E-Business and E-Government*, ICEEG '22, page 109–116, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450396523. doi: 10.1145/3537693.3537725. URL https://doi.org/10.1145/3537693.3537725.

[120] A. Yıldırım. Devops lifecycle: Continuous integration and development. *Electronic document available at https://medium.com/t%C3%BCrk-telekom-bulut-teknolojileri/devops-lifecycle-continuous-integration-and-development-e7851a9c059d*, 2019.

[121] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall, and M. Di Penta. An empirical characterization of bad practices in continuous integration. *Empirical Software Engineering*, 25, 03 2020. doi: 10.1007/s10664-019-09785-8.

[122] E. Zdravevski, P. Lameski, A. Dimitrievski, M. Grzegorowski, and C. Apanowicz. Cluster-size optimization within a cloud-based etl framework for big data. pages 3754–3763, 2019. doi: 10.1109/BigData47090.2019.9006547.

A | Appendix: Extract, Transform, Load pipelines survey

This appendix represents a brief survey we conducted to better understand the main technology used to ingest data by the team we worked with for our proof-of-concept. To support the work made at NXP Semiconductors, we want to briefly report out the current challenges and state-of-the-art of data ingestion pipelines.

A.1. Extract, Transform, Load: the role of pipelines

Extract, Transform, Load: these are three terms used to identify a sequence of actions that needs to be taken when dealing with collection, cleaning and standardization of data. ETL pipelines are the means through which this idea is modeled.

ETL pipelines are at the basis of data marts and data warehouses, since they handle the collection, cleaning and standardization of structured and unstructured data coming from multiple, heterogeneous, sources. This set of processes has been extensively studied, and many frameworks [23, 35, 40] are available. In the past years, many techniques to improve ETL efficiency have been proposed, and many performance indexes have been found. This witnesses the relevance of the topic and the awareness of researchers of the need to further investigate the concept.

However, studies on this topic typically focus on the process itself, neglecting some aspects such as data quality. In this scenario, it is very hard to find sound and complete research works that provide deep analysis of data quality techniques - belonging to, in general, the broad field of software engineering - without, however, providing practical tools and methodologies to implement them, while other studies shift the problem to quality of the ETL pipeline itself rather than quality of the data flowing through it. Most of them, although considering interesting and innovative concepts and ideas [70, 71, 111], are a

presentation of these notions, making little or none experiments and deep investigation of the involved phenomena: they are inconclusive. However, this opinion arises from a preliminary investigation in research literature, and it does not mean that high quality studies have never been conducted: a deeper investigation is required. Nonetheless, the aforementioned studies are not useless since they give us an idea which direction the research is going.

One crucial aspect of ETL pipelines is the lack of automation: this is not referred to code generation or single steps of the pipeline, for instance merging two data sources, where automation already exists, but what is lacking in the process is a comprehensive automation between steps and automation when errors arise. Some articles suggest reaching a higher level of automation in ETL pipelines, i.e., reducing as much as possible human intervention, to achieve the following improvements: higher data quality - as an automated process is less prone to human error - and better decision making - working on high quality data has been shown to produce benefits at business decision levels [111].

In this scenario, machine learning techniques might be helpful to increase quality levels [71]. One crucial challenge is finding the areas in an ETL process where automation is actually helpful, using known performance indexes, and where ML could be implemented to improve the final product of the pipeline, through the investigation of known but also possibly new performance indexes. Performance indexes are crucial in this scenario, since are the means through which is possible to measure if improvements are made or not [103, 111].

Even though ETL pipelines are presented as the crucial component of Business Intelligence and powerful tool for enterprise-level decision-making, research in this field seems to be conducted in a non-systematic way, and much of the studies represent a wrap-up of what has been done in the last years. In particular, traditional ETL pipelines, which are often called "Batch ETL" [45, 67], are well documented and many commercial tools exist. The only aspect which is acknowledged to be improved in batch ETL is data quality, which, however, is a problem which affects many fields of computer engineering. Albeit data quality is a well-studied subject, every consideration is derived from what existing tools can provide and, thus, all the possible improvements might be found in proprietary systems: however, companies are jealous about sharing their achievements in this strategic area and the studies that propose novel solutions claim to bring improvements which are confined to very specific use-cases and tested only on very small datasets [122]. Thus, scalability remains an issue.

Moving the focus away from data quality alone, three critical areas are still under research:

(near) real-time ETL pipelines, AI-Driven ETL monitoring and standardization [1, 2]

A.2. (Near) Real-Time ETL Pipelines

Near real-time ETL pipelines represent the evolution of batch ETL: many studies [45, 67, 79, 108] describe this recent concept as a necessity for for Big Tech companies, which need to analyze much more data nowadays, in real-time. Indeed, the main issue of batch ETL is that they are typically run overnight and require a lot of time to complete; furthermore, their output is highly affected by the quality of the input: as they say, garbage in, garbage out [66]. Near real-time ETL applies data streaming tools to batch ETL, often by using a combination of these tools such as Apache Kafka and Apache Spark [45, 67, 79]. The most efficient technique seems to be applying Change Data Capture (CDC) design patter to streaming data on top of the aforementioned tools, as some studies claim its effectiveness. Anyways, other techniques are still under investigation but, overall, they try to minimize the amount of data on which the Transformation step works on, by taking small chunks of data each time. Therefore, they are called "near" real-time, because typically these pipelines produce data every hour or less but are not properly real-time. The crucial aspect of this new kind of ETL process is the fact that it is designed to work on a cloud-based distributed setting. Following this new technique of designing ETL pipelines, one aspect which is typically not considered at research level is cost-optimization: using services such as AWS has a cost, and companies want the most effective but cheapest solution [122].

A.3. AI-Driven ETL Monitoring

For what concerns AI-Driven monitoring, things are quite tricky. AI has been shown to be effective in many situations, and sometimes perform even better than humans. Given the large amount of data ETL processes analyze, it is clear that there is space for Machine Learning and AI techniques in this scenario. Most of the attention [4, 37, 61, 114] is turned to the application of automated tasks to ETL monitoring. An ETL process should produce consistent results regardless the flow of data under analysis: a sudden change could be detected by an automated algorithm and generate an alert. Also, machine learning can provide real-time detection for bottlenecks, allowing developers to address problems faster. Furthermore, it can reduce cost of operation since, given a problem, can send alerts directly to who is devoted to solve it, letting the rest of the departments free.

Another powerful application is to directly use ML to develop ETL solutions [12, 114]. This could enhance software engineering quality [49]. There are already ETL-developing

software that provide this feature, but ML action is still limited [49, 116]. Overall, ML is capable of reducing human intervention and costs.

However, there is a caveat. ML and AI sound as great solutions, but in order to use them in an effective way, developers should trust them. This does not seem to be the case. The reasons of this distrust reside in the fact that the technology is typically hard to comprehend. This is the reason why it has been documented that managers, even with clear evidence provided by software - for instance, non AI-powered ETL pipelines -, typically make decisions on their own [43].

A.4. ETL pipelines: towards standardisation

Another important problem, that has widespread impact on ETL pipelines and, in general, data engineering, regards lack of standardization in data integration, data mining, evaluation and interpretation [1, 2]. Different needs in the past years have influenced the development of heterogeneous system that, even if they are really powerful and rely on common technologies, work very differently [116] and this is an issue at development stage since programmers might slow the development due to the need of learning a new programming paradigm; furthermore, complex systems rely on different technologies that at a certain point must be integrated together [91]. This is a problem, especially if we consider that, in many situations, data engineering tasks are left to data scientists and analysts which, in many cases, are not familiar with such tools [91, 116]. Indeed, nowadays there is a wide use of notebooks [1, 91], even at enterprise level, for data integration, cleaning and evaluation task: although notebooks ease the job to programmers for their flexibility and their use of general-purpose languages - which are more familiar to developers -, lack of the power legacy tools provide, and thus it might limit the effectiveness of their tasks [91].

At the basis of all these problems, there is a lack of standardization. This holds not only for what concerns the development stage of data integration systems, but also it is true for research. Indeed, there is a discrepancy between research and industrial projects [116]. Typically, research projects rely unrealistic assumption and tests are conducted on small datasets. Research projects most of the time try to find improvements in optimization and design methods but the effectiveness on the field is still unknown for the aforementioned reasons. However, recently there has been movement towards standardization [1, 91, 112, 116]. On the other hand, industrial projects mainly focus on cost-reduction and ad hoc solutions which, although might be great for a specific application, they worsen the problem. Also, it is hard to measure the cost-effectiveness of new systems since, in most

cases, it is hard to measure the return of investment. This is the reason why the whole standardization process is very slow.

Since standardization is inevitable to achieve improvement in this field, It is crucial to make a step forwards in this direction [91].

B | Appendix: Additional Insights on The Proof-of-Concept

In this appendix we would like to share some additional insights and picture of the experiments we conducted, since we did not want to use too much space on the main block of our work.

Here we will just share the images of the function orchestrators models created with BPMN4FO and the deployment models designed with the RADON Graphical Modeling Tool (Eclipse Winery), so that the reader has a clearer idea of the work we did during our experiment. Indeed, the differences among each experiment will become even clearer with this direct comparison.

In this way we are able to directly observe the connections between the BPMN models and the TOSCA-compliant deployment.

B.1. BPMN4FO Models

Figures B.1, B.2 and B.3 show the different models obtained at each complexity stage of our experiments. At a first glance, the complexity gap is much higher between the low and medium complexity solutions rather than the medium and high complexity ones.

This is due to the fact that in the first level of complexity we envisioned just the checks for ETL job concurrency, and only two serverless functions are required to manage all the steps involved: the main one, delimited by the start and end events, and the internal one, denoted by the sub-process named *parallelJobExecution* which implements the main logic of the orchestrator.

However, as soon as we consider the medium complexity solution, the complexity increases significantly due to the presence of more tasks and three different function orchestrators. In particular, we want to highlight how the sub-process named *retryExecutionLoop*.

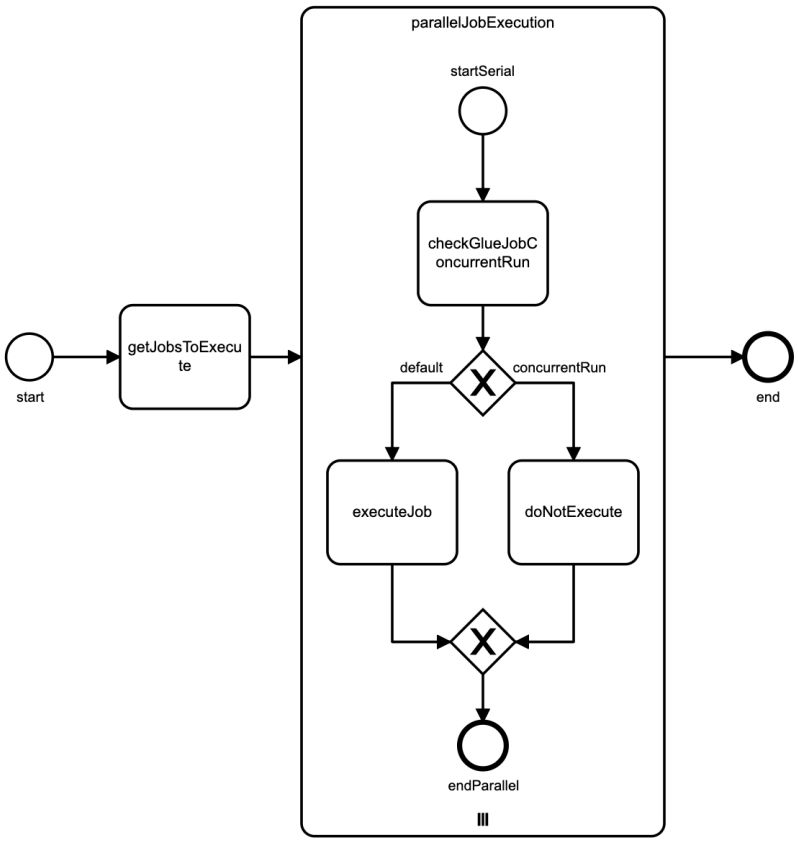

Figure B.1: Low complexity BPMN4FO function orchestrator model.

The *retryExecutionLoop* represent a loop that is executed at most three times, since when the execution of a job fails, it is re-attempted. In principle, this loop could be modeled by connecting the final state to the first one with a choice state, if we use directly the tools offered by cloud vendors to model function orchestrators. However, since with BPMN4FO we are dealing with a standardized way of working, and the loop is represented differently by other solutions, in order to achieve standardization, a sub-process is required.

This leads in the end to the creation of another target-specific file for a function orchestrator that implements this logic. If done directly on the cloud provider platform, the same objective could be achieved without the need of another intermediate serverless function. This, however, does not change the final result since both solution accomplish the same objective. It is just a different implementation style.

For what concerns the high-complexity solution, the model is very similar to the medium complexity one, with the main difference that there is an additional sub-process that is in charge of splitting all the jobs that needs to be executed on a smalls sets.

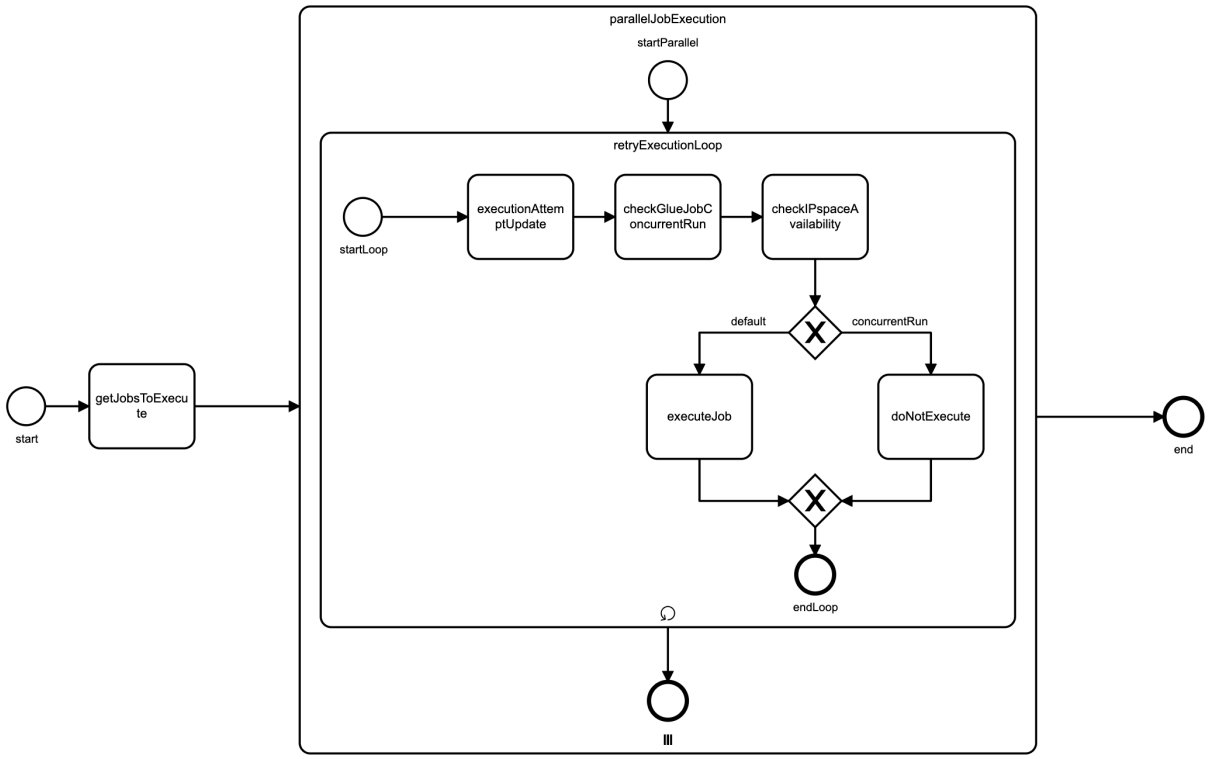

Figure B.2: Medium complexity BPMN4FO function orchestrator model.

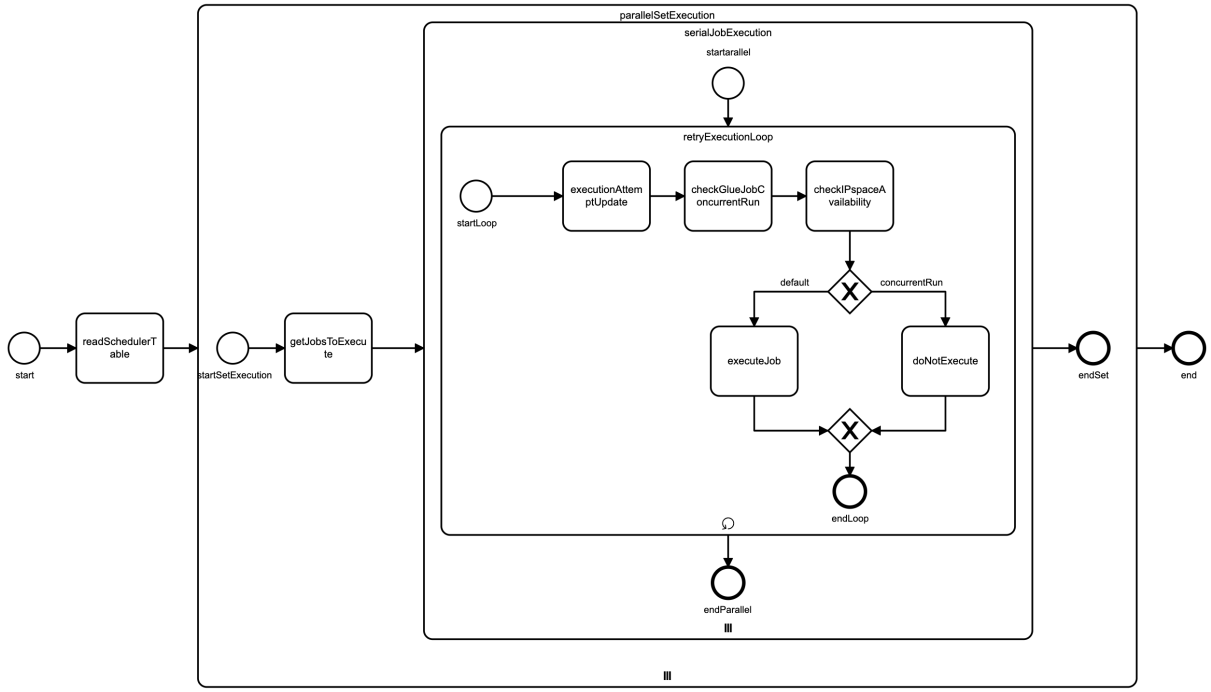

Figure B.3: High complexity BPMN4FO function orchestrator model.

B.2. RADON GMT Blueprints

It is finally interesting to observe how the deployment models differ from each other, and how they are connected to the BPMN Function Orchestrator models. In this section we present the deployment model for a specific target platform: the Amazon Web Services. Considering that in BMPN4FO a process or a sub-process represents a target function orchestrator and a task translated to a target serverless function, we obtained deployment models with two, three and four function orchestrators for the low, medium and high complexity solutions, respectively.

As we can observe in Figures B.4, B.5 and B.6, for each solution the number of Lambda Function nodes and the number of Step Function nodes (namely *AwsSFOrchestration*) are exactly the same as the number of tasks and processes or sub-processes present in the BPMN models shown in the previous section.

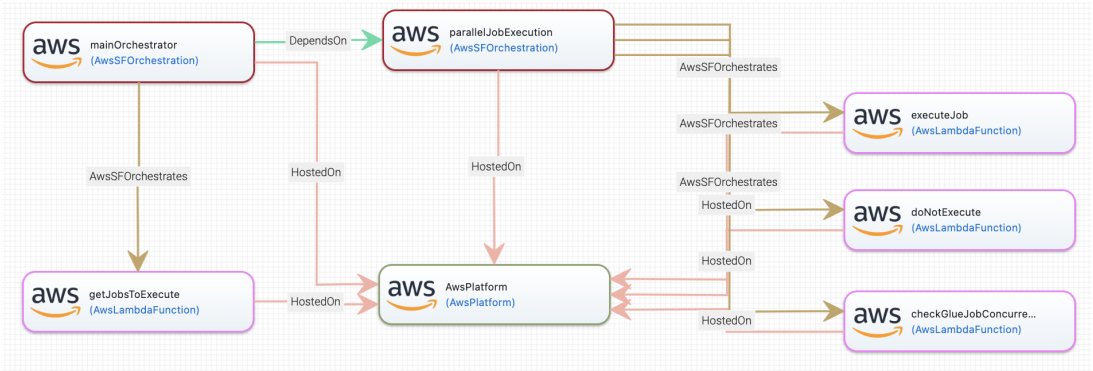

Figure B.4: Low complexity Eclipse Winery deployment model.

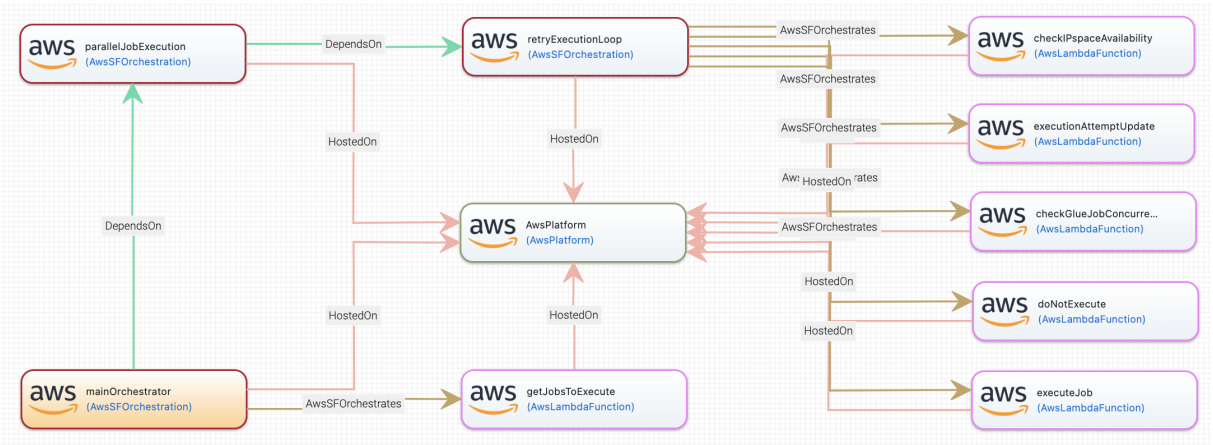

Figure B.5: Medium complexity Eclipse Winery deployment model.

In particular, each *AwsSFOrchestration* node contains (attached as an artifact) the ASL

file generated by the BPMN4FO tool. Then, each of these nodes is connected to the Step Function it orchestrates with the *AwsSFOrchestrates* relationship. Notice how, instead, when a Step Function contains a nested Step Function (a sub-process), it connects to it with the *DependsOn* relationship.

Then, each *AwsSFOrchestration* and *AwsLambdaFunction* node connects to the only *AwsPlatform* node with the *HostedOn* relationship. The *AwsPlatform* node contains the general deployment attributes of the target AWS account, such as the region (in our case, eu-west-1). Each *AwsLambdaFunction* node containers on the other hand some specific properties such as the memory size (we used for every node of this type 128 MB), the additional layers (if needed) and most importantly we attached the respective python file with the function handler.

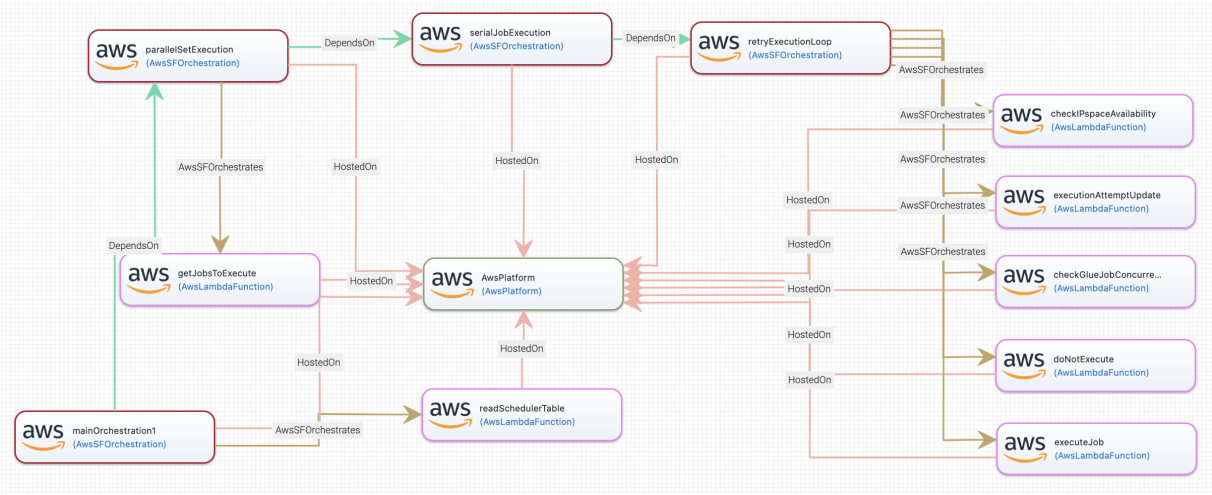

Figure B.6: High complexity Eclipse Winery deployment model.

Each of these deployment models have been exported as a CSAR file. Then, we deployed them using xOpera CLI. At the deployment stage, the CSAR file is extracted and all the nodes, relationship and artifacts are managed such that can be deployed on the target environment using Ansible IaC code, which is automatically generated. The target account is automatically retrieved in the Python virtual environment where xOpera and AWS CLI are both installed: the access keys to the target account are retrieved from the AWS CLI configuration file, which is defined using the *aws configure* command.

List of Figures

List of Tables

Acknowledgements

Even though this work is the result of my effort, there should be the names of all those who, in different ways, offered me guidance, assistance and encouragement, without whom all of this would not have been possible. I am very grateful to all of these people.

This thesis represents my final challenge at Politecnico di Milano, and it would have not been possible without the opportunity that my supervisor, Professor Damian Andrew Tamburri, gave me: the help and support was fundamental for this work, but most importantly I am very grateful to him since, from the very beginning, he trusted me.

I am very grateful also to NXP Semiconductors that welcomed me and allowed me to carry out this work in an avant-garde environment: in particular, I must thank all the team members I worked with, and everybody in the company that helped me.

Thank you all.