IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 30, 2021, accepted December 19, 2021, date of publication December 22, 2021,
date of current version January 7, 2022.

Digital Object Identifier 10.1109/ACCESS.2021.3137671

Developing Docker and Docker-Compose Specifications: A Developers' Survey

DAVID REIS[1], BRUNO PIEDADE[1], FILIPE F. CORREIA[1,2], JOÃO PEDRO DIAS[1,2], AND ADEMAR AGUIAR[1,2]

[1]Faculty of Engineering, University of Porto, 4200-465 Porto, Portugal
[2]INESC TEC, FEUP Campus, 4200-465 Porto, Portugal

Corresponding author: Filipe F. Correia (filipe.correia@fe.up.pt)

This work was supported in part by FEUP, in part by INESC-TEC, and in part by national funds through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT) under Project UIDB/50014/2020. Some of the early results of this research are part of David Reis' and Bruno Piedade's masters theses [1], [2].

ABSTRACT Cloud computing and Infrastructure-as-Code (IaC), supported by technologies such as Docker, have shaped how many software systems are built and deployed. Previous research has identified typical issues for some types of IaC specification but not why they come to be, or they have delved into collaboration aspects but not into technical ones. This work aims to characterize the *activities* around two particular kinds of IaC specification—Dockerfiles and *docker-compose.yml* files. We seek to know how they can be better supported and therefore study also what *approaches* and *tools* practitioners employ. We used an online questionnaire to gather data. The first part of the study reached 68 graduate students from a study program on informatics engineering, and the second one 120 professional software developers. The results show that most of the activities of the process of developing a *Dockerfile* are perceived as time-consuming, especially when the respondents are beginners with this technology. We also found that solving issues using trial-and-error approaches is very common and that many developers do not use ancillary tools to support the development of *Dockerfiles* and *docker-compose.yml* files.

INDEX TERMS Docker, docker-compose, orchestration, cloud computing, survey.

I. INTRODUCTION

The use of IaC [3], namely with container technologies such as Docker, LXC, and Kubernetes [4], has become commonplace, gaining prominence with the widespread of cloud computing and the increasing need for development and operations teams to collaborate efficiently [5]. Docker is a popular example of container technology that has become a *de facto* standard in software development [6] and played an important role in shifting the paradigm away from full-stack virtualization. Docker uses domain-specific languages for defining *contained environments* (*i.e.* containers) and *orchestration specifications* for containers [7].

Given the importance gained by these technologies [5], it is relevant to study how they are used, characterize current practice when creating and deploying Docker-based infrastructures and envision improved ways to support these practices as well as to approach any underlying difficulties. There are a number of differences between the development of IaC specifications and programs using general-purpose programming languages. While some aspects and activities of the process may be the same, others seem to be different—from testing to debugging, to the error-proneness and longer feedback loops [8].

Although there is a fair amount of ancillary tools for the development of infrastructure specifications [5], [9], few works try to empirically demonstrate the improvements they may bring to the development process [10]–[12]. In particular, there is scarce empirical evidence that the *issues* that many of these tools address are worth addressing and that the *approaches* that they prescribe are addressing such issues effectively.

In this work, we seek insights on how software professionals perceive the use they do of *Docker* and *Docker-Compose* and to generate new hypotheses of how best to address existing challenges.

In the remainder of this paper, Section II overviews works related to our study, Section III identifies the main goals of the research and its questions, and Section IV the methodology to answer them. Sections V and VI then describe, respectively, the data handling and analysis, offering this work's main insights. Section VII overviews the main threats to validity

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaowen Chu.

and, finally, Section VIII provides final remarks and outlines our research following up on these results.

II. RELATED WORK

There are still few empirical studies exploring the activities that underlie the development of specifications of containers and their orchestration, and even fewer proposals of how to address such problems. In this section, we review some of the most relevant works on both categories.

Guerriero *et al.* [8] conducted semi-structured interviews with 44 professionals and were able to identify **bad and best practices for IaC specifications**, and acknowledge that these are often conflicting and imply trade-offs. They also found that the available **tools hardly offer support for maintenance and evolution while ensuring quality** and that the most frequent **challenges pertain to testability and understandability**.

Dalla Palma *et al.* [13] identify a set of **46 metrics to evaluate the quality of IaC specifications** and propose a future study to empirically investigate the relationship between these metrics and the quality of specifications.

A study using data collected from Stack Overflow[1] was conducted by Haque *et al.* [14]. The researchers used topic modelling based on *latent Dirichlet allocation* and found that most of the questions asked by developers pertaining to the use of Docker fit into a few specific categories, in particular *application development*, *configuration*, *networking*, *basic concepts* and *debugging*. The study suggests that **these categories represent well the topics related to the use of Docker that are the most challenging to the Stack Overflow community**.

Cito *et al.* [15] evaluated a sample of 560 open-source projects from GitHub and revealed that **more than a third of the *Dockerfiles* found in those projects could not be built without errors**. The same study shows that **quality issues**, such as the lack of a `MAINTAINER` tag or the lack of version pinning of the base image and dependencies, **were also present in a considerable amount of *Dockerfiles***, including those in the top 100 repositories according to GitHub's star rating. These results show that errors and quality issues are not uncommon during the *Dockerfile* development process.

A study by Rahman *et al.* [16] analyzed more than 2K IaC scripts, in more than 12K commits, from four different organizations (Mirantis, Mozilla, Openstack, and Wikimedia Commons) and showed that **approximately half of the IaC specifications in their sample contained some kind of syntax and configuration-related defect at one of their revisions**, suggesting that this kind of defects may be more common in IaC than in non-IaC solutions. In other works, some of the same authors observe that **defects related with input configuration data are very frequent** [17], and identify five **development anti-patterns when developing IaC specifications** [18].

[1] Q&A website for developers, available at https://stackoverflow.com.

Ibrahim [19] studied more than 4K open-source Github projects that use Docker-Compose and concluded that **more than a quarter of the projects analyzed needlessly used Docker-Compose**, as they used one single component. They also found that the remaining multi-component specifications **mostly use the basic features of Docker-Compose**, ignoring advanced configurations such as the ones related to security and monitoring.

Some other works exist that attempt to improve the existing tools to develop and maintain IaC specifications, including the works by Harter *et al.* [20], Huang *et al.* [21], and Piedade *et al.* [12]. These works identify some **opportunities for improving the efficiency of developing Docker and Docker-Compose specifications** and for tools able to **reduce the time spent in the development of these specifications**.

Considering the reviewed literature, we observe that, as far as we could find, there are a few research questions that have been overlooked by the scientific community.

Most of these works do not study how the development of container and orchestration specifications is tackled from a technical standpoint—what activities play the most relevant part in the process, which ones are surrounded by more difficulties, and which practices developers employ to address them.

Finding which topics around the use of Docker raise the most concerns can help to steer further research [14]. Existing works show that it is common for faulty container specifications to be produced [15] and that many of these faults are due to incorrect configuration data [17]. The work by Guerriero *et al.* [8] is one of the few that gathers insights from the industry regarding how the development of IaC is perceived. There is still very little empirical evidence specifically on the technical activities that the development of containers entails and how variables such as developer experience influence them.

The anti-patterns identified by Rahman *et al.* [18] capture some understanding of practices that lead to defective container specifications. Still, they focus most on collaboration aspects and not on technical ones.

Studies on orchestration specifications are even fewer than those on container specifications. Ibrahim [19] found that some Docker-Compose features are rarely used but also did not seek to explain why that may be so.

In sum, there seem to exist severe limitations regarding the *maintenance and evolution while ensuring quality* of IaC specifications, as observed by Guerriero *et al.* [8] and supported by works such as those of Rahman *et al.* [18] and Cito *et al.* [15], even though the latter focuses on the produced artifacts and in their analysis and not so much on how developers perceive the process of developing them. From an empirical point of view, no study focuses on the IaC development activities that are most time-consuming for developers. This knowledge could guide future research on IaC development tools and processes, including those that are the most time-consuming for Docker and Docker-Compose.

III. GOALS OF THE STUDY

We hypothesize that the workflow of a developer configuring a Docker environment is often winding and based on trial-and-error. Creating a Docker container implies editing a specification (*e.g.*, a *Dockerfile*), build it into an image, instantiate the image into a container, understand if it is working as expected, identify the causes when it is not, and return to the first step to improve the specification. Next, often, developers need to *orchestrate* a set of created images, along with other services (*e.g.*, databases or message queues) writing another specification file (*e.g.*, a *docker-compose.yml*) to deploy the system to a host. This, too, is frequently an exploratory endeavor. Therefore, we postulate that there is a need to study and improve these development activities.

This study seeks to characterize the activities when specifying Docker containers and their orchestrations and contribute to understanding the most relevant concerns better. Namely, it aims to understand which parts of the process are more troublesome and take significant time to perform and what, in practice, the difficulties may be. We also try to identify approaches and tools developers use to support their work, particularly diagnosing and correcting common issues.

For a high-level illustration of what such a process may look like, Figure 1 shows one of the possible sequences of steps used when developing a *Dockerfile*. These steps and their associated activities are subjects of our study. Some of them warrant a close look; in particular, regarding the writing of the specification itself—the *Dockefile* in this example—we study activities such as the use of external documentation, the selection of an appropriate base image, or finding what the required dependencies are, among others.

This study answers three research questions in the context of working with *Dockerfiles* and *docker-compose.yml* files, as enumerated below. The first one is the primary question of our study (RQ1) and is the one that leads us to *characterize the activities* inherent to the development of container and orchestration specifications. Our secondary questions (RQ2 and RQ3) seek to identify possible *approaches* and *tools* for supporting such improvements.

RQ1. *Which activities are regarded as time-consuming?*
By looking for activities that are *time-consuming—i.e.*, we aim to find those where there is some developer efficiency to be gained and where the biggest needs may lie for improved approaches and tools.

RQ2. *Which approaches are used to diagnose and correct problems?*
Identifying some of the current approaches for existing issues can provide insights on how such approaches can be better supported or even which new alternatives could be proposed.

RQ3. *What role is played by ancillary software tools?*
Ancillary tools are those used *with* the containerization platforms—Docker and Docker-Compose in this research—but that are not a constituent part of them. We expect that identifying these tools may help to frame the needs most felt when developing specifications for these platforms, as well as possible gaps in the support that they provide out-of-the-box.

By searching for the answers to these RQs, we aim to provide evidence on current issues when developing these particular IaC specifications—i.e., for Docker and Docker-compose—to guide future research.

IV. METHODOLOGY

We used an online survey to reach software developers with some level of experience with Docker technologies. The survey allowed us to gather data from a number of developers, which would be difficult to match with other exploratory research methods, such as interviews, that require a considerable investment of time from the researchers. The checklist by Molléri *et al.* [22] was used to guide and assess the planning, execution, and reporting of the survey.

A. SAMPLING AND RECRUITMENT

Docker is one of the technologies supporting DevOps and operational agility; therefore, we distributed this questionnaire in communities and forums directed towards Docker, DevOps, and general-purpose programming. This methodology makes our sample a *convenience sample*, possibly with *referral-chain sampling* in those cases in which participants may have shared the questionnaire with colleagues [23]. To avoid oversampling specific subpopulations, we sought to reach participants from varied experiences and contexts by advertising the questionnaire through multiple methods. In particular, this included communities gathering through platforms such as *Slack*, *Discord* and *Reddit*, and social networking platforms such as *Linkedin*, *Twitter* and *Facebook*. We have also advertised the survey among the participants of the XP conference—*International Conference on Agile Software Development*. The strong presence of professionals in XP makes it a vehicle to potentially reach a significant number of respondents within our intended audience. Through these multiple means, we registered a total of 120 responses.

The first set of questions were used to characterize the participants. Some of the questions that followed focused specifically on *Dockerfiles*, and others on *docker-compose.yml* files. When analyzing the results of both types of questions, we considered only the respondents who reported having some experience creating or modifying that kind of file. This meant considering 119 responses for *Dockerfile*-related questions and 107 responses for *docker-compose.yml*-related questions.

To persuade the participation in the survey, we offered the potential respondents first-hand access to the results. We explained the goals of the study and the importance of participating in it through a short promotional video that we distributed with the questionnaire.[2]

[2]The video is available at https://vimeo.com/426652252

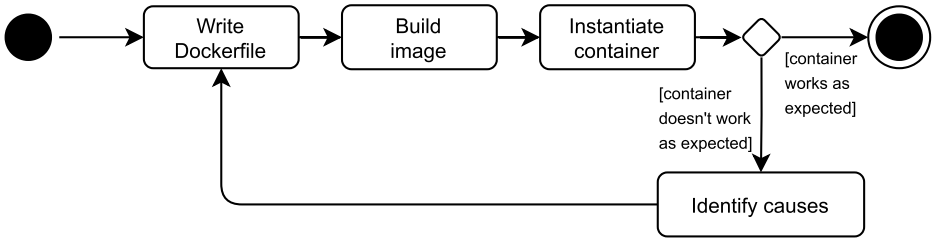


FIGURE 1. Motivational example of a workflow when developing a *Dockerfile*.

B. INSTRUMENT DESIGN

The questionnaire was designed and made available as an online form with a few questions to keep the response time below five minutes. They were written to be as precise and targeted as possible and as not to lead the respondents. Most questions are close-ended and capture numeric values (*e.g.*, years) or use a five-point Likert scale [24]; the use of open-ended questions was limited to just a few to identify issues and approaches that might not have been anticipated in the close-ended questions.

The questions are organized around four main groups, the latter three of which are directly aligned with the research questions introduced in Section III: a) *characterization of the participant*; b) *identification of existing issues*, c) *identification of existing approaches*, and d) *identification of ancillary tools and how they are used*.

An early version of the questionnaire was used in an academic setting (*cf.* Section VI-A). Before being used with professional software developers, the survey was piloted by two researchers with experience in Docker technologies. This allowed us to detect potential problems with the instructions, their usability, and if the questions themselves support assessing what is intended.

Both questionnaires are available as part of a replication package, published under a CC BY-SA license and identified by the Digital Object Identifier (DOI) `10.5281/zenodo.4660238` [25].

C. RESEARCH VARIABLES

We statistically analyze the answers to all close-ended questions. The questions used to characterize the respondents allow us to contextualize the results since only developers with some Docker experience are expected to be able to provide informed answers. The *years of experience* with *Docker* and with *Docker-Compose* are *independent variables* in our study. They are used to segment the results and gain insights over how the issues and the employed approaches will vary according to expertise.

The *dependent variables* are the attitudes of respondents towards statements that *a considerable amount of time is spent* in various development activities. We use these variables to identify which of these activities are the most time-consuming and to answer *RQ1*.

The few open questions are analyzed to identify unforeseen strategies and tools that may be used to tackle existing issues. By collecting responses about the general approaches taken to solve problems, we seek to answer *RQ2*, while questions about the tools used during development help answer *RQ3*.

V. DATA HANDLING

The questionnaire was built using Google Forms,[3] and the results were stored in a spreadsheet with shared access to the researchers. The name and other personal elements were not collected as mandatory fields but respondents were allowed to provide their email addresses in the last question (*General Comments*) if they were interested in receiving more details about the results of the study. We have used them exclusively to share the results of the study with the respondents, complying with data protection regulations.

The dataset and analysis scripts are also made available as part of the replication package (*cf.* Section IV-B).

VI. ANALYSIS OF THE RESULTS

A preliminary run of the study was conducted with students to test the survey and gather the first insights. The second run of the study was done with professionals to gather evidence that allows to answer our research questions. The steps we followed, the results obtained in each of them, and the sections that describe their analysis in this article, are summarized in Figure 2.

A. PRELIMINARY RUN WITH STUDENTS

The preliminary version of the questionnaire was distributed among graduate students of MSc in Informatics and Computing Engineering at the Faculty of Engineering of the University of Porto, resulting in 68 responses. Most of the respondents were beginners, having just worked on one or two *Dockerfiles* specifications on average.

In this first part of our study, the results showed respondents perceiving they spend much time understanding why a Docker container is not running as intended (54%) and that few participants (4.4%) use ancillary tools when working with Docker-Compose.

B. DEMOGRAPHICS

When distributing the survey targeting professionals, one of the primary concerns was to ensure that we reached developers from a wide diversity of contexts and varying

[3]Google Forms, https://www.google.com/forms/about/

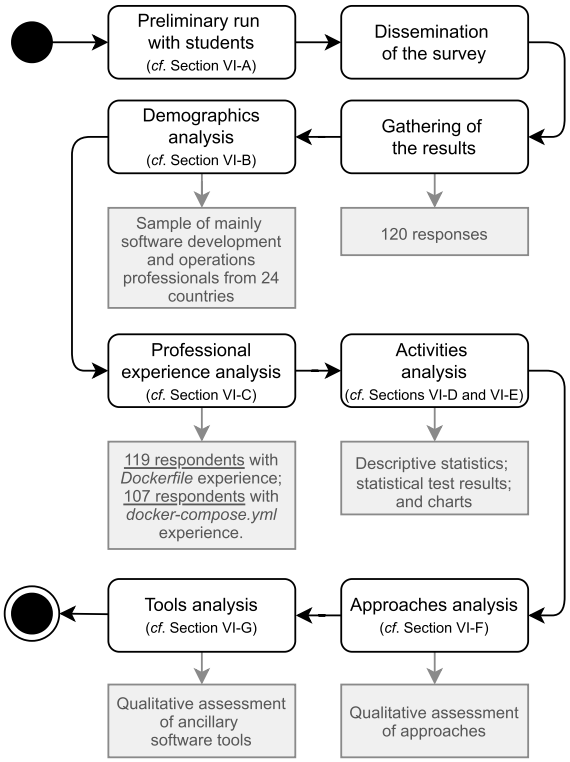


FIGURE 2. Results in each one of the steps taken in this study.

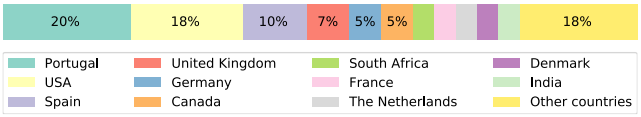


FIGURE 3. Distribution of the respondents by country.

degrees of experience. The answers to the demographics questions reveal that participants are from 24 different countries. The most represented countries are Portugal (20%), the USA (18%), and Spain (10%), as depicted in Figure 3.

The questions about professional background show that most participants work in the industry (90%) and some work in academia (12%). It also shows that most of them have responsibilities as software developers (87%), and many have responsibilities in operations (54%).

Even though we are working with a convenience sample, these results give us confidence that it is considerably diverse, which we expect will mitigate any potential bias that could be present due to the under-representation of some groups of developers.

C. PROFESSIONAL EXPERIENCE

To assess the professional experience of the participants with *Dockerfile* and *docker-compose.yml* development, three questions are performed twice, once for *Dockerfiles* and once for *docker-compose.yml* files. They allow to capture the independent variables in our study (*cf.* Section IV-C):

TABLE 1. Descriptive statistics of the experience in *Dockerfile* and *docker-compose.yml*.

Question	Dockerfile		docker-compose.yml	
	$\bar{x}$	σ	$\bar{x}$	σ
Q1	**3.53**	1.58	**3.01**	1.59
Q2	**3.29**	1.70	**2.76**	1.77
Q3	**3.31**	1.63	**2.84**	1.67

Q1 How much experience (in years) do you have working on projects *that had a* [*Dockerfile/docker-compose.yml file*]?

Q2 How much experience (in years) do you have working on projects *where you have used* [*Dockerfiles/docker-compose.yml files*] *created by others (colleagues or third parties)?*

Q3 How much experience (in years) do you have working on projects *where you created/updated a* [*Dockerfile/docker-compose.yml file*]?

We present a summary of the answers to these questions in Table 1, which shows that participants have some years of experience with *Docker* technologies, as expected. It is also observable that participants have, in general, slightly more experience working with *Dockerfiles* than with *docker-compose.yml files*.

D. TIME-CONSUMING ACTIVITIES IN Dockerfile DEVELOPMENT

To better understand the *Dockerfile* development process, we decompose it into eight different activities, namely:

A1 Reading Docker documentation.

A2 Finding out what are the right Dockerfile commands that I need.

A3 Finding out what parent image is the most suitable.

A4 Finding out what are the dependencies of my system that must be added to the docker image.

A5 Confirming if the resulting container is working as intended.

A6 Trying to understand why the resulting container is not working as intended.

A7 Finding out which commands are responsible for the container misbehavior.

A8 Rebuilding the image and re-running the container to confirm that it is working as intended.

Figure 4 shows the respondents' attitude towards the statement that each of these activities is time-consuming. It is worth noting that more than half of the participants agree that almost all activities are time-consuming. Activities **A4**, **A5**, **A6** and **A8** show at the top of the list, with A5, A6 and A8, in particular, being believed to be time-consuming by two thirds of the participants. These three activities have in common the fact that they are typical when diagnosing and fixing a misbehaving container.

Despite providing a general summary of the responses, the information presented in Figure 4 does not allow us

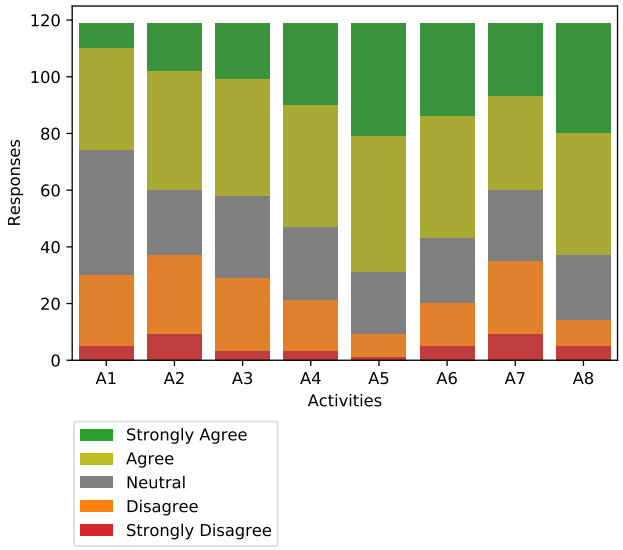


FIGURE 4. **Attitude from respondents towards a considerable amount of time being spent in each of the development activities.**

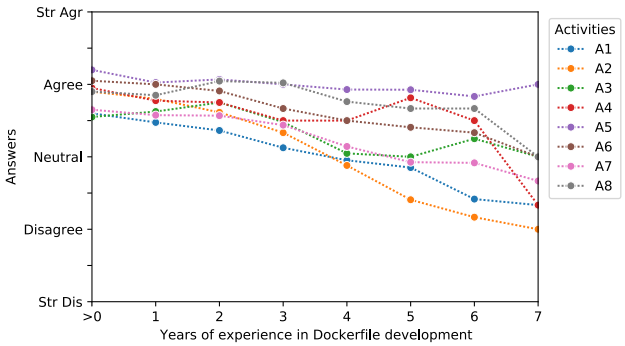


FIGURE 5. **Average attitude from respondents towards a considerable amount of time being spent in each of the development activities, by level of Dockerfile development experience.**

to see how the responses may be affected by professional experience. For this purpose, we try to measure the experience in *Dockerfile* development by asking specifically how many years of experience each participant has in creating or modifying *Dockerfiles*. We use this information to analyze the responses given by respondents with different years of experience. As we show in Figure 5, those with more experience in creating and modifying *Dockerfiles* tend to perceive most *Dockerfile* development activities as less time-consuming than respondents with less experience. This observation seems to hold for most of these activities, but not all. While some activities become less of a concern as experience is gained, others remain perceived as time-consuming, even among the most experienced professionals.

We looked beyond the average responses to understand how each activity is affected by the participants' experience. The respondents were divided into two groups, which we designate as the **inexperienced** group, consisting of participants who have three or fewer years of experience (65 responses), and as the **experienced** group, consisting of participants who have more than three years of experience with Docker (54 responses). By comparing the answers provided by both groups, we can see how the attitude towards the time consumed by each development activity changes with the participants' experience. Figure 6, indeed, shows that *experienced* developers tends to perceive most of the *Dockerfile* development activities as more time-consuming, given that there is a prevalence of *strongly agree* and *agree* answers in this group, and that the same cannot be observed for the *experienced* group.

To verify if the differences visible in Figure 6 are statistically significant, we have used a one-tailed Mann-Whitney U test [26]. The **null hypothesis** for this test is that the answers from both groups belong to the same distribution, while the **alternative hypothesis** is that the *inexperienced* group shows significantly more agreement towards the statements. The results of these tests and descriptive statistics are included in Table 2, and further show that developers with less experience tend to perceive most *Dockerfile* development activities as more time-consuming. Namely, the tests (with a 95% level of confidence; $\rho < 0.05$) show that *inexperienced* developers perceive the **A1**, **A2**, **A3**, **A6** and **A7** activities as significantly more time-consuming than *experienced* developers. For activities **A4**, **A5** and **A8**, no significant difference could be found, supporting the idea that these activities are not as dependent on experience and that they are perceived as time-consuming even as more experience is gained. However, we should note that there is a generally low agreement in the mean response of the *experienced* group for *all* items.

In sum, these insights help us answer the first research question, **RQ1**. They show that developers tend to perceive as time-consuming most of the *Dockerfile* development activities, and especially those related to *finding the dependencies of my system* (A4), *confirming if the resulting container is working as intended* (A5), *trying to understand why the resulting container is not working* (A6) and *rebuilding the image and re-running the container to confirm that it is working* (A8). The last three of these have in common being typical activities when debugging a misbehaving container. Furthermore, the less experienced developers seem, in general, more likely to find activities as time-consuming than experienced developers are. This suggests that developers tend to become more efficient as they progress through the learning curve of *Dockerfile* development, as would be expected. However, not all activities seem to be equally improved with experience. In particular, the time spent *finding the dependencies of my system* (A4), *confirming if the resulting container is working as intended* (A5), and *rebuilding the image and re-running the container to confirm that it is working* (A8) seem to be regarded as time-consuming even after gaining experience in this technology.

The most interesting opportunities for improvement may lie in the activities A4, A5, A6, and A8 since they are perceived as the most time-consuming globally. Possible

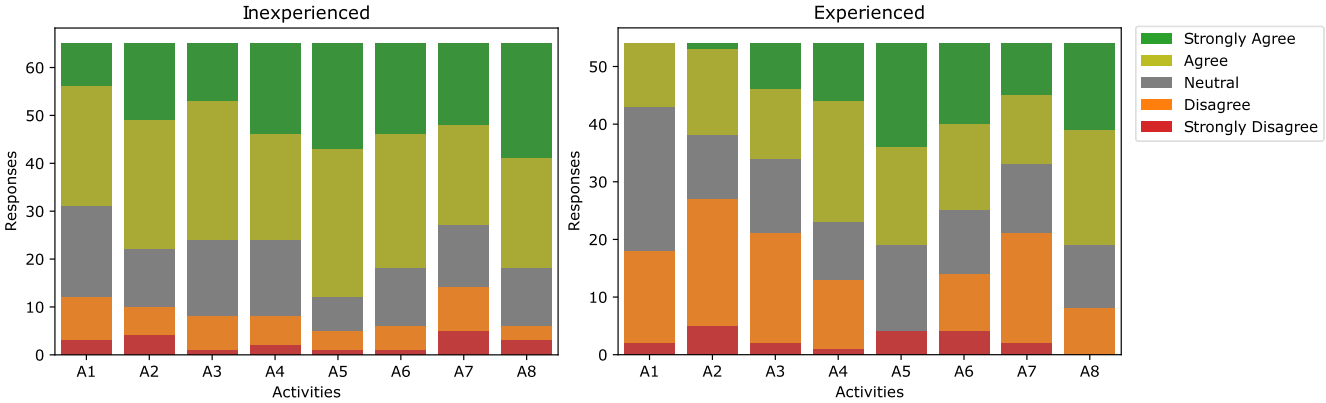


FIGURE 6. Attitude from respondents towards a considerable amount of time being spent in each of the development activities according to years of *Dockerfile* development experience.

TABLE 2. Mean, standard deviation and one-tailed Mann-Whitney U test for the answers to the likert items for each activity of *Dockerfile* development. The values are comprised in the scale between *strongly disagree* (1) and *strongly agree* (5).

Activity	Inexperienced		Experienced		Mann-Whitney U	
	$\bar{x}$	σ	$\bar{x}$	σ	U	ρ
A1	**3.43**	1.04	**2.83**	0.79	2371	$\underline{< \mathbf{0.01}}$
A2	**3.69**	1.12	**2.72**	1.03	2581	$\underline{< \mathbf{0.01}}$
A3	**3.68**	0.95	**3.09**	1.14	2282	$\underline{< \mathbf{0.01}}$
A4	**3.77**	1.06	**3.5**	1.08	1999	**0.09**
A5	**4.06**	0.9	**3.9**	0.95	1931	**0.16**
A6	**3.9**	0.96	**3.46**	1.26	2084	**0.03**
A7	**3.55**	1.23	**3.09**	1.22	2134	**0.02**
A8	**3.95**	1.07	**3.74**	1.09	1963	**0.12**

improvements to activity A6 are likely to benefit *inexperienced* developers more than *experienced* ones.

E. TIME-CONSUMING ACTIVITIES IN docker-compose.yml DEVELOPMENT

To evaluate the activities concerning the development of Docker-Compose specifications, we were interested in studying both those that pertained to their *writing* (W) and to their *reading* (R), as we suspected that these two perspectives pose different challenges to software developers.

We have considered the following activities concerning the *writing* of Docker-Compose specifications:

W1 Reading Docker documentation;

W2 Finding out what are the keys that I need;

W3 Finding out what images are available;

W4 Trying to understand why the services are not working as intended;

W5 (Re)starting the services to confirm that they are working as intended;

W6 Configuring the properties of each service (*e.g.*, port mapping, name, ...);

W7 Configuring the dependencies between the services (*e.g.*, depends_on);

W8 Configuring volumes and how they are attached to the services;

W9 Configuring networks and how they are connected to the services;

W10 Configuring configs and how they are accessed by the services;

W11 Configuring secrets and how they are accessed by the services.

Additionally, we have considered the following activities as concerning the *reading* of Docker-Compose specifications:

R1 Trying to understand what the services are;

R2 Trying to understand the dependencies between services (*e.g.*, depends_on);

R3 Trying to understand what volumes are used and how they are attached to the services;

R4 Trying to understand what networks are used and how they are connected to the services.

Figure 7 depicts the distribution of the answers to these questions. The answers seem to, in general, concentrate around the *neutral* sentiment. To more clearly understand the data, we have computed the mean, median and interquartile range, as shown in Table 3 and Table 4, which reinforced this perception (3) and showed a very slight skewness towards an *agreement* sentiment (4), as indicated by the mean and

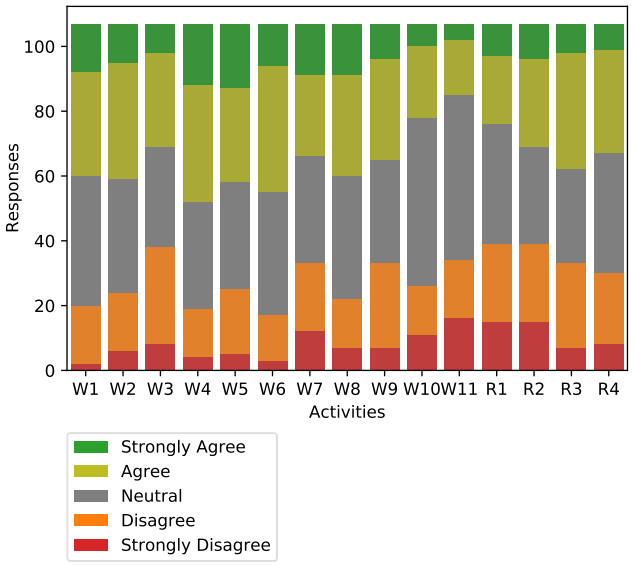


FIGURE 7. Attitude from respondents towards a considerable amount of time being spent in each of the development activities for writing and reading perspectives.

TABLE 3. ***Writing*** **activities of** ***Docker-Compose*** **development, and mean, median and IQR for the answers to the respective likert items. The values are comprised in the scale between** ***strongly disagree*** **(1) and** ***strongly agree*** **(5).**

Activity	$\overline{x}$	$\tilde{x}$	IQR
W1	3.37	3	1
W2	3.28	3	1
W3	3.01	3	2
W4	3.48	4	1
W5	3.36	3	1
W6	3.42	3	1
W7	3.11	3	2
W8	3.32	3	1
W9	3.12	3	2
W10	2.99	3	1
W11	2.79	3	1

TABLE 4. ***Reading*** **activities of** ***Docker-Compose*** **development, and mean, median and IQR for the answers to the respective likert items. The values are comprised in the scale between** ***strongly disagree*** **(1) and** ***strongly agree*** **(5).**

Activity	$\overline{x}$	$\tilde{x}$	IQR
R1	2.98	3	2
R2	2.95	3	2
R3	3.13	3	2
R4	3.09	3	2

median. The only exception is the result of activity **W4**, which is directly related to *debugging*, and point to an agreement towards this being a time-consuming activity. By analyzing the IQR, we can also conclude that some of the answers appear not to be consensual among respondents, especially those related to reading activities.

The general summary of the responses shown in Figure 7 does not allow us to see how professional experience

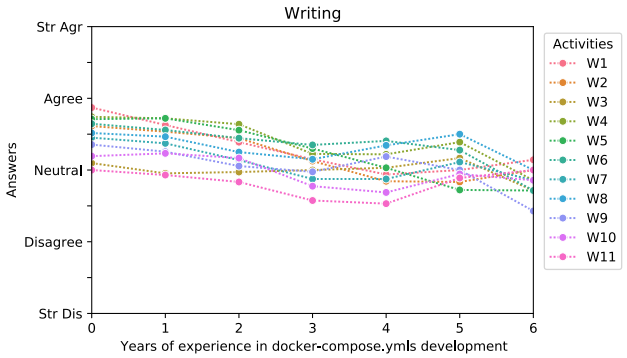


FIGURE 8. Attitude from respondents towards a considerable amount of time being spent in each of the *writing* development activities, by level of Docker-Compose development experience.

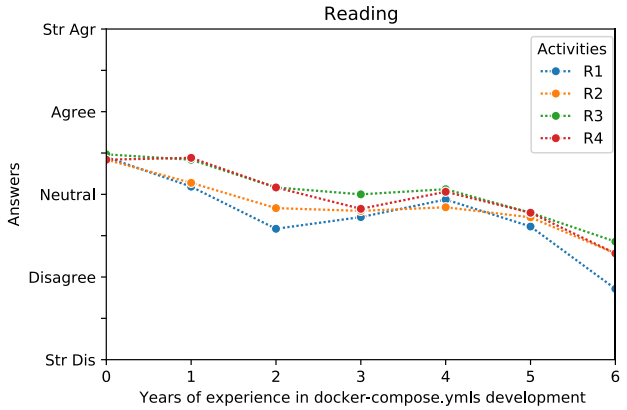


FIGURE 9. Attitude from respondents towards a considerable amount of time being spent in each of the *reading* development activities, by level of Docker-Compose development experience.

may affect the responses. To better understand the results, we specifically searched for correlations between experience and the attitude towards each activity. Figure 8 and Figure 9 display the evolution of the mean perception by years of experience of manipulating *docker-compose.yml* files, respectively for *writing* and for *reading* activities. We can observe that those with more experience in writing or reading *docker-compose.yml* files tend to perceive most activities as less time-consuming than respondents with less experience.

We then adopted a similar approach to the one that we used for analyzing the items related to *Dockerfiles* to understand in more detail how *each activity* is affected by the participants' experience. We have split the sample into two distinct groups—**inexperienced** (48 responses) and **experienced** (59 responses).

The distribution of answers between these two groups is displayed in Figure 10 and the statistical results in Table 5.

The data for the *inexperienced* group, as shown by the figure, supports a different reading than the one possible by inspecting the responses globally. On average, *inexperienced* respondents seem to agree that the writing-related activities **W1**, **W2**, **W4**, **W5**, and **W6** are time-consuming. In particular, *debugging* is not only regarded as time-consuming by the

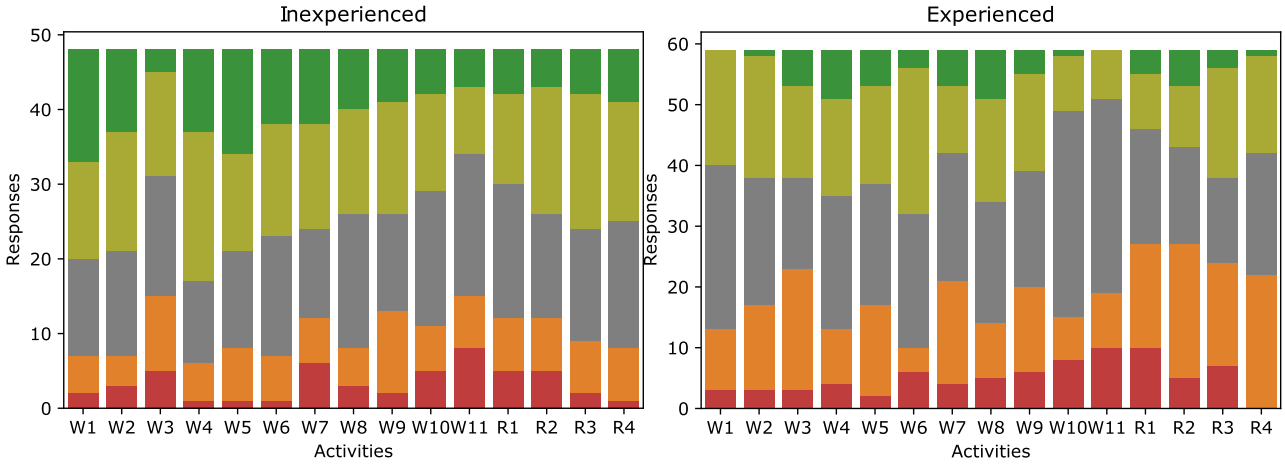


FIGURE 10. Attitude from respondents towards a considerable amount of time being spent in each of the development activities, by level of Docker-Compose development experience.

respondents globally, as observed before but the *debugging*-related activities W4 and W5 are especially regarded as time-consuming by *inexperienced* developers. The data also allows us to identify an agreement by *inexperienced* developers that the activities related to *documentation* are time-consuming (W1 and W2), which makes sense when considering the higher reliance on such resources that is expected from those with less experience. The configuration of properties related to services (W6) is also reported as time-consuming. This is not surprising as services are the core building block of orchestration specifications. Therefore, they are used more frequently, and the element for which the most properties are available. As to the reading-related activities, *inexperienced* respondents identify as the most time-consuming the understanding of how volumes are being used (**R3**).

The one-tailed Mann-Whitney U test results included in Table 5 allow us to assess if the differences between the two groups are statistically significant. The **null hypothesis** for this test is that the answers from both groups belong to the same distribution, while the **alternative hypothesis** is that the *inexperienced* group shows significantly more agreement towards the statements. The tests (with a 95% level of confidence; $\rho < 0.05$) suggest that experience significantly affects the perception of how time-consuming are *some* of the writing-related activities (**W1**, **W2**, **W4**, **W5**, **W7** and **W10**) and *all* of the reading-related activities (**R1**, **R2**, **R3** and **R4**). These results are in-line with the conclusions that we took when analysing the *Dockerfile*-related items, suggesting that *experienced* developers perceive many of the activities as less time-consuming as they become more proficient with the technology. In fact, the most likely sentiment in general for *experienced* developers is slightly skewed towards *disagreement*.

In sum, these results help us answer our first research question, **RQ1**, in light of the level of experience. Most activities seem to become easier with experience, and all the perceptions of activities being time-consuming come from the developers with less experience. In particular, these developers report requiring considerable time for *configuring properties related to services* (W6), and for activities related to *debugging* (W4 and W5) and to *documentation* (W1 and W2).

This suggests that *inexperienced* developers may be the ones to benefit the most from improvements to these activities and that the most relevant opportunities for improvement may lie in better supporting their work, and in particular in what regards the *debugging* of Docker-Compose specifications. This is consistent with the view [27] that less experienced developers tend to follow a process of *trial-and-error* when developing code artifacts.

F. APPROACHES FOR DIAGNOSING AND CORRECTING PROBLEMS

The survey items that support answering **RQ2** are open text questions, and they ask what steps or strategies are being followed by respondents to diagnose and fix bugs in the creation of *Dockerfiles* and *docker-compose.yml files*.

We could find no significant differences in the approaches described by experienced participants compared to the less experienced participants.

Most of the approaches that were mentioned for solving problems in *Dockerfiles* alluded to some form of trial-and-error, searching Web resources, or entering the container to execute commands which may help to diagnose the issue manually. Some respondents also mention structured approaches, such as continuous integration pipelines and end-to-end testing.

Regarding Docker-Compose, the respondents mention similar tactics, often referencing the analysis of the output logs and execution of commands within the running containers. Some describe more systematic approaches such as isolating services individually, making sure they work as expected, and afterward test their dependencies, as well as more advanced strategies to test network connectivity.

TABLE 5. Mean, standard deviation and one-tailed Mann-Whitney U test for the answers to the likert items for each activity of *docker-compose.yml* development. The values are comprised in the scale between *strongly disagree* (1) and *strongly agree* (5).

Activity	Inexperienced		Experienced		Mann-Whitney U	
	$\bar{x}$	σ	$\bar{x}$	σ	U	ρ
W1	**3.71**	1.15	**3.1**	0.74	927	**< 0.01**
W2	**3.58**	1.13	**3.03**	0.93	991	**< 0.01**
W3	**3**	1.09	**3.02**	1.11	1402	**0.46**
W4	**3.73**	1.01	**3.27**	1.06	1062	**0.01**
W5	**3.67**	1.12	**3.12**	1.08	1037	**< 0.01**
W6	**3.56**	1.03	**3.31**	0.9	1224	**0.1**
W7	**3.33**	1.29	**2.93**	1.13	1130	**0.03**
W8	**3.4**	1.09	**3.25**	1.11	1314	**0.26**
W9	**3.29**	1.11	**2.98**	1.08	1202	**0.08**
W10	**3.19**	1.14	**2.83**	0.87	1132	**0.03**
W11	**2.92**	1.2	**2.68**	0.88	1251	**0.14**
R1	**3.15**	1.15	**2.66**	1.14	1073	**0.01**
R2	**3.21**	1.15	**2.75**	1.23	1092	**0.02**
R3	**3.4**	1.03	**2.92**	1.09	1070	**0.01**
R4	**3.44**	0.99	**2.81**	1.03	973	**< 0.01**

These approaches all imply leaving the environment where the specification is being written to gather feedback about existing problems by collecting data and conducting experiments. Therefore, we formulate the hypothesis that an approach, or family of approaches, that allows the same kind of feedback to be gathered within the same environment, and even possibly without abandoning the writing of the specification, could help streamline the entire process.

G. ANCILLARY SOFTWARE TOOLS

We also ask participants if they use any plugins or tools other than a general-purpose IDE to develop *Dockerfiles* and *docker-compose.yml* files.

According to the responses that we have collected, experience does not seem to influence the use of plugins or tools, as experienced and inexperienced participants provided similar comments in this matter. In fact, most of the participants report **not** using any ancillary tools during the development of *Dockerfiles* (78%) and of *docker-compose.yml* files (83%). The other participants report using just a few types of tools: *syntax highlighters*, *linters*, and tools for *managing local images and containers*. Answering the research question **RQ3**, these results show that ancillary tools aimed towards Docker development exist but are not widely used, and the most common ones only offer basic and simple features and are often based on static analysis.

Considering that many of the studied activities are perceived as time-consuming—especially those related to the creation of *Dockerfiles* (*cf.* Section VI-D) but also some related to the creation of *docker-compose.yml* files when done by those will less experience (*cf.* Section VI-E)—we think that there is a real need for new tools that improve the process of writing these specifications. However, to provide more sophisticated tools than the ones currently available, we believe that they may need to go beyond what is possible with static analysis alone.

VII. THREATS TO VALIDITY AND LIMITATIONS

We have considered a few possible threats to the validity of our results when designing the research methodology, as described in Section IV. This section discusses some of our efforts to mitigate the main threats and identify those that we can not entirely discard.

Questionnaire design. Online questionnaires are particularly suitable for simple and generally closed questions, but this could limit our capability to explain some of the results. To mitigate this issue, we considered a very restricted set of open-ended questions in the survey that allowed us to obtain rich free-text responses without taking so much time from the participants.

Clarity of the questionnaire. We sought to write the questionnaire clearly and directly. However, there is always the chance that some participants might be confused—or even have a different interpretation—of the questions than the one we intended. To mitigate this issue, we have first run an initial version of the questionnaire with students and performed a trial run with two researchers, as described in Section IV-B. This allowed us to identify and fix potentially dubious questions before running the survey with professionals. Notwithstanding, we designed the final item of the questionnaire as an open-ended question that sought to gather general comments. One of the respondents used it to mention that the questionnaire as a whole did not make sense and another that some questions were not clear enough. These two participants constitute a small fraction of our sample (1.7%), and we are confident that *most* respondents had no problems understanding the

questionnaire, but this is a threat that we can not fully discard.

Honesty of the respondents. We cannot guarantee that the participants did not answer the questionnaire more than once or in an untruthful way. Participants were not offered any form of material gain to answer the questionnaire, and we believe that they had no incentive for the responses to be fabricated or dishonest. Therefore, we believe that that is not the case for the vast majority of the responses that we have collected, but this is a threat that we can not consider to be completely discarded.

Assessing developer experience. The questions on *professional experience* asked the participants the *number of years* working with *Dockerfiles* and *docker-compose.yml* files. This may not be a reliable way to assess experience since it depends on the participants' memory, since that working with a given technology for a long time may not always equate to having used more of its potential or knowing more of its intricacies. However, since we found consistent and significant correlations between the answers to these experience-related questions and the other questions in the questionnaire, we are confident that we could measure experience with an adequate level of precision.

Representativeness of the sample. Collecting results through an online questionnaire allowed us to reach a wide and diverse audience. Notwithstanding, it is not possible to guarantee that a *convenience sample* such as ours is representative of the entire population of software developers with some level of experience developing *Dockerfiles* or *docker-compose.yml* files. For this reason, the specific point estimates that we consider in this study cannot be considered representative of the entire population, but we expect the main conclusions that stem from this work to reflect the current state of practice [23].

Generalization to other IaC platforms. We chose specifically Docker and Docker-Compose as the targets of our study since this allows us to focus our questions on specific software artifacts and workflows. We are necessarily limiting the scope of the results to these specific technologies, and we can not assume that they generalize to other kinds of IaC platforms that rely on different specification languages. Notwithstanding, these results can still have implications for a very relevant number of professionals. The prevalence of Docker for creating and managing software containers makes us confident of the significance of our results on the use of *Dockerfiles*. On the other hand, Docker-Compose is hardly one of the most commonly used orchestration solutions and is used mostly for small service stacks, prototyping purposes, or confined to development environments. For this reason, our results about the of *docker-compose.yml* files may not be as significant.

VIII. CONCLUSION

The study presented in this article tries to reach insights on the development of container and orchestration specifications. We can say that it complements the interviews conducted by Guerriero *et al.* [8], given that those address IaC in general. At the same time, our work characterizes particular activities when developing specifications for two IaC technologies in particular—Docker and Docker-Compose.

Our results suggest that the *Dockerfile* development process does present challenging activities, and developers tend to perceive most of them as time-consuming. *Finding the dependencies* of a system and *debugging* seem to be the activities that would most benefit from improvements. In what concerns Docker-Compose, while the results were not as relevant, *debugging* activities also seem to be one of the most common concerns, especially for inexperienced developers. Lastly, we observed that most developers do not use ancillary tools in their development process.

This work opens directions for future research, which may focus on understanding our findings further or proposing new ways to improve some of the activities. A first study could be a usability evaluation, where participants perform *Dockerfile* and *docker-compose.yml* development tasks and are observed to identify the major bottlenecks of the process, using methods such as task analysis or cognitive walkthrough. A follow-up one could propose new approaches or environments to develop Docker and Docker-Compose specifications to address the most time-consuming activities identified in this work and empirically demonstrate their merits and liabilities. Notions such as *liveness* [28], and of *live software development* [11], [29], [30] may play a relevant role in designing such environments, given their goal of proactively bringing feedback to users or even making the environment automatically act upon this feedback). We also consider that environments that enable visual programming and leverage model-driven engineering techniques can have an important role, especially for developers with less experience [11], [12], [31].

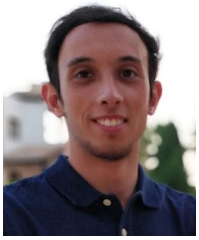
ACKNOWLEDGMENT

The authors would like to thank their respondents for their willingness to participate in this study. Some of the early results of this research are part of David Reis' and Bruno Piedade's masters theses [1], [2].

REFERENCES

[1] D. Reis, "Live Docker containers," M.S. thesis, Dept. Eng., Univ. Porto, Porto, Portugal, 2020.
[2] B. Piedade, "Visual programming language for orchestration with Docker," M.S. thesis, Dept. Eng., Univ. Porto, Porto, Portugal, 2020.
[3] M. Artac, T. Borovssak, E. Di Nitto, M. Guerriero, and D. A. Tamburri, "DevOps: Introducing infrastructure-as-code," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion (ICSE-C)*, May 2017, pp. 497–498.
[4] D. Bernstein, "Containers and cloud: From LXC to Docker to kubernetes," *IEEE Cloud Comput.*, vol. 1, no. 3, pp. 81–84, Sep. 2014.
[5] J. Cito, P. Leitner, T. Fritz, and H. C. Gall, "The making of cloud applications: An empirical study on software development for the cloud," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, Aug. 2015, pp. 393–403.

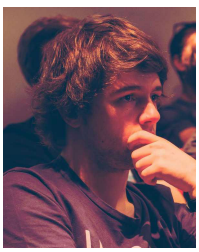
[6] Y. Zhang, G. Yin, T. Wang, Y. Yu, and H. Wang, "An insight into the impact of dockerfile evolutionary trajectories on quality and latency," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 138–143.
[7] R. Smith, *Docker Orchestration*. Birmingham, U.K.: Packt, 2017.
[8] M. Guerriero, M. Garriga, D. A. Tamburri, and F. Palomba, "Adoption, support, and challenges of infrastructure-as-code: Insights from industry," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2019, pp. 580–589.
[9] A. Rahman, R. Mahdavi-Hezaveh, and L. Williams, "A systematic mapping study of infrastructure as code research," *Inf. Softw. Technol.*, vol. 108, pp. 65–77, Apr. 2019.
[10] D. Weerasiri, M. C. Barukh, B. Benatallah, and C. Jian, "CloudMap: A visual notation for representing and managing cloud resources," in *Proc. 28th Int. Conf. (CAiSE)*, 2016, pp. 427–443.
[11] P. Lourenço, J. Dias, A. Aguiar, and H. Ferreira, "CloudCity: A live environment for the management of cloud infrastructures," in *Proc. 14th Int. Conf. Eval. Novel Approaches Softw. Eng.*, 2019, pp. 27–36.
[12] B. Piedade, J. P. Dias, and F. F. Correia, "An empirical study on visual programming Docker compose configurations," in *Proc. 23rd ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst., Companion*, Oct. 2020, pp. 24–34.
[13] S. Dalla Palma, D. Di Nucci, F. Palomba, and D. A. Tamburri, "Toward a catalog of software quality metrics for infrastructure code," *J. Syst. Softw.*, vol. 170, Dec. 2020, Art. no. 110726.
[14] M. U. Haque, L. H. Iwaya, and M. A. Babar, "Challenges in Docker development: A large-scale study using stack overflow," in *Proc. 14th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Oct. 2020, pp. 1–11.
[15] J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall, "An empirical analysis of the Docker container ecosystem on GitHub," in *Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories (MSR)*, May 2017, pp. 323–333.
[16] A. Rahman, S. Elder, F. H. Shezan, V. Frost, J. Stallings, and L. Williams, "Bugs in infrastructure as code," 2018, *arXiv:1809.07937*.
[17] A. Rahman, E. Farhana, C. Parnin, and L. Williams, "Gang of eight: A defect taxonomy for infrastructure as code scripts," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, vol. 20, Jun. 2020, pp. 752–764.
[18] A. Rahman, E. Farhana, and L. Williams, "The 'as code' activities: Development anti-patterns for infrastructure as code," *Empirical Softw. Eng.*, vol. 25, no. 5, pp. 3430–3467, Sep. 2020.
[19] H. Ibrahim, "A study of the use of Docker compose and dockerhub images," M.S. thesis, School Comput., Queen's Univ., Kingston, ON, Canada, 2019.
[20] T. Harter, B. Salmon, R. Liu, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Slacker: Fast distribution with lazy Docker containers," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*, 2016, pp. 181–195.
[21] Z. Huang, S. Wu, S. Jiang, and H. Jin, "FastBuild: Accelerating Docker image building for efficient development and deployment of container," in *Proc. 35th Symp. Mass Storage Syst. Technol. (MSST)*, May 2019, pp. 28–37.
[22] J. S. Molléri, K. Petersen, and E. Mendes, "An empirically evaluated checklist for surveys in software engineering," *Inf. Softw. Technol.*, vol. 119, Mar. 2020, Art. no. 106240.
[23] S. Baltes and P. Ralph, "Sampling in software engineering research: A critical review and guidelines," 2020, *arXiv:2002.07764*.
[24] S. Jamieson, "Likert scales: How to (ab)use them," *Med. Educ.*, vol. 38, no. 12, pp. 1217–1218, Dec. 2004.
[25] D. Reis and B. Piedade, *Davidreis97/Challenges_With_Docker: Replication Package*. Zenodo, Apr. 2021.
[26] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *Ann. Math. Statist.*, vol. 18, no. 1, pp. 50–60, Mar. 1947.
[27] A. Luxton-Reilly, E. Mcmillan, E. Stevenson, E. Tempero, and P. Denny, "Ladebug: An online tool to help novice programmers improve their debugging skills," in *Proc. 23rd Annu. ACM Conf. Innov. Technol. Comput. Sci. Educ.*, New York, NY, USA, Jul. 2018, pp. 159–164, doi: 10.1145/3197091.3197098.
[28] S. L. Tanimoto, "A perspective on the evolution of live programming," in *Proc. 1st Int. Workshop Live Program. (LIVE)*, May 2013, pp. 31–34.
[29] A. Aguiar, A. Restivo, F. F. Correia, H. S. Ferreira, and J. P. Dias, "Live software development: Tightening the feedback loops," in *Proc. Conf. Companion 3rd Int. Conf. Art, Sci., Eng. Program.*, Apr. 2019, pp. 1–6.
[30] D. Amaral, G. Domingues, J. P. Dias, H. S. Ferreira, A. Aguiar, R. Nóbrega, and F. F. Correia, "Live software development environment using virtual reality: A prototype and experiment," in *Proc. Int. Conf. Eval. Novel Approaches Softw. Eng.* Cham, Switzerland: Springer, 2019, pp. 83–107.
[31] A. Bhattacharjee, Y. Barve, A. Gokhale, and T. Kuroda, "CloudCAMP: Automating the deployment and management of cloud services," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, Jul. 2018, pp. 237–240.



DAVID REIS received the master's degree in informatics and computing engineering from the Faculty of Engineering of University of Porto (FEUP), Portugal. He has been working as a Software Engineer with Semasio, since 2020, where he develops and deploys highly concurrent and parallelized software components, and employing DevOps practices.

BRUNO PIEDADE received the master's degree in informatics and computing engineering from the Faculty of Engineering of University of Porto (FEUP), Portugal. He has a particular interest in service orchestrators, such as Docker-Compose, and on improving software development tools' experience.

FILIPE F. CORREIA is currently an Assistant Professor with the Faculty of Engineering of University of Porto (FEUP), Portugal. He is also a part of the Software Engineering Group, FEUP, and INESC TEC. His research interests include software engineering topics, namely, software architecture, design patterns, continuous delivery, agile methods, and live software development.



JOÃO PEDRO DIAS received the B.Sc. and M.Sc. degrees in informatics and computing engineering from the Faculty of Engineering of University of Porto (FEUP), where he is currently pursuing the Ph.D. degree in informatics engineering (holding an FCT grant). Since 2017, he has been an Invited Assistant Lecturer with FEUP. He works in the area of software engineering, with a special focus on the Internet-of-Things, with more than 25 published and indexed articles.

ADEMAR AGUIAR is currently an Associate Professor with the Faculty of Engineering of University of Porto (FEUP) and a Researcher with INESC TEC, Porto, with over more than 25 years of experience in software development, software architecture, and design (patterns, frameworks, and infrastructures), agile methods, wikis, and open collaboration tools.