Pós-Graduação em Ciência da Computação

Luís Henrique de Souza Melo

Using Docker to Assist Q&A Forum Users

Federal University of Pernambuco

posgraduacao@cin.ufpe.br

www.cin.ufpe.br/~posgraduacao

Recife

2019

Luís Henrique de Souza Melo

Using Docker to Assist Q&A Forum Users

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação na Universidade Federal de Pernambuco como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Concentration Area: Software Engineering

Advisor: Marcelo Bezerra d'Amorim

Recife

2019

Catálogo na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

M528u   Melo, Luís Henrique de Souza
          Using docker to assist Q&A forum users / Luís Henrique de Souza Melo. –
          2019.
          56 f.: il., fig., tab.

          Orientador: Marcelo Bezerra d'Amorim.
          Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn,
          Ciência da Computação, Recife, 2019.
          Inclui referências.

          1. Engenharia de software. 2. Docker. I. d'Amorim, Marcelo Bezerra
          (orientador). II. Título.

          005.1   CDD (23. ed.)   UFPE- MEI 2019-066

Luís Henrique de Souza Melo

"Using Docker to Assist Q&A Forum Users"

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação na Universidade Federal de Pernambuco como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 21/03/2019.

BANCA EXAMINADORA

Prof. Dr. Paulo Henrique Monteiro Borba
Centro de Informática / UFPE

Prof. Dr. Rohit Gheyi
Departamento de Sistemas e Computação / UFCG

Prof. Dr. Marcelo Bezerra d'Amorim
Centro de Informática / UFPE
(Orientador)

I dedicate this thesis to all my family, friends and professors who gave me the necessary support to get here.

ACKNOWLEDGEMENTS

I would like to express my thanks to everyone who helped me along my journey, notably:

- My parents, Antônio and Célia, for all the support and unconditional love, even on harsh situations.
- My fiancée Renata, for all the love, affection and support.
- My brothers, Antônio Jr. and Sérgio, for friendship and support.
- My cousin and best friend, Davi Souza, for being able to keep my mind away from studies once in a while.
- My undergraduate advisor (more like aunt), Gilka Barbosa, for her great influence in my C.S. career.
- My partners, Pedro Santos, Caio Masaharu, Marcos Azevedo, Augusto Santos and Rodrigo Barbosa, for all the support.
- My working colleagues, Jea(derson) Cândido, Igor Simões, Waldemar Pires and Davino Junior for the funny moments and hangouts.
- My advisor, Marcelo d'Amorim, for everything he taught me during this last couple of years.
- FACEPE, CAPES, and Bitcoin, for funding my studies.

ABSTRACT

Q&A forums are today an important tool to assist developers in programming tasks. Unfortunately, contributions to these forums are often unclear and incomplete as developers typically adopt a liberal style when writing their posts. This dissertation reports on a study to evaluate the feasibility of using Docker to address that problem. Docker is a virtualization solution that enables a developer to encapsulate an operating environment—that could show how to manifest or fix a problem—and transfer that environment to others. Our study is organized in two parts. We conducted a feasibility study to broadly assess willingness and effort required to adopt the technology. We also conducted two user studies to assess how well people works the idea. In summary, our results indicate that Docker is useful the most to support configuration-related posts of medium and high difficulty, which we found to be an important class of posts. We also noted that interest of the community on a tool we developed to support our experiments was high. We believe that these results provide early evidence indicating that the use of Docker to assist developers in Q&A forums should be encouraged in certain cases.

Keywords: DevOps. Docker. Q&A forums. Web frameworks.

RESUMO

Os fóruns de perguntas e respostas (Q&A) são hoje ferramentas importantes para auxiliar os desenvolvedores nas tarefas de programação. Infelizmente, as contribuições nesses fóruns geralmente são imprecisas e incompletas, uma vez que desenvolvedores adotam um estilo liberal ao escrever suas perguntas e respostas. Este trabalho reporta um estudo para avaliar a viabilidade de usar Docker para resolver este problema. Docker é uma solução de virtualização que permite o desenvolver encapsular um ambiente operacional—que poderia demonstrar um problema ou a solução em execução—e transferir este ambiente para outros. Nosso estudo está organizado em duas partes. Nós conduzimos um estudo de viabilidade para avaliar de forma ampla a disposição dos desenvolvedores e o esforço necessário para adotar a tecnologia de virtualização. Também realizamos dois estudos com usuários para avaliar a performance usuários trabalham esta idéia. Resumidamente, nossos resultados indicam que Docker é útil na maioria das questões relacionadas à configuração de dificuldade média e alta, que descobrimos ser uma categoria importante de posts. Também notamos a alta expectativa da comunidade em uma ferramenta que desenvolvemos para auxiliar nossos experimentos. Acreditamos que esses resultados fornecem uma evidência primária indicando que o uso de Docker para auxiliar os desenvolvedores em fóruns de perguntas e respostas deve ser encorajado em certos casos.

Palavras-chave: DevOps. Docker. Q&A forums. Web frameworks.

LIST OF FIGURES

Figure 1	– StackOverflow question number 7023052.	17
Figure 2	– Linux containers.	18
Figure 3	– Example dockerfile.	18
Figure 4	– File “app.py”. Issue at the left-side and fix at the right-side.	20
Figure 5	– File “Dockerfile”. It spawns Python app app.py.	20
Figure 6	– Distribution of general and configuration questions. Horizontal line indicates average value (22%) of configuration questions across frameworks.	25
Figure 7	– Distribution of configuration questions per framework.	26
Figure 8	– Answers for the survey.	28
Figure 9	– Difficulty levels per category (configuration).	32
Figure 10	– Students’ performance in preparing dockerfiles.	36
Figure 11	– FRISK homepage screenshot.	37
Figure 12	– FRISK editor screenshot.	37
Figure 13	– FRISK screenshot.	38
Figure 14	– File “index.js”.	40
Figure 15	– File “index.js” in FRISK editor.	41
Figure 16	– File “Dockerfile”. It spawns Express.js app index.js.	41
Figure 17	– FRISK toolbar. Arrow A indicates the Build button, arrow B indicates the Run button and arrow C indicates the link to the container port.	42

LIST OF TABLES

Table 1	– Stats extracted from GitHub server-side framework showcase [1]. Highlighted rows indicate the frameworks we selected.	23
Table 2	– Characterization of question kinds. Considering general questions, <i>Presentation</i> relates to the presentation of the data, <i>Database</i> questions are those related to data access, <i>API</i> questions asks for help on a framework function, and <i>Documentation</i> questions ask clarification on some concept/behavior of the framework. Considering configuration questions, <i>Versioning</i> refers to issues related to incompatibility of library versions, <i>Environment</i> refers to issues related to incorrect permissions or missing dependencies, <i>Misc. Files</i> refers to issues related to misconfigured files, <i>Missing Files</i> corresponds to missing files, and <i>Library</i> refers to problems with the setup of libraries in the framework.	24
Table 3	– Breakdown of problems found while generating dockerfiles. Column “ Σ -P*” indicates the total number of posts reproduced per framework. P1 = Unsupported. P2 = Lack of details. P3 = Conceptual. P4 = Clarification. P5 = User interaction. P6 = OS-specific.	30
Table 4	– Number of cases dockerfiles are identical (Same), Average size of dockerfiles (Size), and average similarity of dockerfiles (Sim.). Table 3 shows the absolute numbers of questions for each pair of framework and category.	33
Table 5	– Application artifacts (e.g., source and configurations files) modified in boilerplate code while preparing containers.	33
Table 6	– Data obtained from FRISK analytics.	43

LIST OF ACRONYMS

CSS	Cascading Style Sheets
JSON	JavaScript Object Notation
LOC	Lines of code
LAMP	Linux, Apache, MySQL and PHP
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
OS	Operating System
PWD	Play-With-Docker
Q&A	Question and Answer
UI	User Interface
URL	Uniform Resource Locator
XML	Extensible Markup Language

CONTENTS

1	INTRODUCTION	13
1.1	Research Methodology	13
1.2	Statement of Contributions	15
1.3	Outline	15
2	BACKGROUND	16
2.1	StackOverflow	16
2.2	Docker	16
2.2.1	Images and containers	18
2.3	Motivating Example	19
3	DATASET	21
3.1	Selection Methodology	21
3.1.1	Frameworks	21
3.1.2	Questions	21
3.2	Characterization of Questions	22
3.2.1	Popularity	25
3.2.2	Prevalence	25
4	FEASIBILITY STUDY	27
4.1	Adoption Resistance	27
4.2	Effort	29
5	USER STUDY	35
5.1	Students	35
5.2	Developers	36
5.2.1	FRISK	36
5.2.1.1	<i>User Interface</i>	36
5.2.1.2	<i>Design</i>	39
5.2.1.3	<i>Using FRISK</i>	40
5.2.2	Design	42
5.2.3	Results	43

6	DISCUSSION	45
6.1	Threats to Validity	47
6.1.1	External Validity	47
6.1.2	Internal Validity	47
6.1.3	Construct Validity	48
7	RELATED WORK	49
7.1	Educational tools and Collaborative IDEs	49
7.2	Mining repositories	49
8	CONCLUSIONS	51
	REFERENCES	53

1

INTRODUCTION

Question and Answer (Q&A) forums, such as StackOverflow, have become widely popular today. Unfortunately, it is not uncommon to find posts in Q&A forums with problematic instructions on how to reproduce issues [2; 3; 4]. For example, Terragni et al. [3] and Balog et al. [4] independently showed that code snippets often contain compilation errors and, more recently, Horton and Parnin [5] showed that 75.6% of the code snippets they analyzed from GitHub required non-trivial configuration-related changes to be executed (e.g., including missing dependencies).

This dissertation evaluates the extent to which virtualization technology can mitigate this problem. It reports on a study to assess the feasibility of using Docker [6] to assist reproduction of Q&A posts. Docker provides an infrastructure to build “containers”, which enable one to efficiently save and restore the state of a running environment. Intuitively, the use of Docker in Q&A forums would enable discussion based on concrete code artifacts rather than subjective textual descriptions. However, different factors could justify the impracticality of this idea, including inexperience with Docker, simplicity of posts, and concerns with security. We pose the following question:

- Would the adoption of Docker improve the experience of developers in Q&A forums?

1.1 Research Methodology

The study is organized in two parts. We first ran a feasibility study to broadly assess the potential of the idea. Then, we ran two user studies to evaluate the approach on more realistic grounds. The first user study has been conducted in a lab and involved students with no prior knowledge of the technology and the problems related to the posts they were requested to answer. The second user study involved StackOverflow developers using FRISK, the Docker integration tool we developed to support this experiment.

We conducted a feasibility study that covers two dimensions of observation: (i) Adoption Resistance and (ii) Effort. The first dimension assesses interest of the StackOverflow community in using containers for reproduction of Q&A posts. If there is strong evidence that interest in the approach is low, pursuing it brings low value. The second dimension evaluates cost of producing containers. Intuitively, the use of Docker in Q&A posts would unlikely pick up if cost was too high, even if resistance was low. We chose StackOverflow as the Q&A platform for its popularity and wide range of web frameworks it covers. We focused on web development in this study because, according to a recent survey [7], most StackOverflow users recognize themselves as web developers. The dataset for this study consists of sampled questions from the six most popular web frameworks according to a GitHub showcase [1] (see Table 1) and selected a hundred questions from each framework (600 in total) according to a selection criterion similar to those used in other studies. For this study, we pose the following questions:

- Adoption Resistance
  - **RQ1.** What are the perceptions of StackOverflow users towards the use of Docker to reproduce posts?
- Effort
  - **RQ2.** How often can developers dockerize posts?
  - **RQ3.** How hard is it for developers to dockerize posts?
  - **RQ4.** How big and similar are dockerfiles?

The second study is focused on the effort of using Docker for answering StackOverflow questions. We conducted two experiments with users to more directly assess feasibility of our proposal. The studies have different goals. [Students] We ran a preliminary study to understand how students without prior background in related technologies would perform in preparing containers for addressing Q&A posts. The event of most students performing poorly in the experiment would send us a signal that preparing better infrastructure to evaluate our proposal would not be worth the effort. We trained eight students, enrolled in a testing class, on Docker and web frameworks, and asked them to prepare containers for five existing StackOverflow questions of different difficulty levels–“Easy”, “Medium”, and “Hard”. To sum, most students were able to reproduce solutions to “Easy” posts within the time budget. Although students were optimistic with the approach and admitted they would perform better with more experience and time, we considered results non-negative (/inconclusive) and decided to run a study with real users. [Developers] To support this experiment, we implemented a tool, dubbed FRISK, that enables one to create containers from templates, save them on the cloud, and share those

containers through URLs that could be added to forum messages. Users can access FRISK anonymously through those URLs and restore a copy of the running environment. For this study, we pose the following questions:

- How difficult is it for developers with elementary training in Docker to dockerize Q&A posts?
- How popular is a tool to assist dockerfile creation?

1.2 Statement of Contributions

In summary, our results suggest that linking Docker containers to Q&A forums may be useful for certain kinds of posts.

- The categorization of a group of Q&A posts;
- A set of dockerized questions publicly available [8];
- A prototype tool to link Q&A community with Docker;
  - The tool is publicly accessible at <http://docker.lhsm.com.br>
- Publications:
  - Using Docker to Assist Q&A Forum Users, currently under submission;
  - Test Suite Parallelization in Open-Source Projects: a Study on its Usage and Impact [9];
  - Beware of the App! On the Vulnerability Surface of Smart Devices through their Companion Apps [10], by the time of writing, accepted at SafeThings '19 [11].

The last publication recently got some media spotlights in blogs like The Register [12], TechRadar [13], Hacker News [14], Naked Security [15], and Cibersecurity [16].

1.3 Outline

The rest of this work is structured as follows. Chapter 2 presents a background of Web Applications, StackOverflow and Docker, together with an example. Chapter 3 presents our methodology to select the subjects to conduct the study and describes our data set. Chapter 4 evaluates the feasibility study regarding the adoption resistance and effort in using Docker. Chapter 5 presents the user studies, including students and real-world developers. Chapter 6 discusses the results obtained during this study and presents the threats to the validity of this work. Chapter 7 discusses related works to this study. Finally, chapter 8 concludes this dissertation.

2

BACKGROUND

In this chapter, we explain the main concepts used in our work. Initially, in Section 2.1, we explain what is StackOverflow and how it holds knowledge. In Section 2.2, we explain what is Docker and how it works. Finally, in Section 2.3 we provide an overview of how one could use Docker to solve StackOverflow questions and define the scope of our study.

2.1 StackOverflow

StackOverflow is a Q&A forum that focuses on a wide range of topics in Computer Science that combines social media with technical problems to facilitate knowledge exchange between programmers. This knowledge is manifested in the form of questions and answers, that is often in a sequence of a code snippet and a text.

StackOverflow allows users to post, comment, search and edit questions, and answer posted questions. Most users are registered, allowing moderators and other users to track the questions, answers, and comments. Questions are usually composed by a title, a textual description of the problem that might contain a code snippet in the body, and tags to organize questions and highlight the main characteristics of the post (e.g., language, framework or environment). For a given question, it is possible to have multiple answers given by different users, which the original user asking the question can indicate one of the answers as correct. As the StackOverflow is composed of a community, other users can rate either the question and answers assuring the quality of the content. Figure 1 shows a snapshot of a StackOverflow question about Flask and the correct answer indicated by the original poster.

2.2 Docker

Docker is an open-source application that allows a developer to pack an application into a virtual environment called Linux Container with all its dependencies. A container is a

Figure 1: StackOverflow question number 7023052.

Configure Flask dev server to be visible across the network

296

I'm not sure if this is Flask specific, but when I run an app in dev mode (`http://localhost:5000`), I cannot access it from other machines on the network (with `http://[dev-host-ip]:5000`). With Rails in dev mode, for example, it works fine. I couldn't find any docs regarding the Flask dev server configuration. Any idea what should be configured to enable this?

python flask werkzeug

69

share improve this question

edited Oct 11 '17 at 0:19

asked Aug 11 '11 at 8:40

64.7k 12 171 187

11.6k 30 93 141

add a comment

7 Answers

active oldest votes

520

While this is possible, you should not use the Flask dev server in production. The Flask dev server is not designed to be particularly secure, stable, or efficient. See the docs on [deploying](#) for correct solutions.

Add a parameter to your `app.run()` . By default it runs on localhost, change it to `app.run(host='0.0.0.0')` to run on your machines IP address.

Documented on the Flask site under "Externally Visible Server" on the [Quickstart page](#):

Externally Visible Server

If you run the server you will notice that the server is only available from your own computer, not from any other in the network. This is the default because in debugging mode a user of the application can execute arbitrary Python code on your computer. If you have debug disabled or trust the users on your network, you can make the server publicly available.

Just change the call of the `run()` method to look like this:

```
app.run(host='0.0.0.0')
```

This tells your operating system to listen on a public IP.

share improve this answer

edited Oct 11 '17 at 0:19

answered Aug 11 '11 at 13:53

64.7k 12 171 187

5,748 1 17 14

virtualization technology that differs from conventional virtual machines. A container is able to run isolated processes without the need for virtualization of the hardware.

Figure 2 shows the concept of containers. Observe that the kernel is shared between the containers, it, therefore, uses fewer resources than virtual machines. All of the dependencies of the applications, from code to system libraries are included in these containers. Docker makes use of *images* to serve as templates for these containers. A Docker image is built upon a series

Figure 2: Linux containers.

of layers. Each represents an instruction (e.g., move a file or run a command). Each layer in the image is read-only. This architecture allows Docker to simplify the file sharing between images, and, in turn, can help reduce disk storage and speed up uploading and downloading of images [17]. The major difference between a Docker image and a container is that the last layer of a container is not read-only. All changes made to the running container (e.g., new log files, deleted and modified files) are written to this top writable layer [18].

2.2.1 Images and containers

One feature that might be the main cause of Docker's popularity is the possibility of describing the environment as a *code*. Dockerfile is a text document that contains all the necessary instructions a developer could call on the command line to assemble all dependencies and configurations. Each line of a dockerfile represents a layer in the final image.

Figure 3: Example dockerfile.

```
1 FROM ubuntu:19.04
2 LABEL maintainer=lhsm cin.ufpe.br"
3
4 #Install dependencies
5 RUN apt-get update
6 RUN apt-get install -y figlet
7
8 CMD echo "Hello, World!" | figlet
```

Figure 3 shows a dockerfile example to print in a sample message in a banner using the `figlet` [19] tool. The `FROM` instruction in a dockerfile defines the base system of an image.

Figure 3 shows an image based on Ubuntu Linux. The colon is used to specify the version of the base image. In this case, we use the build 19.04 of the Ubuntu Linux. The `LABEL` instruction is used to add metadata to an image. The `RUN` instruction will execute commands during the image building. The command is executed directly from within the container. The `CMD` instruction provides defaults for an executing container. In summary, this instruction is the command to be executed on container initialization.

Creating a Docker image is possible using the command `docker build -t <tag_name> <path>`. The `<tag_name>` argument gives a name to a newly built image. In the name, the user can reference the version of the image. Later, this same name and version could be used as an image base. The build process downloads the base image and creates a new layer for each instruction given in the dockerfile. The `<path>` parameter is the location of the dockerfile and necessary files to build the image. It is important to note that to speed up this process, Docker creates cache images for commands that do not involve in copying files into the image.

Running Docker containers is as simple as building the image. With the command `docker run <image_name>` a user can initialize a container from a specified image. This command creates a new writable layer on top of the image and saves every change in the container on that layer. When stopped, a user could restore the context of a container by restarting the container referencing the layer name.

2.3 Motivating Example

Let us consider the StackOverflow question shown in Figure 1 to illustrate the reproduction of a very simple post. In this case, a developer reports an issue that she cannot access the web application outside the local network. Figure 4 illustrates an example code to represent the issue (left side) and corresponding fix (right side). The symbol “|” highlights the changed line. This code is written in Flask, a popular web development framework based on Python. The intent is to handle an HTTP request and respond with a plain-text “Hello World” message. Unfortunately, running the problematic version of the code makes the web service invisible outside the local machine. The annotation `@app.route($apath)` in the code from Figure 4 indicates that the function `hello` is the handler of requests for the `$apath` URL. The variable `app` reflects the web application. The effect of calling `app.run()` is to make the web application listen to HTTP/S requests on a given address and port(s) [20]. When these arguments are not provided, the default value is `127.0.0.1` (i.e., localhost), port 5000. Unaware of this default setting, the user asked for help. The recommended change was to set the parameter `host` to `0.0.0.0`, which denotes all available *IPv4* addresses on the local machine. Figure 5

shows a dockerfile to spawn a web service for this Flask code. This script loads an Ubuntu image containing a recent version of Python, adds Flask to that image, creates a directory for the app, copy the file `app.py` from the host file-system to that directory, and finally spawns the Python app. Considering our example, the command `docker build -t example $adir` looks for a dockerfile in directory `$adir` and creates a corresponding image that can be referred by the name `example`. Running the command `docker run -p5000:5000 example` creates a container for that image mapping the port 5000, which is the default port for Flask applications to listen for requests, from the host to the same port on the container.

Figure 4: File “`app.py`”. Issue at the left-side and fix at the right-side.

<pre> 1 from flask import Flask 2 app = Flask(__name__) 3 @app.route('/') 4 def hello(): 5 return 'Hello World' 6 app.run()</pre>		<pre> 1 from flask import Flask 2 app = Flask(__name__) 3 @app.route('/') 4 def hello(): 5 return 'Hello World' 6 app.run(host='0.0.0.0')</pre>

It is worth noting that fixes are typically small, as in this particular example. However, in contrast to this example, 68.7% of the fixes we analyzed involve multiple artifacts, highlighting the limitations of tools like Repl.it [21] and JSFiddle [22] to address this problem. Our results also indicate that changes involve configuration files in 20.7% of the cases we analyzed. Note that Docker supports the creation of containers from scripts involving multiple files and also that it is possible to access configuration files, mentioned in StackOverflow posts, from Docker containers.

Figure 5: File “`Dockerfile`”. It spawns Python app `app.py`.

```

1 FROM python:2
2 # update image with necessary libraries to run Flask
3 RUN pip install flask
4 # copy app files
5 RUN mkdir app && cd app
6 WORKDIR /app
7 ADD app.py /app
8 # spawn the python (web service) app
9 CMD python app.py
```

3

DATASET

3.1 Selection Methodology

This chapter describes the methodology to select frameworks and questions associated with these frameworks.

3.1.1 Frameworks

We used GitHub Showcases to identify frameworks for analysis. Showcases is a GitHub service that groups projects by topics of general public interest and provides basic statistics for them. The web framework showcase [1] lists the most popular server-side web frameworks hosted on GitHub according to their number of stars and forks, which are popular metrics for measuring the popularity of hosted projects [23; 24; 25]. Note that this list is restricted to GitHub; it does not include some frameworks but it includes many highly-popular frameworks, according to alternative ranking websites [26; 27; 28]. Table 1 shows the frameworks grouped by the target programming language. Rows are sorted by the language, number of stars, and number of forks; in this order. Given that inspection of developer’s questions in Q&A forums is an activity that requires human cognizance, we restricted our analysis to a relatively small number of frameworks as to balance depth and breadth in our investigation. We selected frameworks from the listing that have more than 20K stars and more than 5K forks. Five frameworks have been selected according to this criteria. We additionally included `Meteor` as it has the highest number of stars amongst all framework. Table 1 shows our selection in gray color.

3.1.2 Questions

To identify questions, we used Data Explorer [29], a service provided by Stack Exchange [30], a network of Q&A forums. The query we used is publicly available [31]. We considered the following selection criteria. (i) We only selected questions tagged with the name

of the framework and with the name of the programming language we provided. We found that the framework name alone was insufficient to filter corresponding queries as posts related to different tools with similar names would also be captured. Beyer and Pinzger [32] also used tags as criteria for selecting questions. (ii) We only selected questions not marked as closed. For example, a question can be closed (by the community or the StackOverflow staff) because it appears to be a duplicate. Ahasanuzzaman et al. [33] performed a similar cleansing procedure when mining questions from StackOverflow. (iii) We only selected questions that the owner of the question selected a preferred answer. As we need humans to analyze questions, we set a bound of a hundred questions per framework. We prioritized questions in reverse order of their *scores* and extracted the first hundred entries. Similar procedure was adopted in other StackOverflow mining studies [34; 35; 36; 37; 38]. The score of a question is given by the difference between the up- and down-votes associated to all answers to that question. After inspecting the result sets obtained with this methodology, we realized that some questions, albeit tagged with framework labels, described issues unrelated to the framework itself but related to the used programming language. Considering Rails, for instance, nearly 20% of the questions returned in the original result set was related to Ruby (the language) as opposed to Rails (the framework). To address this issue and complete a set with a hundred questions, we manually inspected each question and manually removed language-specific questions and fetched the next questions in the result set.

3.2 Characterization of Questions

This chapter characterizes the questions we analyzed. It identifies the question kinds (i.e., what's their purpose), popularity scores (i.e., how well they are rated by users), and prevalence (i.e., how often they appear in posts).

Kinds. We used card sorting [39] to identify the categories of questions. In summary, the method consists of three steps: (i) preparation — in this step, a participant prepares cards with the title and link to the StackOverflow post, (ii) execution — in this step, participants give labels to the cards, and (iii) analysis — in this step, participants create hierarchies from the labels that emerged, solving potentially differences in terminology across participants. We apply this method in two iterations. In the first iteration the goal is to find broad categories that cover all cases. In the second iteration the goal is to discriminate the case within the broad categories. The cards were grouped into two broad categories: *general* and *configuration*. The category *general* includes general questions. For example, a question related to the presentation of the data or a clarification question about a particular framework feature. The category *configuration*

Table 1: Stats extracted from GitHub server-side framework showcase [1]. Highlighted rows indicate the frameworks we selected.

Language	Framework	Stars	Forks	Webpage
Crystal	Kemal	1,273	77	kemalcr.com
C#	Asp.Net Boilerplate	2,138	1,162	aspnetboilerplate.com
	Nancy	4,777	1,185	nancyfx.org
Go	Revel	7,732	1,081	revel.github.io
Java	Ninja	1,575	460	ninjaframework.org
	Spring	11,635	9,155	spring.io
JavaScript	Derby	4,178	240	derbyjs.com
	Express	29,136	5,335	expressjs.com
	Jhipster	5,749	1,291	jhipster.github.io
	Mean	9,714	2,912	mean.io
	Meteor	36,619	4,612	meteor.com
	Nodal	3,940	213	nodaljs.com
	Sails	16,189	1,657	sailsjs.com
Perl	Catalyst	239	96	catalystframework.org
	Mojolicious	1,778	424	mojolicious.org
Php	CakePHP	6,866	3,108	cakephp.org
	Laravel	28,436	9,392	laravel.com
	Symfony	13,538	5,255	symfony.com
Python	Django	22,822	9,224	djangoproject.com
	Flask	24,291	7,745	flask.pocoo.org
	Frappe	500	364	frappe.io
	Web2py	1,280	655	web2py.com
Ruby	Hanami	3,487	349	hanamirb.org
	Padrino	2,952	471	padrinorb.com
	Pakyow	722	59	pakyow.org
	Rails	33,910	13,793	rubyonrails.org
	Sinatra	8,553	1,599	sinatrarb.com
Scala	Play	8,754	3,035	playframework.com

includes questions related to the installation and configuration of the framework. For example, questions about misconfigurations of the environment where the framework was installed (e.g., insufficient privileges to access files and directories). It is very important to mention that the general questions we analyzed typically follow the pattern “how to implement X in framework Y?”. Considering configuration questions, many of the questions (40.15%) follow the pattern “how to fix this issue in framework Y?”.

We also categorized the questions within each of these two broad categories. For general questions, *Presentation* relates to the presentation of the data, *Database* questions are those related to data access, *API* questions ask for help on a framework function, and *Documentation* questions ask clarification on some concept/behavior of the framework. For configuration questions, *Versioning* refers to issues related to incompatibility of library versions, *Environment*

Table 2: Characterization of question kinds. Considering general questions, *Presentation* relates to the presentation of the data, *Database* questions are those related to data access, *API* questions asks for help on a framework function, and *Documentation* questions ask clarification on some concept/behavior of the framework. Considering configuration questions, *Versioning* refers to issues related to incompatibility of library versions, *Environment* refers to issues related to incorrect permissions or missing dependencies, *Misc. Files* refers to issues related to misconfigured files, *Missing Files* corresponds to missing files, and *Library* refers to problems with the setup of libraries in the framework.

	Subcategory	Question Id	Question	Answer
general	Presentation	86653	How can I “pretty” format my JSON output in Ruby on Rails?	Use the <code>pretty_generate()</code> function, built into later versions of JSON.
	Database	17006309	How to use “order by” for multiple columns in Laravel 4?	Simply invoke <code>orderBy()</code> as many times as you need it.
	API	2260727	How to access the local Django webserver from outside world?	You have to run the development server such that it listens on the interface to your network E.g. <code>python manage.py runserver 0.0.0.0:8000</code>
	Documentation	20036520	What is the purpose of Flask’s context stacks?	Because the request context is internally maintained as a stack you can push and pop multiple times. This is very handy to implement things like internal redirects.
configuration	Versioning	19962736	I am trying to run statsd/graphite which uses django 1.6, I get Django import error - no module named <code>django.conf.urls.defaults</code>	Type from <code>django.conf.urls</code> import <code>patterns</code> , <code>url</code> , <code>include</code> .
	Environment	11783875	When I run my main Python file on my computer, it works,when I activate venv and run the Flask Python, it says “No Module Named bs4.”	Activate the virtualenv, and then install <code>BeautifulSoup4</code>
	Misc. Files	19189813	Flask is initialising twice when in Debug mode.	You have to disable the “ <code>use_reloader</code> ” flag.
	Missing Files	30819934	When I try to execute migrations with “ <code>php artisan migrate</code> ” I get a “Class not found” error.	You need to have your migrations folder inside the project classmap, or redefine the classmap in your <code>composer.json</code> .
	Library	18371318	I’m trying to install Bootstrap 3.0 on my Rails app. What is the best gem to use in my Gemfile? I have found a few of them.	Actually you don’t need gem for this, install Bootstrap 3 in RoR: download bootstrap from getbootstrap.com .

refers to issues related to incorrect permissions or missing dependencies, *Misc. Files* refers to issues related to misconfigured files, *Missing Files* corresponds to missing files, and *Library* refers to problems with the setup of libraries in the framework. Our results are consistent with previous studies [40].

Table 2 shows example questions for each of those categories. For example, the StackOverflow question 86653 asks how to format a *json* object in Rails using the function `pretty_generate()` from module `json`. As another example, question 17006309 shows how to sort multiple columns in a dataset using the Laravel function `orderBy`. Considering configuration posts, the question 19962736 reports a case where the owner of the question found a “django module error” when trying to import module `django.conf.urls.defaults`. The issue, in this case, is that the user was using Django version 1.6 which no longer uses that name for the module; the new module name is `django.conf.urls`.

Figure 6: Distribution of general and configuration questions. Horizontal line indicates average value (22%) of configuration questions across frameworks.

3.2.1 Popularity

We used metrics previously used in other studies to characterize popularity of Q&A posts [41; 42; 43; 44; 45; 46], namely: the *score* of the question — this number is adjusted by the crowd according to their appreciation to the question, the number of *views* — this number increases every time a user visits the question (whether (s)he likes or not), and the number of *favorites* — this number is adjusted every time a user bookmarks the corresponding question. We ran tests of hypothesis to compare general and configuration questions w.r.t. these metrics. For a given metric, we propose the null hypothesis that the distributions associated with general and configuration questions have the same median values. The alternative hypothesis being that the corresponding medians differ. As usual, we first used a normality test to check adherence of the data to a Normal distribution [47]. According to the Kolmogorov-Smirnov (K-S) normality test, we observed that data did *not* follow Normal distributions. For that reason, to evaluate our hypotheses, we used non-parametric tests, which make no assumption on the kind of the distribution. We used two tests previously applied in similar contexts: Wilcoxon-Matt-Whitney and Kruskal-Wallis [47]. The use of an additional test enables one to cross-check results given the inherent noise associated with non-parametric tests. The null hypotheses was not rejected in any test we ran: *p*-values were much higher than 0.05, the threshold to reject the null hypothesis with 95% probability. To sum, considering the metrics we analyzed, there is no statistically significant difference in popularity between general and configuration posts.

3.2.2 Prevalence

Figure 6 shows the distribution of general and configuration questions for each framework. Considering the six frameworks we analyzed, it is noticeable that general questions are considerably more prevalent compared to configuration questions. It is also noticeable that Meteor manifests the lowest proportion of configuration questions to general questions. That happens because Meteor, in contrast to alternative frameworks, provides pre-configured options and a rich set of libraries built-in.

Figure 7 shows the distribution of configuration questions per framework obtained using

Figure 7: Distribution of configuration questions per framework.

card sorting. Notice that categories “Environment” and “Misc. Files” were more prevalent, considering all six frameworks. We highlight the distribution of configuration questions as they are particularly relevant for this study—reproducing these questions is more challenging compared to general questions (see Chapter 4). For example, these questions often contain multiple configuration files, missing dependencies, etc. Docker can provide an advantage in that respect. Note that, although general questions are prevalent in this scenario, configuration questions are also common and popular.

4

FEASIBILITY STUDY

The study to assess feasibility is organized around two dimensions of analysis—Adoption Resistance and Effort. The dimension “Adoption Resistance” assesses interest of the StackOverflow community in obtaining executable scripts for posts. If there is strong evidence that general interest is low, pursuing the idea brings low value. The dimension “Effort” assesses complexity of the task associated with building containers. If the task is too complex then only few developers would embrace it.

4.1 Adoption Resistance

- **RQ1: What are the perceptions of StackOverflow users towards the use of Docker to reproduce posts?**

The goal of this research question is to assess user’s attitude towards the use of Docker for reproducing Q&A posts. To answer this question, we surveyed StackOverflow users. We selected users from the five frameworks that we successfully created Docker containers (see Chapter 4.2). For any given framework, we pre-selected 1K users with the best reviewing scores. Since StackOverflow does not allow users to publish e-mails on their pages, we attempted to establish links between StackOverflow and GitHub accounts. More specifically, for a given user, we searched for her GitHub username from her StackOverflow user’s account and then looked for a matching e-mail in her GitHub account. Using this approach, we identified a total of 1,548 potential participants from a total of 5K users (1K users per framework). Finally, we sent invitations to participate in a survey. The survey questions are as follows.

1. Are you familiar with Docker?
  - (a) Never heard of it;
  - (b) Have played with it a bit;

- (c) Use it frequently.
2. Do you think executable Dockerfiles could help developers understanding Q&As from StackOverflow?
- (a) Yes;
- (b) No;
- (c) I don't know.
3. What do you think are the main challenges in using Dockerfiles at StackOverflow?
- (a) Security concerns;
- (b) It is time consuming to read and write dockerfiles;
- (c) Lack of sysadmin skills;
- (d) Most Q&As are pretty straight-forward;
- (e) I don't know.

The goal of the survey is to identify developer's perceptions about the idea of using Docker at StackOverflow. For the first question, the intuition is that it would be challenging to incentivize adoption if familiarity with the technology is very low. The second question assesses perceived utility of our proposal. Finally, the third question evaluates technical concerns of users about dockerization at StackOverflow. A total of 106 users answered this survey. Of which, we discarded 13 invalid answers (e.g., auto-reply answers). It is important to note that not every participant answered all questions. For example, someone that answered "a" to the first question would not answer the remaining questions. However, most participants answered most questions. Figure 8 shows the distributions of answers for the first three questions.

Figure 8: Answers for the survey.

Considering question one, we found, with some surprise, that $\sim 90\%$ of participants who answered the survey were familiarized with Docker and a large proportion of them (35.5%) use Docker frequently. Considering question two, 39.2% of the participants were optimistic about using Docker to reproduce Q&A posts. Participants in this group mentioned that Docker would help to reproduce complex environments and version-pinned questions. It is worth mentioning that most of those participants (95% of them) were familiar with Docker (i.e., answered “b” or “c” to question one). However, we also found that 54.7% of the participants do not think that Docker would help. For example, some developers of the Express framework commented that, when the post did not depend on server-side features, Docker would not be necessary. When we asked to indicate main challenges of the approach, developers pointed to effort (option “b”) and need (option “d”), with respectively 32.3% and 33.1% of the answers. To sum, despite the optimism signaled by developers, a large proportion of them answered that reading and writing dockerfiles could be time-consuming and posts could be either straight-forward or not require fully-functioning code for understanding. Furthermore, participants that selected option “c” commented that creating dockerfiles could be challenging to new developers and a total of 12.6% of the participants were worried about security (option “a”), however, none of them specified the reason why. Participants had the opportunity to send their comments with their answers, but they did not go beyond that.

Answering RQ1: To sum, a high number of participants knew Docker and a total of 39.2% of the participants thought Docker would improve user’s experience in StackOverflow. In contrast, 54.7% of the participants considered Docker an overkill in this context. Participants were mainly concerned with cost of writing scripts and need.

The following chapter addresses some of the concerns raised by the participants, including need and cost of writing.

4.2 Effort

- **RQ2: How often can developers dockerize posts?**

The goal of this question is to estimate the amount of posts that could be translated into executable scripts and to understand the reasons that prevent the creation of those scripts. To create containers, we used a Debian 8.6 Jessie machine [48] with `docker` and `docker-compose` [6] installed. Two developers with over three years of professional experience in web development carried out the task of writing dockerfiles to the 600 posts from our dataset. One developer had working experience with JavaScript and another developer, the first

Table 3: Breakdown of problems found while generating dockerfiles. Column “ Σ -P*” indicates the total number of posts reproduced per framework. P1 = Unsupported. P2 = Lack of details. P3 = Conceptual. P4 = Clarification. P5 = User interaction. P6 = OS-specific.

			Unreproducible				Costly		
		Σ	P1	P2	P3	P4	P5	P6	Σ -P*
General	Express	71	-	1	26	1	-	-	43
	Meteor	91	91	-	-	-	-	-	0
	Laravel	72	-	17	13	2	-	-	40
	Django	76	-	5	12	8	-	-	51
	Flask	84	-	2	19	5	-	-	58
	Rails	74	-	-	32	-	2	-	40
	Total	468							232
Configuration	Express	29	-	12	-	-	1	-	16
	Meteor	9	9	-	-	-	-	-	0
	Laravel	28	-	9	-	-	-	6	13
	Django	24	-	8	-	-	7	3	6
	Flask	16	-	4	-	-	-	-	12
	Rails	26	-	11	-	-	1	5	9
	Total	132							56

author of this dissertation, had working experience with Laravel (PHP) and Django (Python). The task of writing a dockerfile for a given post consists of the following steps: (1) understand the post, (2) reproduce the post on the developer’s host machine, (3) create the dockerfile, and (4) spawn the container and check correctness according to the instructions in the post. For general questions, which typically follow the “how-to” pattern (see Chapter 3.2), developers were asked to produce one dockerfile with the solution to the question. For configuration posts, which typically follow the “issue-fix” pattern, developers were asked to produce two dockerfiles: one to reproduce the issue and another to illustrate the fix. Developers used stack traces, when available in the posts, to validate correctness of their scripts. For example, if the post reports an issue, the developer used the trace to validate both the “issue” script and the corresponding “repair” script for the presence (respectively, absence) of the manifestation in the trace. Developers also validated each other’s containers for mistakes. It is important to highlight that, while preparing those reproduction scripts, the two developers noticed that the files they produced were very similar. For that reason, they prepared per-framework template files as to facilitate the remaining work. For dockerfiles, this task was manual. The developers installed each dependency described in the installation guide for each framework and adapted the install commands for the Dockerfiles. For application code, three of the framework—Django, Laravel, and Rails—provide tools to generate boilerplate code.

As expected, some posts (48% from the entire dataset) could *not* be reproduced either because they were unreproducible or because they were too expensive to reproduce. Table 3

shows the breakdown of those problems per framework and category and illustrates how many of the 600 posts could be translated. Column “ Σ ” shows the total number of posts associated with a given framework. Columns “P1-P6” show the number of posts that could *not* be reproduced due to a given problem. Column “ Σ -P*”, appearing at the rightmost position in the table, shows the total number of posts that developers could reproduce with Docker using the setup we described. A dash is a shorthand for zero, i.e., it indicates that no problem has been found. The problems developers found are as follows: *P1 (Unsupported)*: A feature necessary to dockerize the post is still unsupported. For example, as of this date, Docker does not support a particular feature from `tar` necessary to run Meteor [49; 50]. *P2 (Lack of details)*: The question lacks important details to reproduce the problem (e.g., post 26270042). *P3 (Conceptual)*: The question is a conceptual question about the framework usage (e.g., post 20036520). *P4 (Clarification)*: The question is a clarification question about the framework (e.g., post 14105452). *P5 (User interaction)*: Console interaction is necessary to create a container (e.g., post 4316940). *P6 (OS-specific)*: The post is specific to a non-Linux OS (e.g., post 10557507).

It is worth highlighting that the questions associated with problems P5 and P6 could be addressed, in principle, but, given our limited resources, we decided to restrict our study to posts that could be reproduced without console interaction and to posts that are specific to Unix-based distributions. Only a small fraction of posts (4.1%) did not satisfy these two constraints. Considering P6, for instance, it is possible to create Windows containers, but only on Windows hosts running proprietary virtualization software (e.g., Microsoft’s Hyper-V). We also note that quite a few posts (69) could not be reproduced because the writing was unclear (P2). We did expect that textual descriptions could lead to this problem but still we were surprised by the considerable number of cases, 11.5% of the total. Overall, developers translated 49.6% of the general posts and 43.2% of the configuration posts. If we remove from these counts posts that are, in principle, reproducible (P5 and P6) we increase those numbers to 49.8% and 52.7%, respectively. If we discard conceptual posts (P3), the numbers of general posts reproduced becomes 63.4%. If we discard unclear posts (P2), the numbers of configuration posts reproduced becomes 63.6%.

Answering RQ2: We found that many of the posts in our dataset were unreproducible, but a higher incidence of those cases were observed in general posts.

- **RQ3. How hard is it for developers to dockerize posts?**

Determining complexity of posts is important. On the one hand, questions can be so simple that would render reproduction scripts useless. On the other hand, they can be so complex that would discourage developers. Determining complexity levels of Q&A posts requires

Figure 9: Difficulty levels per category (configuration).

human cognizance. The two developers involved in RQ2 also attributed difficulty to posts during the dockerization task. The methodology used to assign difficulty levels is as follows. The developers first analyzed the question and corresponding answers, then reproduced the question in her local environment, and then created a corresponding Docker container. Developers only determined difficulty for cases where they could reproduce in the local machine. (See RQ2 for details.) In some cases, developers could not reproduce a container. These steps were timed but developers used mostly their perception of difficulty—“Easy”, “Medium”, or “Hard”. Informally, “Easy” questions are those that could be solved with basic entry-level framework and language knowledge. , “Hard” questions are those that require knowledge acquired after implementing a complete web application, and “Medium” questions are those that fall in between these cases. After separately assigning difficulty levels to questions, developers discussed conflicting cases. There was disagreement in $\sim 20\%$ of the cases. In none of these cases, however, the disagreement was of the kind “Easy” versus “Hard”. In all of these cases, developers found agreement after discussion.

Considering *general* questions, developers observed that most of them fell in the “Easy” class: answers to those questions can be found in documentation and tutorials of the corresponding framework. This observation is consistent with the results obtained by Treude et al. [40] and also by Beyer and Pinzger [32], who analyzed posts from broad Q&A forums. To note that their study did not focus on web development. Preparing Docker scripts for those cases is certainly not cost-effective. Compared to the posts from the general group, the posts from the configuration group had perceived difficulty significantly higher: 61.5% of the configuration posts were classified as “Medium” (40.1%) or “Hard” (21.4%). Figure 9 shows the distribution of difficulty levels per kind of configuration question. Note that most questions of “Medium” or higher difficulty are of the kind “Environment” and “Misc. Files”.

Considering time, we observed, as expected, that “Medium” and “Hard” questions were the most time consuming. Developers took, on average, ~ 3 minutes to analyze the post and ~ 11 minutes to reproduce the post on the host machine. These times do *not* include the preparation of dockerfiles. Developers realized that it was unnecessary to measure and report time for writing the dockerfile because they are typically implemented quickly (recall from RQ2 that developers

Table 4: Number of cases dockerfiles are identical (Same), Average size of dockerfiles (Size), and average similarity of dockerfiles (Sim.). Table 3 shows the absolute numbers of questions for each pair of framework and category.

		Same	Size (LOC)	Sim.
General	Express	48.8%	6.6	90.95%
	Laravel	100%	12.0	100.00%
	Django	41.1%	11.9	93.63%
	Flask	47.5%	11.4	96.38%
	Rails	55.0%	15.4	92.44%
Configuration	Express	42.9%	6.4	92.39%
	Laravel	84.2%	11.7	95.50%
	Django	57.1%	11.1	92.39%
	Flask	84.0%	13.2	96.78%
	Rails	75.0%	15.3	95.07%

used reference dockerfiles for each framework) and because the practice of repeatedly writing these files could lead to over-optimistic (unreal) time estimates.

Answering RQ3: Results suggest that configuration questions are harder to reproduce than general questions. Furthermore, understanding and reproducing the problem in the host machine was found to be costly whereas writing dockerfiles is typically done very quickly.

- **RQ4: How big and similar are dockerfiles?**

Table 5: Application artifacts (e.g., source and configurations files) modified in boilerplate code while preparing containers.

		# Files	Churn	# Ins.	# Mod.	# Del.
General	Express	1.5	9.4	3.8	5.5	0.1
	Laravel	3.7	25.4	18.6	4.7	2.1
	Django	3.9	20.1	18.3	1.8	0.0
	Flask	1.6	8.7	5.7	2.9	0.1
	Rails	8.0	22.1	21.8	0.2	0.1
Configuration	Express	1.2	9.9	4.0	4.9	1.0
	Laravel	1.8	6.8	5.3	1.3	0.2
	Django	2.4	3.5	2.0	1.5	0.0
	Flask	1.6	4.7	2.5	1.8	0.4
	Rails	1.0	3.2	3.0	0.2	0.0

In the following, we report size and similarity of the artifacts to reproduce a post.

Table 4 shows results grouped by frameworks. Columns “Size” and “Sim.” show, respectively, size and similarity of dockerfiles associated with a given framework. Size refers to

the average size across all dockerfiles whereas similarity refers to the average across all pairs of dockerfiles. We used the Jaccard coefficient [51] for that. We did *not* embed application code within dockerfiles as they vary with each post. Column “Same” shows the percentage of cases where the dockerfile was identical to the reference file (see Chapter 4.2). In those cases, the developer only changed application files (e.g., source and configuration files) to run a container (as in Figure 5). Note that in many cases it was unnecessary to modify the reference dockerfile to reproduce the post. Laravel was an extreme case: all 40 scripts from the general category for this framework were identical to the reference dockerfile; changes were made only in application files. This peculiar case happens because, for some frameworks, including Laravel, the corresponding boilerplate project comes with a built-in package manager [52] that resolves dependencies on-the-fly. For frameworks other than Laravel and Express, note that the number of identical dockerfiles is smaller for general posts than for configuration posts. The typical reasons for these cases are that the dockerfile includes instructions to create a database with data that is necessary to reproduce the post. Considering size, results shows that dockerfiles are typically very short, ranging from a minimum of 6.6LOC in Express to a maximum of 15.4LOC in Rails. In addition, the size of dockerfiles for Express are significantly smaller compared to other frameworks. That happens because the Docker official image of Node.js [53], which Express builds on, comes with a fairly complete set of packages that an application needs to run. This is clearly a distinct feature compared to other frameworks. Finally, results show that dockerfiles are very similar to each other with an average similarity score above 94%. Table 5 reports the number of changes made in application files relative to the boilerplate code we used as a reference to create new containers. These files do *not* include the dockerfile. Column “# Files” shows the average number of files modified or created relative to the reference code whereas column “Churn” shows code churn as the amount of lines added, changed, or deleted while reproducing the post. Columns “# Ins.”, “# Mod.” and “# Del.” show the kind of change. All reproduced posts modified at least one application file. Considering general questions, we noticed that developers modified more files preparing containers for Rails compared to other frameworks. Despite that, we observed that developers did not take longer to write code for these cases.

Answering RQ4: Results indicate that reproduction artifacts are typically small and very similar to each other.

5

USER STUDY

This chapter presents two different user studies—one involving students with limited knowledge about the technology and problem domain and another study involving StackOverflow developers, more familiarized with the technology.

5.1 Students

The goal of this experiment was to evaluate ability of developers to create containers from Q&A posts in a pessimistic scenario. This experiment involved students from a grad-level Software Testing course at the authors' institution. No student in class had previous experience with Docker but most of them have heard recently about it. We dedicated a 2h in-lab class to train students—1h for Docker and 1h for the basics of server-side web development. Given the limited time budget, we restricted the training to Flask (in Python), for its popularity and simplicity. All students had access to a similar desktop computer. Students met again two days after the training class to run the actual experiment. The activity was realized in class with the supervision of the authors of this dissertation. We assigned each student the task of reproducing five Q&A posts: two Easy, two Medium, and one Hard (see Chapter 4.2). We randomly selected those posts limiting the quantity according to each difficulty. As a basis of correctness, we confirmed if the result of the container is similar to the output generated by the answer selected by the original poster of the question. The first 30 minutes of the class was dedicated to instruction. After that, students were asked to prepare the scripts and a short critique—pros and cons—of the approach by e-mail. They had 90 minutes maximum for that.

Figure 10 shows a bar plot indicating the performance of the students enrolled in the class. Two of the eight participants did not submit any answer (S.4 and S.8). Of those who submitted, four participants submitted two correct answers and two submitted one correct answer. All questions answered correctly were in the category “Easy”. The main reasons students gave for not being able to reproduce an issue were (i) lack of knowledge in the language or

Figure 10: Students’ performance in preparing dockerfiles.

the framework and (ii) incomplete excerpts of code in Q&A posts. Students firmly indicated in their reports that the training session on Docker was enough for the assignment but they felt they needed more experience in the target programming language and framework. To sum, we considered the results of this study inconclusive. On the one hand, only easy questions were answered and not all students could answer one question. On the other hand, most students could solve at least one problem, suggesting that they could have been able to solve harder problems if they had more experience with the language or framework.

5.2 Developers

This chapter elaborates on a study we conducted with StackOverflow developers in a more realistic setting where developers would have the assistance of a tool to support many steps in the creation of a container answering a post.

5.2.1 FRISK

To support our experiments, we developed a system, dubbed FRISK, to enable rapid creation and sharing of solutions to server-side problems. This section describes every aspect of FRISK.

5.2.1.1 User Interface

FRISK is available online¹ and, to optimize adoption, it works in modern browsers and does not require user authentication. Similar rationale is used in JSFiddle [22], a system to facilitate front-end development (HTML, CSS or JavaScript). FRISK is a fork of “Play-With-Docker” [54; 55] (PWD), a system recently sponsored by Docker Inc. to train people on Docker. In this section we will describe the user interface of FRISK.

Figure 11 shows the homepage of FRISK. This screen allows the user to select one template, from a list of templates, defined based on the experiments from Chapter 4.2. These

¹<http://docker.lhsm.com.br>

Figure 11: FRISK homepage screenshot.

templates are used to create a fresh pre-configure FRISK session available for two hours—to save our server resources. These sessions are essentially the files needed by a framework and a dockerfile declaring all necessary dependencies. Fine tuning is possible by modifying the dockerfile associated with a session using the code editor discussed later.

Figure 12: FRISK editor screenshot.

Figure 12 shows the UI for customizing these artifacts. The screen is divided into three vertical panes. The left pane shows running virtual machines, and a button to create up to five new ones—we limited to save resources. The central pane is divided into two rows. The top row

Figure 13: FRISK screenshot.

is where the controls are available. At the top, Frisk displays the available ports (and links) to access the container created at the virtual machine. Below that ports, is available the command to access the virtual machine using plain ssh. Finally, we provide several buttons to interact with the selected machine through Docker. At the bottom row, there is a console available to run Linux into the virtual machine. The right pane shows a simple file tree and a editor for the files.

A typical FRISK scenario of use consists of selecting a template, modifying necessary files, clicking the *Build* button to create a Docker image, clicking the *Run* button to spawn the corresponding Docker container (it refers to the image created last in the session), and, finally, clicking the *Share* button to generate a URL for the session. A basic tutorial is available online [56]. The share button provides an important feature to support this experiment. When a user accesses the URL created with the share button, FRISK creates a copy of the corresponding files and creates a virtual machine to isolate that session from other users, who could modify the corresponding containers however they want in their own sessions. Using these URLs, StackOverflow users can recover FRISK sessions and visualize solutions to posted issues.

5.2.1.2 Design

PWD is a tool which allows developers to run Docker commands in an in-browser virtual machine. Compared with PWD, the main differences of FRISK are the ability to share sessions and to bootstrap sessions from templates created inside the tool. Other differences include minor changes in the UI and the Docker toolbar including buttons to run Docker commands with default parameters. We noticed, from our experiments, that changing those parameters is rarely necessary. Consequently, users can interact with the system without much knowledge of Docker commands.

FRISK is composed of two modules: `Front` and `PWD`. The first is responsible for implementing the infrastructure for sharing and restoring sessions. While the second is responsible for the Docker playground.

The `Front` module was built on top of Ruby on Rails for its simplicity. The first function is to serve as a home page for FRISK. This function lists the templates created for the frameworks. These templates are sessions adapted and saved for FRISK. The second function is to save the users sessions. When requested by the user, FRISK accesses each VM in a given session, and for each VM, FRISK saves the contents of the `/root` directory in a zip file to reduce the number of files needed to be managed. Then it is created a directory for the corresponding session to place the zip files. A URL is generated for the session. The last function is to restore these sessions. This is possible by accessing the session linked to the URL and creating a new live VM for every zip file.

The `PWD` module is, in summary, is the Play-With-Docker with modifications to allow users to share sessions. The first modification made at PWD was the reduction of the session limit from a 4-hour session to a 2-hour session to be compatible with our budget. The second modification was UI based in the editor. We modified the file editor to be present in the same page as a panel. The addition of a `Share` button was necessary to enable users to share their sessions. In summary, this button evokes a function in the `Front` module to access each VM created in the session, and save the contents of the `/root` directory in a zip file. We decided to save the contents of the VM in zip files to reduce the amount of files to manage while restoring these sessions. Minor UI changes includes the removal of some components, such as the timeout clock and the IP field in the toolbar. Also, the inclusion of a file editor as a panel and the FRISK logo. These changes were made to disassociate the Play-With-Docker from FRISK.

The Docker toolbar was included in the PWD editor is composed of five buttons. The `Build` button creates the Docker image using the `build -t mycontainer .` command. This command starts the build process of the image and stores the finished image with the name `mycontainer`. The `Run` button starts a container using the `docker run -P mycontainer`

command. Using the `-P` option in the run command, Docker automatically assigns every port specified in the dockerfile with `EXPOSE` to a random port in the host machine. The *Stop* button runs two commands. First, FRISK runs `docker ps -a -q` to get a list of all containers in the virtual machine. Then it stops every container using `docker stop <container_id>`. The *Delete* button runs a similar set of commands. The first is also used to get a list of containers. Then it deletes every container using `docker rm -f <container_id>`. Observe that the `-f` is used to force the deletion of running containers. Finally, the *List* command is used to list every container in the virtual machine. The button runs the `docker ps -a` to present a list of containers in the terminal.

5.2.1.3 Using FRISK

In this section, we will describe a simple walkthrough FRISK. Using Frisk is possible with an internet connection and a modern browser. In this example, we will deploy a minimalistic Express.js app using FRISK. A very similar method can be used to prototype apps for other frameworks.

The first step, at the home screen (see Figure 11), by selecting the Express.js card, the user will be redirected to the editor interface (see Figure 12) and the following effects will take place:

- it creates a FRISK session with one virtual machine in it.
- it adds a dockerfile for Express.js
- it adds a boilerplate code—`index.js`—for a simple web service

At this point, the user should be facing the terminal at the `/root` directory. This is the base directory for making changes in the virtual environment. The file editor is also visible in case the user prefers to edit files using a visual editor. Alternatively, the user could use `vim` [57] on the shell to create and edit files.

Figure 14: File “`index.js`”.

```
1 var express = require("express");
2 var app = express();
3
4 app.get("/", function(req, res){
5   res.send("Hello world!"); // <-- here
6 });
7
8 app.listen(8080);
```

After checking the environment, opening the file `/root/index.js` (shown in Figure 14), a user could modify to print a different message. This file contains Express.js code (a framework of Node.js) to respond to an HTTP request to the base URL of the app (specified at line 4 with the string `'/'`). Modifying the string `"Hello world!"` (at line 5), the user would get a customized message, as in Figure 15. Note that the string is passed to the function `send` from object `res`, which denotes the response to an HTTP request.

Figure 15: File `"index.js"` in FRISK editor.

Figure 16 shows the default dockerfile created by FRISK. Note that some instructions were introduced in Chapter 2.2. The `WORKDIR` instruction sets the working directory that is used by other dockerfile instructions. The `COPY` instruction copies the source files from the host (in this case a FRISK VM) into the image, so the container can access those files to run the application. Observe that in Figure 14, at line 8, the `index.js` file spawns the Express.js server at port 8080. The same port must be specified at the dockerfile with the `EXPOSE` instruction. This instruction informs the Docker to redirect a port (selected at runtime) to the container, allowing the user to make HTTP calls.

Figure 16: File `"Dockerfile"`. It spawns Express.js app `index.js`.

```

1 FROM node:6.9.5
2 RUN mkdir /app && cd /app
3 WORKDIR /app
4 RUN npm install --save express
5 COPY . /app
6 EXPOSE 8080
7 CMD node index.js

```

Building the image automatically is possible by clicking the *Build* button, as seen in Figure 17 as arrow A. Running the container is as simple as building it. Clicking the *Run* button runs the generic command to run a container, as seen in Figure 17 as arrow B. With the container running, FRISK will automatically detect the port it is running on the VM, creating a

link to access the container. The link will be available on top of the page, as seen in Figure 17 as arrow C. Clicking the link, FRISK will open a new window with a connection to the newly created container.

Figure 17: FRISK toolbar. Arrow A indicates the Build button, arrow B indicates the Run button and arrow C indicates the link to the container port.

Sharing sessions is one of the main differences from the Play-with-Docker platform. By clicking the share button that appears on the top of the screen, FRISK will create a backup of the VM and associate that backup to an URL. Clicking on that URL, FRISK will recover that backup and set up a new VM with the modified files.

5.2.2 Design

Our goal with the experiment was to assess willingness of StackOverflow developers in adopting FRISK. We initially considered the idea of asking developers to prepare FRISK sessions, but, we realized people would likely be discouraged. Although we thought the effort for that task would not be high, we thought people would have no incentives for doing that work on a system they did not know. Instead, our plan was to ask people to evaluate FRISK sessions that we created for the StackOverflow posts they created. The rationale is that developers would relate to their own work and they could play from an existing example that they could modify. To sum, we created FRISK sessions for previously-created posts, sent e-mails to developers and added comments to posts as to advertise the FRISK solution, and then monitored user activity.

Dataset. We prepared FRISK sessions for a selection of configuration-related posts. Each session reproduces the preferred answer to the corresponding StackOverflow question. We selected the top 200 questions involving distinct people, i.e., question makers and respondents. In total, we prepared 100 sessions, 20 for each framework.

StackOverflow policy. Our initial attempt

for this experiment was to edit the preferred answer adding a link to the FRISK container showing how to reproduce the solution and then monitor reaction on StackOverflow. Unfortunately, we realized after-the-fact that the StackOverflow policy rejects posts that may seem as a tool advertisement. As consequence, the updates we created were rejected by the StackOverflow community. To address that, we contacted developers through e-mails and comments. In both cases we provided a link to the FRISK container, explain what it offers, and ask people to try it out. For the StackOverflow comments, we did not name the tool as to prevent rejection of the post.

5.2.3 Results

Table 6: Data obtained from FRISK analytics.

Framework	Duration	#Sessions	Builds	Runs	Accesses
Django	13m41s	90	62.22%	51.11%	17.78%
Express	9m49s	90	68.89%	58.89%	55.56%
Flask	9m59s	175	86.86%	74.86%	49.14%
Laravel	11m26s	105	87.62%	74.29%	48.57%
Rails	11m38s	103	86.41%	54.37%	50.49%

Table 6 summarizes results obtained over a month monitoring user activity in FRISK. Note that we could monitor activity because all commands are executed in our servers. Results in Table 6 are broken down by framework. Column “Duration” shows the average time the user spent interacting with FRISK. The period of interaction begins from the point the user accesses the URL—created to share the session—and stops at the moment of the last interaction—we looked for inactivity in the logs. Column “#Sessions” shows the number of sessions accessed for a particular framework. Columns “Builds”, “Runs”, and “Accesses” show, respectively, the percentage of cases (i.e., fraction of number from column “#Sessions”) where users clicked the *build* button, the *run* button, and the link generated to access the running service on the browser. Note that the percentages must not increase as one can only run a container if she has built the image and one can only access the service if she has ran the container.

It is interesting to observe the attention received by Flask, provided that this framework is the least popular among the five we selected [58]. Looking at the column “Accesses”, it is possible to observe that a total of 254 accesses were made, i.e., a high number of developers, in absolute terms, completed the steps to reproduce the problem. We were also surprised that Django, which is another Python framework in this group and one very popular, was the case with the smallest rate of successful accesses by developers. We conjecture that the amount of

training in a given framework influenced the number of successful accesses, which is our proxy for interest in FRISK. Finally, we noticed a relatively high gap between columns “Runs” and “Accesses”, provided that to access the service—and count one access—FRISK users only needed to click on a link after spawning a container. One possible reason for that is that users are missing the URL link to make an HTTP request to the running service. This link is dynamically created after the container starts to run.

We observed from these results that developers played with the system for a good amount of time ($\sim 10m$), the system received a substantial number of accesses over the course of a month and considering the number of posts we advertised (563), and many of these accesses to FRISK resulted in the user accessing the corresponding web service ($\sim 45\%$). Overall, we believe this data provides some early evidence on the interest of the community in FRISK as a learning tool that could be used to link Docker and Q&A forums, such as StackOverflow.

6

DISCUSSION

We presented a feasibility study to assess the potential of Docker to assist web developers in using Q&A forums.

Our results suggest that the fears developers manifested during our survey (Chapter 4.1) were not all justified. Developers mentioned concerns of cost in writing dockerfiles, but that task has shown to be short. The artifacts involved in a post are similar to each other (Chapter 4.2 RQ4) and that enabled the construction of templates—including reference dockerfiles and boilerplate code—that enabled developers to be more productive in this task. Developers also manifested concerns about need of using Docker in that context. In fact, we found that was the case for the posts in the general category. However, there is an important group of post for which solutions are non-trivial and integrating Docker could be helpful (Chapter 4.2 RQ3). The study of Horton and Parnin [5] corroborates that. Many code snippets they analyzed from GitHub required non-trivial configuration-related changes to be executed, including missing dependencies, misconfigured files, reliance on a specific operating system, or some other environment issue. Finally, developers also manifested concerns with security, but FRISK containers run on the cloud so compromising user space is not possible.

While preparing our experiments, we found scenarios where Docker could not properly build the container. Those issues can hinder developers from using containers. For example, while creating Meteor containers, Docker would throw an error and prevent the developer to continue building the container. These issues are related to how Docker handles the storage driver, which is not compatible with some Meteor dependencies, which could be extended to other use cases, apart from Meteor. Changing the storage driver used by Docker, or allowing developers to specify which driver is going to use on the Dockerfile, would prevent this problem from occurring.

We believe our results encourages the use of Docker, in certain cases, to assist developers in Q&A forums. It is natural to expect that much better support is needed to realize that vision in practice as FRISK is still a proof-of-concept tool. We also believe that improved

versions of FRISK could be used for other purposes, including training students in new technologies and outsourcing debugging activities. In the near future, we plan to add a simple debugger to the FRISK IDE and use it in Software Engineering undergrad-level courses at the authors' institution.

6.1 Threats to Validity

In this section, we discuss the limitations of our study and our approach to handle them. In the following, we describe the external, internal and construct threats to the validity of our results.

6.1.1 External Validity

The extent results can be generalized is limited by our dataset, which includes Q&A posts from a selection of web frameworks. In principle, there could be frameworks and posts with different characteristics that could lead to different findings. To mitigate those issues, we selected the six most popular frameworks, according to a recent showcase from GitHub and selected questions according to an objective criteria, described in Chapter 3.1. It remains to evaluate the extent to which our observations would change when using different frameworks (e.g., frameworks not in the listing from Figure 1) and a different criteria for selecting questions for each framework. Another threat is related to the generalization of the templates prepared in this study. In principle, there could be scripts unfit to those templates, i.e., scripts that would require significant changes. Another threat is related to the number of cases we used to build the template. Developers considered a relatively small number of cases to prepare those scripts and validated them against a large number of scripts.

6.1.2 Internal Validity

Our results could be influenced by unintentional mistakes made by humans involved in this study. For example, students were involved in a user study whereas developers manually categorized questions in difficulty levels and elaborated dockerfiles. All those tasks could introduce bias. We used Card Sorting [39] to mitigate the problem of incorrectly categorizing questions. To make sure the scripts were correct, developers were instructed to strictly follow the instruction from Q&A post preferred answers to reproduce corresponding problems. We also encouraged developers to do their best to reproduce as many questions as possible. As for the answer of students in the user study, we analyzed their answers carefully, comparing them with the solution prepared by the instructors. It is important to note that all artifacts produced during this study are publicly available for scrutiny. Finally, the monitoring infrastructure that we used for tracking FRISK usages did not take into account the possibility of a user accessing the same session multiple times. However, we analyzed manually the logs and did not notice a high number of accesses for individual FRISK containers, suggesting that that was not an issue.

6.1.3 Construct Validity

We considered a number of metrics in this study that could influence some of our interpretations. For example, we used metrics of document similarity to assess how (dis)similar the dockerfiles produced by developers are. To mitigate the bias associated with metric selection we used multiple metrics and confirmed that the similarity was very high as to not compromise corresponding conclusions.

7

RELATED WORK

We organized related work in two groups—work related to educational tools and collaborative IDEs and work related to mining repositories.

7.1 Educational tools and Collaborative IDEs

Tools such as Repl.it [21] and JSFiddle [22] provide support to create and share self-contained code examples. Platforms such as Jupyter Notebooks [59] provide support to create interactive guides and tutorials, including self-contained code with gaps for students to fill and create a running code. These platforms are great for teaching, but they are not well suited for the creation of complex environments, including databases, web servers, etc. The configuration posts that we analyzed in this dissertation involve at least one or more of these aspects. Collaborative IDEs, such as Cloud9 [60] and CodeAnywhere [61], can, in principle, build more complete local environments but these are private, making sharing more difficult. It is important to note that exploring live collaboration seems an important feature to have in this context that should be explored in FRISK.

7.2 Mining repositories

We elaborate below work that reports on issues in repository data and work that proposes ways to fix those issues.

Recent work studied various aspects of development behavior manifested through StackOverflow data. For example, Yang et al. [2] criticized StackOverflow code quality, indicating that code is written mostly for illustrative purposes and “compilability” is not typically considered. Terragni et al. [3] and Balog et al. [4] also found that compilation issues are common. Bajaj et al. [42] analyzed StackOverflow questions to understand common difficulties and misconceptions among JavaScript developers. They focused on a restricted domain; in their

case JavaScript, in our case server-side frameworks. In a different study, Treude et al. [40] found that often answers to questions become a substitute for official documentation. Considering the general category of questions, the results we found are consistent with theirs. Allamanis and Sutton [62] automatically analyzed arbitrary StackOverflow questions using standard data mining techniques. In contrast to them, we explored a narrower domain and involved humans in the analysis of questions. Beyer and Pinzger [32] presented an automatic approach to classify documented Android issues in StackOverflow using the Apache Lucene search engine [63]. They used manual classification of questions using Card Sorting as we did but for a different reason—to build the ground truth to base the computation of accuracy of automatic classification techniques. The idea is complementary to ours. Searching for good post candidates for creating containers is could help engage developers in using FRISK. Yang et al. [64] automatically analyzed code snippets from StackOverflow to measure how often these snippets originate from open source projects. They found that in many cases the link could be recovered. One interesting avenue of future work is to slice minimal FRISK containers from those projects.

Recent work proposed solutions to existing problems in StackOverflow or GitHub. For example, Terragni et al. [3] proposed CSNIPPEX, a technique to automatically transform StackOverflow code snippets into compilable Java code. Their technique looks for fixes to compilation errors, such as missing import declarations. More recently, Horton and Parnin [5] proposed Gistable, a tool to automatically transform Python code snippets from GitHub into runnable Dockerfiles, and DockerizeMe [65], a tool that runs combined with Gistable for inferring the missing dependencies needed to execute a Python snippet. As CSNIPPEX, their tools also makes simple transformations, if necessary, to repair the Gist code. Differently from CSNIPPEX, Gistable tries to write Dockerfiles from a given StackOverflow post, creating a large database of Dockerfiles based on real-world questions. In contrast to Gistable, FRISK provides an infrastructure for sharing solutions and focuses on problems (or solutions to those problems) that may require multiple files and services (e.g., database, templates) to demonstrate those problems whereas Gistable focuses on compiling self-contained snippets.

Finally, Balog et al. [4] proposed DeepCoder, a technique that uses Deep Learning to synthesize code from StackOverflow code snippets. In principle, DeepCoder could capitalize on better code snippets to improve code synthesis. These works provide evidence on the importance of writing quality code at Q&A forums. Note, however, that high-quality code alone is insufficient to demonstrate certain kinds of issues. This is noticeable on the configuration questions mentioned in this dissertation. Executable scripts can help on that.

8

CONCLUSIONS

This dissertation reports on study to assess the feasibility of using Docker to reproduce Q&A posts related to development with web frameworks. This is a timely and important problem given the constant pressure for increased productivity in this domain [66] and the observation that web developers heavily rely on Q&A [7] forums nowadays.

Feasibility study. Considering the dimension Adoption Resistance, we found that most participants of a survey we ran are familiar with Docker: 35.5% of the participants use it frequently and other 54.8% have played with it. We also found that 39.2% of the participants think that Docker could improve productivity of StackOverflow users whereas 54.7% of the participants consider it an overkill. Considering the dimension Effort, our results show that many of the posts analyzed require little context and could be answered with short snippets. These posts are rarely configuration-related and, for these, reproduction scripts are certainly of little help. We observed that reproduction scripts helps the most in configuration posts of medium and high difficulty. Considering that, 22% of the 600 posts are configuration related. Of these, 61.5% are of medium or high difficulty. We also found, preparing containers ourselves, that the reproduction of the problem in the host environment is the most time-consuming activity in addressing a post, taking ~ 11 m per post. That step occurs regardless of the adoption of Docker. Preparing the dockerfile after that step can be done quickly—these scripts are typically very short and similar to each other (see Tables 4 and 5). To sum, we felt encouraged to look deeper into the problem as results suggested that there is a sweet spot in the kind of posts that would benefit from the proposed solution.

User study. Over the course of a month, we monitored user activity of a total of 563 FRISK sessions, associated with solutions we created in FRISK to a total of a 100 StackOverflow questions. A session is created when a user accesses a link—that we provided through emails or post comments—to the FRISK solution we prepared. To sum, we found that, on average, users spent almost ten minutes playing with the system and that 255 of the 563 (=45.3%) sessions resulted in a successful access to the web service associated with the post, i.e., users were

able to build the image, run the container, and access the service from an HTTP request in the browser. Our perception was that FRISK brought attention and interest of StackOverflow users. In summary, our results provide early evidence that the integration of reproduction scripts (e.g., Docker scripts) in Q&A forums (e.g., StackOverflow) should be encouraged in certain cases. As a future work, we plan to evolve the infrastructure and apply it in other scenarios such as:

- In classrooms and workshops, allowing users to learn with live code replication;
- Competitive environments, with time limits and code evaluations;
- Professional environments, with code debugging and fast prototyping.

REFERENCES

- [1] GitHub. (2017) Web application frameworks server-side showcase.
<https://github.com/showcases/web-application-frameworks>.
- [2] D. Yang, A. Hussain, and C. V. Lopes, “From query to usable code: an analysis of stack overflow code snippets,” in *MSR*. ACM, 2016.
- [3] V. Terragni, Y. Liu, and S.-C. Cheung, “Csnippex: Automated synthesis of compilable code snippets from q&a sites,” in *ISSTA*, 2016, pp. 118–129.
- [4] M. Balog, A. L. Gaunt, M. Brockschmidt, S. Nowozin, and D. Tarlow, “Deepcoder: Learning to write programs,” *CoRR*, vol. abs/1611.01989, 2016.
- [5] E. Horton and C. Parnin, “Gistable: Evaluating the executability of python code snippets on github,” in *ICSME*, 2018.
- [6] Docker. (2017) Docker website. <https://www.docker.com/>.
- [7] (2017) Stack-overflow. <https://stackoverflow.com/insights/survey/2017>.
- [8] L. Melo and M. d’Amorim. (2019) Paper artifacts. <https://docker-so-study.github.io/>.
- [9] J. Candido, L. Melo, and M. d’Amorim, “Test suite parallelization in open-source projects: A study on its usage and impact,” in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Oct 2017, pp. 838–848.
- [10] D. Mauro Junior, L. Melo, H. Lu, M. d’Amorim, and A. Prakash, “Beware of the app! on the vulnerability surface of smart devices through their companion apps,” in *CORR*, 2019.
- [11] (2019) Safethings 2019. <https://www.ieee-security.org/TC/SPW2019/SafeThings/>.
- [12] (2019) Good news! only half of internet of crap apps fumble encryption | the register. https://www.theregister.co.uk/2019/02/04/iot_apps_encryption/.
- [13] (2019) Insecure apps put half of iot devices at risk | techradar. <https://www.techradar.com/news/insecure-apps-put-half-of-iot-devices-at-risk>.
- [14] (2019) Almost 31% of applications for iot devices do not use encryption | hacker news. <https://hackernews.blog/almost-31-of-applications-for-iot-devices-do-not-use-encryption/>.
- [15] (2019) Half of iot devices let down by vulnerable apps | naked security. <https://nakedsecurity.sophos.com/2019/02/05/half-of-iot-devices-let-down-by-vulnerable-apps/>.
- [16] (2019) Iot expõe residências a invasores | cibersecurity. <https://www.cibersecurity.net.br/iot-expoe-residencias-a-invasores/>.
- [17] (2019) Why Docker? <https://www.docker.com/why-docker>.
- [18] (2017) Docker engine documentation. <https://docs.docker.com/engine/userguide/storagedriver/imagesandcontainers/>.

- [19] (2019) figlet - Linux man page. <https://linux.die.net/man/6/figlet>.
- [20] (2017) Flask api doc. <http://flask.pocoo.org/docs/0.12/api/>.
- [21] (2017) repl.it. <https://repl.it>.
- [22] (2019) JSFiddle. <https://jsfiddle.net>.
- [23] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, “Usage, costs, and benefits of continuous integration in open-source projects,” in *ASE*, 2016, pp. 426–437.
- [24] H. Borges, A. Hora, and M. T. Valente, “Understanding the factors that impact the popularity of github repositories,” in *ICSME*, 2016, pp. 334–344.
- [25] J. Zhu, M. Zhou, and A. Mockus, “Patterns of folder use and project popularity: A case study of github repositories,” in *ESEM*, 2014, pp. 30:1–30:4.
- [26] (2017) HotFrameworks. <http://hotframeworks.com/>.
- [27] (2017) Hurricane Software.
<http://www.hurricanesoftwares.com/most-popular-web-application-frameworks/>.
- [28] (2017) Coding Dojo. <http://www.codingdojo.com/blog/best-programming-languages-full-stack-web-developer/>.
- [29] S. Exchange. (2017) Stack Exchange Data Explorer website.
<http://data.stackexchange.com/>.
- [30] ——. (2017) Stack Exchange website. <http://stackexchange.com/>.
- [31] Anonymous. (2017) Dataexplorer q&a selection query.
<https://data.stackexchange.com/stackoverflow/query/621859>.
- [32] S. Beyer and M. Pinzger, “A manual categorization of android app development issues on stack overflow,” in *ICSME*, 2014, pp. 531–535.
- [33] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, “Mining duplicate questions of stack overflow,” in *MSR*, 2016, pp. 402–412.
- [34] Y. Yao, H. Tong, T. Xie, L. Akoglu, F. Xu, and J. Lu, “Want a good answer? ask a good question first!” *CoRR*, vol. abs/1311.6876, 2013. [Online]. Available: <http://arxiv.org/abs/1311.6876>
- [35] Y. Yuan, T. Hanghang, X. Feng, and L. Jian, “Predicting long-term impact of cqa posts: a comprehensive viewpoint,” in *SIGKDD*, 2014.
- [36] Z. Yanzhen, Y. Ting, L. Yangyang, M. John, and Z. Lu, “Learning to rank for question-oriented software text retrieval,” in *ASE*, 2015, pp. 1–11.
- [37] Y. Yuan, T. Hanghang, X. Tao, A. Leman, X. Feng, and L. Jian, “Joint voting prediction for questions and answers in cqa,” in *ASONAM*, 2014, pp. 340–343.
- [38] Y. Ting, X. Bing, Z. Yanzhen, and C. Xiuzhao, “Interrogative-guided re-ranking for question-oriented software text retrieval,” in *ASE*, 2014, pp. 115–120.

- [39] M. Lorr, *Cluster analysis for social scientists*. Jossey Bass, 1983.
- [40] C. Treude, O. Barzilay, and M. A. Storey, “How do programmers ask and answer questions on the web?” in *International Conference on Software Engineering (ICSE NIER)*, 2011, pp. 804–807.
- [41] J. Sillito, F. Maurer, S. M. Nasehi, and C. Burns, “What makes a good code example?: A study of programming q&a in stackoverflow,” in *ICSM*, 2012, pp. 25–34.
- [42] K. Bajaj, K. Pattabiraman, and A. Mesbah, “Mining questions asked by web developers,” in *MSR*, 2014, pp. 112–121.
- [43] P. S. Kochhar, “Mining testing questions on stack overflow,” in *5th International Workshop on Software Mining, 2016*, 2016, pp. 32–38.
- [44] A. S. Badashian, A. Esteki, A. Gholipour, H. Abram, and E. Stroulia, “Involvement, contribution and influence in github and stack overflow,” in *CASCON*, 2014, pp. 19–33.
- [45] B. Gregoire, Y. He, and H. Alani, “A question of complexity: measuring the maturity of online enquiry communities,” in *24th ACM Conference on Hypertext and Social Media*, 2013, pp. 1–10.
- [46] I. Srba and B. Maria, “A comprehensive survey and classification of approaches for community question answering,” in *ACM Trans. on the Web (TWEB)*, vol. 10, no. 3, Aug. 2016, pp. 18:1–18:63.
- [47] E. Lehmann and J. Romano, *Testing Statistical Hypotheses*, ser. Springer Texts in Statistics. Springer New York, 2008.
- [48] (2017) Debian. <http://www.debian.org/>.
- [49] G. user. (2017) Tar problem when installing meteor. <https://github.com/meteor/meteor/issues/5762>.
- [50] ——. (2017) Automated build fails on 'tar' with: "directory renamed before its status could be extracted". <https://github.com/docker/hub-feedback/issues/727>.
- [51] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Boston, MA, USA: Addison-Wesley Longman Publ. Co., Inc., 2005.
- [52] Laravel. (2017) Laravel. <https://laravel.com/docs/installation>.
- [53] (2017) Node.js official docker image website. https://hub.docker.com/_/node/.
- [54] M. Liljedhal and J. Leibiusky. (2019) Play with docker. <https://github.com/play-with-docker/play-with-docker>.
- [55] (2019) Play with docker labs. <https://labs.play-with-docker.com/>.
- [56] (2019) <http://docker.lhsm.com.br/tutorial>.
- [57] (2019) vim page. <https://www.vim.org/>.
- [58] (2019) Web framework rankings. <https://hotframeworks.com/>.

-
- [59] (2019) Jupyter notebooks. <https://jupyter.org/>.
  - [60] (2017) Cloud9. <https://c9.io>.
  - [61] (2017) Codeanywhere. <https://codeanywhere.com/>.
  - [62] M. Allamanis and C. Sutton, “Why, when, and what: Analyzing stack overflow questions by topic, type, and code,” in *MSR*, 2013, pp. 53–56.
  - [63] Apache. (2017) Lucene. <https://lucene.apache.org/core/>.
  - [64] D. Yang, P. Martins, V. Saini, and C. Lopes, “Stack overflow in github: Any snippets there?” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, May 2017, pp. 280–290.
  - [65] E. Horton and C. Parnin, “Dockerizeme: Automatic inference of environment dependencies for python code snippets,” in *42nd International Conference on Software Engineering*, ser. ICSE ’19, 2019.
  - [66] StackOverflow. (2017) Stackoverflow hiring trends 2017. <https://stackoverflow.blog/2017/03/09/developer-hiring-trends-2017/>.