Studying the Performance Risks of Upgrading Docker Hub Images: A Case Study of WordPress

Mikael Sabuhi, Petr Musilek, Cor-Paul Bezemer
Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Canada
{sabuhi,pmusilek,bezemer}@ualberta.ca

ABSTRACT

The Docker Hub repository contains Docker images of applications, which allow users to do in-place upgrades to benefit from the latest released features and security patches. However, prior work showed that upgrading a Docker image not only changes the main application, but can also change many dependencies. In this paper, we present a methodology to study the performance impact of upgrading the Docker Hub image of an application, thereby focusing on changes to dependencies. We demonstrate our methodology through a case study of 90 official images of the WordPress application. Our study shows that Docker image users should be cautious and conduct a performance test before upgrading to a newer Docker image in most cases. Our methodology can assist them to better understand the performance risks of such upgrades, and helps them to decide how thorough such a performance test should be.

KEYWORDS

Empirical Performance Analysis, Containerized Applications, Dependency Updates, Docker Hub.

ACM Reference Format:
Mikael Sabuhi, Petr Musilek, Cor-Paul Bezemer. 2021. Studying the Performance Risks of Upgrading Docker Hub Images: A Case Study of WordPress. In *Beijing '22: 13th ACM/SPEC International Conference on Performance Engineering, April 09–13, 2022, Beijing, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/xx.xxxx/xxxxxx.xxxxxx

1 INTRODUCTION

The last decade has seen enormous advances in cloud computing, such as faster development cycles, better security and a lower cost of utilizing cloud-based resources. One core-enabling technology for cloud computing is the containerization of applications. Most modern cloud-based applications are dependent on many services and applications. This dependency gives rise to several issues, e.g., a conflict between the dependencies, missing dependencies, and platform differences [17]. A containerization technology, such as Docker, addresses these problems by packaging software code along with its dependencies to run on any computing environment or infrastructure [10]. A containerized application runs in fully isolated environments called *containers*.

The growth in the use of containerized applications has captivated researchers' attention to several aspects of this technology. Most research in this field studies Docker containers and repositories from the security [3, 5, 24, 31] and storage management [25, 32] perspective. For instance, Shu et al. [24] show that both official and community images on Docker Hub have security issues, and that the vulnerabilities propagate from parent images to the child images. Such security issues can often be addressed by upgrading the Docker image. However, upgrading a Docker image may change many dependencies at once. Gholami et al. [8] showed that a median of 8.6 dependencies change in one image upgrade. As prior work showed that changes to dependencies can cause quality issues for the software that depends on them [11, 18, 20], there is always the challenge of deciding how a Docker image upgrade will affect the quality of the containerized application.

One quality aspect that is increasingly important for modern software is performance – Harman and Hearn even state that it should now be considered "the new correctness" [9]. However, performance testing is not a widely implemented practice in industry [1], or by developers [12] in general. Our hypothesis is that the same applies to Docker images. While in cases, the individual performance of the software components inside the image may have been tested, their performance most likely has never been tested when they operate in tandem.

In this paper, we propose a methodology for better understanding the performance risks of upgrading a Docker image, thereby focusing on its dependencies. We demonstrate our methodology through a case study of a performance-critical application, WordPress. First, we conduct load tests on 90 images from the official WordPress Docker Hub repository. These 90 images span a total of 27 WordPress versions with different combinations of dependencies (i.e., ranging from PHP 5.6.x to PHP 7.x.x and from Apache 2.4.10 to 2.4.38). Second, we demonstrate how our methodology can be used to investigate the changes in the performance of the WordPress application between these images. We demonstrate our methodology through the following research questions (RQs):

- **RQ1:** What is the impact of upgrading the Docker image of WordPress on its performance?
- **RQ2:** What is the relation between the performance of the WordPress application and updates of its dependencies?

Our case study shows how our methodology can be employed to study the performance risks of a Docker image upgrade. For example, our case study shows that WordPress users who are planning to upgrade their WordPress image with PHP version 5.x.x to 7.0, can do so safely from a performance point of view.

Beijing '22, April 09–13, 2022, Beijing, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/xx.xxxx/xxxxxx.xxxxxx

2 BACKGROUND AND RELATED WORK

Docker. As an open-source container system, Docker provides a lightweight, low overhead, fast and efficient solution to implement containerized applications [17]. Before Docker, installing, and deploying software on different environments was a painstaking task. Leveraging Docker, one can pull the application images from a repository such as Docker Hub, or one can build their own image and deploy it. Docker images consist of several layers. The first layer is the base image, which is the foundation of the application. Additional layers can be added on top of the base image. The instructions to create the image are specified in the Dockerfile, in which each instruction corresponds to a layer in the image. This Dockerfile can be used by others to recreate that Docker image from a base image. A Docker container is a runnable instance of a Docker image [2], which consists of all the required dependencies required for an application to perform correctly.

With the increased use of Docker containers in the cloud, performance evaluation of these containers is getting more attention. While there are several studies on the performance of Docker [4, 7, 13, 22, 30], we are the first to study the performance impact of upgrading Docker images.

Docker Hub. Docker Hub [6] is a repository for sharing Docker images. In May 2021, Docker Hub hosted more than seven million Docker images. These images are distributed using private or public repositories, which provide a convenient way for software versioning. Public repositories are divided into groups, namely, official and community. Official repositories provide authentic releases of the Docker image of an application since they are reviewed and published by a team that is sponsored by Docker Inc. A Docker image can be retrieved from a repository by its tag. E.g., `wordpress:5.2.2-php7.1-apache` is a Docker image for WordPress version 5.2.2 with PHP version 7.1 and the latest supported version of Apache. Many applications on Docker Hub provide convenience tags for their images, such as `wordpress:latest`, which always points to the latest WordPress image.

Our Case Study Subject: WordPress. WordPress is an open-source content management system (CMS) that is used by more than 35% of all sites across the web [27]. As of May 2020, WordPress' official images were pulled more than 500 million times from Docker Hub [6], which underlines the popularity of this CMS.

The results of a performance evaluation of dynamic and static webpages by Tomisa et al. [26] indicate that WordPress-based websites are performance-sensitive. Hence, an increasing number of users of a website can lead to an unsatisfactory user experience as the response time increases. The most critical factor for users of a website from a performance perspective is the response time. However, none of the WordPress images on Docker Hub mention anything about their performance.

WordPress Docker images come in three variants on Docker Hub: official images, images based on the Alpine Linux project, and command-line interface (CLI) images. In this study, we focus on the official images of WordPress, e.g., `wordpress:5.2.2-php7.1-apache`.

While WordPress indirectly depends on many applications and libraries, the Docker image varies only the Apache and PHP versions, which are considered the main dependencies of WordPress [29].

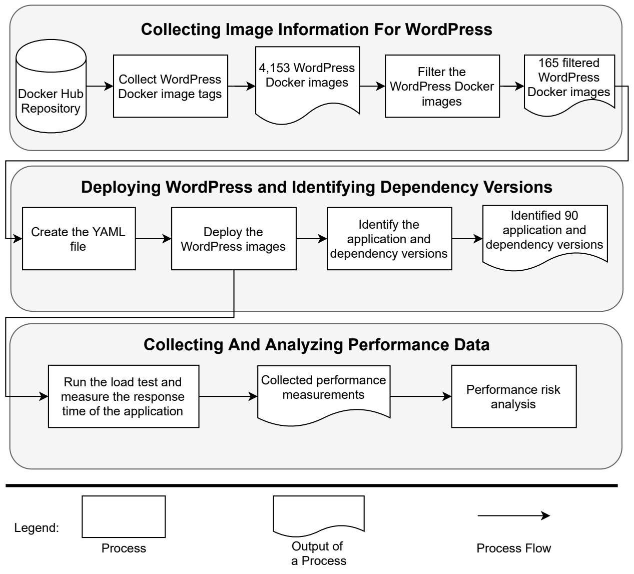
Collecting Image Information For WordPress
Docker Hub Repository
Collect WordPress Docker image tags
4,153 WordPress Docker images
Filter the WordPress Docker images
165 filtered WordPress Docker images
Deploying WordPress and Identifying Dependency Versions
Create the YAML file
Deploy the WordPress images
Identify the application and dependency versions
Identified 90 application and dependency versions
Collecting And Analyzing Performance Data
Run the load test and measure the response time of the application
Collected performance measurements
Performance risk analysis
Legend:
Process
Output of a Process
Process Flow

Figure 1: Overview of the empirical study design.

Hence, in our study, we focus on these two dependencies. To run WordPress, either MySQL or MariaDB is necessary as well. Therefore, we need two containers for running WordPress. One container contains the Apache webserver and PHP with WordPress, and the other container runs the MySQL database.

3 METHODOLOGY

In this section, we present our methodology for analyzing the performance risks of upgrading Docker images. The steps of our methodology are depicted by Fig. 1. We detail each step below.

3.1 Collecting Image Information for WordPress

Collect WordPress Docker Image Tags: As the first step, a web crawler was developed to automatically collect the WordPress Docker image tags from the official WordPress Docker Hub repository[1]. Since Docker images with a WordPress version before 4.7.2 did not have information regarding the operating system and architecture or were not working after deploying, we did not include them in our data set.The crawler collected 4,153 WordPress Docker image tags, including all different WordPress Docker images with different operating systems and architectures.

Filter the WordPress Docker Images: Since not all collected WordPress Docker images are deployable on our available infrastructure, we only included official Docker images that target the `amd64` architecture and depend on the Apache web server. After applying these filtering criteria, we selected 165 WordPress Docker images to conduct the performance test and identify the dependency versions.

[1]https://hub.docker.com/_/wordpress

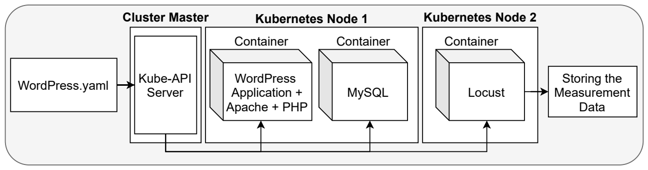

Figure 2: The deployment setup of the WordPress images.

3.2 Deploying WordPress and Identifying Dependency Versions

In this section, we elaborate on the developed two-tier containerized WordPress application and procedure to identify the application and dependency versions at runtime.

We employed three Virtual Machines (VMs), namely Cluster Master and Kubernetes Node 1 and 2. The VMs all had 2 vCPUs, 3.75GB RAM, a 40 GB HDD and were running Ubuntu 1.16.3. All these VMs are deployed on the Cybera cloud[2] with the same resources and configuration. Fig. 2 presents the overview of the Kubernetes cluster. The cluster master manages the deployment of application and services in a cloud environment. The first Kubernetes node contains the WordPress application with the Apache webserver in one container, and MySQL in another container. The second Kubernetes node, contains the Locust [14] application for load testing and monitoring. The rationale behind using separate VMs for Locust and the main application is that generating users adds overhead on the running VM, which may affect the performance of the application under study. The MySQL (version 5.6) and Locust versions are kept constant throughout the experiment. We used MySQL version 5.6 because it is still one of the most used versions of MySQL[3]. The WordPress images are changed through a configuration setting in the YAML file, which is generated automatically. We allocated a 200 milli-core vCPU and 256MB of RAM for each WordPress container.

Deploy The WordPress Images: The created YAML file is sent to the cluster master to control the Kubernetes orchestration platform to first pull the Docker image from Docker Hub and then deploy and run the WordPress application.

Identify the Application and Dependency Versions: To identify the version used by the WordPress images, we used the "Wappalyzer" library [28]. From all 165 images, 6 images could not be analyzed by Wappalyzer and were removed from the data set. After identifying the application and dependency versions of the WordPress Docker images, we realized that some of the Docker images used convenience tags for names and were using the identical dependency versions. E.g., we identified 8 WordPress Docker images with WordPress version 5.3, Apache version 2.4.38 and PHP version 7.3.12, all with different image names. Therefore, we removed 69 duplicate images from our analysis. The complete list of included unique WordPress Docker images with their corresponding dependency versions is available online [23].

[2]https://www.cybera.ca/rapid-access-cloud/
[3]https://www.eversql.com/mysql-8-adoption-usage-rate/

3.3 Collecting and Analyzing Performance Data

Run the Load Test And Measure the Response Time of the Application: To acquire the performance data, an extended version[4] of Locust [14] is used for load testing purposes. This extended version can collect the data with a fixed sample rate and is able to calculate the average response time of the application for these fixed time windows. Furthermore, to send requests to the application and measure the performance metrics, we used the REST API of the load testing tool. Our tool is publicly available online [23].

To measure the performance of each image, we introduced 20 users to the WordPress application through the Locust user generator, and then measured the response time of the application every two seconds and for 5 minutes. We repeated this test 3 times for every image to reduce the impact of variability in the cloud on our measurements (see Section 6 for more details). After studying the measurements, we observed that the measurements across the 3 executions were relatively stable. Therefore, we used only the measurements from the first execution in our analysis. We refer to these values as the average response times ($\overline{RT}$) hereafter. The type of the workload is a simple `HTTP GET` request to the main page of the WordPress website. The motivation behind selecting such a simple workload is to show that the performance can vary considerably even with the simplest workload. The WordPress website is using "Twenty Seventeen" as the default theme.

Performance Risk Analysis: Generally, the version number of software follows the "MAJOR:MINOR:PATCH" semantic versioning principle [19] (e.g. for version "7.2.15" 7, 2, and 15 are the major, minor, and patch version of the application, respectively). According to the semantic versioning definition, major updates will make incompatible API changes, minor updates will add functionality in a backward-compatible manner, and patch updates make backward-compatible bug fixes. Semantic versioning does not officially impose restrictions on how the performance of an application can be affected. However, our expectation is that because of the magnitude of the changes in each type of version, we can use changes in version numbers as a (rough) proxy for changes in performance. Based on the semantic versioning principle, we group the images as follows:

- *Patch groups*: Images with the same major, minor and patch version of an application. E.g., we group all the Docker images of WordPress with major version 4, minor version 8, and patch version 1 into the same patch group (WordPress 4.8.1). This group contains 3 images with PHP versions 5.6.31, 7.0.23, and 7.1.9 and all with Apache version 2.4.10.
- *Minor groups*: Images with the same major and minor version of an application. E.g., the WordPress 4.8.x minor group contains all 12 images with major version 4 and minor version 8 of WordPress.
- *Major groups*: Images with the same major version of an application.

We create patch, minor and major groups for WordPress itself, and the two main dependencies of WordPress on which we focus in this study (PHP and Apache). Note that we only have one minor (2.4.x) and major (2.x.x) group for Apache.

[4]https://github.com/pacslab/pacs_locust

After conducting the load test on each WordPress Docker image, we calculate the average response time for that WordPress Docker image as explained above. To investigate the performance risks of upgrading to that Docker image, we calculate the relative response time by comparing the average response time of that Docker image ($\overline{\mathrm{RT}}_{\mathrm{ver}_n}$) to that of all images in the previous ($\overline{\mathrm{RT}}_{\mathrm{ver}_{n-1}}$) patch, minor, or major group, as shown in Equation 1.

$$\overline{\mathrm{RT}}_{\mathrm{RelativeImprovement}}(\%) : \frac{\overline{\mathrm{RT}}_{\mathrm{ver}_{n-1}} - \overline{\mathrm{RT}}_{\mathrm{ver}_n}}{\overline{\mathrm{RT}}_{\mathrm{ver}_{n-1}}} \qquad (1)$$

If the relative improvement is negative, it means that the performance has degraded; and if it is positive, there was an improvement in the performance. E.g., upgrading a WordPress Docker image from 4.9.1 to another image with WordPress 4.9.2 resulted in 75.5% relative improvement, which means that this patch to patch upgrade improved the average response time of the former WordPress Docker image by 75.5%.

4 CASE STUDY RESULTS

In this section, we present the results of our case study in which we apply our methodology for analyzing the performance risks of upgrading Docker Hub images to the WordPress web application. For each research question, we discuss the motivation, approach, and results accordingly.

4.1 RQ1: What Is the Impact of Upgrading the Docker Image of WordPress on its Performance?

Motivation: Using Docker images allows users to upgrade an application easily. However, instead of a careful, managed upgrade, one may be dealing with an upgrade that changes many dependencies as well. Prior work [8] showed that a median of 8.6 dependencies change when a Docker image is upgraded. These changes may have an impact on the main application's performance. In this RQ, we study how the performance of WordPress is affected by upgrading its official image from Docker Hub.

Approach: We grouped the WordPress images in patch, minor and major groups as explained in Section 3. We then conducted the following analyses:

- *Patch Version Analysis:* In the patch version analysis, we analyzed the distribution of the relative response time improvements as a consequence of upgrading from one patch group to the next available patch group. We compute this by taking the Cartesian product of the images in both groups and computing the relative performance improvement according to Equation 1 for each upgrade in that product. In this analysis, we included upgrades from the last patch version in a minor version to the first available version in the next minor version (e.g., from WordPress 4.7.5 to 4.8), as this would be the next 'natural' upgrade for that version.
- *Minor Version Analysis:* For the minor version analysis, we analyzed the performance change after upgrading from all images in a specific minor group (e.g., 4.8.x) to all images in the next minor group (e.g., 4.9.x). Similar to the patch version analysis, in this analysis, we included upgrades from the last minor version in a major version to the next major version (e.g., from WordPress 4.9.x to 5.0.x).
- *Major Version Analysis:* In the major version analysis, we study the effect of upgrading from all images in a specific major version group (e.g. 4.x.x) to all images in the next major version group (e.g. 5.x.x).

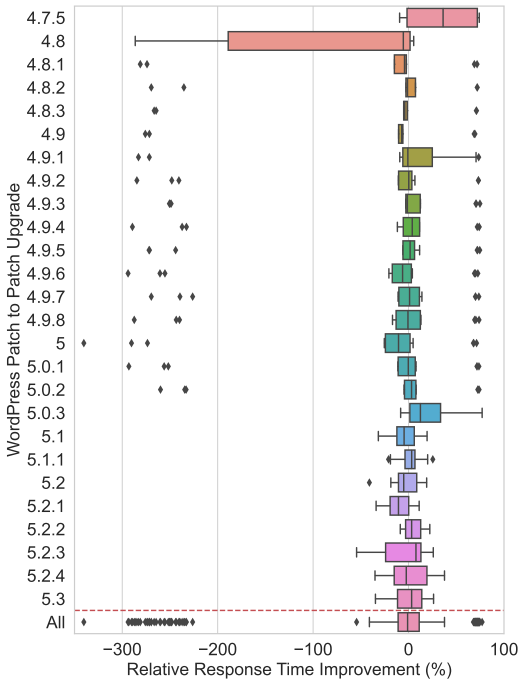

Figure 3: The distribution of relative response time improvements for patch to patch upgrades of the WordPress application images. The last box plot shows the distribution of relative response time improvements for all consecutive patch to patch upgrades of WordPress Docker images.

To compare the distributions of the patch to patch, minor to minor and major to major upgrades, we employ the Mann-Whitney U test [16] with an α-value of 0.05. The null hypothesis of this statistical test is that the two input distributions are equal. If the p-value of the test is smaller than 0.05, the null hypothesis is rejected and we conclude that the difference between the distributions is statistically significant. In addition, we quantify the difference using Cliff's Delta effect size [15]. We use the following thresholds for Cliff's Delta d [21]: *negligible(N)* if $|d| \leq 0.147$, *small(S)* if $0.147 < |d| \leq 0.33$, *medium(M)* if $0.33 < |d| \leq 0.474$ and *large(L)* if $0.474 < |d| \leq 1$. Due to space limitations, we published the average response times of the studied images online [23].

4.1.1 *Patch Version Analysis of WordPress.* **In 158 out of 302 (52%) patch to patch upgrades, WordPress' performance is degraded.** The last box plot in Fig. 3 shows all 302 possible consecutive patch to patch upgrades for the WordPress application. Fig. 3 shows that more than half of the possible patch to patch upgrades of the WordPress Docker images resulted in performance degradation. This finding shows that we cannot make accurate predictions about

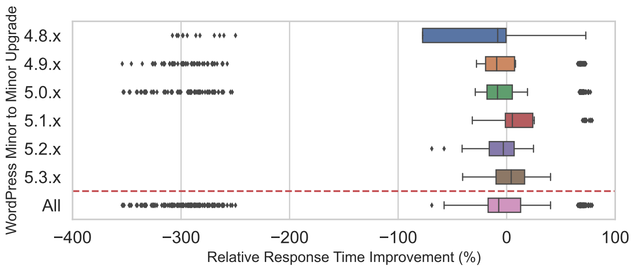

Figure 4: The distribution of relative response time improvements for minor to minor upgrades of the WordPress application images.

the effect of doing an image upgrade on the performance of the application based on the WordPress version of the image alone.

At worst, a patch to patch upgrade of a WordPress image resulted in 9% to 340% degradation in WordPress' performance. This degradation can be small such as when upgrading from WordPress 5.2.1 with PHP version 7.3.6 and Apache 2.4.25 to WordPress 5.2.2 with PHP version 7.1.32 and Apache version 2.4.38 (with a performance degradation of 9%). The worst-case occurred when upgrading WordPress 4.9.8 with PHP 7.2.12 and Apache 2.4.25 to WordPress 5.0.0 with PHP version 5.6.39 and Apache 2.4.25 (with a performance degradation of 340%). Clearly, not all of these upgrades are intuitive ones (e.g., downgrading from PHP 7 to 5), but they do demonstrate the problems that can occur when deciding to upgrade solely based on the WordPress version in the image.

At best, a patch to patch upgrade resulted in a 5% to 77% improvement in the performance of the WordPress application. This improvement can be small (5%), such as upgrading from WordPress 4.7.5 with PHP version 7.1.5 and Apache 2.4.10 to WordPress 4.8 with PHP version 7.0.21 and Apache version 2.4.10. The improvement could also be large (77%), such as when upgrading from WordPress 5.0.2 with PHP version 5.6.39 and Apache 2.4.25 to WordPress 5.0.3 with PHP version 7.3.2 and Apache version 2.4.25.

4.1.2 Minor Version Analysis of WordPress. **In 786 out of 1,218 (65%) minor to minor upgrades of WordPress, the average response time is degraded.** The last box plot in Fig. 4 shows the relative average response time improvements for all possible consecutive upgrades of the WordPress Docker image from a minor version to the next minor version. E.g., we consider upgrading all WordPress Docker images with minor version 4.7.x to 4.8.x and 4.8.x to 4.9.x and so on, resulting in 1,218 possible upgrades. Fig. 4 shows that 65% of these upgrades resulted in a lower performance than the previous version. Comparing this finding with its patch to patch counterpart, we observe that the risk of performance degradation is 13 percent points higher than a patch to patch upgrade, which was 52%. The Mann-Whitney U test shows that the performance risk of a minor to minor upgrade without conducting a performance test is significantly higher than for a patch to patch upgrade (albeit with a negligible effect size).

In the worst case, a minor to minor upgrade of the WordPress image resulted in 32% to 354% degradation of WordPress' performance. Fig. 4 presents the distribution of the relative response time improvements when upgrading the WordPress application from a minor version group to the next one. Fig. 4 shows that doing minor to minor upgrades may result in large variations in performance, for instance, upgrading to 4.8.x, 4.9.x, and 5.0.x from their previous minor version groups. This degradation may be small such as when upgrading from minor version 5.0.x to 5.1.x (with a performance degradation of 32%). The worst-case occurred when upgrading from minor version group 4.8.x to 4.9.x (which resulted in a 354% degradation of the performance).

In the best case, minor to minor upgrades improved WordPress' performance by 25% to 79%. Fig. 4 shows that there are some minor to minor upgrades that considerably improve the performance of the WordPress application. E.g., there was a minor upgrade for the slowest WordPress Docker image in version 5.1.x to 5.2.x that improved the WordPress application's performance by 25%. Also, for a WordPress Docker image in version 5.0.x there was an upgrade to version 5.1.x, which resulted in a 79% boost in the performance.

4.1.3 Major Version Analysis of WordPress. **Conducting a performance test is essential for WordPress Docker images even in a major upgrade.** A major upgrade can result in 400% performance degradation or 78% performance improvement. Moreover, there is a 58% chance of performance degradation. The boxplot of these results can be found online [23].

Summary of RQ1

The wide variation in relative response time improvements for the studied upgrades indicate that it is hard to predict how performance will be affected based on the WordPress version in the image alone. This implies that the performance of WordPress is mostly driven by other components in the image.

4.2 RQ2: What is the Relation Between the Performance of the WordPress Application and Updates of its Dependencies?

Motivation: The two major dependencies of WordPress are the Apache web server and PHP. As we observed in RQ1, upgrading the Docker image of WordPress may change these dependencies as well, which could in turn affect the performance. In this RQ, we analyze the impact of changes in the Apache and PHP versions on the response time of WordPress.

Approach: We grouped the performance measurements of images that use the same patch, minor or major versions of PHP or Apache, as specified in Section 4.1. E.g., we grouped the average response time of each Docker image of WordPress with PHP major version 5, minor version 6, and patch version 33 (5.6.33) in the PHP 5.6.33 patch group. We analyze these groups for both dependencies in a similar fashion as to what was done for WordPress in Section 4.1. Note that in our case study, we have no minor and major version analysis for Apache due to the used Apache versions.

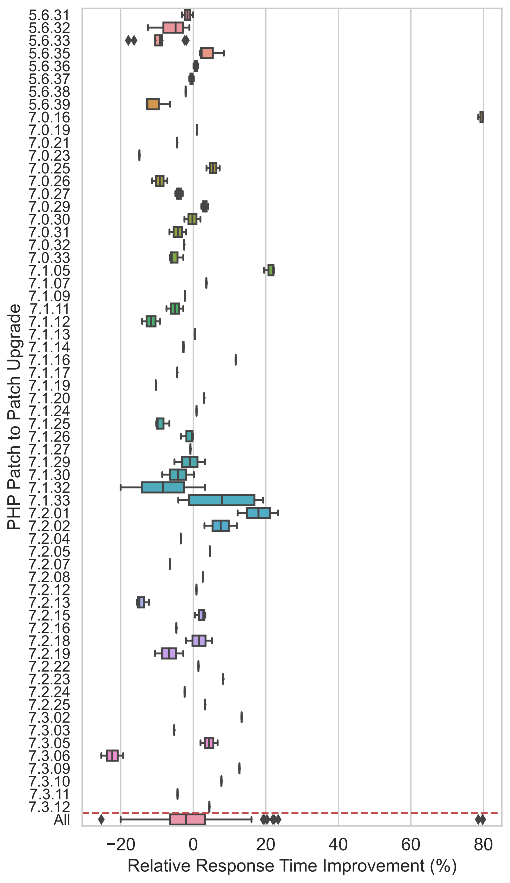

Figure 5: Distribution of relative response time improvements for upgrading PHP to the next available patch version.

4.2.1 Patch Version Analysis of PHP. **Upgrading to the next available PHP patch version degraded WordPress' performance in 71 (58%) out of 123 cases.** The last box plot in Fig. 5 shows the distribution of the relative average response time improvements for all consecutive patch to patch upgrades of PHP. As the box plot shows, 58% of the time, upgrading to the next patch version of PHP degraded WordPress' performance. The most severe degradation after an upgrade was 25%, while the best upgrade resulted in 80% improvement on the average response time.

4.2.2 Minor Version Analysis of PHP. **Upgrading the minor version of PHP in WordPress Docker images resulted in an improved performance in 970 out of 1,474 (66%) cases.** The last box plot in Fig. 6 depicts the distribution of relative average response time improvements when upgrading all WordPress Docker images with the same PHP minor version to the next minor version (1,474 possible upgrades). We observe that doing a minor to minor upgrade is 58% more likely to result in performance improvement than a patch to patch one. The Mann-Whitney U test confirms that this difference is statistically significant, with a small effect size.

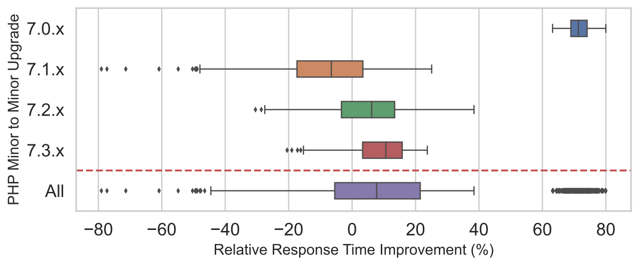

Figure 6: Distribution of relative response time improvements for a minor to minor upgrade for different minor versions of PHP.

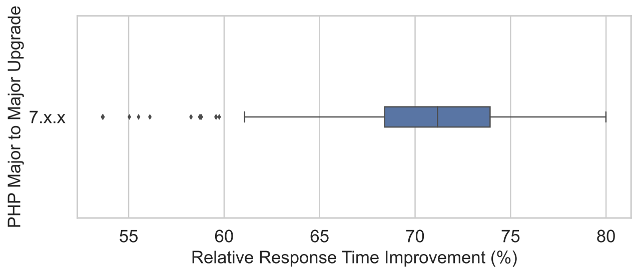

Figure 7: The distribution of the relative response time improvements for all consecutive major to major upgrades of PHP in WordPress Docker images.

In all studied WordPress Docker images, upgrading an image with PHP minor version 5.6 to one with PHP 7.0 improved the performance. Fig. 6 shows that for all 306 possible upgrades from PHP version 5.6.x to 7.0.x, an improvement from 63% to 80% was observed for the average response time. This finding indicates that older minor versions of PHP suffered from a relatively bad performance. Taken into account our other observations, we can conclude that the main contributing factor to the performance of WordPress images that use PHP 5.6.x was the PHP dependency.

4.2.3 Major Version Analysis of PHP. **Performing a major to major upgrade of PHP improved the performance of the WordPress application in all 1,241 cases.** Fig. 7 shows the distribution of the relative average response time improvements for all consecutive major to major upgrades of PHP in WordPress Docker images. Based on the studied WordPress Docker images, in all cases, this upgrade had a positive impact on the performance. In other words, selecting a random WordPress Docker image that uses PHP version 5.x.x and upgrading that to another WordPress Docker image with PHP major version 7.x.x resulted in performance improvements ranging from 54% to 80%.

4.2.4 Patch Version Analysis of the Apache Web Server. **Upgrading to the next available Apache patch version improved WordPress' performance in 1,183 (58%) out of 2,024 cases.** Fig. 8 shows the distribution of relative average response time improvements for all patch to patch upgrades of WordPress Docker images with the same Apache patch version to the next patch version. Fig. 8

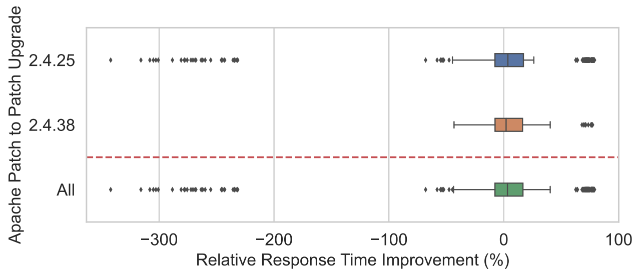

Figure 8: Distribution of relative response time improvements for upgrading an Apache patch version to the next available patch version.

shows that the chance of performance degradation is 42%, indicating that it is more likely that a patch to patch upgrade of Apache version improves the performance. However, these upgrades can result in 342% performance degradation or 79% performance improvement, such as upgrading from Apache 2.4.10 to 2.4.25. Together with our findings in the PHP analysis, we can conclude that it is hard to predict how upgrading an image will affect WordPress performance, based on its Apache version.

Summary of RQ2

It is hard to predict how upgrading a WordPress image will change the performance, based on its WordPress or Apache version. A major version upgrade of PHP considerably improved WordPress' performance in all cases, implying that WordPress' performance is highly dependent on the PHP version.

5 DISCUSSION

Below, we discuss the caveats of our methodology and we make recommendations.

Caveat 1: The recommendations apply to the studied application only. While we were able to extract several recommendations for the WordPress application (e.g., upgrades of images with PHP 5.x.x to PHP 7.x.x always resulted in improved performance), these recommendations apply to the WordPress application only. Until there is a large body of performance measurements available for Docker Hub images of many applications, the recommendations of our approach can be applied to the studied application only. Our vision is that through community effort (e.g., as described in Recommendation 1 below), or a future large-scale study, broader-ranging recommendations can be given.

Caveat 2: Collecting the performance measurements of an application takes a considerable amount of time. While the analysis step of our methodology is lightweight in terms of computation, collecting the performance measurements takes a relatively long time. The (short) performance tests that were conducted during our case study took a total of approximately 3 days to complete. For longer performance tests, or a larger number of Docker images, this can be problematic. One could argue whether doing such analysis is beneficial as compared to simply testing the performance of the target image. However, as said above, it is important to keep in mind that many developers and practitioners prefer to avoid performance tests.

Recommendation 1: Docker Hub should allow users to provide performance measurements of an image. During our case study, we observed that none of the official WordPress images mentioned anything about performance or response time. As a result, users of such images have to either resort to other sources of information, conduct their own performance tests, or simply take a gamble in terms of performance when doing an image upgrade. We recommend that Docker Hub provides a mechanism for submitting the results of performance tests of the offered images. In addition, sensitive Docker images could include an easy way to test the performance of the image. The analysis done through our methodology can then offer insights for users who wonder about the performance changes of upgrading to an image that does not have performance measurements yet.

Recommendation 2: Semantic versioning should be extended to cover performance changes. At the time of writing, the types of changes that are allowed by the semantic versioning principle do not cover performance. However, as mentioned before, performance is increasingly important for software and is considered 'the new correct' [9]. Hence, it is a missed opportunity that changes to performance are not covered by semantic versioning. During our case study, we observed that some recommendations can be made for the WordPress application based on the versioning of its dependencies (in particular, PHP). However, we expect that such recommendations would be much stronger when supported by the official semantic versioning specification.

6 THREATS TO VALIDITY

Internal Validity. One threat to internal validity is the short period of load testing for each image. We emphasize that the goal of our paper is not to provide a deep analysis of the performance of WordPress. Instead, we use WordPress to show how our methodology can be used to study the performance risks of upgrading Docker images of applications with several dependencies.

Another threat is the variability of measurements that are done within a cloud environment. We tested the performance of the images that had convenience tags and studied the variation between the repeated measurements. E.g., if an image had its original tag and three convenience tags, this allowed us to collect four times three repetitions of the performance test of the same image. We found that 95% of the duplicate images have less than $\sim$ 40 ms standard deviation, which is small given that the average response times are roughly between 400 and 1,000 ms. This shows that the variations in the average response times of the duplicate images are negligible and that the selected cloud environment did not impact the performance test measurements.

Another threat to the internal validity of our findings is that some of the studied upgrades are unlikely to be conducted in the real world. For example, it may be unlikely that users of an image with WordPress 4.x.x and PHP 7.x.x upgrade to an image with WordPress 5.x.x and PHP 5.x.x. However, it is important to study such upgrades as well for two reasons: (1) users may have a very specific reason for doing this upgrade (e.g., avoiding a vulnerability in PHP 7.x.x), and (2) users may not be aware that they are also changing the Apache version. While this change is described quite clearly for WordPress, dependency changes for other applications are much less obvious yet plentiful [8].

External Validity. In this study, we proposed a methodology to study the performance risks of upgrading the Docker Hub image of an application. The findings for the specific dependency versions apply to WordPress and its two main dependencies, PHP and Apache only. Also, we relied solely on the average response time of the application to draw conclusions about its performance. Many other metrics exist that can be relevant to capture the performance of an application. While our methodology is agnostic to the studied performance metric and applications, future studies should further our methodology in other settings.

7 CONCLUSION

In this paper, we propose a methodology to study the performance risks of upgrading Docker Hub images. In our case study on 90 official Docker Hub images of the WordPress application, we demonstrated how our methodology can help to analyze the performance risks of Docker image upgrades. Ideally, the performance of an image is thoroughly tested before it is being upgraded to. However, performance tests are not popular among developers [1, 12], and they are hard to execute correctly. Hence, our expectation is that many users of images from Docker Hub will conduct an image upgrade without doing such performance tests. The goal of our methodology is to give such users insights about the performance change of doing an image upgrade without conducting a performance test.

Our methodology can be beneficial for practitioners who wish to be informed about the change in performance they can expect when upgrading a Docker Hub image, without conducting a performance test of that image themselves. In particular, we call upon the community to start collecting performance measurements for Docker Hub images of a wide range of applications. These measurements can then be collected into a performance repository, which in turn can be leveraged by our methodology to provide recommendations about the expected performance of Docker images for which no performance measurements exist yet.

REFERENCES

[1] Cor-Paul Bezemer, Simon Eismann, Vincenzo Ferme, Johannes Grohmann, Robert Heinrich, Pooyan Jamshidi, Weiyi Shang, André van Hoorn, Monica Villavicencio, Jürgen Walter, and Felix Willnecker. 2019. How is Performance Addressed in DevOps?. In *ACM/SPEC International Conference on Performance Engineering (ICPE)*. 45–50.

[2] Gaurav Bhatia, Arjun Choudhary, and Vipin Gupta. 2017. The road to Docker: a survey. *International Journal of Advanced Research in Computer Science* 8, 8 (2017).

[3] Thanh Bui. 2015. Analysis of Docker security. *arXiv preprint arXiv:1501.02967* (2015).

[4] Emiliano Casalicchio and Vanessa Perciballi. 2017. Measuring Docker performance: What a mess!!!. In *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. 11–16.

[5] Theo Combe, Antony Martin, and Roberto Di Pietro. 2016. To Docker or not to Docker: A security perspective. *IEEE Cloud Computing* 3, 5 (2016), 54–62.

[6] Docker. 2021. The world’s leading service for finding and sharing container images with your team and the Docker community. https://www.docker.com Last accessed 2021-10-12.

[7] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. 2015. An updated performance comparison of virtual machines and Linux containers. In *IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 171–172.

[8] Sara Gholami, Hamzeh Khazaei, and Cor-Paul Bezemer. 2021. Should you Upgrade Official Docker Hub Images in Production Environments?. In *ICSE New Ideas and Emerging Results (NIER)*. 1–5.

[9] Mark Harman and Peter O’Hearn. 2018. From start-ups to scale-ups: Opportunities and open problems for static and dynamic program analysis. In *18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 1–23.

[10] IBM Cloud Education. 2020. Containerization. https://www.ibm.com/cloud/learn/containerization Last accessed 2021-10-12.

[11] Noureddine Kerzazi and Bram Adams. 2016. Botched releases: Do we need to roll back? empirical study on a commercial web app. In *23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, 574–583.

[12] Philipp Leitner and Cor-Paul Bezemer. 2017. An exploratory study of the state of practice of performance testing in Java-based open source projects. In *ACM/SPEC International Conference on Performance Engineering*. 373–384.

[13] Ashish Lingayat, Ranjana R Badre, and Anil Kumar Gupta. 2018. Performance evaluation for deploying Docker containers on baremetal and virtual machine. In *3rd International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 1019–1023.

[14] Locust. 2021. Locust - A modern load testing framework. http://locust.io/ Last accessed 2021-10-12.

[15] Jeffrey D Long, Du Feng, and Norman Cliff. 2003. Ordinal analysis of behavioral data. *Handbook of psychology* (2003), 635–661.

[16] H. B. Mann and D. R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50 – 60. https://doi.org/10.1214/aoms/1177730491

[17] Dirk Merkel. 2014. Docker: lightweight Linux containers for consistent development and deployment. *Linux Journal* 2014 (03 2014).

[18] Gianluca Mezzetti, Anders Møller, and Martin Toldam Torp. 2018. Type regression testing to detect breaking changes in Node.js libraries. In *32nd European Conference on Object-Oriented Programming (ECOOP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[19] Preston-Werner, Tom. 2021. Semantic Versioning 2.0.0. https://semver.org/ Last accessed 2021-02-18.

[20] Steven Raemaekers, Arie van Deursen, and Joost Visser. 2017. Semantic versioning and impact of breaking changes in the Maven repository. *Journal of Systems and Software* 129 (2017), 140–158.

[21] Jeanine Romano, Jeffrey D Kromrey, Jesse Coraggio, Jeff Skowronek, and Linda Devine. 2006. Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen’s d indices the most appropriate choices. In *Annual Meeting of the Southern Association for Institutional Research*.

[22] Bowen Ruan, Hang Huang, Song Wu, and Hai Jin. 2016. A performance study of containers in cloud environment. In *Asia-Pacific Services Computing Conference*. Springer, 343–356.

[23] Mikael Sabuhi, Petr Musilek, and Cor-Paul Bezemer. 2021. Docker Image Performance Risk Analysis. https://github.com/asgaardlab/wip-21-Mikael-DHPanalysis-code. Online; accessed 28 July 2021.

[24] Rui Shu, Xiaohui Gu, and William Enck. 2017. A study of security vulnerabilities on Docker Hub. *CODASPY - Proceedings of the 7th ACM Conference on Data and Application Security and Privacy* (2017), 269–280.

[25] Vasily Tarasov, Lukas Rupprecht, Dimitris Skourtis, Wenji Li, Raju Rangaswami, and Ming Zhao. 2019. Evaluating Docker storage performance: from workloads to graph drivers. *Cluster Computing* 22, 4 (2019), 1159–1172.

[26] Mario Tomisa, Marin Milkovic, and Marko Cacic. 2019. Performance Evaluation of Dynamic and Static WordPress-based Websites. *ICSEC 2019 - 23rd International Computer Science and Engineering Conference* (2019), 321–324.

[27] W3Techs. 2021. Usage statistics and market share of WordPress. https://w3techs.com/technologies/details/cm-wordpress/all/all Last accessed 2021-10-12.

[28] Wappalyzer. 2021. Identify technology on websites. https://www.wappalyzer.com/ Last accessed 2021-10-12.

[29] WordPress. [n. d.]. Hosting Requirements for WordPress. https://wordpress.org/about/requirements/ Last accessed 2021-10-12.

[30] Pengfei Xu, Shaohuai Shi, and Xiaowen Chu. 2017. Performance evaluation of deep learning tools in Docker containers. In *2017 3rd International Conference on Big Data Computing and Communications (BIGCOM)*. IEEE, 395–403.

[31] Ahmed Zerouali, Tom Mens, Gregorio Robles, and Jesus M Gonzalez-Barahona. 2019. On the relation between outdated Docker containers, severity vulnerabilities, and bugs. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 491–501.

[32] Nannan Zhao, Vasily Tarasov, Hadeel Albahar, Ali Anwar, Lukas Rupprecht, Dimitrios Skourtis, Arnab K Paul, Keren Chen, and Ali R Butt. 2020. Large-Scale Analysis of Docker Images and Performance Implications for Container Storage Systems. *IEEE Transactions on Parallel and Distributed Systems* 32, 4 (2020), 918–930.